UNIVERSIDADE FEDERAL DE SANTA CATARINA
Ciência da Computação
INE5425 - Modelagem e Simulação 2018.2
Profª Rafael Luiz Cancian
Alunos:
Karla Aparecida Justen (15100746)
e Bruno George de Moraes (14100825)

Objetivo: Desenvolvimento da classe MMC (projeto GenESyS-Reborn), com geração de números aleatórios (random) e de distribuições de probabilidade contínuas e discretas.

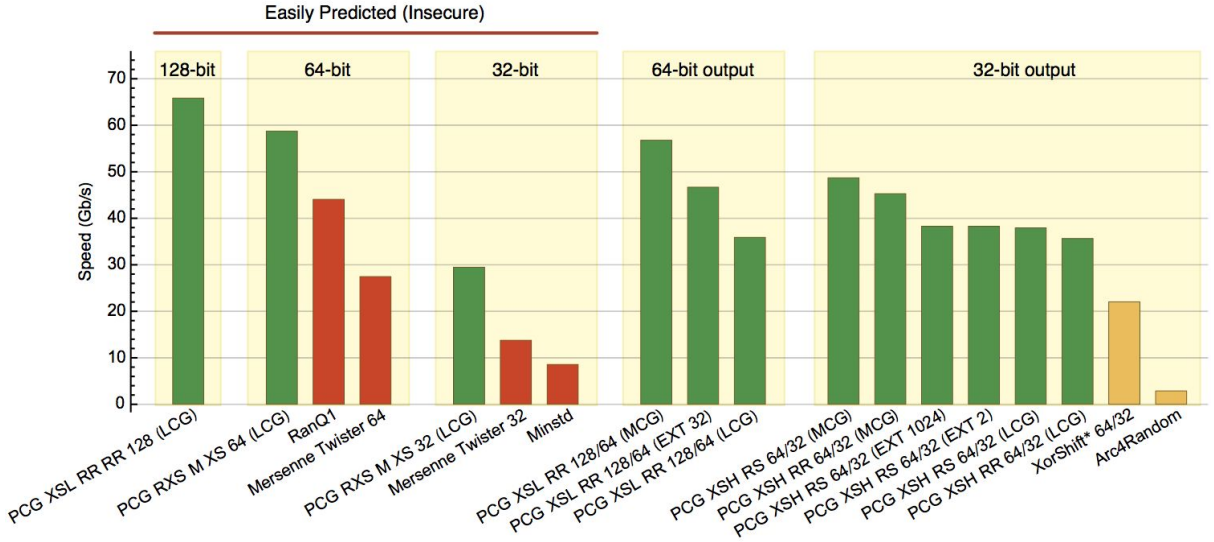
Geração de Números Aleatórios

Método escolhido: PCG-XSH-RR
Justificativa para usar o método PCG-XSH-RR :

Permuted Congruential Generators apresentam características como:

- Passam em testes estatísticos BigCrush com 36bits (gerador ideal necessita de 35);
- período arbitrário pode ser extendido em k-dimensões equidistantes [3];
- N Múltiplas sequências eg. 2^63;
- Robusto quanto previsibilidade[7];
- Oferta saltar à frente/atrás e distância-entre-estados
- Fácil de inicializar

Outra propriedade desejável para simulação é alta velocidade e segundo os testes da autora, PCGs são muito competitivos com o padrão (usado em Python e C++11 mt19937) de propósito geral Mersenne Twister.



PCG-RXS-M-XS passa BigCrush com **36bits** de estado (insecure);
PCG-XSH-RR (32 bits) passa BigCrush com **39bits**;
Para comparação, xorshift *, uma das melhores alternativas, requer 40 bits e o Mersenne twister falha apesar de 19937bits de estado.

“PCG inicia com apenas 1 inteiro já para o std::mt19937, isso é um pouco problemático porque são necessários 624 x 32 bits para o estado interno, mas o código inicializa com meros 32 bits de entropia. Por quê? Porque o C ++ 11 torna a inicialização adequada desnecessariamente difícil” (texto adaptado)[8].

Outras comparacoes em http://www.pcg-random.org/other-rngs.html

Teste de aleatoriedade

Teste de entropia [4] sao uteis em amostragem estatística, algoritmos de compactação e outros aplicativos nos quais a densidade de informações é de interesse.

$ ent.exe scratchfile
Entropy = 8.000000 bits per byte.
Optimum compression would reduce the size of this 412614656 byte file by 0 percent.
Chi square distribution for 412614656 samples is 232.99, and randomly would exceed this value 83.51 percent of the times.

Arithmetic mean value of data bytes is 127.5007 (127.5 = random).
Monte Carlo value for Pi is 3.141780011 (error 0.01 percent).
Serial correlation coefficient is -0.000009 (totally uncorrelated = 0.0).

Tambem fizemos testes para verificar o comportamento com algumas seeds, já que a seed determina o comportamento como ilustrado por Andrej Bauer's **random art** [9]. Para tal usamos o PractRand (versão 0.94) [5]. Tabela 1 apresenta o teste feito com 3 seeds. Foram testadas três vezes obtendo uma média de tempo gasto para gerar 1GB de dados.

RNG = RNG_stdin32, seed = unknown
test set = core, folding = standard (32 bit)
no anomalies in

Tabela 1: Teste de seed em três rodadas

seed	**Resultado (1GB)**			
	Rodada 1 (seg)	**Rodada 2 (seg)**	**Rodada 3 (seg)**	**Média**
0x4d595df4 d0f33173	19.4	19.3	19.3	19,333
0x4ac96dc 578f7d3c7	19.8	19.4	19.4	19,533
0xb4a10f2a d63c43c1	19.4	19.2	19.6	19,4

Geração de Números Aleatórios com Distribuição Conhecida

- Site que explica as diferentes distribuições: https://www.gnu.org/software/gsl/doc/html/randist.html

Nome da distribuição	Método	Descrição/Fonte/Observação
Uniforme	USE_BIASED_INT_MULT	http://www.pcg-random.org/posts/bounded-rands.html
Exponencial	gsl_ran_exponential(const gsl_rng * r, double mu)	https://www.gnu.org/software/gsl/doc/html/randist.html#the-exponential-distribution e funcao no link2
Erlang	dist gamma, com shapelimitado aos inteiros	Boost:: gamma_erland_dist GSL tbm
Normal		http://www.portalaction.com.br/probabilidades/62-distribuicao-normal
Gamma		https://www.gnu.org/software/gsl/doc/html/randist.html#the-gamma-distribution
Beta	Método em: http://www.portalaction.com.br/probabilidades/610-distribuicao-beta e https://www.gnu.org/software/gsl/doc/html/randist.html#c.gsl_ran_beta Explicação para limite inferior e superior: https://support.office.com/pt-br/article/dist-beta-função-dist-beta-11188c9c-780a-42c7-ba43-9ecb5a878d31	
Weibull	W=\lambda (-\ln(U))^{1/k}\,}	https://en.wikipedia.org/wiki/Weibull_distribution
LogNormal	https://pt.wikipedia.org/wiki/Distribuição_log-normal	
Triangular		genesys.pascal, GSL
Discrete	https://github.com/DavidPal/discrete-distribution Generating a sample takes O(1) time. This is in contrast with the naive algorithm that takes O(log N) in many common std libraries.	

Issues integracao

Tentamos a integracao com Sampler_if, mas o problema foi a derivacao de um RNG_Parameters, para PCG_Parameters. Quando isto estave no lugar o compilador nao enxergava corretamente no construtor do gerador pq queria RNG_Param; Ao deixar como PCG_Param o erro era que os parametros extras nao existiam em RNG_Param virtualç entao o getRNGParam() dava issue; Entao para entregar algo rodando vamos deixar comentado ": public RNG_Params" e ":public Sampler_" .

Referências

[1] https://www.iro.umontreal.ca/~lecuyer/myftp/papers/combmrg.ps
[2] https://stat.ethz.ch/R-manual/R-devel/library/parallel/html/RngStream.html
[3] http://www.pcg-random.org/
[4] http://www.fourmilab.ch/random/
[5] http://pracrand.sourceforge.net/

[6] critiquing-pcg-streams
[7] http://www.pcg-random.org/predictability.html
[8] http://www.pcg-random.org/using-pcg-cpp.html
[9] http://www.random-art.org/

Material de Apoio

- Aula RICE: https://www.cs.rice.edu/~johnmc/comp528/lecture-notes/Lecture21.pdf
- Artigo PCG: http://www.pcg-random.org/pdf/hmc-cs-2014-0905.pdf
- Repositório do GenESyS-Reborn: https://github.com/rlcancian/GenESyS-Reborn