Geração de Números aleatorios

João Medeiros

Introdução

- Aleatoriedade está presente em nosso dia a dia: sorteio de loterias, escolha de lados de campos.
- Aplicações em vários campos
- Simulações de Monte Carlo
- Criptografia
- Jogos de computadores

- Gerados por sistemas físicos
  - Fontes radioativas, ruídos em diodos, microfone, tempo entre digitações de teclas
- Gerados por algoritmos (pseudo-aleatórios)
  - Determinístico
  - Aparencia de aleatoriedade

Gerador de números aleatórios

- Inicia-se a sequência com uma semente s_0
- Calcula-se um novo valor $s_n = f(s_{n-1}), n \geq 1$
- O valor aleatório será $u_n = g(n)$. Considera-se que f e g são conhecidas.
- Como s é finito, o gerador eventualmente retornará a um valor já gerado.

Propriedades desejadas

- Uniformidade

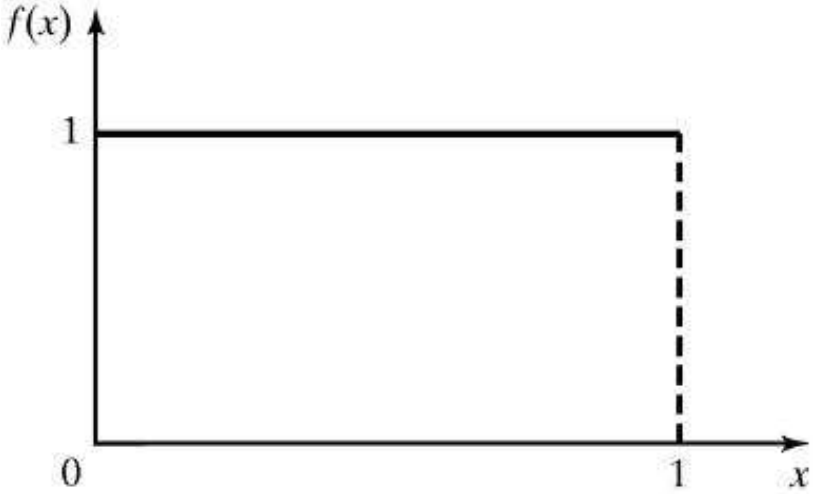
  Igualdade de probalidades entre os diversos valores.
- Independência

  O valor atual não tem qualquer relação com valores anteriores.
- Periodicidade

  O gerador deve fornecer uma sequência grande sem repetições.
- Deve ser rápido e portátil.
- Em simulações deve ser reproduzível, dada a mesma semente inicial.

Propriedades desejadas

Cada número aleatório u_i é uma amostra independente da distribuição uniforme



- Função densidade de probalilidade

$$f_u(x) = \begin{cases} 1, \text{ se } 0 \leq x \leq 1 \\ 0, \text{caso contrario} \end{cases}$$

- Valor médio

$$E(u) = \int_0^1 x dx = \left. \frac{x^2}{2} \right|_0^1 = \frac{1}{2}$$

Alguns Tipos de Geradores mais utilizados

- Geradores Congruentes Lineares
- Geradores de Registradores de Deslocamento

São os mais conhecidos e utilizados.

- Fórmula de recorrência

$$I_{j+1} = (aI_j + c)(\text{mod } m)$$

- onde
  - m é chamado módulo
  - a é o multiplicador
  - c é o incremento
- a e c escolhidos apropriadamente fornecem período máximo.
- Podemos obter numeros reais dividindo I_{j+1} por m, fornecendo números no intervalo $[0, 1)$ ou por $m - 1$, fornecendo números no intervalo $[0, 1]$.

Gerador Congruente Linear Multiplicativo

Obtido no caso $c = 0$

$$I_{j+1} = (aI_j)(\text{mod } m)$$

- São mais rápidos, não realizam a operação de adição.
  Exemplo: $I_{j+1} = 5I_j(\text{mod } 2^5)$.
  Sejam, $a = 5$, $c = 0$ e $m = 2^5$ e $I_0 = 1$. A sequência obtida será $5, 25, 29, 17, 21, 9, 13, 1, 5, 25, \ldots$. Período é 8
- Park e Miller, propuseram um gerador baseado nas escolhas

$$a = 7^5 = 16807, \qquad m = 2^{31} - 1 = 2147483647$$

- Esse gerador tem sido usado por muito tempo e passou na maioria dos testes teóricos.
- Para ser usado em computadores com representação de inteiros em 32 bits, usamos o algoritmo de Schrage.

Algoritmo de Schrage

- Baseado na fatoração de m.

$$m = aq + r, \qquad \text{ou } q = [m/a], r = m \bmod a$$

onde [] denota a parte inteira.

- Se $r < q$ e $0 < z < m - 1$, pode-se mostrar que $a(z \bmod q)$ e $r[z/q]$ permanecem no intervalo $0, \ldots, m - 1$ e que

$$az \bmod m = \begin{cases} a(z \bmod q) - r[z/q] & \text{se é } \geq 0 \\ a(z \bmod q) - r[z/q] + m & \text{caso contrário} \end{cases}$$

- Os valores para as constantes q e r são $q = 127773$ e $r = 2836$.

Uma implementação - ran0

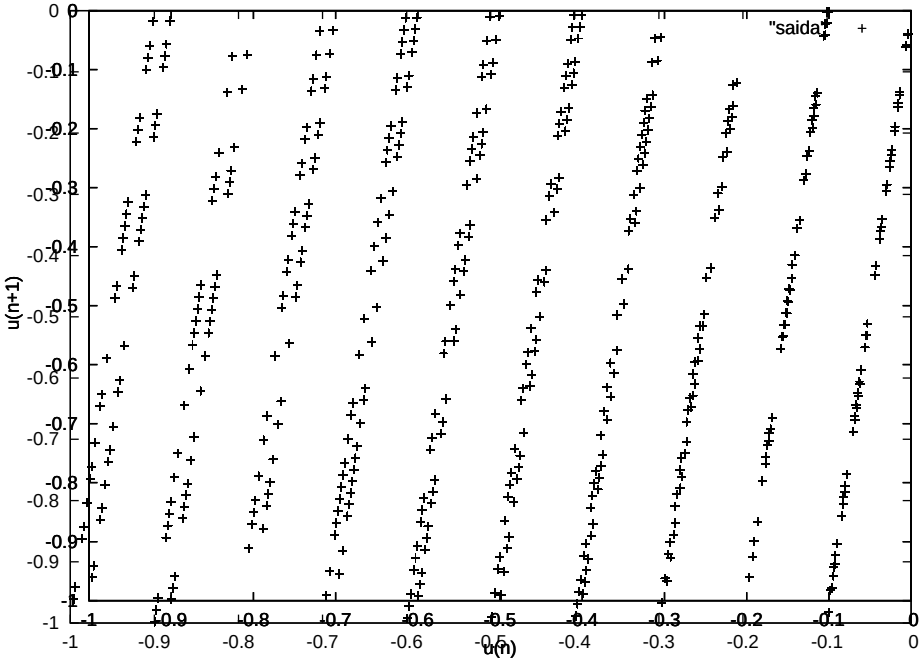

```
/* note #undef's at end of file */
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define MASK 123459876

float ran0(long *idum)
{
long k;
float ans;

*idum ^= MASK;
k=(*idum)/IQ;
*idum=IA*(*idum-k*IQ)-IR*k;
if (*idum < 0) *idum += IM;
ans=AM*(*idum);
*idum ^= MASK;
return ans;
}
```

Gerador Congruente Linear Multiplicativo

- Não recomendado como palavra final em geradores aleatórios.
- Existem algumas adaptações dos geradores congurentes lineares, veja o Numerical Recipes para mais implementações.
- Ocorrem estruturas em tuplas obtidas de números aleatórios consecutivos.
- Exemplo para $u_{n+1} = (10u_n) \bmod 509$



Geradores de Registradores de Deslocamento

- Consideram os bits de uma palavra do computador como os elementos de um vetor binário.
- Seja $\beta_{1xn} = (b_1, b_2, \ldots, b_n)$ um vetor binário e $T_{(nxn)}$ uma matriz binária.
- Para produzir iterativamente os vetores binários, os geradores utilizam $\beta, \beta T, \beta T^2, \ldots$.
- A matriz T é escolhida de tal maneira que o produto βT é realizado com operações elementares.

Os dois mais conhecidos são o Algoritmo de Kirkpatrick e Stoll e o ”Mersenne Twister” criado em 1996 por Matsumora e Nishimura. São caracterizados por terem um período muito longo.

Algoritmo de Kirkpatrick e Stoll

```
float ranks()
{
unsigned j;
*(b + na_cont) = *(b+ na_cont-250) ^*(b+ na_cont-103);
*(b+ na_cont-250) = *(b+ na_cont);
j = *(b+ na_cont);
if(na_cont==499)
 na_cont = 250;
else^M
(na_cont)++;
return ( (float)j/nupper);
}
```

O vetor b necessita ser inicializado com valores aleatórios utilizando algum outro gerador.

Testes de números aleatórios

Após selecionar os geradores, é importante testar suas propriedades: Uniformidade e Independência

- Testes de uniformidade
  - Teste de frequência
- Testes de independência
  - testes de sequência
  - testes de autocorrelação

- O principal método é o chamado Chi-quadrado
- Consiste no seguinte

  Dividir o intervalo $[0, 1]$ em k sub-intervalos

  Gerar amostrar $X_1, X_2, \ldots, X_n$

  Calcular o número de amostras em cada um dos intervalos $N_1, N_2, \ldots, N_k$

  Calcular o valor estístico

$$\chi^2 = \frac{k/n}{\sum_{j=1}^{k}} (N_j - n/k)^2$$

  Recorrer à alguma tabela ou método numérico que calcule $\chi^2_{k-1}1 - \alpha$ e comparar com o valor calculado. Se $\chi^2 < \chi^2_{k-1}1 - \alpha$, aceitamos que a sequência passou no teste.

Exemplo

0.12	0.01	0.23	0.28	0.89	0.31	0.64	0.28	0.83	0.93
0.99	0.15	0.33	0.35	0.91	0.41	0.60	0.27	0.75	0.88
0.68	0.49	0.05	0.43	0.95	0.58	0.19	0.36	0.69	0.87

- Aparentemente o quinto, décimo, décimo-quinto ... são similares

Testes de autocorrelação

- Testamos a autocorrelação entre cada m números, iniciando com o i-ésimo número.

  Temos que calcular ρ_{im} entre os seguintes números

$$X_i, X_{i+m}, \ldots, X_{i+(M+1)m} \qquad \text{com } M = \frac{N-1}{m} - 1$$

- Se a autocorrelação não for zero, os números não são totalmente independentes.

Testes de autocorrelação

- Para grandes valores de M, a distribuição ρ_{im} é aproximadamente normal se os valores $X_i, X_{i+m}, \ldots, X_{i+(M+1)m}$ forem descorellacionados.
- Podemos efetuar o teste estatístico

$$Z_0 = \frac{\rho_i m}{\sigma_{\rho_{im}}}$$

$$\rho_{im} = \frac{1}{M+1}\left[\sum_{k-0}^{M} X_{i+km}X_{i+(k+l)m}\right] - 0.25$$

$$\sigma_{\rho_{im}} = \frac{\sqrt{(13M+7)}}{12(M+1)}$$

Deveremos ter $-z_{\alpha/2} \leq Z_0 \leq z_{\alpha/2}$ para que o teste seja aceito.

Exemplo

Para a sequencia anterior, os terceiro, oitavo, décimo-terceiro ..., temos

$$\alpha = 0.05, i = 3, m = 5, N = 30 e M = 4.$$

$$\begin{aligned}
\hat{\rho}_{35} &= \frac{1}{4+1}\left[\begin{array}{l}(0.23)(0.28)+(0.28)(0.33)+(0.33)(0.27)\\ +(0.27)(0.05)+(0.05)(0.36)\end{array}\right]-0.25 \\
&= -0.1945 \\
\sigma_{\hat{\rho}_{35}} &= \frac{\sqrt{13(4)+7}}{12(4+1)} = 0.128 \\
Z_0 &= -\frac{0.1945}{0.1280} = -1.516
\end{aligned}$$

$z_{0.025} = 1.96$, então não podemos rejeitar o teste.

Aplicação - Cálculo de π


```
#include "stdio.h"
#include "math.h"
#include "time.h"

float ran0(long *idum);
main() {
float num;
float x,y;
long idum = time(0);
long i;
long N=10;
long total=0;
long atual=0;
float yx;
for(N=20;N<200;N+=20){
for(i=0; i<N; i++) {
x=ran0(&idum);
y=ran0(&idum);
yx = sqrt(1-x*x);
if(y<yx) atual++;
total++;
}
float pi = 4.0*atual/total;
printf("N %ld pi %f erro %f\n", N, pi, fabs(M_PI-pi));
}
```

Saída do programa


```
N 20 pi 3.400000 erro 0.258407
N 40 pi 3.133333 erro 0.008259
N 60 pi 3.233333 erro 0.091741
N 80 pi 3.220000 erro 0.078407
N 100 pi 3.186667 erro 0.045074
N 120 pi 3.200000 erro 0.058407
N 140 pi 3.164286 erro 0.022693
N 160 pi 3.183333 erro 0.041741
N 180 pi 3.182222 erro 0.040629
```

Resumo

- Geração de pseudo-número aleatórios
- Testes de uniformidade e independência

Alguns comentários:

- Mesmo geradores testados e utilizados a anos podem ser inadequados para o tipo de problema a ser tratado.
- Mesmo geradores que passam em todos os testes, podem tem algum padrão ainda não identificado.