Estatística Computacional

Patrícia de Siqueira Ramos

PPGEAB
UNIFAL-MG

11 de Abril de 2018

Geração de variáveis aleatórias uniformes

Introdução

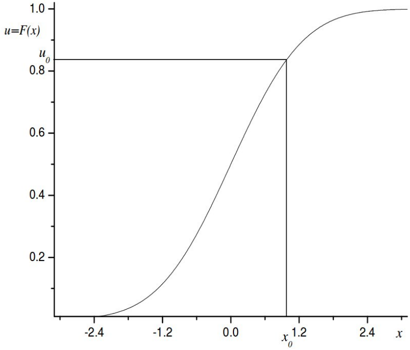
- Para gerar a.a. (amostras aleatórias) de uma distribuição de probabilidade usamos programas de computador, que são determinísticos
- Números pseudoaleatórios são gerados, pois qualquer sequência gerada é previsível
- Veremos como gerar números aleatórios para o modelo uniforme e, a partir dele, poderemos gerar realizações de variáveis aleatórias de qualquer outro modelo probabilístico

Números uniformes $\times$ números aleatórios

- Números uniformes: são observações retiradas da distribuição uniforme, ou seja, $\sim U$, que variam em uma faixa de valores com probabilidade constante
- Números aleatórios: realizações de qualquer distribuição de probabilidade, $\sim f(x)$, obtidas a partir de transformações nos números uniformes

Números uniformes × números aleatórios

- Um método muito utilizado para gerar v.a.s de alguma distribuição a partir da uniforme (ou seja, para gerar números aleatórios) é o método da transformação inversa ou amostragem inversa



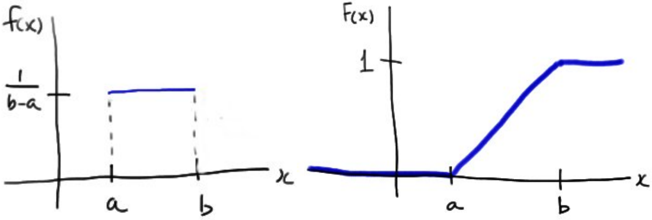
Exemplo do método da transformação inversa

$$
\begin{aligned}
f(x) &= 2x \\
F(x) &= x^2 = U \\
x &= \sqrt{U} = F^{-1}(U)
\end{aligned}
$$

Tal método pode ser ineficiente para muitas distribuições em que seja difícil obter F^{-1}

Números aleatórios uniformes

- Números aleatórios uniformes se situam, geralmente, entre $a = 0$ e $b = 1$, com f.d.p. constante



Caracteríticas de um bom gerador de números aleatórios uniformes

- Os valores gerados devem estar uniformemente distribuídos no intervalo $(0, 1)$, mas a estrutura não deve ser muito regular

Caracteríticas de um bom gerador de números aleatórios uniformes

- Os valores gerados devem estar uniformemente distribuídos no intervalo $(0, 1)$, mas a estrutura não deve ser muito regular
- Como, em simulações Monte Carlo, é muito comum precisarmos de milhares de valores de aleatórios, velocidade e bom uso de memória são essenciais

Caracteríticas de um bom gerador de números aleatórios uniformes

- Os valores gerados devem estar uniformemente distribuídos no intervalo $(0, 1)$, mas a estrutura não deve ser muito regular
- Como, em simulações Monte Carlo, é muito comum precisarmos de milhares de valores de aleatórios, velocidade e bom uso de memória são essenciais
- O período do gerador (número de valores gerados na sequência sem haver repetição) deve ser muito grande. Atualmente, um período de 10^{19} não é considerado satisfatório (antes era)

Caracteríticas de um bom gerador de números aleatórios uniformes

- Os valores gerados devem estar uniformemente distribuídos no intervalo $(0, 1)$, mas a estrutura não deve ser muito regular
- Como, em simulações Monte Carlo, é muito comum precisarmos de milhares de valores de aleatórios, velocidade e bom uso de memória são essenciais
- O período do gerador (número de valores gerados na sequência sem haver repetição) deve ser muito grande. Atualmente, um período de 10^{19} não é considerado satisfatório (antes era)
- Em criptografia, imprevisibilidade é crucial, mas para simulações Monte Carlo é mais importante que a sequência pareça aleatória e independente (i.i.d.)

Método da congruência

Método da congruência

- Introduzidos por Lehmer (1949) e foram populares por muitos anos
- Porém, ele apresenta algumas desvantagens que não o tornam atrativo atualmente
- Há alta correlação serial entre os números gerados, não atendendo à aleatoriedade:
  - por exemplo, dependendo das escolhas das variáveis que fazem parte dos cálculos do algoritmo, valores pequenos simulados serão sempre acompanhados de outros valores pequenos
  - num contexto de simulação seria fazer com que valores raros acontecessem juntos demais ou frequentemente demais)

Método da congruência

- Sejam os números uniformes inteiros U_1, U_2, $\dots$ entre 0 e $m-1$ (sendo m um número inteiro grande)
- O método usa a relação recursiva

$$U_{i+1} = (aU_i + c) mod\ m,$$

em que:

- a é o multiplicador
- U_i é o número uniforme criado no passo i
- c é o incremento
- mod é o resto da divisão
- m é chamado de módulo
- U_{i+1} é o número uniforme obtido no passo $i+1$

Método da congruência

- Por esse método, a sequência de números pseudoaleatórios gerada se repete e o tamanho é $\leq m$ (se a, c e m forem bem escolhidos)
- O valor do número uniforme correspondente no intervalo de 0 a 1 é dado por

$$U_{i+1}/m,$$

que é sempre menor que 1, mas podendo ser igual a zero

Método da congruência

- Por esse método, a sequência de números pseudoaleatórios gerada se repete e o tamanho é $\leq m$ (se a, c e m forem bem escolhidos)
- O valor do número uniforme correspondente no intervalo de 0 a 1 é dado por

$$U_{i+1}/m,$$

que é sempre menor que 1, mas podendo ser igual a zero

Exemplos:

- $m = 8,\ a = 5,\ c = 1,\ U_0 = 0$
- $m = 10,\ a = c = U_0 = 7$

Método da congruência - no R

```
congruencial = function(n, m, a, c, U0) {
   U = c()
   Ui = U0
   for (i in 1:n) {
      Ui = (a * Ui + c) %% m
      U[i] = Ui / m    # para resultados entre 0 e 1
   }
   return(U)
}
```

Método da congruência - no R

```
congruencial = function(n, m, a, c, U0) {
   U = c()
   Ui = U0
   for (i in 1:n) {
      Ui = (a * Ui + c) %% m
      U[i] = Ui / m    # para resultados entre 0 e 1
   }
   return(U)
}

m = 8; a = 5; c = 1; seed = 0; n = 10
X = numeric()
X[1] = seed / m
Y = congruencial(n, m, a, c, seed)
X = c(X, Y)
X
```

Método da congruência

- Valores usados por muitos anos: $a = 65.539$ e $m = 2^{31} = 2.147.483.648$
- m, geralmente, é o maior inteiro representado pela máquina, 2^{32}
- Park; Miller (1988) propuseram um gerador padrão mínimo com

$$a = 7^5 = 16.807, \qquad m = 2^{31} - 1 = 2.147.483.647,$$

- mas o produto aU_i excedia o limite de 32 bits para inteiros (não era possível implementar o método em linguagem de alto nível)

Método da congruência

- Truque de Schrage (1979) que se baseia na fatoração de m dada por:

$$m = aq + r, \qquad q = \left\lfloor \frac{m}{a} \right\rfloor, \qquad r = m \bmod a,$$

em que $\lfloor z \rfloor$ denota a parte inteira do valor de z (o mesmo que divisão inteira).

- Schrage também mostrou que:

$$U_{i+1} = \begin{cases} aU_i \bmod m = a(U_i \bmod q) - r\lfloor U_i/q \rfloor, & se\ U_i > 0 \\ aU_i \bmod m = a(U_i \bmod q) - r\lfloor U_i/q \rfloor + m, & caso\ contrário \end{cases}$$

Método da congruência

- Computacionalmente

$$a(U_i \ mod \ q) = a(U_i - q\lfloor U_i/q \rfloor),$$

- Para aplicar o algoritmo às constantes $a = 7^5$ e $m = 2^{31} - 1$ deve-se usar:

$$q = 127.773, \qquad r = 2.836$$

- O algoritmo[1] gerador padrão mínimo de números aleatórios é:

[1] Algoritmo disponível em FERREIRA, D. F. Estatística Computacional utilizando R. UFLA, 2014.

Gerador padrão mínimo de números aleatórios gna0

```
gna0 = function(n, sem=0){
   gnu0 = function(sem){ # função local
      k = sem %/% iq # divisao de inteiros
       # calculando (ia * sem mod im) sem provocar overflow - Schrage
      sem = ia * (sem %% iq) - ir * k
      if (sem < 0) sem = sem + im
      ran0 = am * sem # converte sem para ponto flutuante
      return(list(ran0 = ran0, sem = sem))
   }
   ia = 16807; im = 2147483647; am = 1.0 / im
   iq = 127773; ir = 2836
   if(sem <= 0){
      t = as.numeric(substring(Sys.time(),
         c(1,6,9,12,15,18),c(4,7,10,13,16,19))) # relógio/sist.
      sem = t[6] + t[5] * 60 + t[4] * 3600
      # retirar o efeito inicial
      sem = ia * (sem %% iq) - ir * (sem %/% iq)
      if(sem <= 0) sem = sem + im
   }
   u = matrix(0, n, 1) # inicia o vetor de resultados
   amostra = gnu0(sem) # chama gnu0
   u[1] = amostra$ran0 # inicia o primeiro elemento
   for (i in 2:n){
      amostra = gnu0(amostra$sem)
      u[i] = amostra$ran0
   }
   return(u)
} # função
```

Usos da gna0

```
# usos da função para gerar o vetor X de n números
# uniformes entre 0 e 1
n = 5
x = gna0(n,0)
x

n = 10
x = gna0(n,0)
x
```

Observações sobre a função gna0

- O valor da semente é definido pelo usuário (se for ≤ 0, a função atribui um número que depende da hora do sistema)
- A função `gna0` retorna números reais entre 0 e 1
- A função `gna0` é recursiva e, em suas sucessivas chamadas, o valor da semente é igual ao valor obtido no último passo
- Período de `gna0`: $2^{31} \cong 2,15 \times 10^9$, ou seja, uma sequência maior do que 2 bilhões de números aleatórios uniformes

Geração de números aleatórios uniformes no R

O R possui seu próprio gerador de números aleatórios: o algoritmo de Mersenne Twister (MATSUMOTO; NISHIMURA, 1998)

- elimina falhas dos geradores existentes
- possui o maior período: $2^{19937} - 1 \cong 4,3154 \times 10^{6001}$
- um dos mais rápidos, embora complexo
- faz uso da memória de forma muito eficiente

Geração de números aleatórios uniformes no R

- Comando usado no R: `runif(n, min, max)`
- É possível definir uma semente para gerar uma sequência reprodutível: set.seed(semente)
- A cada vez que a sequência for gerada deve-se utilizar o comando de definição da semente

```
# uso de semente para gerar sequência reprodutível
set.seed(0)
runif(10)

set.seed(0)
runif(20)
```

Histogramas

Comparar histogramas obtidos com a geração de números uniformes com as funções `gna0` e `runif` para $n = 1000$

Comparar os tempos dos geradores no R

```
# comparar tempos da gna0 e da runif
n = 1000000
t1 = system.time(gna0(n))
t2 = system.time(runif(n))
t1 / t2    # comparação entre os tempos
```