Estatística Computacional

Patrícia de Siqueira Ramos

PPGEAB
UNIFAL-MG

6 de Junho de 2018

Geração de outras variáveis aleatórias importantes

- Continuando a ideia de gerar variáveis aleatórias de algumas distribuições, veremos algoritmos para algumas das mais comuns na Estatística
- Para o caso contínuo focaremos na distribuição normal
- Um método importante para gerar dados da distribuição normal é o de Box-Müller (baseado na generalização do método da transformação de variáveis para mais de uma dimensão)

Caso contínuo - normal

Apenas relembrando: a normal com média μ e variância σ^2 tem f.d.p.

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2},$$

$-\infty < x < \infty$, $-\infty < \mu < \infty$, $\sigma > 0$.

Algoritmo de Box-Müller

Sejam p variáveis aleatórias $X_1, \ldots, X_p$ com função de densidade conjunta $f(x_1, \ldots, x_p)$
e sejam p variáveis $Y_1, \ldots, Y_p$, funções de todos os Xs, então a função densidade conjunta dos Ys é

$$f(y_1, \ldots, y_p) = f(x_1, \ldots, x_p) \quad \text{abs} \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \cdots & \frac{\partial x_1}{\partial y_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_p}{\partial y_1} & \cdots & \frac{\partial x_p}{\partial y_p} \end{vmatrix},$$

em que $J = |\partial()/\partial()|$ é o Jacobiano da transformação dos Xs em relação aos Ys.

Algoritmo de Box-Müller

Passos para gerar dados normais usando o método:

(1) Gerar duas realizações x_1 e x_2 a partir de $U(0,1)$

(2) Funções:
$y_1 = \sqrt{-2 \ln x_1} \cos(2\pi x_2)$
$y_2 = \sqrt{-2 \ln x_1} \sin(2\pi x_2)$

(3) Explicitar x_1 e x_2:
$x_1 = e^{-\frac{1}{2}(y_1^2+y_2^2)}$
$x_2 = \frac{1}{2\pi} \arctan\left(\frac{y_2}{y_1}\right)$

(4) Jacobiano da transformação: $-\frac{1}{2\pi} e^{-\frac{1}{2}(y_1^2+y_2^2)}$

Algoritmo de Box-Müller

Dessa forma,

$$f(y_1, y_2) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y_1^2} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y_2^2}.$$

Como a densidade conjunta de Y_1 e Y_2 é o produto de normais padrão independentes, podemos afirmar que as duas variáveis geradas (Y_1 e Y_2) são normais padrão independentes.

Podemos usar este método para gerar variáveis aleatórias normais a partir de números uniformes.

Algoritmo de Box-Müller

Dificuldade computacional: o uso de funções trigonométricas como o seno e o cosseno pode tornar o algoritmo lento.

- Truque para evitar diretamente o uso de funções trigonométricas:

  - Usar X_1 como $R^2 = U_1^2 + U_2^2$

  - O ângulo que o ponto (u_1, u_2) determina com o eixo 1 pode ser usado como ângulo aleatório $\theta = 2\pi X_2$

  - Com isso,

$$\cos(2\pi x_2) = u_1/\sqrt{r^2}$$

$$\sin(2\pi x_2) = u_2/\sqrt{r^2}$$

  (evitando o uso das funções trigonométricas)

Algoritmo de Box-Müller

Então,

$$y_1 = \sqrt{\frac{-2 \ln r^2}{r^2}} u_1$$
$$y_2 = \sqrt{\frac{-2 \ln r^2}{r^2}} u_2$$

Obs.: Para transformar uma $N(0,1)$ em $N(\mu, \sigma)$:

$$Z = \frac{X - \mu}{\sigma}$$
$$X - \mu = Z\sigma$$
$$X = Z\sigma + \mu$$

Algoritmo de Box-Müller

```
BoxMuller = function(n, mu=0, sigma=1){
   # Polar: função que retorna dois números normais
   Polar = function(){  # função sem argumento
      repeat{
         u = runif(2, -1, 1) # 2 v. uniformes U(-1,1)
         r2 = as.numeric(u %*% u) # toma o quadrado
         if((r2 > 0) & (r2 < 1)) break
      } # fim do repeat
      ff = sqrt(-2 * log(r2) / r2)
      y = ff * u # vetor de dim 2 com var. normais padrao ind.
      y
   } # fim de polar
   if(n %% 2 == 0){ # n par
      k = n %/% 2 # pega a parte inteira da div.
      for(ki in 1:k){
         if(ki == 1) x = c(Polar()) else x = c(x, Polar())
      }  # for
   } else{ # n impar
      k = n %/% 2
      if(k == 0){
         x = Polar()[1]
      } else{
         for(ki in 1:k){
            if(ki == 1) x = c(Polar()) else x = c(x, Polar())
         } # for
         x = c(x, Polar()[1])
      } # else interno
   } # else n par
   x = x * sigma + mu  # transformar de N(0, 1) para N(mu, sigma)
   return(x)
} # fim de BoxMuller
```

Algoritmo de Box-Müller

Exemplos de uso:

```
n = 12
(x = BoxMuller(n))
plot(density(x))
mean(x)
sd(x)
(y = BoxMuller(n, 100, 10))
plot(density(y))
mean(y)
sd(y)
par(mfrow = c(1,1))
n = 1000
w = BoxMuller(n, 100, 10)
plot(density(w))
hist(w, prob = T)
```

Outras aproximações para a normal (valor apenas didático)

- Aproximação 1: aproxima a normal igualando os quatro primeiros momentos (desvantagem: uso da exponenciação que é uma operação lenta). Para obter uma v.a. normal X a partir de uma uniforme $U \sim U(0,1)$:

$$X = [U^{0,135} - (1-U)^{0,135}]/0,1975$$

- Aproximação 2: soma de 12 ou mais v.a.s uniformes independentes. A v.a. $X = \sum_i^{12} U_i - 6$ tem distribuição aproximadamente normal padrão (isso ocorre em decorrência do TCL e porque cada uma das v.a.s uniformes tem média $1/2$ e variância $1/12$).

Geração de v.a. discreta - binomial

- Distribuição binomial: muito importante em aplicações estatísticas
- Relembrando a função de probabilidade binomial:

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}, \qquad x = 0, 1, \ldots, n,$$

em que n é o tamanho da amostra (n ensaios Bernoulli) e p é a probabilidade de sucesso.

- p deve ser constante em todos os ensaios
- os ensaios devem ser independentes
- Se $n = 1$, a binomial se especializa na Bernoulli
- Assim, a distribuição binomial é a repetição de n ensaios Bernoulli independentes e com probabilidade de sucesso constante

Teorema

Seja X o número de sucessos em n ensaios Bernoulli com probabilidade de sucesso p,

$$X = \sum_{i=1}^{n} I(U_i \leq p),$$

em que $U_1, \ldots, U_p$ são v.a.s $U(0,1)$ i.i.d. e $I()$ é a função indicadora. Então, $X \sim bin(n,p)$.

Lema

Soma de binomiais: Se $X_1, \ldots, X_k$ *são v.a.s binomiais independentes com* $(n_1, p), \ldots, (n_k, p)$,

$$\sum_{i=1}^{k} X_i$$

tem distribuição binomial com parâmetros $\left(\sum_{i=1}^{k} n_i, p \right)$.

Lema

Sejam $G_1, G_2, \ldots$ *v.a.s geométricas independentes e* X *o menor inteiro tal que*

$$\sum_{i=1}^{X+1} G_i > n.$$

X *tem distribuição bin*(n, p).

Distribuição geométrica

- Se G tem distribuição geométrica com probabilidade de sucesso constante $p \in (0, 1)$, então a função de probabilidade é

$$P(G = g) = (1 - p)^{g-1} p, \qquad g = 1, 2, \ldots$$

- A geométrica é a distribuição do tempo de espera até a ocorrência do primeiro sucesso no g-ésimo evento, numa sequência de ensaios Bernoulli independentes, incluindo o primeiro sucesso
- Assim, supõem-se $g - 1$ fracassos, cada um com probabilidade de ocorrência constante $1 - p$, antes da ocorrência de um sucesso no g-ésimo ensaio, com probabilidade p

Lema

Sejam $E_1, E_2, \ldots$ *v.a.s exponenciais i.i.d. e* X *o menor inteiro tal que*

$$\sum_{i=1}^{X+1} \frac{E_i}{n-i+1} > -\ln(1-p).$$

Logo, X *tem distribuição bin*(n, p).

Algoritmo BU

- As propriedades especiais da distribuição binomial, descritas no teorema e nos lemas, formam a base para dois algoritmos binomiais
- Estes dois algoritmos são baseados na propriedade de que uma v.a. binomial é a soma de n variáveis Bernoulli obtidas em ensaios independentes e com probabilidade de sucesso constante p
- O algoritmo binomial mais básico baseia-se na geração de n variáveis independentes $U(0,1)$ e no total das que são menores ou iguais a p

Algoritmo BU

1. Fazer $x = 0$ e $k = 0$
2. Gerar u de $U(0, 1)$ e fazer $k = k + 1$
3. Se $u \leq p$ fazer $x = x + 1$
4. Se $k < n$ voltar ao passo 2
5. Retornar x de uma $bin(n, p)$

Algoritmo BU

```
BU = function(n,p)
{
  x = 0; k = 0
  repeat
  {
    u = runif(1)
    k = k + 1
    if (u <= p) x = x + 1
    if (k == n) break
  } # repeat
  return(x)
} # função BU
```

Exemplo de uso - BU

```
x = BU(10, 0.5)
x
n = 1000
for (i in 2:n) x = c(x, BU(10, 0.5))
par(mfrow = c(1,1))
hist(x)
mean(x)
var(x)
```

Algoritmo BG

1. Fazer $y = 0$, $x = 0$ e $c = \ln(1 - p)$
2. Se $c = 0$, ir para o passo 6
3. Gerar u de $U(0, 1)$
4. $y = y + \lfloor \ln(u)/c \rfloor + 1$
5. Se $y \leq n$, fazer $x = x + 1$ e ir para o passo 3
6. Retornar x de uma $bin(n, p)$

Observações sobre o algoritmo BG

- O passo 4 vem do fato que

$$X = \left\lfloor \frac{\ln(1-U)}{\ln(1-p)} \right\rfloor + 1 = \left\lfloor \frac{\ln(U)}{\ln(1-p)} \right\rfloor + 1 \sim geom(p)$$

- Se $p > 0,5$, uma forma de melhorar o tempo de execução do algoritmo é usar o fato de que se $X \sim bin(n,p)$, $n - X \sim bin(n, 1-p)$
- Dessa forma, substituimos p por $\min(p, 1-p)$ e retornamos x se $p \leq 0,5$ ou retornamos $n - x$, caso contrário

Algoritmo BG

```
BG = function(n, p){
  if (p > 0.5) pp = 1 - p else pp = p
  y = 0; x = 0; c = log(1 - pp)
  if (c < 0){
    repeat{
      u = runif(1)
      y = y + trunc(log(u) / c) + 1
      if (y <= n){
        x = x + 1
      } else break
    } # repeat
    if (p > 0.5) x = n - x
  }
  return(x)
} # função BG
```

Exemplo de uso - BG

```
x = BG(10, 0.5)
x
n = 1000
for (i in 2:n) x = c(x, BG(10, 0.5))
par(mfrow = c(1,1))
hist(x)
mean(x)
var(x)
```

Exercícios

1. Obter o *QQplot* com as funções `qqnorm(x);qqline(x)` de amostras geradas com o algoritmo Box-Müller usando diferentes valores de n (100, 1.000 e 10.000).

2. Faça o mesmo usando a função `qqplot()` utilizando os argumentos corretos.

3. Criar a função `BoxMuller1` que retorna uma amostra de tamanho n de uma distribuição normal com média μ e desvio padrão σ utilizando o método de Box-Müller original (com as funções trigonométricas).
a) Gere $n = 1000$ realizações de v.a.s normais usando as funções `BoxMuller` e `BoxMuller1`.
b) Aplique o teste de Shapiro-Wilk nas duas amostras e interprete o resultado. A função é `shapiro.test(amostra)`.
c) Compare os tempos de execução das duas funções ao gerarem $n = 10.000$ realizações com:

```
n = 10000
system.time(BoxMuller(n))
system.time(BoxMuller1(n))
```