PhamNgocHai_21002139_LyThuyet

March 9, 2024

1 Intro

1.1 Create sample with sample()

[ ]:
```
# Dùng sample() để tạo mẫu
# x = sample(num_start:num_end, size=number_sample, replace=TRUE) # replace =
 ↪True là lấy ra có hoàn lại
x = sample(0:2, size=200, replace=TRUE, prob=c(.2, .3, .5))
table(x)
```

```
x
 0  1  2
39 62 99
```

[ ]:
```
x = sample(letters)
x
```

1. 'u' 2. 'c' 3. 'd' 4. 'f' 5. 'z' 6. 'a' 7. 't' 8. 's' 9. 'k' 10. 'b' 11. 'n' 12. 'h' 13. 'e' 14. 'x' 15. 'q' 16. 'm' 17. 'l' 18. 'y' 19. 'v' 20. 'p' 21. 'o' 22. 'j' 23. 'g' 24. 'r' 25. 'i' 26. 'w'

1.2 Random Generators of Common Probability Distributions in R

[ ]:
```
# The probability mass function (pmf) or density (pdf),
# cumulative distribution function (cdf),
# quantile function,
# and random generator of many commonly used probability distributions are
 ↪available.
# For example, four functions are documented in the help topic Binomial:
dbinom(x, size, prob, log = FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rbinom(n, size, prob)
```

2 Phương pháp sinh dữ liệu 1: The Inverse Transform Method

Đây là phương pháp sinh dữ liệu biến ngẫu nhiên dựa trên 2 hệ quả nổi tiếng.

Lý thuyết 1: (Probability Integral Transformation - Xác suất tích phân chuyển đổi). If X is a continuous random variable with cdf FX(x), then U = FX (X) Uniform(0, 1).

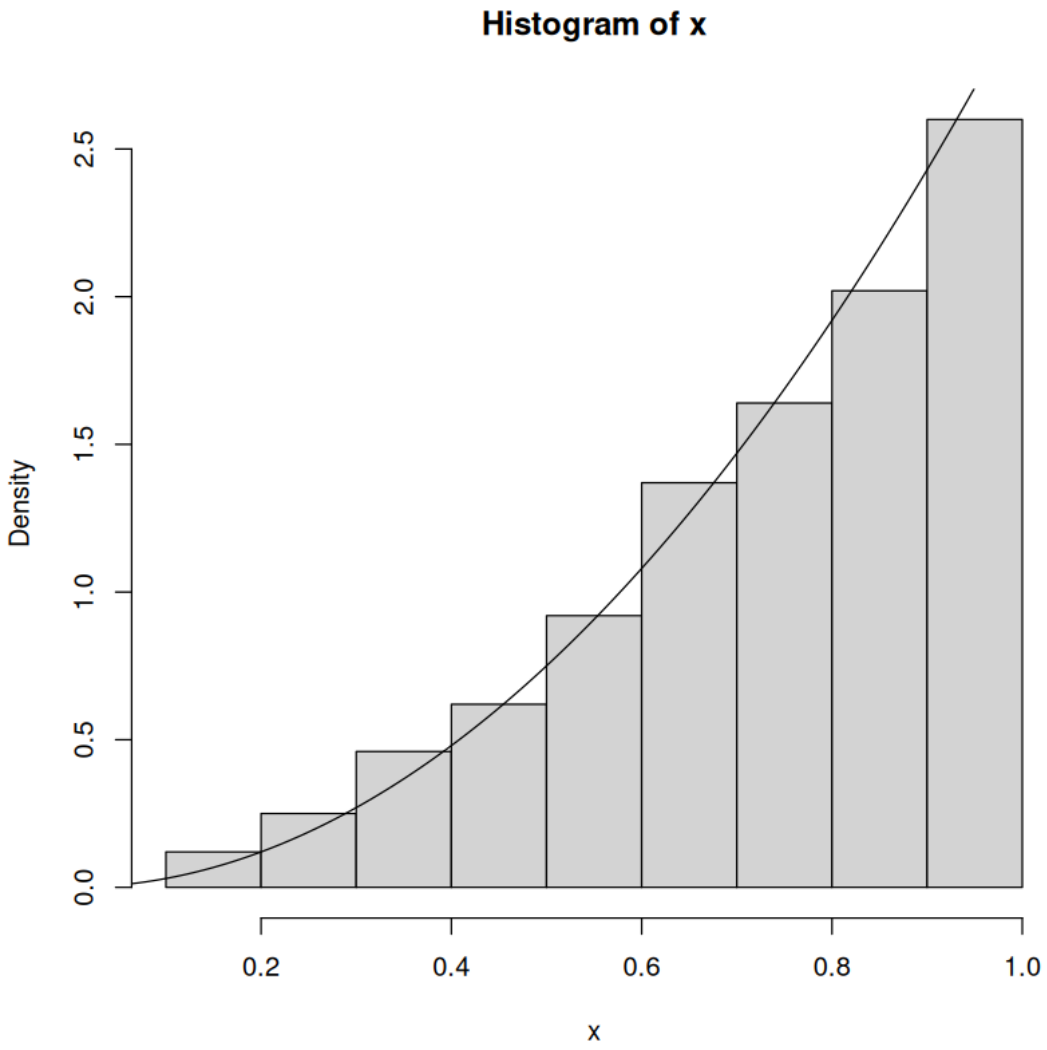
Phương pháp này có thể tóm tắt như sau:
1. Chuyển về hàm nghịch của nguyên hàm của hàm mật độ FX(u) (tạm gọi là f(u))
2. Tính toán hàm f này
3. Với mỗi biến ngẫu nhiên được yêu cầu:
- Tạo 1 u ngẫu nhiên từ Uniform(0, 1)
- Gán x = f(u)

2.1 Sử dụng phương pháp với trường hợp dữ liệu liên tục

Ví dụ 2. Biến X có hàm mật độ $f(x) = 3x^2$ với $0 < x < 1$.

-> $F(x) = x^3$ với 0 < x < 1.
-> $F^{-1}(u) = u^{-1/3}$

[ ]:
```
n <- 1000
u <- runif(n)   # Tạo một vector u gồm n giá trị ngẫu nhiên từ phân phối đều liên tục trên khoảng [0,1]
x <- u^(1 / 3)  # Tính toán một vector x bằng cách lấy căn bậc ba của từng phần tử trong vector u. Điều này tạo ra một phân phối không đều, với mật độ tập trung ở các giá trị nhỏ hơn của x.
hist(x, prob = TRUE)    # Vẽ histogram của vector x, với prob = TRUE để hiển thị mật độ xác suất thay vì số lượng trong từng bin.
y <- seq(0, 1, .01)     # Tạo một vector y chứa các giá trị từ 0 đến 1 với bước nhảy là 0.01. Điều này sẽ được sử dụng để vẽ đường cong mật độ xác suất.
lines(y, 3*y^2) # Vẽ đường cong mật độ xác suất trên histogram. Đường cong được xác định bởi hàm mật độ xác suất f(x) = 3x^2, nơi y là biến độc lập và 3*y^2 là giá trị mật độ xác suất tương ứng.
```



Ví dụ 3. (Phân phối mũ) Sử dụng phương pháp inverse transform để tạo mẫu ngẫu nhiên từ phân phối mũ với giá trị trung bình là $\frac{1}{\lambda}$.

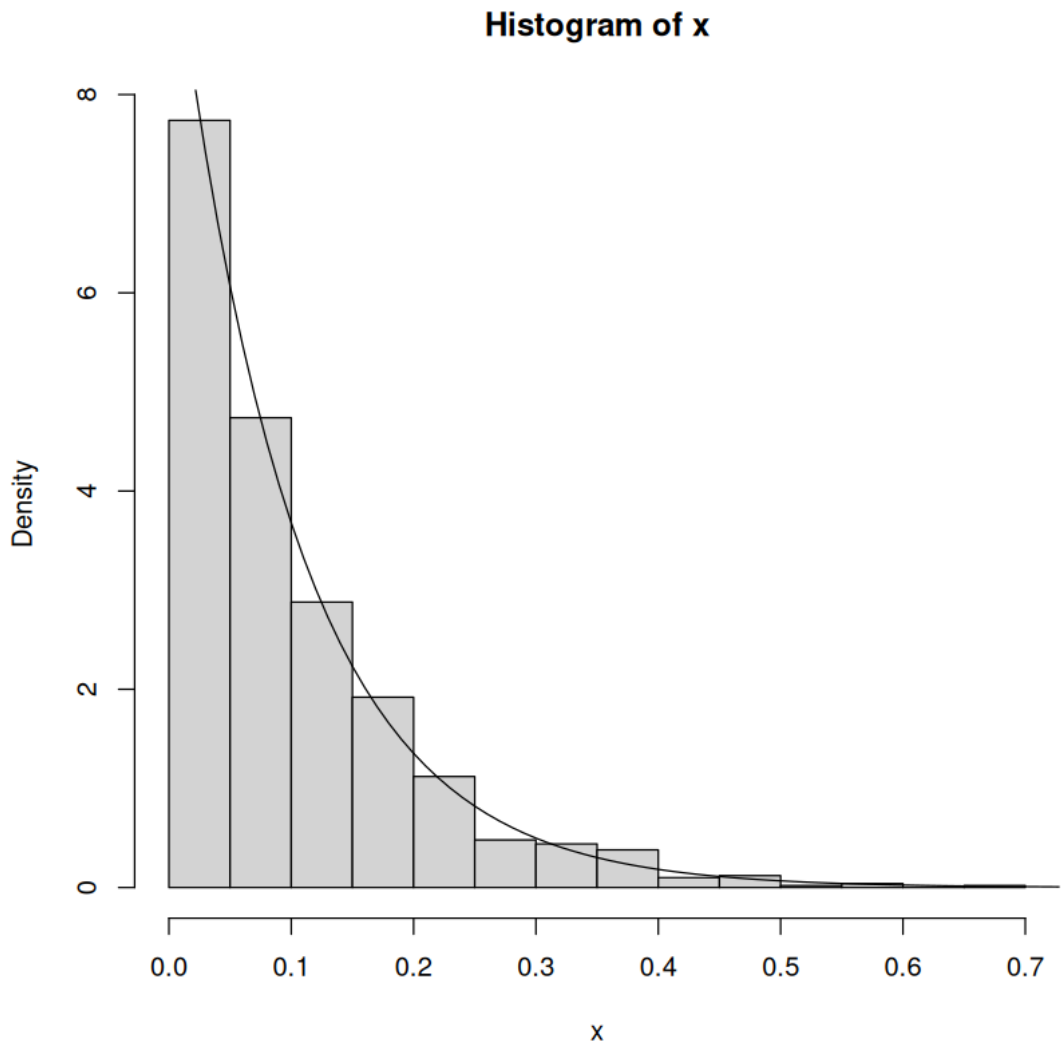
Nếu X theo phân phối Exp(lambda) thì với x $>$ 0 X có nguyên hàm của hàm mật độ là $F(x) = 1 - e^{-\lambda x}$

Và như vậy Inverse Transformation là hàm $F^{-1}(u) = \frac{-1}{\lambda}\log(1-u)$

Sau cùng để tạo 1 mẫu random với size n với tham số λ là: $\frac{-\log(runif(n))}{\lambda}$.

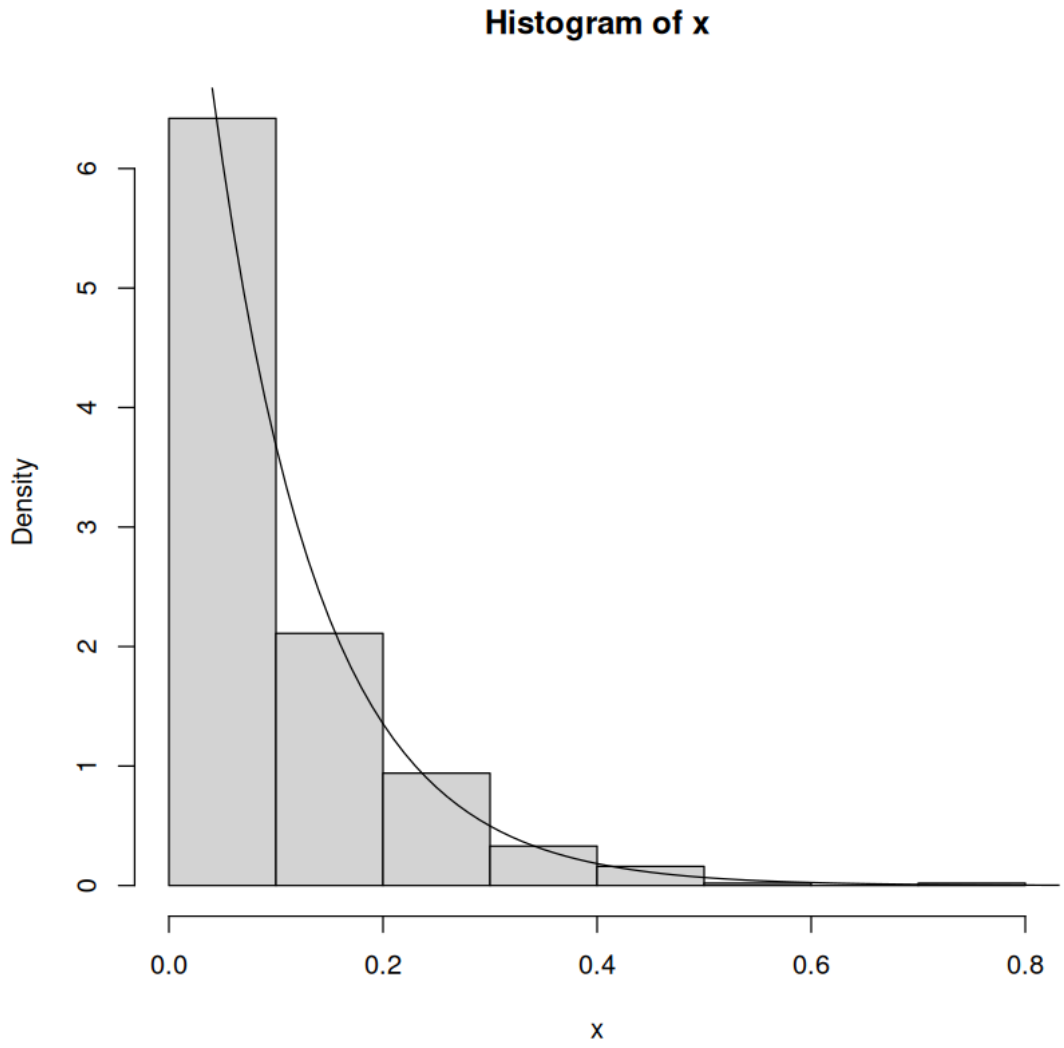
[ ]:
```
lambda <- 10
n <- 1000
u <- runif(n)    # Tạo một vector u gồm n giá trị ngẫu nhiên từ phân phối đều
 ↪liên tục trên khoảng [0,1]
```

```r
x <- - 1 / lambda * log(1 - u)  # Tính toán một vector x bằng cách lấy căn bậc
 ↪ba của từng phần tử trong vector u. Điều này tạo ra một phân phối không đều,
 ↪với mật độ tập trung ở các giá trị nhỏ hơn của x.
hist(x, prob = TRUE)    # Vẽ histogram của vector x, với prob = TRUE để hiển
 ↪thị mật độ xác suất thay vì số lượng trong từng bin.
y <- seq(0, 1, .01)     # Tạo một vector y chứa các giá trị từ 0 đến 1 với bước
 ↪nhảy là 0.01. Điều này sẽ được sử dụng để vẽ đường cong mật độ xác suất.
lines(y, lambda * exp(-lambda * y)) # Hàm mật độ xác suất cho phân phối mũ là
 ↪f(y)= ×e(- y), nơi   là tham số của phân phối.
```



[ ]:
```r
# cách sử dụng trực tiếp hàm rexp có sẵn trong R
lambda <- 10
```

```r
x <- rexp(1000, lambda) # Khởi tạo 1000 giá trị
hist(x, prob = TRUE)
y <- seq(0, 1, .01)     # Tạo một vector y chứa các giá trị từ 0 đến 1 với bước␣
 ↪nhảy là 0.01. Điều này sẽ được sử dụng để vẽ đường cong mật độ xác suất.
lines(y, lambda * exp(-lambda * y)) # Hàm mật độ xác suất cho phân phối mũ là␣
 ↪f(y)= ×e(- y), nơi   là tham số của phân phối.
```



2.2 Sử dụng phương pháp với dữ liệu rời rạc

If X is a discrete random variable and
$\ldots < x_{i-1} < x_i < x_{i+1} < \ldots$
are the points of discontinuity of $F_X(x)$, then the inverse transformation is $F_X^{-1}(u) = x_i$, where $F_X(x_{i-1}) < u \leq F_X(x_i)$. For each random variate required: 1. Generate a random u from

Uniform(0,1). 2. Deliver x_i where $F_X(x_{i-1}) < u \leq F_X(x_i)$.

Ví dụ 4. (Phân phối nhị thức)
Mẫu ngẫu nhiên của Bernoulli (p = 0.4). Mặc dù có phương pháp đơn giản hơn cho phân phối này trong R.
Trong ví dụ này, $F_X(0) = f_X(0) = 1 - p$ và $F_X 1 = 1$ nếu u > 0.6 và $F_X^{-1}(u) = 0$ nếu $u \leq 0.6$.
Code gen dưới đây sẽ để biểu thức điều kiện u > 0.6.

[ ]:
```
n <- 1000   # gán n = 1000
p <- 0.4    # gán p = 0.4
u <- runif(n)   # tạo ra 1000 giá trị ngẫu nhiên trong khoảng (0, 1)
x <- as.integer(u > 0.6) # (u > 0.6) is a logical vector
mean(x)
var(x)
```

0.409

0.241960960960961

[ ]:
```
# cách 2: sử dụng hàm rbinom có sẵn trong R
n <- 1000   # gán n = 1000
p <- 0.4    # gán p = 0.4
rbinom(n, size = 1, prob = p)
sample(c(0,1), size = n, replace = TRUE, prob = c(.6,.4))
```

1. 1 2. 0 3. 0 4. 0 5. 1 6. 1 7. 1 8. 0 9. 1 10. 0 11. 0 12. 1 13. 0 14. 1 15. 0 16. 0 17. 1 18. 1 19. 0
20. 1 21. 0 22. 1 23. 0 24. 1 25. 0 26. 1 27. 0 28. 0 29. 0 30. 0 31. 1 32. 0 33. 0 34. 1 35. 0 36. 1 37. 1
38. 1 39. 0 40. 0 41. 1 42. 0 43. 0 44. 0 45. 1 46. 0 47. 1 48. 0 49. 1 50. 0 51. 0 52. 0 53. 0 54. 0 55. 0
56. 1 57. 1 58. 0 59. 0 60. 1 61. 0 62. 1 63. 0 64. 0 65. 0 66. 1 67. 0 68. 1 69. 1 70. 0 71. 1 72. 1 73. 1
74. 0 75. 0 76. 1 77. 0 78. 0 79. 0 80. 0 81. 0 82. 0 83. 1 84. 0 85. 1 86. 1 87. 1 88. 0 89. 0 90. 0 91. 1
92. 0 93. 1 94. 0 95. 0 96. 1 97. 1 98. 0 99. 0 100. 0 101. 0 102. 0 103. 1 104. 1 105. 0 106. 1 107. 1
108. 0 109. 0 110. 0 111. 1 112. 1 113. 1 114. 0 115. 0 116. 0 117. 0 118. 1 119. 0 120. 0 121. 1 122. 0
123. 0 124. 0 125. 0 126. 0 127. 1 128. 0 129. 0 130. 0 131. 0 132. 0 133. 1 134. 1 135. 0 136. 0 137. 0
138. 1 139. 1 140. 0 141. 0 142. 0 143. 1 144. 1 145. 1 146. 0 147. 0 148. 1 149. 1 150. 1 151. 0 152. 0
153. 0 154. 0 155. 0 156. 1 157. 0 158. 0 159. 0 160. 0 161. 0 162. 0 163. 0 164. 1 165. 1 166. 0 167. 1
168. 0 169. 1 170. 1 171. 1 172. 0 173. 1 174. 1 175. 1 176. 1 177. 0 178. 0 179. 1 180. 1 181. 1 182. 0
183. 0 184. 1 185. 1 186. 1 187. 0 188. 0 189. 1 190. 0 191. 1 192. 0 193. 0 194. 1 195. 0 196. 0 197. 0
198. 1 199. 0 200. 1 201. 202. 0 203. 0 204. 0 205. 0 206. 0 207. 0 208. 1 209. 1 210. 1 211. 1 212. 1
213. 1 214. 0 215. 1 216. 0 217. 0 218. 0 219. 0 220. 1 221. 0 222. 0 223. 0 224. 0 225. 1 226. 0 227. 0
228. 0 229. 0 230. 1 231. 0 232. 1 233. 1 234. 0 235. 1 236. 1 237. 0 238. 0 239. 0 240. 1 241. 0 242. 1
243. 1 244. 1 245. 1 246. 0 247. 0 248. 1 249. 1 250. 0 251. 1 252. 1 253. 0 254. 1 255. 0 256. 1 257. 0
258. 1 259. 1 260. 1 261. 0 262. 0 263. 1 264. 1 265. 0 266. 1 267. 0 268. 1 269. 1 270. 0 271. 0 272. 0
273. 1 274. 1 275. 1 276. 1 277. 0 278. 1 279. 0 280. 0 281. 0 282. 0 283. 1 284. 0 285. 0 286. 1 287. 0
288. 0 289. 0 290. 0 291. 1 292. 1 293. 0 294. 0 295. 0 296. 0 297. 1 298. 0 299. 0 300. 0 301. 1 302. 0
303. 1 304. 1 305. 0 306. 0 307. 0 308. 0 309. 0 310. 1 311. 0 312. 1 313. 1 314. 1 315. 0 316. 0 317. 0
318. 1 319. 1 320. 0 321. 1 322. 0 323. 0 324. 0 325. 0 326. 0 327. 0 328. 0 329. 0 330. 0 331. 0 332. 0
333. 0 334. 0 335. 0 336. 0 337. 0 338. 0 339. 0 340. 0 341. 1 342. 1 343. 1 344. 0 345. 1 346. 1 347. 1
348. 0 349. 0 350. 1 351. 0 352. 1 353. 0 354. 1 355. 0 356. 0 357. 0 358. 0 359. 1 360. 0 361. 0 362. 0
363. 0 364. 0 365. 1 366. 1 367. 0 368. 0 369. 1 370. 0 371. 0 372. 0 373. 0 374. 1 375. 0 376. 1 377. 0
378. 0 379. 0 380. 0 381. 0 382. 1 383. 1 384. 0 385. 0 386. 1 387. 0 388. 1 389. 0 390. 0 391. 0 392. 0

393. 0 394. 0 395. 0 396. 0 397. 1 398. 1 399. 1 400. 0 401. 1

1. 1 2. 0 3. 0 4. 1 5. 0 6. 1 7. 0 8. 0 9. 1 10. 1 11. 0 12. 1 13. 1 14. 1 15. 0 16. 1 17. 1 18. 0 19. 0 20. 1 21. 0 22. 0 23. 0 24. 0 25. 0 26. 0 27. 0 28. 1 29. 0 30. 0 31. 0 32. 1 33. 0 34. 0 35. 0 36. 1 37. 1 38. 1 39. 0 40. 0 41. 1 42. 0 43. 0 44. 1 45. 1 46. 1 47. 0 48. 0 49. 0 50. 1 51. 0 52. 1 53. 1 54. 1 55. 0 56. 0 57. 1 58. 0 59. 0 60. 1 61. 0 62. 1 63. 0 64. 1 65. 1 66. 1 67. 0 68. 0 69. 1 70. 0 71. 0 72. 0 73. 0 74. 0 75. 1 76. 0 77. 0 78. 0 79. 1 80. 1 81. 0 82. 0 83. 0 84. 0 85. 0 86. 0 87. 0 88. 0 89. 0 90. 0 91. 0 92. 0 93. 1 94. 1 95. 1 96. 0 97. 0 98. 1 99. 0 100. 0 101. 1 102. 0 103. 1 104. 0 105. 0 106. 0 107. 0 108. 0 109. 1 110. 1 111. 1 112. 0 113. 1 114. 0 115. 0 116. 0 117. 0 118. 0 119. 0 120. 0 121. 1 122. 0 123. 0 124. 0 125. 1 126. 1 127. 1 128. 0 129. 0 130. 0 131. 1 132. 1 133. 0 134. 0 135. 1 136. 0 137. 0 138. 1 139. 0 140. 0 141. 0 142. 1 143. 0 144. 1 145. 1 146. 1 147. 0 148. 1 149. 0 150. 0 151. 0 152. 0 153. 1 154. 0 155. 0 156. 0 157. 1 158. 0 159. 0 160. 0 161. 0 162. 0 163. 0 164. 1 165. 0 166. 0 167. 0 168. 0 169. 0 170. 1 171. 0 172. 1 173. 0 174. 0 175. 0 176. 0 177. 1 178. 0 179. 1 180. 1 181. 0 182. 0 183. 0 184. 1 185. 0 186. 0 187. 1 188. 0 189. 1 190. 0 191. 0 192. 1 193. 1 194. 1 195. 0 196. 0 197. 0 198. 0 199. 0 200. 1 201. 202. 0 203. 0 204. 1 205. 0 206. 0 207. 0 208. 0 209. 1 210. 0 211. 1 212. 1 213. 0 214. 1 215. 1 216. 0 217. 0 218. 0 219. 1 220. 0 221. 1 222. 1 223. 0 224. 0 225. 1 226. 0 227. 1 228. 0 229. 0 230. 1 231. 0 232. 1 233. 0 234. 0 235. 0 236. 1 237. 1 238. 0 239. 0 240. 0 241. 1 242. 1 243. 1 244. 1 245. 1 246. 1 247. 0 248. 0 249. 0 250. 0 251. 0 252. 1 253. 0 254. 1 255. 0 256. 1 257. 0 258. 0 259. 1 260. 1 261. 1 262. 0 263. 0 264. 0 265. 0 266. 1 267. 1 268. 0 269. 0 270. 0 271. 0 272. 1 273. 1 274. 0 275. 0 276. 0 277. 0 278. 0 279. 1 280. 0 281. 0 282. 0 283. 0 284. 0 285. 0 286. 0 287. 1 288. 0 289. 1 290. 0 291. 1 292. 0 293. 1 294. 0 295. 1 296. 1 297. 1 298. 0 299. 1 300. 1 301. 0 302. 0 303. 0 304. 1 305. 1 306. 1 307. 0 308. 0 309. 0 310. 0 311. 0 312. 0 313. 0 314. 0 315. 0 316. 1 317. 0 318. 1 319. 1 320. 0 321. 0 322. 1 323. 1 324. 0 325. 0 326. 1 327. 0 328. 0 329. 1 330. 0 331. 1 332. 0 333. 1 334. 0 335. 0 336. 0 337. 1 338. 1 339. 1 340. 1 341. 1 342. 0 343. 0 344. 0 345. 0 346. 1 347. 0 348. 1 349. 0 350. 1 351. 0 352. 1 353. 1 354. 0 355. 1 356. 0 357. 0 358. 1 359. 1 360. 0 361. 0 362. 0 363. 1 364. 1 365. 1 366. 1 367. 0 368. 0 369. 0 370. 1 371. 0 372. 1 373. 0 374. 0 375. 0 376. 0 377. 1 378. 1 379. 1 380. 0 381. 1 382. 1 383. 0 384. 0 385. 0 386. 0 387. 0 388. 0 389. 1 390. 1 391. 1 392. 1 393. 0 394. 0 395. 0 396. 0 397. 0 398. 1 399. 0 400. 0 401. 0

Ví dụ 5. (Phân phối hình học) Tạo mâu phân phối hình học với tham số p = 0.25.

Hàm mật độ (pmf) là $f(x) = pq^x$ với $x = 0, 1, 2, ...$ khi $q = 1 - p$.
Và như vậy, cdf là $F(x) = 1 - q^{x+1}$.
Với mỗi phần tử của mẫu ngẫu nhiên ta có 1 biến u ngẫu nhiên chuẩn và giải:
$1 - q^x < u \leq 1 - q^{x+1}$.
Giải ra với x ta được
$x < \frac{log(1-u)}{log(q)} \leq x + 1$
-> $x + 1 = \lceil \frac{log(1-u)}{log(q)} \rceil$

[ ]:
```
n <- 1000
p <- 0.25
u <- runif(n)
k <- ceiling(log(1 - u) / log(1 - p)) - 1
k
```

1. 5 2. 2 3. 11 4. 4 5. 2 6. 10 7. 5 8. 6 9. 0 10. 5 11. 1 12. 0 13. 1 14. 2 15. 12 16. 8 17. 2 18. 3 19. 2 20. 12 21. 1 22. 6 23. 6 24. 15 25. 7 26. 2 27. 5 28. 4 29. 4 30. 1 31. 2 32. 1 33. 1 34. 0 35. 3 36. 1 37. 0 38. 4 39. 2 40. 1 41. 4 42. 1 43. 1 44. 1 45. 0 46. 1 47. 5 48. 9 49. 2 50. 7 51. 0 52. 1 53. 12 54. 4 55. 9 56. 2 57. 2 58. 5 59. 3 60. 0 61. 0 62. 5 63. 1 64. 6 65. 2 66. 6 67. 8 68. 0 69. 6 70. 0 71. 3

72. 13 73. 3 74. 7 75. 2 76. 0 77. 1 78. 1 79. 0 80. 0 81. 4 82. 0 83. 1 84. 4 85. 0 86. 1 87. 0 88. 1 89. 1 90. 23 91. 0 92. 1 93. 3 94. 0 95. 8 96. 1 97. 2 98. 0 99. 4 100. 8 101. 2 102. 3 103. 0 104. 2 105. 1 106. 1 107. 6 108. 2 109. 2 110. 1 111. 3 112. 4 113. 2 114. 0 115. 5 116. 6 117. 1 118. 1 119. 2 120. 1 121. 1 122. 0 123. 3 124. 10 125. 6 126. 0 127. 1 128. 10 129. 0 130. 2 131. 1 132. 0 133. 6 134. 0 135. 4 136. 12 137. 3 138. 4 139. 3 140. 0 141. 2 142. 2 143. 1 144. 1 145. 3 146. 3 147. 2 148. 2 149. 0 150. 9 151. 1 152. 8 153. 2 154. 0 155. 0 156. 0 157. 3 158. 5 159. 0 160. 1 161. 12 162. 6 163. 4 164. 4 165. 0 166. 4 167. 0 168. 5 169. 16 170. 5 171. 4 172. 1 173. 0 174. 0 175. 15 176. 2 177. 5 178. 3 179. 5 180. 1 181. 6 182. 1 183. 1 184. 0 185. 3 186. 0 187. 6 188. 7 189. 4 190. 2 191. 7 192. 5 193. 0 194. 2 195. 6 196. 9 197. 0 198. 6 199. 2 200. 2 201. 202. 1 203. 1 204. 6 205. 0 206. 0 207. 4 208. 0 209. 1 210. 4 211. 1 212. 10 213. 3 214. 8 215. 3 216. 0 217. 0 218. 1 219. 11 220. 5 221. 3 222. 7 223. 5 224. 2 225. 6 226. 0 227. 1 228. 0 229. 2 230. 9 231. 0 232. 4 233. 1 234. 7 235. 11 236. 1 237. 0 238. 6 239. 1 240. 0 241. 9 242. 0 243. 7 244. 4 245. 1 246. 1 247. 0 248. 1 249. 3 250. 9 251. 8 252. 4 253. 0 254. 2 255. 9 256. 2 257. 3 258. 0 259. 3 260. 1 261. 7 262. 1 263. 0 264. 3 265. 16 266. 1 267. 3 268. 2 269. 0 270. 2 271. 4 272. 7 273. 4 274. 4 275. 7 276. 1 277. 0 278. 7 279. 0 280. 2 281. 2 282. 2 283. 0 284. 7 285. 9 286. 1 287. 2 288. 0 289. 1 290. 3 291. 5 292. 4 293. 2 294. 3 295. 0 296. 2 297. 5 298. 3 299. 1 300. 3 301. 2 302. 5 303. 1 304. 8 305. 2 306. 7 307. 3 308. 6 309. 0 310. 3 311. 2 312. 2 313. 0 314. 4 315. 4 316. 1 317. 1 318. 0 319. 3 320. 3 321. 1 322. 2 323. 0 324. 1 325. 1 326. 1 327. 0 328. 6 329. 13 330. 0 331. 2 332. 4 333. 1 334. 0 335. 8 336. 3 337. 10 338. 5 339. 2 340. 12 341. 1 342. 0 343. 1 344. 5 345. 2 346. 0 347. 1 348. 0 349. 14 350. 3 351. 3 352. 2 353. 0 354. 3 355. 1 356. 2 357. 1 358. 6 359. 3 360. 3 361. 0 362. 3 363. 3 364. 2 365. 2 366. 1 367. 0 368. 3 369. 5 370. 5 371. 1 372. 3 373. 3 374. 1 375. 4 376. 0 377. 4 378. 2 379. 0 380. 3 381. 11 382. 2 383. 2 384. 5 385. 2 386. 0 387. 10 388. 1 389. 9 390. 2 391. 4 392. 6 393. 4 394. 1 395. 6 396. 7 397. 3 398. 2 399. 1 400. 1 401. 8

Ví dụ 6. (Phân phối Logarithmic)

[ ]:
```r
rlogarithmic <- function(n, theta) {
    # returns a random logarithmic(theta) sample size n
    u <- runif(n)
    # set the initial length of cdf vector
    N <- ceiling(-16 / log10(theta))
    k <- 1:N
    a <- -1 / log(1 - theta)
    fk <- exp(log(a) + k * log(theta) - log(k))
    Fk <- cumsum(fk)
    x <- integer(n)
    for (i in 1:n) {
        x[i] <- as.integer(sum(u[i] > Fk)) # F^{-1}(u)-1
        while (x[i] == N) {
            # if x==N we need to extend the cdf
            # very unlikely because N is large
            logf <- log(a) + (N + 1) * log(theta) - log(N + 1)
            fk <- c(fk, exp(logf))
            Fk <- c(Fk, Fk[N] + fk[N + 1])
            N <- N + 1
            x[i] <- as.integer(sum(u[i] > Fk))
        }
    }
```

```
    x + 1
}
```

[ ]:
```
n <- 1000
theta <- 0.5
x <- rlogarithmic(n, theta)
#compute density of logarithmic(theta) for comparison
k <- sort(unique(x))
p <- -1 / log(1 - theta) * theta^k / k
se <- sqrt(p * (1 - p) / n) # standard error
round(rbind(table(x)/n, p, se),3)
```

A matrix: 3×9 of type dbl

	1	2	3	4	5	6	7	8	11
	0.724	0.193	0.056	0.017	0.005	0.001	0.002	0.001	0.001
p	0.721	0.180	0.060	0.023	0.009	0.004	0.002	0.001	0.000
se	0.014	0.012	0.008	0.005	0.003	0.002	0.001	0.001	0.000