1. 파레토분포 $Pa(a,b)$는 다음과 같은 분포함수를 갖는다.

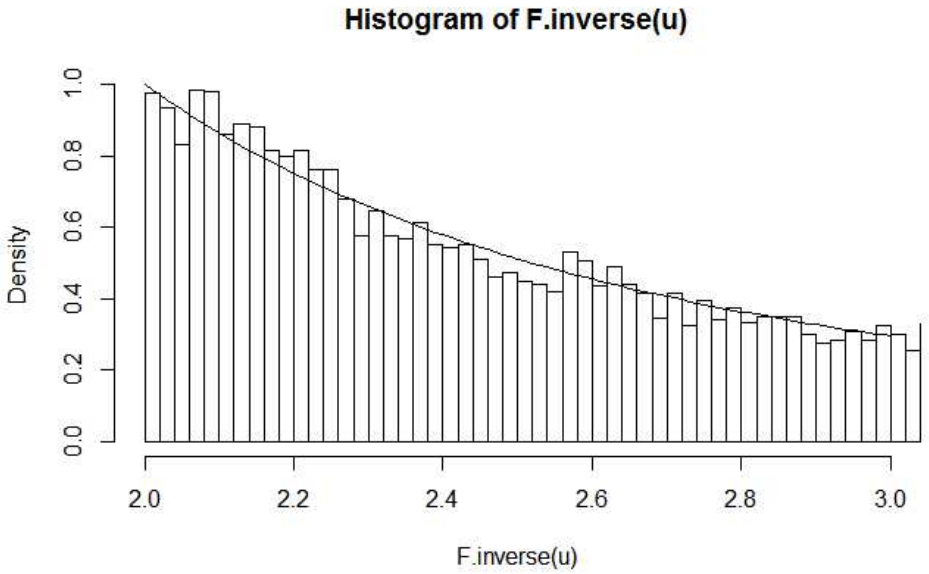
$$F(x) = 1 - \left(\frac{b}{x}\right)^a, x \geq b > 0, a > 0.$$

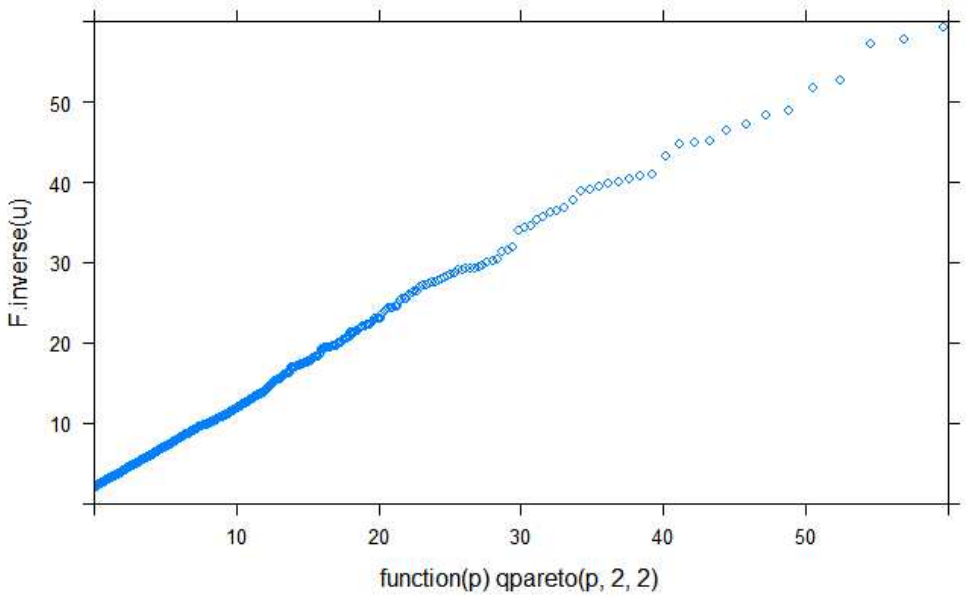
역변환법을 이용하여 $Pa(2,2)$분포로부터 난수 10,000개를 발생시키시오. `curve()`함수를 이용하여 $Pa(2,2)$의 밀도함수를 그리고 `add = TRUE`를 이용하여 동일한 구간에 발생된 난수의 히스토그램을 겹쳐 그리고, 발생된 난수에 대하여 QQ-plot을 그려서 역변환법이 잘 구현되었는지 확인하시오.

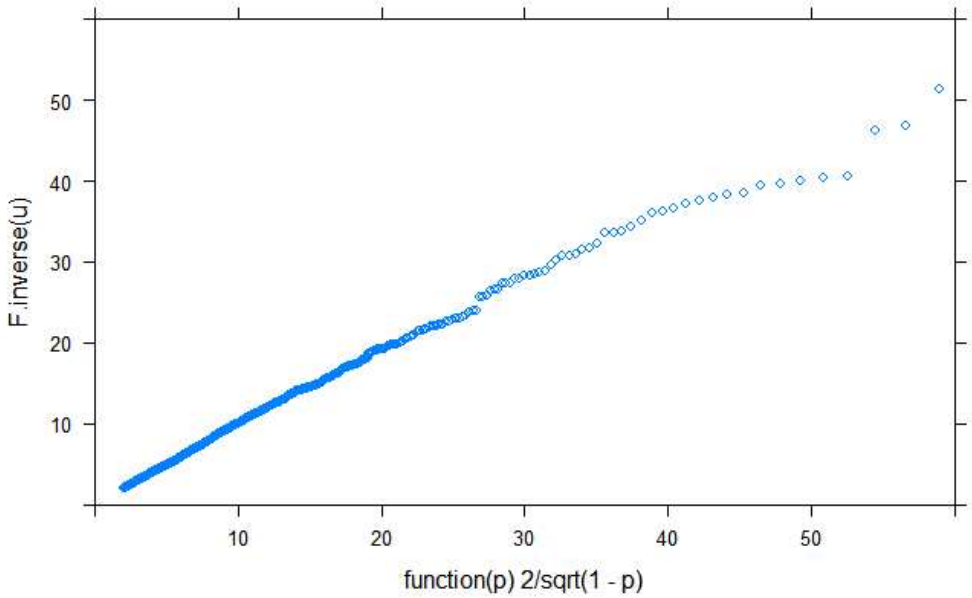
$Pa(2,2)$의 분포함수 $F(x) = 1 - (\frac{2}{x})^2$이고 $F^{-1}(x) = \frac{2}{\sqrt{1-x}}$ 입니다. (단, $x < 1$)

따라서, 0과 1사이의 난수를 10,000개 생성한 후 $F^{-1}(x) = \frac{2}{\sqrt{1-x}}$ 에 대입했습니다.

```
> #Q1. Pareto Distribution
> u = runif(10000)
> Fx = function(x){
+   return(1 - (2/x)^2)
+ }
> F.inverse = function(x){
+   return(2/sqrt(1-x))
+ }
> F.density = function(x){
+   return(8/x^3)
+ }
> hist(F.inverse(u), xlim = c(2, 3), freq = FALSE, breaks = 10000)
> curve(F.density, add = TRUE)
> library(lattice)
> qqmath(F.inverse(u),distribution =  function(p) 2/sqrt(1-p), xlim = c(0, 60), ylim = c(0, 60))
> #qpareto함수를 사용하기 위해 rmutil 패키지를 이용했습니다.
> #qpareto 함수를 이용해도 같은 결과를 얻을 수 있습니다.
> library(rmutil)
> qqmath(F.inverse(u),distribution =  function(p) qpareto(p, 2, 2), xlim = c(0, 60), ylim = c(0, 60))
```




F.inverse(u)
10
20
30
40
50
10
20
30
40
50
function(p) qpareto(p, 2, 2)


F.inverse(u)
10
20
30
40
50
10
20
30
40
50
function(p) 2/sqrt(1 - p)

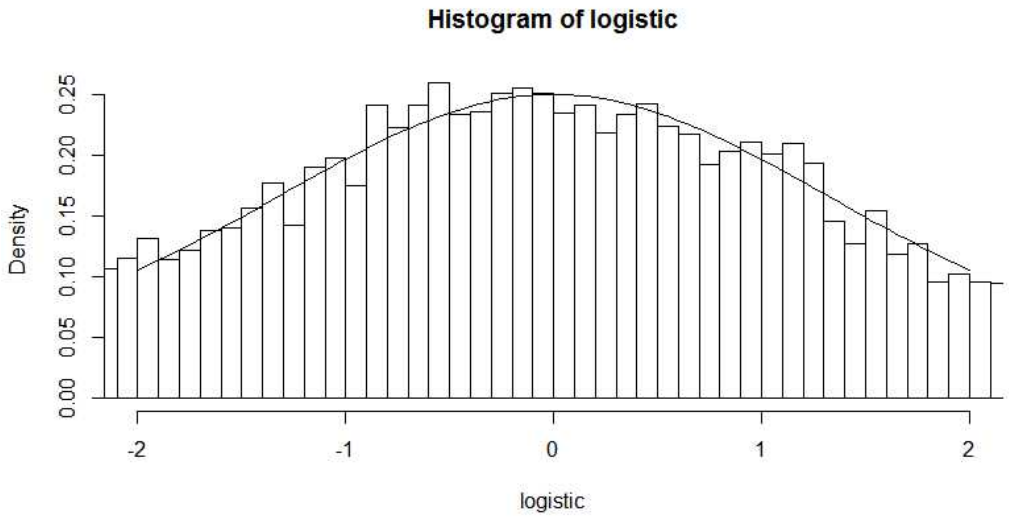
2. 역변환법을 이용하여 다음과 같은 확률밀도함수를 갖는 로지스틱분포를 따르는 확률난수를 생성하는 R함수를 작성하라.

$$f(x) = e^x/(1+e^x)^2, -\infty < x < \infty.$$

확률밀도함수는 $f(x) = \frac{e^x}{(1+e^x)^2}$ 이므로 분포함수 $F(x) = -\frac{1}{1+e^x}$ 입니다.

그리고 $F^{-1}(x) = \ln(-\frac{1}{x} - 1)$ 입니다. $(단, -1 < x < 0)$

따라서, -1에서 0까지의 난수를 10,000개 생성한 뒤 $F^{-1}(x) = \ln(-\frac{1}{x} - 1)$에 대입했습니다. 아래 코드와 그림은 각각 로지스틱분포를 따르는 확률난수를 생성하고, 그 분포가 확률밀도함수와 유사한지 확인해 보았습니다. Plot는 1번 문제와 같이 확률 난수의 분포를 그릴 히스토그램과 확률밀도함수를 겹쳐 그렸습니다.




```
> #Q2. Random Number with Logistic
> random.number = runif(10000, -1, 0)
> f = function(x){
+   return(exp(x) / (1 + exp(x))^2)
+ }
> logistic = log(-1-1/random.number)
> hist(logistic, xlim = c(-2, 2), freq = FALSE, breaks = 200)
> curve(f, add = TRUE)
> logistic
   [1] -1.351399029  0.064023119  0.127420764  1.865392867  0.838275635  1.112783134
   [7]  0.399545885  0.492110325 -1.162439543  0.528982346 -3.571754212 -0.054789057
  [13] -0.849106196  0.969518155 -1.080556384  2.086498600 -0.673779073 -0.051185330
  [19] -3.841529907  0.973435614 -1.122783172 -0.445088369 -2.363945007  0.931832823
  [25] -1.442638516 -0.137119393 -0.020882401 -0.625677538 -0.434631103  0.495480955
  [31]  2.740286924 -3.132635143 -1.591854140  0.560009158 -0.900870178 -2.167769963
  [37]  3.763598892  1.876799270 -3.767657705 -3.474596304 -1.273314980 -1.015786276
  [43]  0.462536180  0.258724155 -1.004971185 -1.445969400  0.101633969  1.560816054
  [49] -1.581539235 -0.807609308 -2.879554746  0.865953316  4.578933511 -1.916966800
  [55]  2.836636204  1.526446430  0.493553139  0.069996892  0.593423782 -0.716588911
  [61] -0.568488224  0.402657470 -3.860592464 -1.033779704 -0.599454028  2.082005575
  [67] -1.170700353  1.607845200  1.578818536 -1.203945716  0.428320290  1.638780961
  [73] -2.025589028  2.211732187 -0.098341152  0.517196346  1.358532806  0.158750253
  [79]  0.288878675  5.054306782 -0.222555717  0.526589490  0.507496230  0.657348509
  [85]  0.341692889  0.808879202 -1.772518910 -0.376966209  0.445998580  0.046531604
  [91]  0.300802015  0.933144761 -1.642141908  0.714121666  0.336875773  3.607922226
  [97] -2.632298499 -0.591883324  1.665542147  1.091274904 -1.547334793  1.013752178
 [103]  0.673121652  2.143410591  0.008528313  0.377542260  1.594535054 -0.124800208
```

3. 기각법에 의하여 모수가 $\alpha, \beta \geq 1$인 베타분포로부터 확률난수를 발생하는 R 함수를 작성하라. 베타분포의 확률밀도함수는

$$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, 0 < x < 1$$

이다. 여기서 $\Gamma(a)$은 감마함수로 R 함수 `gamma(a)`로 계산할 수 있다. 단 $h(x) = \alpha x^{\alpha-1}$로 했을 때 분포함수 $H(x)$와 그 역함수 $H^{-1}(x)$를 구하고 $0 \leq g(x) \leq 1$을 만족하는지도 확인하여라.

$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$이고, $h(x) = \alpha x^{\alpha-1}$이므로

$H(x) = x^{\alpha}$이고, $H^{-1}(x) = \sqrt[\alpha]{x}$ 입니다. (단, $0 < x < 1$)

$c = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} (c > 1), h(x) = \alpha x^{\alpha-1}, g(x) = \frac{(1-x)^{\beta-1}}{\alpha}$이라 하면,

$0 < x < 1$이므로 $0 < (1-x) < 1$입니다.

$\alpha \geq 1$, $\beta - 1 \geq 0$이므로 $0 \leq \frac{(1-x)^{\beta-1}}{\alpha} = g(x) \leq 1$입니다.

```
> #Q3. Beta Distribution
> #기각법을 이용해서 Gamma Distribution 에서 난수 생성
> #교재에 있는 코드를 참고했습니다.
> rgamma = function(n, alpha, beta){
+   rgamm = rep(0, n)
+   if(alpha > 1){
+     lambda = sqrt(2*alpha - 1)
+     aln4 = alpha - log(4)
+     delta = alpha + lambda
+     for(i in 1:n){
+       repeat{
+         uniform = runif(2)
+         v = log(uniform[1] / (1 - uniform[1])) / lambda
+         rgamm[i] = alpha * exp(v)
+         if(log(uniform[1]*uniform[1]*uniform[2]) ==
+            (aln4 + delta * v - rgamm[i])) break
+       }
+     }
+   }
+   else{
+     e = exp(1)
+     kappa = (e + alpha) / e
+     for(i in 1:n){
+       repeat{
+         uniform = runif(2)
+         p = uniform[1] * kappa
+         if(p>1){
+           rgamm[i] = -log((kappa - p) / alpha)
+           if(uniform[2] < rgamm[i]^(alpha-1)) break
+         }
+         else{
+           rgamm[i] = p^(1/alpha)
+           if(uniform[2] <= exp(-x)) break
+         }
+       }
+     }
+   }
+   return(beta * gamma)
+ }
```

```
> #감마분포에서 만든 난수로 베타분포 난수 생성
> gamma.to.beta = function(n, alpha, beta){
+   x1 = rgamma(n, alpha, 1)
+   x2 = rgamma(n, beta, 1)
+   return(x1/(x1 + x2))
+ }
> #h(x) = ax^(a - 1) 로 설정한 경우 기각법으로 난수를 생성하는 함수
> beta.rejection = function(n, alpha, beta){
+   h = function(x){
+     return(alpha * x^(alpha - 1))
+   }
+   H = function(x){
+     return(x^(alpha))
+   }
+   H.inverse = function(x){
+     return(x^(1/alpha))
+   }
+   random.number = c()
+   while(length(random.number) < n){
+     uniform = runif(2)
+     inverse = H.inverse(uniform[1])
+     #g(x) = ((1-x)^(beta - 1)) / alpha
+     temp = ((1 - inverse)^(beta - 1))/alpha
+     if(uniform[2] < temp){
+       random.number = c(random.number, inverse)
+     }
+   }
+   return(random.number)
+ }
```

```
> beta.rejection(10, 2, 3)
 [1] 0.2129640 0.1767583 0.5573795 0.7076829 0.6731661 0.3697761 0.1405485 0.2580570 0.6491971
[10] 0.5046442
> gamma.to.beta(10, 2, 3)
 [1] 0.6442528 0.4531156 0.3493064 0.4976191 0.4727159 0.6477228 0.3629256 0.2035626 0.5247815
[10] 0.1772705
```

감마분포를 이용한 방법과 기각벅 두 방법으로 생성된 베타분포 난수는 비슷한 분포를 가지는 것으로 추정됩니다.

4. Box-Müller법과 극좌표법에 의해 정규난수를 발생시키는 R함수를 작성하고 100,000개의 난수를 발생시켜 보아라. system.time()을 이용하여 실행시간 측면에서 어느 방법이 더 효율적인지 비교하여라.

```
> #Q4. Box-Muller Transform
> box.muller = function(n){
+   #n이 짝수인 경우에만 사용 가능
+   u1 = runif(n/2)
+   u2 = runif(n/2)
+   z1 = sqrt(-2*log(u1)) * cos(2*pi*u2)
+   z2 = sqrt(-2*log(u1)) * sin(2*pi*u2)
+   return(cbind(z1, z2))
+ }
> #Q4. Polar Method
> polar = function(n){
+   random.number = c()
+   i = 0
+   while(length(random.number)< n){
+     i = i+1
+     v1 = runif(1, -1, 1)
+     v2 = runif(1, -1, 1)
+     w.square = v1^2 + v2^2
+     if(w.square < 1){
+       z1 = v1 * sqrt(-1*log(w.square) / w.square)
+       z2 = v2 * sqrt(-1*log(w.square) / w.square)
+       random.number = c(random.number, z1, z2)
+     }
+   }
+   print(i)
+   return(random.number)
+ }
> system.time(polar(100000))
[1] 63789
 사용자  시스템 elapsed
  43.67    0.66   44.92
> head(polar(100000), 10)
[1] 63622
 [1] -0.5622733  0.6698924  0.8252688 -0.1015848  0.9294571  1.6125303 -0.4702942  0.5566273
 [9] -0.3913966  0.2121385
```

```
> system.time(box.muller(100000))
 사용자  시스템 elapsed
   0.04    0.00    0.04
> head(box.muller(100000), 10)
             z1         z2
 [1,] -1.3181098 -0.2918491
 [2,]  0.3882431 -1.1382068
 [3,] -0.6724323 -0.9148942
 [4,] -1.2292437 -1.2231041
 [5,]  0.8861350  1.0982877
 [6,] -1.0710508 -1.6081728
 [7,] -0.8822401  1.0407158
 [8,]  0.3013126 -0.5251254
 [9,] -0.2805840 -0.8621669
[10,] -0.8647902 -0.4873956
```

Box-Muller 변환 : 정규성 이탈 문제는 없지만, sin, cos 등의 복잡한 계산이 필요합니다.

극좌표 : sin, cos 계산이 필요 없으나 $w^2 \leq 1$을 만족시킬 확률은 $\frac{\pi}{4} \approx 0.785$로 100개의 난수를 발생시키려면 대학 127개 정도의 난수를 발생시켜야 합니다. 그리고, 실제 수행에서도 10만개의 난수를 발생시키기 위해서 $63789 \times 2 = 127578$개와 $63622 \times 2 = 127244$개의 난수를 발생시켜야 했습니다. system.time() 함수를 이용한 실행시간 측정은 해당 컴퓨터의 사양, 알고리즘의 효율성 등 많은 요인에 영향을 받습니다. 제가 작성한 함수에서 Box-Muller 변환에 비해 극좌표법이 훨씬 더 긴 시간이 걸린 것은 알고리즘의 효율성, 난수를 저장하는 데이터 타입의 차이 등 여러 요인이 있습니다. 하지만, 그 중에서 가장 큰 요인은 극좌표법에서 조건문과 반복문을 실행하는 시간이 오래 걸렸기 때문이었습니다. 이를 통해서, 상대적으로 R에서 sin이나 cos를 계산하는 Cost에 비해서 반복문과 조건문을 돌리는 Cost가 더 크다는 사실을 알 수 있었습니다.