Homework 1 - Monte Carlo Methods

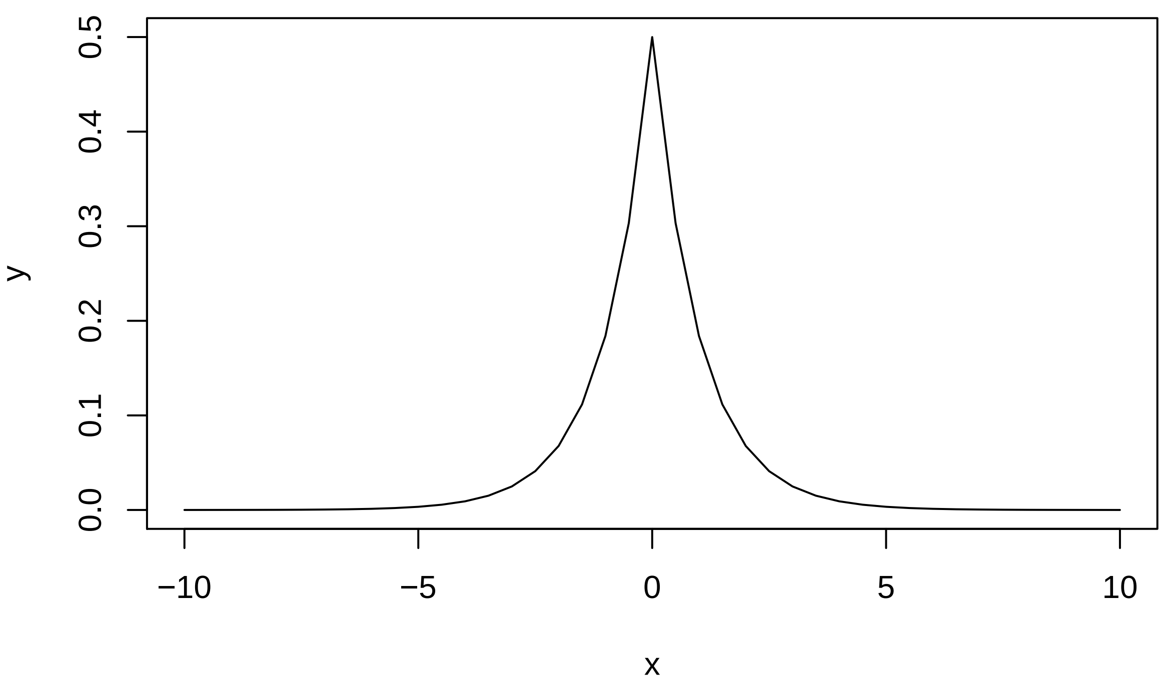
Alyssa Vanderbeek (amv2187)

Problem 1

The standard Laplace distribution has density $f(x) = 0.5e^{-|x|}, x \in (-\infty, \infty)$. Please provide an algorithm that uses the inverse transformation method to generate a random sample from this distribution. Use the $U(0,1)$ random number generator in **R**, *write a* **R**-function to implement the algorithm. Use visualization tools to validate your algorithm (i.e., illustrate whether the random numbers generated from your function truely follows the standard Laplace distribution.)

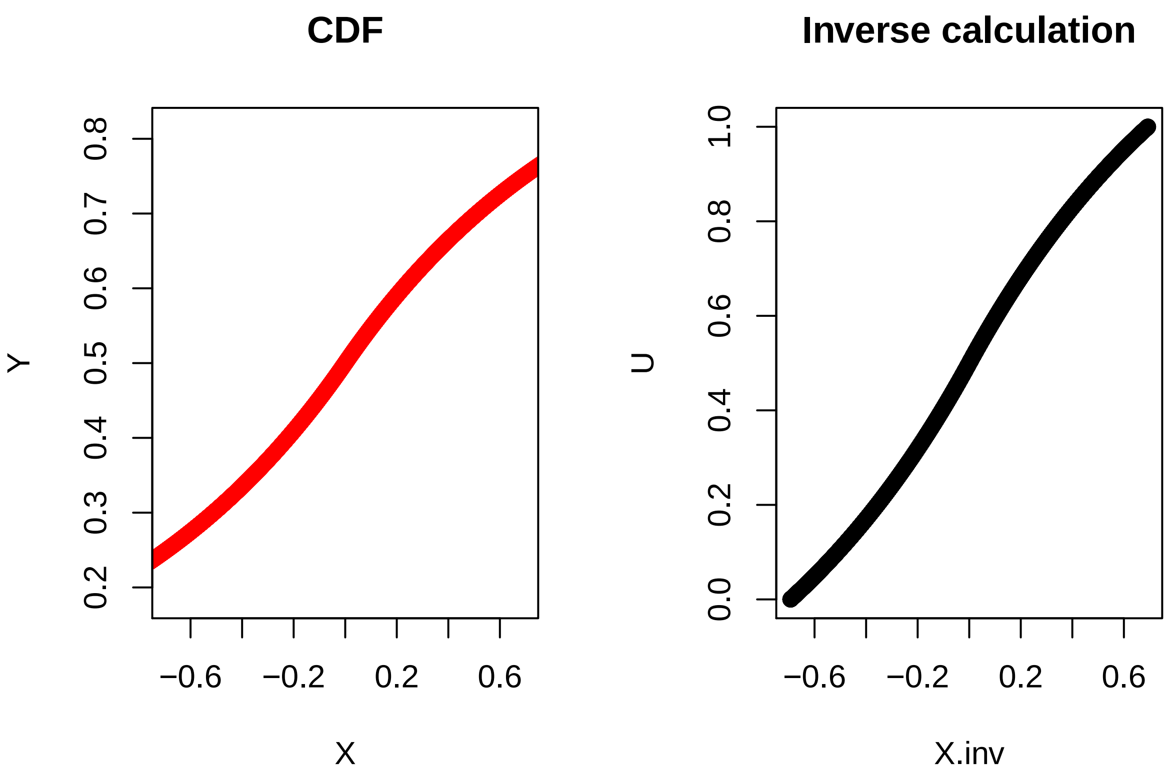
Answer: your answer starts here...

```
x = seq(-10, 10, 0.5)
y = 0.5 * exp(-abs(x))
plot(x, y, type = 'l')
```




```
N = 10000 # number of variables to simulate

set.seed(2020) # set seed
U = runif(N, min = 0, max = 1) # generate values from Uniform(0,1)

# (1) observed: calculation of X from U
X.inv = -sign(U - 0.5)*log(1 - abs(U - 0.5)) # inverse calculation of x

# (2) expected: calculation of U from X
X = sort(runif(N, -1, 1))
neg.index = which(X <= 0)
```

```r
Y = vector(mode = "numeric", length = N)
Y[neg.index] = 0.5*exp(X[neg.index])
Y[-neg.index] = 1 - 0.5*exp(-X[-neg.index])


# comparison of expected and observed
par(mfrow = c(1, 2))
plot(X, Y, type = "p", col = "red",
     main = "CDF",
     xlim = range(X.inv))
plot(X.inv, U, type = "p",
     main = "Inverse calculation")
```



Problem 2

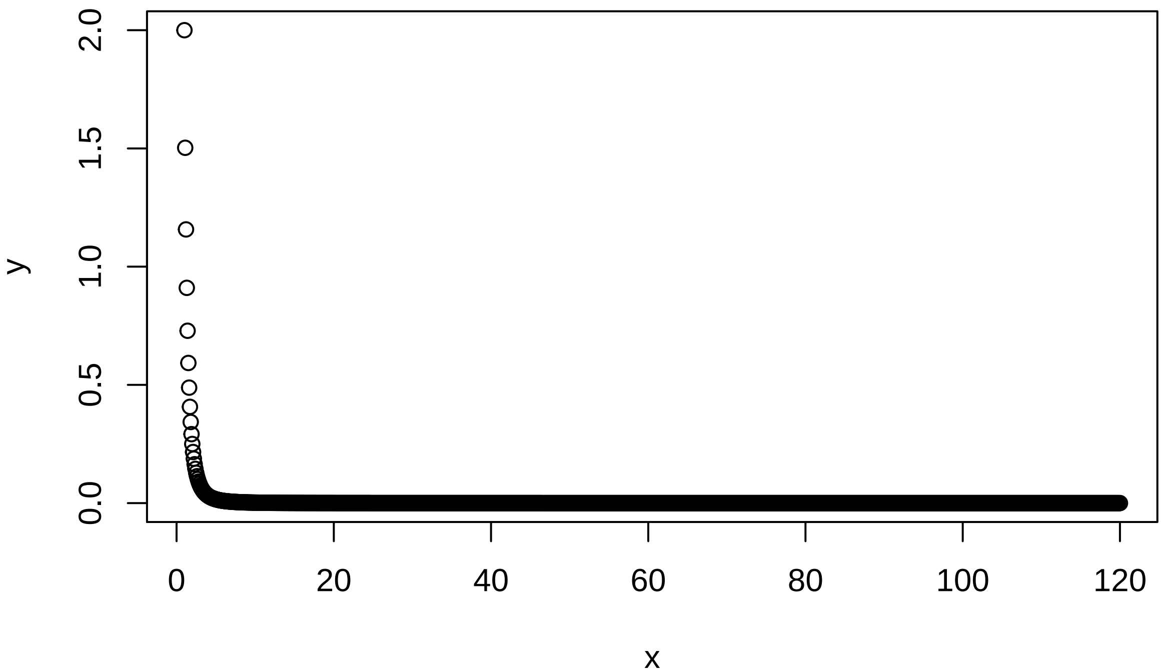
Use the inverse transformation method to derive an algorithm for generating a Pareto random number with $U \sim U(0, 1)$, where the Pareto random number has a probability density function

$$f(x; \alpha, \gamma) = \frac{\gamma \alpha^{\gamma}}{x^{\gamma+1}} I\{x \geq \alpha\}$$

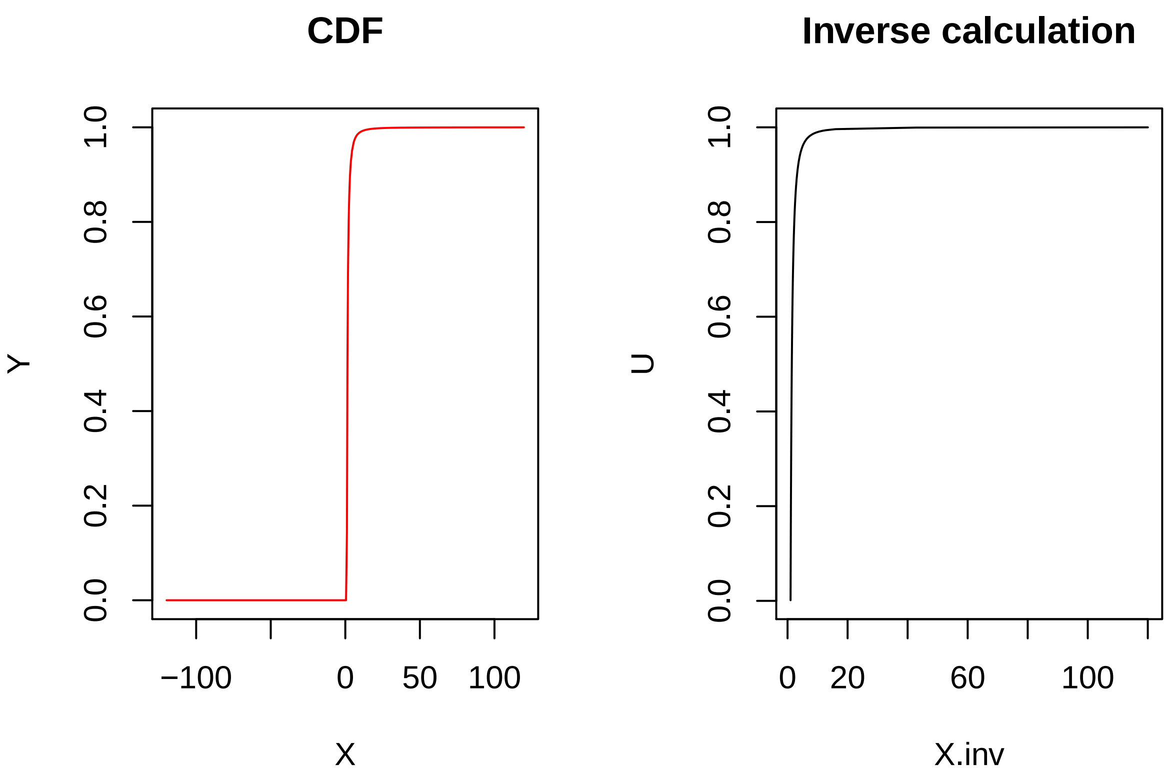
with two parameters $\alpha > 0$ and $\gamma > 0$. Use visualization tools to validate your algorithm (i.e., illustrate whether the random numbers generated from your function truely follows the target distribution.)

Answer: your answer starts here. . .

```r
alpha = 1
gamma = 2

x = seq(alpha, 120, 0.1) # x must be >= alpha
y = ((gamma * alpha^gamma) / x^(gamma + 1)) # true distribution
plot(x, y, type = 'p')
```




```r
N = 1000
set.seed(1)
U = sort(runif(N, min = 0, max = 1)) # generate values from Uniform(0,1)
X.inv = alpha / pracma::nthroot(1 - U, gamma) # plug U in to inverse function

# direct generation of CDF
X = sort(runif(N, -120, 120))
Y = (X >= alpha)*(1 - (alpha/X)^gamma)

# comparison of expected and observed
par(mfrow = c(1, 2))
plot(X, Y, type = "l", col = "red",
     main = "CDF")
plot(X.inv, U, type = "l",
     main = "Inverse calculation")
```




```r
#legend(x = 0.2, y = 0.4, legend = c("Inverse calculation", "CDF"),
#       col = c("black", "red"), pch = rep(19, 2))
```

Problem 3

Construct an algorithm for using the acceptance/rejection method to generate 100 pseudorandom variable from the pdf

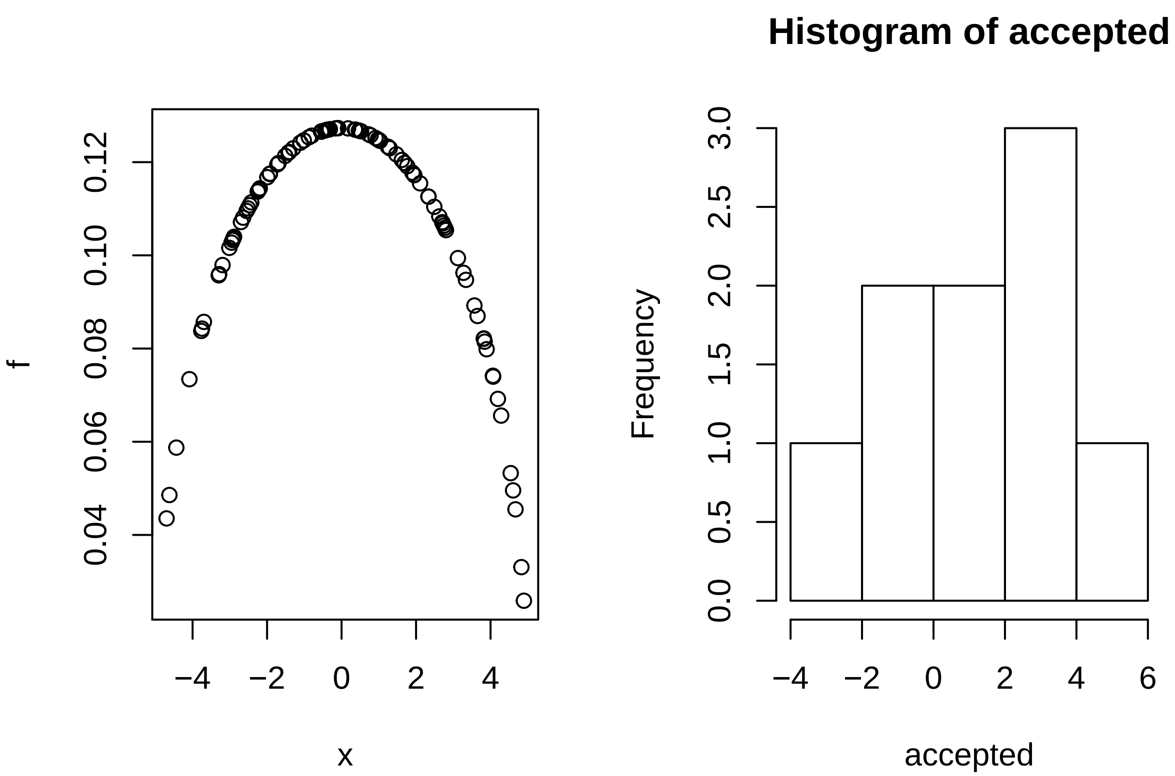
$$f(x) = \frac{2}{\pi\beta^2}\sqrt{\beta^2 - x^2},\ -\beta \leq x \leq \beta.$$

The simplest choice for $g(x)$ is the $U(-\beta, \beta)$ distribution but other choices are possible as well. Use visualization tools to validate your algorithm (i.e., illustrate whether the random numbers generated from your function truely follows the target distribution.)

Answer: your answer starts here…

```r
n.sim = 100
beta = 5

set.seed(100)
# choose value of M, which is the supremum of f/g
x = runif(n.sim, -beta, beta)
f = 2/(pi*(beta^2)) * sqrt(beta^2 - x^2)
g = 1/(2*beta)
M = 2*beta #max(f/g)

y = runif(n.sim, -beta, beta)
```

```r
u = runif(n.sim)

fdens = function(x){
  y = 2/(pi*(beta^2)) * sqrt(beta^2 - x^2)
  return(y)
}

accepted = NULL
for (i in 1:n.sim) {
  ff = fdens(x[i])
  g = 1/(2*beta)
  if (u[i] <= (ff / (M*g))) {
    accepted = c(accepted, x[i])
  }
}

par(mfrow = c(1, 2))
plot(x, f)
hist(accepted)
```



Problem 4

Develop two Monte Carlo methods for the estimation of $\theta = \int_0^1 e^{x^2} dx$ and implement in **R** .

Answer: your answer starts here...

```r
# Method 1: for loop
N = 10000
g = numeric(N)
for (i in 1:N) {
  x = runif(1)
  g[i] = exp(x^2)
}
mean(g)
```

```
## [1] 1.462016
```

```r
# Method 2: trapezoidal integration
N = 10000
u = runif(N)
mean(exp(u^2))
```

```
## [1] 1.45612
```

The two methods above estimate the integrl as 1.4620155 and 1.4561198.

Problem 5

Show that in estimating $\theta = E\sqrt{1-U^2}$ *it is better to use* U^2 *rather than* U *as the control variate, where* $U \sim U(0,1)$*. To do this, use simulation to approximate the necessary covariances. In addition, implement your algorithms in* **R**.

Answer: your answer starts here...

```r
#Your R codes/functions
gfun = function(x) sqrt(1 - x^2)
mfun1 = function(x) x^2 # cor(m1, g) = -0.98
mfun2 = function(x) x # cor(m2, g) = -0.92

set.seed(123)
N = 10000
U = runif(N)
g = gfun(U)
m1 = mfun1(U)
m2 = mfun2(U)

beta_1 = lm(g ~ m1)$coef[2]
beta_2 = lm(g ~ m2)$coef[2]

theta_g = mean(g)
hha_1 = beta_1*(1/3) + (g - beta_1*m1)
theta_m1 = mean(hha_1) # U^2
hha_2 = beta_2*(1/2) + (g - beta_2*m2)
theta_m2 = mean(hha_2) # U
```

```r
var.u2 = (var(g) - var(hha_1))/var(g) # variance reduction by using U^2
var.u = (var(g) - var(hha_2))/var(g) # variance reduction by using U
```

The variance reduction by using U^2 is 0.9673335 vs. 0.8479443 for U. Therefore, U^2 is the preferred control variate.

Problem 6

Obtain a Monte Carlo estimate of

$$\int_1^\infty \frac{x^2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$$

by importance sampling and evaluate its variance. Write a **R** *function to implement your procedure.*

Answer: your answer starts here...

```r
## Monte Carlo with importance sampling

problem_6 = function(){
  N = 10000
  x = rnorm(N, 0, 2) # importance distribution is N(0, 2)
  f = (x^2/sqrt(2*pi))*exp(-(x^2)/2)
  g = dnorm(x, 0, 2)

  return(c(estimate = sum((x >= 1)*f/g)/N,
           variance = var(f/g)/N
  ))
}

set.seed(123)
problem_6()
```

```
##      estimate     variance
## 0.4000445665 0.0000482617
```