Homework 5

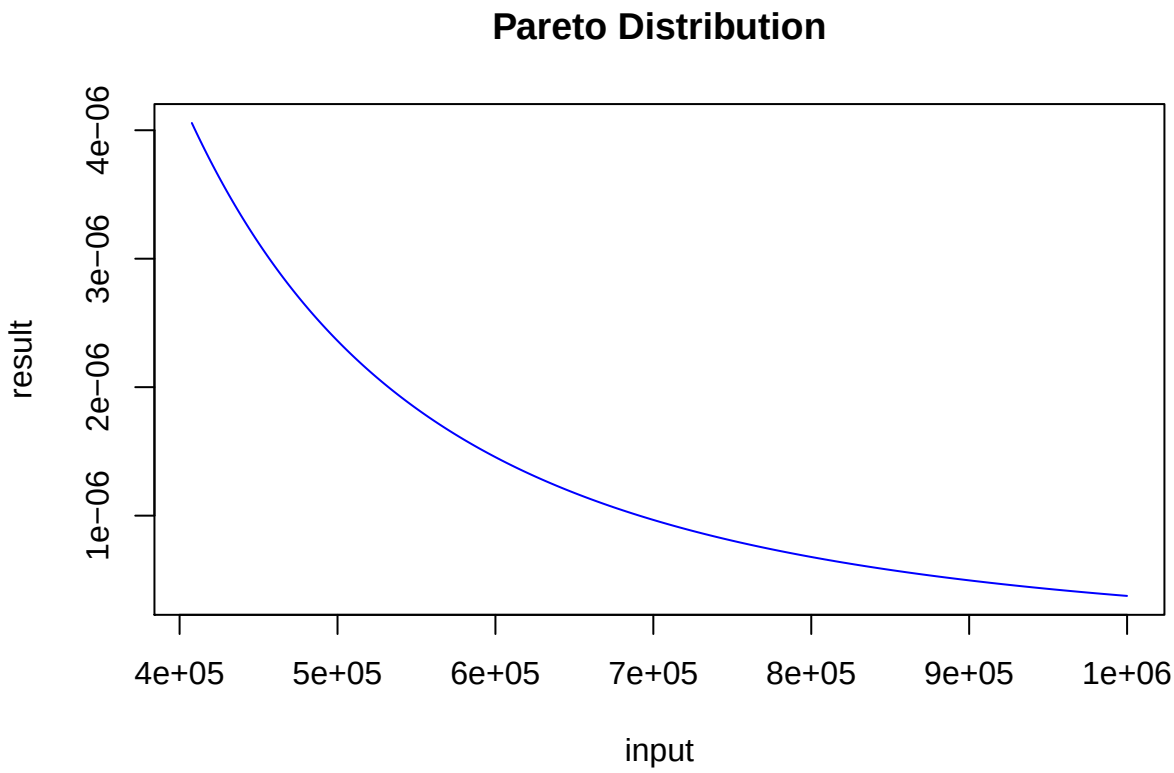
Christine Chong cc4190

June 23, 2017

```
wtid <- read.csv("wtid-report.csv", header = TRUE, sep=",")
```

Perform the following tasks: 1. Define a function f which takes three inputs x, a vector, and scalars a and xmin having default values of a = ^a and xmin = $407, 760. The function should output f(x) for a given input x > xmin. Plot the function between xmin and 1, 000, 000. Make sure your plot is labeled appropriately.

```
f<- function(x, a = 2.654 , xmin = 407760){
   return(ifelse(x<xmin, 0, ((a-1)/xmin)*(x/xmin)^-a))
}
x<- seq(407760,1000000, by=100)
plot(x,f(x), type ="l", col ="blue", main = "Pareto Distribution", xlab = "input", ylab = "result")
```



2. Find the inverse function F and define a function upper.income. The function should have three inputs u, a vector, and scalars a and xmin taking default values of a = ^a and xmin = $407, 760.

```
upper.income <- function(u, a=2.654, xmin = 407760){
  if(0<u | u<1 ){
    y = xmin*(1-u)^(-1/(a-1))
    return(y)
  }else{
    return(NA)
```

```
  }
}
upper.income(.5)
```

```
## [1] 620020.2
```

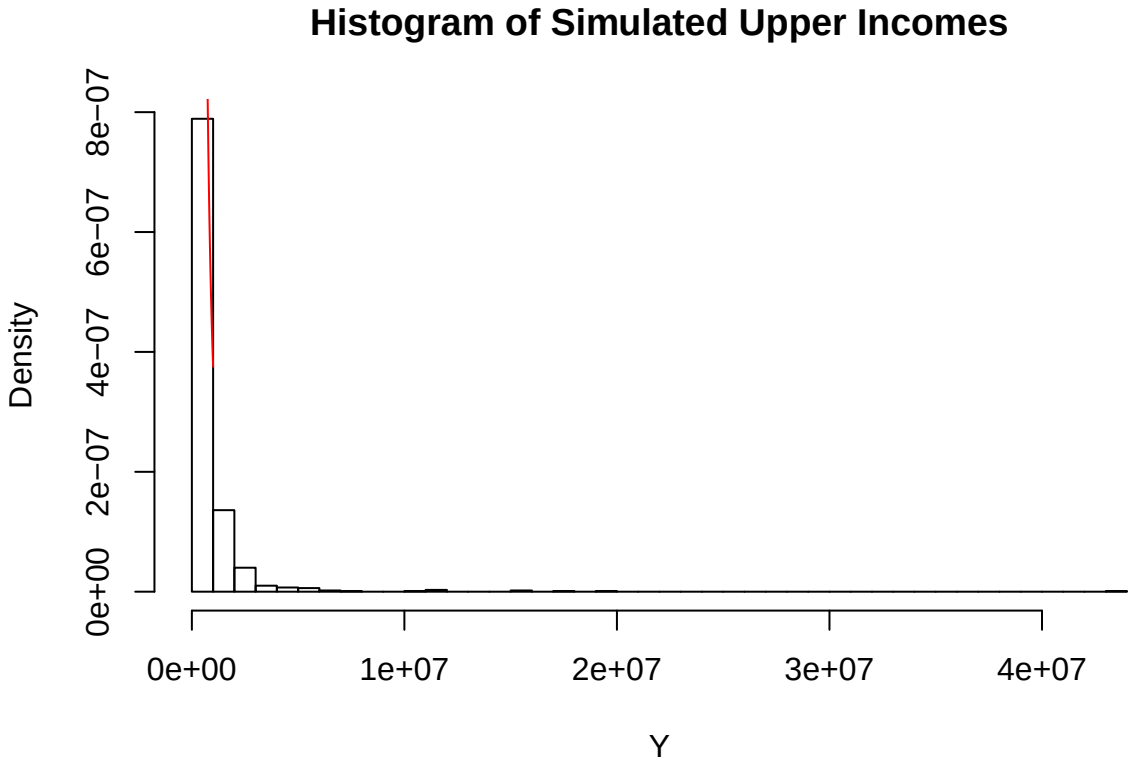
3. Using the Inverse Transform Method, simulate 1000 draws from the Pareto distribution

(1) and plot a histogram of your values. Overlay the simulated distribution with the Pareto density (1). Make sure to label the histogram appropriately.

```
U <- runif(1000)
Y <- upper.income(U)
```

```
## Warning in if (0 < u | u < 1) {: the condition has length > 1 and only the
## first element will be used
```

```
hist(Y, probability = T, breaks = 50, main ="Histogram of Simulated Upper Incomes")
lines(x,f(x), col ="red")
```

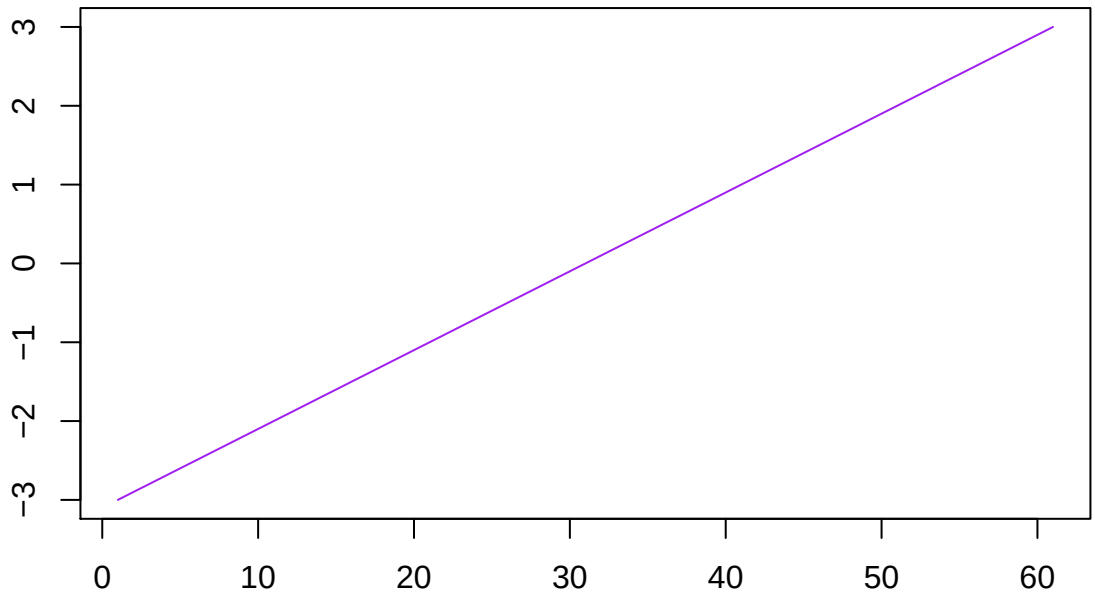


4. Using your simulated set, estimate the median income for the richest 1% of the world. Recall from lab that the proportion of people whose income is at least xmin whose income is also at or above any level w ??? xmin is Compare your estimated 50th percentile to the actual 50th percentile of the Pareto distribution.

Part 2: Reject-Accept Method Perform the following tasks: 5. Write a function f that takes as input a vector x and returns a vector of f(x) values. Plot the function between ???3 and 3. Make sure your plot is labeled appropriately.

```
f <- function(x){
  if(-1<=x && x<=2){
```

```
    return(1/9(4-x^2))
  }
}
x <- seq(-3,3, by=0.1)
plot(x, f(x), type ="l", col = "purple", xlab = "", ylab="" )
```



6. Determine the maximum of f(x) and find an envelope function e(x) by using a uniform density for g(x). Write a function e which takes as input a vector x and returns a vector of e(x) values. 7. Using the Accept-Reject Algorithm, write a program that simulates 1000 draws from the probability density function f(x) from Equation 2.

```
#draws <- 1000
#accepted <- 0
#samps <- numeric(draws)
#while(accepted < draws){
 # y <- upper.income(runif(1))
#  u <- runif(1)
 # if(u<f(y)/e(y)){
  #  accepted <- accepted+1
  #  samps[accepted] <- y
  #}
#}
```

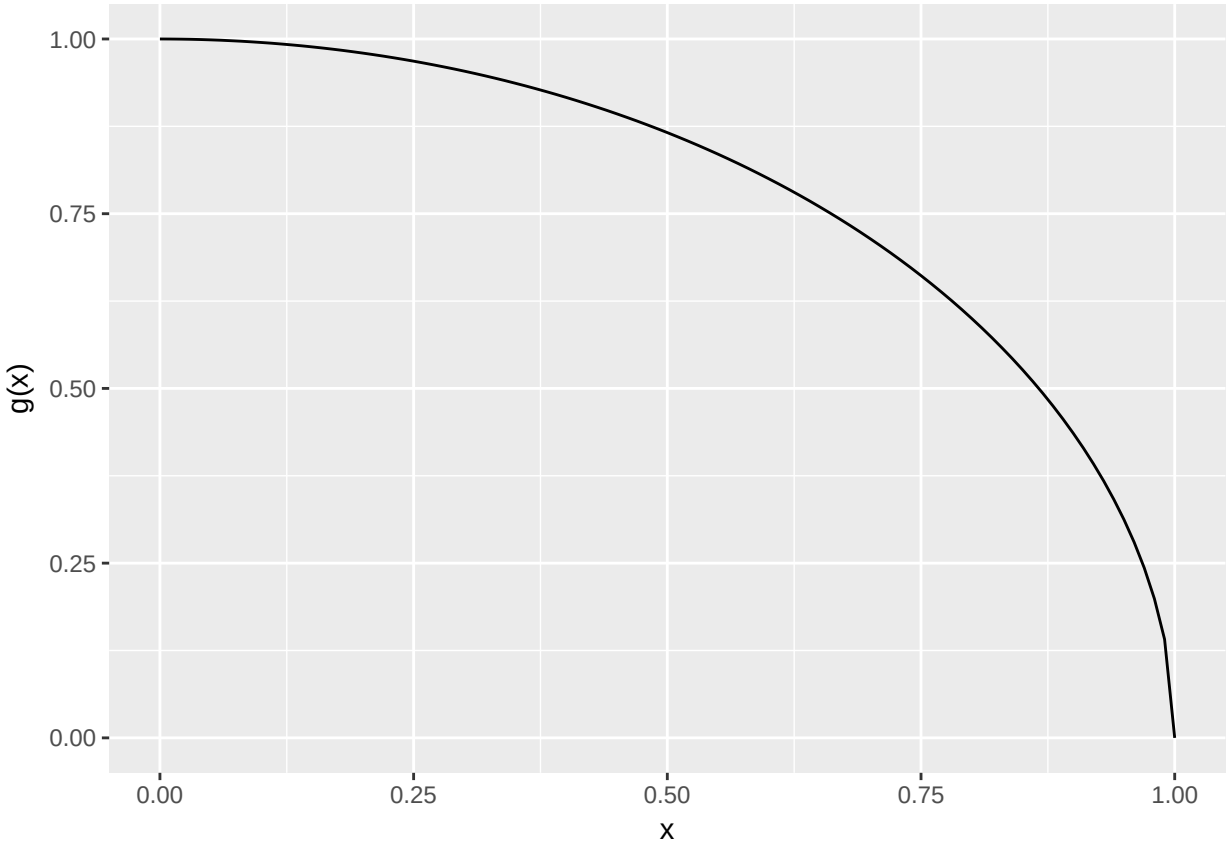
8. Plot a histogram of your simulated data with the density function f overlaid in the graph. Label your plot appropriately. 2 Part 3: Simulation with Built-in R Functions Consider the following "random walk" procedure: . Start with x = 5 . Draw a random number r uniformly between ???2 and 1. . Replace x with x + r . Stop if x ??? 0 . Else repeat Perform the following tasks:

9. Write a while() loop to implement this procedure. Importantly, save all the positive values of x that were visited in this procedure in a vector called x.vals, and display its entries.
10. Produce a plot of the random walk values x.vals from above versus the iteration number. Make sure the plot has an appropriately labeled x-axis and and y-axis. Also use type="o" so that we can see both points and lines.
11. Write a function random.walk() to perform the random walk procedure that you implemented in question (9). Its inputs should be: x.start, a numeric value at which we will start the random walk, which takes a default value of 5; and plot.walk, a boolean value, indicating whether or not we want to produce a plot of the random walk values x.vals versus the iteration number as a side effect, which takes a default value of TRUE. The output of your function should be a list with elements: x.vals, a vector of the random walk values as computed above; and num.steps, the number of steps taken by the random walk before terminating. Run your function twice with the default inputs, and then twice times with x.start equal to 10 and plot.walk = FALSE.
12. We'd like to answer the following question using simulation: if we start our random walk process, as defined above, at x = 5, what is the expected number of iterations we need until it terminates? To estimate the solution produce 10, 000 such random walks and calculate the average number of iterations in the 10, 000 random walks you produce. You'll want to turn the plot off here.
13. Modify your function random.walk() defined previously so that it takes an additional argument seed: this is an integer that should be used to set the seed of the random number generator, before the random walk begins, with set.seed(). But, if seed is NULL, the default, then no seed should be set. Run your modified function 3 random.walk() function twice with the default inputs, then run it twice with the input seed equal to (say) 33 and plot.walk = FALSE. Part 4: Monte Carlo Integration The goal of this exercise is to estimate the mathematical constant ?? using Monte Carlo Integration. Consider the function

(3) g(x) = (??? 1 ??? x 2 0 ??? x ??? 1 0 otherwise The above function traces out a quartile circle over the interval [0, 1]. Perform the following tasks:

14. Run the following code:

```
g <- function(x) {
return(sqrt(1-x^2))
}
library(ggplot2)
ggplot() +
geom_line(aes(x=seq(0,1,.01),y=g(seq(0,1,.01))))+
labs(x="x",y="g(x)")
```



The above code should produce the plot of a quarter circle. 15. Identify the true area under the curve g(x) by using simple geometric formulas. 16. Using Monte Carlo Integration, approximate the mathematical constant ?? within a 1/1000 of the true value. When performing this simulation, make sure to choose a probability density function that has support over the unit interval, i.e. uniform(0,1) or beta(??,??).