Homework 6 Solutions

Cynthia Rush (cgr2130)

November 7, 2016

Part 1: Inverse Transform Method

We continue working with the World Top Incomes Database, and the Pareto distribution, as in Lab 4 and Homework 5. Recall that for most countries in most time periods, the upper end of the income distribution roughly follows a Pareto distribution, with probability density function

$$f(x) = \frac{(a-1)}{x_{min}} \left(\frac{x}{x_{min}} \right)^{-a}$$

for incomes $X \geq x_{min}$. In Homework 5, we estimated the parameter a based on the `wtid-report` dataset for years ranging from 1913 to 2015.

Now suppose that we are interested in simulating the upper end of income just for 2015 using the Pareto distribution written above. Let the 'upper end' begin at the 99th annual income percentile for 2015 (meaning, let $x_{min} = \$407,760$). Then we can model the upper end of income for 2015 by the Pareto distribution having pdf

$$f(x) = \frac{\hat{a}-1}{x_{min}} \left(\frac{x}{x_{min}} \right)^{-\hat{a}} = \frac{1.654}{407760} \left(\frac{x}{407760} \right)^{-2.654}$$

using $\hat{a} = 2.654$ which was the Pareto exponent estimate for 2015 from Homework 5.

Perform the following tasks:

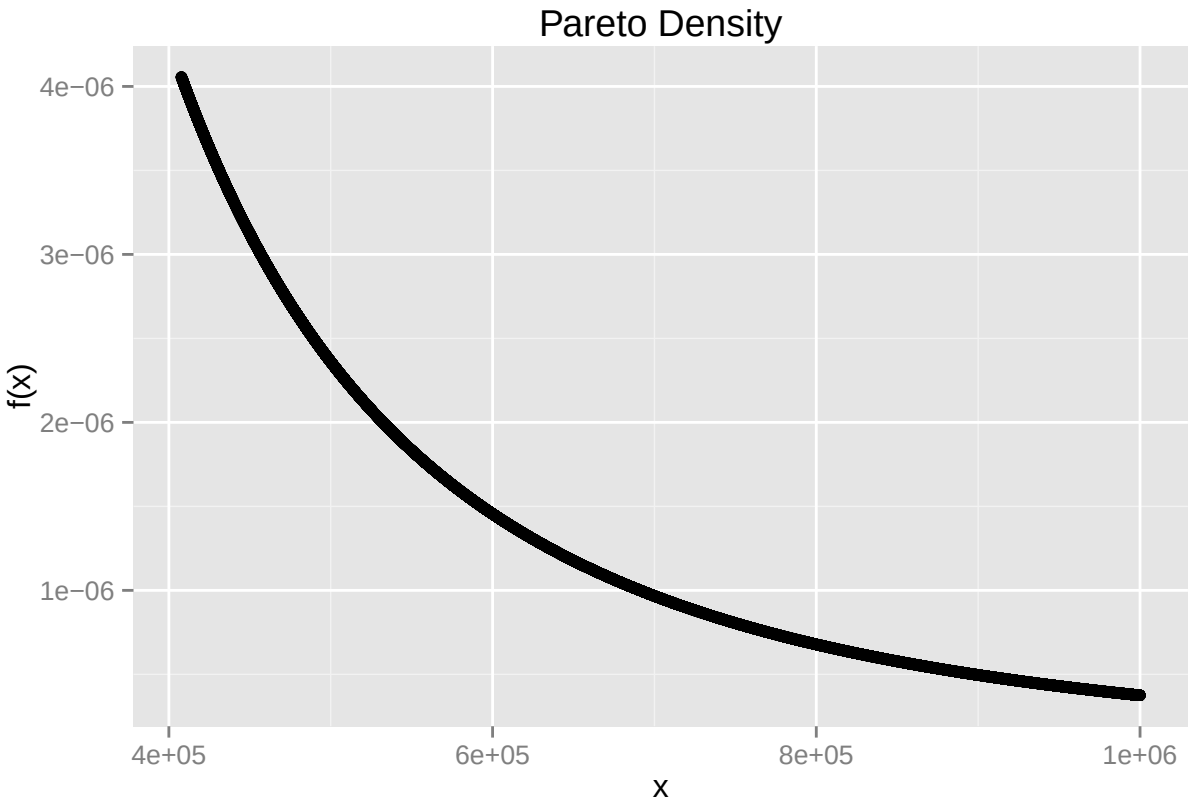
(1) Define a function `f` which takes three inputs x, a vector and scalars a and x_{min} having default values of $a = \hat{a}$ and $x_{min} = \$407,760$. The function should output $f(x)$ for a given input $x > x_{min}$. Plot the function between x_{min} and $1,000,000$. Make sure your plot is labeled appropriately.

```
f <- function(x, xmin = 407760, a = 2.654) {
  return(ifelse(x < xmin, 0, ((a-1)/xmin)*(x/xmin)^(-a)))
}

xmin <- 407760
x <- seq(xmin, 1000000, by = 10)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.1.3
```

```
ggplot() +
  geom_point(mapping = aes(x = x, y = f(x))) +
  labs(title = "Pareto Density")
```



(2) For $x > x_{min}$, the cdf equals

$$F(x) = 1 - \left(\frac{x}{x_{min}}\right)^{-a-1}$$

Find the inverse function $F^{-1}(u)$ and define a function `upper.income`. The function should have three inputs, a vector u and scalars a and x_{min} having default values of $a = \hat{a}$ and $x_{min} = \$407,760$. The function should output $F^{-1}(u)$ for a given input $u \in (0, 1)$. Make sure *upper.income*(.5) returns 620020.2.

$$u = 1 - \left(\frac{x}{x_{min}}\right)^{-\hat{a}+1} \rightarrow x = x_{min}(1-u)^{-\frac{1}{\hat{a}-1}}$$

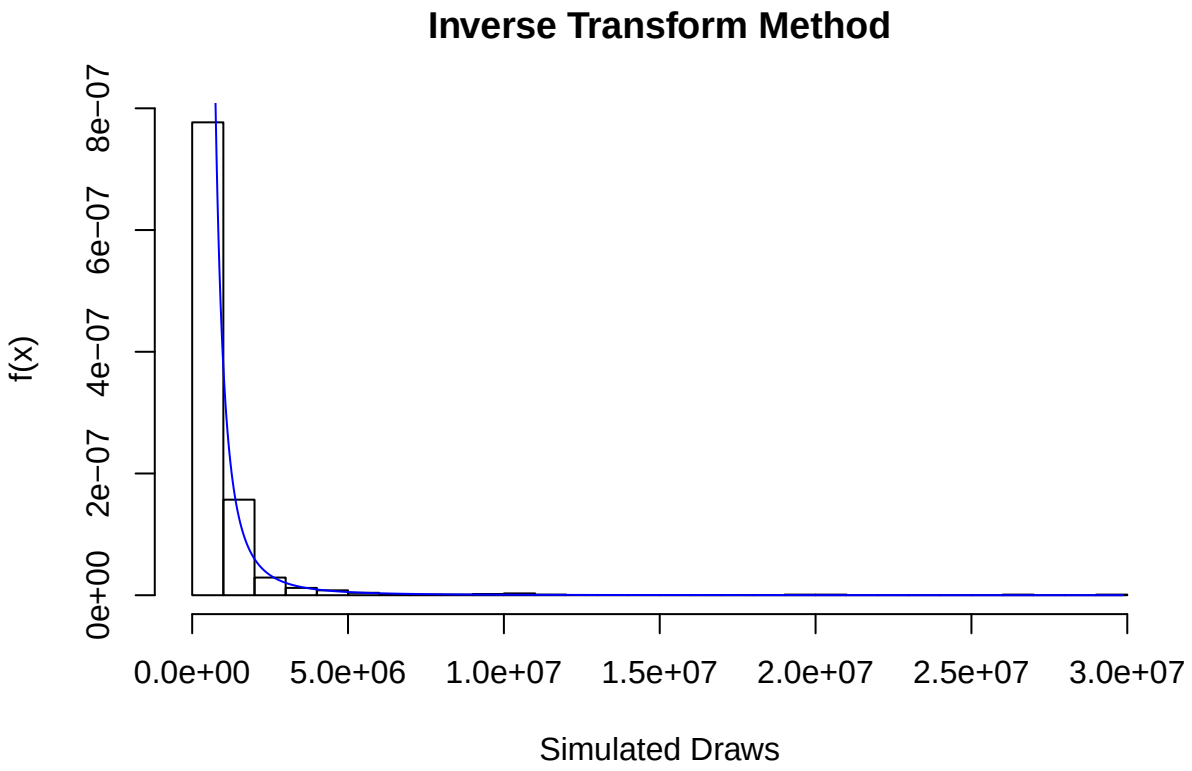
```
upper.income <- function(u, xmin = 407760, a = 2.654) {
  return(ifelse((u < 0 | u > 1), NA, xmin*(1-u)^(-1/(a - 1))))
}
upper.income(0.5)
```

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

(3) Using the *Inverse Transform Method*, simulate 1000 draws from the Pareto distribution (**??**) and plot a histogram of your values. Overlay the simulated distribution with the Pareto density (**??**). Make sure to label the histogram appropriately.

```r
n    <- 1000
sims <- upper.income(runif(n))

hist(sims, prob = TRUE, breaks = 35, xlab = "Simulated Draws", ylab = "f(x)", main = "Inverse Transform
y <- seq(min(sims), max(sims), by = 10000)
lines(y, f(y), col = "blue")
```



(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 x_{min} whose income is also at or above any level $w \geq x_{min}$ is

$$\mathbf{Pr}(X \geq w) = \left(\frac{w}{x_{min}}\right)^{-a+1}.$$

Compare your estimated 50th percentile to the actual 50th percentile of the Pareto distribution.

The actual 50th percentile of the Pareto distribution is given by

$$.5 = \left(\frac{w}{x_{min}}\right)^{-a+1} \rightarrow w = x_{min}(.5)^{\frac{1}{-a+1}}$$

```r
ahat <- 2.654
a    <- ahat
sim_med    <- median(sims)
actual_med <- xmin*(.5)^(1/(-a+1))
sim_med; actual_med
```

```
## [1] 625080.6
```

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

Part 2: Reject-Accept Method

Let random variable X denote the temperature at which a certain chemical reaction takes place. Suppose that X has probability density function

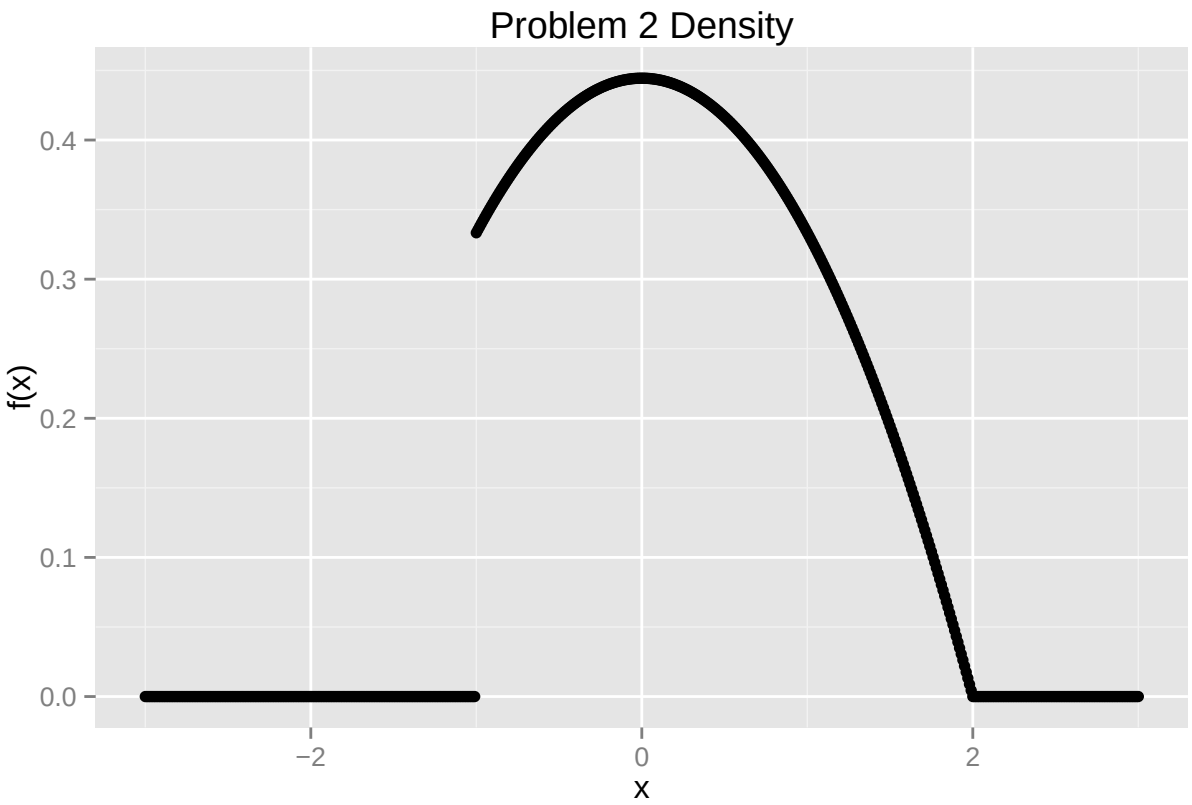
$$f(x) = \begin{cases} \frac{1}{9}(4 - x^2) & -1 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

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) {
  ifelse((x < -1 | x > 2), 0, (1/9)*(4 - x^2))
}

x <- seq(-3, 3, by = .01)
ggplot() +
  geom_point(mapping = aes(x = x, y = f(x))) +
  labs(title = "Problem 2 Density")
```



(6) Determine the maximum of $f(x)$ and find an envelope function $e(x)$ by using a uniform density for $g(x)$ and setting $\alpha = 1/\max_x\{f(x)\}$ as in class. Write a function `e` which takes as input a vector `x` and returns a vector of `e(x)` values.

Note that

$$f'(x) = \begin{cases} -\frac{2x}{9} & -1 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

Setting the above equal to 0 we find that the maximum occurs at $x = 0$ and $f(0) = 4/9$.

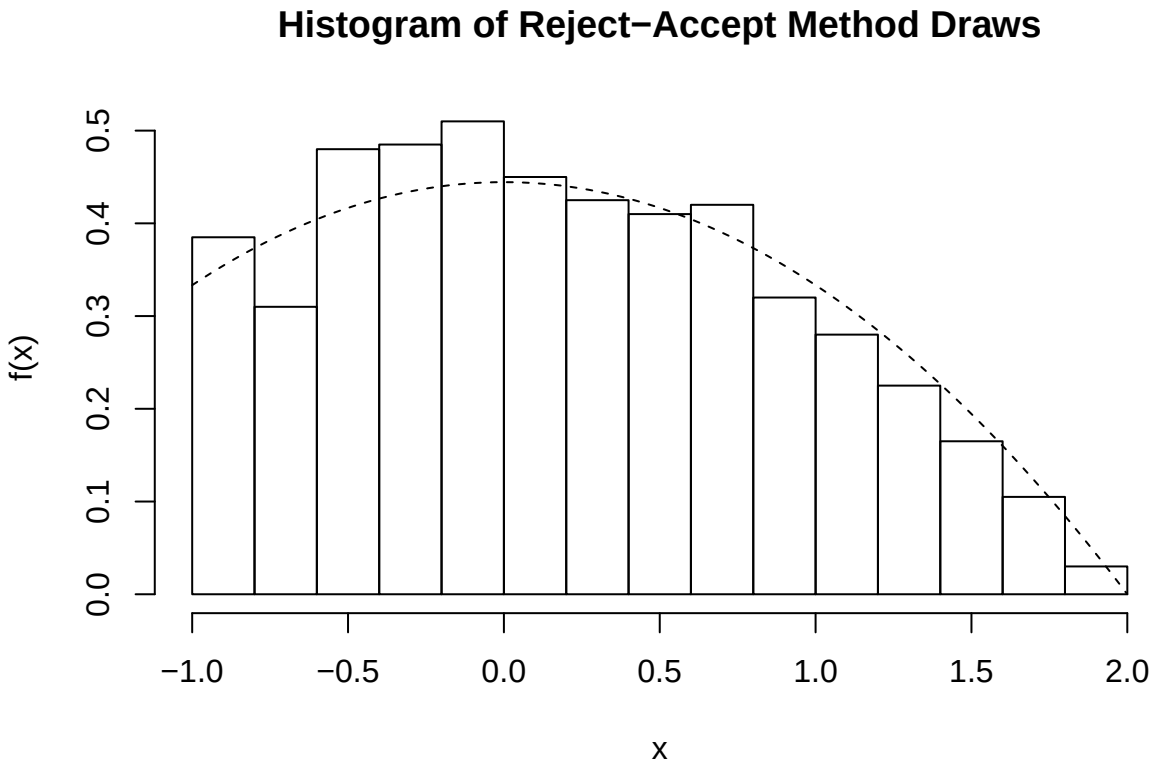
```
f.max <- 4/9
e <- function(x) {
  ifelse((x < -1 | x > 2), Inf, f.max)
}
```

(7) Using the *Accept-Reject Algorithm*, write a program that simulates 1000 draws from the probability density function $f(x)$ from Equation **??**.

```
n.samps <- 1000   # Samples desired
n       <- 0      # counter for number samples accepted
samps   <- numeric(n.samps) # initialize the vector of output
while (n < n.samps) {
  y <- runif(1, min = -1, max = 2)    # random draw from g
  u <- runif(1)
  if (u < f(y)/e(y)) {
    n        <- n + 1
    samps[n] <- y
  }
}
```

(8) Plot a histogram of your simulated data with the density function f overlaid in the graph. Label your plot appropriately.

```
x <- seq(-1, 2, length = 100)
hist(samps, prob = T, ylab = "f(x)", xlab = "x", main = "Histogram of Reject-Accept Method Draws")
lines(x, f(x), lty = 2)
```



Part 3: Simulation with Built-in R Functions

Consider the following "random walk" procedure:

a. Start with $x = 5$
b. Draw a random number r uniformly between -2 and 1.
c. Replace x with $x + r$
d. Stop if $x \leq 0$
e. Else repeat

Perform the following tasks:

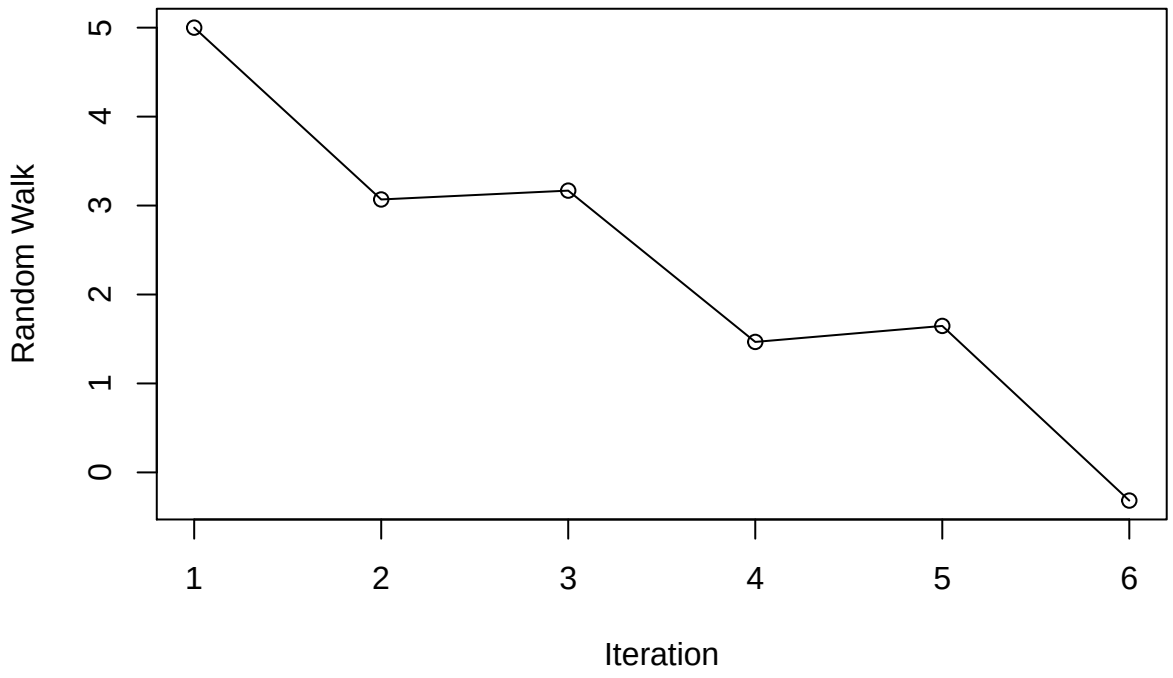
(9) Write a `while()` loop to implement this procedure. Importantly, save all the positive values of x that were visited in this procedure in a vector called `x.vals`, and display its entries.

```
x.vals <- 5
i <- 1
while(x.vals[i] > 0) {
  r <- runif(1, min = -2, max = 1)
  x.vals <- c(x.vals, x.vals[i] + r)
  i <- i+1
}
x.vals
```

```
## [1]  5.0000000  3.0683868  3.1679399  1.4670255  1.6468158 -0.3165256
```

(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.

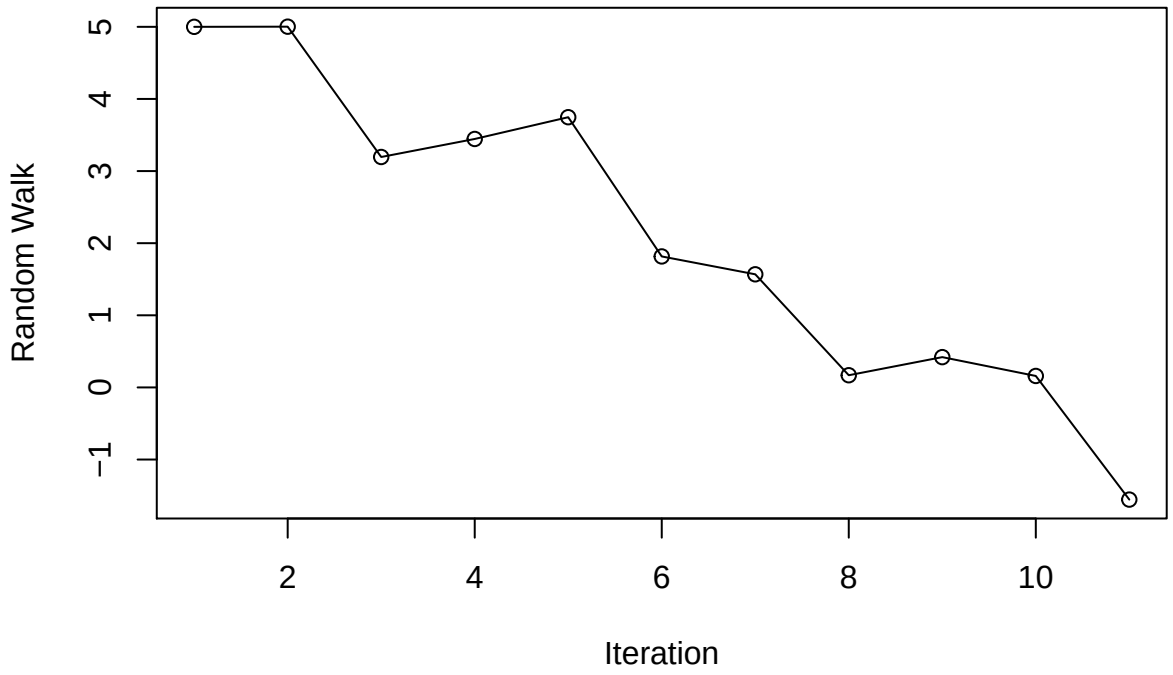
```
plot(1:length(x.vals), x.vals, type = "o", xlab = "Iteration", ylab = "Random Walk")
```



(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`.

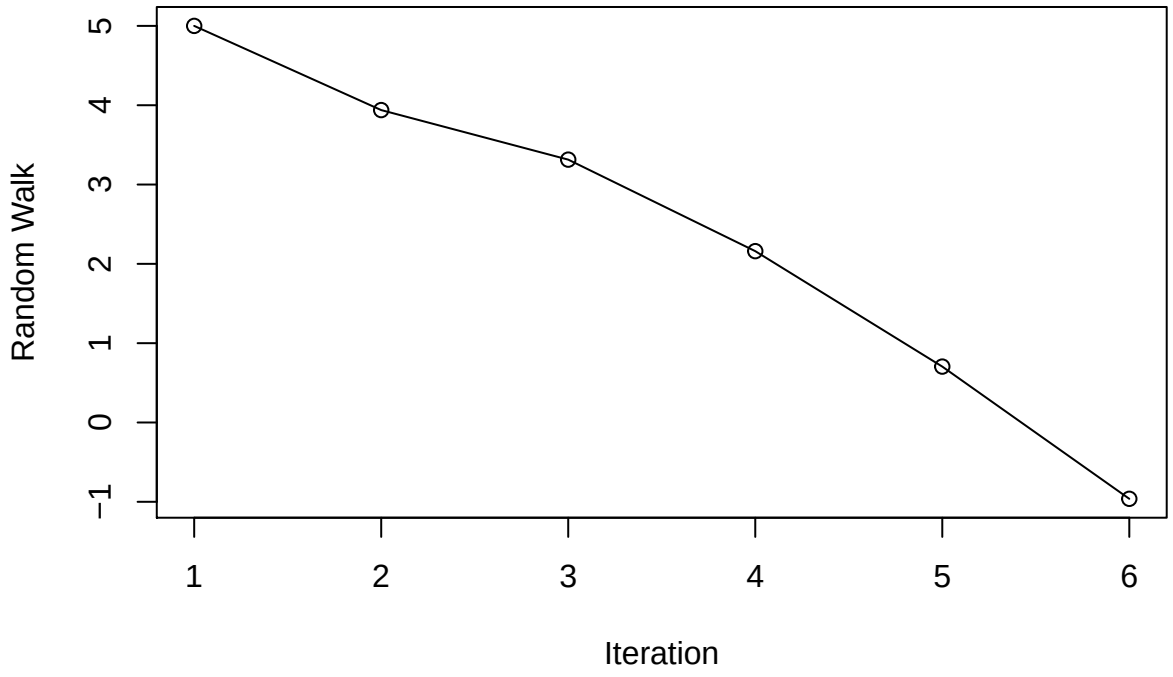
```
random.walk <- function(x.start = 5, plot.walk = TRUE) {
  x.vals <- 5
  i <- 1
  while(all(x.vals[i] > 0)) {
    r <- runif(1, min = -2, max = 1)
    x.vals <- c(x.vals, x.vals[i] + r)
    i <- i+1
  }
  if (plot.walk) {plot(1:length(x.vals), x.vals, type = "o", xlab = "Iteration", ylab = "Random Walk")}
  return(list(x.vals = x.vals, num.steps = i))
}

random.walk()
```




```
## $x.vals
##  [1]  5.0000000  5.0020346  3.1951972  3.4451838  3.7468971  1.8161659
##  [7]  1.5686613  0.1687576  0.4200910  0.1590635 -1.5551778
##
## $num.steps
## [1] 11
```

```
random.walk()
```




```
## $x.vals
```

```
## [1]  5.0000000  3.9386230  3.3139500  2.1601050  0.7047665 -0.9621036
##
## $num.steps
## [1] 6
```

```
random.walk(x.start = 10, plot.walk = FALSE)
```

```
## $x.vals
##  [1]  5.0000000  3.3566405  3.0595386  2.4131730  1.3230240  1.8098644
##  [7]  2.8063637  2.2181567  0.3672678 -0.7808803
##
## $num.steps
## [1] 10
```

```
random.walk(x.start = 10, plot.walk = FALSE)
```

```
## $x.vals
## [1]  5.0000000  3.1168751  1.4840327 -0.2007009
##
## $num.steps
## [1] 4
```

(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.

```
n     <- 10000
iters <- rep(NA, n)
for (i in 1:n) {
  iters[i] <- random.walk(plot.walk = FALSE)$num.steps
}
mean(iters)
```

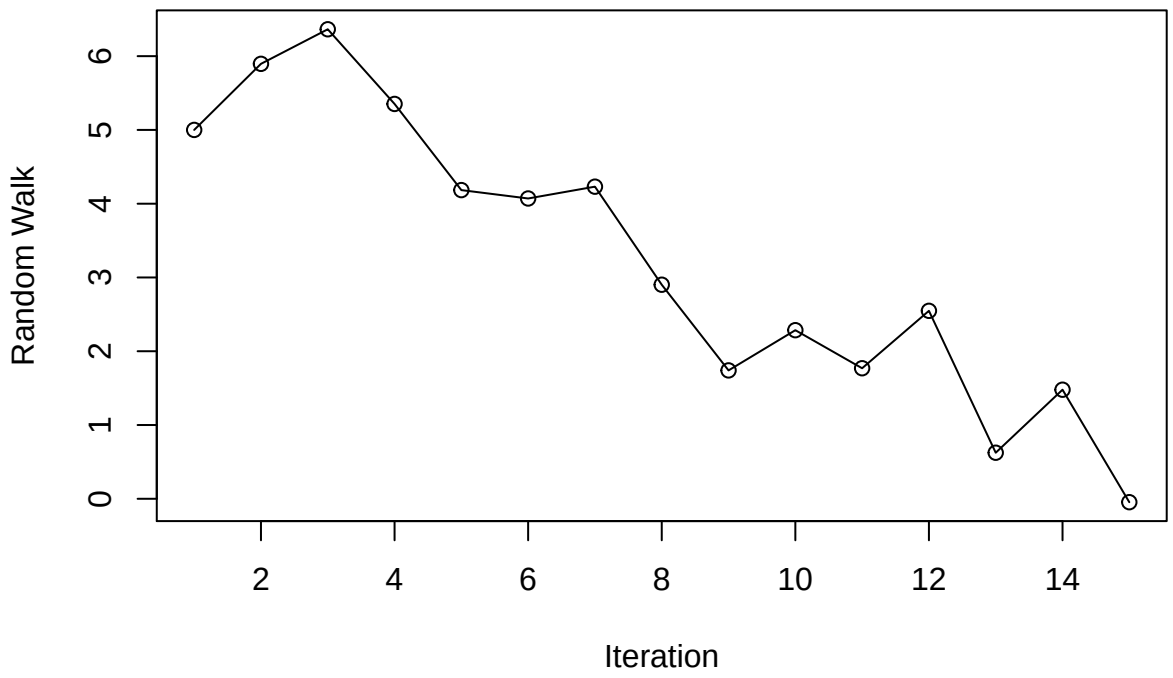
```
## [1] 12.3393
```

(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 `random.walk()` function several times with the default inputs, then run it several times with the input seed equal to (say) 33.

```
random.walk <- function(x.start = 5, plot.walk = TRUE, seed = NULL) {
  if (!is.null(seed)) {set.seed(seed)}
  x.vals <- 5
  i <- 1
  while(all(x.vals[i] > 0)) {
    r <- runif(1, min = -2, max = 1)
    x.vals <- c(x.vals, x.vals[i] + r)
    i <- i+1
  }
```

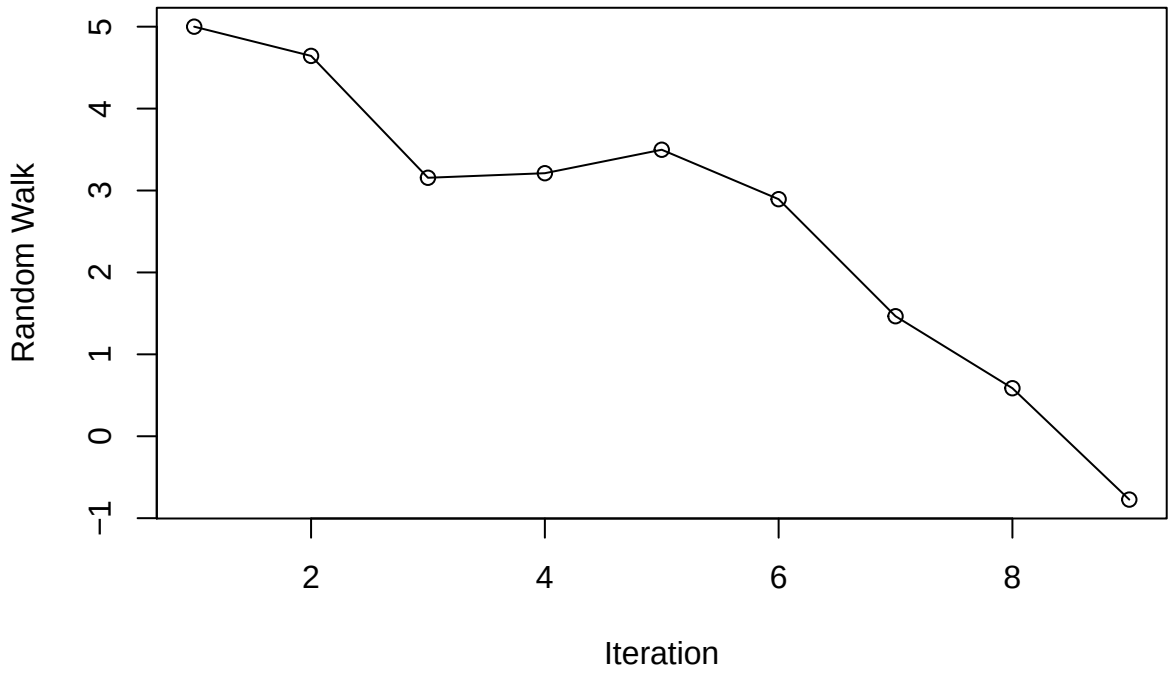
```
  if (plot.walk) {plot(1:length(x.vals), x.vals, type = "o", xlab = "Iteration", ylab = "Random Walk")}
  return(list(x.vals = x.vals, num.steps = i))
}

random.walk()
```




```
## $x.vals
##  [1]  5.0000000  5.8957313  6.3642005  5.3521735  4.1843288  4.0705807
##  [7]  4.2307974  2.9015960  1.7401569  2.2854255  1.7696370  2.5478363
## [13]  0.6243468  1.4784888 -0.0461679
##
## $num.steps
## [1] 15
```

```
random.walk()
```




```
## $x.vals
## [1]  5.0000000  4.6434571  3.1554243  3.2112247  3.4975483  2.8940356
## [7]  1.4652718  0.5863835 -0.7730589
##
## $num.steps
## [1] 9
```

```
random.walk(seed = 33, plot.walk = FALSE)
```

```
## $x.vals
##  [1]  5.0000000  4.3378214  3.5217724  2.9729590  3.7295869  4.2612312
##  [7]  3.8132800  3.1246550  2.1542497  0.2008006 -1.4452259
##
## $num.steps
## [1] 11
```

```
random.walk(seed = 33, plot.walk = FALSE)
```

```
## $x.vals
##  [1]  5.0000000  4.3378214  3.5217724  2.9729590  3.7295869  4.2612312
##  [7]  3.8132800  3.1246550  2.1542497  0.2008006 -1.4452259
##
## $num.steps
## [1] 11
```

Please submit the knitted .pdf file!