# hw6

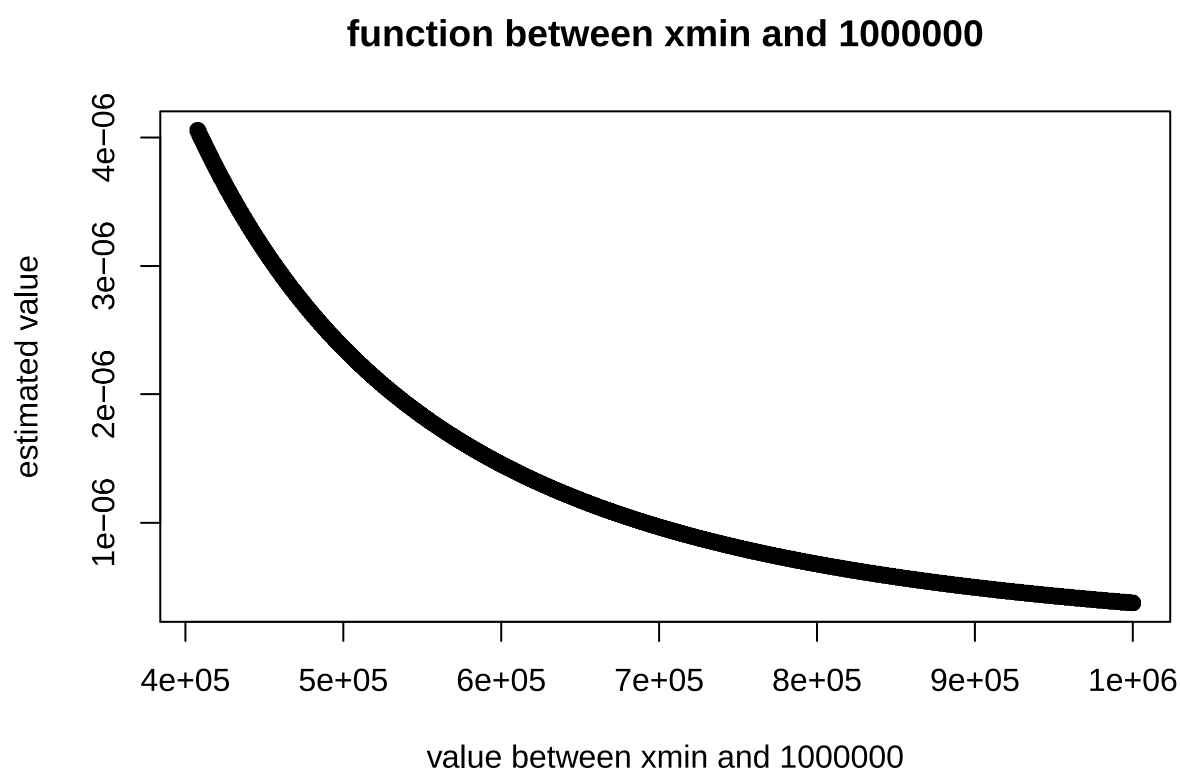
*Autumn Li UNI:ql2280*

*11/7/2016*

## Part 1: Inverse Transform Method

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.

```
report <- read.csv("~/Desktop/report.csv", header=FALSE)
report = report[-1,]
x.min = 407760
a.hat = 2.654
x = c(407760:1000000)
f = function (x,x.min = x.min, a.hat = a.hat){
  result = ((a.hat - 1)/x.min)*((x/x.min)^-a.hat)
  return(result)
  }
ans = f(x,x.min,a.hat)
plot(x,ans,main = "function between xmin and 1000000",ylab = "estimated value",xlab = "value between xm
```




2. For x > xmin, the cdf equals

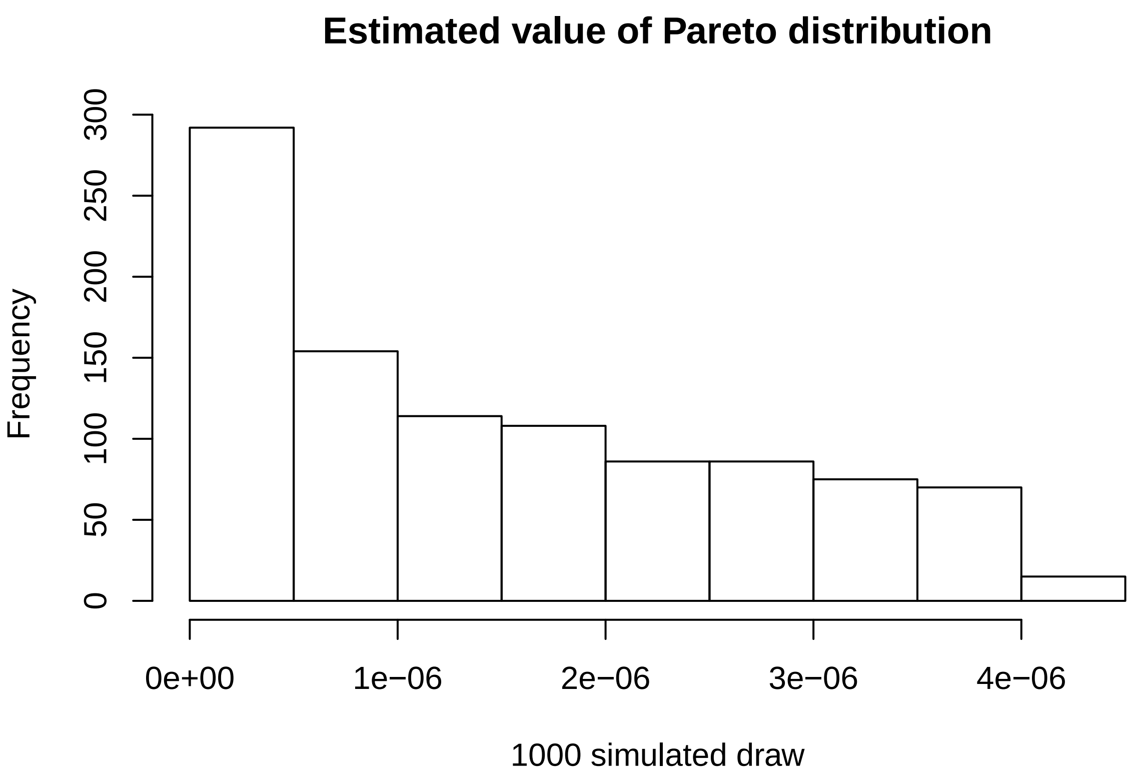
```
u = 0.5
upper.income = function (u = u,x.min = 407760,a = 2.654 ){
  result = (1-u)^(1/(1-a))*x.min
```

```
  return(result)
  }
upper.income(0.5)
```

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

3. Using the Inverse Transform Method

```
x = runif(1000)
result.1 = upper.income(x)
ans.1 = f(result.1,x.min,a.hat)
hist(ans.1,main = "Estimated value of Pareto distribution", xlab = "1000 simulated draw")
```




4. Using your simulated set, estimate the median income for the richest 1% of the world.

```
result.1 = upper.income(x)
result.2 = median(result.1);result.2
```

```
## [1] 644500.6
```

```
upper.income(0.5)
```

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

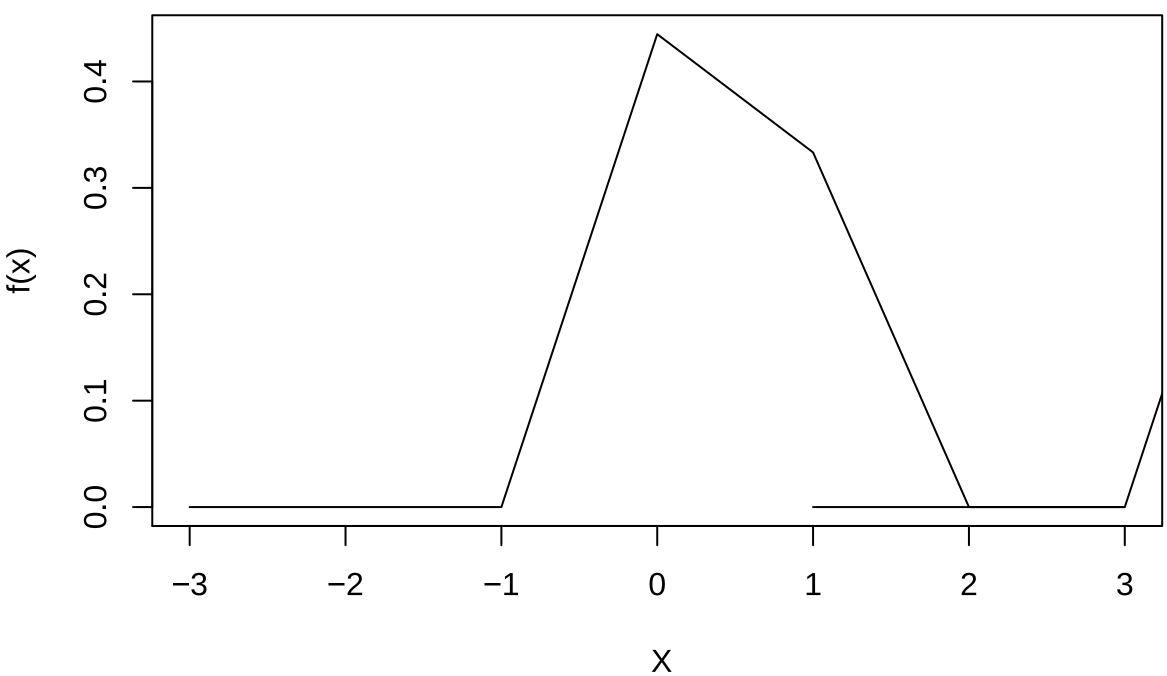
## Part 2: Reject-Accept Method

5. Write a function f that takes as input a vector x and returns a vector of f(x) values.

```r
x = c(-3:3)
f <- function(x) {
  return(ifelse((x <= -1 | x >= 3), 0, 1/9*(4-x^2)))
  }
plot(x,f(x),xlimylab = "Estimated value",xlab =" X",type = "l")
```

```
## Warning in plot.window(...): "xlimylab" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "xlimylab" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "xlimylab" is
## not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "xlimylab" is
## not a graphical parameter

## Warning in box(...): "xlimylab" is not a graphical parameter

## Warning in title(...): "xlimylab" is not a graphical parameter
```

```r
lines(f(x))
```




6. Determine the maximum of f(x) and find an envelope function e(x) by using a uniform density for g(x).

```r
xmax = 0
f.max = 4/9
e <- function(x) {
  return(ifelse((x <= -1 | x > 3), Inf, f.max)) }
e(x)
```

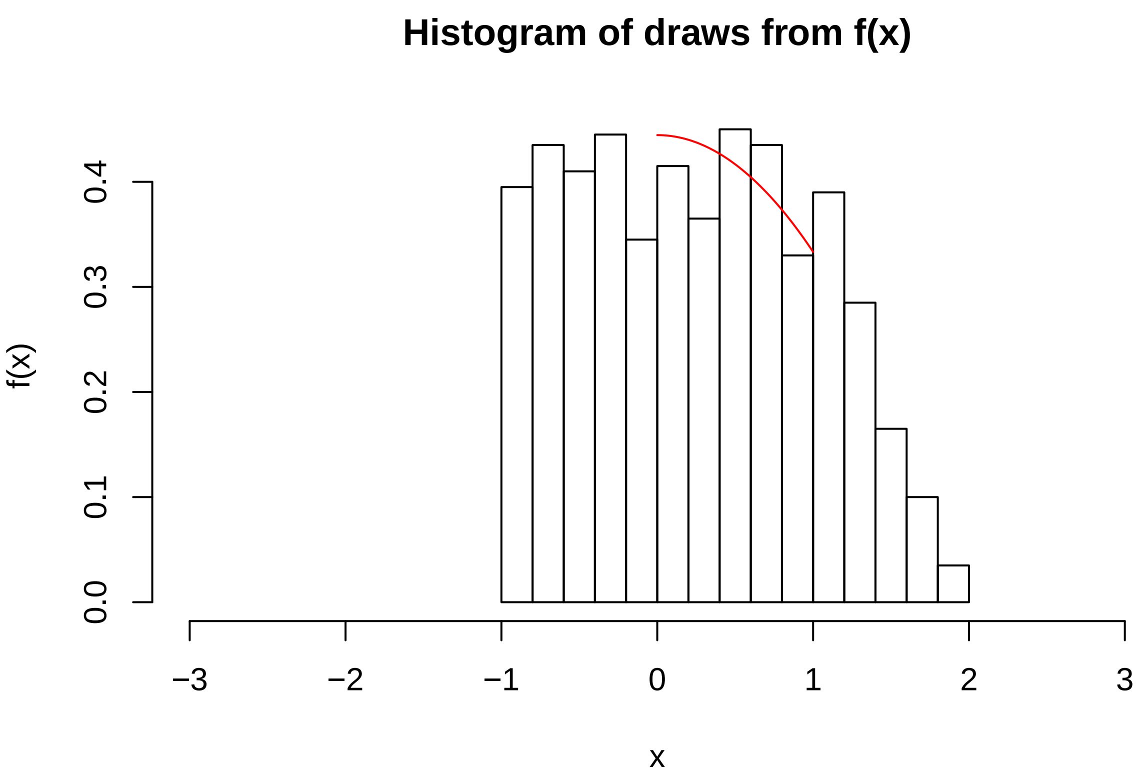
```
## [1]       Inf       Inf       Inf 0.4444444 0.4444444 0.4444444 0.4444444
```

7. Using the Accept-Reject Algorithm

```
n.samps <- 1000
n <- 0
samps <- numeric(n.samps)
while (n < n.samps) {
  y <- runif(1,-3,3)
  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(0, 1, length = 100)
hist(samps, prob = T, xlim=c(-3,3),ylab = "f(x)", xlab = "x",main = "Histogram of draws from f(x)")
lines(x, f(x),col = "red")
```




#Part 3: Simulation with Built-in R Functions 9. Write a while() loop to implement this procedure.

```
x.start = 5
n.1 = 1
x.vals =NULL
while(x.start > 0) {
```

```
  r = runif(1,min = -2, max = 1)
  x.start= x.start + r
  x.vals[n.1] = x.start
  n.1 = n.1 +1
  }
x.vals
```

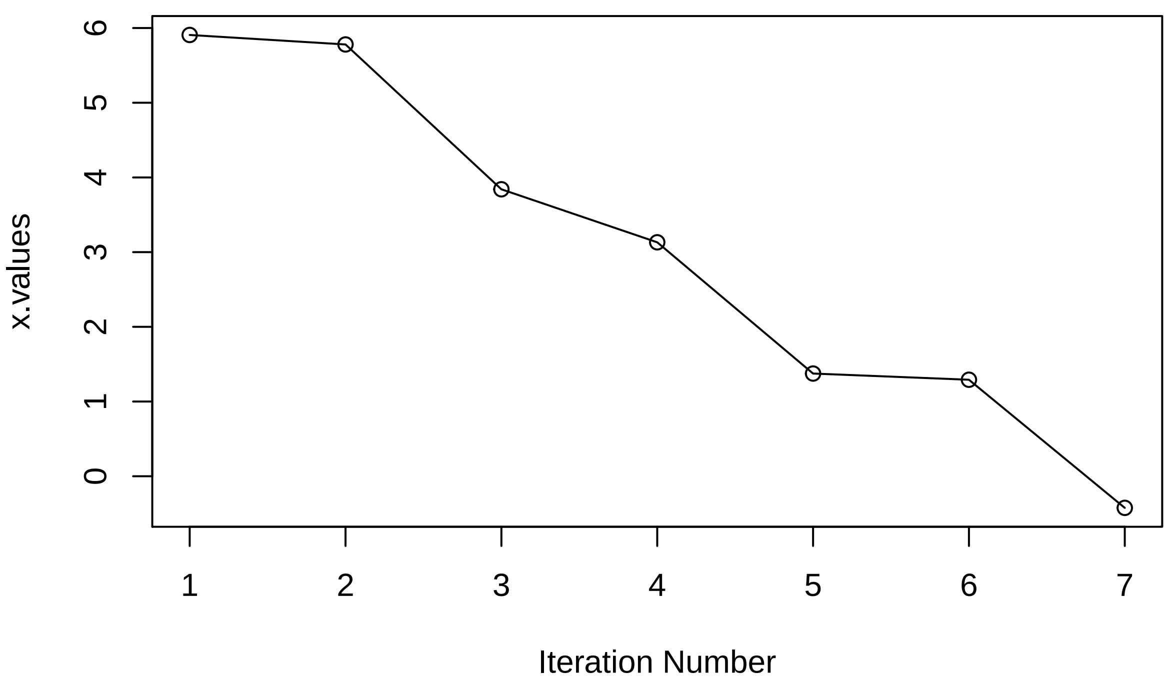
```
## [1]  5.9072239  5.7803463  3.8415435  3.1319049  1.3747008  1.2916489
## [7] -0.4234373
```

```
n.1
```

```
## [1] 8
```

10. Produce a plot of the random walk values x.vals from above versus the iteration number.

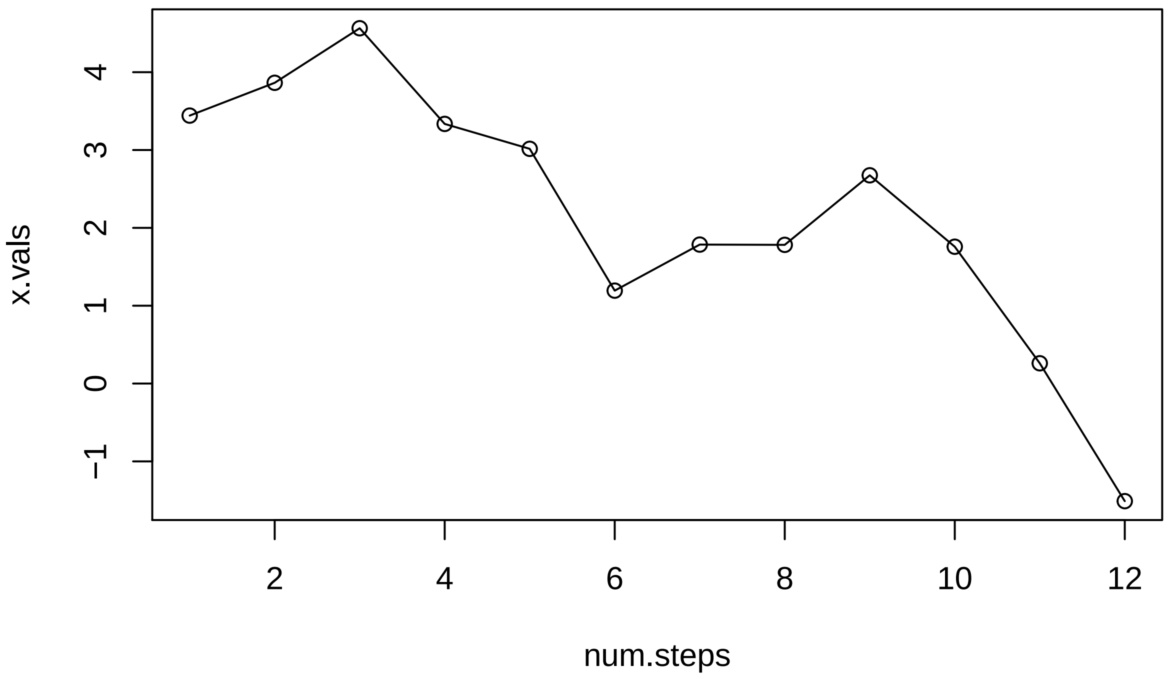
```
plot(c(1:(n.1-1)),x.vals,xlab = "Iteration Number",ylab = "x.values")
lines(x.vals)
```




11. Write a function random.walk() to perform the random walk procedure that you implemented in question (9)

```
random.walk = function (x.start = 5, plot.walk = TRUE){
  x.vals =NULL
  n.1 = 1
  while(x.start > 0) {
    r = runif(1,min = -2, max = 1)
    x.start= x.start + r
    x.vals[n.1] = x.start
    n.1 = n.1 +1
    }
  num.steps = c(1:(n.1-1))
```

```r
  result = list(x.vals,num.steps)
  if(plot.walk ==TRUE){
    plot(num.steps,x.vals)
    lines(x.vals)
    }
  return(result)
  }
random.walk(x.start = 5, plot.walk = TRUE)
```




```
## [[1]]
##  [1]  3.4423278  3.8636747  4.5646497  3.3355148  3.0154850  1.1937006
##  [7]  1.7851374  1.7817050  2.6753371  1.7578124  0.2602989 -1.5108099
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

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

```
## [[1]]
##  [1]  8.6806724  8.9312857  8.5911580  8.4479113  8.1246535  8.0421754
##  [7]  7.6360227  8.4393841  6.5523918  6.5771548  5.6786985  5.5627030
## [13]  4.3040302  4.9494297  5.4131939  5.2401056  3.4520828  3.8146443
## [19]  4.7086890  4.3986092  4.4730617  4.4761404  2.8266542  3.5003436
## [25]  2.8051520  2.2061924  2.5258602  0.8584726  1.1244277 -0.2754550
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30
```

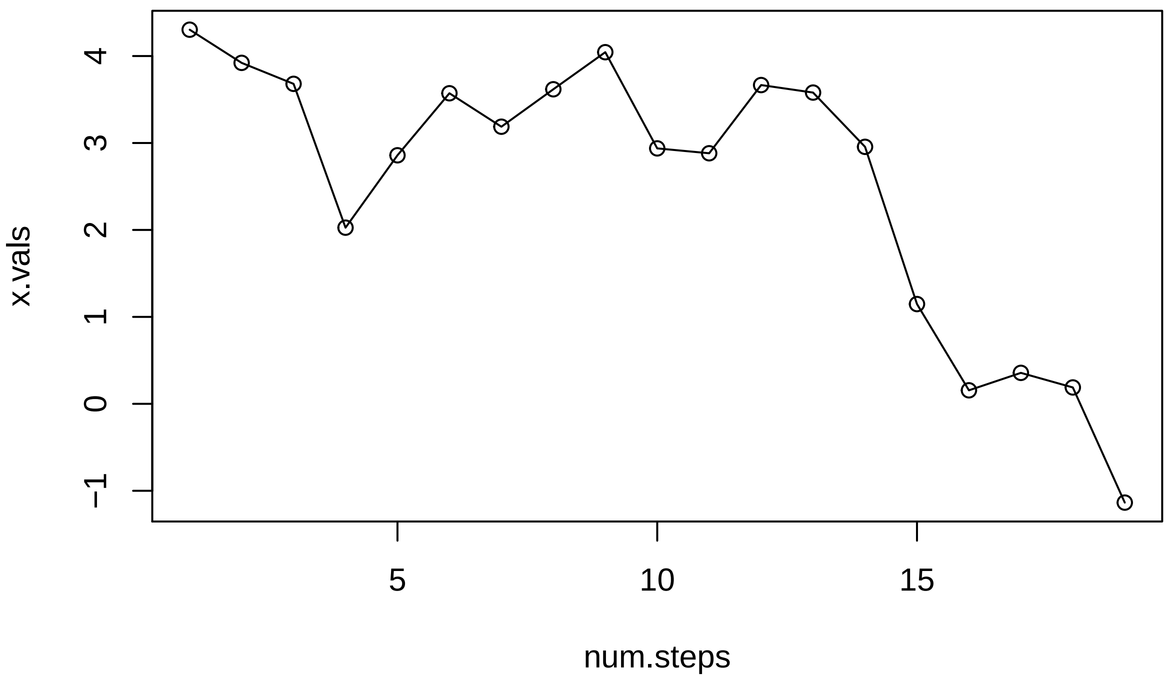
12. We'd like to answer the following question using simulation

```
rep = 10000
num = NULL
for (i in 1:rep){
  num[i] = max(unlist(random.walk(x.start = 5, plot.walk = FALSE)[2]))
}
avg_rep = mean(num);avg_rep
```

```
## [1] 11.3061
```

13. Modify your function random.walk() defined previously so that it takes an additional argument seed

```
set.seed(NULL)
random.walk(x.start = 5, plot.walk = TRUE)
```



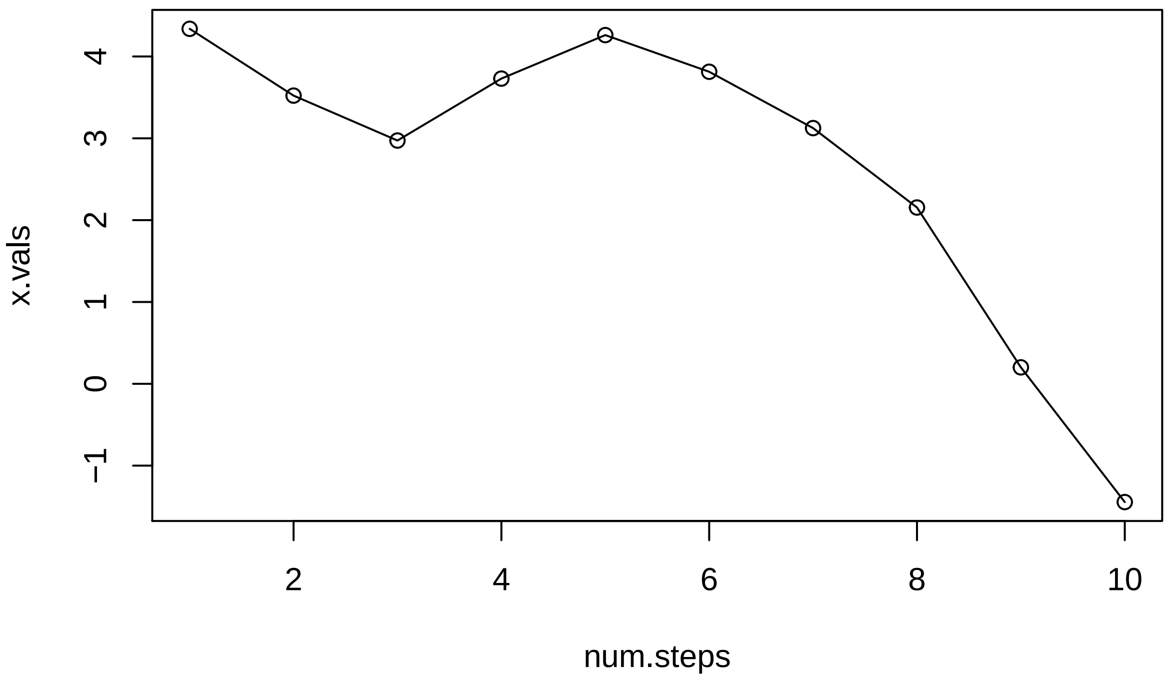

```
## [[1]]
##  [1]  4.3030224  3.9217208  3.6803076  2.0261046  2.8584561  3.5720286
##  [7]  3.1873618  3.6180963  4.0442244  2.9379038  2.8818989  3.6663330
## [13]  3.5799386  2.9559051  1.1475252  0.1554408  0.3562864  0.1884755
## [19] -1.1358723
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
```

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

```
## [[1]]
##  [1]  9.0598999  8.4068770  8.5308302  9.1875312  9.8668077  9.2463965
##  [7]  7.5590742  6.1966720  5.3419754  6.1287745  5.3527507  5.9768351
```

```
## [13]  6.8499774  5.8674075  3.9111589  3.5283445  3.5366305  2.4400718
## [19]  1.0600053  0.3034476 -0.9109537
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
```

```
set.seed(33)
random.walk(x.start = 5, plot.walk = TRUE)
```




```
## [[1]]
##  [1]  4.3378214  3.5217724  2.9729590  3.7295869  4.2612312  3.8132800
##  [7]  3.1246550  2.1542497  0.2008006 -1.4452259
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
set.seed(33)
random.walk(x.start = 10, plot.walk = FALSE)
```

```
## [[1]]
##  [1]  9.3378214  8.5217724  7.9729590  8.7295869  9.2612312  8.8132800
##  [7]  8.1246550  7.1542497  5.2008006  3.5547741  3.6277318  2.4091888
## [13]  1.0843424  0.1115011  0.4571649  0.9869050  1.3111516  0.4727306
## [19] -1.1199742
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
```