Hwk6_bs2996

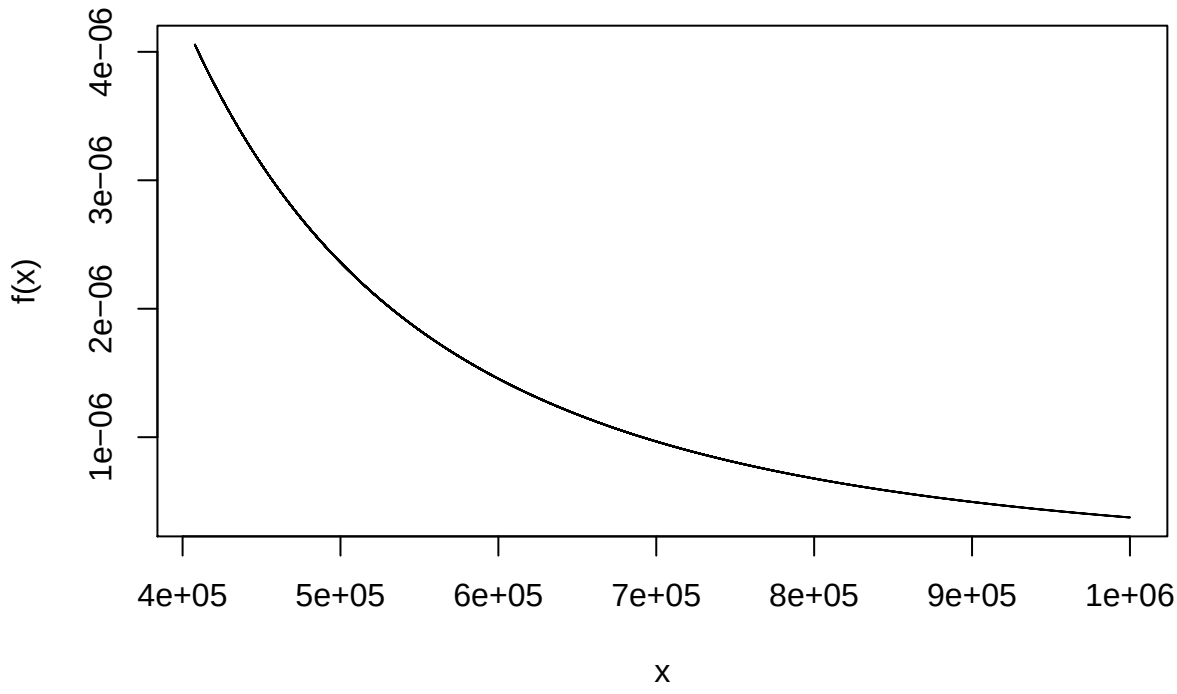
Bangda Sun

November 5, 2016

Part 1: Inverse Transform Method

1.

```
f <- function(x, a = 2.654, x.min = 407760){
  # p.d.f of Pareto distribution
  f <- (a - 1)/x.min * (x / x.min)^(-a)
  # the support of x is [xmin, infty)
  return(ifelse((x < x.min), 0, f))
}
x <- 407760:1000000
plot(x, f(x), xlab = "x", ylab = "f(x)", type = 'l')
```



2.

$$u = 1 - \left(\frac{x}{x_{min}}\right)^{-a+1} \Rightarrow x^{-a+1} = x_{min}^{-a+1}[1-u] \Rightarrow F^{-1}(u) = x_{min}[1-u]^{\frac{1}{-a+1}}.$$

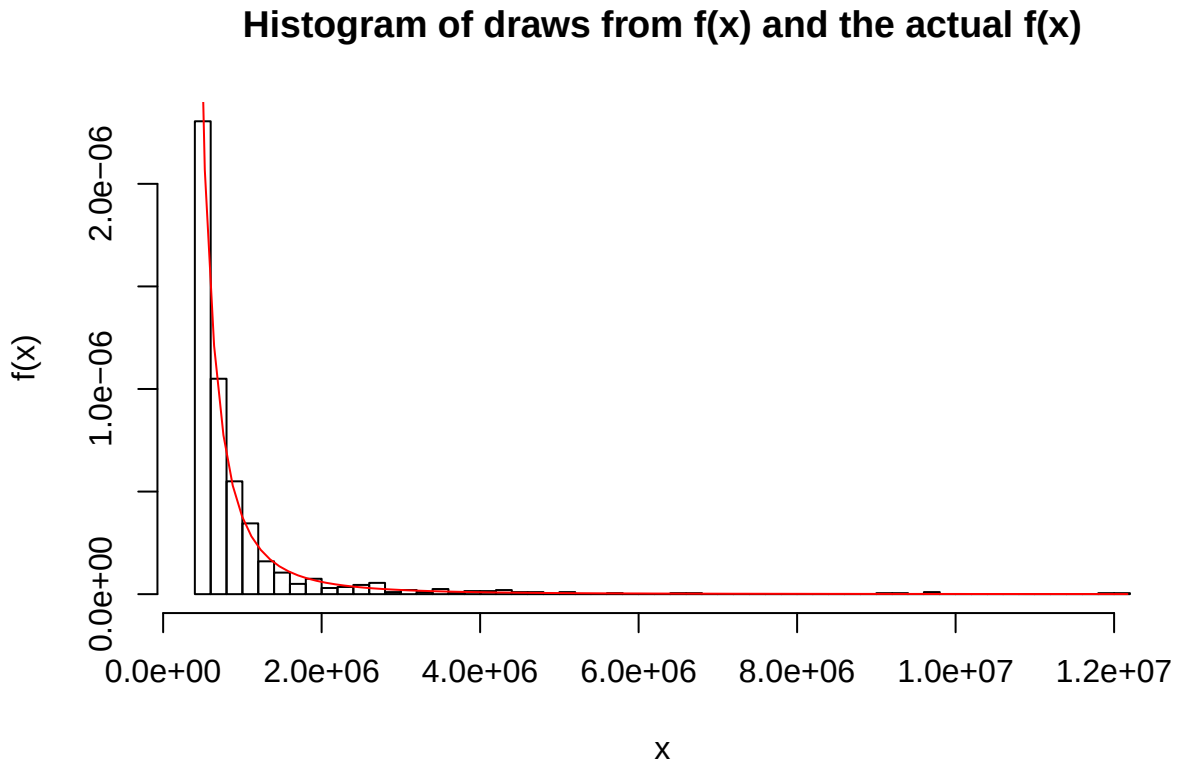
```r
upper.income <- function(u, a = 2.654, x.min = 407760){
  # calculate and return F inverse
  Finv <- x.min * (1 - u)^(1/(-a+1))
  # the support of u is (0, 1)
  return(ifelse((u < 0|u > 1), 0, Finv))
}
upper.income(.5)
```

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

3.

Use Inverse Transform Method,

```r
u <- runif(1000)
X <- upper.income(u)
hist(X, breaks = 50, probability = TRUE, xlab = "x", ylab = "f(x)",
     main = "Histogram of draws from f(x) and the actual f(x)")
curve(f, min(X), max(X), add = TRUE, col = "red")
```



4.

```r
# median of the simulate set
median(X)
```

```
## [1] 627708.3
```

```r
# 50th percentile of the Pareto
upper.income(.5)
```

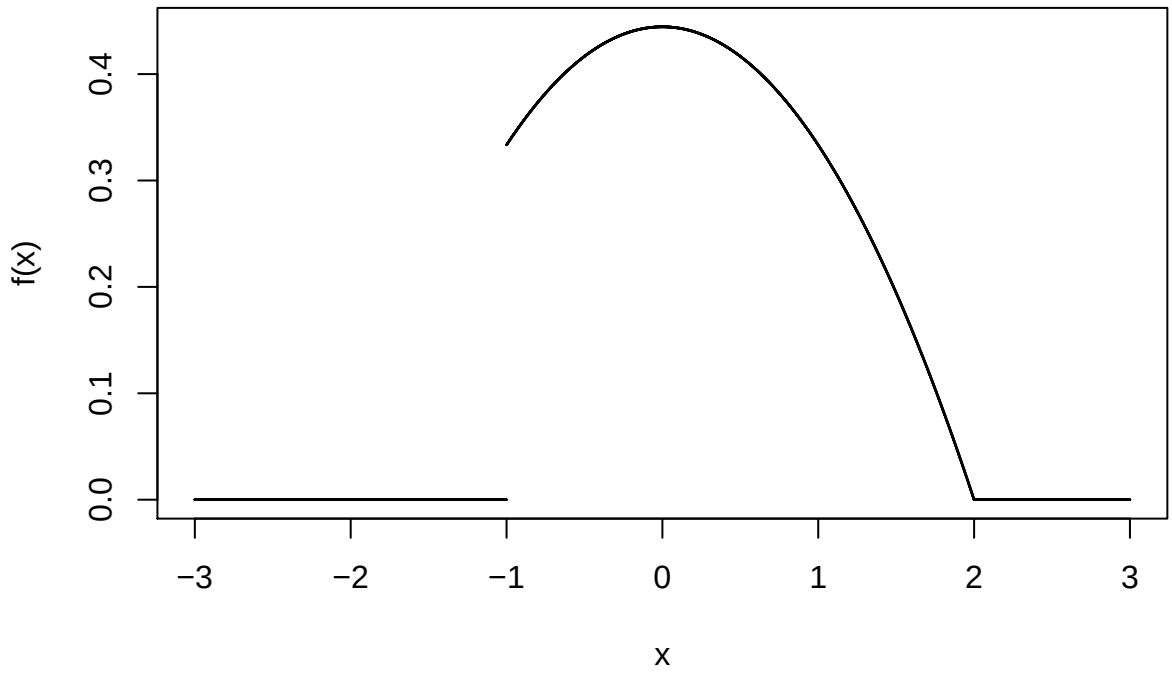
```
## [1] 620020.2
```

As we can see, the estimate 50th percentile is approximately 627708.3, and the actual 50th percentile of the Pareto distribution is approximately 620020.2. The estimate is a little bit larger than the actual value.

Part 2. Reject-Accept Method

5.

```r
f <- function(x){
    return(ifelse((x < -1|x > 2), 0, (4-x^2)/9))
}
x <- seq(-3, 3, length = 5000)
plot(x, f(x), xlab = "x", ylab = "f(x)", type = 'p', cex = .05)
```



6.

$f(x)$ is a quadratic function on [-1, 2],

$$\max f(x) = f(0) = \frac{4}{9}.$$

We can set $g \sim Unif[0, 1]$, and therefore

$$e(x) = \frac{1}{\alpha} = \frac{4}{9},\ 0 \leq x \leq 1.$$

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

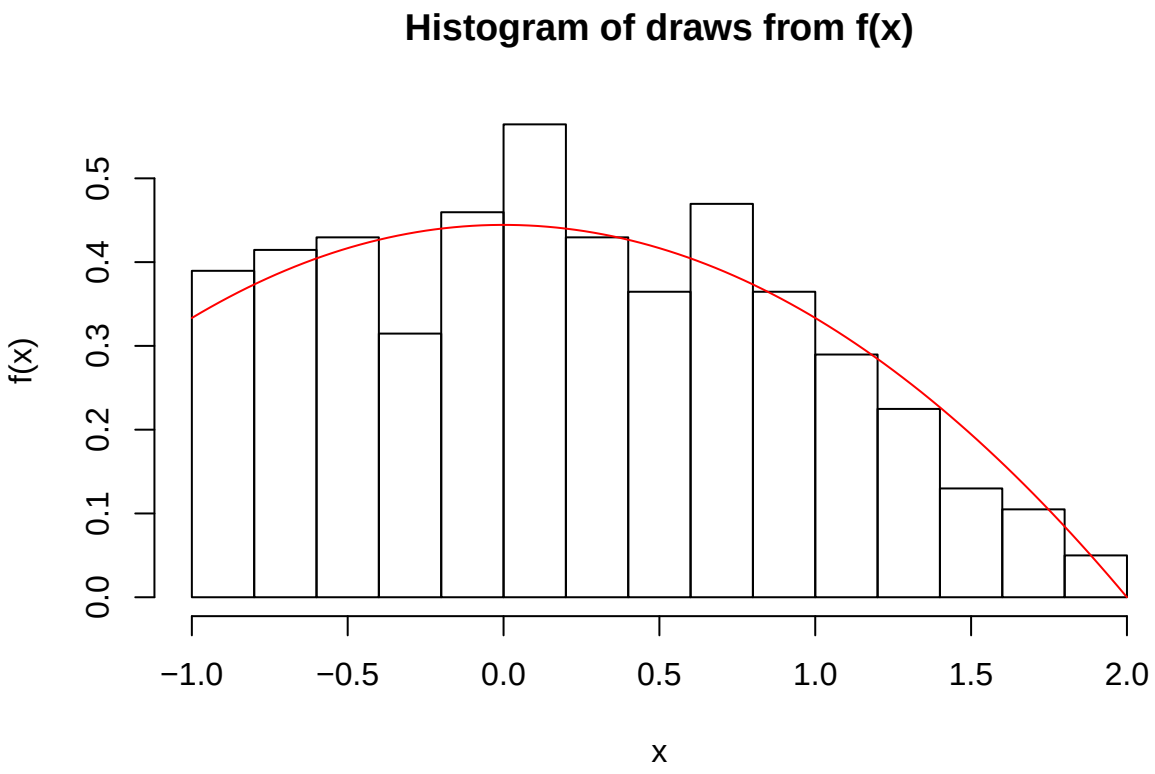
7.

```
# number of sampled needed
n        <- 1000
# count the accepted sample
n.accept <- 0
# the simulated sample
sample   <- numeric(n)
while (n.accept <= n){
  u <- runif(1)
  # sample from g(x)
  y <- runif(1, -1, 2)
  # accept condition
  if (u < f(y)/e(y)){
    n.accept <- n.accept + 1
    sample[n.accept] <- y
  }
}
head(sample)
```

```
## [1] -0.05817770  1.21946921 -0.66735830 -0.56797983 -0.74439124 -0.08669805
```

8.

```
hist(sample, probability = TRUE, xlab = "x", ylab = "f(x)",
     main = "Histogram of draws from f(x)")
curve(f, -1, 2, add = TRUE, col = "red")
```



Part 3. Simulation with Built-in R Functions

9.

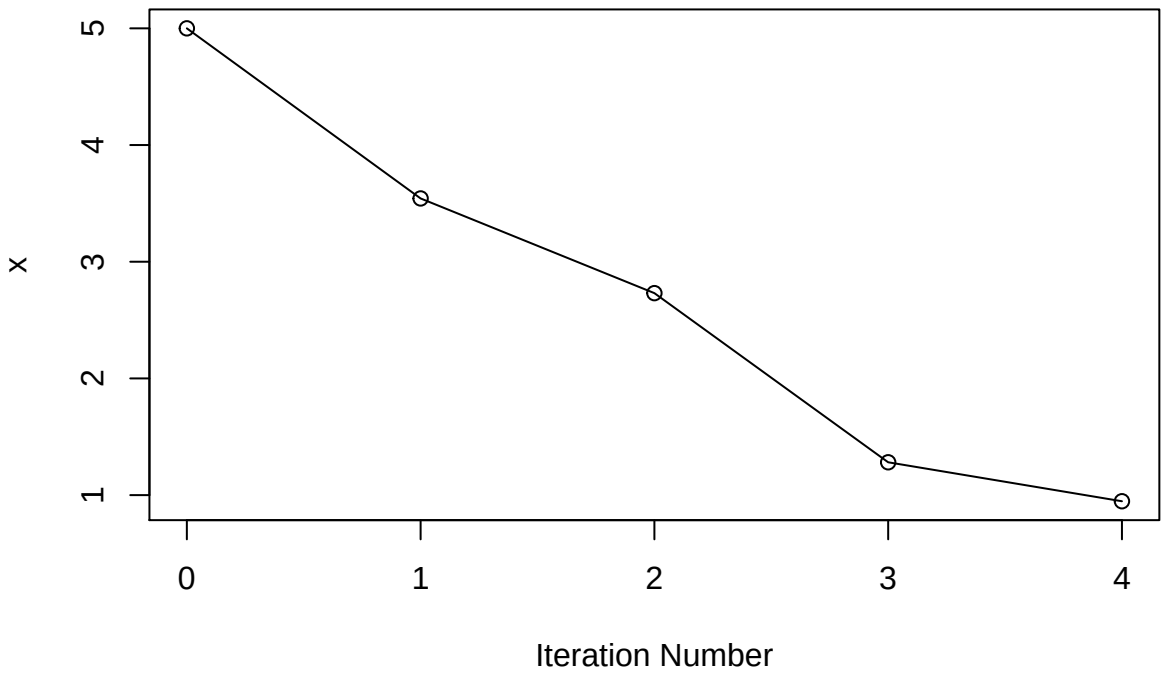
```
x <- 5
x.vals <- c()
while (x > 0){
  r <- runif(1, -2, 1)
  x.vals <- c(x.vals, x)
  x <- x + r
}
print(x.vals)
```

```
## [1] 5.0000000 3.5421849 2.7307045 1.2811342 0.9474109
```

10.

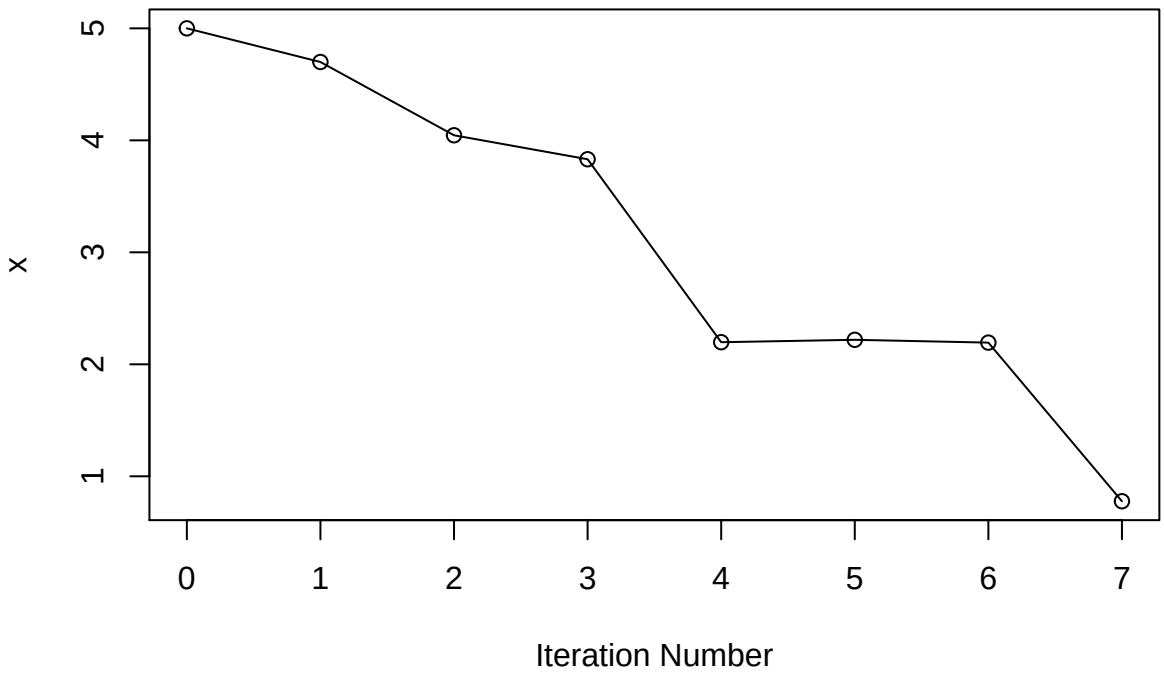
Here we included the start point, therefore $x = 5$ is the 0th iteration.

```
plot(0:(length(x.vals) - 1), x.vals, xlab = "Iteration Number", ylab = "x", type = "o")
```



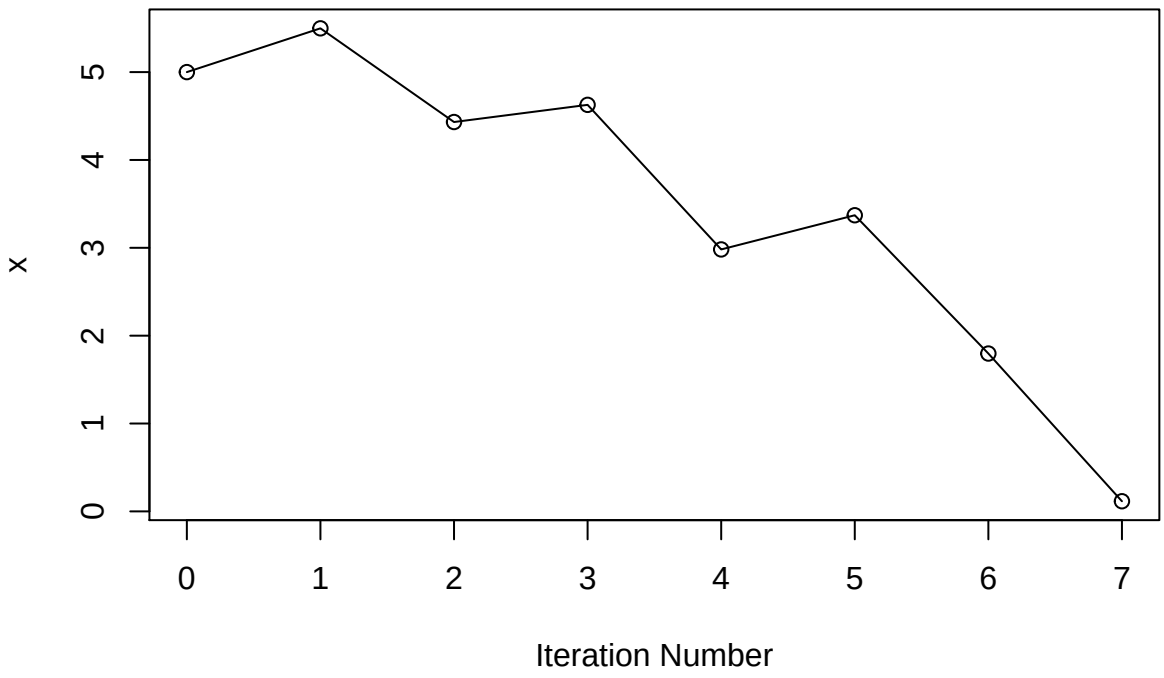
11.

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




```
## $x.vals
## [1] 5.0000000 4.6995741 4.0451947 3.8311013 2.1973332 2.2187913 2.1936725
## [8] 0.7771912
##
## $num.steps
## [1] 7
```

```
random.walk()
```




```
## $x.vals
## [1] 5.0000000 5.4980096 4.4318726 4.6273809 2.9813420 3.3705941 1.7964940
## [8] 0.1152974
##
## $num.steps
## [1] 7
```

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

```
## $x.vals
##  [1] 10.000000 10.024782  9.967698  8.100147  7.402354  6.586321  6.687999
##  [8]  6.756222  5.770178  5.400136  4.054377  4.354057  3.073540  3.421948
## [15]  2.380766  2.629459  1.826201  1.444158  1.853373  1.284794  1.501416
## [22]  1.446930  1.631903
##
## $num.steps
## [1] 22
```

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

```
## $x.vals
##  [1] 10.0000000  8.8317054  7.3750332  5.8452135  6.8231692  6.8756875
##  [7]  5.9267288  3.9310333  4.3002607  4.6729248  3.1203971  3.8239536
## [13]  3.3992134  3.4724583  4.3206961  5.1553933  3.6786894  2.0638491
```

```
## [19]  0.2805194
##
## $num.steps
## [1] 18
```

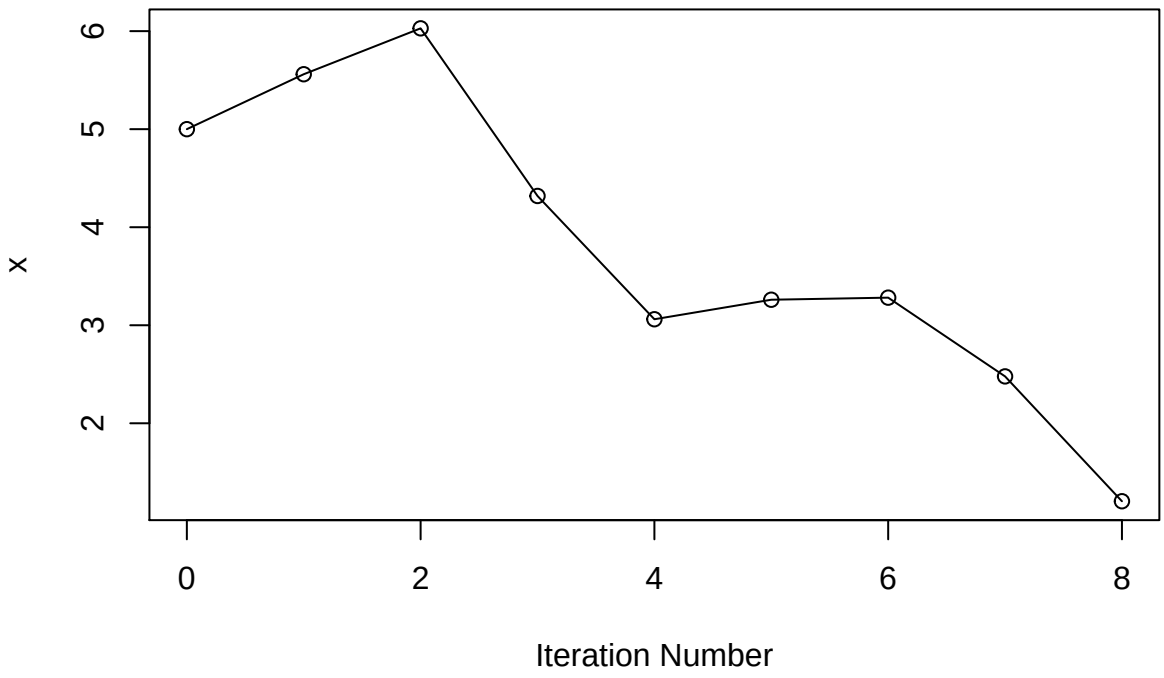
12.

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

```
## [1] 10.2507
```

13.

```
random.walk <- function(x.start = 5, seed = NULL, plot.walk = TRUE){
  set.seed(seed)
  x <- x.start
  x.vals <- c()
  while (x > 0){
    r <- runif(1, -2, 1)
    x.vals <- c(x.vals, x)
    x <- x + r
  }
  if (plot.walk == TRUE){
  plot(0:(length(x.vals) - 1), x.vals, xlab = "Iteration Number", ylab = "x",
            type = "o")
  }
  return(list(x.vals = x.vals, num.steps = length(x.vals) - 1))
}
random.walk()
```




```
## $x.vals
## [1] 5.000000 5.560485 6.028404 4.318719 3.061329 3.260207 3.281636 2.478341
## [9] 1.204875
##
## $num.steps
## [1] 8
```

```
random.walk()
```


```
## $x.vals
##  [1] 5.000000 5.862642 6.653126 6.434600 6.171287 5.145683 4.146858
##  [8] 4.951774 4.372457 3.273866 3.651723 2.601807 1.427249 1.081143
##
## $num.steps
## [1] 13
```

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

```
## $x.vals
##  [1] 5.0000000 4.3378214 3.5217724 2.9729590 3.7295869 4.2612312 3.8132800
##  [8] 3.1246550 2.1542497 0.2008006
##
## $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 3.8132800
##  [8] 3.1246550 2.1542497 0.2008006
##
## $num.steps
## [1] 9
```