hw6_yw3204

Yuhao Wang, yw3204

11/24/2018

Part 1: Inverse Transform Method

1.

The c.d.f of a standard Cauchy distribution is $F_X(x) = \frac{1}{\pi}arctan(x) + \frac{1}{2}$. And thus its inverse function is $y = tan(\pi * (x - \frac{1}{2}))$. According to the inverse transformation method, the transformation of U should be $tan(\pi * (U - \frac{1}{2}))$, which has a standard Cauchy distribution.

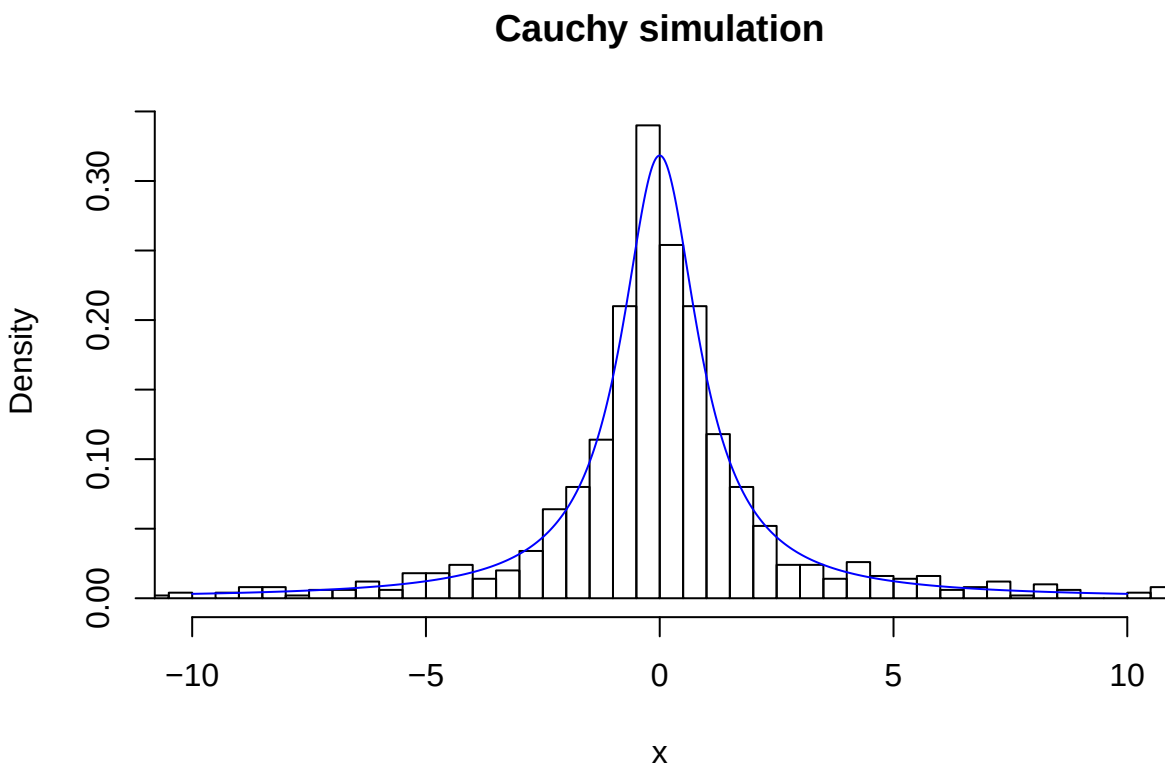
2.

```
cauchy.sim <- function(n) {
  u <- runif(n)
  cauchy <- tan(pi*(u-1/2))
  return(cauchy)
}

# test 10 draws
cauchy.sim(10)
```

```
##  [1]  2.9723830 -0.9070131 -0.4248394  0.2329576  3.3710605 -1.3611982
##  [7]  3.0255173  5.6954298  0.5530230  0.4294381
```

3.

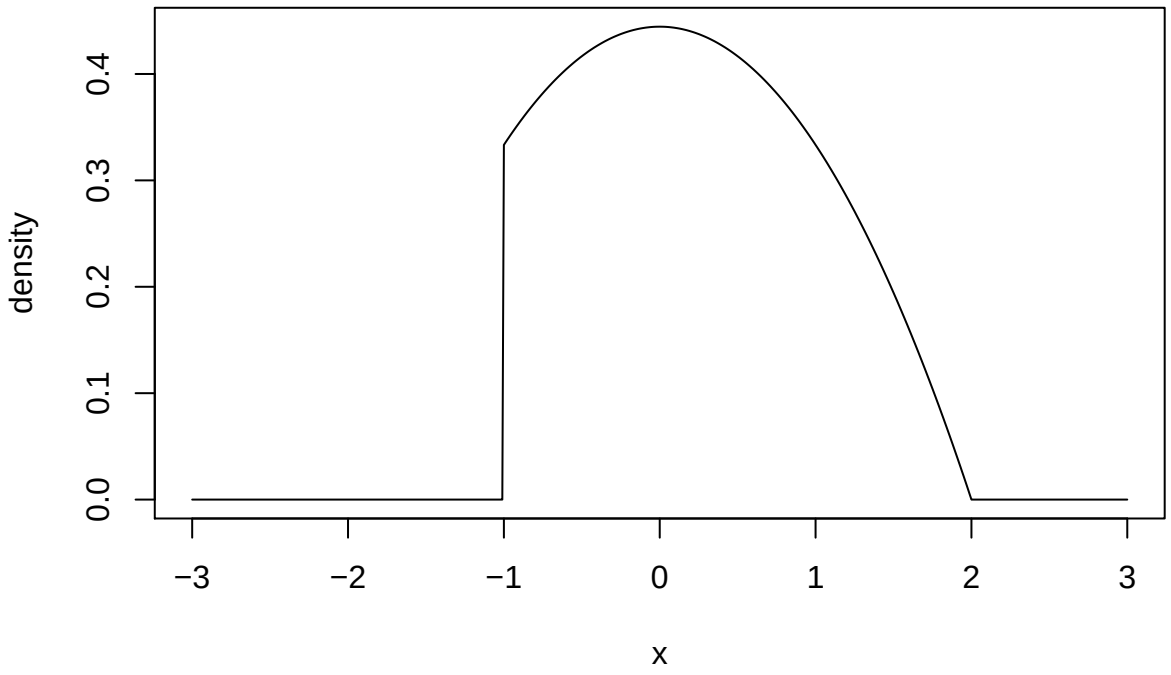
```
cauchy.draws <- cauchy.sim(1000)
hist(cauchy.draws, breaks = 10000, prob = T, xlim = c(-10, 10), xlab = "x", main = "Cauchy simulation")

# draw the true density function
x <- seq(-10, 10, .01)
lines(x, 1 / (pi*(1+x^2)), col = "blue")
```



Part 2: Reject-Accept Method

4.

```
# density
f <- function(vec) {
  # vectorization
  res <- ifelse(vec >= -1 & vec <= 2, 1/9*(4-vec*vec), 0)
  return(res)
}

x1 <- seq(-3, 3, 0.01)
plot(x1, f(x1), type = "l", xlab = "x", ylab = "density")
```



5.

The maximum of $f(x)$ is $\frac{4}{9}$.

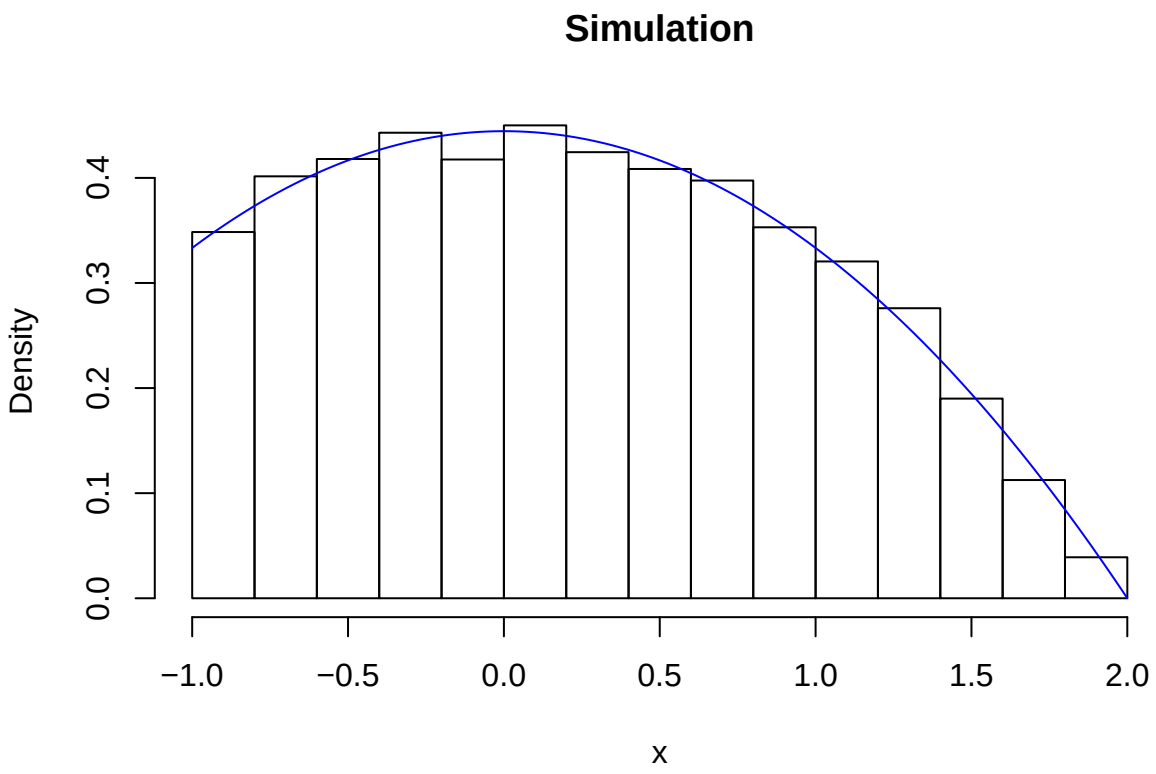
```
# envelope function
e <- function(vec) {
  res <- ifelse(vec >= -1 & vec <= 2, 4/9, 0)
  return(res)
}
```

6.

```
size <- 10000
f.draws <- c()

# rejection algorithm
while(length(f.draws) < size) {
  y <- runif(1, -1, 2)
  u <- runif(1)
  if(u < f(y)/e(y)) {
    f.draws <- c(f.draws, y)
  }
}
```

7.

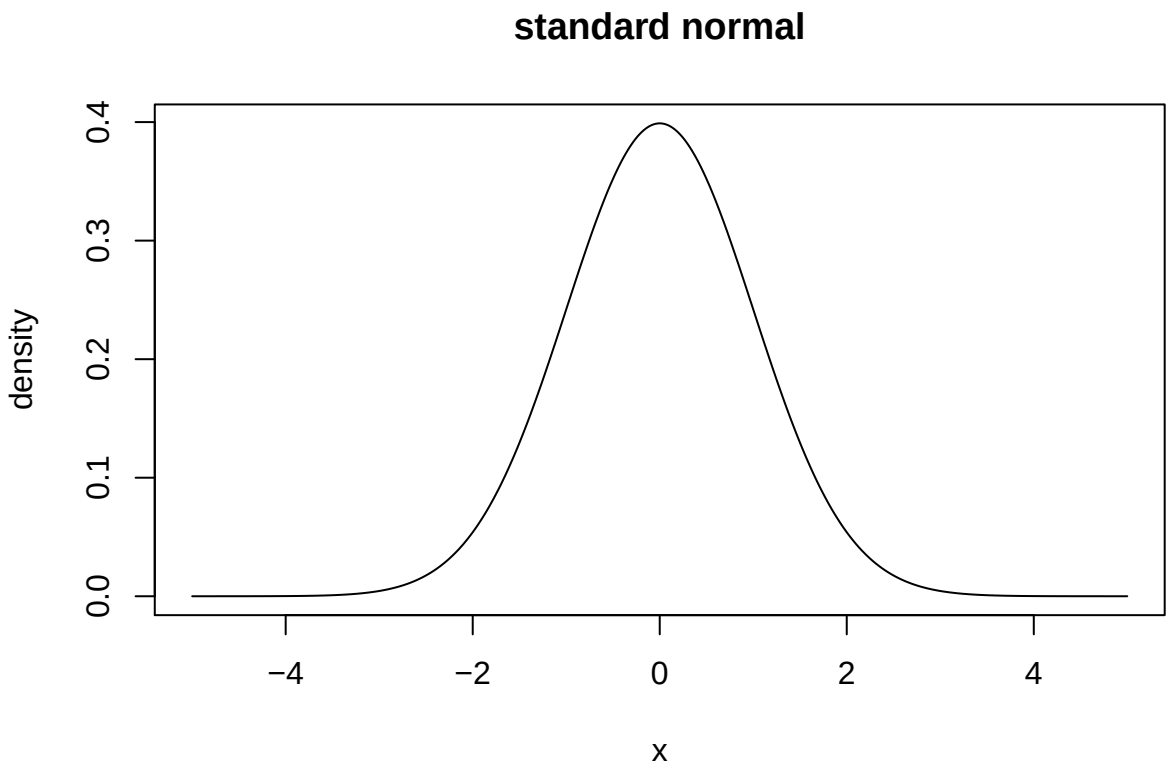
```
hist(f.draws, prob = T, xlab = "x", main = "Simulation")
x2 <- seq(-1, 2, 0.01)
lines(x2, 1/9 * (4-x2^2), col = "blue")
```



Problem 3: Reject-Accept Method Continued

8.

```r
# standard normal
f1 <- function(vec) {
  res <- 1/sqrt(2*pi) * exp(-1/2*vec^2)
  return(res)
}

x3 <- seq(-5, 5, 0.01)
plot(x3, f1(x3), type = "l", xlab = "x", ylab = "density", main = "standard normal")
```



9.

```r
# envelope
e1 <- function(x, alpha) {
  stopifnot(alpha > 0, alpha < 1)
  res <- 1/(pi * (1+x^2) * alpha)
  return(res)
}
```

10.

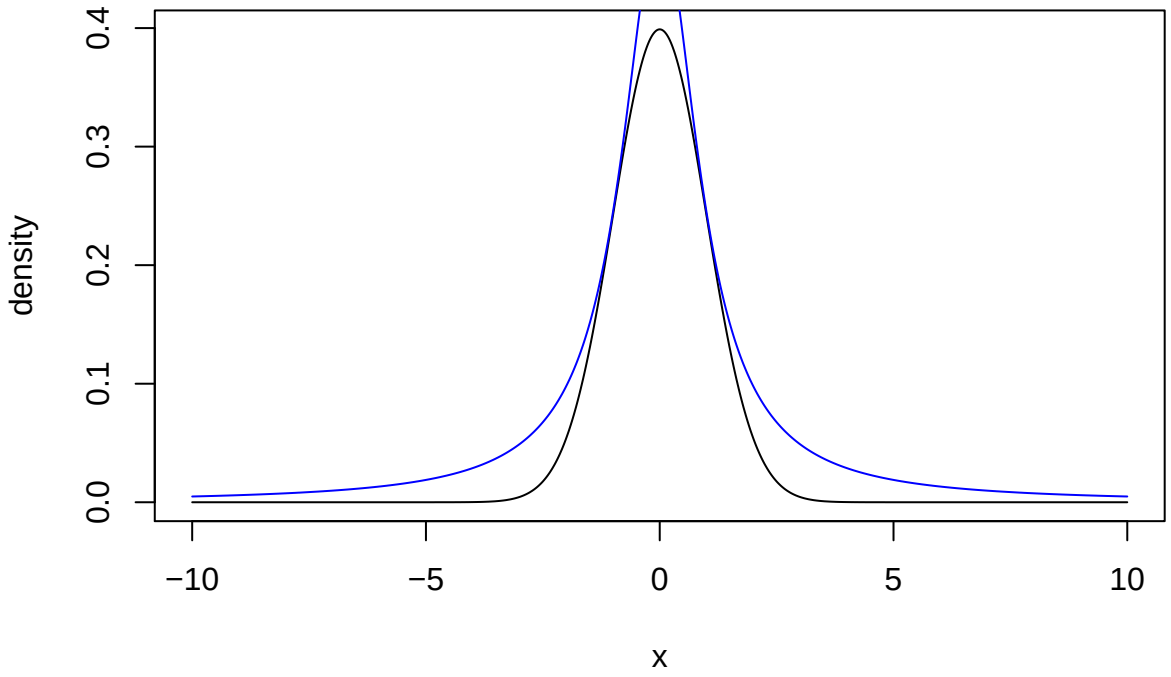
Clearly, we want to choose α that satisfies $g(x)/\alpha >= f(x), \forall x \in \mathbb{R}$. Specifically, it is equivalent to the following optimization problem:

$\alpha = min_{x\in\mathbb{R}} \frac{g(x)}{f(x)} = min_{x\geq 0} \frac{e^x}{1+2*x} * \sqrt{\frac{2}{\pi}}$

By taking the derivative of the target function, we have $(\frac{e^x}{1+2*x})' = \frac{e^x(2x-1)}{(1+2x)^2}$.

Apparently, the minimum is achieved at $x = 1/2$, which is 0.658. Therefore, we choose $\alpha = 0.65$.

```r
x4 <- seq(-10, 10, 0.01)
plot(x4, f1(x4), type = "l", xlab = "x", ylab = "density")
lines(x4, e1(x4, 0.65), col = "blue")
```



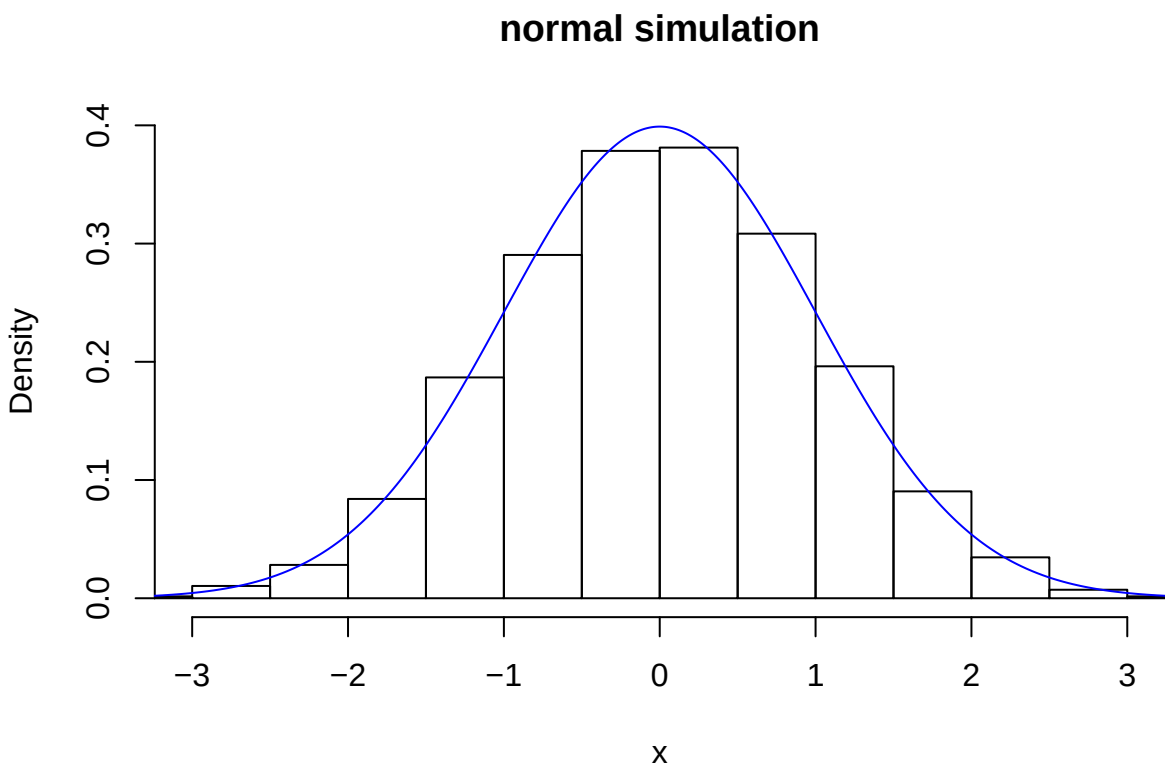
11.

```
# normal distribution simulation function
normal.sim <- function(n) {
  res <- c()

  while(length(res) < n) {
    y <- cauchy.sim(1)
    u <- runif(1)

    if(u < f1(y)/e1(y, alpha = 0.65)) {
      res <- c(res, y)
    }
  }

  return(res)
}

normal.sim(10)
```

```
##  [1] -1.2733231 -0.3452198  0.1276412  0.9416057 -0.6668286  0.2782496
##  [7] -0.8005083  0.0507360 -0.8621806 -0.4424169
```

12.

```
normal.draws <- normal.sim(10000)

hist(normal.draws, prob = T, xlim = c(-3, 3), xlab = "x", main = "normal simulation", ylim = c(0, 0.4))
x5 <- seq(-4, 4, 0.01)
lines(x5, 1/sqrt(2*pi) * exp(-1/2*x5^2), col = "blue")
```



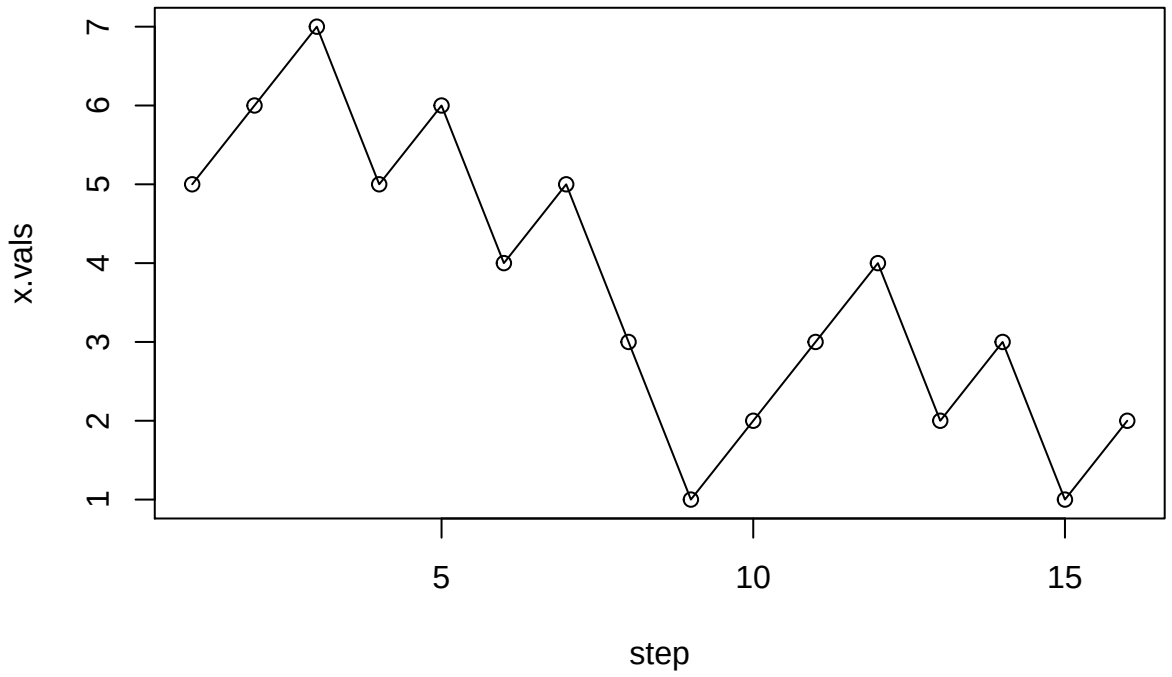
Part 3: Simulation with Built-in R Functions

13.

```
x <- 5
x.vals <- c(x)
# random walk
while(x > 0) {
  d <- sample(c(-2, 1), 1)
  x <- x+d
  if(x > 0) {
    x.vals <- c(x.vals, x)
  }
}
```

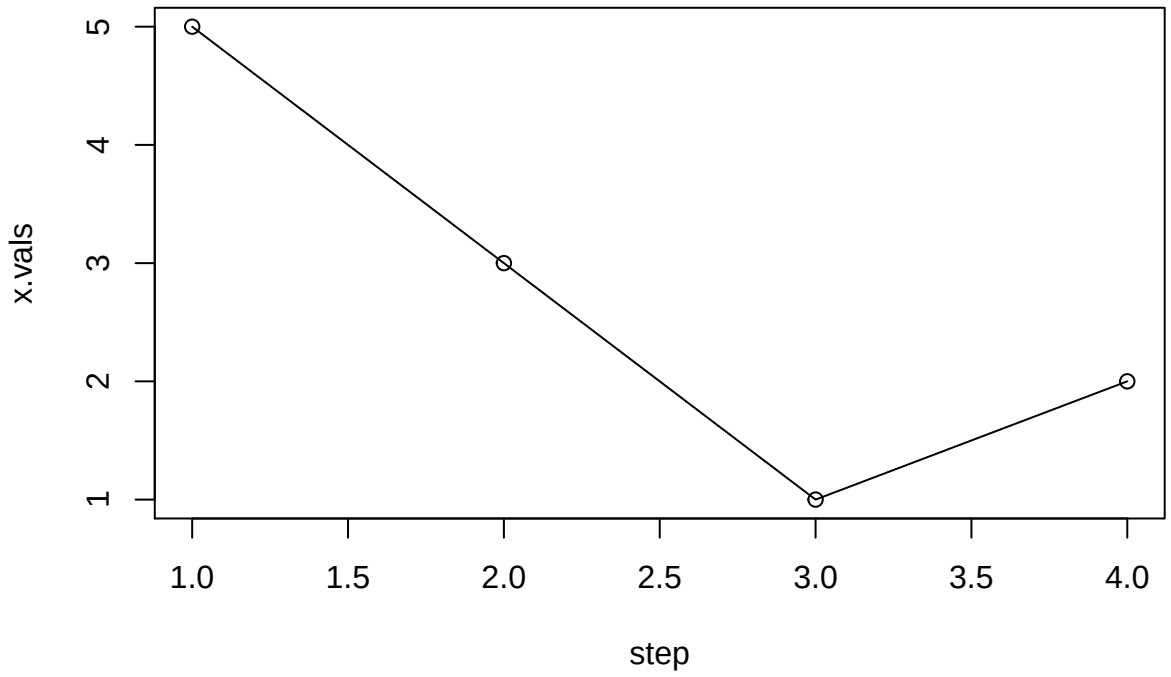
14.

```
plot(x.vals, type = "o", xlab = "step")
```



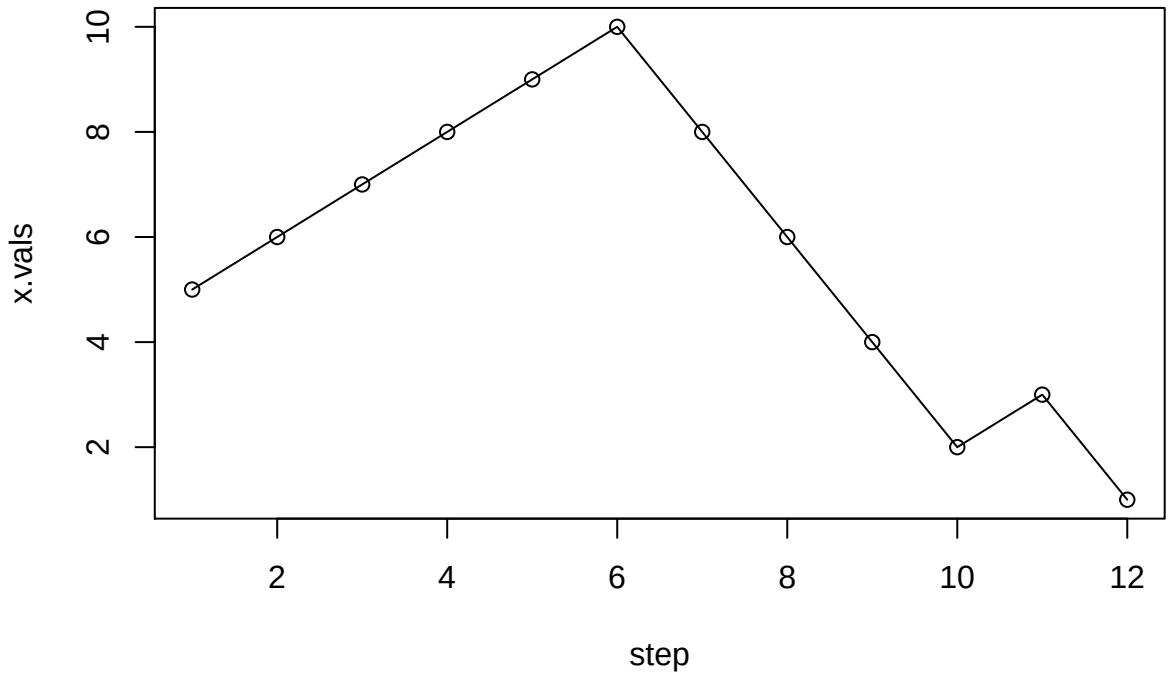
15.

```
# random walk function
random.walk <- function(x.start = 5, plot.walk = T) {
  num.steps <- 0
  x <- x.start
  x.vals <- c(x)

  while(x > 0) {
    d <- sample(c(-2, 1), 1)
    x <- x+d
    if(x > 0) {
      x.vals <- c(x.vals, x)
    }
    num.steps <- num.steps+1
  }

  if(plot.walk) {
    plot(x.vals, type = "o", xlab = "step")
  }

  return(list(steps = num.steps, vals = x.vals))
}

random.walk()
```




```
## $steps
## [1] 4
##
## $vals
## [1] 5 3 1 2
```

```
random.walk()
```




```
## $steps
## [1] 12
##
## $vals
```

```
##  [1]  5  6  7  8  9 10  8  6  4  2  3  1
```

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

```
## $steps
## [1] 47
##
## $vals
##  [1] 10 11  9  7  8  9 10  8  6  7  8  6  4  5  6  7  5  3  4  5  3  1  2
## [24]  3  4  5  3  4  5  6  7  5  6  4  2  3  4  5  6  7  5  3  1  2  3  1
## [47]  2
```

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

```
## $steps
## [1] 32
##
## $vals
##  [1] 10 11 12 13 14 12 10 11  9 10 11 12 13 11 12 13 11 12 13 14 15 13 14
## [24] 15 13 11 12 10  8  6  4  2
```

16.

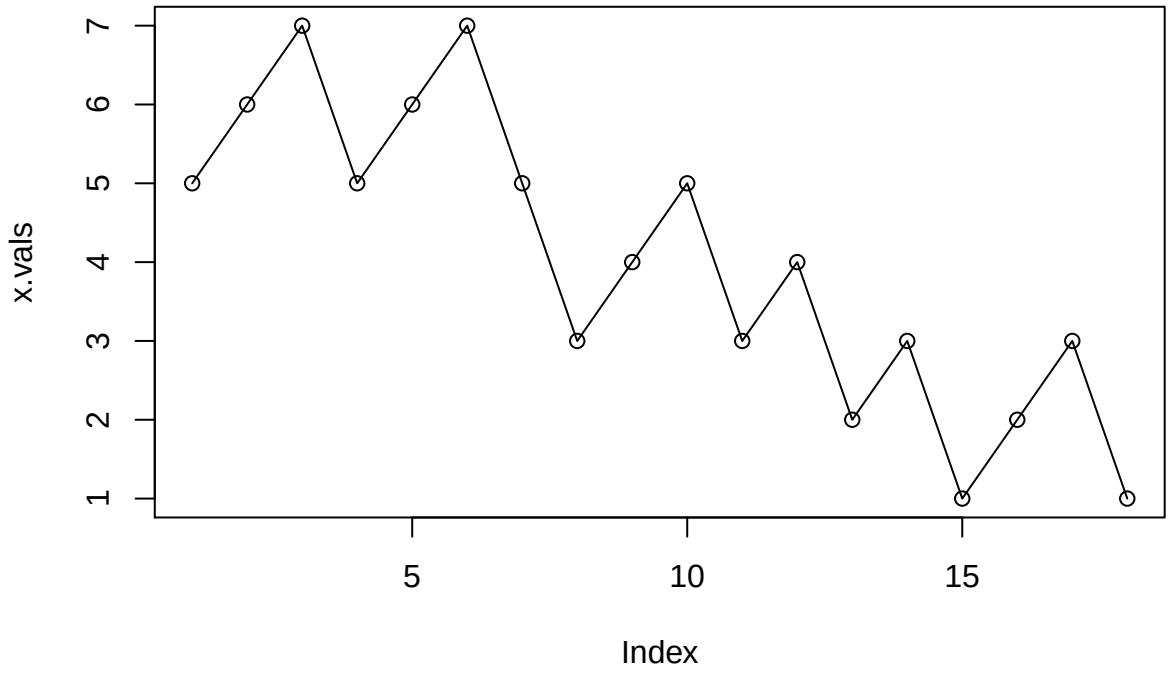
```
steps <- c()
for(i in c(1:10000)) {
  steps <- c(steps, random.walk(plot.walk = F)[[1]])
}

# average number of steps
mean(steps)
```

```
## [1] 10.803
```

17.

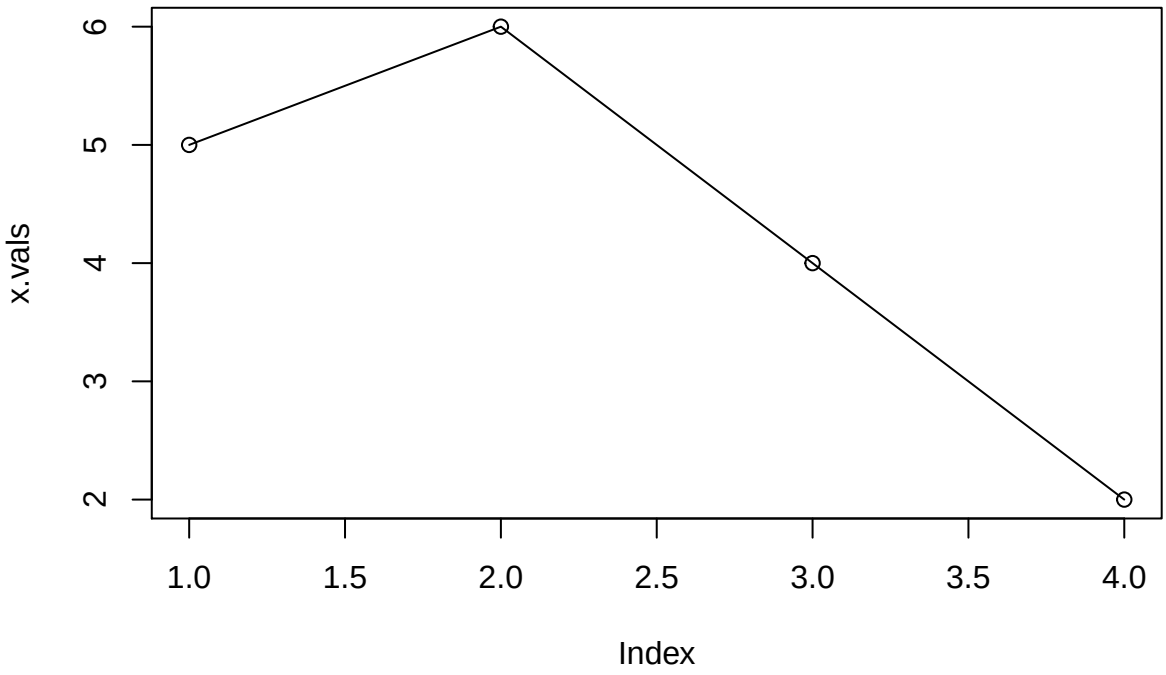
```
random.walk <- function(x.start = 5, plot.walk = T, seed = NULL) {
  if(!is.null(seed)) {
    set.seed(seed)
  }

  num.steps <- 0
  x <- x.start
  x.vals <- c(x)

  while(x > 0) {
    d <- sample(c(-2, 1), 1)
    x <- x+d
    if(x > 0) {
      x.vals <- c(x.vals, x)
    }
    num.steps <- num.steps+1
  }
```

```
  if(plot.walk) {
    plot(x.vals, type = "o")
  }

  return(list(steps = num.steps, vals = x.vals))
}

random.walk()
```




```
## $steps
## [1] 18
##
## $vals
##  [1] 5 6 7 5 6 7 5 3 4 5 3 4 2 3 1 2 3 1
```

```
random.walk()
```




```
## $steps
## [1] 4
##
## $vals
## [1] 5 6 4 2
```

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

```
## $steps
## [1] 3
##
## $vals
## [1] 5 3 1
```

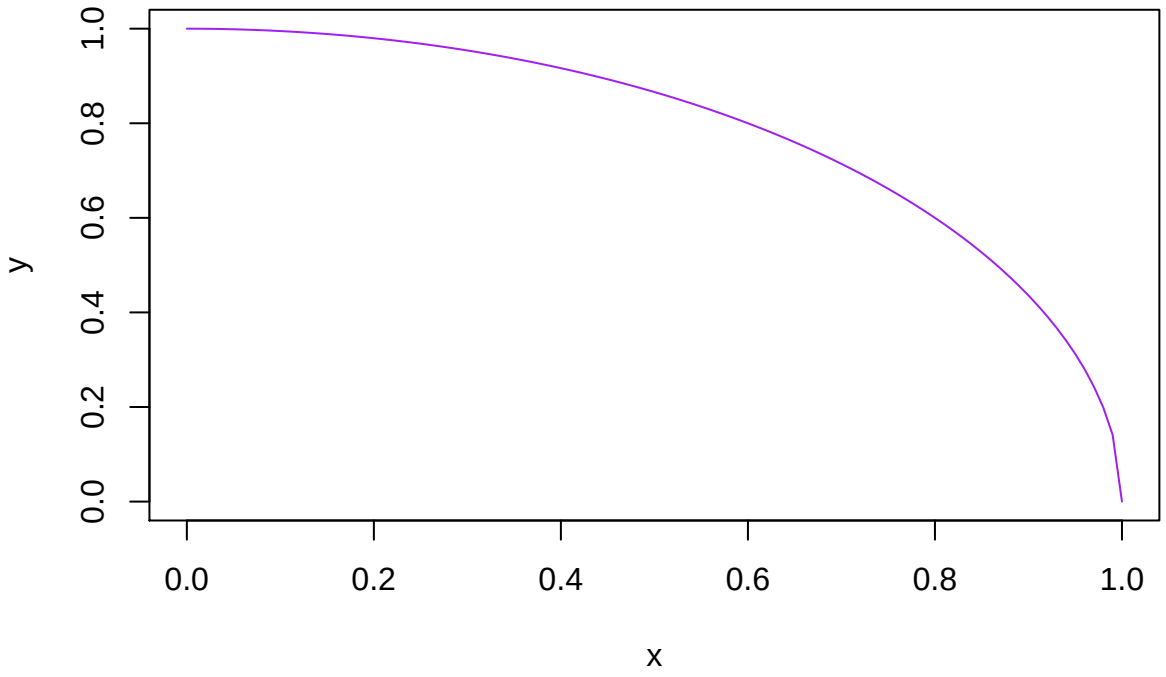
```
random.walk(x.start = 5, plot.walk = F, seed = 33)
```

```
## $steps
## [1] 3
##
## $vals
## [1] 5 3 1
```

Part 4: Monte Carlo Integration

18.

```
g <- function(x) {
  return(sqrt(1-x^2))
}

plot(seq(0,1,.01), g(seq(0,1,.01)), type="l", col="purple", xlab = "x", ylab = "y")
```



19.

$area = \frac{\pi}{4}$

20.

```
pi_hat <- 0
n1 <- 0
n <- 0

# MC
while(abs(pi - pi_hat) > 1/1000) {
  u1 <- runif(1)
  u2 <- runif(1)
  if(u1^2 + u2^2 < 1) {
    n1 <- n1+1
  }
  n <- n+1
  pi_hat <- 4*n1/n
}

pi_hat
```

```
## [1] 3.140625
```