STAT 202C - HW 3

John Rapp Farnes / 405461225

4/24/2020

1)

a)

My guess is that alternative 3 will converge faster than alternative 2, since the sampled points will be closer to those of alternative 1. With alternative 2, as the standard deviation is small, only a fraction of the points will reach into the area that is likely for alternative 1, the others will get low weights and therefore many samples are needed for an accurate answer.

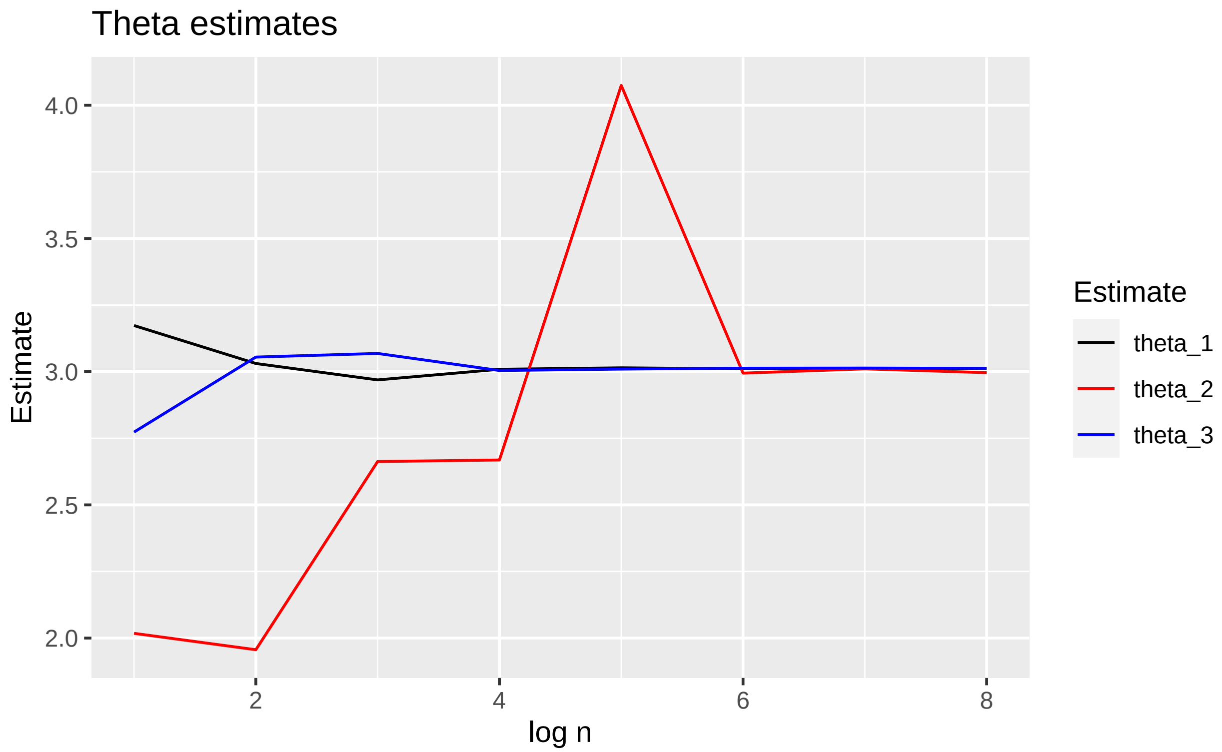
```
library(tidyverse)
library(reshape2)

set.seed(2020)
integral_func <- function(x, y) {
  sqrt(x^2+y^2)
}

sample_theta <- function(x, y, w) {
  sum(integral_func(x, y) * w) / sum(w)
}

pi_func <- function(x, y) {
  1/(2 * pi)*exp(-1/2*((x-2)^2+(y-2)^2))
}

g_func <- function(x, y, sigma0) {
  1/(2*pi*sigma0^2)*exp(-1/(2*sigma0^2)*(x^2 + y^2))
}

weight_quotient <- function(x, y, sigma0) {
  pi_func(x, y) / g_func(x, y, sigma0)
}
```

```
sample_alt1 <- function(n) {
  x = rnorm(mean = 2, sd = 1, n = n)
  y = rnorm(mean = 2, sd = 1, n = n)
  w = rep(1, n)

  return (list(
    w = w,
    theta = sample_theta(x, y, w)
  ))
}

sample_alt2 <- function(n, sigma0) {
  x <- rnorm(mean = 0, sd = sigma0, n = n)
```

```r
  y <- rnorm(mean = 0, sd = sigma0, n = n)
  w <- weight_quotient(x, y, sigma0)

  return (list(
    w = w,
    theta = sample_theta(x, y, w)
  ))
}

log_ns <- seq(from=1, to=8, by=1)
ns <- 10^log_ns

thetas <- data.frame()

for (n in ns) {
  thetas <- rbind(thetas, data.frame(
    theta_1 = sample_alt1(n)$theta,
    theta_2 = sample_alt2(n, 1)$theta,
    theta_3 = sample_alt2(n, 4)$theta
  ))
}

cbind(n = ns, thetas)
```

```
##       n  theta_1  theta_2  theta_3
## 1 1e+01 3.173275 2.017771 2.773152
## 2 1e+02 3.030681 1.956193 3.054825
## 3 1e+03 2.969024 2.662641 3.068418
## 4 1e+04 3.008998 2.668434 3.004413
## 5 1e+05 3.014471 4.074548 3.009723
## 6 1e+06 3.010757 2.994551 3.013122
## 7 1e+07 3.012519 3.010595 3.013593
## 8 1e+08 3.012538 2.996235 3.012456
```

```r
df <- melt(cbind(n = log_ns, thetas), id.var = "n")

ggplot(df, aes(x = n, y = value, color = variable)) +
  geom_line() +
  labs(title = "Theta estimates", x = "log n", y = "Estimate") +
  scale_color_manual("Estimate", values=c("black", "red", "blue"))
```



Looking at the table and plots, alternative 2 turned out to converge much slower than alternative 3. Alt 1 converges after $\approx 10^x$, alt 2 after $\approx 10^x$ and alt 3 after $\approx 10^x$.

b)

```
ess <- function(n, var_w) {
  n / (1 + var_w)
}

ess_star <- function(n, var_pi, var_g) {
  n * var_pi / var_g
}

ns <- c(10^4, 10^5, 10^6)

M = 10

ess_stars <- data.frame()
esss <- data.frame()

for (n in ns) {
  thetas <- data.frame()
  ws <- data.frame()

  for (k in 1:M) {
    s1 <- sample_alt1(n)
    s2 <- sample_alt2(n, 1)
```

```
    s3 <- sample_alt2(n, 4)

    thetas <- rbind(thetas, data.frame(
      theta_1 = s1$theta,
      theta_2 = s2$theta,
      theta_3 = s3$theta
    ))

    ws <- rbind(ws, data.frame(w1 = s1$w, w2 = s2$w, w3 = s3$w))
  }
  var_pi <- var(thetas$theta_1)

  ess_stars <- rbind(ess_stars, data.frame(
    n = n,
    theta_1 = n / n,
    theta_2 = ess_star(n, var_pi, var(thetas$theta_2)) / n,
    theta_3 = ess_star(n, var_pi, var(thetas$theta_3)) / n
  ))

  esss <- rbind(esss, data.frame(
    n = n,
    theta_1 = ess(n * M, var(ws$w1)) / (n * M),
    theta_2 = ess(n * M, var(ws$w2)) / (n * M),
    theta_3 = ess(n * M, var(ws$w3)) / (n * M)
  ))
}

print(ess_stars)
```

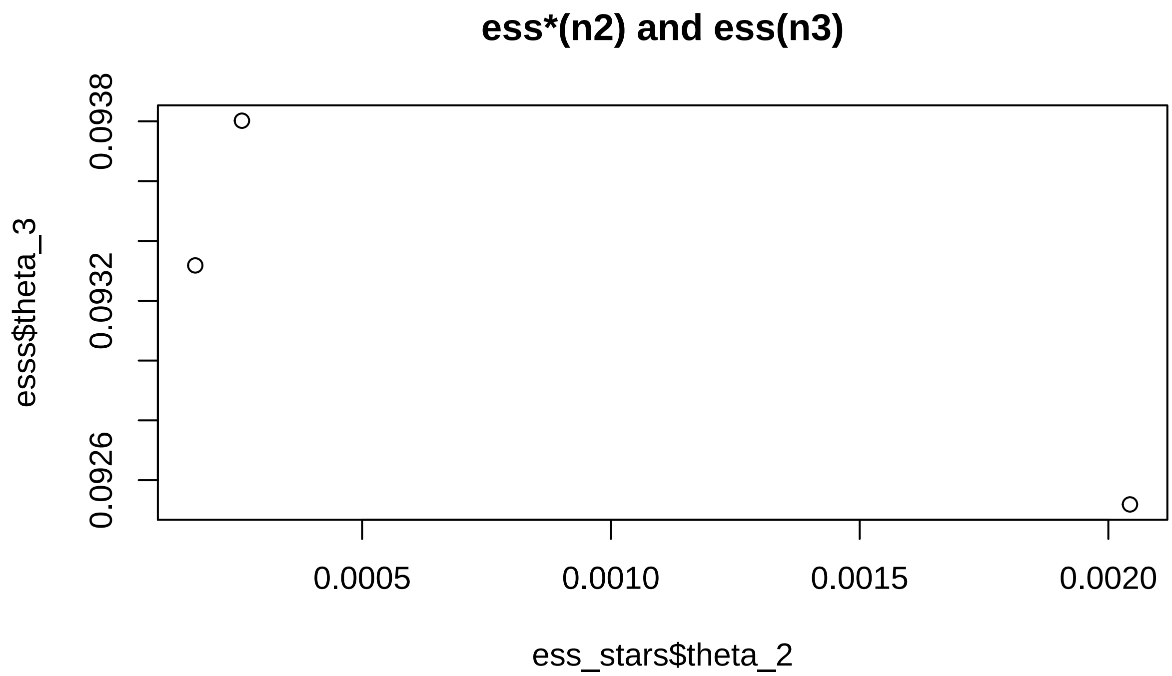
```
##        n theta_1      theta_2    theta_3
## 1 1e+04       1 0.0020434515 0.61203739
## 2 1e+05       1 0.0001645332 0.02962505
## 3 1e+06       1 0.0002582995 0.09753178
```

```
print(esss)
```

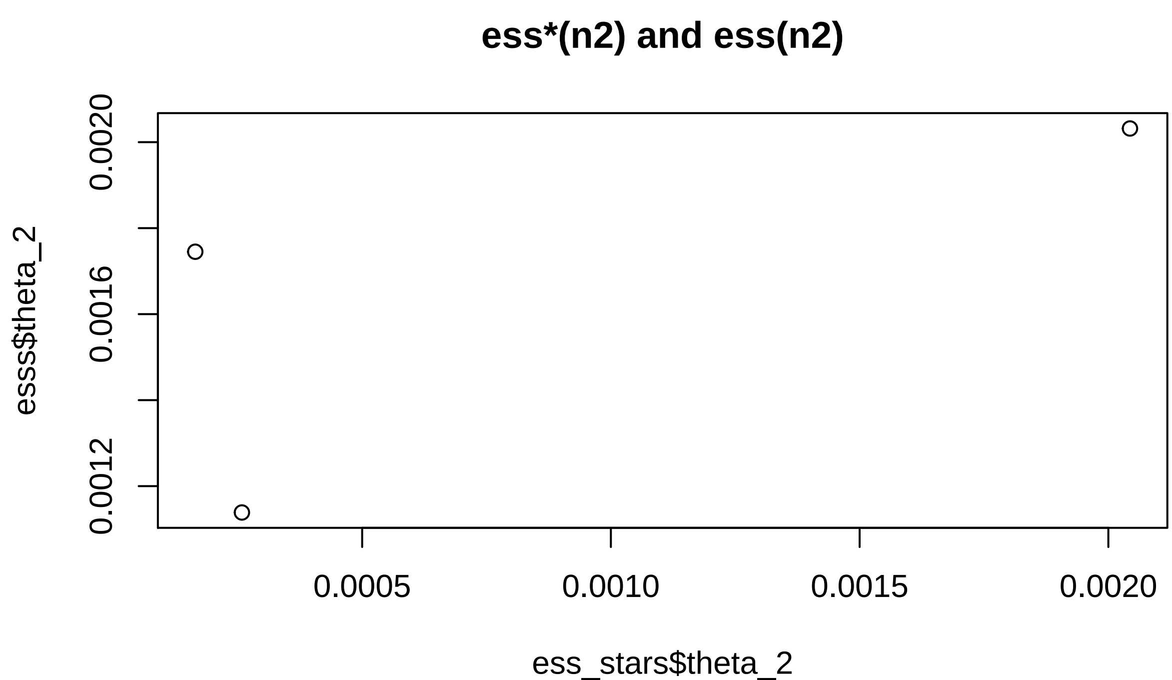
```
##        n theta_1     theta_2    theta_3
## 1 1e+04       1 0.002031786 0.09251889
## 2 1e+05       1 0.001745128 0.09331809
## 3 1e+06       1 0.001138710 0.09380209
```

In the above table, the ess and ess$*$ values are given in relation to n. Both $\frac{ess*(n_1)}{n_1}$ and $\frac{ess(n_1)}{n_1}$ are 1 which is expected. $\frac{ess*(n_2)}{n_1} < 0.1\%$ which implies that the sample size with alternative 2 must be a lot higher than when sampling π directly. $\frac{ess*(n_3)}{n_1} \approx 30\%$ which implies that the sample size with alternative 3 must be higher, but not several magnitudes so, than when sampling π directly.

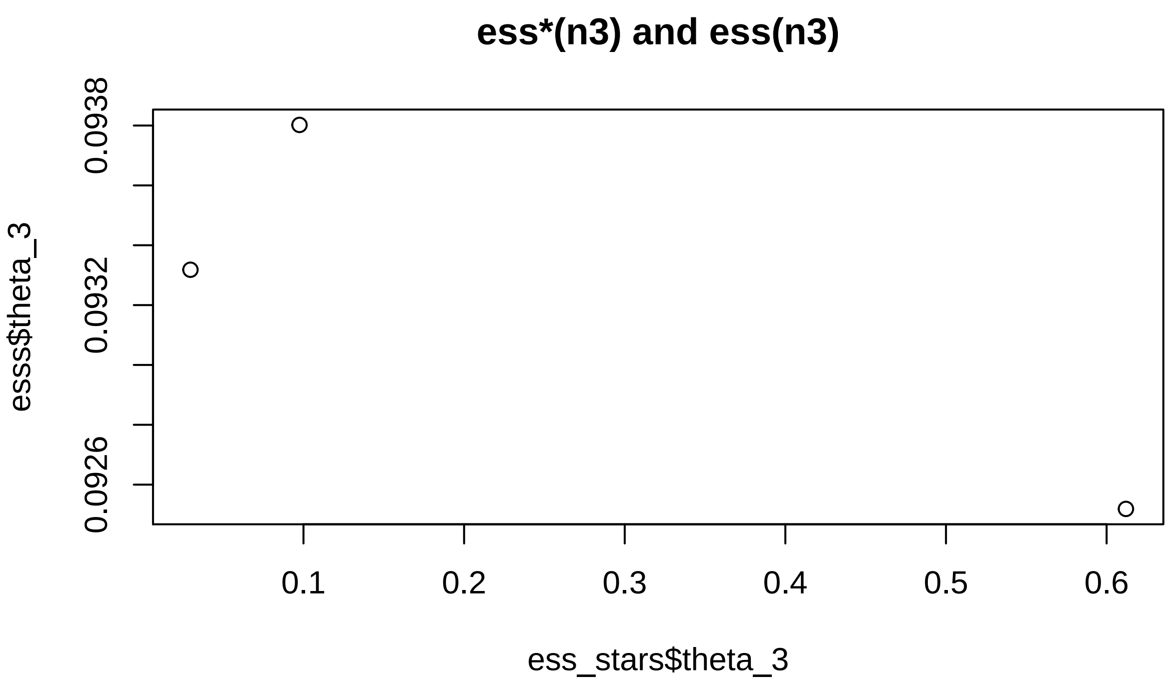
```
plot(ess_stars$theta_2, esss$theta_3, main = "ess*(n2) and ess(n3)")
```




```
plot(ess_stars$theta_2, esss$theta_2, main = "ess*(n2) and ess(n2)")
```



```r
plot(ess_stars$theta_3, esss$theta_3, main = "ess*(n3) and ess(n3)")
```



Comparing the ess estimates to the "true" $ess*$ values. The approximation seems to be within the same magnitude as the true value. This approximation is hence considered "good".

2)

a)

I have implemented two methods to calculate K. The first one is the naive approach of creating random walks and terminating when there are no more possible directions to go. Calculating $p(r) = \prod \frac{1}{n_t}$, where n_t is the number of options at step t. The second extends on the naive approach, saving states where the walk has walked a predetermined length (40 is the set parameter) where there are more than one option, and continuing from there, recursively saving such states along the way. This "skips" a lot of the early walks that are calculated again and again, and is therefore more efficient. Another possible possible approach would be a "greedy" algorithm, that e.g. though one step ahead in order to not intersect itself as fast and generate longer paths, resulting in a more accurate estimate.

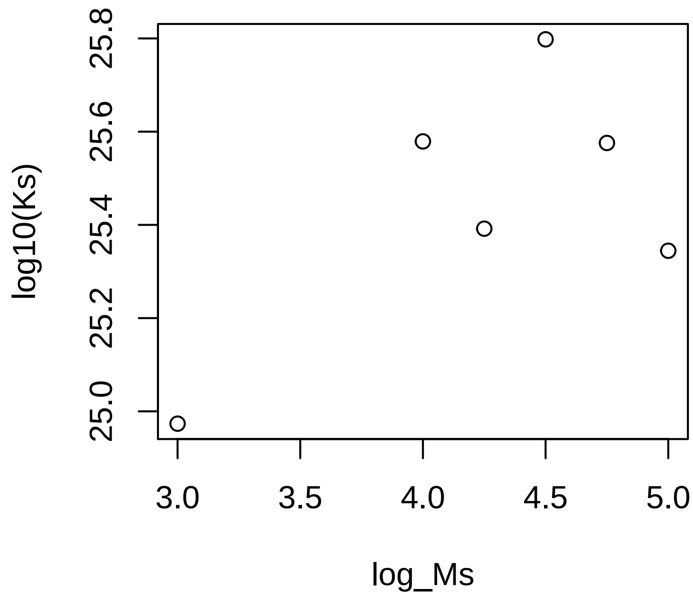
As my algorithm is not efficient enough, as well as my laptop not fast enough, I have only manged to run the algorithm for $n <= 10^5$.

For each of the methods, a log-log plot of K agains N is provided, as well as the length and walk of the longest walk. In addition, a histogram is provided of the length of the walks, together with a histogram weighed by $p(r)$

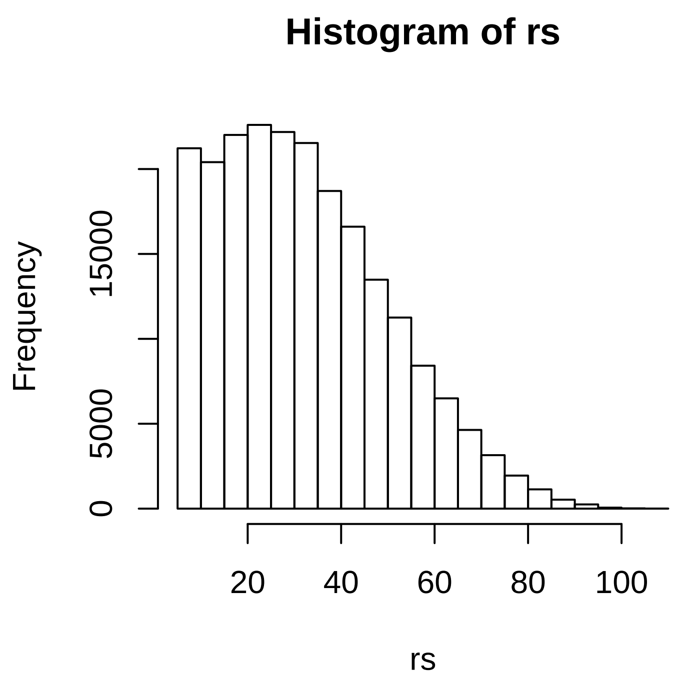
Naive approach

```r
library(dqrng)
library(plotrix)

N <- 10
L <- N + 1

lattice = matrix(0, nrow=L, ncol=L)

plot_lattice <- function(lattice) {
  image(t(apply(lattice, 1, rev)))
}

POSSIBLE_DIRS <- list(
  c(1, 0),
  c(-1, 0),
  c(0, 1),
  c(0, -1)
)
allowed_directions <- function(lattice, r, c) {
  allowed_dirs <- list()

  k <- 1
  for (dir in POSSIBLE_DIRS) {
    rr <- r + dir[1]
    cc <- c + dir[2]

    is_allowed <- !((rr > L || rr < 1 || cc > L || cc < 1) || lattice[rr, cc] != 0)
    if (is_allowed) {
      allowed_dirs[[k]] <- dir
      k <- k + 1
    }
  }
  return (allowed_dirs)
}

set.seed(2020)

run_SAW <- function() {
  lattice = matrix(0, nrow=L, ncol=L)

  r <- 1
  c <- 1

  R <- 0
  p <- 1

  while (TRUE) {
    lattice[r, c] = 100 + R
    R <- R + 1

    dirs <- allowed_directions(lattice, r, c)
```

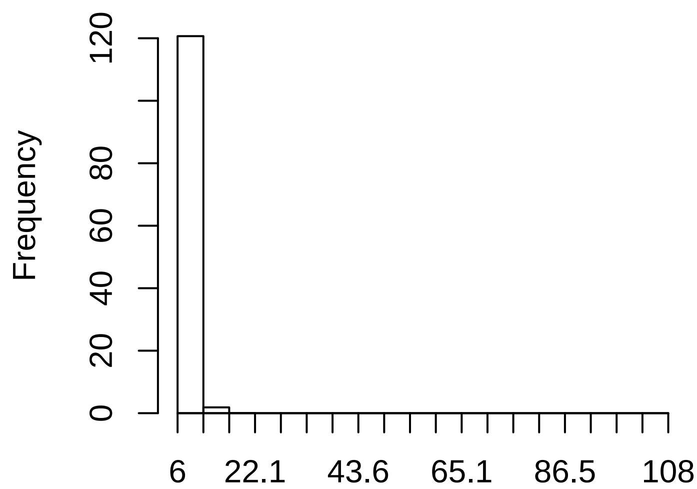
```
    m <- length(dirs)

    if (m < 1) {
      break;
    }

    p <- p * 1/m

    dir <- dirs[[dqsample.int(m, size = 1)]]
    r <- r + dir[1]
    c <- c + dir[2]
  }

  return (list(
    lattice = lattice,
    p = p,
    r = R,
    in_corner = r == L && c == L
  ))
}

log_Ms <- c(3, seq(from=4, to=5, by=0.25))
Ms = 10^log_Ms

Ks <- c()

pps <- c()
rs <- c()

longest_r <- 0
longest_lattice <- NULL

for (M in Ms) {
  ps <- c()
  for (m in 1:M) {
    res <- run_SAW()
    ps <- c(ps, res$p)
    rs <- c(rs, res$r)

    if (res$r > longest_r) {
      longest_r <- res$r
      longest_lattice <- res$lattice
    }
  }
  K <- 1/M * sum(1 / ps)
  Ks <- c(Ks, K)
  pps <- c(pps, ps)
}

plot(log_Ms, log10(Ks))
```


log10(Ks)
25.0
25.2
25.4
25.6
25.8
3.0
3.5
4.0
4.5
5.0
log_Ms

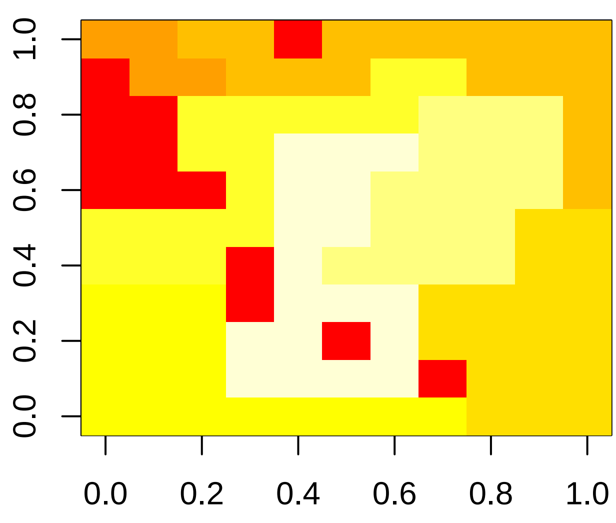
```r
hist(rs)
```


Histogram of rs
Frequency
0
5000
15000
20
40
60
80
100
rs

```r
weighted.hist(rs, pps)
```




```r
plot_lattice(longest_lattice)
```



```r
print(longest_r)
```

```
## [1] 108
```

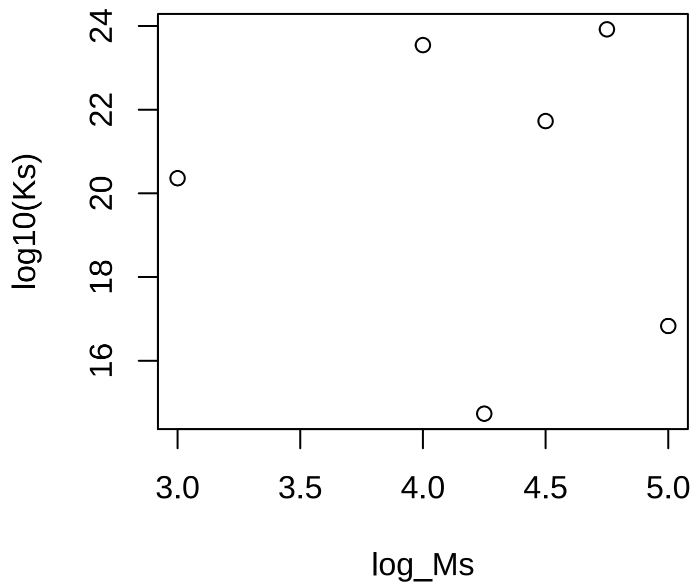
```r
print(longest_lattice)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
##  [1,]  100    0    0    0    0  159  158  153  152   147   146
##  [2,]  101  102    0    0    0  160  157  154  151   148   145
##  [3,]  104  103  166  165    0  161  156  155  150   149   144
##  [4,]  105  106  167  164  163  162    0    0  197   196   143
##  [5,]    0  107  168  205  204  201  200  199  198   195   142
##  [6,]  109  108  169  206  203  202  189  190    0   194   141
##  [7,]  110  171  170  207  186  187  188  191  192   193   140
##  [8,]  111  172  173  174  185  184  183  134  135     0   139
##  [9,]  112  113  176  175  180  181  182  133  136   137   138
## [10,]  115  114  177  178  179  122  123  132  131   130   129
## [11,]  116  117  118  119  120  121  124  125  126   127   128
```

Extended method

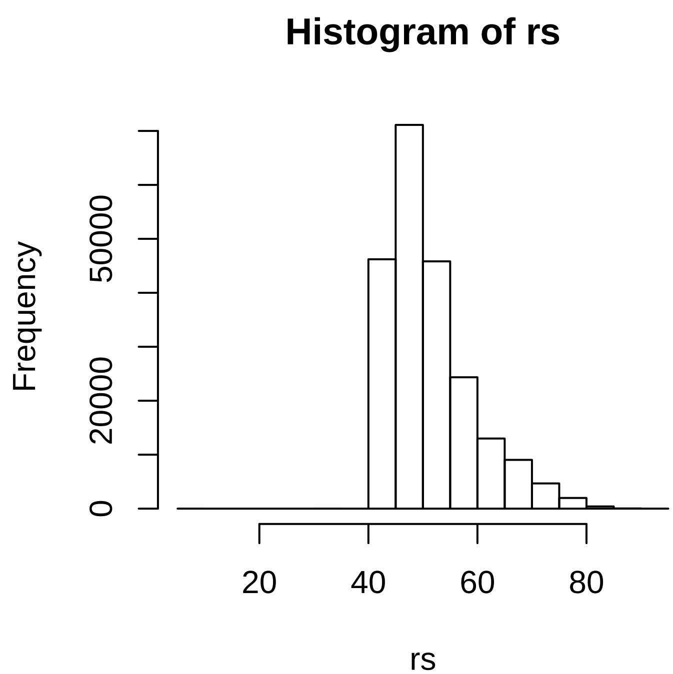
```r
r_lim <- 40

run_SAW2 <- function(lattice = matrix(0, nrow=L, ncol=L), r = 1, c = 1, R = 0, p = 1) {
  states <- list()
  while (TRUE) {
    lattice[r, c] = 100 + R
    R <- R + 1

    dirs <- allowed_directions(lattice, r, c)
    m <- length(dirs)

    if (m < 1) {
      break;
    }

    if (R >= r_lim && m > 1) {
      states <- append(states, list(lattice=lattice,r=r,c=c,R=R-1,p=p))
    }

    p <- p * 1/m

    dir <- dirs[[dqsample.int(m, size = 1)]]
    r <- r + dir[1]
    c <- c + dir[2]
  }

  return (list(
    states = states,
    p = p,
```

```
    r = R,
    lattice = lattice,
    in_corner <- r == L && c == L
  ))
}

Ks <- c()

pps <- c()
rs <- c()

longest_r <- 0
longest_lattice <- NULL

for (M in Ms) {
  ps <- c()
  m <- 1
  states <- list()
  while (m < M) {
    res <- NULL

    if (length(states) < 1) {
      res <- run_SAW2()
    } else {
      state <- states[[length(states)]]
      states <- states[-length(states)]

      res <- run_SAW2(lattice=state$lattice, r=state$r, c=state$c, R=state$R, p=state$p)
    }
    ps <- c(ps, res$p)
    rs <- c(rs, res$r)

    if (res$r > longest_r) {
      longest_r <- res$r
      longest_lattice <- res$lattice
    }

    if (length(res$states) > 0) {
      states <- append(states, list(res$states))
    }

    m <- m + 1
  }
  K <- 1/M * sum(1 / ps)
  Ks <- c(Ks, K)

  pps <- c(pps, ps)
}

plot(log_Ms, log10(Ks))
```



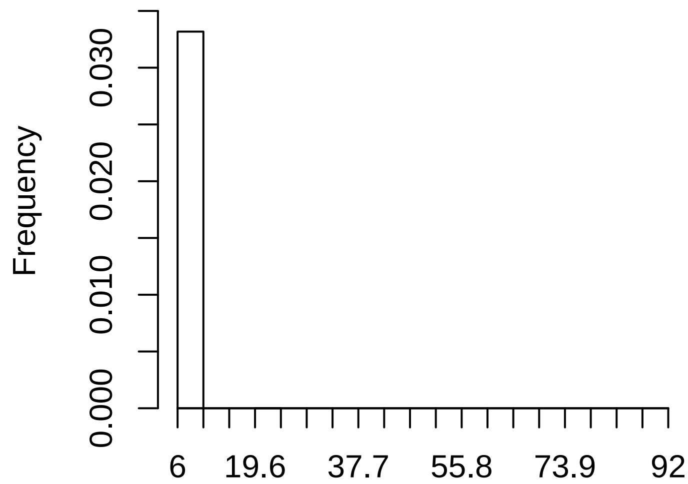
24
22
20
18
16
log10(Ks)
3.0
3.5
4.0
4.5
5.0
log_Ms


```
hist(rs)
```

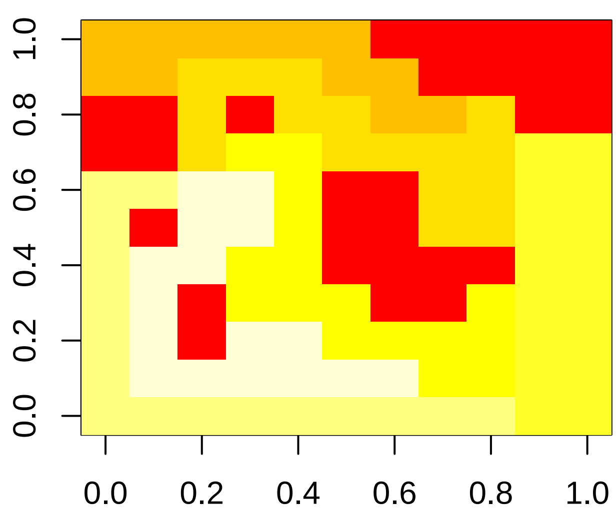


Histogram of rs
Frequency
50000
20000
0
20
40
60
80
rs

```r
weighted.hist(rs, pps)
```




```r
plot_lattice(longest_lattice)
```



```r
print(longest_r)
```

```
## [1] 92
```

```r
print(longest_lattice)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
##  [1,]  100  101    0    0  174  173  172  171  170   169   168
##  [2,]  103  102    0    0  175    0  181  182  183   184   167
##  [3,]  104  125  126  127  176  179  180    0    0   185   166
##  [4,]  105  124    0  128  177  178  133  134  187   186   165
##  [5,]  106  123  122  129  130  131  132  135  188   189   164
##  [6,]  107  108  121  120    0    0    0  136  137   190   163
##  [7,]    0  109  110  119    0    0    0    0  138   191   162
##  [8,]    0    0  111  118  117  116    0    0  139   140   161
##  [9,]    0    0  112  113  114  115    0  143  142   141   160
## [10,]    0    0    0  148  147  146  145  144  155   156   159
## [11,]    0    0    0  149  150  151  152  153  154   157   158
```

b)

For both approaches, I attempted to run the same algorithm, but only "counting" the walks that ended in the (N, N) corner. However, the algorithm didn't run fast enough to produce any results on my laptop.

Naive approach

```r
# Ms = 10^log_Ms
#
# Ks <- c()
#
# pps <- c()
# rs <- c()
#
# longest_r <- 0
# longest_lattice <- NULL
#
# for (M in Ms) {
#   ps <- c()
#   m <- 1
#   while (m < M) {
#     res <- run_SAW()
#
#     if (!res$in_corner) {
#       next;
#     }
#
#     ps <- c(ps, res$p)
#     rs <- c(rs, res$r)
#
#     if (res$r > longest_r) {
```

```
#       longest_r <- res$r
#       longest_lattice <- res$lattice
#     }
#     M <- M + 1
#   }
#   K <- 1/M * sum(1 / ps)
#   Ks <- c(Ks, K)
#   pps <- c(pps, ps)
# }
#
# plot(log_Ms, log10(Ks))
#
# hist(rs)
#
# weighted.hist(rs, pps)
#
# plot_lattice(longest_lattice)
# print(longest_r)
# print(longest_lattice)
```

Extended method

```
# Ks <- c()
#
# pps <- c()
# rs <- c()
#
# longest_r <- 0
# longest_lattice <- NULL
#
# for (M in Ms) {
#   ps <- c()
#   m <- 1
#   states <- list()
#   while (m < M) {
#     res <- NULL
#
#     if (length(states) < 1) {
#       res <- run_SAW2()
#     } else {
#       state <- states[[length(states)]]
#       states <- states[-length(states)]
#
#       res <- run_SAW2(lattice=state$lattice, r=state$r, c=state$c, R=state$R, p=state$p)
#     }
#
#     if (!res$in_corner) {
#       next;
#     }
#
#     ps <- c(ps, res$p)
```

```
#     rs <- c(rs, res$r)
#
#     if (res$r > longest_r) {
#       longest_r <- res$r
#       longest_lattice <- res$lattice
#     }
#
#     if (length(res$states) > 0) {
#       states <- append(states, list(res$states))
#     }
#
#     m <- m + 1
#   }
#   K <- 1/M * sum(1 / ps)
#   Ks <- c(Ks, K)
#
#   pps <- c(pps, ps)
# }
#
# plot(log_Ms, log10(Ks))
#
# hist(rs)
#
# weighted.hist(rs, pps)
#
# plot_lattice(longest_lattice)
# print(longest_r)
# print(longest_lattice)
```

c)

These are provided in a) and b) respectively.