Stats 102C Final

Mateo Umaguing, 005318989

August 1, 2022

Problem 1

(1) Transition matrix

$$K = \begin{bmatrix} 1/2 & 1/4 & 1/4 \\ 1/4 & 1/2 & 1/4 \\ 1/4 & 1/4 & 1/4 \end{bmatrix}$$

$$p^{(t+1)} = p^{(t)} K$$

If states were places with people, $p^{(t)}$ is the distribution of people in each state at time t. K is the transition matrix, which can be interpreted as a list of proportions of people who will move to a different state. Thus, multiplying $p^{(t)} = \begin{bmatrix} a & b & c \end{bmatrix}$ by a matrix K:

$$p^{(t)} K = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} 1/2 & 1/4 & 1/4 \\ 1/4 & 1/2 & 1/4 \\ 1/4 & 1/4 & 1/4 \end{bmatrix}$$

will multiply the row vector with each column of K. Each column of K can represent the amount of people gained from each state including itself.

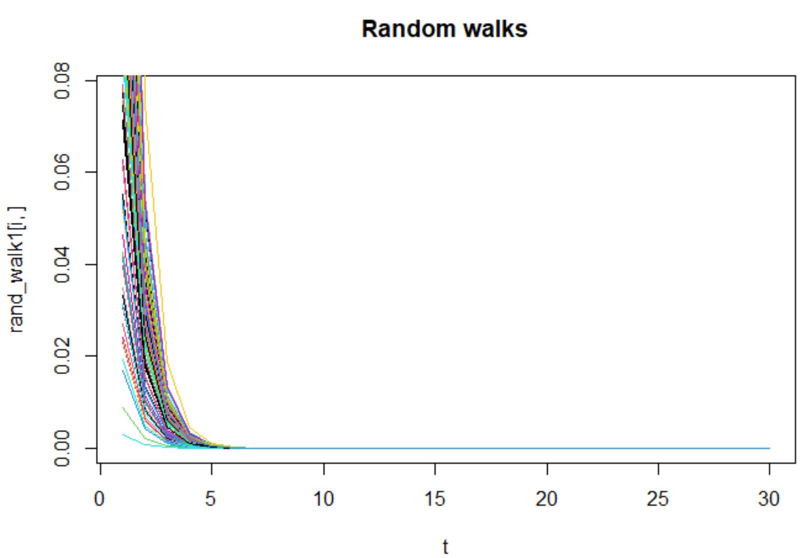
$$p^{(t+1)} = \begin{bmatrix} a/2 + b/4 + c/4 & a/4 + b/2 + c/4 & a/4 + b/4 + c/2 \end{bmatrix}$$

Each element of $p^{(t+1)}$ is a state, and the same state as $p^{(t)}$: half of the people in a, a quarter of the people in b, and a quarter of the people in c are now in a; a quarter of the people in a, half of the people in b and a quarter of the people in c are now in b; and so on. Thus, matrix multiplying $p^{(t)}$ by K yields another transition of the states.

(2) Computing $p^{(t)}$

```
K = matrix(c(0.5, 0.25, 0.25,
             0.25, 0.5, 0.25,
             0.25, 0.25, 0.5), nrow = 3, ncol = 3, byrow = T)

one_step_transition = function(p, K){
  return(p %*% K)
}

n = 100 # number of simulations
m = 30 # number of transitions per simulation

rand_walk = function(n, m){
  walk = matrix(0, nrow = n, ncol = m)
  for(i in 1:n){
    p = runif(3)  # Some initialization of p
```

```
    p = p / sum(p) # Normalization to add up to 1
    for(j in 1:m){
      p = one_step_transition(p, K) # transition one step
      walk[i,j] = sum(abs(p - 1/3)) # Obtain relative distance to uniformity
    }
  }
  return(walk)
}

rand_walk1 = rand_walk(n = 100, m = 30)

plot_func = function(rand_walk){
  for(i in 1:nrow(rand_walk)){
    if(i == 1){
      plot(1:30, rand_walk1[i,], type = "l", main = "Random walks", xlab = "t", col = i)
    }
    else{
      lines(rand_walk1[i,], col = i)
    }
  }
}

plot_func(rand_walk1)
```

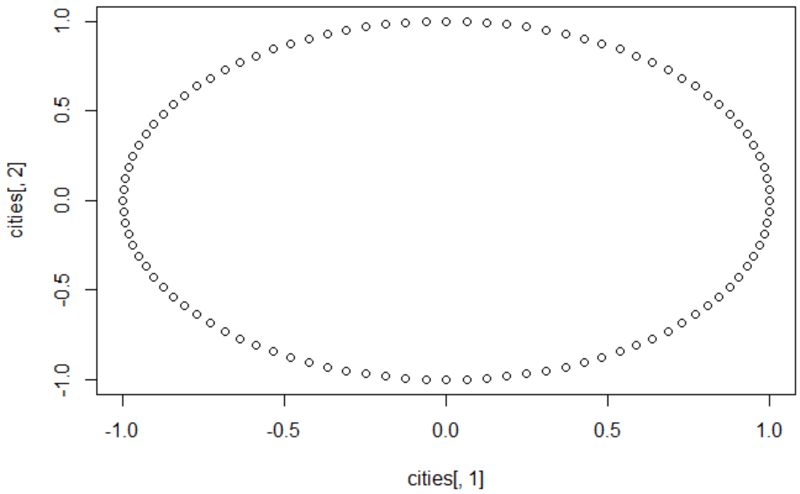


Problem 2

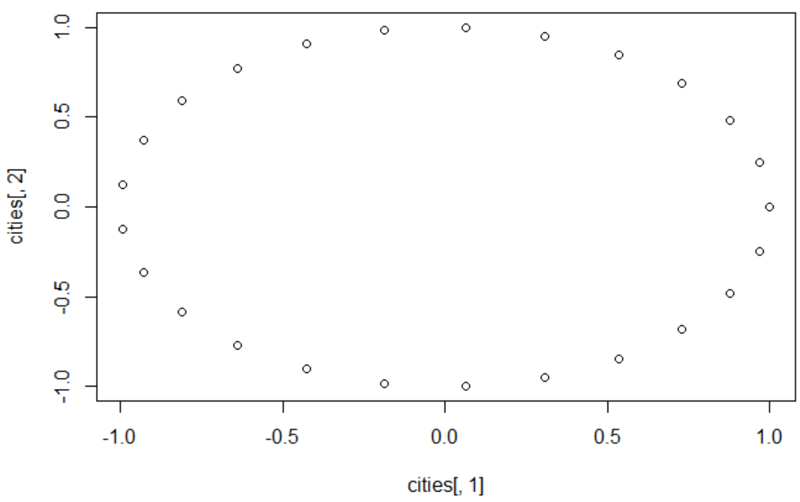
(1) Generating cities

```
# num_cities = m + 1 cities
gen_cities = function(num_cities){
  theta = seq.int(from = 0, to = 2*pi, length.out = num_cities+1)
  theta = theta[-length(theta)] # remove off the last number since last position and
  # first position will be the same
```

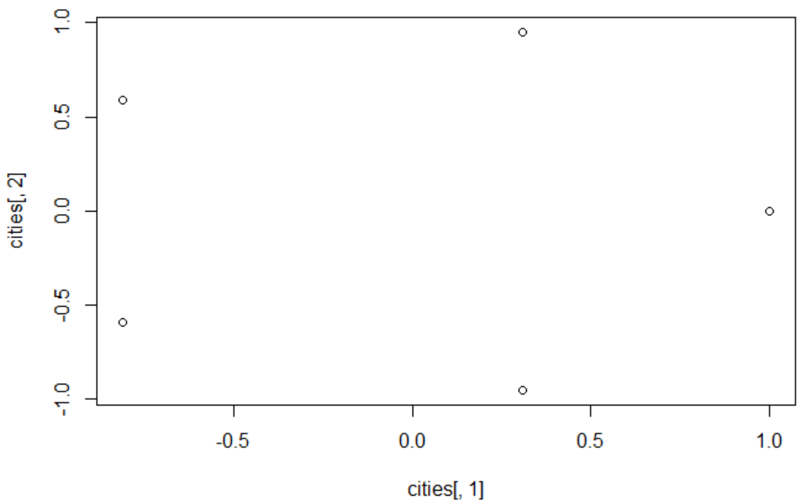
```
  x_cord = cos(theta)
  y_cord = sin(theta)
  return(cbind(x_cord, y_cord))
}

cities = gen_cities(100)
plot(cities[,1], cities[,2]) # To check if we have points around a circle
```




```
cities = gen_cities(25)
plot(cities[,1], cities[,2]) # To check if we have points around a circle
```



```r
cities = gen_cities(5)
plot(cities[,1], cities[,2]) # To check if we have points around a circle
```



Distance between each city in a matrix:

```r
distance_cities = function(generated_cities){
  dist_matrix = matrix(0, nrow = nrow(generated_cities), ncol = nrow(generated_cities))
  for(i in 1:nrow(generated_cities)){
    for(j in 1:nrow(generated_cities)){
      dist_matrix[i,j] = sqrt(sum((generated_cities[i,] - generated_cities[j,])^2))
    }
  }
  return(dist_matrix)
}

dist_matrix = distance_cities(cities)
dist_matrix
```

Output:

```
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 0.000000 1.175571 1.902113 1.902113 1.175571
[2,] 1.175571 0.000000 1.175571 1.902113 1.902113
[3,] 1.902113 1.175571 0.000000 1.175571 1.902113
[4,] 1.902113 1.902113 1.175571 0.000000 1.175571
[5,] 1.175571 1.902113 1.902113 1.175571 0.000000
```

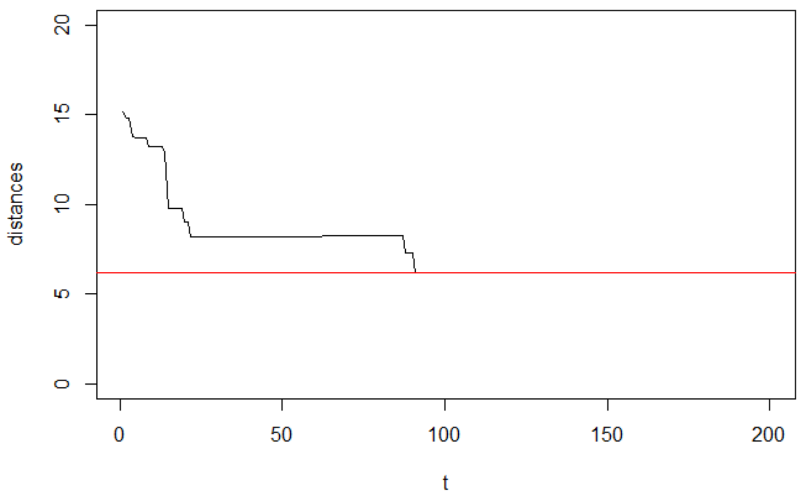
(2) Plotting distance of paths

```r
path_distance = function(path, dist_matrix){
  start = path[1]
  path_dist = 0
  for(i in 2:length(path)){
    end = path[i]
```

```
    path_dist = path_dist + dist_matrix[start, end]
    start = path[i]
  }
  # Include distance back to original start
  path_dist = path_dist + dist_matrix[start, path[1]]
  return(path_dist)
}

metropolis = function(path, dist_matrix, temp) {
  orig_dist = path_distance(path, dist_matrix)
  orig_path = path

  # swap two random indices
  swap = sample(seq_along(path), 2)
  path[swap] = path[rev(swap)]

  new_path = path
  new_dist = path_distance(path, dist_matrix)
  prob_accept = min(1, exp((orig_dist - new_dist)/temp))
  # randomly accept or reject
  if (runif(1) > prob_accept) {
    new_path = orig_path
  }
  return(new_path)
}

met_exp = function(temp, num_cities, loops){
  cities = gen_cities(num_cities)
  dist_matrix = distance_cities(cities)
  path = 1:nrow(dist_matrix)
  path = path[sample(path)]
  track_distance = numeric(loops)
  for(j in 1:loops){
    path = metropolis(path, dist_matrix, temp)
    track_distance[j] = path_distance(path, dist_matrix)
  }
  return(track_distance)
}
```

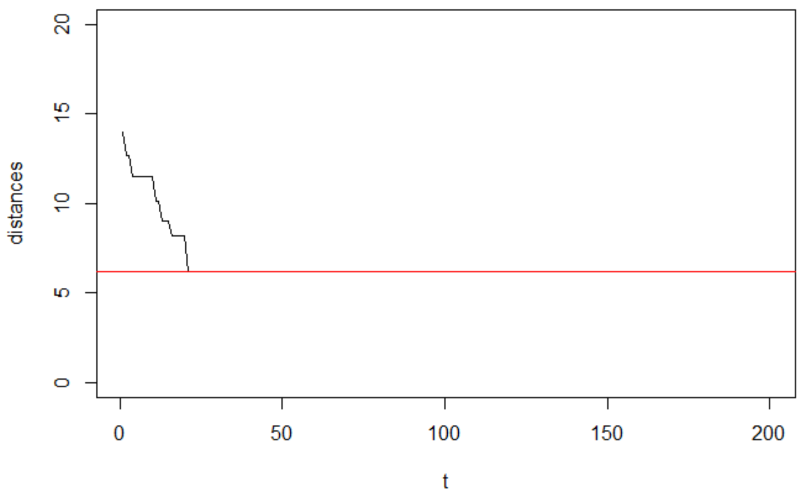
$T = 0.1$ random start 1

```
# Run through it multiple times for random starts
num_cities = 10
temp = 0.1
loops = 200
distances = met_exp(temp = 0.1, num_cities = 10, loops = 200)
dist_matrix = distance_cities(gen_cities(10))
shortest_dist = dist_matrix[1,2]*num_cities

plot(1:loops, distances, type = "l", xlab = "t", ylim = c(0, 20))
abline(h = shortest_dist, col = 'red')
```



$T = 0.1$ random start 2

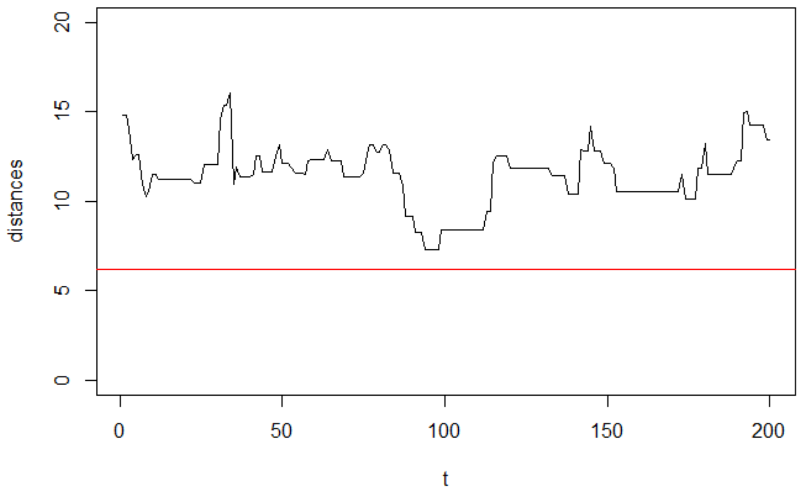
```
distances = met_exp(temp = 0.1, num_cities = 10, loops = 200)
dist_matrix = distance_cities(gen_cities(10))
shortest_dist = dist_matrix[1,2]*num_cities

plot(1:loops, distances, type = "l", xlab = "t", ylim = c(0, 20))
abline(h = shortest_dist, col = 'red')
```



$T = 1$ random start 1

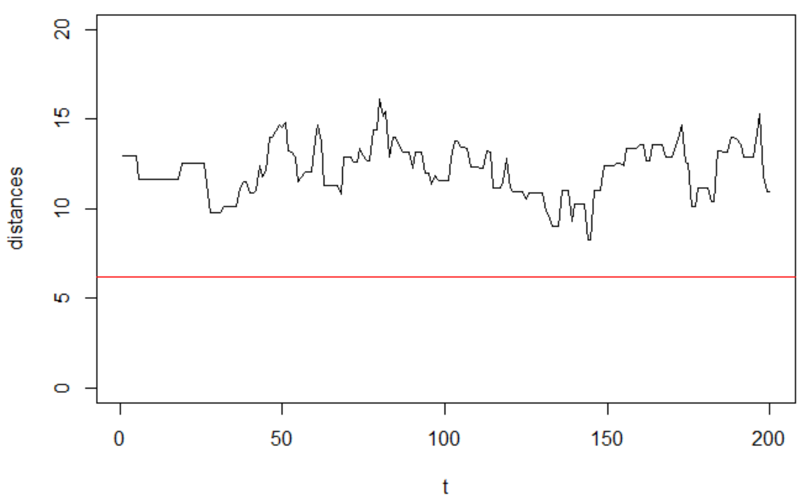
```
num_cities = 10
temp = 1
loops = 200
distances = met_exp(temp = 1, num_cities = 10, loops = 200)
```

```
dist_matrix = distance_cities(gen_cities(10))
shortest_dist = dist_matrix[1,2]*num_cities

plot(1:loops, distances, type = "l", xlab = "t", ylim = c(0, 20))
abline(h = shortest_dist, col = 'red')
```



$T = 1$ random start 2

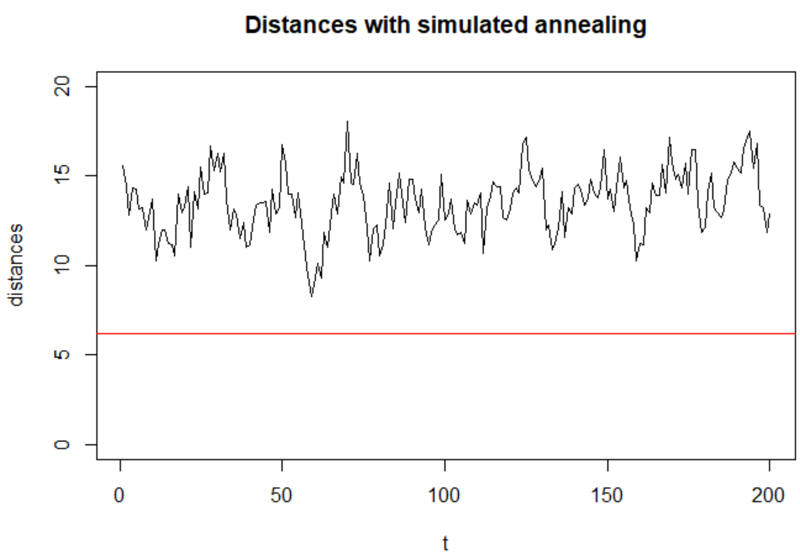
```
distances = met_exp(temp = 1, num_cities = 10, loops = 200)
dist_matrix = distance_cities(gen_cities(10))
shortest_dist = dist_matrix[1,2]*num_cities

plot(1:loops, distances, type = "l", xlab = "t", ylim = c(0, 20))
abline(h = shortest_dist, col = 'red')
```



(3) Simulated Annealing

$T = 100$

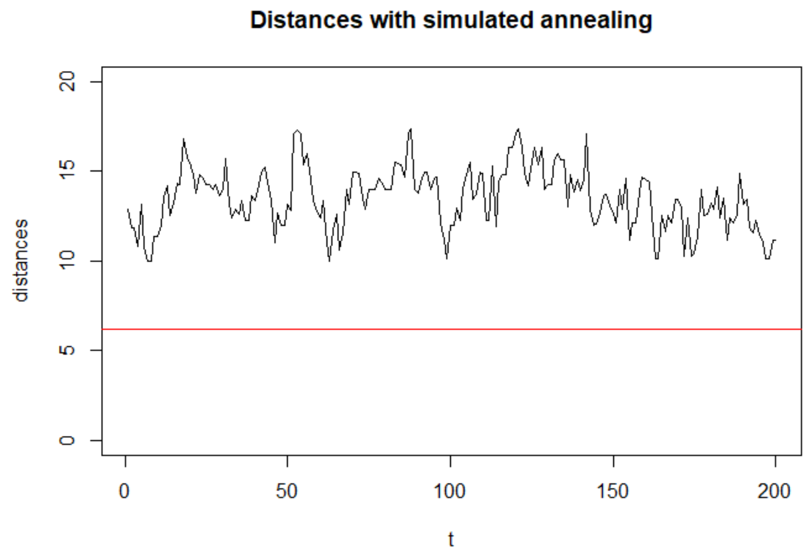
```
temp = c(100, 10, 1, 0.1, 0.01)

num_cities = 10
#temp = 100
loops = 200
distances = met_exp(temp = temp[1], num_cities = 10, loops = 200)
dist_matrix = distance_cities(gen_cities(10))
shortest_dist = dist_matrix[1,2]*num_cities

plot(1:loops,
     distances,
     type = "l",
     xlab = "t",
     ylim = c(0, 20))
title("Distances with simulated annealing")
abline(h = shortest_dist, col = 'red')
```



$T = 10$

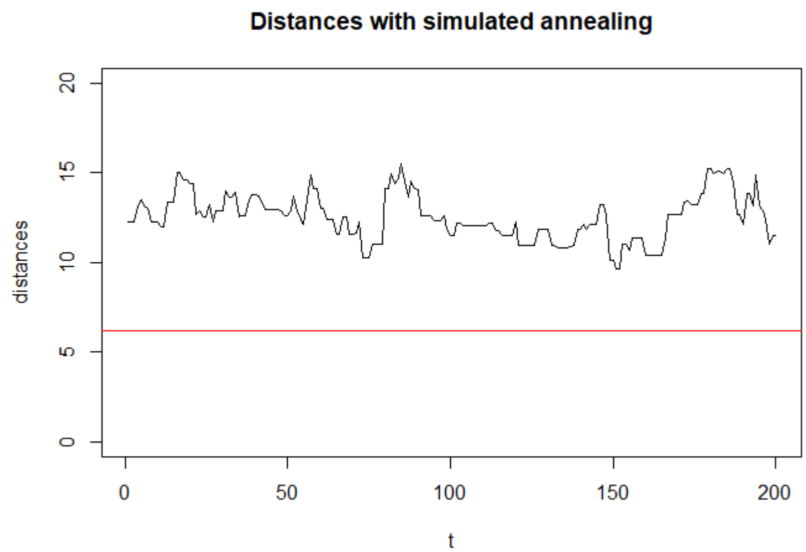
```
temp = c(100, 10, 1, 0.1, 0.01)

num_cities = 10
#temp = 10
loops = 200
distances = met_exp(temp = temp[2], num_cities = 10, loops = 200)
dist_matrix = distance_cities(gen_cities(10))
shortest_dist = dist_matrix[1,2]*num_cities

plot(1:loops,
     distances,
     type = "l",
```

```
      xlab = "t",
      ylim = c(0, 20))
title("Distances with simulated annealing")
abline(h = shortest_dist, col = 'red')
```



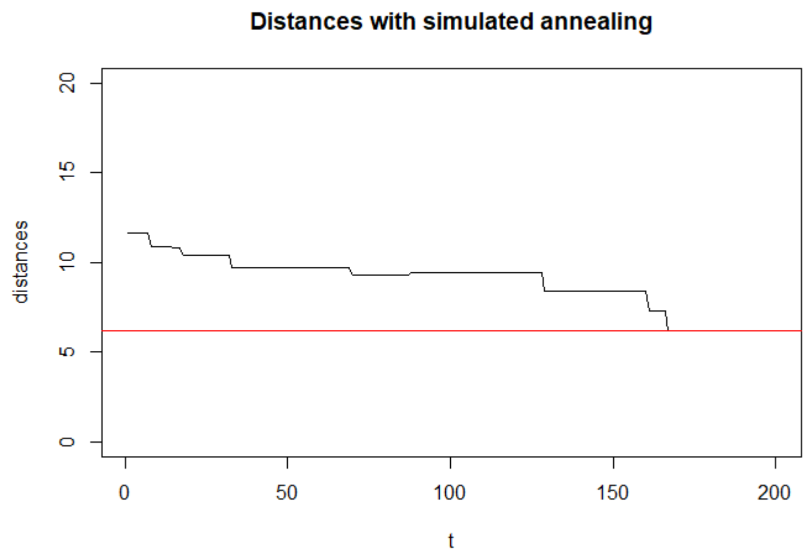
$T = 1$

```
temp = c(100, 10, 1, 0.1, 0.01)

num_cities = 10
#temp = 1
loops = 200
distances = met_exp(temp = temp[3], num_cities = 10, loops = 200)
dist_matrix = distance_cities(gen_cities(10))
shortest_dist = dist_matrix[1,2]*num_cities

plot(1:loops,
     distances,
     type = "l",
     xlab = "t",
     ylim = c(0, 20))
title("Distances with simulated annealing")
abline(h = shortest_dist, col = 'red')
```



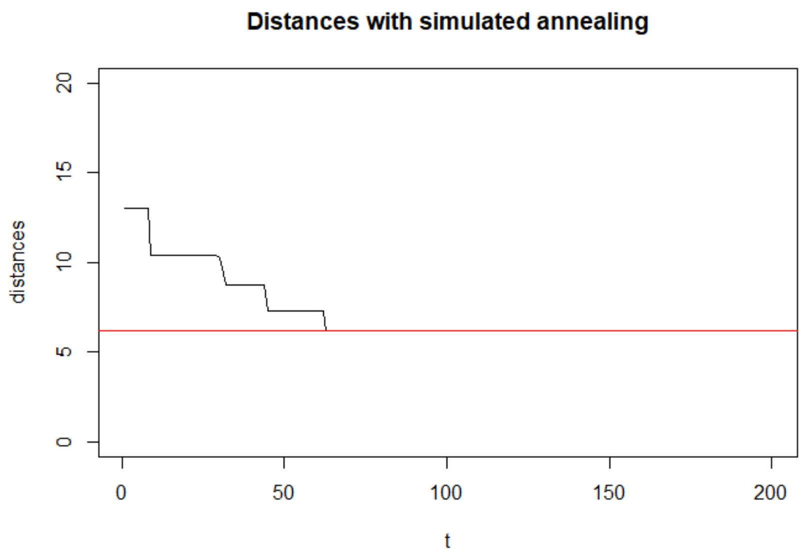
$T = 0.1$

```
temp = c(100, 10, 1, 0.1, 0.01)

num_cities = 10
#temp = 0.1
loops = 200
distances = met_exp(temp = temp[4], num_cities = 10, loops = 200)
dist_matrix = distance_cities(gen_cities(10))
shortest_dist = dist_matrix[1,2]*num_cities

plot(1:loops,
     distances,
     type = "l",
     xlab = "t",
     ylim = c(0, 20))
title("Distances with simulated annealing")
abline(h = shortest_dist, col = 'red')
```



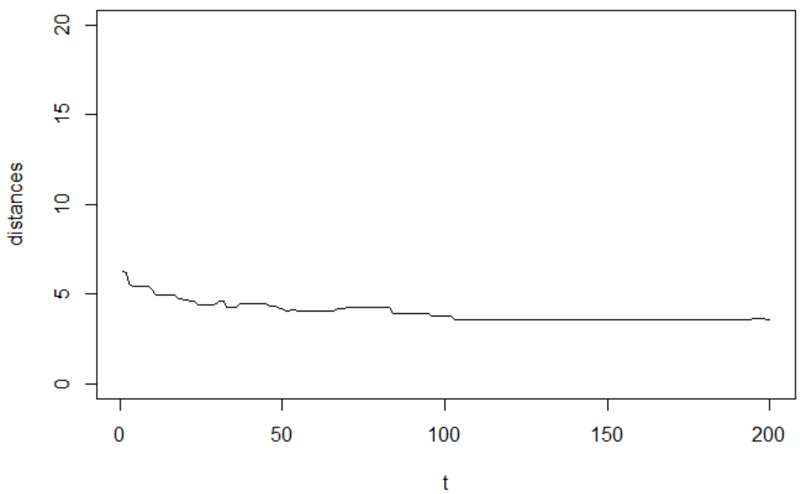
$T = 0.01$

```
temp = c(100, 10, 1, 0.1, 0.01)

num_cities = 10
#temp = 0.01
loops = 200
distances = met_exp(temp = temp[5], num_cities = 10, loops = 200)
dist_matrix = distance_cities(gen_cities(10))
shortest_dist = dist_matrix[1,2]*num_cities

plot(1:loops,
     distances,
     type = "l",
     xlab = "t",
     ylim = c(0, 20))
title("Distances with simulated annealing")
abline(h = shortest_dist, col = 'red')
```



(4) Cities in unit square

```
gen_cities = function(num_cities){
  x_cords = c(0,runif(num_cities, min = 0, max = 1))
  y_cords = c(0,runif(num_cities, min = 0, max = 1))
  return(cbind(x_cords, y_cords))
}



# Plot is the same

num_cities = 10
temp = 0.1
loops = 200
distances = met_exp(temp = .1, num_cities = 10, loops = 200)
dist_matrix = distance_cities(gen_cities(10))

plot(1:loops, distances, type = "l", xlab = "t", ylim = c(0, 20))
```



Problem 3

Gibbs sampler

```
gibbs <- function(rho, T_max, x_0, y_0){
  x_t <- x_0
  y_t <- y_0
  x <- c()
  y <- c()

  for (t in 1:T_max) {
    x_t <- rho * y_t + sqrt(1 - rho^2)*rnorm(1)
    y_t <- rho * x_t + sqrt(1 - rho^2)*rnorm(1)
    x <- c(x, x_t)
    y <- c(y, y_t)
  }
  return(list(x = x, y = y))
}
```

$\rho = 0.1, \ T = 100, \ M = 100, \ (X_0, Y_0) = (0, 0)$

```
rho = 0.1
T_max = 100
M = 100
x_0 = 0
y_0 = 0

M_chains = data.frame(matrix(0, ncol = 4, nrow = M * T_max))
colnames(M_chains) = c("t", "m", "x", "y")

for (m in 0:(M-1)) {
  gibbs_result = gibbs(rho, T_max, x_0, y_0)

  iter_start = m * T_max + 1
```

```
  iter_end = (m+1) * T_max

  M_chains$x[iter_start:iter_end] = gibbs_result$x
  M_chains$y[iter_start:iter_end] = gibbs_result$y
  M_chains$t[iter_start:iter_end] = 1:T_max
  M_chains$m[iter_start:iter_end] = rep(m, T_max)
}
```

$\rho = 0.5,\ T = 100,\ M = 100,\ (X_0, Y_0) = (0, 0)$

```
rho = 0.5
T_max = 100
M = 100
x_0 = 0
y_0 = 0

M_chains = data.frame(matrix(0, ncol = 4, nrow = M * T_max))
colnames(M_chains) = c("t", "m", "x", "y")

for (m in 0:(M-1)) {
  gibbs_result = gibbs(rho, T_max, x_0, y_0)

  iter_start = m * T_max + 1
  iter_end = (m+1) * T_max

  M_chains$x[iter_start:iter_end] = gibbs_result$x
  M_chains$y[iter_start:iter_end] = gibbs_result$y
  M_chains$t[iter_start:iter_end] = 1:T_max
  M_chains$m[iter_start:iter_end] = rep(m, T_max)
}
```

$\rho = 0.9,\ T = 100,\ M = 100,\ (X_0, Y_0) = (0, 0)$

```
rho = 0.9
T_max = 100
M = 100
x_0 = 0
y_0 = 0

M_chains = data.frame(matrix(0, ncol = 4, nrow = M * T_max))
colnames(M_chains) = c("t", "m", "x", "y")

for (m in 0:(M-1)) {
  gibbs_result = gibbs(rho, T_max, x_0, y_0)

  iter_start = m * T_max + 1
  iter_end = (m+1) * T_max

  M_chains$x[iter_start:iter_end] = gibbs_result$x
  M_chains$y[iter_start:iter_end] = gibbs_result$y
  M_chains$t[iter_start:iter_end] = 1:T_max
  M_chains$m[iter_start:iter_end] = rep(m, T_max)
}
```

$\rho = 0.99,\ T = 100,\ M = 100,\ (X_0, Y_0) = (0, 0)$

```
rho = 0.99
T_max = 100
M = 100
x_0 = 0
y_0 = 0

M_chains = data.frame(matrix(0, ncol = 4, nrow = M * T_max))
colnames(M_chains) = c("t", "m", "x", "y")

for (m in 0:(M-1)) {
  gibbs_result = gibbs(rho, T_max, x_0, y_0)

  iter_start = m * T_max + 1
  iter_end = (m+1) * T_max

  M_chains$x[iter_start:iter_end] = gibbs_result$x
  M_chains$y[iter_start:iter_end] = gibbs_result$y
  M_chains$t[iter_start:iter_end] = 1:T_max
  M_chains$m[iter_start:iter_end] = rep(m, T_max)
}
```

$\rho = 0.1,\ T = 100,\ M = 100,\ (X_0, Y_0) = (-10, -10)$

```
rho = 0.1
T_max = 100
M = 100
x_0 = -10
y_0 = -10

M_chains = data.frame(matrix(0, ncol = 4, nrow = M * T_max))
colnames(M_chains) = c("t", "m", "x", "y")

for (m in 0:(M-1)) {
  gibbs_result = gibbs(rho, T_max, x_0, y_0)

  iter_start = m * T_max + 1
  iter_end = (m+1) * T_max

  M_chains$x[iter_start:iter_end] = gibbs_result$x
  M_chains$y[iter_start:iter_end] = gibbs_result$y
  M_chains$t[iter_start:iter_end] = 1:T_max
  M_chains$m[iter_start:iter_end] = rep(m, T_max)
}
```

$\rho = 0.99,\ T = 100,\ M = 100,\ (X_0, Y_0) = (-10, -10)$

```
rho = 0.99
T_max = 100
M = 100
x_0 = -10
y_0 = -10

M_chains = data.frame(matrix(0, ncol = 4, nrow = M * T_max))
colnames(M_chains) = c("t", "m", "x", "y")
```

```r
for (m in 0:(M-1)) {
  gibbs_result = gibbs(rho, T_max, x_0, y_0)

  iter_start = m * T_max + 1
  iter_end = (m+1) * T_max

  M_chains$x[iter_start:iter_end] = gibbs_result$x
  M_chains$y[iter_start:iter_end] = gibbs_result$y
  M_chains$t[iter_start:iter_end] = 1:T_max
  M_chains$m[iter_start:iter_end] = rep(m, T_max)
}
```

The following output displays the last 6 movements of "person" number 99 using a Gibbs sampler with the parameters above.

```
        t  m           x          y
9995   95 99 -0.91559507 -1.1058019
9996   96 99 -1.20830278 -1.3063715
9997   97 99 -0.96394778 -0.8884588
9998   98 99 -0.07881351 -0.8211467
9999   99 99 -0.86235848 -1.4113724
10000 100 99 -1.47113885 -1.3286050
```

Animation:

```r
library(ggplot2)
library(gganimate)
library(gifski)

p1 <- ggplot(M_chains, aes(x, y)) +
  geom_point(alpha = 0.7, show.legend = FALSE) +
  labs(title = 't = : {frame_time}')+
  transition_time(t) +
  ease_aes('linear')

animate(p1, renderer = gifski_renderer())
```

GIFs are attached in the BruinLearn submission. An animation with $\rho = 0.5$ and $(X_0, Y_0) = (-10, -10)$ would be named

`anim_0.5_n10_n10.gif`

while an animation with $\rho = 0.99$ and $(X_0, Y_0) = (0, 0)$ would be named

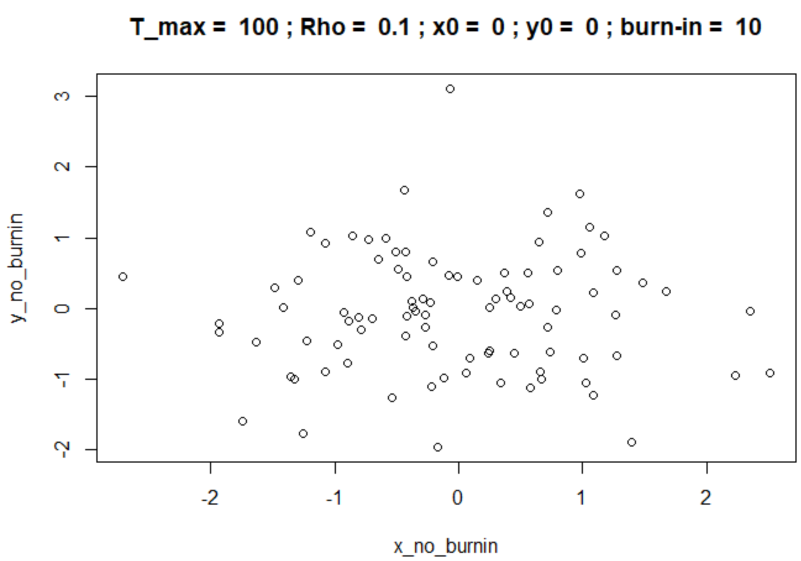
`anim_0.99_0_0.gif`

Burn-in period $B = 10$:

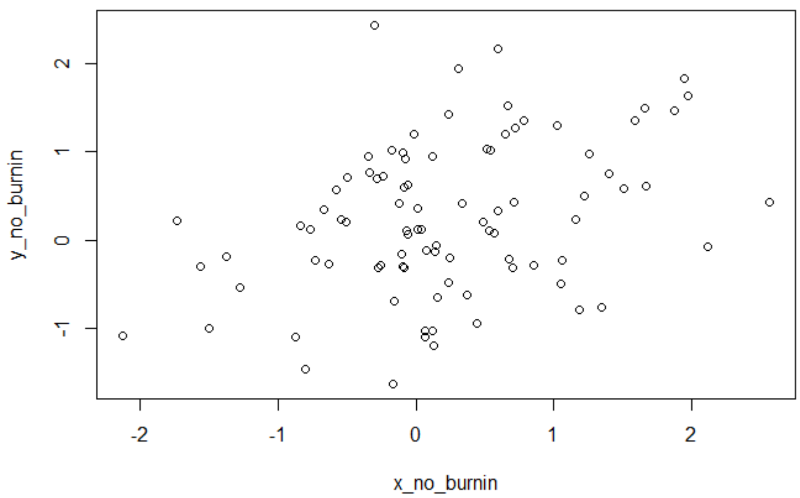
```r
rho = 0.1
T_max = 100
x_0 = 0
y_0 = 0

# remove burn-in
burn_in = 10
gibbs_result = gibbs(rho, T_max, x_0, y_0)
x_no_burnin = gibbs_result$x[(burn_in+1):T_max]
y_no_burnin = gibbs_result$y[(burn_in+1):T_max]
```

```r
plot(x_no_burnin, y_no_burnin, main = paste("T_max = ", T_max, "; Rho = ", rho, ";
    x0 = ", x_0, "; y0 = ", y_0, "; burn-in = ", burn_in ))

rho = 0.5
T_max = 100
x_0 = 0
y_0 = 0

# remove burn-in
burn_in = 10
gibbs_result = gibbs(rho, T_max, x_0, y_0)
x_no_burnin = gibbs_result$x[(burn_in+1):T_max]
y_no_burnin = gibbs_result$y[(burn_in+1):T_max]

plot(x_no_burnin, y_no_burnin, main = paste("T_max = ", T_max, "; Rho = ", rho, ";
    x0 = ", x_0, "; y0 = ", y_0, "; burn-in = ", burn_in ))

rho = 0.9
T_max = 100
x_0 = 0
y_0 = 0

# remove burn-in
burn_in = 10
gibbs_result = gibbs(rho, T_max, x_0, y_0)
x_no_burnin = gibbs_result$x[(burn_in+1):T_max]
y_no_burnin = gibbs_result$y[(burn_in+1):T_max]

plot(x_no_burnin, y_no_burnin, main = paste("T_max = ", T_max, "; Rho = ", rho, ";
    x0 = ", x_0, "; y0 = ", y_0, "; burn-in = ", burn_in ))

rho = 0.99
T_max = 100
x_0 = 0
y_0 = 0

# remove burn-in
burn_in = 10
gibbs_result = gibbs(rho, T_max, x_0, y_0)
x_no_burnin = gibbs_result$x[(burn_in+1):T_max]
y_no_burnin = gibbs_result$y[(burn_in+1):T_max]

plot(x_no_burnin, y_no_burnin, main = paste("T_max = ", T_max, "; Rho = ", rho, ";
    x0 = ", x_0, "; y0 = ", y_0, "; burn-in = ", burn_in ))


rho = 0.1
T_max = 100
x_0 = -10
y_0 = -10

# remove burn-in
```

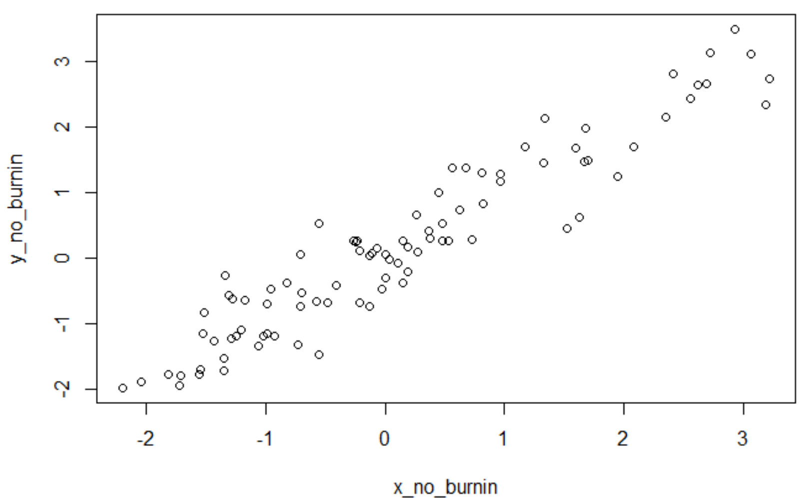
```r
burn_in = 10
gibbs_result = gibbs(rho, T_max, x_0, y_0)
x_no_burnin = gibbs_result$x[(burn_in+1):T_max]
y_no_burnin = gibbs_result$y[(burn_in+1):T_max]

plot(x_no_burnin, y_no_burnin, main = paste("T_max = ", T_max, "; Rho = ", rho, ";
    x0 = ", x_0, "; y0 = ", y_0, "; burn-in = ", burn_in ))

rho = 0.99
T_max = 100
x_0 = -10
y_0 = -10

# remove burn-in
burn_in = 10
gibbs_result = gibbs(rho, T_max, x_0, y_0)
x_no_burnin = gibbs_result$x[(burn_in+1):T_max]
y_no_burnin = gibbs_result$y[(burn_in+1):T_max]

plot(x_no_burnin, y_no_burnin, main = paste("T_max = ", T_max, "; Rho = ", rho, ";
    x0 = ", x_0, "; y0 = ", y_0, "; burn-in = ", burn_in ))
```

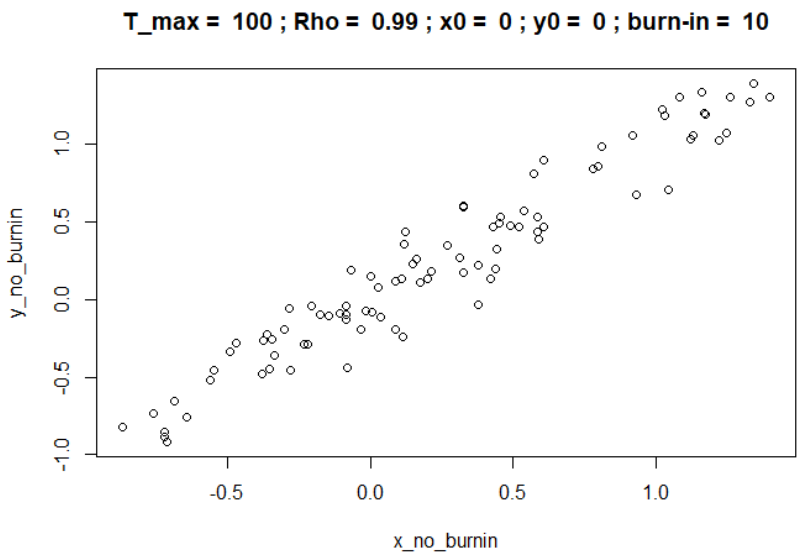


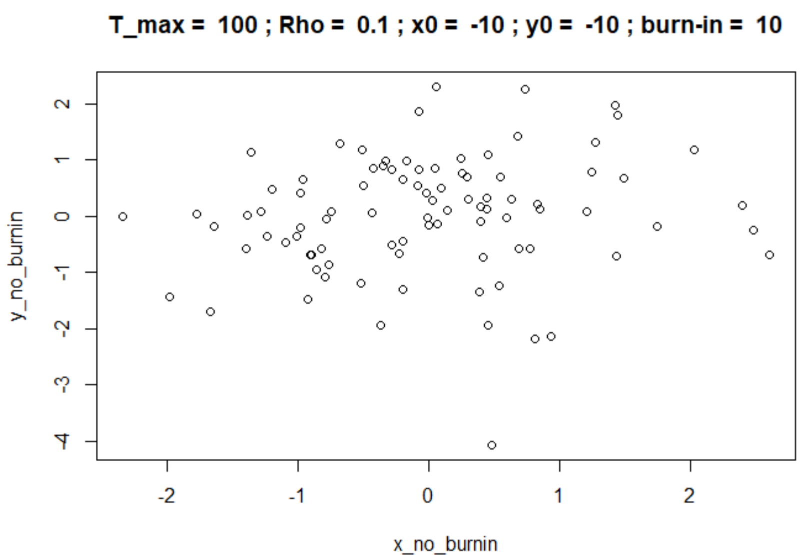
T_max = 100 ; Rho = 0.5 ; x0 = 0 ; y0 = 0 ; burn-in = 10

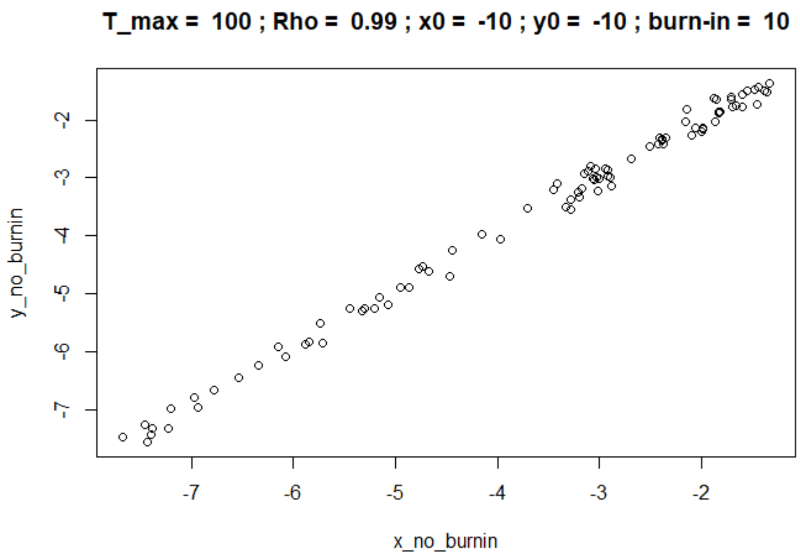


T_max = 100 ; Rho = 0.9 ; x0 = 0 ; y0 = 0 ; burn-in = 10




T_max = 100 ; Rho = 0.99 ; x0 = 0 ; y0 = 0 ; burn-in = 10
y_no_burnin
x_no_burnin
-1.0
-0.5
0.0
0.5
1.0


T_max = 100 ; Rho = 0.1 ; x0 = -10 ; y0 = -10 ; burn-in = 10
y_no_burnin
x_no_burnin
-4
-3
-2
-1
0
1
2


T_max = 100 ; Rho = 0.99 ; x0 = -10 ; y0 = -10 ; burn-in = 10
y_no_burnin
-2
-3
-4
-5
-6
-7
-7
-6
-5
-4
-3
-2
x_no_burnin