Simulating the Luria-Delbrück experiment in `R`

Jun Ishigohoka

```r
T <- 14 # Number of generations
n_0 <- 100 # Number of cells to in a tube
mu <- 1e-3 # Mutation rate
r <- 50 # Number of plates (A) or tubes (B)
n_sample <- 10000 # Number of cells to plate
```

Implementation of the spontaneous mutation model

Let's write steps in R.

Step 1. Simulation of passaging cells in a tube from a flask

Let `parents` a vector of n_0 zeros, representing n_0 wild type cells. For now, we (as in the original paper) assume that there are no standing genetic variation in these cells. Later you are free to relax this assumption and investigate how the presence of standing genetic variation in this step might affect the result.

```r
parents <- rep(0, n_0)
```

Step 2. Growth

Let cells replicate T times and mutate. Try with small T

```r
T_test <- 5
# Loop over T generations
for(t in 1:T_test){
        # Initialise a vector children, into which we put genotype of daughter cells
        children <- c()
        # Loop over parent cells
        # In each iteration, `cell` is the genotype of focal cell
        for(cell in parents){
                # Loop over two daughter cells
                for(i in 1:2){
                        # Bernoulli sampling of muation
                        # Mutation is represented as addition of 1
                        # The genotype of the focal daughter cell is appended
                        children <- c(children, cell + rbinom(1, 1, mu))
                }
        }
        # vector parents is updated
        parents <- children
}

# frequency distribution of cells freq(genotype == x)
table(parents)
```

```
## parents
##    0    1
## 3189   11
```

Entries greater than 1 are converted into 1 because cells whose ancestors experienced at least one muation are resistant.

```r
parents <- as.integer(parents > 0)
table(parents)
```

```
## parents
##    0    1
## 3189   11
```

Initial optimisation of steps 1 and 2

The `for` loop in lines 7-15 in the above code block can be substituted with a single binomial sampling of `n = 2 * length(parents)` ($= n_0 \times 2^T$). We can directly update `parents` instead of making an intermediate vector `children`. And without children naming of parents wouldn't make sense, so let us call the vector `genotypes`.

```r
genotypes <- rep(0, n_0)

for(t in 1:T){
    genotypes <- rep(genotypes, 2) + rbinom(n_0 * 2^t, 1,  mu)
}

genotypes <- as.integer(genotypes > 0)
table(genotypes)
```

```
## genotypes
##       0       1
## 1610230   28170
```

The number of resistant cells after T generations is the sum of the vector.

```r
sum(genotypes)
```

```
## [1] 28170
```

Make a function `sim_tube` for steps 1 and 2

Let's make a function for steps 1 and 2.

```r
sim_tube <- function(n_gens = T, mut_rate = mu, ncells_init = n_0){
    genotypes <- rep(0, ncells_init)
    for(t in 1:n_gens){
        genotypes <- rep(genotypes, 2) + rbinom(ncells_init * 2^t, 1,  mut_rate)
    }
    genotypes <- as.integer(genotypes > 0)
    return(genotypes)
}
```

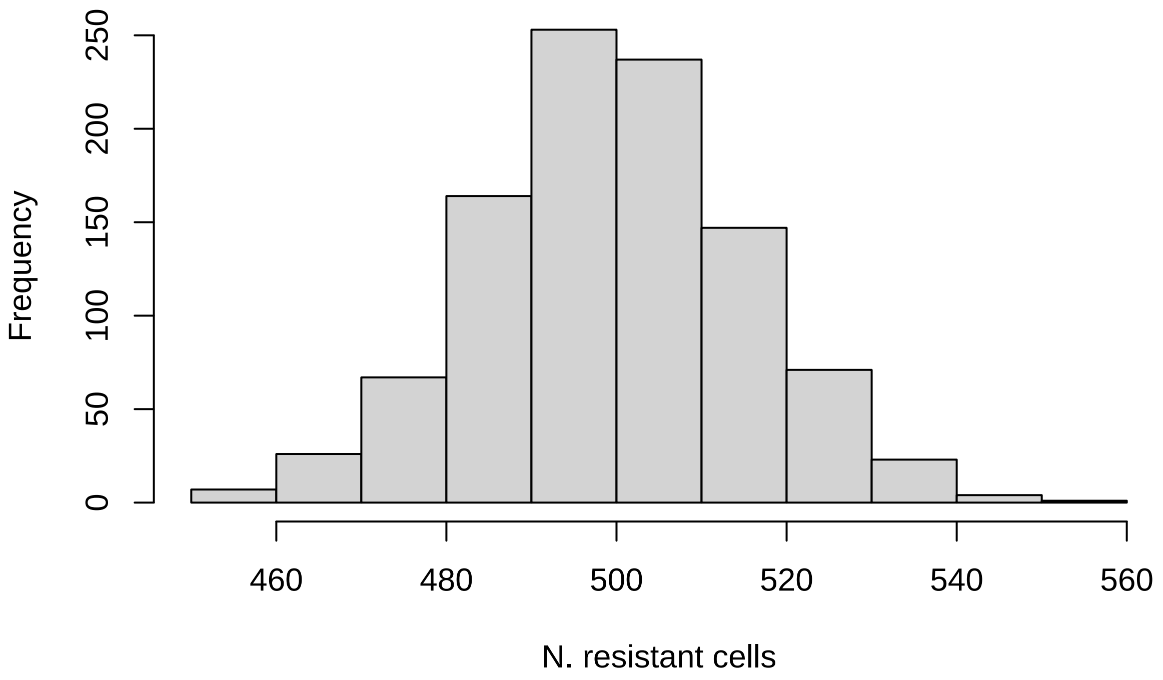
Let's simulate a tube and count the number of resistant cells

```r
tube <- sim_tube(T, mu, n_0)
sum(tube)
```

```
## [1] 26396
```

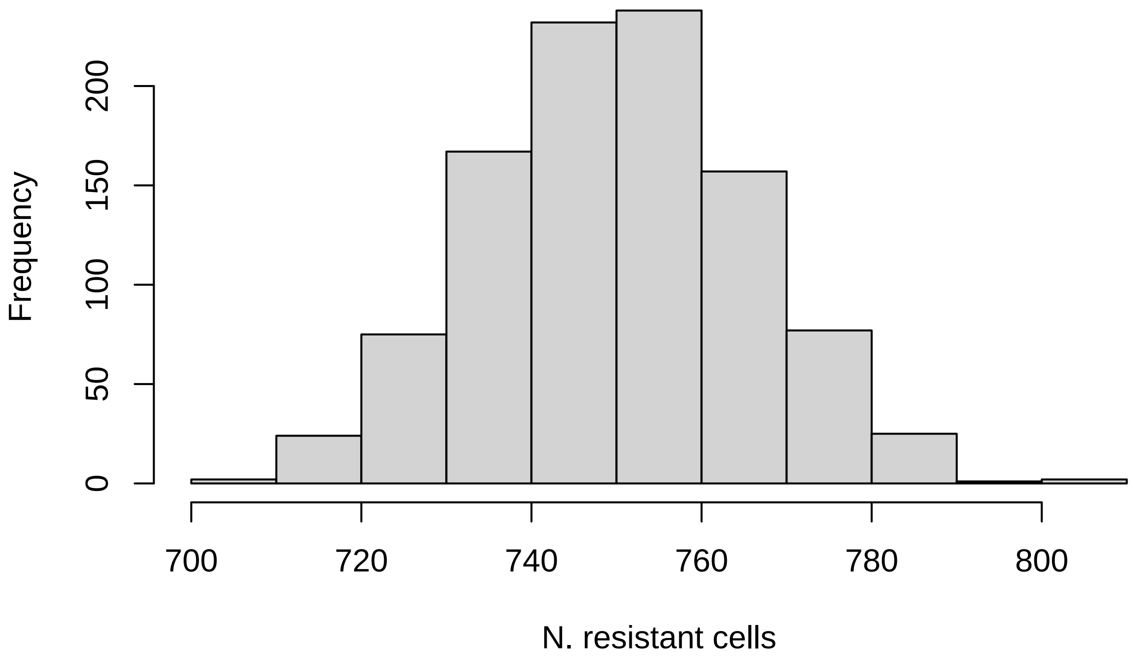
Let's see if this function actually does a right job. If we let $T = 1$, then the number of resistant cells should be around $2n_0\mu$. So, when $\mu = 0.5$ and $n_0 = 500$, there should be around 500 resistant cells. Let's simulate it 1,000 times and check the distribution of the number of resistant cells.

```r
hist(
        sapply(1:1000,
               function(x){
                       return(sum(sim_tube(1, 0.5, 500)))
               }
        ),
     xlab = "N. resistant cells",
     main = ""
)
```



If we let $T = 2$, then the number of resistant cells should be around $n_0 \times 2^2 \times (1 - \mu^2)$. So, when $\mu = 0.5$ and $n_0 = 250$, there should be around 750 resistant cells. Let's simulate it 1,000 times and check the distribution of the number of resistant cells.

```r
hist(
        sapply(1:1000,
               function(x){
                       return(sum(sim_tube(2, 0.5, 250)))
               }
        ),
     xlab = "N. resistant cells",
     main = ""
)
```



The simulator `sim_tube` seems to be working properly.

Simulate resistant cells in experiment A

In experiment A, n_0 cells are let grow over T generations (to $n_0 \times 2^T$ cells). The cells are plated to r plates from this tube. So, we will simulate one tube before plating. The genotypes of cells are recorded in a vector `tube_a` of length $n_0 \times 2^T$.

```
tube_a <- sim_tube(n_gens = T, mut_rate = mu, ncells_init = n_0)
head(tube_a)
```

```
## [1] 0 0 0 0 0 0
```

```
length(tube_a)
```

```
## [1] 1638400
```

Simulate resistant cells in experiment B

In experiment B, there are r tubes, each of which accommodates n_0 cells and let them grow over T generations. So, we will simulate r tubes. The genotypes are recorded in a matrix `tubes_b` with r rows and $n_0 \times 2^T$ columns.

```
tubes_b <- t(
             sapply(1:r,
                    function(x){
                            sim_tube(n_gens = T, mut_rate = mu, ncells_init = n_0)
                    }
             )
)
```

Step 3. Simulation of plating

In experiment A, we plate n_{sample} cells from 1 tube to r plates. In experiment B, we plate n_{sample} cells from each of r tubes to each of r plates, respectively.

Experiment A

In experiment A, given a vector of genotypes of cells in a tube, we want a matrix with r rows and n_{sample} columns representing genotypes of cells. One can shuffle the input vector representing genotypes of cells in a tube and take the first n_sample cells for the first plate, second n_sample cells for the second plate, and so on.

Let's start with plating 2 cells from 6 cells in a tube. Genotypes of the plated 2 cells can be stored in a vector.

```r
n_plates <- 1
n_cells_plate <- 2

tube_test <- c(1, 1, 0, 0, 0, 0)
# shuffle
tube_test <- sample(tube)
plate_test <-tube_test[1:n_cells_plate]
```

Then let's try plating 2 cells from 10 cells in each of 3 plates. The output genotypes are stored in a matrix with 3 rows and 2 columns.

```r
n_plates <- 3
n_cells_plate <- 2

tube_test <- c(1, 1, 0, 0, 0, 0, 1, 0, 1, 0)
# shuffle
tube_test <- sample(tube)
# A vector to store cells to plate in n_plates plates
plates_test <-tube_test[1:(n_cells_plate * n_plates)]
# Make it into a matrix
plates_test <- matrix(plates_test, nrow = n_plates, byrow = T)
```

Based on the above exercise we can write a function `sim_plate`.

```r
sim_plate <- function(tube, n_plates, ncells_plate){
        tube_shuf <- sample(tube)
        plates <-tube_shuf[1:(ncells_plate * n_plates)]
        plates <- matrix(plates, nrow = n_plates, byrow = T)
        return(plates)
}
```

By running this function on our simulated `tube_a`, we can obtain `plates_a`, a matrix of genotypes.

```r
plates_a <- sim_plate(tube = tube_a, n_plates = r, ncells_plate = n_sample)
```

Experiment B

In experiment B, given a matrix with r rows representing genotypes of cells in r tube, we want a matrix with r rows and n_{sample} columns representing genotypes of cells. Conveniently, for each tube we can use `sim_plate` function with `n_plates = 1`.

For example, for the first tube,

```r
plate_b_1 <- sim_plate(tube = tubes_b[1,], n_plates = 1, ncells_plate = n_sample)
```

We can apply this to each row of matrix `tubes_b` (and transpose the output) to obtain a matrix of genotypes `plates_b`.

```r
plates_b <- t(
              sapply(1:r,
                     function(x){
                             return(sim_plate(tube = tubes_b[x,],
                                                  n_plates = 1,
                                                  ncells_plate = n_sample))
                      }
               )
)
```

Step 4. Compute mean and variance

Both `plates_a` and `plates_b` are matrices whose row corresponds to a replicate plate. The entry of the matrix is genotype, where 0 is wild type and 1 is resistant mutant. Therefore the sum of each row is the number of resistant colonies in each plate. We can summarise the results in two values: mean and variance of the number of colonies per plate.

```r
result <- c(mean_a = mean(rowSums(plates_a)),
            mean_b = mean(rowSums(plates_b)),
            var_a = var(rowSums(plates_a)),
            var_b = var(rowSums(plates_b)))
result
```

```
##    mean_a    mean_b     var_a     var_b
##  171.1800  142.7000  190.9261 1263.0714
```

Implementation of induced mutation model

In both experiments A and B, cells should mutate at the same rate (mutation rate) after plating. So, this is binomial sampling with rate parameter of μ and size parameter of n_sample

```r
sim_ld_ind <- function(n_plates, n_sample, mu){
        plates_a <- rbinom(n_plates, n_sample, mu)
        plates_b <- rbinom(n_plates, n_sample, mu)
        res <- c(mean_a = mean(plates_a),
             mean_b = mean(plates_b),
             var_a = var(plates_a),
             var_b = var(plates_b)
        )
        return(res)
}
```

```r
sim_ld_ind(n_plates = r, n_sample = n_sample, mu = mu)
```

```
##    mean_a    mean_b     var_a     var_b
##  9.920000  9.480000  7.585306 12.458776
```

We can run it multiple times using `sapply`

```r
t(
  sapply(1:5,
         function(x){
                return(sim_ld_ind(r, n_sample, mu))
         }
  )
)
```

```
##      mean_a mean_b     var_a     var_b
## [1,]   9.30  10.02  8.091837 12.958776
## [2,]   9.56  10.02  7.394286  9.775102
## [3,]  10.00  10.12 13.510204  8.720000
## [4,]   8.98   9.94  8.264898  9.567755
## [5,]  10.20  10.44 10.693878 10.169796
```

Questions

Let's address questions with our simulators. Note that our simulators are not very well optimised, and some questions may be still iplausible (e.g. large T, large μ). We will further optimise the simulators in the next section, but you are already ready to play around with what you have to get an intuition of the experiment.

1. aaa
2. aaa
3. aaa

Further optimisations of the spontaneous mutation simulator

In the above simulators `sim_tube` and `sim_plate`, we recorded the genotypes of all cells in tubes and plates until the final step. This is very inefficient for our purpose. The information of index of the genotype vector is not used: Even if someone shuffled our vector at any step of our simulation, we would not suffer. The only information we need is actually the numbers of wild type cells adn resistant cells, instead of genotype of millions of cells.

Below, we will try to improve the scripts to make it more scalable and faster.

Step 1. Simulation of passaging cells in a tube from a flask

Instead of recording genotypes of n_0 cells, we can have two objects to keep the number of wild type and resistant cells.

```r
n_wt <- n_0
n_re <- 0
```

Step 2. Growth

In each generation, daughter cells of resistant cells are all resistant. Some of daughter cells of wild type cells mutate to resistant, and the number of such cells follow a binomial distribution with the size parameter of 2 x n. wild type parent cells.

```r
for(t in 1:T){
        n_re <- 2 * n_re + rbinom(1, 2 * n_wt, mu)
        n_wt <- n_0 * 2^t - n_re
}

print(c(n_wt = n_wt, n_re = n_re))
```

```
##    n_wt    n_re
## 1619291   19109
```

Make a function `sim_tube_count`

```r
sim_tube_count <- function(n_gens = T, mut_rate = mu, ncells_init = n_0, ncells_res_init = 0){
        n_wt <- ncells_init - ncells_res_init
        n_re <- ncells_res_init
        for(t in 1:n_gens){
                n_re <- 2 * n_re + rbinom(1, 2 * n_wt, mut_rate)
                n_wt <- ncells_init * 2^t - n_re
        }
        return(n_re)
}
```

Note that I added an optional argument `ncells_res_init`, the number of resistant cells in the initial passage, reflecting standing variation (default: 0).

Now, this `sim_tube_count` returns an integer, the number of resistant cells. By using this function, we can simulate the number of resistant cells in experiment A.

```
tube_count_a <- sim_tube_count(n_gens = T, mut_rate = mu, ncells_init = n_0)
```

By applying this function, we can simulate the number of resistant cells in experiment B. The counts are stored in a vector `tubes_count_b`.

```
tubes_count_b <- sapply(1:r,
                        function(x){
                                sim_tube_count(n_gens = T, mut_rate = mu, ncells_init = n_0)
                        }
)
```

Step 3. Simulation of plating

Before plating, we have `n_re` resistant cells and `n_wt` wild type cells. In experiment A, we sample `n_sample` cells without replacement sequentially over r times. Each plating is equivalent to taking some balls from a box with some red and blue balls without replacement, and the number of balls with one colour follows a hypergeometric distribution.

```
sim_plate_count <- function(n_re, n_wt, n_plates, n_sample){
        # Number of resistant cells in plates
        plates_count <- c()
        # Loop over n_plates plates
        for(i in 1:n_plates){
                plates_count <- c(plates_count, rhyper(1, n_re, n_wt, n_sample))
                n_re <- n_re - plates_count[i]
                n_wt <- n_wt - (n_sample - plates_count[i])
        }
        return(plates_count)
}
```

For experiment A, we use this function to obtain the number of resistant cells on r plates.

```
plates_count_a <- sim_plate_count(n_re = tube_count_a,
                                  n_wt = n_0 * 2^T - tube_count_a,
                                  n_plates = r,
                                  n_sample = n_sample
)
```

For experiment B, we can apply this function with `n_plates = 1` over r times.

```
plates_count_b <- sapply(tubes_count_b,
                         function(x){
                                 sim_plate_count(n_re = x,
                                                 n_wt = n_0 * 2^T - x,
                                                 n_plates = 1,
                                                 n_sample = n_sample
                                 )
```

```r
                         }
)
```

Step 4. Compute mean and variance

```r
result <- c(mean_a = mean(plates_count_a),
            mean_b = mean(plates_count_b),
            var_a = var(plates_count_a),
            var_b = var(plates_count_b))

result
```

```
##    mean_a    mean_b     var_a     var_b
##  151.6200  139.6000  109.7506 1255.8367
```

Make a function to do all. . .

```r
sim_ld_spo <- function(n_gens, mut_rate , ncells_init , n_sample , n_plates,  ncells_res_init = 0){
        tube_count_a <- sim_tube_count(n_gens = n_gens, mut_rate = mut_rate, ncells_init = ncells_init)
        tubes_count_b <- sapply(1:n_plates,
                                function(x){
                                        sim_tube_count(n_gens = n_gens, mut_rate = mut_rate, ncells_ini
                                }
        )
        plates_count_a <- sim_plate_count(n_re = tube_count_a,
                                          n_wt = ncells_init * 2^n_gens - tube_count_a,
                                          n_plates = n_plates,
                                          n_sample = n_sample
        )
        plates_count_b <- sapply(tubes_count_b,
                                 function(x){
                                         sim_plate_count(n_re = x,
                                                         n_wt = ncells_init * 2^n_gens - x,
                                                         n_plates = 1,
                                                         n_sample = n_sample
                                         )
                                 }
        )
        result <- c(mean_a = mean(plates_count_a),
                    mean_b = mean(plates_count_b),
                    var_a = var(plates_count_a),
                    var_b = var(plates_count_b))
        return(result)
}
```

We can run this multiple times using `sapply`

```r
t(sapply(1:5,
       function(x){
        sim_ld_spo(n_gens = 30,
               mut_rate = 1e-4,
               ncells_init = n_0,
               n_sample = n_sample,
               n_plates = r,
                ncells_res_init = 0)
        }
)
)
```

```
##      mean_a mean_b    var_a     var_b
## [1,]  27.54  27.92 26.94735  72.40163
## [2,]  29.96  29.74 25.10041 137.42082
## [3,]  24.04  30.78 30.40653 145.23633
## [4,]  28.36  28.44 27.58204  94.82286
## [5,]  26.18  29.44 19.17102 147.51673
```

Solutions to some of the questions

```r
T <- 15
mu <- 1e-4


#for(i in 1:length(seq(0,1,0.2))){
#        p <- seq(0,1,0.2)[i]
#        t(sapply(1:10,
#               function(x){
#                sim_ld_spo(n_gens = T,
#                       mut_rate = mu,
#                       ncells_init = n_0,
#                       n_sample = (n_0 * 2^T)/100,
#                       n_plates = r,
#                       ncells_res_init = n_0 * p)
#               }
#        )
#        )
#}
```