Virus Transmission Bottleneck Size Estimation Using Patterns of Clonal Passenger Mutations

Research Project in Koelle Lab

Yike Teresa Shi

2023-08-07

Contents

1. Introduction **3**

1.1. Overall offspring distribution 3

1.2. Number of mutations on a viral particle 3

1.3. Wild-type and mutant offspring distribution 3

2. Offspring Distribution in a Single Generation **4**

3. Probability of Lineage Establishment **6**

3.1. Probability of wt extinction with initial population size $N = 1$ 6

3.2. Probability of wt establishment with arbiturary initial population size N 6

3.3. Examine how R_0 and μ affect probability of extinction 7

4. Wild-type Final Size Distribution **9**

4.1. Final size distribution with initial population size $N = 1$ 9

4.2. Extend to arbiturary initial population size n using recursive function 10

4.3. Normalized final size distribution with initial population size N 12

5. Mutant Offspring Generated Distribution **14**

5.1. Mutant offspring distribution with initial population size $N = 1$ 14

5.2. Extend to arbiturary initial population size N 15

6. Established Mutant Offspring Distribution **17**

7. Clonal Mutation Distribution **19**

7.1. P_X, P_0, and P_{1^+} 19

7.2. Resolving P_{1^+} 21

8. Applications **25**

8.1. Initial population distribution as Poisson distribution 25

8.2. Initial population distribution as negative binomial distribution 32

8.3. Initial population distribution With only $N = 1$ or ∞ 36

9. Source of Fixed Allele **38**

References **40**

1. Introduction

1.1. Overall offspring distribution

We model the dynamics of viral reproduction with a special case of negative binomial distribution $NB(r = 1, p)$, which is in fact a geometric distribution, with mean offspring generated to be R_0. All mutations considered in the model are passenger mutations and thus do not affect the fitness of the viral particle. The negative binomial distribution allows us to model for heterogeneity in different viral populations and is consistent with the SIR model.

For $0 \leq R_0 < 1$, the overall birth rate is less than the death rate and the probability of extinction is $P_x = 1$. Consequently, the probability for an establishment to occur (with the number of clonal mutations being 0 or higher) is $P_{0^+} = 0$. We will focus on the situation when $R_0 > 1$ and the lineage has the probability to establish.

1.2. Number of mutations on a viral particle

We model the number of mutations that may occur on a particular viral particle to follow a Poisson distribution with mean value of μ. The probability of a viral particle to be wild-type (i.e. bearing 0 mutation) is thus given by

$$P(mutation = 0; \mu) = \frac{\mu^0 e^{-\mu}}{0!} = e^{-\mu}$$

Thus, the mean number of wild-type offspring is thus given by $R_0 e^{-\mu}$, and the mean number of mutant offspring is thus $R_0(1 - e^{-\mu})$.

1.3. Wild-type and mutant offspring distribution

The overall offspring distribution can be divided into two distributions: wild-type offspring distribution and mutant offspring distribution, each follows a negative binomial distribution with same p.

When two negative binomial distribution with same p are added together, the parameters for the new distribution has the following characteristic:

$$NB(r, p) + NB(s, p) = NB(r + s, p)$$

The mean of a negative binomial distribution is given by $mean = \frac{r(1-p)}{p}$.

For the overall offspring distribution, $mean = \frac{1-p}{p} = R_0$.

For the wild-type offspring distribution, $mean = \frac{r(1-p)}{p} = R_0 e^{-\mu}$, so $r = e^{-\mu}$.

For the mutant offspring distribution, $mean = \frac{r(1-p)}{p} = R_0(1 - e^{-\mu})$, so $r = 1 - e^{-\mu}$.

2. Offspring Distribution in a Single Generation

The offspring distribution of a viral particle follows a negative binomial distribution $NB(k, p)$ in which k is the dispersion parameter ($k = 1$ for overall distribution since we model the overall offspring as geometric distribution).

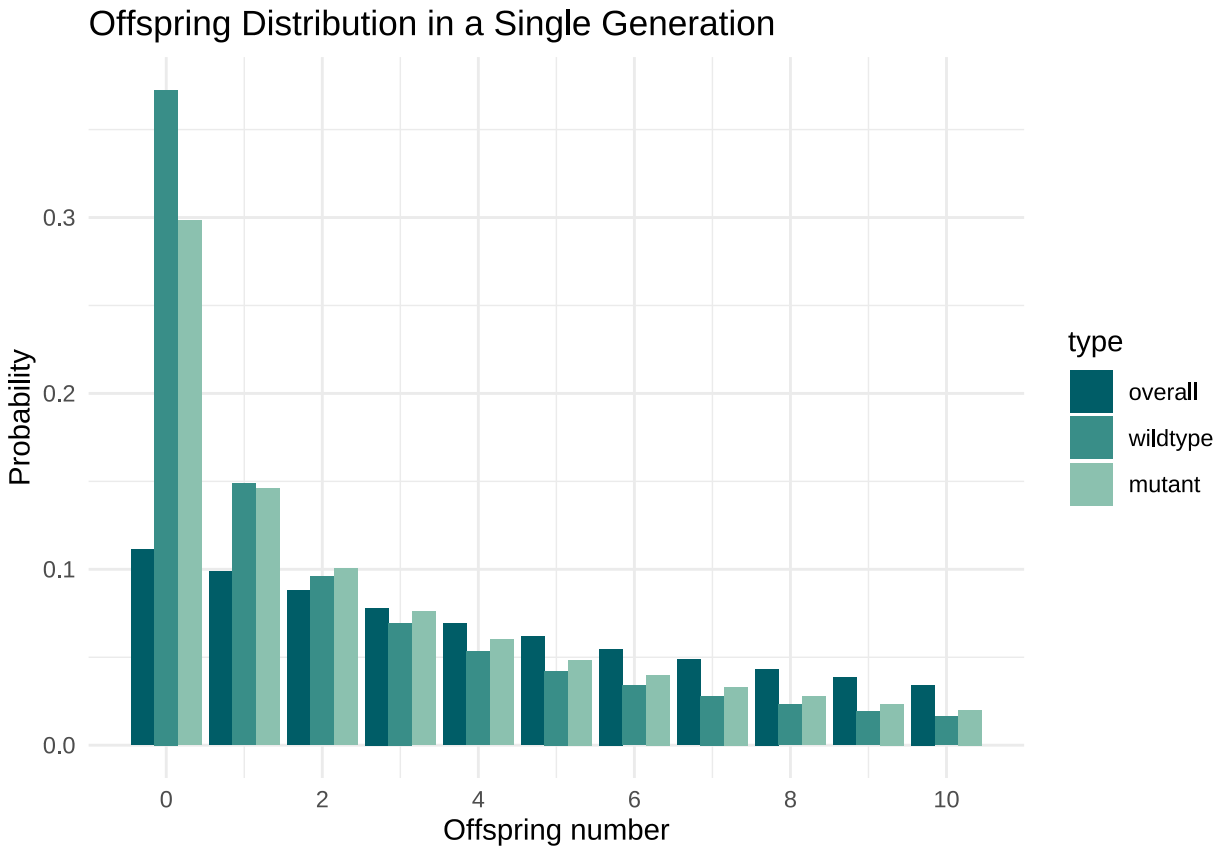
Overall offspring distribution: $NB(1, \frac{1}{R_0+1})$;

wild-type offspring distribution: $NB(e^{-\mu}, \frac{1}{R_0+1})$;

Mutant offspring distribution: $NB(1 - e^{-\mu}, \frac{1}{R_0+1})$.

Since the addition of two negative binomial distribution $NB(r, p)$ and $NB(s, p)$ is $NB(r+s, p)$, the wild-type and mutant offspring distribution will add up to the geometric overall distribution.

```r
# maxOff arguement defines the maximum number of offspring you want to calculate
pmf.alloff <- function(R0, mu, maxOff) {
  k_overall <- 1
  k_wt <- exp(-mu)
  k_mu <- (1 - exp(-mu))
  p <- 1 / (R0 + 1)
  pmf <- data.frame(
    x = 0:maxOff,
    overall = dnbinom(0:maxOff, k_overall, p),
    wildtype = dnbinom(0:maxOff, k_wt, p),
    mutant = dnbinom(0:maxOff, k_mu, p)
  )
  pmf <- melt(pmf, id.vars = "x")
  names(pmf) <- c("size", "type", "prob")
  return(pmf)
}

# sample usage and plot
three_pmf <- pmf.alloff(8, 0.8, 10)
```


Offspring Distribution in a Single Generation
Probability
0.0
0.1
0.2
0.3
0
2
4
6
8
10
Offspring number
type
overall
wildtype
mutant

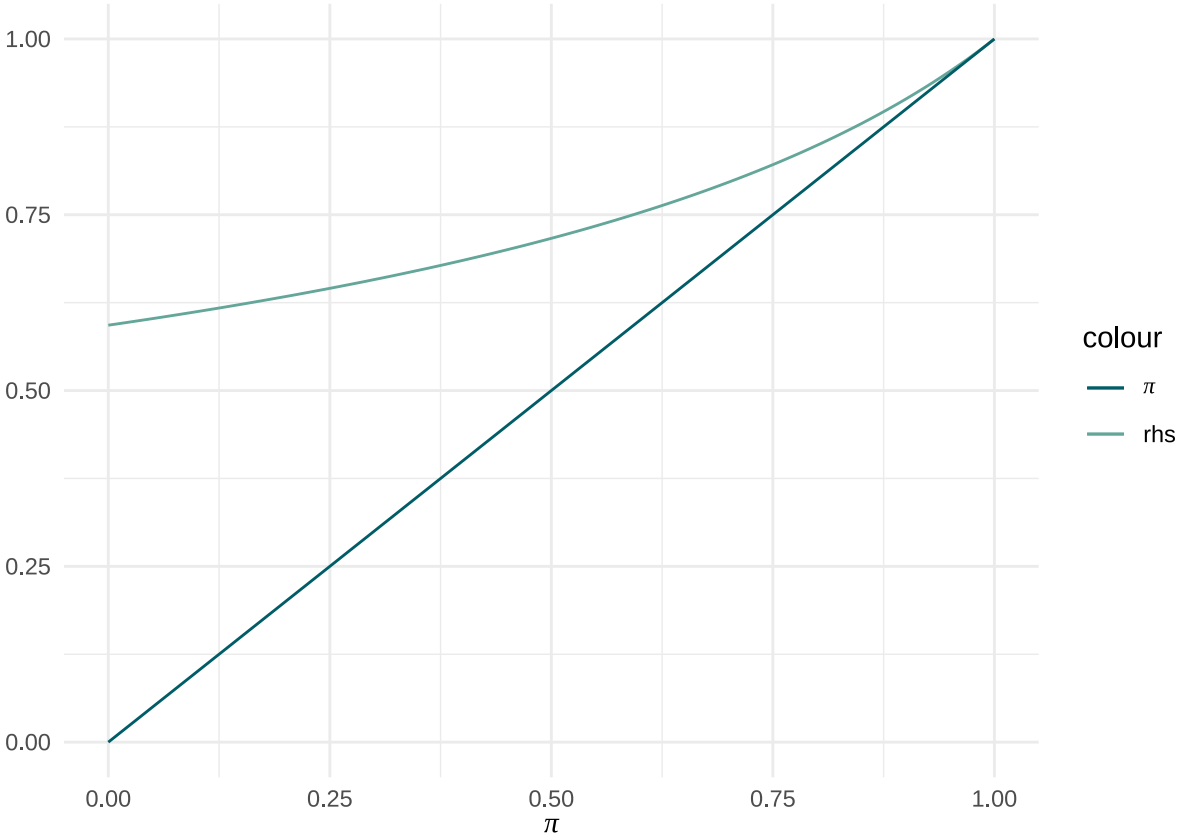
3. Probability of Lineage Establishment

3.1. Probability of wt extinction with initial population size $N = 1$

The probability of wild-type lineage going extinct in a population starting with 1 viral particle is provided in Nishiura et al. (2012).

$$P_{wt,ext,1} = \pi = \frac{1}{(1 + \frac{R_0 e^{-\mu}(1-\pi)}{e^{-\mu}})^{e^{-\mu}}}$$

Plot the left hand side and right hand side respectively.



The intersection of the two lines reveals the probability of wild-type lineage going extinct, which can be calculated by the following:

```
# use (0,0.9) interval instead of (0,1) because using the latter will always return 1
uniroot(function(p) rhs(p, 1.6, 0.4) - lhs(p), c(0, 0.9), extendInt = "yes")$root
```

```
## [1] 0.9455103
```

3.2. Probability of wt establishment with arbiturary initial population size N

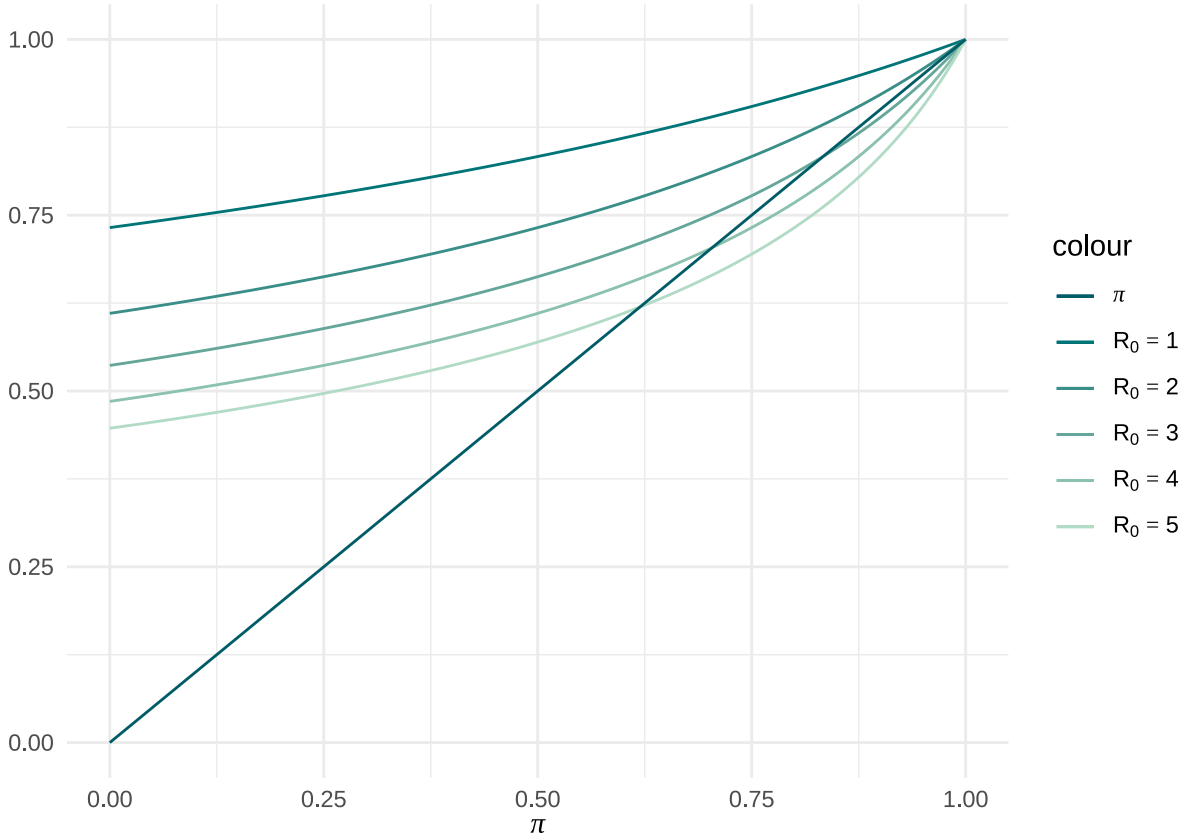
If there are multiple initial viral particles, the probability that at least one wild-type lineage establishes is given by 1 minus the probability that all lineages goes extinct, which is $P_{wt,est,n} = 1 - (P_{wt,ext,1})^n$.

```r
P_wt_est <- function(n, R0, mu) {
  lhs <- function(p) {p}
  rhs <- function(p, R0, mu) {
    rhs <- 1 / (1 + (R0 * exp(-mu) * (1 - p)) / (exp(-mu)))^exp(-mu)
    return(rhs)
  }
  uni <- try(uniroot(function(p) rhs(p, R0, mu) - lhs(p), c(0, 0.9),
                     extendInt = "yes"), silent = TRUE)
  uni <- if (inherits(uni, "try-error")) 1 else uni$root
  if (uni < 1 & uni > 0) {
    ext_1 <- uni
  } else {
    ext_1 <- 1
  }
  est_n <- 1 - ext_1^n
  return(est_n)
}

# sample usage
P_wt_est(3, 8, 0.8)
```

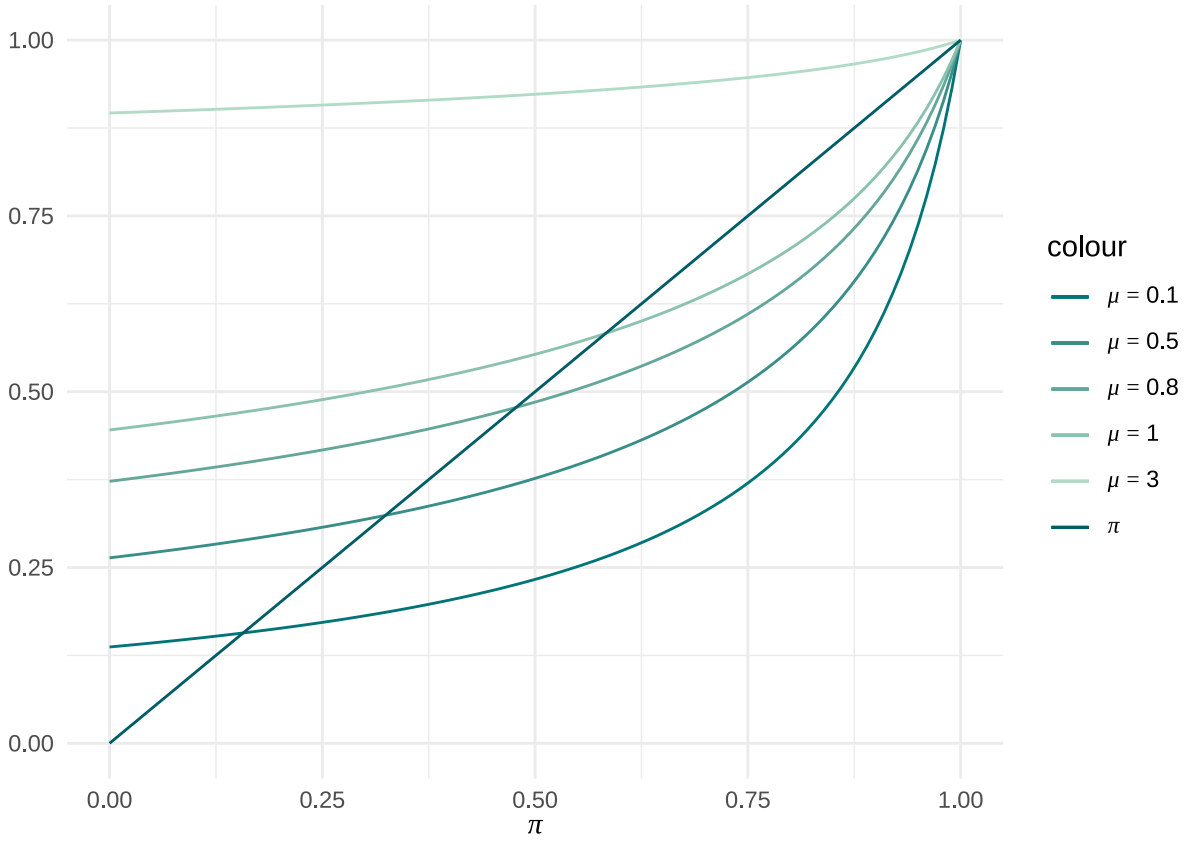
```
## [1] 0.8910643
```

3.3. Examine how R_0 and μ affect probability of extinction

We can examine the effect of increasing R_0 and μ respectively on the probability of extinction.



From the plot we can observe the trend that as R_0 rises, the probability of wild-type lineage going extinction (represented by the intersection point) is lower, which is consistent with our intuition.



From the plot we can observe the trend that as μ rises (higher mean mutation rate per viral particle), the probability of wild-type lineage going extinction (represented by the intersection point) is higher, which is consistent with our intuition.

4. Wild-type Final Size Distribution

4.1. Final size distribution with initial population size $N = 1$

The final size distribution formula is provided by Nishiura et al. (2012).

When considering an initial population size of $N = 1$, in an offspring distribution following $NB(k, \frac{1}{R_0+1})$, the probability of getting a final size of $y = 1$ is

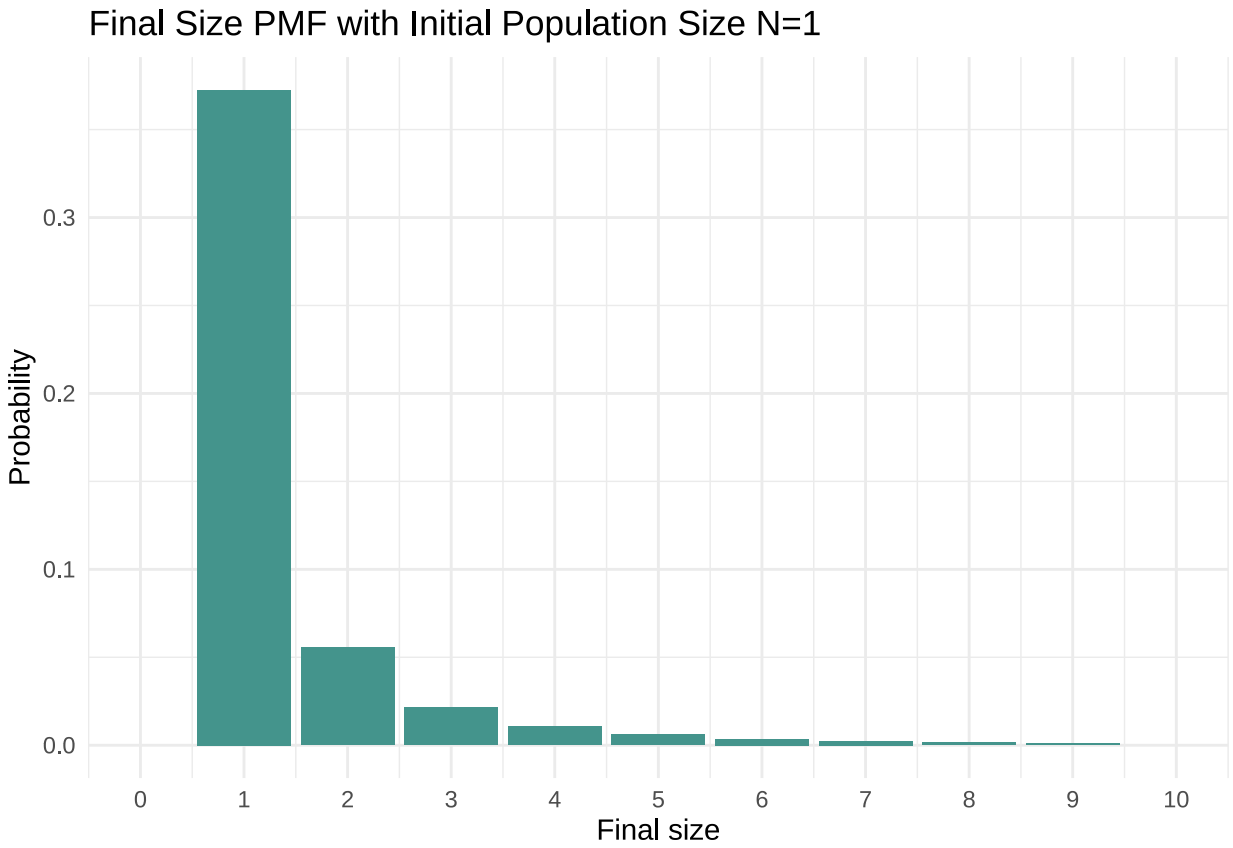
$$p_1 = g'(0) = \frac{1}{(1 + \frac{R_0}{k})^k}$$

and for $y \geq 2$ is

$$p_y = \frac{1}{y!} g^y(0) = \frac{\Pi_{j=0}^{y-2}(\frac{j}{k} + y)}{y!} (\frac{k}{R_0 + k})^{ky} (\frac{R_0 k}{R_0 + k})^{y-1}$$

We can plot out the final size distribution for **wild-type lineage** following $NB(e^{-\mu}, \frac{1}{R_0+1})$.

```
FS <- function(y, R0, mu) {
  k <- exp(-mu)
  meanR0 <- R0 * k
  if (y == 1) {
    prob <- 1 / (1 + meanR0 / k)^k
  } else if (y > 1) {
    prod <- 1
    for (j in 0:(y - 2)) {
      prod <- prod * (j / k + y)
    }
    prob <- prod/factorial(y) * (k/(meanR0 + k))^(k*y) * (meanR0*k/(meanR0+k))^(y-1)
  } else {
    prob <- 0
  }
  return(prob)
}

# store pmf in a data frame
# maxFS arguement defines the maximum number of final size you want to calculate
pmf.FS <- function(R0, mu, maxFS) {
  prob <- rep(0, maxFS)
  for (i in 1:maxFS) {
    prob[i] <- FS(i, R0, mu)
  }
  names(prob) <- c(1:maxFS)
  df <- data.frame(FinalSize = names(prob), prob)
  df$FinalSize <- as.numeric(df$FinalSize)
  return(df)
}

# sample usage and plot
df_pmfFS1 <- pmf.FS(8, 0.8, 50)
```



Since if the lineage establishes, the final size would be infinite, the sum of probabilities with a certain final size should equal to the probability that the lineage goes extinct. That is to say, if we add p_y together, we should get the probability of extinction $P_{wt,ext,1} = \pi = \Sigma p_y$.

```
# sum p_y together
sum(df_pmfFS1$prob[1:50])
```

```
## [1] 0.4775946
```

```
# compare to the probability of extinction we calculated earlier
1 - P_wt_est(1, 8, 0.8)
```

```
## [1] 0.4775917
```

4.2. Extend to arbiturary initial population size n using recursive function

The probability of getting a final size of y from a initial population size N ($P(Y_N = y)$) can be regarded as the probabilities of getting a final size of 1 from the n^{th} particle and a final size of $y-1$ from the previous $n-1$ particles, plus the probability of getting a final size of 2 from the n^{th} particle and a final size of $y-2$ from the previous $n-1$ particles, etc. We can thus express the probability as

$$P(Y_N = y) = \sum_{i=1}^{y-n+1} P(Y_1 = i) \times P(Y_{n-1} = y - i)$$

```r
FSn <- function(n, y, R0, mu) {
  prob <- 0
  if (y >= n) {
    if (n == 1) {
      prob <- FS(y, R0, mu)
    } else if (n > 1) {
      for (i in 1:(y - n + 1)) {
        prob <- prob + FS(i, R0, mu) * FSn(n - 1, y - i, R0, mu)
      }
    }
  }
  return(prob)
}

# store pmf in a data frame
pmf.FSn_noNorm <- function(n, R0, mu, maxFS) {
  df_n1 <- pmf.FS(R0, mu, maxFS)[1:maxFS, ]
  prob <- rep(0, maxFS)
  if (n == 1) {
    df <- df_n1
  } else if (n > 1) {
    df_n_one_less <- pmf.FSn_noNorm(n - 1, R0, mu, maxFS)
    for (i in (1:(maxFS - n))) {
      for (j in ((n - 1):(maxFS - i))) {
        prob[i + j] <- prob[i + j] + df_n1[i, 2] * df_n_one_less[j, 2]
      }
    }
    names(prob) <- seq(1, maxFS)
    df <- data.frame(FinalSize = names(prob), prob)
  }
  l <- nrow(df)
  sum_prob <- sum(df$prob)
  if (sum_prob < 1) {
    df[l + 1, 2] <- (1 - sum_prob)
  }
  return(df)
}
```

Check if the summation equals to the probability of extinction.

```r
# sum p_y together
sum(pmf.FSn_noNorm(3, 8, 0.8, 50)$prob[1:50])
```

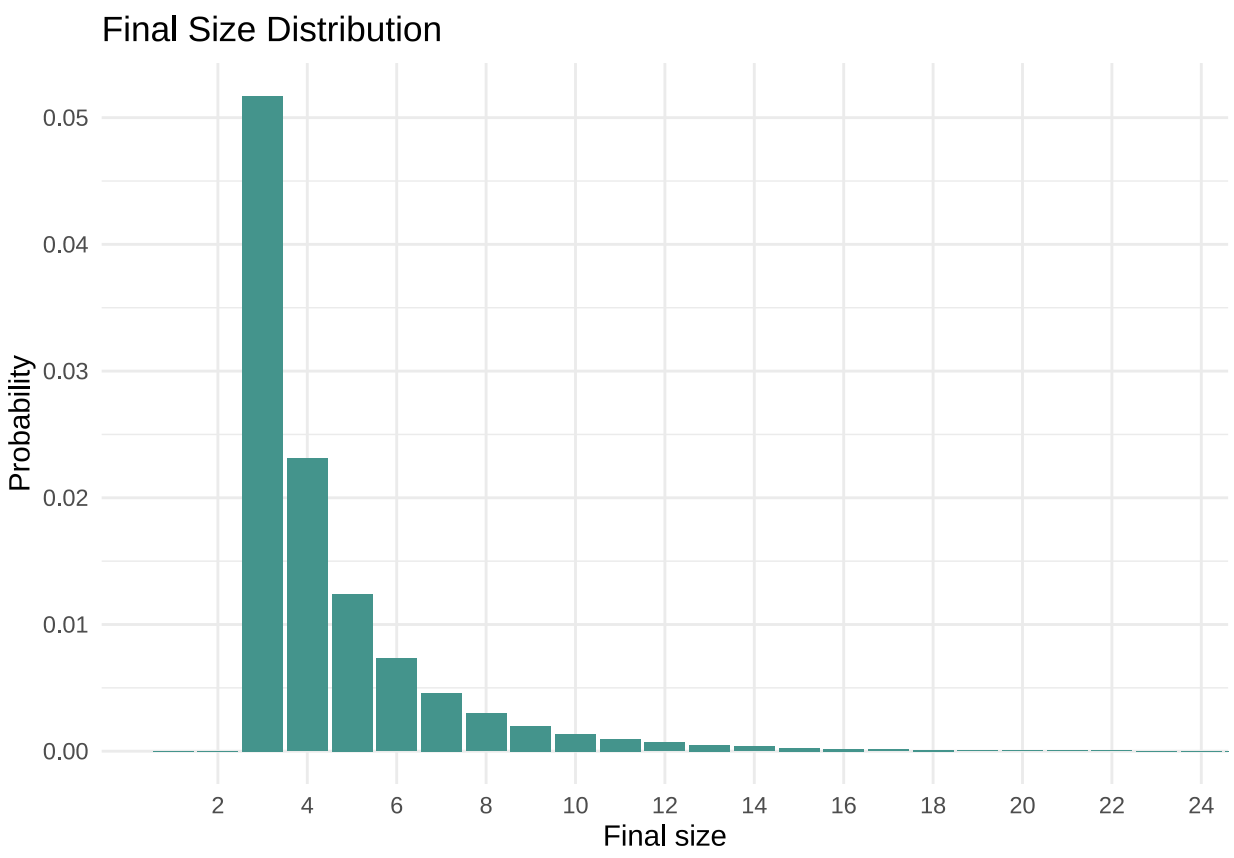
```
## [1] 0.1089377
```

```r
# compare to the probability of extinction we calculated earlier
1 - P_wt_est(3, 8, 0.8)
```

```
## [1] 0.1089357
```

We can now plot out the final size distribution for wild-type lineage following $NB(e^{-\mu}, \frac{1}{R_0+1})$ with any initial population size.

```r
# sample usage and plot
df_FS <- pmf.FSn_noNorm(3, 8, 0.8, 50)
```

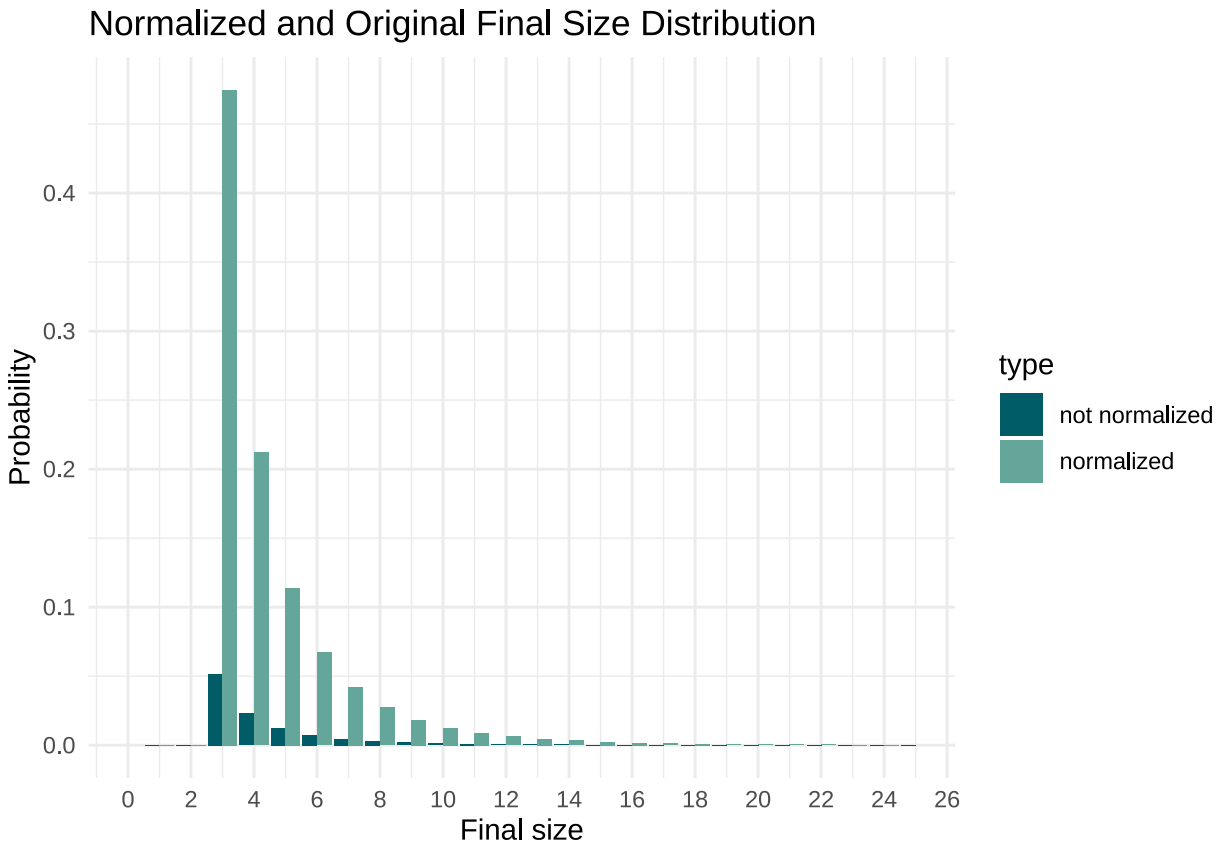


4.3. Normalized final size distribution with initial population size N

We can normalize the probability by dividing the original probability by the probability of all lineages go extinct (multiply $\frac{1}{\pi^n}$). So now the final size distribution represents the probability of having i final size of wild-type offspring given that the wild-type goes extinct at the end.

```r
FSn_normal <- function(n, y, R0, mu) {
  prob <- 0
  if (n == 1) {
    prob <- FS(y, R0, mu)
  } else if (n > 1) {
    for (i in 1:y - n + 1) {
      prob <- prob + FS(i, R0, mu) * FSn(n - 1, y - i, R0, mu)
    }
  } else {
    prob <- 0
  }
  pi <- 1 - P_wt_est(n, R0, mu)
  prob_normal <- prob / pi
  return(prob_normal)
}
```

```r
pmf.FSn <- function(n, R0, mu, maxFS) {
  df_n1 <- pmf.FS(R0, mu, maxFS)
  prob <- rep(0, maxFS)
  if (n == 1) {
    df <- df_n1
  } else if (n > 1) {
    df_n_one_less <- pmf.FSn(n - 1, R0, mu, maxFS)
    for (i in (1:(maxFS - n))) {
      for (j in ((n - 1):(maxFS - i))) {
        prob[i + j] <- prob[i + j] + df_n1[i, 2] * df_n_one_less[j, 2]
      }
    }
    names(prob) <- seq(1, maxFS)
    df <- data.frame(FinalSize = names(prob), prob)
  }
  pi <- max((1 - P_wt_est(n, R0, mu)), sum(df$prob))
  df$prob_normal <- df$prob / pi
  df$FinalSize <- as.numeric(df$FinalSize)
  return(df)
}

# sample usage and plot
df_FSnormal <- pmf.FSn(3, 8, 0.8, 50)
```



5. Mutant Offspring Generated Distribution

5.1. Mutant offspring distribution with initial population size $N = 1$

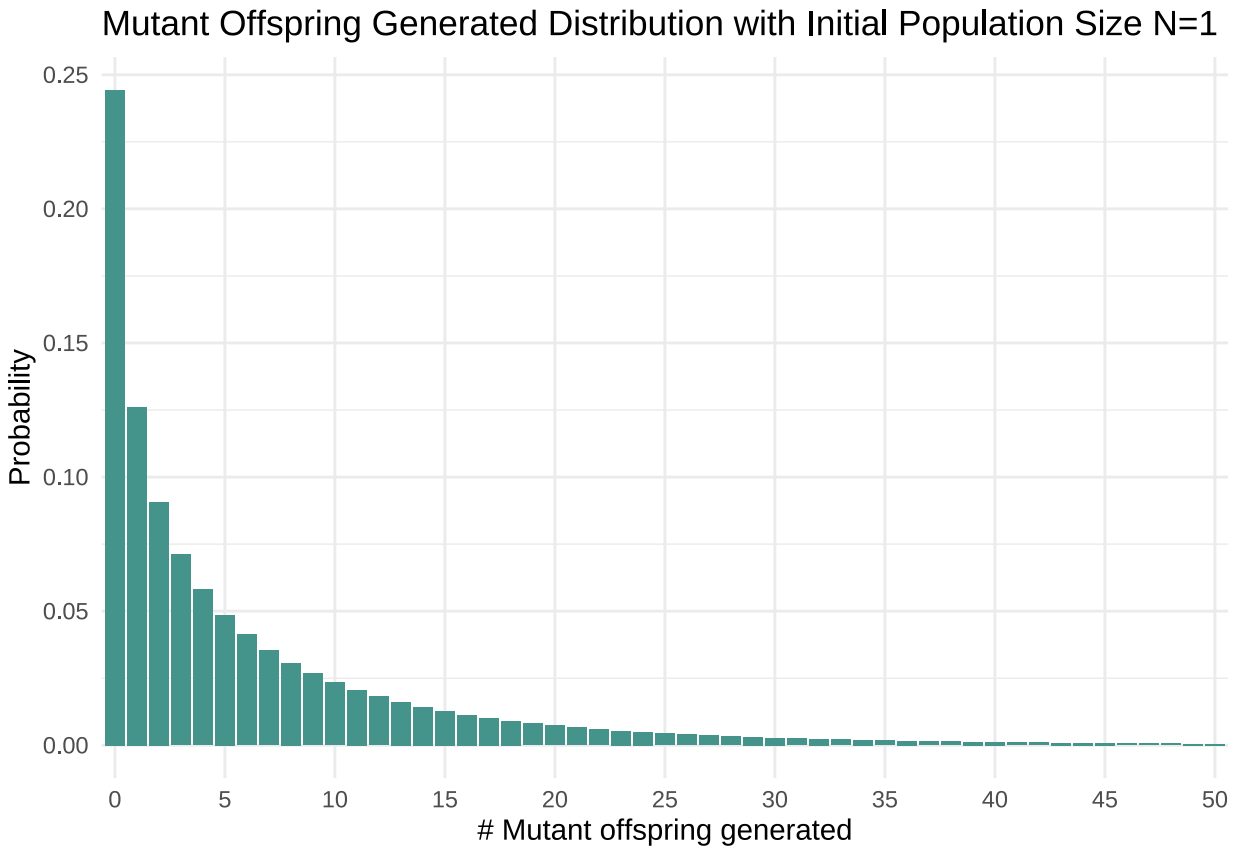
Each individual wild-type viral particle's probability of generating mutant offspring follows a negative binomial distribution $NB(1 - e^{-\mu}, \frac{1}{R_0+1})$ with a mean number of $R_0(1 - e^{-\mu})$.

The mutant offspring distribution of multiple wild-type viral particles is a simple addition of multiple independent negative binomial distributions. The new distribution now follows $NB(n \times (1 - e^{-\mu}), \frac{1}{R_0+1})$.

We can generate the pmf for mutant offspring distribution by calculate the **weighted probability** of getting a fixed number of mutant offspring from the final size distribution.

$$P(mu = m) = \sum_{i=1}^{\infty} P(mu = m|y = i) \times P(y = i)$$

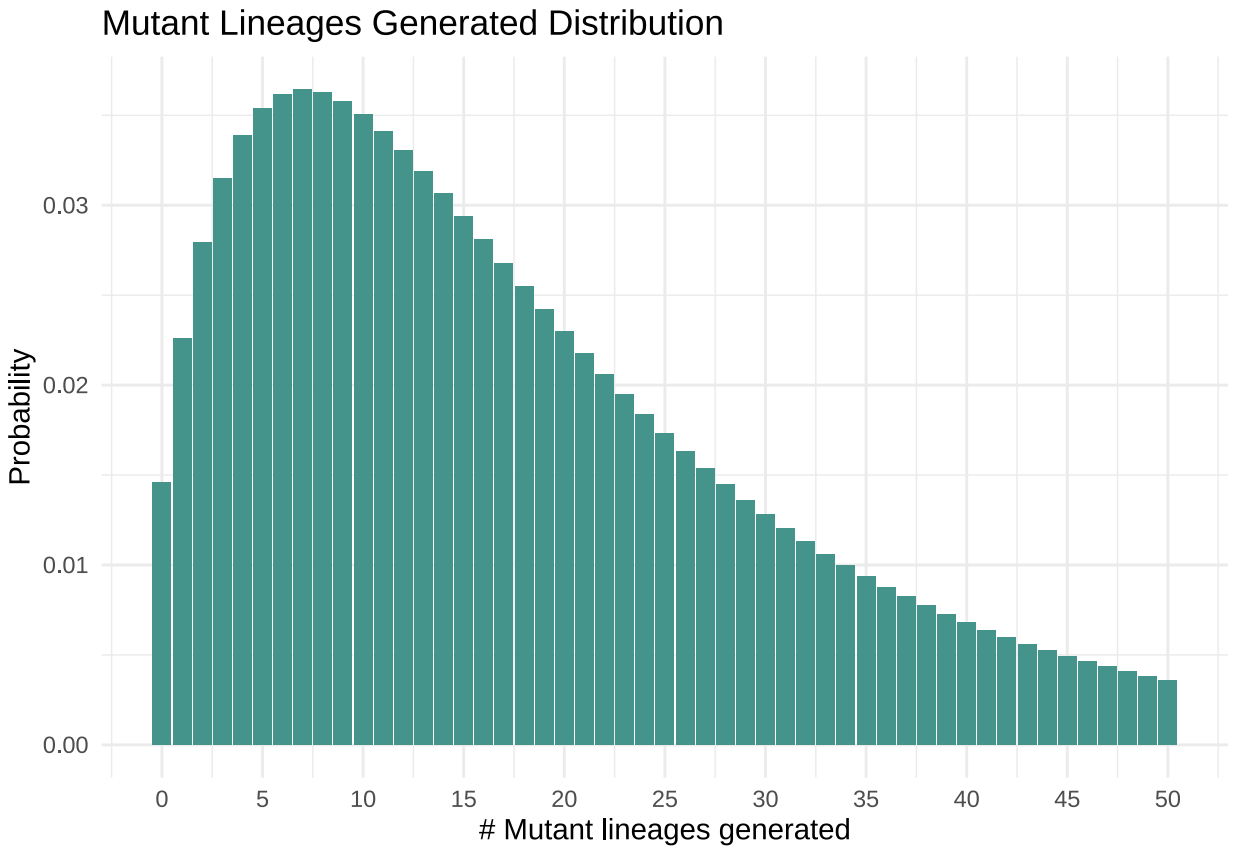
```r
P_mu <- function(m, R0, mu) {
  prob <- 0
  for (i in 1:50) {
    k <- (1 - exp(-mu)) * i
    p <- 1 / (R0 + 1)
    NBmu <- dnbinom(m, k, p)
    prob <- prob + FSn_normal(1, i, R0, mu) * NBmu
  }
  return(prob)
}

# store pmf in a data frame
pmf.P_mu <- function(R0, mu) {
  prob <- rep(0, 51)
  for (i in seq(0, 50, by = 1)) {
    prob[i + 1] <- prob[i + 1] + P_mu(i, R0, mu)
  }
  names(prob) <- seq(0, 50)
  df <- data.frame(MuOffspring = names(prob), Probability = prob)
  return(df)
}

# sample usage and plot
df_pmfMu <- pmf.P_mu(8, 0.8)
```



5.2. Extend to arbiturary initial population size N

```r
P_mu_n <- function(n, m, R0, mu, maxFS) {
  prob <- 0
  df <- pmf.FSn(n, R0, mu, maxFS)
  for (i in 1:maxFS) {
    k <- (1 - exp(-mu)) * i
    p <- 1 / (R0 + 1)
    NBmu <- dnbinom(m, k, p)
    prob <- prob + df[i, 3] * NBmu
  }
  return(prob)
}

# store pmf in a data frame
pmf.P_mu_n <- function(n, R0, mu, maxMuGen, maxFS) {
  prob <- rep(0, maxMuGen + 1)
  df_FS <- pmf.FSn(n, R0, mu, maxFS)
  for (i in 0:maxMuGen) {
    prob_mugen <- 0
    for (j in 1:maxFS) {
      k <- (1 - exp(-mu)) * j
      p <- 1 / (R0 + 1)
      NBmu <- dnbinom(i, k, p)
      prob_mugen <- prob_mugen + df_FS[j, 3] * NBmu
```

```
    }
    prob[i + 1] <- prob[i + 1] + prob_mugen
  }
  names(prob) <- c(0:maxMuGen)
  df <- data.frame(MuLinGen = names(prob), prob)
  df$MuLinGen <- as.numeric(df$MuLinGen)
  return(df)
}

# sample usage and plot
df_pmfMu_n <- pmf.P_mu_n(3, 8, 0.8, 50, 50)
```



6. Established Mutant Offspring Distribution

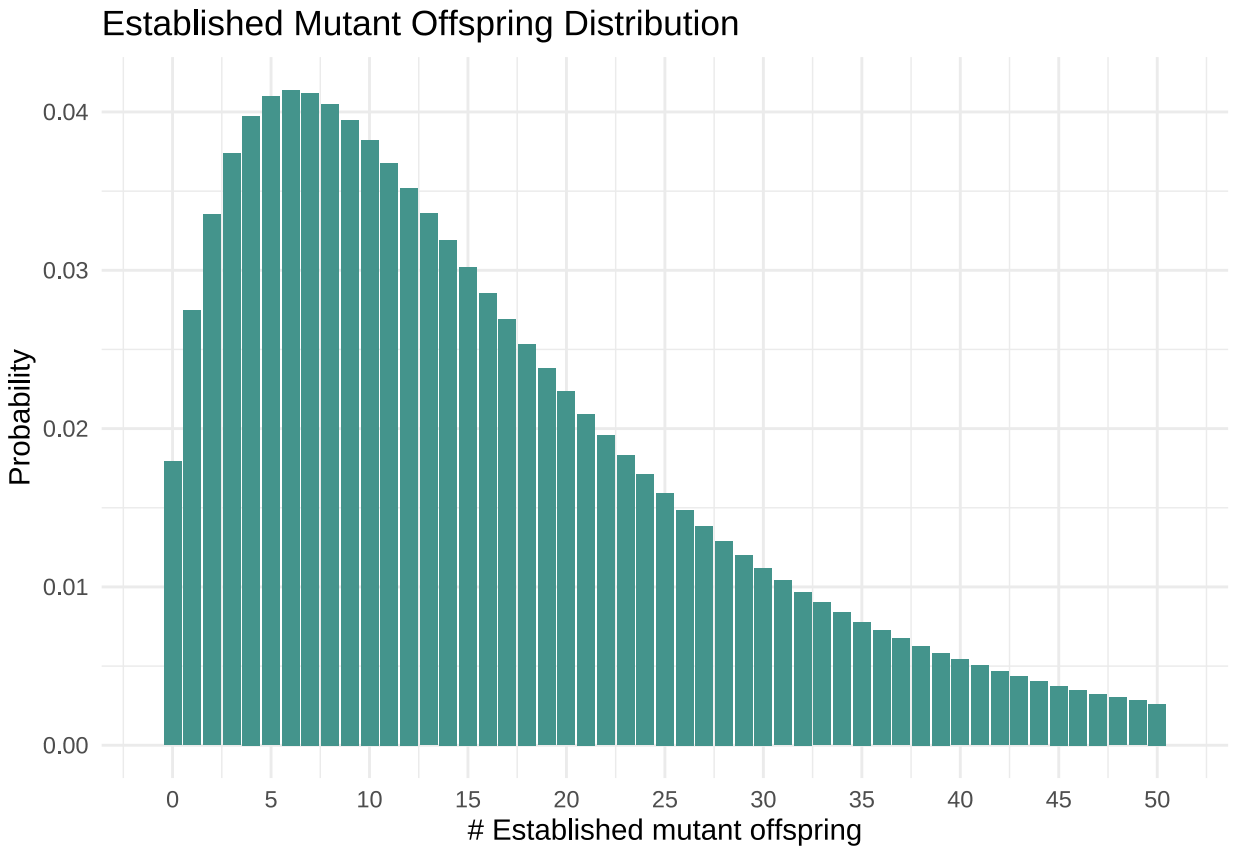
For any single viral particle (regardless of wild-type or mutant), the overall offspring distribution (including all wild-type and mutant offspring) follows a geometric distribution with a mean of R_0, and the probability of establishment is thus $1 - \frac{1}{R_0}$.

With a known probability of establishment, if we have n mutant offspring, the probability that a fixed number of offspring establishes can be calculated as a binomial distribution with `dbinom(number_established, number_mutant, prob_establish)`.

We can thus generate the pmf of established mutant lineages distribution similar to how we generate the mutant offspring distribution.

$$\begin{aligned} P(mu, est = l) &= \sum_{i=l}^{\infty} P(mu, est = l | mu = i) P(mu = i) \\ &= \sum_{i=l}^{\infty} \binom{i}{l} (p_{est})^l (1 - p_{est})^{i-l} \times P(mu = i) \end{aligned}$$

```r
P_mu_est <- function(n, m, R0, mu, maxMuGen, maxFS) {
  p_est <- 1 - 1 / R0
  prob <- rep(0, 1)
  pmf_mu_gen <- pmf.P_mu_n(n, R0, mu, maxMuGen, maxFS)
  for (i in m:maxMuGen) {
    p_m <- pmf_mu_gen[i + 1, 2]
    prob <- prob + dbinom(m, i, p_est) * p_m
  }
  return(prob)
}

# store pmf in a data frame
pmf.P_mu_est <- function(n, R0, mu, maxMuGen, maxFS) {
  p_est <- 1 - 1 / R0
  df_mu_gen <- pmf.P_mu_n(n, R0, mu, maxMuGen, maxFS)
  prob <- rep(0, maxMuGen + 1)
  for (j in 0:maxMuGen) {
    p_m <- df_mu_gen[j + 1, 2]
    for (i in 0:j) {
      prob[i + 1] <- prob[i + 1] + dbinom(i, j, p_est) * p_m
    }
  }
  names(prob) <- c(0:maxMuGen)
  df <- data.frame(MuLinEst = names(prob), prob)
  df$MuLinEst <- as.numeric(df$MuLinEst)
  return(df)
}

# sample usage and plot
df_pmfMu_est <- pmf.P_mu_est(3, 8, 0.8, 100, 100)
```



Notice that the probability of no mutant offspring established ($P(mu, est = 0)$) normalized by wild-type extinction is equal to the probability that the whole population goes extinct normalized by wild-type extinction. We can thus check the correctness of the function by comparing between the two.

```
# prob of whole population extinct
px <- 1 / 8
pwt_ext <- 1 - P_wt_est(1, 8, 0.8)
px / pwt_ext
```

```
## [1] 0.2617299
```

```
# prob of established mutant = 0
P_mu_est(1, 0, 8, 0.8, 50, 50)
```

```
## [1] 0.2617282
```

7. Clonal Mutation Distribution

7.1. P_X, P_0, and P_{1+}

7.1.1. P_X

Since the overall offspring distribution follows a geometric distribution, the probability of the whole lineage goes extinction with initial population N is given by $\frac{1}{R_0^n}$. Thus, we can obtain the analytical P_X as

```
Px_analytical <- function(n, R0) {
  px <- 1 / R0^n
  return(px)
}
```

Meanwhile, notice that P_x can also be calculated by the probability that the wild-type lineage goes extinct and no mutant lineages are generated. We can thus write our own method for calculating P_x as

```
Px <- function(n, R0, mu, maxMuGen, maxFS) {
  p_wt_ext <- 1 - P_wt_est(n, R0, mu)
  p_mu_ext <- P_mu_est(n, 0, R0, mu, maxMuGen, maxFS)
  prob <- p_mu_ext * p_wt_ext
  return(prob)
}
```

The comparison between the two can test for the accuracy of our method.

```
Px_analytical(3, 8)
```

```
## [1] 0.001953125
```

```
Px(3, 8, 0.8, 80, 80)
```

```
## [1] 0.001953088
```

7.1.2. P_0

The probability of having no clonal mutations is the sum of the following two probabilities:

1. Wild-type lineage establishes (so there's always new mutant lineages being generated);
2. More than two mutant lineages establish.

The first probability is given by $1-\pi^n$ and is calculated in part `1.2` with the function `P_wt_est`. The second probability can be calculated using the established mutant offspring pmf and subtract $P(mu, est = 0)$ and $P(mu, est = 1)$ from 1.

```
P0 <- function(n, R0, mu, maxMuGen, maxFS) {
  # S stands for the number of established mutant lineages
  S_inf <- P_wt_est(n, R0, mu)
  S_2plus <- (1 - P_mu_est(n, 0, R0, mu, maxMuGen, maxFS) -
    P_mu_est(n, 1, R0, mu, maxMuGen, maxFS)) * (1 - S_inf)
  prob <- S_inf + S_2plus
  return(prob)
}
```

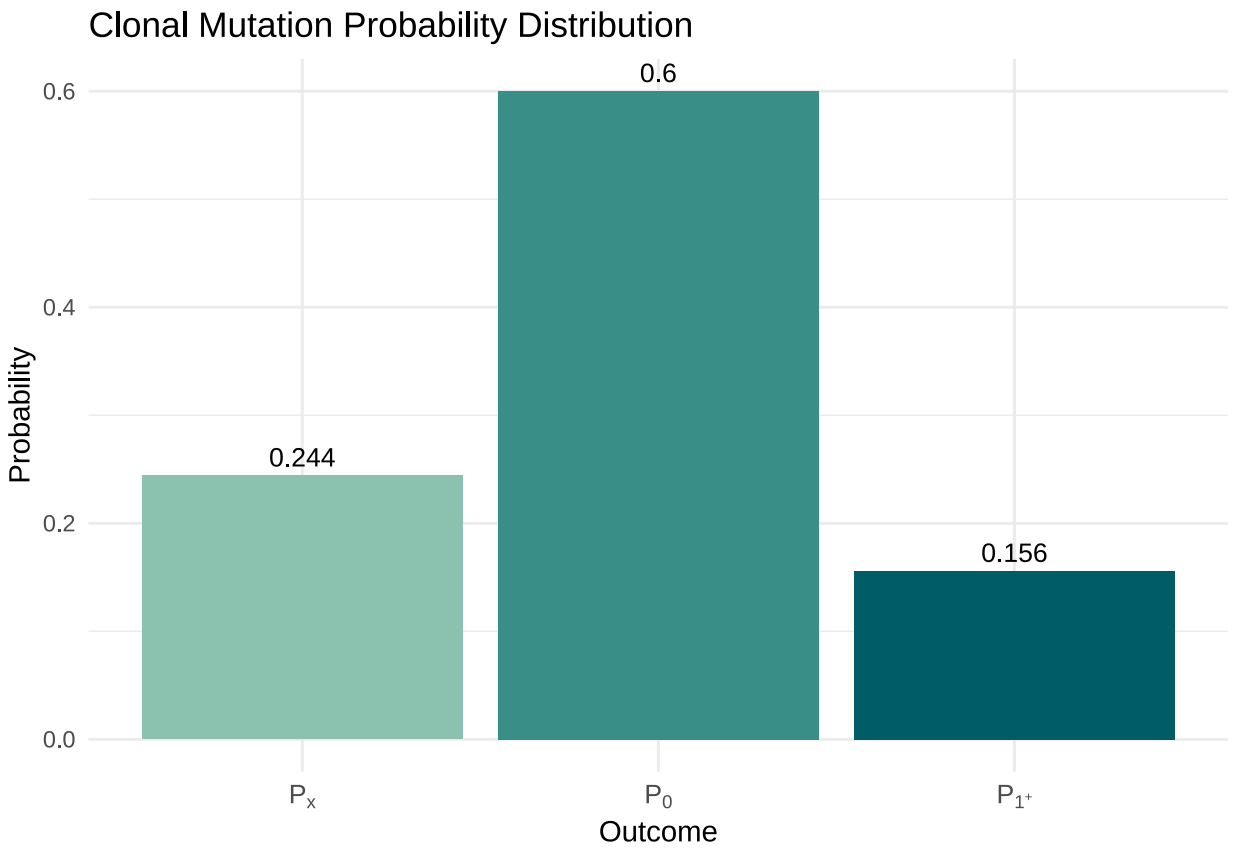
7.1.3. P_{1+}

The probability of having more than 1 clonal mutation is the probability of having exactly one established mutant lineage and any number of mutations in this mutant lineage.

```
P1plus <- function(n, R0, mu, maxMuGen, maxFS) {
  p_wt_ext <- 1 - P_wt_est(n, R0, mu)
  prob <- P_mu_est(n, 1, R0, mu, maxMuGen, maxFS) * p_wt_ext
  return(prob)
}
```

Integrate P_X, P_0, and P_{1+} into a single function.

```
pmf.P_clonal_3 <- function(n, R0, mu, maxMuGen, maxFS) {
  prob <- rep(0, 3)
  prob[1] <- Px_analytical(n, R0)
  prob[2] <- P0(n, R0, mu, maxMuGen, maxFS)
  prob[3] <- P1plus(n, R0, mu, maxMuGen, maxFS)
  names(prob) <- c("Px", "P0", "P1plus")
  df <- data.frame(Type = names(prob), Probability = prob)
  return(df)
}

# sample usage and plot
df_clonal_3 <- pmf.P_clonal_3(3, 1.6, 0.4, 50, 50)
```



7.2. Resolving P_{1+}

Now we scrutinize the case that only one mutant offspring is established (P_{1+}) and resolve for the probabilities for P_1, P_2, etc.

The number of mutations in a single viral particle follows a Poisson distribution with a mean of μ. The probability of having m mutations in this established mutant lineage can thus be calculated by

$$Q_m = P_{1+} \times \frac{Poisson(k = m; \mu)}{1 - Poisson(k = 0; \mu)}$$

The equation is normalized by $1 - Poisson(k = 0; \mu)$ because we already know this is a mutant lineage.

```
P_clonal_round <- function(n, m, R0, mu, maxMuGen, maxFS) {
  prob <- P1plus(n, R0, mu, maxMuGen, maxFS) * dpois(m, mu) / (1 - dpois(0, mu))
  return(prob)
}

# store pmf in a data frame
pmf.P_clonal_round <- function(n, R0, mu, maxMuGen, maxFS, maxClonal) {
  p1plus <- P1plus(n, R0, mu, maxMuGen, maxFS)
  prob <- rep(0, maxClonal + 2)
  prob[1] <- 1 / R0^n
  prob[2] <- P0(n, R0, mu, maxMuGen, maxFS)
  if (maxClonal != 0) {
    for (i in 1:maxClonal) {
      prob[i + 2] <- p1plus * dpois(i, mu) / (1 - dpois(0, mu))
    }
  }
  names(prob) <- c("x", 0:maxClonal)
  df <- data.frame(ClonalMu = names(prob), prob)
  return(df)
}
```

Notice that the offspring of this single mutant individual might also generate more mutations, so the Q_1 we calculated is actually an overestimation of the true P_1. Some density of P_{2+} might also be included in the Q_1 we just calculated. To solve for this, we will need to regard this mutant individual as the starting individual and again calculate the clonal mutation distribution of its offspring and polish the original distribution.

For example, if we regard the mutant individual as the original starting viral particle (e.g. regard as initial wild-type particle), the $P_{1,N=1}$ of this new lineage would result in a mutant lineage with 2 mutations in total, and this should count toward P_2 for the initial population size.

For clarity in notation, we will call the previous not-exact clonal mutation probability as Q_n, and the new exact clonal mutation probability to be P_n. We can thus express the exact clonal mutation probability as

$$\begin{aligned}
P_{m,N=n} &= Q_{m,N=n} \times \frac{P_{0,N=1}}{1 - P_{x,N=1}} + \sum_{i=1}^{m-1} Q_{i,N=n} \frac{P_{m-i,N=1}}{1 - P_{x,N=1}} \\
&= \sum_{i=1}^{m} Q_{i,N=n} \frac{P_{m-i,N=1}}{1 - P_{x,N=1}} \\
&= \sum_{i=1}^{m} P_{1+} \times \frac{Poisson(k = m; \mu)}{1 - Poisson(k = 0; \mu)} \times \frac{P_{m-i,N=1}}{1 - P_{x,N=1}}
\end{aligned}$$

```r
P_clonal <- function(n, m, R0, mu, maxMuGen, maxFS) {
  df_clonal_round <- pmf.P_clonal_round(n, R0, mu, maxMuGen, maxFS, m)[-1, ]
  px <- 1 / R0
  prob <- 0
  if (m == 0) {
    prob <- df_clonal_round[1, 2]
  } else {
    for (i in 1:m) {
      prob <- prob + df_clonal_round[i + 1, 2] *
        P_clonal(1, m - i, R0, mu, maxMuGen, maxFS) / (1 - px)
    }
  }
  return(prob)
}
```

```r
# compare the previous P1 and the new P1
P_clonal_round(3, 1, 8, 0.8, 50, 50)
```

```
## [1] 0.001952587
```

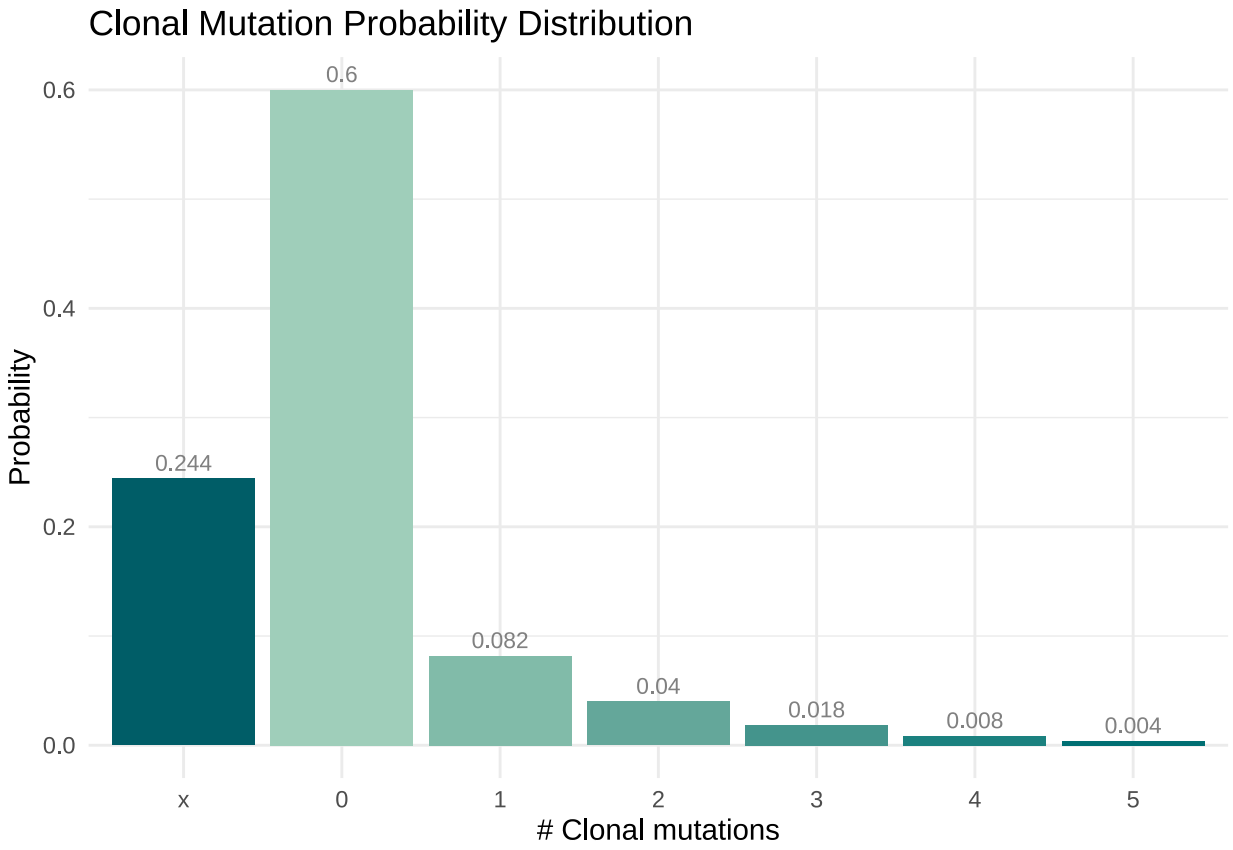
```r
P_clonal(3, 1, 8, 0.8, 50, 50)
```

```
## [1] 0.001810187
```

Comparison between the round and exact clonal probabilities shows that the round is larger than the exact. This makes sense because the round probability serves as an upper bound of the actual probability.

Combine the resolved P_{1+} with the previously calculated P_x and P_0 to a single function and store the pmf in a data frame.

```r
pmf.P_clonal_withPx <- function(n, R0, mu, maxMuGen, maxFS, maxClonal) {
  df_clonal_round <- pmf.P_clonal_round(n,R0,mu,maxMuGen,maxFS,maxClonal)[-1, ]
  px <- 1 / R0
  prob <- rep(0, maxClonal + 2)
  prob[1] <- 1 / R0^n
  prob[2] <- df_clonal_round[1, 2]
  if (maxClonal != 0) {
    df_clonal_max_one_less <-
      pmf.P_clonal_withPx(1, R0, mu, maxMuGen, maxFS, maxClonal - 1)
    for (m in 1:maxClonal) {
      for (i in 1:m) {
        prob[m + 2] <- prob[m + 2] + df_clonal_round[i + 1, 2] *
          df_clonal_max_one_less[m - i + 2, 2]/(1 - px)
      }
    }
  }
  names(prob) <- c("x", 0:maxClonal)
  df <- data.frame(ClonalMu = names(prob), prob)
  return(df)
}

# sample usage and plot
df_clonal <- pmf.P_clonal_withPx(3, 1.6, 0.4, 50, 50, 5)
```




```
# check if the resolved probabilities add up to P_1+
P1plus(3, 1.6, 0.4, 50, 50)
```

```
## [1] 0.1558086
```

```
sum(df_clonal$prob[3:8])
```

```
## [1] NA
```

Since only the established infections will be observed and recorded, we can also make a normalized version of the function representing $\rho_i = \frac{P_i}{1-P_x}$.

```
pmf.P_clonal <- function(n, R0, mu, maxMuGen, maxFS, maxClonal) {
  df_clonal_round <- pmf.P_clonal_round(n, R0, mu, maxMuGen, maxFS, maxClonal)[-1, ]
  px <- 1 / R0
  prob <- rep(0, maxClonal + 2)
  prob[1] <- 1 / R0^n
  prob[2] <- df_clonal_round[1, 2]
  if (maxClonal != 0) {
    df_clonal_max_one_less <-
      pmf.P_clonal_withPx(1, R0, mu, maxMuGen, maxFS, maxClonal - 1)
    for (m in 1:maxClonal) {
      for (i in 1:m) {
        prob[m + 2] <- prob[m + 2] + df_clonal_round[i + 1, 2] *
          df_clonal_max_one_less[m - i + 2, 2] / (1 - px)
```

```
      }
    }
  }
  names(prob) <- c("x", 0:maxClonal)
  df <- data.frame(ClonalMu = names(prob), prob)
  fold <- max(1 - df$prob[1], sum(df[-1,]$prob))
  df$prob <- df$prob / fold
  df <- df[-1, ]
  return(df)
}
```

For the convenience of subsequent calculations, we will make a function that calculates for the clonal mutation probabilities across various pairs of N and μ values.

```
list_clonal <- function(n_values, R0, mu_values, maxMuGen, maxFS, clonal) {
  listClonal <- 1:length(mu_values) %>%
    mclapply(function(x) {
      1:length(n_values) %>%
        lapply(function(y) {
          pmf.P_clonal(n_values[y], R0, mu_values[x], maxMuGen, maxFS, clonal)
        })
    }, mc.cores = detectCores())
  # names the elements in the list with mu and n values
  names(listClonal) <- mu_values
  for (i in 1:length(mu_values)) {
    names(listClonal[[i]]) <- n_values
  }
  return(listClonal)
}

# sample usage
listClonal <- list_clonal(1:8, 1.6, seq(0.01, 0.8, by = 0.02), 50, 50, 8)
```

8. Applications

We will now introduce several stochastic simulations to test the model's accuracy.

```
# 100 stochastic simulations that established successfully
# R0 = 1.6
sample1 <- data.frame(ClonalMu = 0:5, freq = c(76, 14, 5, 4, 0, 1))
sample2 <- data.frame(ClonalMu = 0:5, freq = c(82, 9, 3, 3, 1, 2))

# flu data
# R0 = 11.1
flu <- data.frame(ClonalMu = 0:3, freq = c(42, 5, 2, 3))
```

8.1. Initial population distribution as Poisson distribution

8.1.1. Initial population distribution

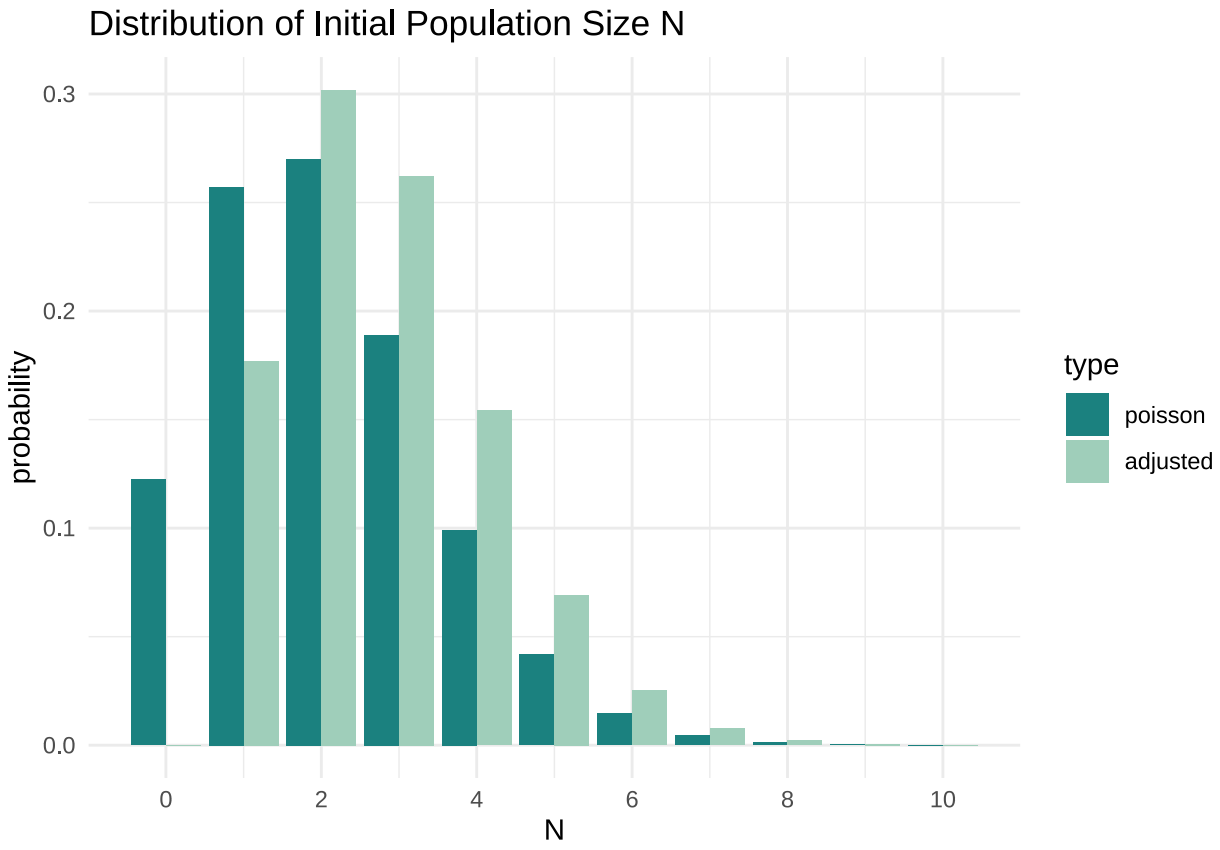
We can model the distribution of initial population size N as a Poisson distribution with a mean of λ. However, the actual distribution of N is not exactly a Poisson distribution: since the case that all initial particles goes extinct is not observed and is logged as uninfected, we should normalize the distribution by the probability that at least one of the initial viral particles establish. This is given by

$$P(N = n; \lambda) = \frac{Poisson(N = n; \lambda) \times (1 - \frac{1}{R_0^n})}{\sum P(N = n; \lambda)}$$

The adjusted probabilities is normalized by the sum of the adjusted probabilities.

```
pmf.ini_size <- function(lambda, R0, maxIni) {
  df <- data.frame(N = 0:maxIni, pois = dpois(0:maxIni, lambda))
  df$adj_prob <- 0
  for (n in 0:maxIni) {
    Pest <- 1-1/R0^n
    df$adj_prob[n+1] <- df$pois[n+1] * Pest
  }
  df$adj_prob <- df$adj_prob/sum(df$adj_prob)
  df <- melt(df, id.vars = "N")
  names(df) <- c("N", "type", "prob")
  df$type <- factor(df$type)
  levels(df$type) <- c("poisson", "adjusted")
  return(df)
}

# sample usage and plot
df_pois <- pmf.ini_size(2.1, 1.6, 10)
```

We can compare the adjusted distribution of N to the original Poisson distribution under same parameter λ.



8.1.2. Estimate λ of initial population distribution

Having known the distribution of N under each λ, we are able to estimate the most likely combination of (λ, μ). The probability of observing i clonal mutations with given parameters can be expressed as

$$\begin{aligned} P_{i|\lambda,\mu,R0} &= \sum_{n=1}^{\infty} P(N=n;\lambda) \times \rho_{i|n,\mu,R0} \\ &= \sum_{n=1}^{\infty} \frac{Poisson(N=n;\lambda) \times (1-\frac{1}{R_0^n})}{\sum P(N=n;\lambda)} \times \frac{P_{i|n,\mu,R0}}{1-1/R_0^n} \end{aligned}$$

The likelihood function for (λ, μ) is given by

$$L(\lambda,\mu) = \prod_{i=0}^{m} (P_{i|\lambda,\mu,R0})^{k_i}$$

where $P_{i|\lambda,\mu,R0}$ is the probability that the successfully infected recipient's viral population harbors i clonal mutations given the parameters, k_i stands for the number of cases i clonal mutations is observed in the population, and m is the maximum number of clonal mutations observed in the population.

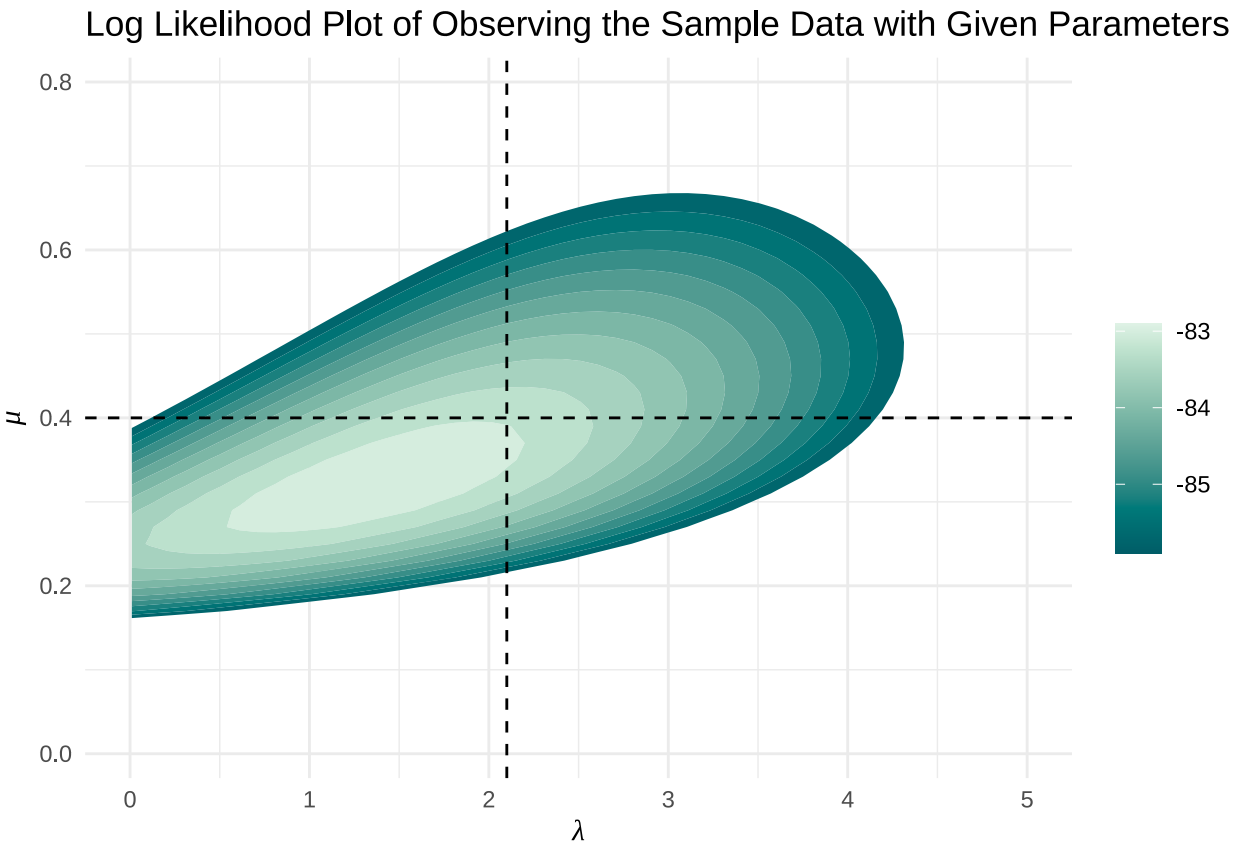
And the log likelihood function for (λ, μ) is simply

$$
\begin{aligned}
l(\lambda, \mu) &= logL(\lambda, \mu) \\
&= log \prod_{i=0}^{m} (P_{i|\lambda,\mu,R0})^{k_i}
\end{aligned}
$$

```
LL_lambda <- function(df, listClonal, lambda, R0, mu, maxMuGen, maxFS, maxIni) {
  df$prob <- df$freq / sum(df$freq)
  dfPn <- pmf.ini_size(lambda, R0, maxIni)
  dfPn <- dfPn[dfPn$type == "adjusted",]
  prob <- 1

  list <- mclapply(df$ClonalMu,
                   pmf.clonal_params,
                   listClonal = listClonal,
                   n_values = 1:maxIni,
                   R0 = R0,
                   mu_values = mu,
                   maxMuGen = maxMuGen,
                   maxFS = maxFS,
                   mc.cores = detectCores())
  names(list) <- df$ClonalMu

  for (i in df$ClonalMu) {
    loc <- which(names(list) == i)
    dfClonalN <- list[[loc]]
    PclonalLam <- 0
    for (n in 1:maxIni) {
      Pn <- dfPn$prob[dfPn$N == n]
      PclonalN <- dfClonalN$prob[dfClonalN$n == n]
      PclonalLam <- PclonalLam + Pn*PclonalN
    }
    k <- df$freq[df$ClonalMu == i]
    prob <- prob * PclonalLam^k
  }
  prob <- log(prob)
  return(prob)
}

# store pmf in a data frame
LL_lambda.df <- function(df, listClonal, lambda_values, R0,
                         mu_values, maxMuGen, maxFS, maxIni) {
  prob <- 1:length(mu_values) %>%
    mclapply(function(x) {
      1:length(lambda_values) %>%
        lapply(function(y) {
          LL_lambda(df, listClonal, lambda_values[y],
                    R0, mu_values[x], maxMuGen, maxFS, maxIni)
        })
    }, mc.cores = detectCores())
  names(prob) <- mu_values
  for (i in 1:length(mu_values)) {
    names(prob[[i]]) <- lambda_values
  }
```

```
  LLdf <- expand.grid(lambda = paste0("", lambda_values),
                      mu = paste0("", mu_values))
  LLdf$prob <- unlist(prob)
  LLdf$lambda <- as.numeric(paste(LLdf$lambda))
  LLdf$mu <- as.numeric(paste(LLdf$mu))
  return(LLdf)
}
```

```
# sample usage and plot
df_lambdaLL <- LL_lambda.df(sample1, listClonal, seq(0.01, 5.01, by = 0.1),
                            1.6, seq(0.01, 0.8, by = 0.02), 50, 50, 8)
```




```
df_meanNbLL[which(df_meanNbLL$prob == max(df_meanNbLL$prob)),]
```

```
##     lambda   mu      prob    meanN   meanNb
## 831   1.41 0.33 -82.92672 2.168526 1.287474
```

The dashed lines represents the actual parameters used to generate the sample data ($\lambda = 2.1, \mu = 0.4$). The estimated maximum likelihood estimators are $\lambda = 1.41, \mu = 0.33$.

8.1.3. Estimate mean initial population size

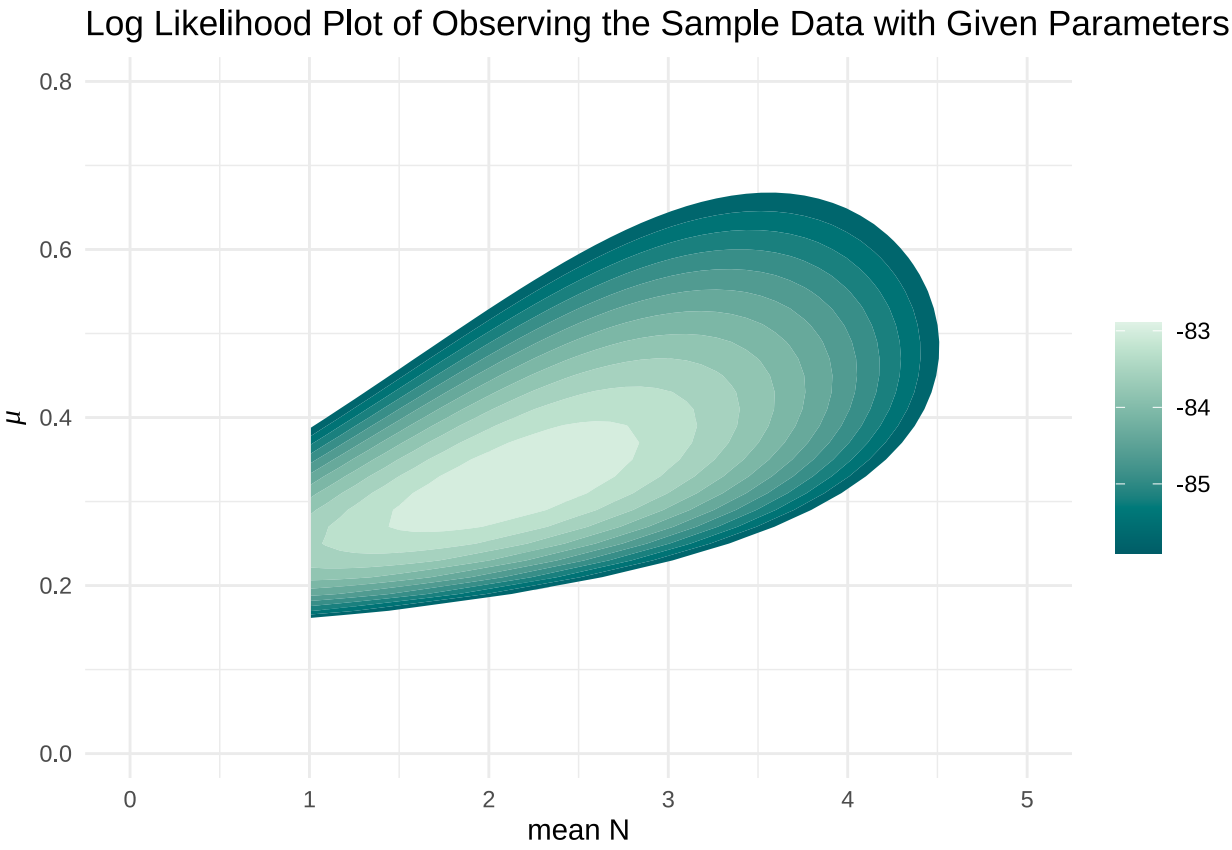
Because the distribution of initial population size N is an altered version of Poisson distribution rather than a typical Poisson distribution, the mean of N is not λ. It is given by:

$$\bar{N} = \sum_{n=1}^{\infty} nP(N = n; \lambda)$$

We can thus calculate the $\bar{N}$ for each of the λ and plot the same log-likelihood landscape now with $\bar{N}$ on x-axis.

```
LL_meanN.df <- function(df, listClonal, lambda_values, R0,
                        mu_values, maxMuGen, maxFS, maxIni) {
  prob <- 1:length(mu_values) %>%
    mclapply(function(x) {
      1:length(lambda_values) %>%
        lapply(function(y) {
          LL_lambda(df, listClonal, lambda_values[y],
                    R0, mu_values[x], maxMuGen, maxFS, maxIni)
        })
    }, mc.cores = detectCores())
  names(prob) <- mu_values
  for (i in 1:length(mu_values)) {
    names(prob[[i]]) <- lambda_values
  }

  LLdf <- expand.grid(lambda = paste0("", lambda_values), mu = paste0("", mu_values))
  LLdf$prob <- unlist(prob)
  LLdf$lambda <- as.numeric(paste(LLdf$lambda))
  LLdf$mu <- as.numeric(paste(LLdf$mu))
  LLdf$meanN <- 0

  for (j in lambda_values) {
    dfPn <- pmf.ini_size(j, R0, maxIni)
    dfPn <- dfPn[dfPn$type == "adjusted",]
    dfPn$prob_N <- dfPn$N * dfPn$prob
    mean_n <- sum(dfPn$prob_N)
    LLdf[which(near(LLdf$lambda, j)),]$meanN <- mean_n
  }

  return(LLdf)
}
```

```
# sample usage and plot
df_meanNLL <- LL_meanN.df(sample1, listClonal, seq(0.01, 5.01, by = 0.1),
                          1.6, seq(0.01, 0.8, by = 0.02), 50, 50, 10)
```



8.1.4. Estimate the mean bottle neck size N_b

Now we have the most likely combination of (λ, μ) and $(\bar{N}, \mu)$, we can use them to estimate the transmission bottleneck size by applying the probability of establishment to the initial population size. The probability mass distribution for bottleneck size is given by

$$
\begin{aligned}
P(N_b = k) &= \sum_{n=k}^{\infty} P(N = n; \lambda) \times Binom(k; n, 1 - \frac{1}{R_0}) / [1 - Binom(0; n, 1 - \frac{1}{R_0})] \\
&= \sum_{n=k}^{\infty} \frac{Poisson(N = n; \lambda) \times (1 - \frac{1}{R_0^n})}{\sum P(N = n; \lambda)} \times \binom{n}{k} (1 - \frac{1}{R_0})^k (\frac{1}{R_0})^{n-k} / [1 - \binom{n}{0} (\frac{1}{R_0})^n]
\end{aligned}
$$

The expected number of transmission bottleneck size is thus

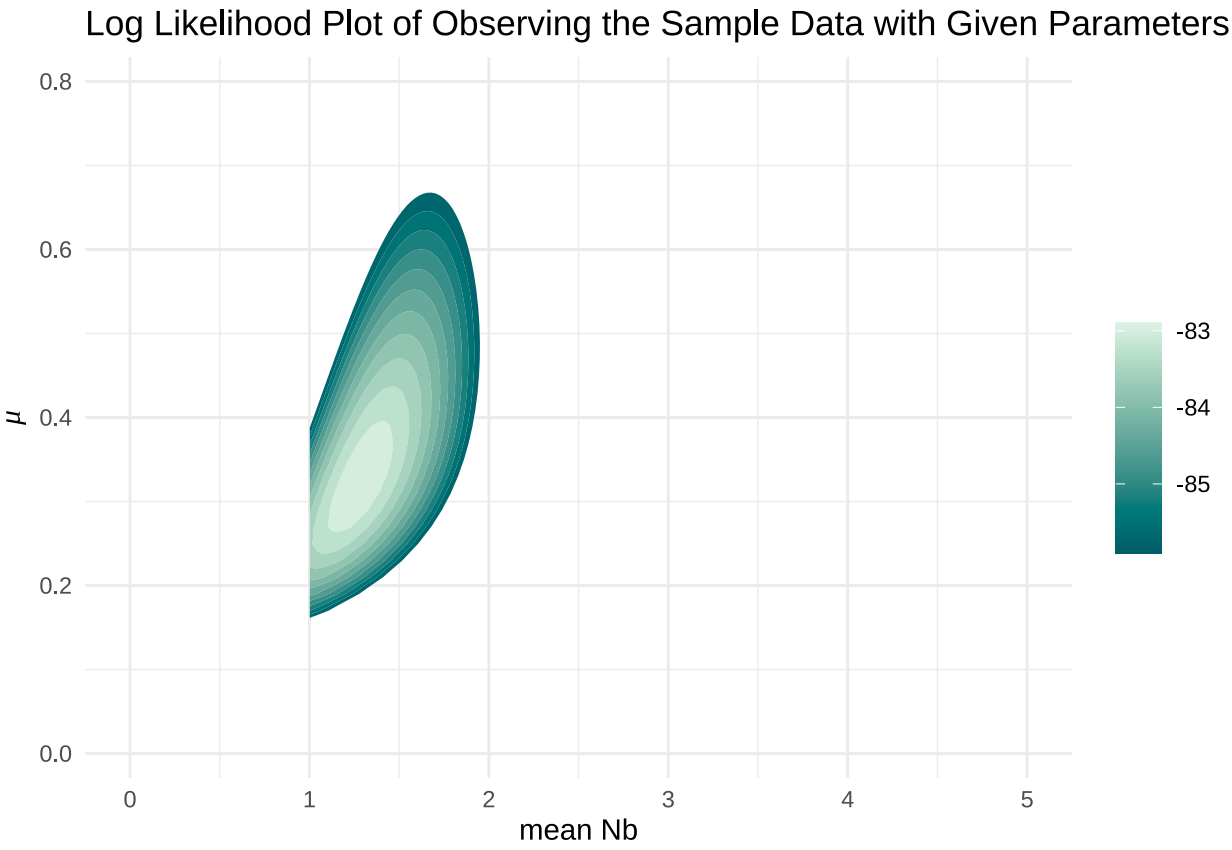
$$
\bar{N_b} = \sum_{k=1}^{\infty} k P(N_b = k)
$$

With the formula above, we are able to plot the same log-likelihood landscape now with $\bar{N_b}$ on x-axis.

```
LL_meanNb.df <- function(df, listClonal, lambda_values, R0,
                         mu_values, maxMuGen, maxFS, maxIni) {
  prob <- 1:length(mu_values) %>%
    mclapply(function(x) {
```

```
      1:length(lambda_values) %>%
        lapply(function(y) {
          LL_lambda(df, listClonal, lambda_values[y],
                    R0, mu_values[x], maxMuGen, maxFS, maxIni)
        })
    }, mc.cores = detectCores())
  names(prob) <- mu_values
  for (i in 1:length(mu_values)) {
    names(prob[[i]]) <- lambda_values
  }

  LLdf <- expand.grid(lambda = paste0("", lambda_values), mu = paste0("", mu_values))
  LLdf$prob <- unlist(prob)
  LLdf$lambda <- as.numeric(paste(LLdf$lambda))
  LLdf$mu <- as.numeric(paste(LLdf$mu))
  LLdf$meanN <- 0
  LLdf$meanNb <- 0

  for (j in lambda_values) {
    # calculate mean N
    dfPn <- pmf.ini_size(j, R0, maxIni)
    dfPn <- dfPn[dfPn$type == "adjusted",]
    dfPn$prob_N <- dfPn$N * dfPn$prob
    mean_n <- sum(dfPn$prob_N)
    LLdf[which(near(LLdf$lambda, j)),]$meanN <- mean_n

    # calculate mean Nb
    mean_nb <- 0
    for (k in 1:maxIni) {
      PNb <- 0
      for (n in k:maxIni) {
        PNb <- PNb + dfPn$prob[n+1] * dbinom(k, n, 1-1/R0) / (1-dbinom(0, n, 1-1/R0))
      }
      mean_nb <- mean_nb + k * PNb
    }
    LLdf[which(near(LLdf$lambda, j)),]$meanNb <- mean_nb
  }

  return(LLdf)
}
```

```
# sample usage and plot
df_meanNbLL <- LL_meanNb.df(sample1, listClonal, seq(0.01, 5.01, by = 0.1),
                            1.6, seq(0.01, 0.8, by = 0.02), 50, 50, 10, 10)
```



The λ values and $\bar{N}$ values are integrated into the function `LL_meanNb.df`, so you can use the single function to plot out the above three graphs to save time.

8.2. Initial population distribution as negative binomial distribution

8.2.1. Initial population distribution

It might be argued that due to the equality of mean and variance in a Poisson distribution, an estimated λ as small as 0.01 might be problematic since the variance is also forced to be zero when in reality the variance might be very large. We can thus implement an alternative approach to model the initial population distribution as a negative binomial distribution with dispersion parameter $r = 0.01$, which will have a huge variance. Same as before, we will need to normalize the distribution by the probability that at least one of the initial viral particles establish. This is given by

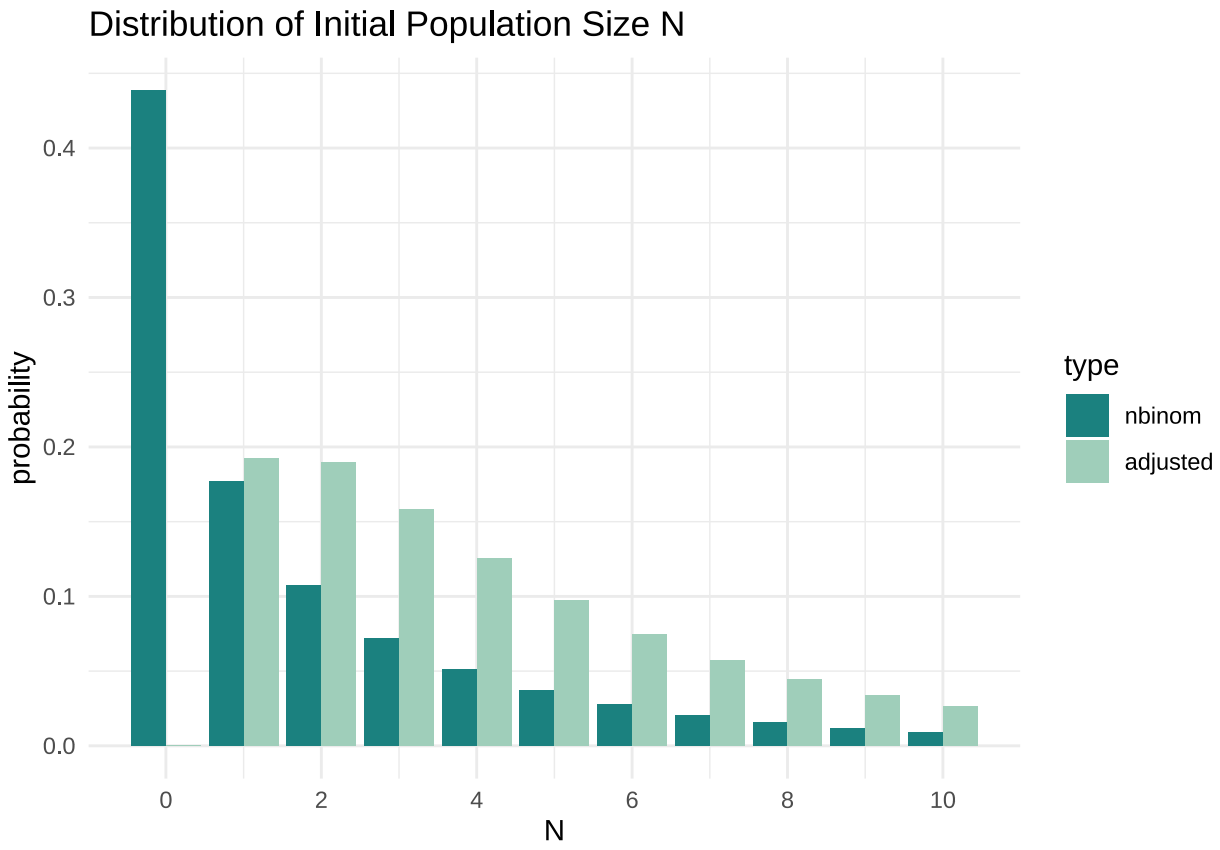
$$P(N = n; r, p) = \frac{NB(N = n; r, p) \times (1 - \frac{1}{R_0^n})}{\sum P(N = n; r, p)}$$

where $r = 0.01$ and $p = \frac{r}{r+mean}$. We will still use λ to denote the mean of the negative binomial distribution.

```
pmf.ini_size_nb <- function(lambda, R0, maxIni, k) {
  df <- data.frame(N = 0:maxIni, nbin = dnbinom(0:maxIni, size = k, mu = lambda))
  df$adj_prob <- 0
  for (n in 0:maxIni) {
    Pest <- 1-1/R0^n
    df$adj_prob[n+1] <- df$nbin[n+1] * Pest
  }
```

```
  df$adj_prob <- df$adj_prob/sum(df$adj_prob)
  df <- melt(df, id.vars = "N")
  names(df) <- c("N", "type", "prob")
  df$type <- factor(df$type)
  levels(df$type) <- c("nbinom", "adjusted")
  return(df)
}

# sample usage and plot
df_pois_nb <- pmf.ini_size_nb(2.1, 1.6, 10, 0.5)
```

We can compare the adjusted distribution of N to the original Poisson distribution under same parameter λ.



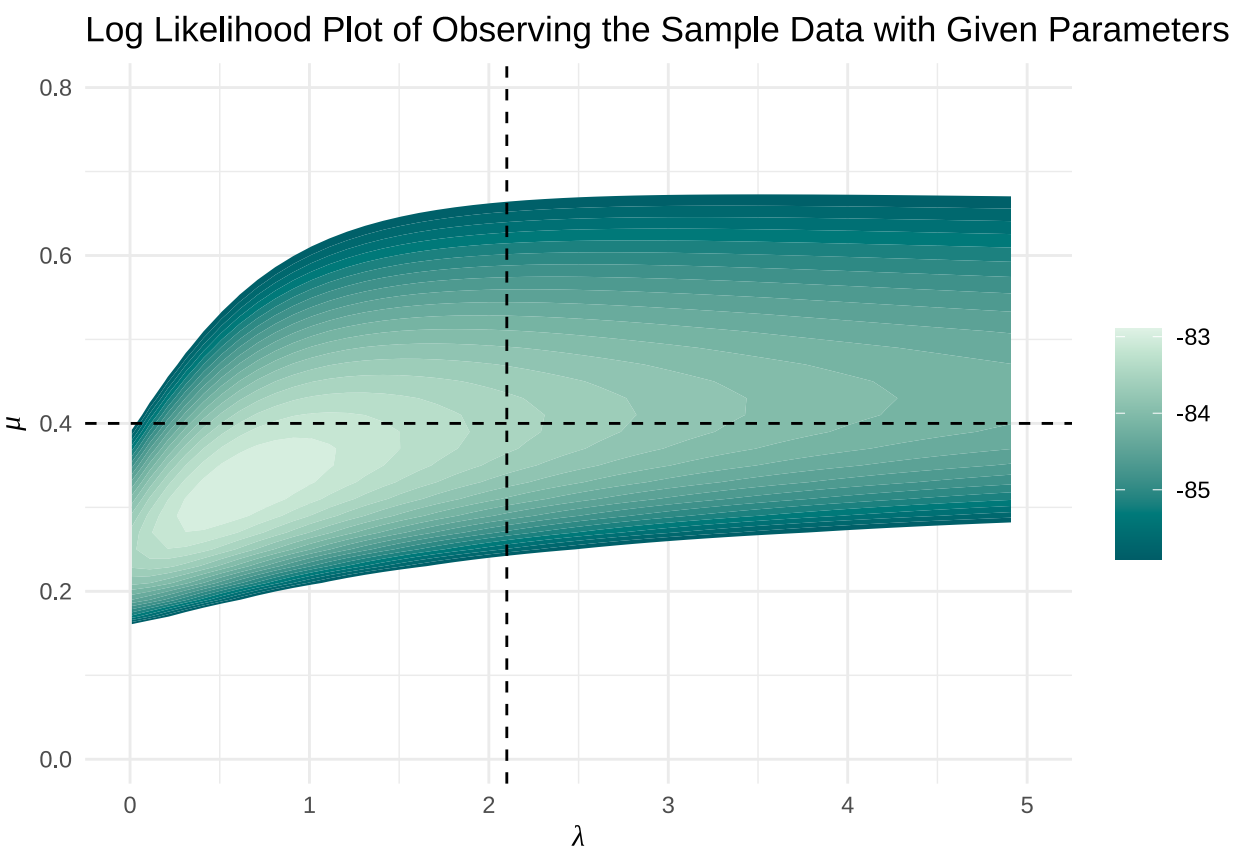
8.2.2. Estimate the mean bottle neck size N_b

With this new initial population size distribution, we can adjust the way we calculate the maximum likelihood estimation for λ, mean initial population size, and mean bottle neck size.

```
LL_lambda_nb <- function(df, listClonal, lambda, R0, mu, maxMuGen, maxFS, maxIni, k) {
  df$prob <- df$freq / sum(df$freq)
  dfPn <- pmf.ini_size_nb(lambda, R0, maxIni, k)
  dfPn <- dfPn[dfPn$type == "adjusted",]
  prob <- 1

  list <- mclapply(df$ClonalMu,
```

```
                    pmf.clonal_params,
                    listClonal = listClonal,
                    n_values = 1:maxIni,
                    R0 = R0,
                    mu_values = mu,
                    maxMuGen = maxMuGen,
                    maxFS = maxFS,
                    mc.cores = detectCores())
  names(list) <- df$ClonalMu

  for (i in df$ClonalMu) {
    loc <- which(names(list) == i)
    dfClonalN <- list[[loc]]
    PclonalLam <- 0
    for (n in 1:maxIni) {
      Pn <- dfPn$prob[dfPn$N == n]
      PclonalN <- dfClonalN$prob[dfClonalN$n == n]
      PclonalLam <- PclonalLam + Pn*PclonalN
    }
    m <- df$freq[df$ClonalMu == i]
    prob <- prob * PclonalLam^m
  }
  prob <- log(prob)
  return(prob)
}

LL_meanNb_nb.df <- function(df, listClonal, lambda_values, R0, mu_values,
                            maxMuGen, maxFS, maxIni, k) {
  prob <- 1:length(mu_values) %>%
    mclapply(function(x) {
      1:length(lambda_values) %>%
        lapply(function(y) {
          LL_lambda_nb(df, listClonal, lambda_values[y],
                       R0, mu_values[x], maxMuGen, maxFS, maxIni, k)
        })
    }, mc.cores = detectCores())
  names(prob) <- mu_values
  for (i in 1:length(mu_values)) {
    names(prob[[i]]) <- lambda_values
  }

  LLdf <- expand.grid(lambda = paste0("", lambda_values), mu = paste0("", mu_values))
  LLdf$prob <- unlist(prob)
  LLdf$lambda <- as.numeric(paste(LLdf$lambda))
  LLdf$mu <- as.numeric(paste(LLdf$mu))
  LLdf$meanN <- 0
  LLdf$meanNb <- 0

  for (j in lambda_values) {
    # calculate mean N
    dfPn <- pmf.ini_size_nb(j, R0, maxIni, k)
    dfPn <- dfPn[dfPn$type == "adjusted",]
    dfPn$prob_N <- dfPn$N * dfPn$prob
```

```
    mean_n <- sum(dfPn$prob_N)
    LLdf[which(near(LLdf$lambda, j)),]$meanN <- mean_n

    # calculate mean Nb
    mean_nb <- 0
    for (m in 1:maxIni) {
      PNb <- 0
      for (n in m:maxIni) {
        PNb <- PNb + dfPn$prob[n+1] * dbinom(m, n, 1-1/R0) / (1-dbinom(0, n, 1-1/R0))
      }
      mean_nb <- mean_nb + m * PNb
    }
    LLdf[which(near(LLdf$lambda, j)),]$meanNb <- mean_nb
  }

  return(LLdf)
}

# sample usage and plot
df_lambdaLL_nb <- LL_meanNb_nb.df(sample1, listClonal, seq(0.01, 5.01, by = 0.1),
                                  1.6, seq(0.01, 0.8, by = 0.02), 50, 50, 8, 0.5)
```




```
df_lambdaLL_nb[which(df_lambdaLL_nb$prob == max(df_lambdaLL_nb$prob)),]
```

```
##     lambda   mu      prob   meanN   meanNb
## 823   0.61 0.33 -82.92813 2.23664 1.315027
```

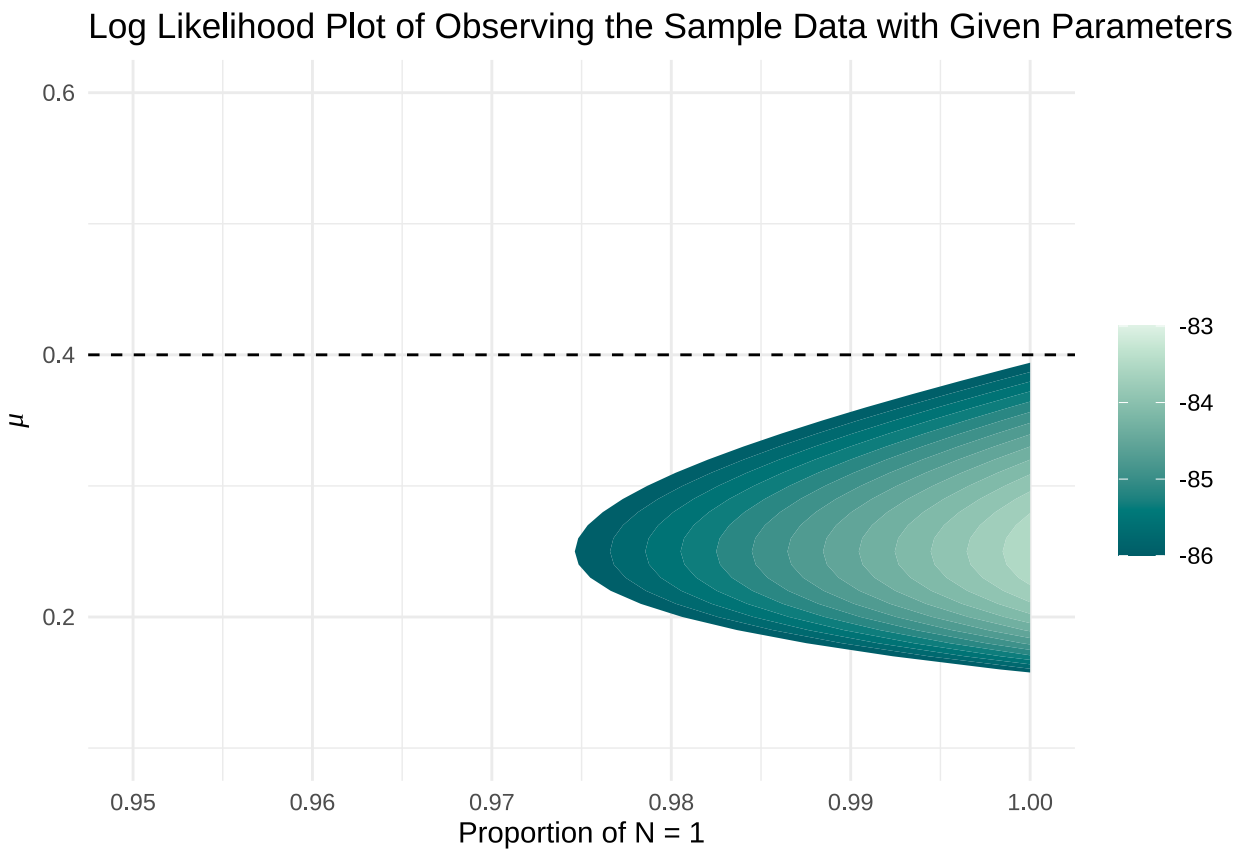
The dashed lines represents the actual parameters used to generate the sample data ($\lambda = 2.1, \mu = 0.4$). The estimated maximum likelihood estimators are $\lambda = 0.61, \mu = 0.33$.

8.3. Initial population distribution With only $N = 1$ or ∞

Another way to model the initial population distribution is to assume that there are some proportion of $N = 1$ and all others count as infinity and are ignored in the calculation process.

```
LL_prop <- function(df, listClonal, prop, R0, mu, maxMuGen, maxFS) {
  prob <- 1
  list <- mclapply(df$ClonalMu,
                   pmf.clonal_params,
                   listClonal = listClonal,
                   n_values = 1,
                   R0 = R0,
                   mu_values = mu,
                   maxMuGen = maxMuGen,
                   maxFS = maxFS,
                   mc.cores = detectCores())
  names(list) <- df$ClonalMu

  for (i in df$ClonalMu) {
    loc <- which(names(list) == i)
    dfClonalN <- list[[loc]]
    PclonalLam <- prop*dfClonalN$prob
    k <- df$freq[df$ClonalMu == i]
    prob <- prob * PclonalLam^k
  }
  prob <- log(prob)
  return(prob)
}


LL_prop.df <- function(df, listClonal, prop_values, R0,
                       mu_values, maxMuGen, maxFS, maxIni) {
  prob <- 1:length(mu_values) %>%
    mclapply(function(x) {
      1:length(prop_values) %>%
        lapply(function(y) {
          LL_prop(df, listClonal, prop_values[y],R0, mu_values[x], maxMuGen, maxFS)
        })
    }, mc.cores = detectCores())
  names(prob) <- mu_values
  for (i in 1:length(mu_values)) {
    names(prob[[i]]) <- prop_values
  }

  LLdf <- expand.grid(prop = paste0("", prop_values), mu = paste0("", mu_values))
  LLdf$prob <- unlist(prob)
  LLdf$prop <- as.numeric(paste(LLdf$prop))
  LLdf$mu <- as.numeric(paste(LLdf$mu))
  return(LLdf)
}
```

```r
# sample usage and plot
listClonal_prop <- list_clonal(1, 1.6, seq(0.1, 0.6, by = 0.01), 50, 50, 8)
df_prop_nb <- LL_prop.df(sample1, listClonal_prop, seq(0.9, 1, by = 0.01),
                         1.6, seq(0.1, 0.6, by = 0.01), 50, 50, 8)
```



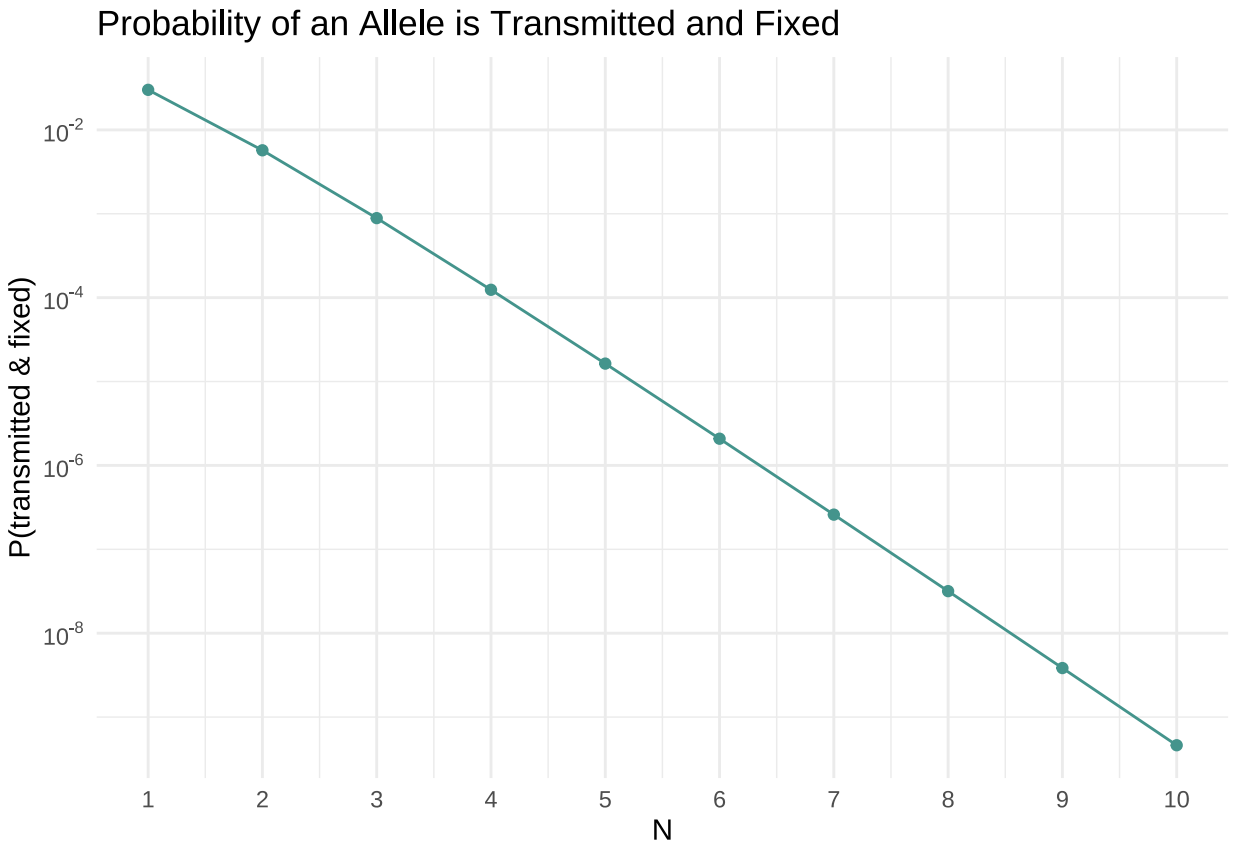
9. Source of Fixed Allele

In empirical data, we determine the clonal mutations based on variant-calling threshold $\alpha\%$ (e.g. 0.5%, 3%, or 7%). If an allele is below $\alpha\%$ or above $1-\alpha\%$ we will call this site monomorphic. With this in mind, if a site is monomorphic in both donor and receptor but with different alleles, we will call this a *de novo* clonal variant. However, one might argue that an allele below variant-calling threshold might be transmitted and eventually fix in the receptor. To address this question, we can calculate the probability that a fixed allele is transmitted.

Suppose that the allele frequency in donor is $q\%$, then the probability that k numbers of this allele is transmitted to the N initial viral particles in receptor is given by $Binom(k, N, q\%)$. For the allele to fix, all particles that don't bear this allele should extinct and at least one particle in these k particles should establish. In other words,

$$P(transmitted \cap fixed) = \sum_{k=1}^{N} Binom(k, N, q\%) \times (\frac{1}{R_0})^{N-k} \times (1 - Binom(0, k, 1 - \frac{1}{R_0}))/(1 - P_X)$$

```
P_transAndFix <- function(q, R0, maxIni) {
  freq <- q/100
  df <- data.frame(N = 1:maxIni, prob = 0)
  for (N in 1:maxIni) {
    P_trans_fix <- 0
    for (k in 1:N) {
      P_trans_fix <- P_trans_fix + dbinom(k, N, freq) * (1/R0)^(N-k) * (1-dbinom(0, k, 1-1/R0))
    }
    P_trans_fix <- P_trans_fix/(1-Px_analytical(N, R0))
    df$prob[N] <- P_trans_fix
  }
  return(df)
}

# sample usage and plot
df_P_trans <- P_transAndFix(3, 11.1, 10)
```


Probability of an Allele is Transmitted and Fixed
P(transmitted & fixed)
10^{-2}
10^{-4}
10^{-6}
10^{-8}
1
2
3
4
5
6
7
8
9
10
N

References

Nishiura, Hiroshi, Ping Yan, Candace K. Sleeman, and Charles J. Mode. 2012. "Estimating the Transmission Potential of Supercritical Processes Based on the Final Size Distribution of Minor Outbreaks." *Journal of Theoretical Biology* 294 (February): 48–55. https://doi.org/10.1016/j.jtbi.2011.10.039.