Multiple-visit occupancy and N-mixture models

Point count data analysis workshop 2025

Péter Sólymos

2025-11-12

Table of contents

Preamble

```r
suppressPackageStartupMessages({
    library(dplyr)
    library(ggplot2)
    library(unmarked)
})
set.seed(1234)
```

Introduction to simulations

The goal is to implement

- a data generating mechanisms
- using random numbers

Here is the most basic simulation: the coin flip.

We assume that the coin is fair, therefore:

- the probability of getting head ($y = 1$) is $p = 0.5$
- the probability of getting tail ($y = 0$) is $1 - p = 0.5$

Here is code for setting the probability value p and getting the outcome y using Uniform random numbers:

1. We generate a random number (`u`) between 0 and 1
2. The outcome is 1 if the number is $< p$
3. The outcome is 0 if the random number of $\geq p$

```r
p <- 0.5

u <- runif(n = 1, min = 0, max = 1)
u
```

```
[1] 0.1137034
```

```r
y <- ifelse(u < p, 1, 0)
y
```

```
[1] 1
```

Reproducibility with random numbers

Every time you run the code above, you'll get a different value for `u` and possibly a different outcome. To make this reproducible, we can set the random seed with e.g. `set.seed(123)`.

The `runif()` function takes 3 arguments:

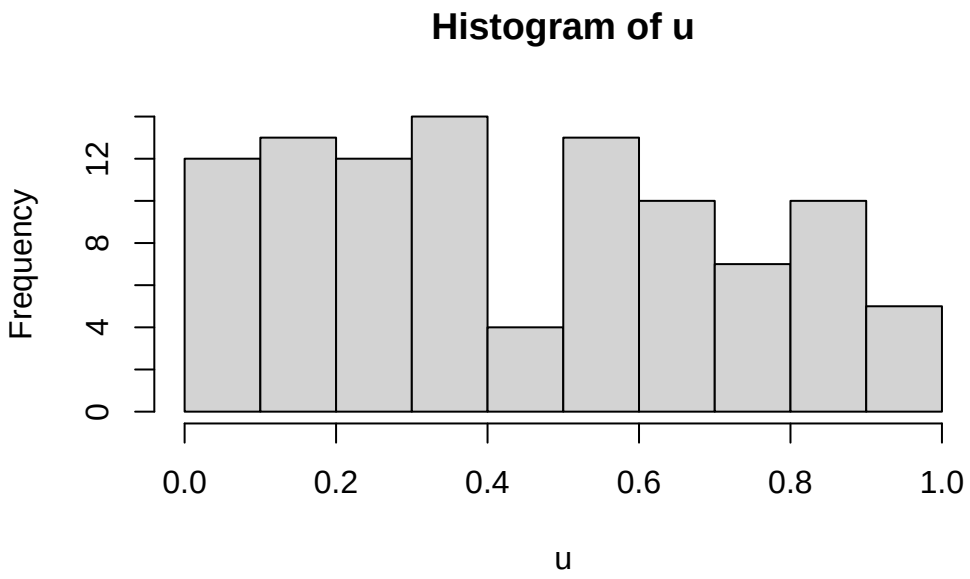
- `n` is the number of trials, i.e. number of coin flips
- `min` and `max` define the range, here we used the unit range $0 - 1$

Here is the histogram of a $Uniform(0, 1)$ random variable:

```
u <- runif(n = 100, min = 0, max = 1)
summary(u)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.009496 0.199980 0.386883 0.436715 0.666687 0.992150
```

```
hist(u)
```



We use the random numbers to repeat the coin flip experiment 100 times and tabulate the results:

```r
ifelse(u < p, 1, 0) |>
    table(y = _) |>
    as.data.frame() |>
    mutate(Prop = Freq / sum(Freq))
```

```
  y Freq Prop
1 0   45 0.45
2 1   55 0.55
```

We will see uses of the Uniform distribution later.

Another way to run the coin flip experiment in R is to use the Bernoulli distribution.

```r
y <- rbinom(n = 100, size = 1, prob = p)
summary(y)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.00    1.00    0.53    1.00    1.00
```

```r
table(y = y) |>
    as.data.frame() |>
    mutate(Prop = Freq / sum(Freq))
```

```
  y Freq Prop
1 0   47 0.47
2 1   53 0.53
```

The `rbinom()` function takes 3 arguments:

- `n` is the number of observations
- `size` is the number of trials
- `prob` is the probability of the outcome to be 1

The `rbinom` name refers to the Binomial distribution, which is multiple independent Bernoulli trials. This is what the `size` argument refers to, i.e. `size = 1` means Bernoulli.

Simulating occupancy data

Let's see how we can use the Bernoulli distribution and the `rbinom` function to simulate occupancy data.

Let us sample $n = 50$ locations and observe the occupancy status of Ovenbirds at each of the locations:

- The true occupancy status at the ith location is W_i.
- The probability of occupancy is $P(W_i = 1) = \phi$.

The true occupancy probability $ `is denoted with`phi.true‘:

```
n <- 50
phi.true <- 0.4

W <- rbinom(n = n, size = 1, prob = phi.true)

table(W = W)
```

```
W
 0  1
32 18
```

There are many reasons why we might not observe the true occupancy status W. When the data that we collect (Y) is not the true occupancy status W, we say that there is detection error. It can manifest the following way:

We can capture detection error also as a probability p which is the probability of detecting the species when present $(W_i = 1)$.

- If the species is absent $(W_i = 0)$, we will always observe 0, thus $Y_i = 0$: $P(Y_i = 0|W_i = 0) = 1$.
- If $W_i = 1$, we might observe 0 (missed all the birds) or 1:
  - species detected when present, $P(Y_i = 1|W_i = 1) = p$
  - species not detected when present, $P(Y_i = 0|W_i = 1) = 1 - p$

Let's simulate what we observe, the `Y` vector. We use the `rbinom()` function with `n = n` as before, but the size argument is not a fixed 1, but the true status `W`. When `W` is 0, the function will always return 0, when `W` is 1, the function will reyrun 1 or 0 according the the detection probability p, denoted as `p.true`:

```r
p.true <- 0.6
Y <- rbinom(n = n, size = W, prob = p.true)

table(W = W, Y = Y)
```

```
   Y
W    0  1
  0 32  0
  1  2 16
```

The cross-tabulation show how many times we observed 1's vs 0's when the true status was 1.

Using the observations as if those would represent the true status is called the *naive* approach. The naive approach assume that there is no observation error:

```r
mean(Y)
```

```
[1] 0.32
```

But we see that the mean of `Y` is quite far from `phi.true`.

Fitting a logistic regression (aka Binomial GLM) to the data yields the same result:

```r
m1 <- glm(Y ~ 1, family = binomial)
coef(m1)
```

```
(Intercept)
 -0.7537718
```

```r
plogis(coef(m1))
```

```
(Intercept)
       0.32
```

The coefficient for the intercept is the maximum likelihood estimate on the logit scale. To transform that to the $0-1$ probability scale, we can use the `plogis()` function that implements the inverse logistic transformation.

The unobserved `W` variable is often called a *latent* variable. It is latent in the sense that it cannot be directly observed. We need to find ways to:

- use survey design to minimize the detection error, or
- use statistical models to correct for detection error, or
- the combination of the 2.

Simulating multiple visits

A feature of multiple-visit methods is that we visit the same site T times. Each visit ($t = 1, \ldots, T$) gives us a different observation, Y_{it}.

Key assumptions are the following:

- the visits are independent of each other
- the true status W stays the same over the visits ($W_i = W_{it}$)

We can generalize the simulation code to any number of `T` visits as:

```
T <- 5
Y <- matrix(NA, n, T)
for (t in 1:T) {
    Y[, t] <- rbinom(n = n, size = W, prob = p.true)
}
```

A more concise way of writing the above code without a loop is:

```
Y <- replicate(T, rbinom(n = n, size = W, prob = p.true))
```

If we inspect any row (site) from the `Y` matrix, we see a series of 1's and 0's for sites where the species is present. This is called the *detection history*.

```
data.frame(
    W = head(W[W > 0]),
    Visit = head(Y[W > 0, ])
)
```

```
  W Visit.1 Visit.2 Visit.3 Visit.4 Visit.5
1 1       0       1       1       1       1
2 1       1       0       1       1       1
3 1       0       0       0       1       0
4 1       1       0       1       1       1
5 1       0       1       0       1       1
6 1       1       0       0       0       1
```

The detection history is all 0's for sites where the species is absent:

```
data.frame(
    W = head(W[W == 0]),
    Visit = head(Y[W == 0, ])
)
```

```
  W Visit.1 Visit.2 Visit.3 Visit.4 Visit.5
1 0       0       0       0       0       0
2 0       0       0       0       0       0
3 0       0       0       0       0       0
4 0       0       0       0       0       0
5 0       0       0       0       0       0
6 0       0       0       0       0       0
```

We can compare the maximum of the observed values over the visits at each site:

```
Y_max <- apply(Y, 1, max)
table(W = W, Y_max = Y_max)
```

```
   Y_max
W    0  1
  0 32  0
  1  0 18
```

Fit the logistic regression model to `Y_max`:

```
m2 <- glm(Y_max ~ 1, family = binomial)
coef(m2)
```

```
(Intercept)
 -0.5753641
```

```
plogis(coef(m2))
```

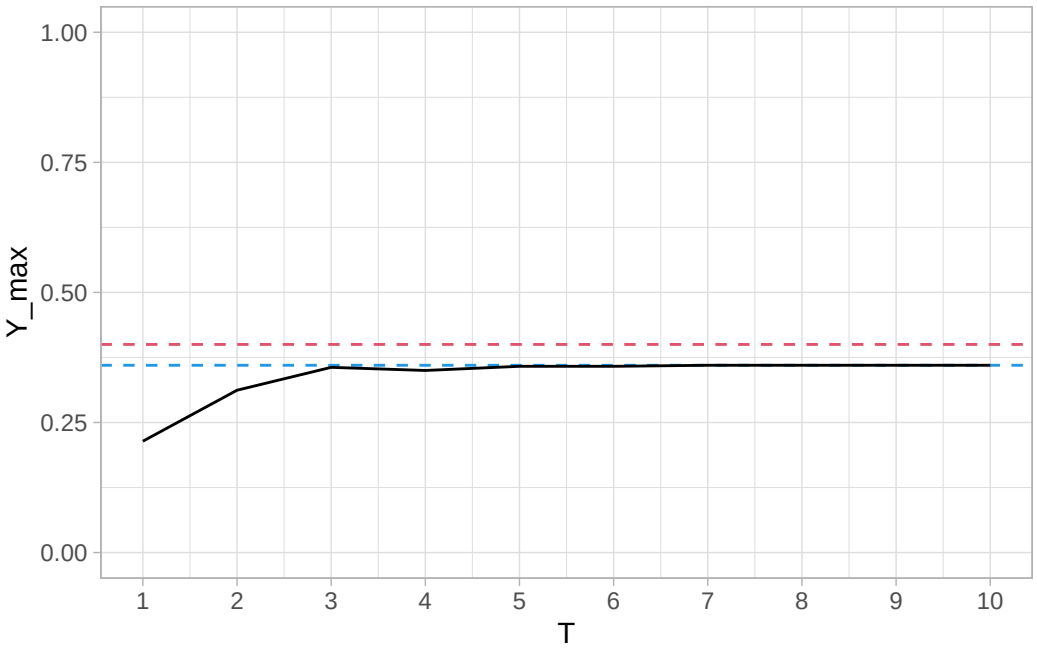
```
(Intercept)
       0.36
```

What happens when we increase the number of visits?

Change the value of `T` and compare the `Y_max` to `W`. What happens to our *naive* estimator of using `Y_max`?

The maximum over a large number of visits will approach the latent variable `W`:

```
n2 <- 500
W2 <- rbinom(n = n2, size = 1, prob = phi.true)
Tvals <- 1:10

data.frame(
    T = Tvals,
    Y_max = sapply(Tvals, \(T) {
        mean(apply(
            replicate(T, rbinom(n = n2, size = W, prob = p.true)),
            1, max
        ))
    })
) |> ggplot(aes(x = T, y = Y_max)) +
    geom_hline(yintercept = phi.true, lty = 2, col = 2) +
    geom_hline(yintercept = mean(W), lty = 2, col = 4) +
    geom_line() +
    ylim(0, 1) +
    scale_x_continuous(breaks = Tvals) +
    theme_light()
```



Fitting occupancy models

The unmarked package implements the multiple-visit occupancy model.

> References
>
> MacKenzie et al., 2002. Estimating site occupancy rates when detection probabilities are less than one. Ecology, 83:2248–2255. Fulltext, DOI 10.1890/0012-9658(2002)083[2248:ESORWD]2.0.CO;2

- Organize the data and an unmarked occupancy data frame
- Fit the model with the `occu`
- the `~1 ~1` formula says that we do not have covariates, just the intercepts

```
umf <- unmarked::unmarkedFrameOccu(y = Y)
summary(umf)
```

```
unmarkedFrame Object

50 sites
Maximum number of observations per site: 5
Mean number of observations per site: 5
```

```
Sites with at least one detection: 18

Tabulation of y observations:
  0   1
198  52
```

```r
m3 <- unmarked::occu(~1 ~ 1, umf)
m3
```

```
Call:
unmarked::occu(formula = ~1 ~ 1, data = umf)

Occupancy:
 Estimate    SE     z P(>|z|)
   -0.552 0.298 -1.86  0.0636

Detection:
 Estimate    SE    z P(>|z|)
    0.279 0.223 1.25   0.211

AIC: 191.4105
```

We use the `plogis()` function again to transform the estimates to the probability scale and compare with our `phi.true` and `p.true` values:

```r
plogis(coef(m3, type = "det"))
```

```
   p(Int)
0.5692042
```

```r
plogis(coef(m3, type = "state"))
```

```
 psi(Int)
0.3654202
```

Maximum likelihood estimator

The `glm()` and `occu()` functions use maximum likelihood to find the parameter estimates for a given data set. In other words, we coefficients (maximum likelihood estimate, or MLE)

maximize the likelihood function for the data set in question. The likelihood function can be relatively simple and easy to calculate, or it can be computationally challenging to compute (e.g. for hierarchical or mixed models).

The likelihood function L for the simple occupancy model with parameters p and ϕ for multiple visits data can be written as:

$$L(p, \phi; y_{1,1}, \ldots, y_{n,T}) = \prod_{i=1}^{n} \left[\phi \left(\binom{T}{y_{i\cdot}} p^{y_{i\cdot}} (1-p)^{T-y_{i\cdot}} \right) + (1-\phi) I(y_{i\cdot} = 0) \right]$$

where $y_{i\cdot} = \sum_{T}^{t=1} y_{i,t}$ and $I(y_{i\cdot} = 0)$ is an indicator function that is equal to 1 if $y_{i\cdot} = 0$.

Here is the R code to calculate the log likelihood for the occupancy model:

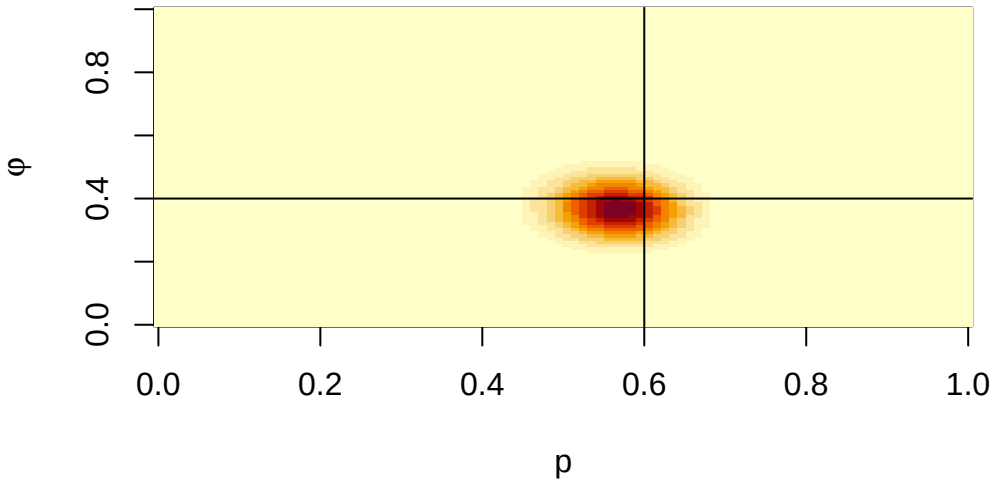
```
L_fun_occu <- function(Y, p, phi) {
    ydot <- rowSums(Y)
    T <- ncol(Y)
    L <- prod(
        phi *
            (choose(T, ydot) * p^ydot * (1 - p)^(T - ydot)) +
            (1 - phi) * (ydot == 0)
    )
    L
}
```

Next, we evaluate the likelihood function at different values of `p` and `phi` while keeping the data `Y` constant. We set up the grid for this using `expand.grid()`:

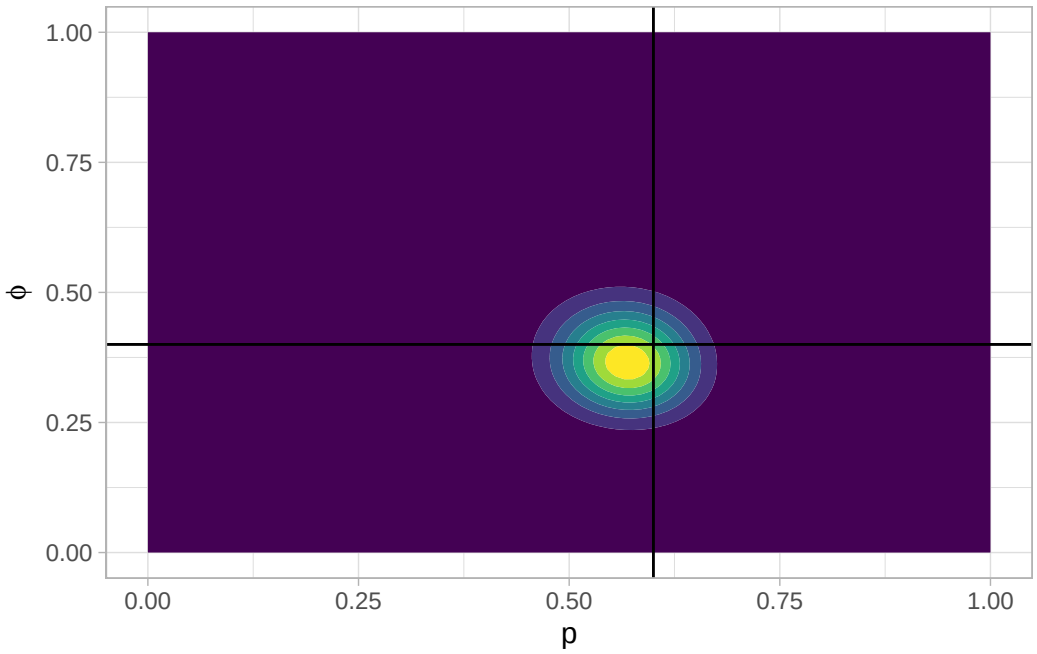
```
g <- 100
grid <- expand.grid(
    p = seq(0, 1, length.out = g),
    phi = seq(0, 1, length.out = g),
    L = NA
)

for (i in 1:nrow(grid)) {
    grid$L[i] <- L_fun_occu(
        Y = Y,
        p = grid$p[i],
        phi = grid$phi[i]
    )
}
```

When we plot the likelihood surface, we see the maximum, the lines indicate the true probability values:

```
image(
    list(
        x = unique(grid$p),
        y = unique(grid$phi),
        z = matrix(grid$L, g, g)
    ),
    xlab = "p",
    ylab = expression(varphi)
)
abline(h = phi.true, v = p.true, col = 1, lwd = 1)
```




```
grid |> ggplot(aes(x = p, y = phi, z = L)) +
    geom_contour_filled(show.legend = FALSE) +
    geom_hline(yintercept = phi.true) +
    geom_vline(xintercept = p.true) +
    xlab("p") +
    ylab(expression(phi)) +
    theme_light()
```




```
grid[which.max(grid$L), ]
```

```
             p       phi             L
3657 0.5656566 0.3636364 3.925967e-26
```

If the rgl package is installed, we can view the surface in 3D:

```
if (interactive()) {
    library(rgl)

    L_mat <- matrix(grid$L, g, g)
    dcpal_grbu <- colorRampPalette(c("#18bc9c", "#3498db"))
    Col <- rev(dcpal_grbu(12))[cut(L_mat, breaks = 12)]

    open3d()
    persp3d(
        x = unique(grid$p),
        y = unique(grid$phi),
        z = L_mat / max(L_mat),
        col = Col,
        theta = 50, phi = 25, expand = 0.75, ticktype = "detailed",
        xlab = "p", ylab = expression(phi), zlab = "L"
```

```
    )
    quads3d(
        x = rep(p.true, 4),
        y = c(0, 0, 1, 1),
        z = c(0, 1, 1, 0),
        alpha = 0.5, col = 2
    )
    quads3d(
        x = c(0, 0, 1, 1),
        y = rep(phi.true, 4),
        z = c(0, 1, 1, 0),
        alpha = 0.5, col = 4
    )
}
```

> **Note**
>
> We will circle back to these plots later, feel free to explore it with different settings of n, T, p, and ϕ.

Simulating count data

Simulating counts is similar to occupancy. We need a count distribution. The most basic count distribution is Poisson which has 1 parameter, λ, which is the mean. The variance also happens to equal the mean. Use the `rpois()` function in R to generate random numbers from the poisson distribution. Here, N_i $(i = 1, \ldots, n)$ is the abundance (number of individuals) at the ith location.

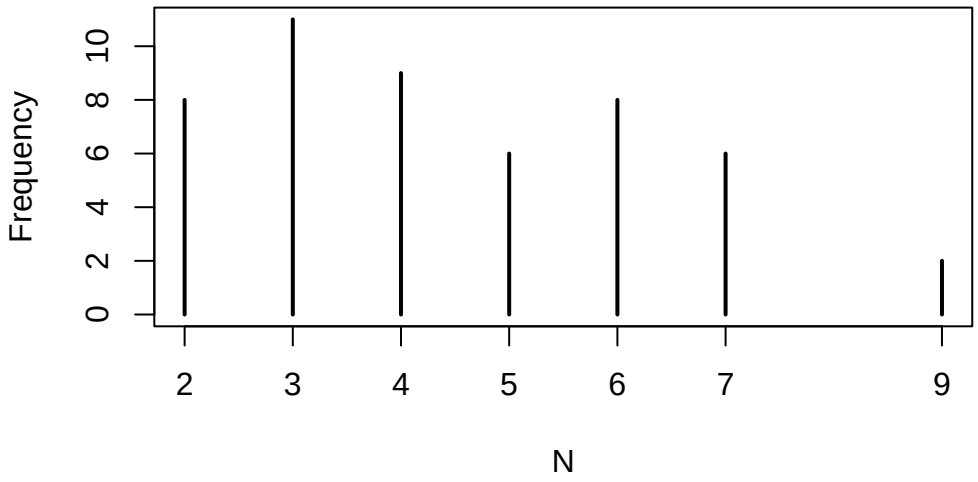
```
lambda.true <- 4.2
N <- rpois(n = n, lambda = lambda.true)

table(N)
```

```
N
 2  3  4  5  6  7  9
 8 11  9  6  8  6  2
```

```
summary(N)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.00    3.00    4.00    4.46    6.00    9.00
```

```
table(N) |> plot(ylab = "Frequency")
```



Observation error for counts

Here is how we introduce observation error to count data:

- select a location,
- take the count N_i,
- for each individual $(1, 2, \ldots, N_i)$, use the Bernoulli distribution with probability p to determine if the individual was detected (1) or not (0)

The number of individuals detected (Y_i) is less than or equal to N_i: $Y_i \leq N_i$. For example, if $N_i = 4$, Y_i can be 0, 1, 2, 3, or 4.

To express things mon concisely, we can also use a Binomial distribution with N_i as the number of trials and probability p:

```
Y <- rbinom(n = n, size = N, prob = p.true)

print(table(N = N, Y = Y), zero.print = ".")
```

```
   Y
N   0 1 2 3 4 5 6
  2 . 4 4 . . . .
```

```
  3 . 4 4 3 . . .
  4 1 1 1 5 1 . .
  5 . . 1 2 1 2 .
  6 . . 1 2 3 1 1
  7 . . . 1 2 1 2
  9 . . . . . 1 1
```

Count data with multiple visits

When visiting each site T times, the observed data will be organized in a n by T matrix as how we saw it for occupancy:

```
Y <- matrix(NA, n, T)
for (t in 1:T) {
    Y[, t] <- rbinom(n = n, size = N, prob = p.true)
}

# alternatively
# Y <- replicate(T, rbinom(n = n, size = N, prob = p.true))
```

Multiple-visit count model have similar assumptions as occupancy models, most importantly that $N_{it} = N_i$. This condition is also referred to as the closed population assumption, i.e. there is no immigration, emigration, birth, or death between visits.

```
data.frame(N = N, Visit = Y) |> head()
```

```
  N Visit.1 Visit.2 Visit.3 Visit.4 Visit.5
1 6       5       1       4       5       3
2 6       5       4       5       3       4
3 4       3       2       2       2       3
4 7       4       3       4       5       4
5 6       3       3       4       5       4
6 2       2       0       1       1       2
```

As we saw before, we can use the `Y_max` to fit the naive count model, but instead of the `plogis()` function, we use `log()` as the inverse of the logarithmic link used for Poisson GLM:

```
Y_max <- apply(Y, 1, max)
m4 <- glm(Y_max ~ 1, family = poisson)
coef(m4)
```

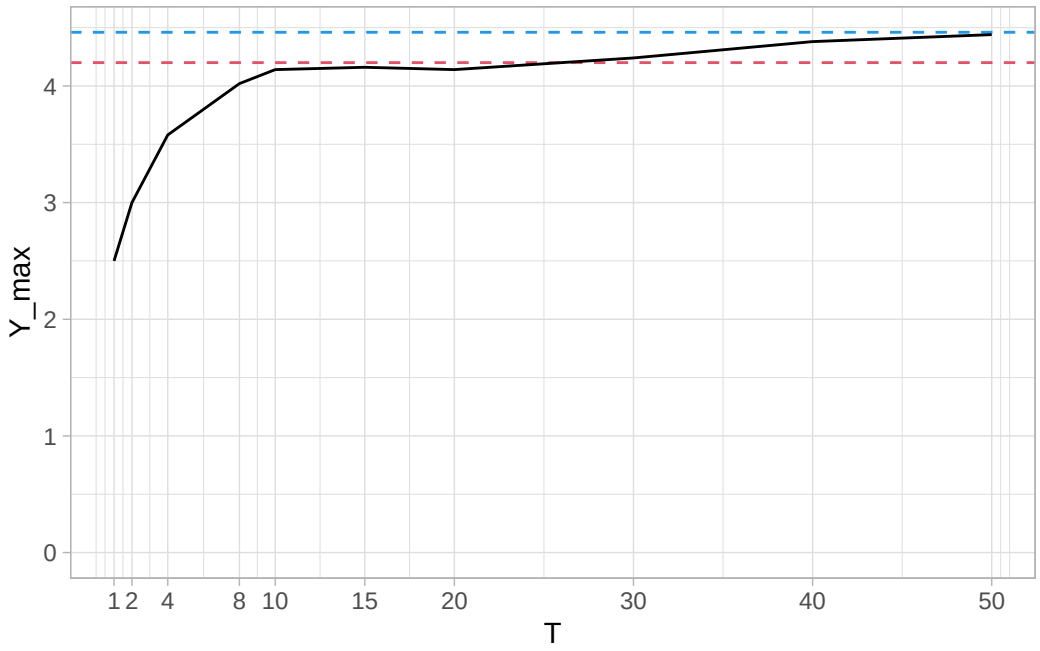
```
(Intercept)
   1.329724
```

```
exp(coef(m4))
```

```
(Intercept)
       3.78
```

With enough visits, tha `Y_max` will approach the latent variable `N`:

```
N2 <- rpois(n = n2, lambda = lambda.true)
Tvals <- c(1, 2, 4, 8, 10, 15, 20, 30, 40, 50)

data.frame(
    T = Tvals,
    Y_max = sapply(Tvals, \(T) {
        mean(apply(
            replicate(T, rbinom(n = n, size = N, prob = p.true)),
            1, max
        ))
    })
) |> ggplot(aes(x = T, y = Y_max)) +
    geom_hline(yintercept = lambda.true, lty = 2, col = 2) +
    geom_hline(yintercept = mean(N), lty = 2, col = 4) +
    geom_line() +
    ylim(0, NA) +
    scale_x_continuous(breaks = Tvals) +
    theme_light()
```



Fitting N-mixture models

Fit the multiple-visit count model, the so called N-mixture model, using the `pcount()` function of the unmarked R package.

The `unmarked::unmarkedFramePCount()` function is used to organize the count data:

```
umf <- unmarked::unmarkedFramePCount(y = Y)
summary(umf)
```

```
unmarkedFrame Object

50 sites
Maximum number of observations per site: 5
Mean number of observations per site: 5
Sites with at least one detection: 50

Tabulation of y observations:
 0  1  2  3  4  5  6  7
17 43 66 59 37 17  7  4
```

```r
m5 <- unmarked::pcount(~1 ~ 1, umf, K = 50)
m5
```

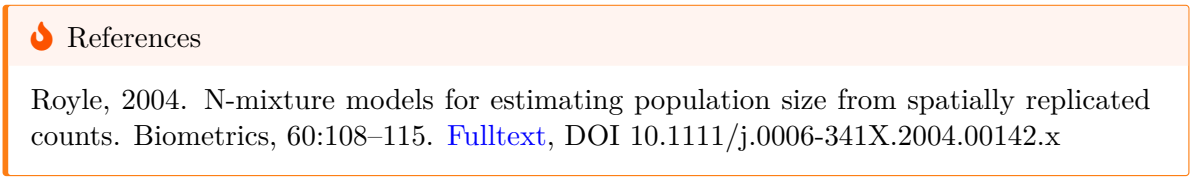
```
Call:
unmarked::pcount(formula = ~1 ~ 1, data = umf, K = 50)

Abundance:
 Estimate     SE    z  P(>|z|)
     1.53 0.0917 16.7 8.92e-63

Detection:
 Estimate    SE    z P(>|z|)
    0.262 0.159 1.65  0.0989

AIC: 816.689
```

```r
plogis(coef(m5, type = "det"))
```

```
   p(Int)
0.5650955
```

```r
exp(coef(m5, type = "state"))
```

```
lam(Int)
4.636385
```



The `K` argument of the `pcount()` function is the upper index of integration for N-mixture. This will make more sense once we take a look at the likelihood function in the next section.

N-mixture likelihood

The likelihood function for the N-mixture model can be written as:

$$L(p, \lambda; y_{1,1}, \dots, y_{n,T}) = \prod_{i=1}^{n} \sum_{N_i=Y_{max,i}}^{K} \prod_{t=1}^{T} e^{-\lambda_i} \frac{\lambda_i^{N_i}}{N_i!} \binom{N_i}{Y_{it}} p^{Y_{it}} (1-p)^{N_i - Y_{it}}$$

This distribution is a mixture of Poisson and Binomial distributions.

The K value strictly speaking should be ∞, but the estimates won't change much as long as K is large enough relative to the maximum of N_i. Of course for real data we do not know what the values of N_i are. The numerical integration (summation) for site i goes from $Y_{max,i}$ (`Y_max`) to K, because we know that $Y_{it} \leq N_i$.

We can write this as a log likelihood function:

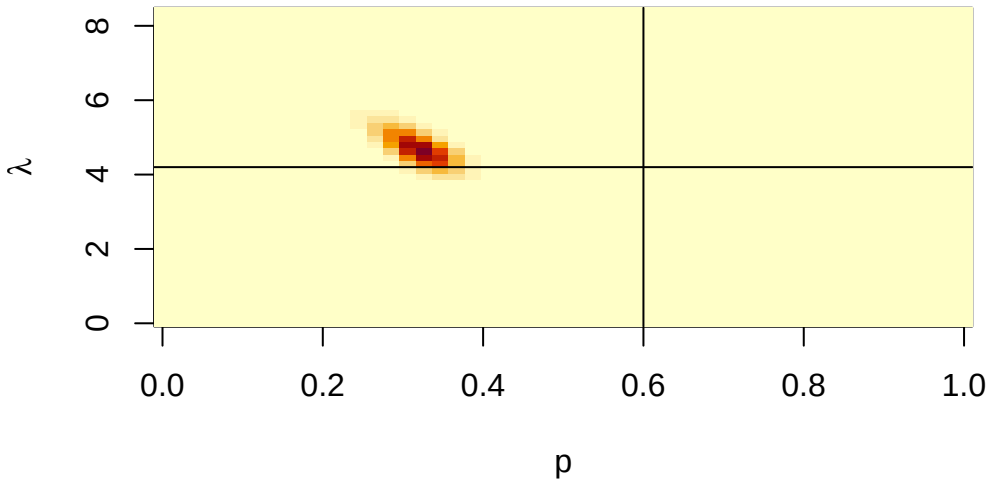
```
logL_fun_pcount_R <- function(Y, p, lambda, n, T, Y_max, K) {
    L <- rep(NA, n)
    for (i in 1:n) {
        S <- 0
        for (Nit in Y_max[i]:K) {
            v <- 0
            for (j in 1:T) {
                v <- v + dbinom(Y[i, j], Nit, p, log = TRUE) +
                    dpois(Nit, lambda, log = TRUE)
            }
            S <- S + exp(v)
        }
        L[i] <- S
    }
    sum(log(L))
}

# using C code from unmarked - much faster
logL_fun_pcount_C <- function(Y, p, lambda, n, T, Y_max, K = 50) {
    nll <- unmarked:::nll_pcount(
        beta = c(log(lambda), qlogis(p)),
        n_param = c(1, 1, 0),
        y = Y,
        X = matrix(1, n, 1),
        V = matrix(1, n * T, 1),
        X_offset = rep(0, n),
        V_offset = matrix(1, n * T, 1),
        K = K,
        Kmin = Y_max,
        mixture = 1,
```

```
        threads = 1
    )
    -nll
}
```

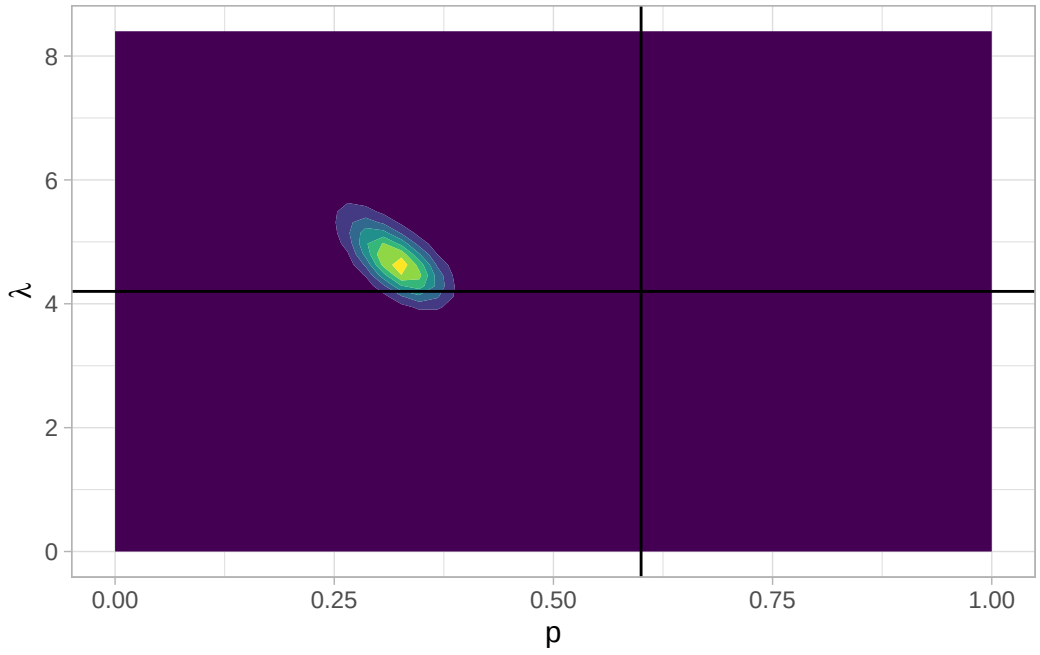
```
g <- 50
grid <- expand.grid(
    p = seq(0, 1, length.out = g),
    lambda = seq(0, lambda.true * 2, length.out = g),
    logL = NA
)

logL_fun_pcount <- logL_fun_pcount_C
for (i in 1:nrow(grid)) {
    grid$logL[i] <- logL_fun_pcount(
        Y = Y,
        p = grid$p[i],
        lambda = grid$lambda[i],
        n = n, T = T, Y_max = Y_max, K = 50
    )
}
grid$logL[is.infinite(grid$logL)] <- NA
```

The images showing the likelihood surface:

```
image(
    list(
        x = unique(grid$p),
        y = unique(grid$lambda),
        z = matrix(exp(grid$logL), sqrt(nrow(grid)))
    ),
    xlab = "p",
    ylab = expression(lambda)
)
abline(h = lambda.true, v = p.true, col = 1, lwd = 1)
```




```
grid |> ggplot(aes(x = p, y = lambda, z = exp(logL))) +
    geom_contour_filled(show.legend = FALSE) +
    geom_hline(yintercept = lambda.true) +
    geom_vline(xintercept = p.true) +
    xlab("p") +
    ylab(expression(lambda)) +
    theme_light()
```



```r
grid[which.max(grid$logL), ]
```

```
             p   lambda      logL
1367 0.3265306 4.628571 -406.3499
```

> **Note**
>
> We will circle back to these plots later, feel free to explore it with different settings of n, T, p, and λ.

Next

Introduction to agent-based simulations

End of Day 1. See you tomorrow!