Bayesian inference of an individual-based mutualistic network

02__02

Net 02__02

```r
library(BayesianNetworks)
library(network.tools)
library(tidyverse)
theme_set(theme_minimal())
options(mc.cores = 4)
```

Data

Load dataset and sampling effort per individual plant:

```r
web <- readr::read_csv(here::here("data/nets_raw", paste0(params$net, "_int.csv"))) |>
  arrange(ind)
```

```
## Rows: 35 Columns: 11
## -- Column specification ------------------------------------------------------------------------
## Delimiter: ","
## chr  (1): ind
## dbl (10): Turdus_philomelos, Vulpes_vulpes, Turdus_merula, Erithacus_rubecula, Turdus_iliacus, Sylvia
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
mat <- as.matrix(web[, -1])
mat <- apply(mat, c(1,2), as.integer)
rownames(mat) <- web$ind


# create numeric vector of sampling effort for each plant with names = plant id
effort <- readr::read_csv(here::here("data/nets_attr", paste0(params$net, "_attr.csv"))) |>
  select(ind, starts_with("se_")) |>
  filter(ind %in% web$ind) |>
  arrange(ind)
```

```
## Rows: 35 Columns: 31
## -- Column specification ------------------------------------------------------------------------
## Delimiter: ","
## chr  (4): ind, fruit_type, fruit_color, area
## dbl (27): canopy_cover_m2, height_cm, crop, d1_cm, d2_cm, x, y, fruit_d1_mm, fruit_d2_mm, fruit_mass_
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## If there is only one column with sampling effort, use it:
if (!net %in% c("01_01", "01_02", "02_01", "02_02", "02_03",
                "10_01", "15_01", "18_01", "18_02", "20_01", "21_01", "21_02")) {
  effort <- effort |>
    pull(starts_with("se_"), name = "ind")
}

# Otherwise, select sampling effort column in some specific nets:
if (net == "10_01") {
  effort <- effort |>
    mutate(se_cam_days = se_cam_h/24) |>
    pull(se_cam_days, name = "ind")
}

if (net == "15_01") {
  effort <- effort |>
    pull(se_cam_days, name = "ind")
}

if (net %in% c("18_01", "18_02", "20_01")) {
  effort <- effort |>
    mutate(se_bc_months = se_bc_days/30) |>
    pull(se_bc_months, name = "ind")
}

if (net %in% c("21_01", "21_02")) {
  effort <- effort |>
    pull(se_obs_h, name = "ind")
}

# For Pistacia and Juniperus, use constant sampling effort
if (net %in% c("01_01", "01_02", "02_01", "02_02", "02_03")) {
  effort <- rep(10, nrow(mat))
  names(effort) <- web$ind
}

## Some nets may require adjusting of the count data or effort values
# if (net %in% c("01_01", "01_02", "02_01", "02_02", "02_03")) {
#   mat <- mat/10  # divide counts by 10 to make modelling feasible
#   mat[mat > 0 & mat < 1] <- 1  # don't miss rare counts
#   mat <- round(mat)
#   mat <- apply(mat, c(1,2), as.integer)
# }

stopifnot(identical(length(effort), nrow(mat)))
stopifnot(identical(names(effort), rownames(mat)))

# summary(mat)
summary(as.numeric(mat))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0     0.0   116.1    57.5  2502.0
```

```
# if (max(mat) > 1000) {
#   stop("More than 1000 counts in some cell(s)")
# }

summary(effort)
```

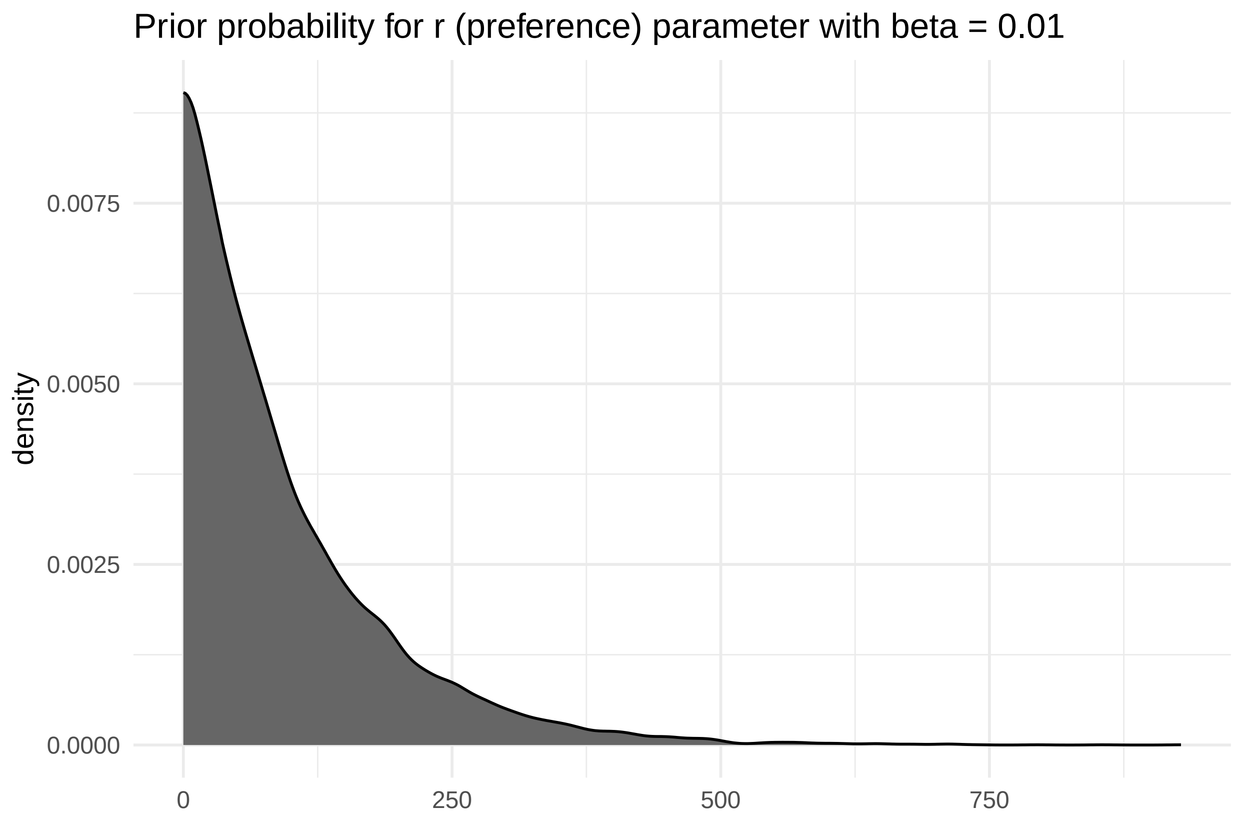
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      10      10      10      10      10      10
```

```
if (max(effort) > 500) {
  stop("Sampling effort > 500 for some plants")
}
```

Bayesian inference of network structure

```
dt <- prepare_data(mat, sampl.eff = effort)

plot_prior(params$beta)
```



```r
fit <- fit_model(dt,
                 refresh = 0,
                 beta = params$beta,
                 model = params$model,
                 # max_treedepth = 15,
                 # init = function() list(r = runif(1, 0, 20000)),
                 iter_warmup = params$iter,
                 iter_sampling = params$iter,
                 thin = 4 * params$iter / 1000)
```

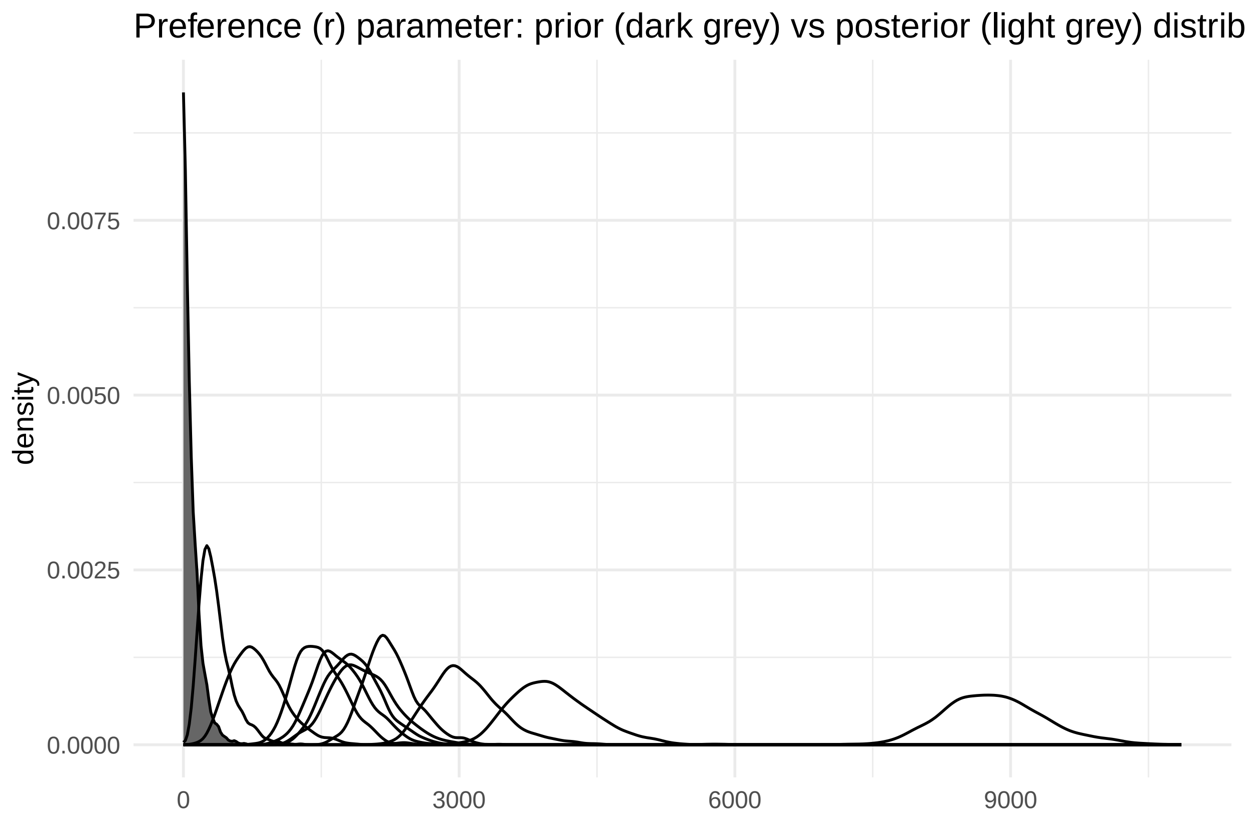
```
## Running MCMC with 4 parallel chains...
##
## Chain 3 finished in 56.2 seconds.
## Chain 4 finished in 57.0 seconds.
## Chain 1 finished in 57.5 seconds.
## Chain 2 finished in 60.3 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 57.8 seconds.
## Total execution time: 60.4 seconds.
```

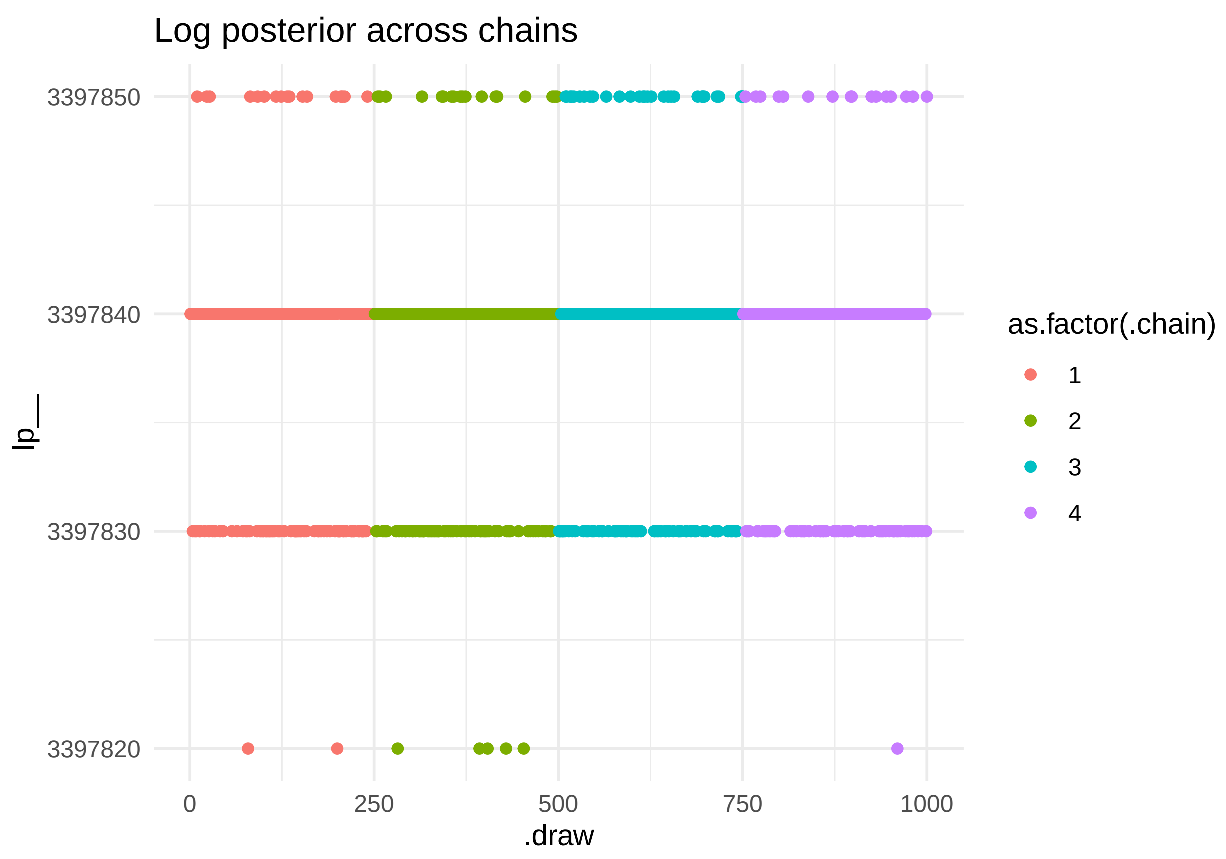
```r
get_seed(fit)
```

```
## [1] 1132120531
```

```r
check_model(fit, data = dt)
```

```
## Processing csv files: C:/Users/frodr/AppData/Local/Temp/Rtmp4CoIrr/varying_preferences-202406251126-
##
## Checking sampler transitions treedepth.
## Treedepth satisfactory for all transitions.
##
## Checking sampler transitions for divergences.
## No divergent transitions found.
##
## Checking E-BFMI - sampler transitions HMC potential energy.
## E-BFMI satisfactory.
##
## Effective sample size satisfactory.
##
## Split R-hat values satisfactory all parameters.
##
## Processing complete, no problems detected.
```


Preference (r) parameter: prior (dark grey) vs posterior (light grey) distrib
density
0.0000
0.0025
0.0050
0.0075
0
3000
6000
9000



Posteriors

Get posterior distributions:

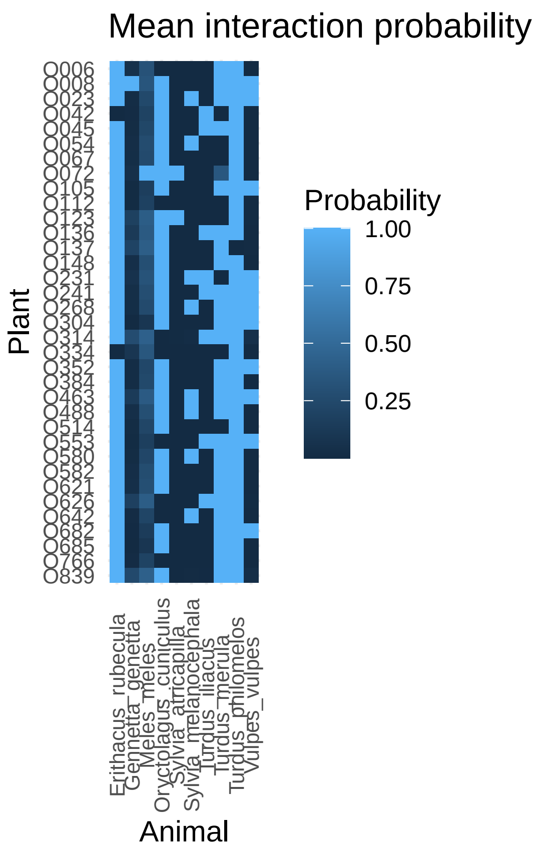
```
post <- get_posterior(fit, dt)

head(post)
```

```
## # A tibble: 6 x 11
## # Groups:   Animal, Plant [6]
##   Plant Animal        .chain .iteration .draw connectance preference plant.abund animal.abund int.pr
##   <chr> <chr>          <int>      <int> <int>       <dbl>      <dbl>       <dbl>        <dbl>    <db
## 1 0006  Turdus_philo~      1          1     1       0.449      8770.      0.0180        0.336
## 2 0008  Turdus_philo~      1          1     1       0.449      8770.      0.0143        0.336
## 3 0023  Turdus_philo~      1          1     1       0.449      8770.      0.0306        0.336
## 4 0042  Turdus_philo~      1          1     1       0.449      8770.      0.0406        0.336
## 5 0045  Turdus_philo~      1          1     1       0.449      8770.      0.0334        0.336
## 6 0054  Turdus_philo~      1          1     1       0.449      8770.      0.0250        0.336
```

Mean edge probability:

```
plot_interaction_prob(post)
```

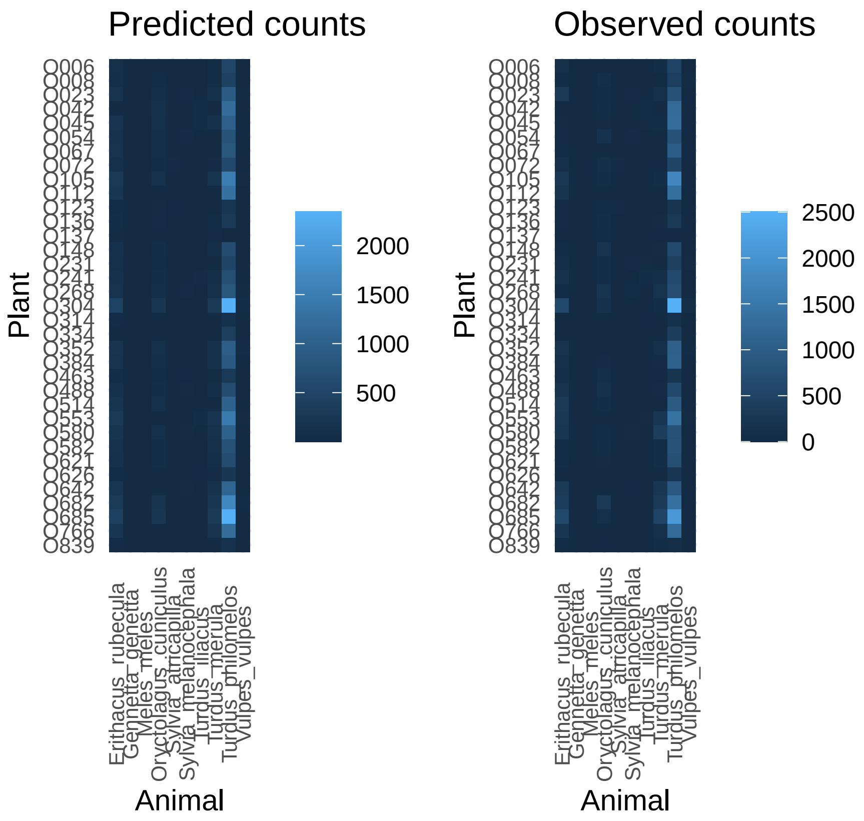


Generate predicted visits for each pairwise interaction

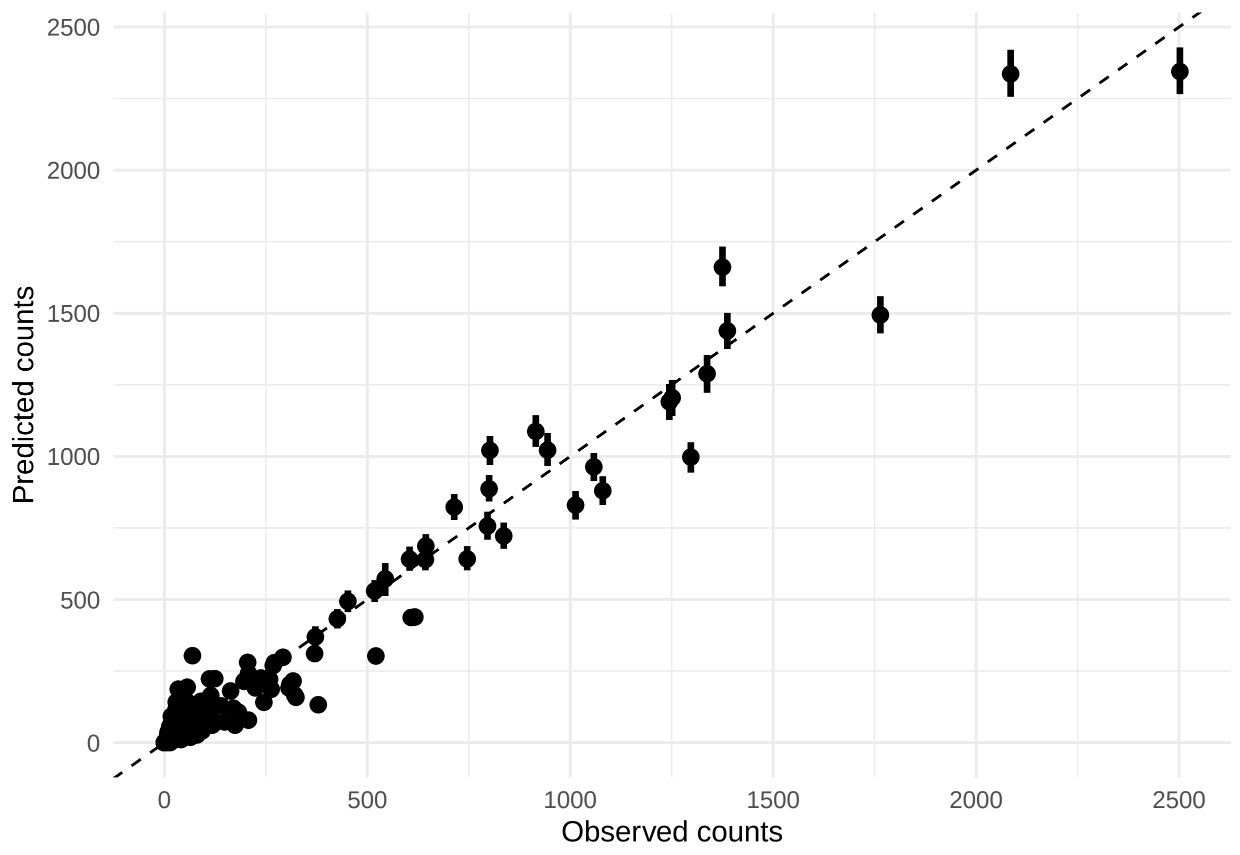
```r
post.counts <- predict_counts(fit, dt)
```

Compare observed and predicted visits by the model:

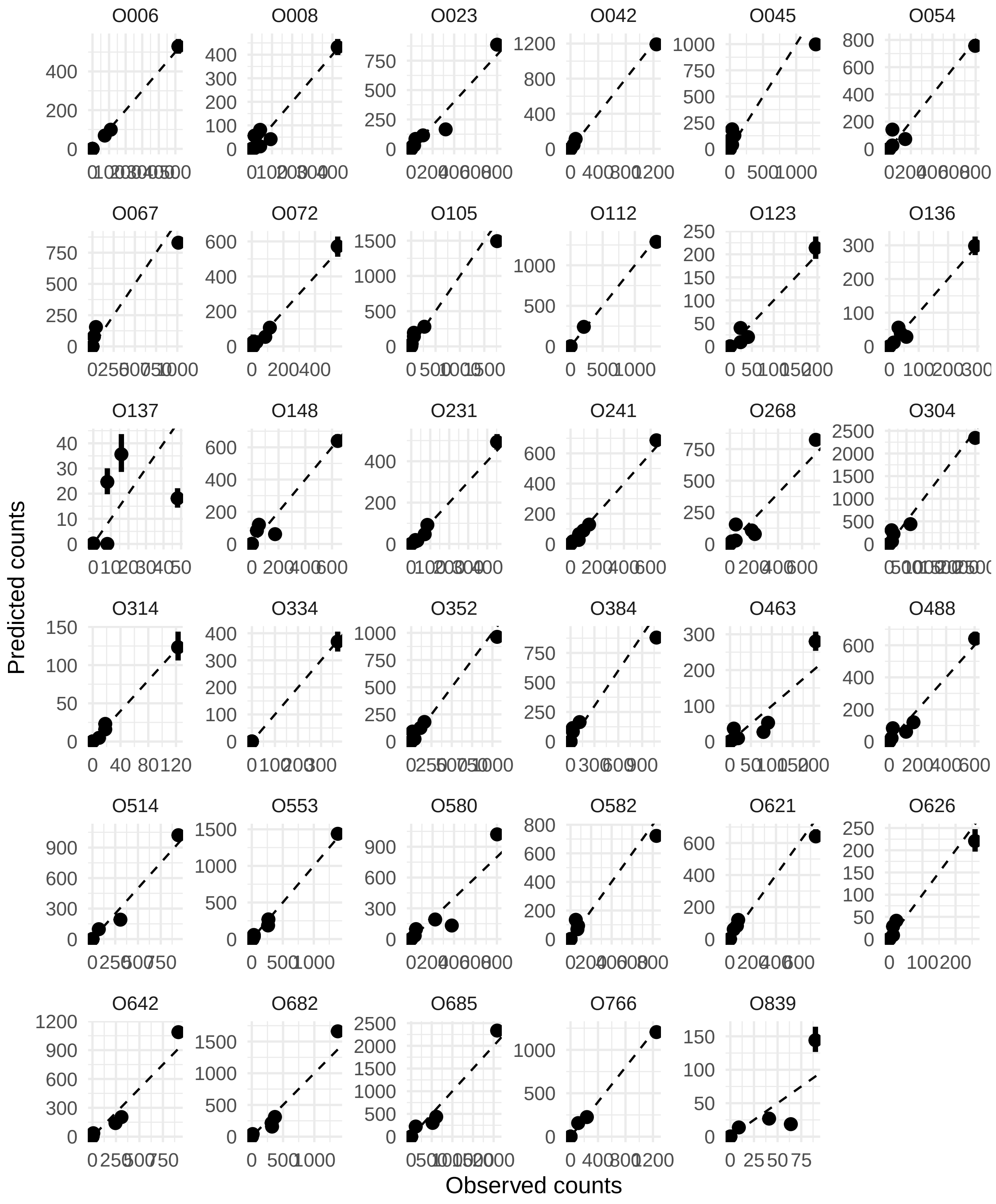
```r
p <- plot_counts_pred(post.counts, sort = FALSE)
o <- plot_counts_obs(mat, sort = FALSE, zero.na = FALSE)
library(patchwork)
p + o
```



```
plot_counts_pred_obs(post.counts, dt)
```



```r
plot_counts_pred_obs(post.counts, dt, byplant = TRUE, scales = "free")
```



```r
saveRDS(post.counts, here::here(paste0("data/nets_post/", params$net, "_post_counts.rds")))
```