Lab 9

```
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2
```

```
## rstan (Version 2.21.3, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
library(rstanarm)
```

```
## Warning: package 'rstanarm' was built under R version 4.1.2
```

```
## Loading required package: Rcpp
```

```
## This is rstanarm version 2.21.3
```

```
## - See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!
```

```
## - Default priors may change, so it's safest to specify priors, even if equivalent to the defaults.
```

```
## - For execution on a local, multicore CPU with excess RAM we recommend calling
```

```
##   options(mc.cores = parallel::detectCores())
```

```
##
## Attaching package: 'rstanarm'
```

```
## The following object is masked from 'package:rstan':
##
##     loo
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::extract() masks rstan::extract()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
```

```
library(bayesplot)
```

```
## Warning: package 'bayesplot' was built under R version 4.1.2
```

```
## This is bayesplot version 1.9.0
```

```
## - Online documentation and vignettes at mc-stan.org/bayesplot
## - bayesplot theme set to bayesplot::theme_default()
##    * Does _not_ affect other ggplot2 plots
##    * See ?bayesplot_theme_set for details on theme setting
```

```
library(rethinking)
```

```
## Loading required package: cmdstanr
## This is cmdstanr version 0.5.1.9000
## - CmdStanR documentation and vignettes: mc-stan.org/cmdstanr
## - CmdStan path: /Users/alexziyujiang/.cmdstan/cmdstan-2.29.2
## - CmdStan version: 2.29.2
## Loading required package: parallel
## rethinking (Version 2.21)
##
## Attaching package: 'rethinking'
## The following object is masked from 'package:purrr':
##
##     map
## The following objects are masked from 'package:rstanarm':
##
##     logit, se
## The following object is masked from 'package:rstan':
##
##     stan
## The following object is masked from 'package:stats':
##
##     rstudent
```

```
library(loo)
```

```
## This is loo version 2.4.1
## - Online documentation and vignettes at mc-stan.org/loo
## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar
##
## Attaching package: 'loo'
## The following object is masked from 'package:rethinking':
##
##     compare
## The following object is masked from 'package:rstan':
##
##     loo
```

Models

Warmup on GLMs

GLMs use link functions to connect non-continuous outcomes to linear predictors. Consider the linear model:

$$y_i \sim N(\alpha + \beta x_i, \sigma), \mu_i = \alpha + \beta x_i$$

We see that μ_i, which we also call the **linear predictor**, is a linear combination of the covariates (in this case, just one covariate) $\alpha + \beta x_i$, and is the expected value of the outcome y_i:

$$E[y_i \mid x_i] = \mu_i$$

The reformulation above is interesting because it tells us the following information:

- Using this model, we are expecting a linear relationship between **predictors** (x_i) and the **expected value** ($E[y_i \mid x_i]$)
- An 'implied' assumption about the model is that y_i should be a continuos variable on the **whole real line**. This is justified by the relationship between the linear predictor μ_i and the expected outcome $E[y_i \mid x_i]$ above.

The Poisson model

The Poisson model is useful for modeling data that takes integer values. **Especially, data that does not have (a known) upper limit**. For a discrete distribution with support on the integer values $\{0, 1, 2, ...\}$, the Poisson distribution seems like a really good choice. This gives us the incentive to use it to model integer data:

$$y_i \sim \text{Poisson}(\lambda_i), \lambda_i \text{'associated with'} x_i$$

Here I use λ_i because ultimately we want to fit a regression model. So it makes sense that the parameter of Poisson distribution is different for each y_i, and that is governed by difference in the covariate x_i. We then see how to characterize this association.

For Poisson distribution, a very important result is that the parameter λ_i is the expected value of the outcome y_i and also the expected variance of the counts y_i. i.e.

$$E[y_i \mid x_i] = \lambda_i, \lambda_i \text{'associated with'} x_i$$

This is one step closer, but we still need to know how λ_i gets associated with x_i. We already knew that $\lambda_i > 0$, but x_i is unrestricted on the whole real line. So it makes sense to 'transform' λ_i so that the transformed value is also on the whole real line. One thing we can do is to use the log function:

$$\log \lambda_i = \alpha + \beta x_i$$

This completes our model. Note the relationship to the regular linear model, where the relationship between the linear predictor and the expected outcome is an identical functiom:

$$E[y_i \mid x_i] = \alpha + \beta x_i$$

where, in our case, it is a log function. The log function here is also called **link function**.

$$\log(E[y_i \mid x_i]) = \alpha + \beta x_i$$

There are a few things to notice here:

- The log link ensures that λ_i is always positive, which is required of the expected value of a count outcome.
- It also implies an exponential relationship between predictors and the expected value. (We need to justify such expoential relationship actually makes sense.)
- The log function is not the only function that works as a link function for the Poisson model, but this is how people usually model count data as a common practice.
- Finally, it also makes sense to center the linear predictor first: $\log(E[y_i \mid x_i]) = \alpha + (\beta x_i - \bar{x})$

Data

We then look at a data example. The data is adapted from Gelman and Hill(2007) and is about pest treatment for roaches in urban apartments. The dataset is used to study the efficacy of a certain pest management system at reducing the number of roaches in urban apartments. The treatment and control were applied to 160 and 104 apartments, respectively, and the outcome measurement in each apartment was the number of roaches caught in a set of traps. Different apartments had traps for different numbers of days.

- `y`(the response variable): number of roaches caught
- `roach1`(covariate): pre-treatment number of roaches
- `treatment`(covariate): a binary variable indicating whether there is treatment
- `senior`(covariate): a binary variable indicating whether the building is for senior residents
- `exposure2`(offset variable): number of days for which traps were used (different for all apartments)

The goal here is to model the number of roaches caught (technically the number of roaches caught per day) for each urban apartment, given the data we have. Some extra points to notice:

- **rescaling data**: the original scale for `roach1` is very large so it needs rescaling due to reasons we covered before. Here what is done in the reference is that they divided the original figures by 100 – we do the same thing here.
- **offset**: the traps were placed for different time periods, so it makes more sense to model the 'capture rate' (i.e. counts divided by time), so we use this log-offset of exposure time length variable in the model, with coefficient fixed to 1. **You can think of offset as a variable in the regression function, but the coefficient is fixed to 1**.

We load in the data here:

```
data("roaches", package = "rstanarm")
roaches <- as_tibble(roaches)
glimpse(roaches)
```

```
## Rows: 262
## Columns: 5
## $ y         <int> 153, 127, 7, 7, 0, 0, 73, 24, 2, 2, 0, 21, 0, 179, 136, 104,~
## $ roach1    <dbl> 308.00, 331.25, 1.67, 3.00, 2.00, 0.00, 70.00, 64.56, 1.00, ~
## $ treatment <int> 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ senior    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ exposure2 <dbl> 0.8000000, 0.6000000, 1.0000000, 1.0000000, 1.1428571, 1.000~
```

```
covariates <-
  roaches %>%
  mutate(roach1 = roach1/100) %>%
  # mutate(roach1 = scale(roach1)[, 1]) %>% # this is an option too!
  dplyr::select(roach1, treatment, senior)

d <- list(
  y = roaches$y,
```

```
  X = as.matrix(covariates),
  offset = log(roaches$exposure2)
)

d$N <- length(d$y)
d$K <- ncol(d$X)
```

The model has the following form:

$$
\begin{aligned}
y_i &\sim \text{Poisson}(\lambda_i) \\
\log(\lambda_i) &= \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \log(\text{Exposure}_i) \\
\alpha &\sim N(0, 2.5), \beta_i \sim N(0, 10), i = 1, .., 3
\end{aligned}
$$

We compile the model in stan. The major difference from what we have done before in the linear models is that here the linear predictor `lambda` is associated with the linear predictor `a + X * b + offset` through an exponential function:

```
stan_code <- "
data {
  // number of observations
  int N;
  // response
  int<lower = 0> y[N];
  // number of covariates
  int K;
  // design matrix X
  matrix[N, K] X;
  // an offset is a term with known coefficient 1
  vector[N] offset;
}
parameters {
  real a;
  // regression coefficient vector
  vector[K] b;
}
transformed parameters {
  vector<lower = 0> [N] lambda;
  lambda = exp(a + X * b + offset);
}
model {
  a ~ normal(0, 10);
  b ~ normal(0, 2.5);
  // likelihood
  y ~ poisson(lambda);
}
generated quantities {
  // simulate data from the posterior
  vector[N] y_rep;
  // log-likelihood posterior
  vector[N] log_lik;
  for (i in 1:N) {
    y_rep[i] = poisson_rng(lambda[i]);
```

```
    log_lik[i] = poisson_lpmf(y[i] | lambda[i]);
  }
}
"

fit_poisson <- stan(model_code = stan_code, data = d)
```

```
## Warning in '/var/folders/42/k2fwbxpn19j8ysn3rvw51_h00000gn/T/Rtmpi3U4ho/model-bbf26cce1ba9.stan', li
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/42/k2fwbxpn19j8ysn3rvw51_h00000gn/T/Rtmpi3U4ho/model-bbf26cce1ba9.stan', li
##     name 'offset' will be a reserved word starting in Stan 2.32.0. Please
##     rename it!
## Warning in '/var/folders/42/k2fwbxpn19j8ysn3rvw51_h00000gn/T/Rtmpi3U4ho/model-bbf26cce1ba9.stan', li
##     name 'offset' will be a reserved word starting in Stan 2.32.0. Please
##     rename it!
```

```
## Running MCMC with 1 chain...
##
## Chain 1 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1 finished in 0.5 seconds.
```

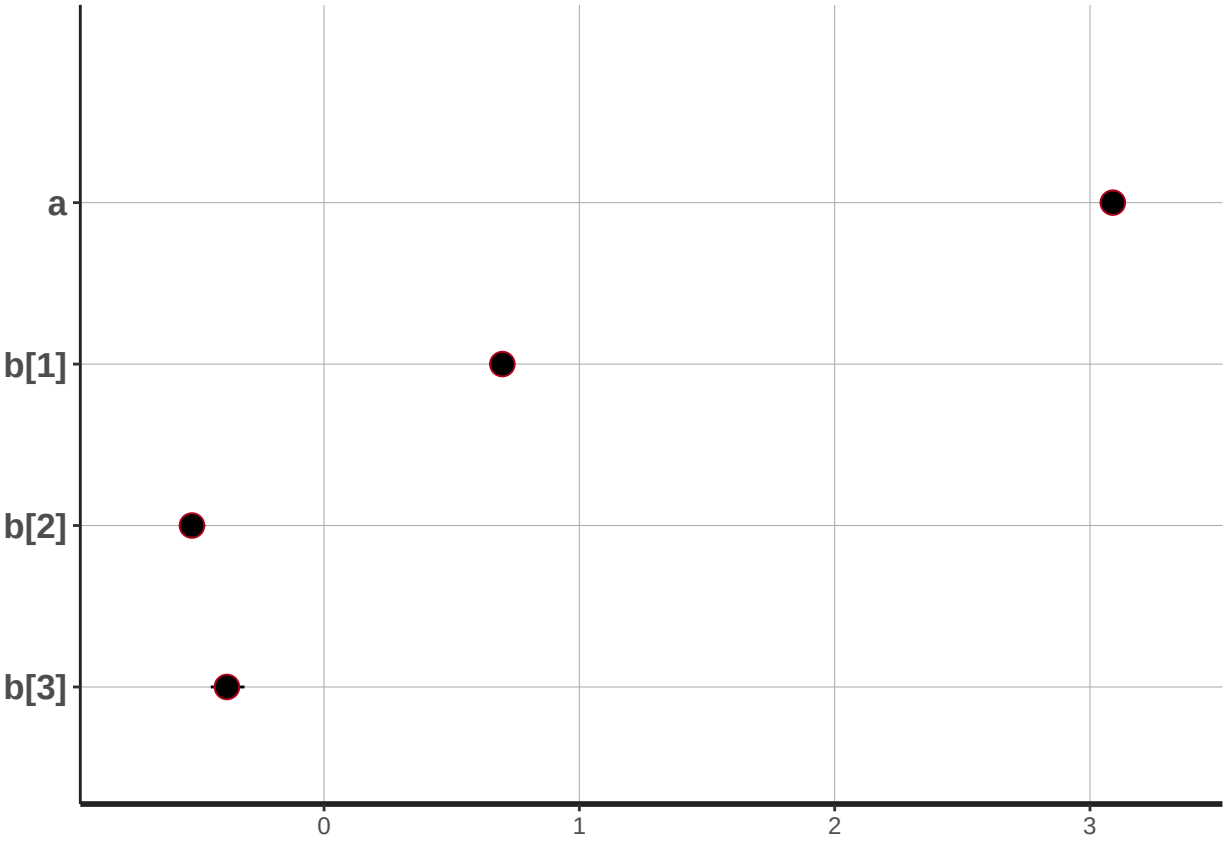
```
print(fit_poisson, pars = c("a", "b"))
```

```
## Inference for Stan model: rt_cmdstanr_cd92411a556e7dd97e569ab1a496d42e-202205261320-1-426fd0.
## 1 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=500.
##
##       mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## a     3.09       0 0.02  3.05  3.08  3.09  3.11  3.13   272 1.00
## b[1]  0.70       0 0.01  0.68  0.69  0.70  0.70  0.71   333 1.00
## b[2] -0.52       0 0.02 -0.57 -0.53 -0.52 -0.50 -0.47   368 1.01
## b[3] -0.38       0 0.03 -0.44 -0.40 -0.38 -0.35 -0.31   317 1.00
##
## Samples were drawn using NUTS(diag_e) at Thu May 26 13:20:53 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
plot(fit_poisson, pars = c("a", "b"))
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```



We can also do this in jags! Note that sometimes jags does not support vectorization as well as stan, so we have to be a little careful and better write all iterations.

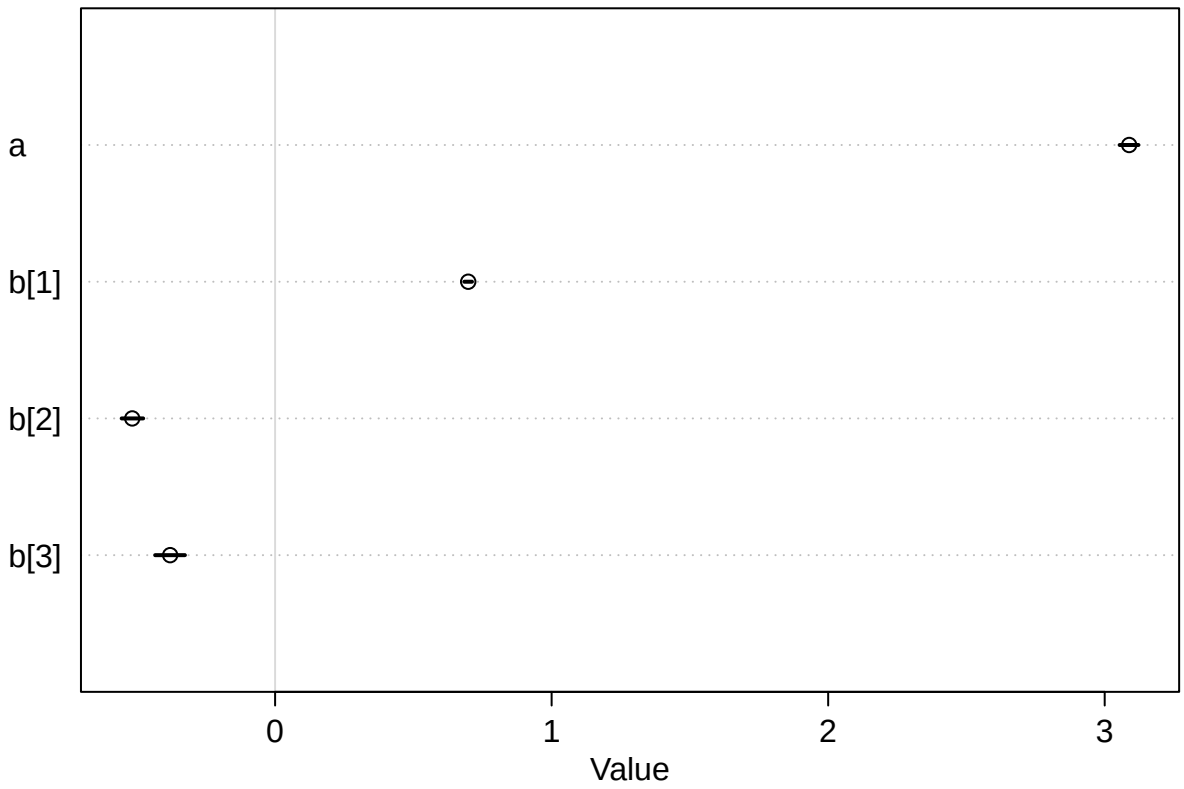
```
rjags_code <- " data {
  D <- dim(X)
  N <- D[1] # number of observations
  K <- D[2] # number of covariates
}
model {
  a ~ dnorm(0, pow(10,-2));
  for (i in 1:K) {
    b[i] ~ dnorm(0, pow(2.5,-2))
  }
  for (i in 1:N) {
    lambda[i] = exp(a + X[i,] %*% b + offset[i])
  }
  # likelihood
  for (i in 1:N) {
    y[i] ~ dpois(lambda[i])
    # posterior predictive
    ynew[i] ~ dpois(lambda[i])
  }
}
"
library(rjags)
```

```
## Loading required package: coda
##
## Attaching package: 'coda'
## The following object is masked from 'package:rstan':
##
##     traceplot
## Linked to JAGS 4.3.0
## Loaded modules: basemod,bugs
```

```
fit_poisson_jags <- jags.model(file = textConnection(rjags_code),
                          data = list(
                            X = d$X,
                            y = d$y,
                            offset = d$offset
                          ))
```

```
## Compiling data graph
##    Resolving undeclared variables
##    Allocating nodes
##    Initializing
##    Reading data back into data table
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 262
##    Unobserved stochastic nodes: 266
##    Total graph size: 2421
##
## Initializing model
```

```
fit_poisson_jags_sampls <- coda.samples(fit_poisson_jags,
                            variable.names = c("a", "b"),
                            n.iter = 1e4)
fit_poisson_jags_sampls_df <- as.data.frame(fit_poisson_jags_sampls[[1]])
plot(precis(fit_poisson_jags_sampls_df, depth = 2))
```




```
#print(fit_poisson, pars = c("a", "b"))
#plot(fit_poisson, pars = c("a", "b"))
```

A generalization of Poisson model - Negative Binomial

Now we look back again at the regression model we have been fitting. The poisson regression allows us to fit models to integer variables, but this is predicated on restriction of the Poisson distribution itself: Poisson distribution is a one-parameter distribution, so the flexibility we can achieve with it is very limited. Also, we know that $E[Y_i \mid X_i] = Var[Y_i \mid X_i] = \lambda_i$, which may not make much sense in some cases.

Negative binomial model allow us to relax the assumption by introducing an extra parameter.

$$y_i \sim \text{Neg-Bin}(\mu_i, \phi), \log \mu_i = \alpha + \beta x_i$$

There are two parameters in the negative binomial distribution: - μ_i (location/mean parameter): similar to λ_i in the Poisson regression model, $E[Y_i \mid X_i] = \mu_i$ - ϕ (dispersion parameter): a parameter that controls the variance: $Var[Y_i \mid X_i] = \mu_i + \mu_i^2/\phi$. If $\phi \to \infty$, the model converges to the Poisson model, for finite values of ϕ, it suggests a variance inflation of μ_i^2/ϕ compared to the Poisson model.

Finally, we see that the log link applies to here again. We write out the whole model:

$$\begin{aligned} y_i &\sim \text{Neg-Bin}(\mu_i, \phi) \\ \log(\lambda_i) &= \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \log(\text{Exposure}_i) \\ \alpha &\sim N(0, 2.5), \beta_i \sim N(0, 10) \\ \phi^{-1} &\sim \text{Exp}(1) \end{aligned}$$

- **How to choose prior for the overdispersion parameter?**: We want the negative binomial model to 'cover' poisson model, in the sense that we want the model to accomodate when there is no overdispersion in the data. Again, when there is no overdispersion, $\phi \to \infty$. So we want a fair part of prior support on ϕ close to positive infinity. However, that would be really hard to do! A cleverer way is to place a prior on ϕ^{-1} instead of ϕ, so that we can find a prior that has good support near zero for ϕ^{-1} – an exponential distribution may just do that.

```
stan_code <- "
data {
  // number of observations
  int N;
  // response
  int<lower = 0> y[N];
  // number of covariates
  int K;
  // design matrix X
  matrix[N, K] X;
  // an offset is a term with known coefficient 1
  vector[N] offset;
}
parameters {
  // regression coefficients
  real a;
  vector[K] b;
  // will put prior on 1 / sqrt(phi)
  real<lower = 0.> sqrt_phi_reciprocal;
}
transformed parameters {
  real<lower = 0.> phi;
  vector<lower = 0.>[N] mu;
  phi = 1. / sqrt_phi_reciprocal^2;
  mu = exp(a + X * b + offset);
}
model {
  a ~ normal(0., 10.);
  b ~ normal(0., 2.5);
  // putting prior on 1 / sqrt(phi) allows shrinking to no dispersion
  sqrt_phi_reciprocal ~ exponential(1);
  // likelihood
  y ~ neg_binomial_2(mu, phi);
}
generated quantities {
  // simulate data from the posterior
  vector[N] y_rep;
  vector[N] log_lik;
  // log-likelihood posterior
  for (i in 1:N) {
    y_rep[i] = neg_binomial_2_rng(mu[i], phi);
    log_lik[i] = neg_binomial_2_lpmf(y[i] | mu[i], phi);
  }
}
"
fit_nb <- stan(model_code = stan_code, data = d)
```

```
## Warning in '/var/folders/42/k2fwbxpn19j8ysn3rvw51_h00000gn/T/Rtmpi3U4ho/model-bbf221db61a7.stan', li
```

```
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/42/k2fwbxpn19j8ysn3rvw51_h00000gn/T/Rtmpi3U4ho/model-bbf221db61a7.stan', lin
##     name 'offset' will be a reserved word starting in Stan 2.32.0. Please
##     rename it!
## Warning in '/var/folders/42/k2fwbxpn19j8ysn3rvw51_h00000gn/T/Rtmpi3U4ho/model-bbf221db61a7.stan', lin
##     name 'offset' will be a reserved word starting in Stan 2.32.0. Please
##     rename it!
## Running MCMC with 1 chain...
##
## Chain 1 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1 Informational Message: The current Metropolis proposal is about to be rejected because of the
## Chain 1 Exception: neg_binomial_2_lpmf: Precision parameter is 0, but must be positive finite! (in '/
## Chain 1 If this warning occurs sporadically, such as for highly constrained variable types like covar
## Chain 1 but if this warning occurs often then your model may be either severely ill-conditioned or mi
## Chain 1
## Chain 1 Informational Message: The current Metropolis proposal is about to be rejected because of the
## Chain 1 Exception: neg_binomial_2_lpmf: Precision parameter is 0, but must be positive finite! (in '/
## Chain 1 If this warning occurs sporadically, such as for highly constrained variable types like covar
## Chain 1 but if this warning occurs often then your model may be either severely ill-conditioned or mi
## Chain 1
## Chain 1 Informational Message: The current Metropolis proposal is about to be rejected because of the
## Chain 1 Exception: neg_binomial_2_lpmf: Location parameter[1] is inf, but must be positive finite! (i
## Chain 1 If this warning occurs sporadically, such as for highly constrained variable types like covar
## Chain 1 but if this warning occurs often then your model may be either severely ill-conditioned or mi
## Chain 1
## Chain 1 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1 Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1 finished in 1.0 seconds.
```

```
print(fit_nb, pars = c("a", "b", "phi"))
```

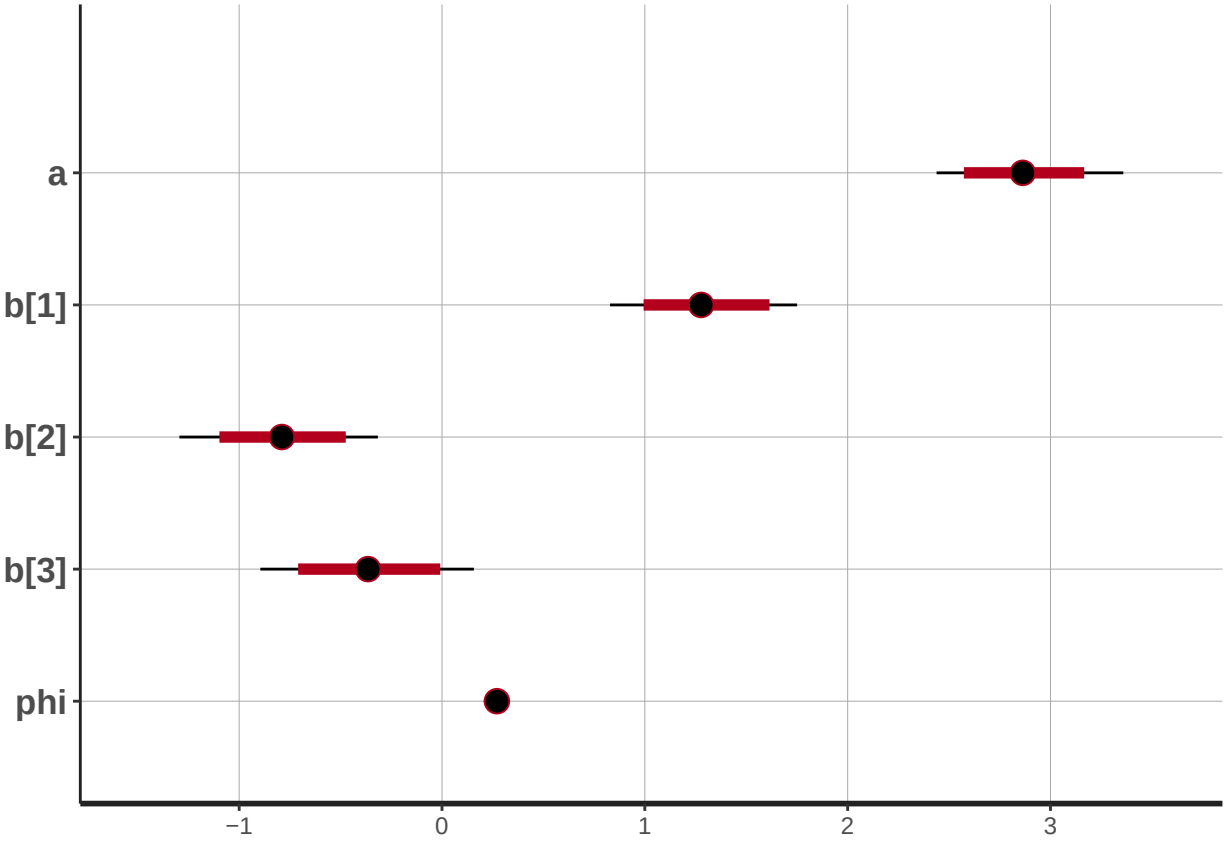
```
## Inference for Stan model: rt_cmdstanr_946b96d1a0384804a7e4c7ee7982f3e8-202205261321-1-49ce56.
## 1 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=500.
```

```
##
##       mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## a     2.87    0.02 0.24  2.44  2.69  2.86  3.03  3.36   203    1
## b[1]  1.29    0.01 0.25  0.83  1.12  1.28  1.46  1.75   350    1
## b[2] -0.79    0.02 0.25 -1.29 -0.93 -0.79 -0.62 -0.32   238    1
## b[3] -0.36    0.01 0.27 -0.90 -0.53 -0.36 -0.17  0.16   419    1
## phi   0.27    0.00 0.02  0.23  0.25  0.27  0.29  0.32   494    1
##
## Samples were drawn using NUTS(diag_e) at Thu May 26 13:21:20 2022.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
plot(fit_nb, pars = c("a", "b", "phi"))
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```



Posterior predictive checks

density comparison

We can compare the empirical density of the observed data to the posterior predictive density from the fitted models, to compare the goodness-of-fit and these models:

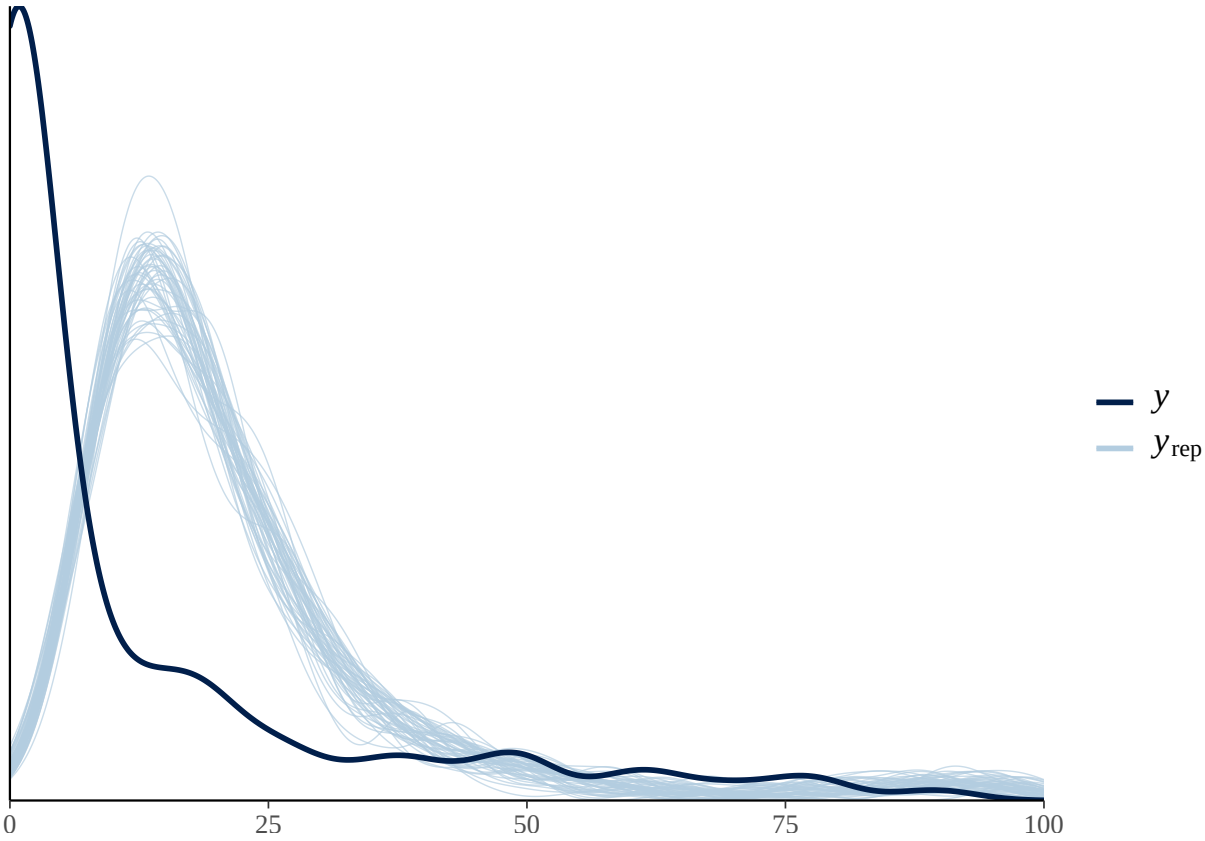
```
yrep_pois <- rstan::extract(fit_poisson, pars = "y_rep")$y_rep
```

```
# yrep_pois_alt <- as.matrix(fit_poisson, pars = "y_rep")
# extract() permutes the order of the draws,
# so these two matrices aren't in the same order

ppc_dens_overlay(y = d$y, yrep = yrep_pois[1:50, ]) + xlim(0, 100)
```

```
## Warning: Removed 303 rows containing non-finite values (stat_density).
```

```
## Warning: Removed 23 rows containing non-finite values (stat_density).
```

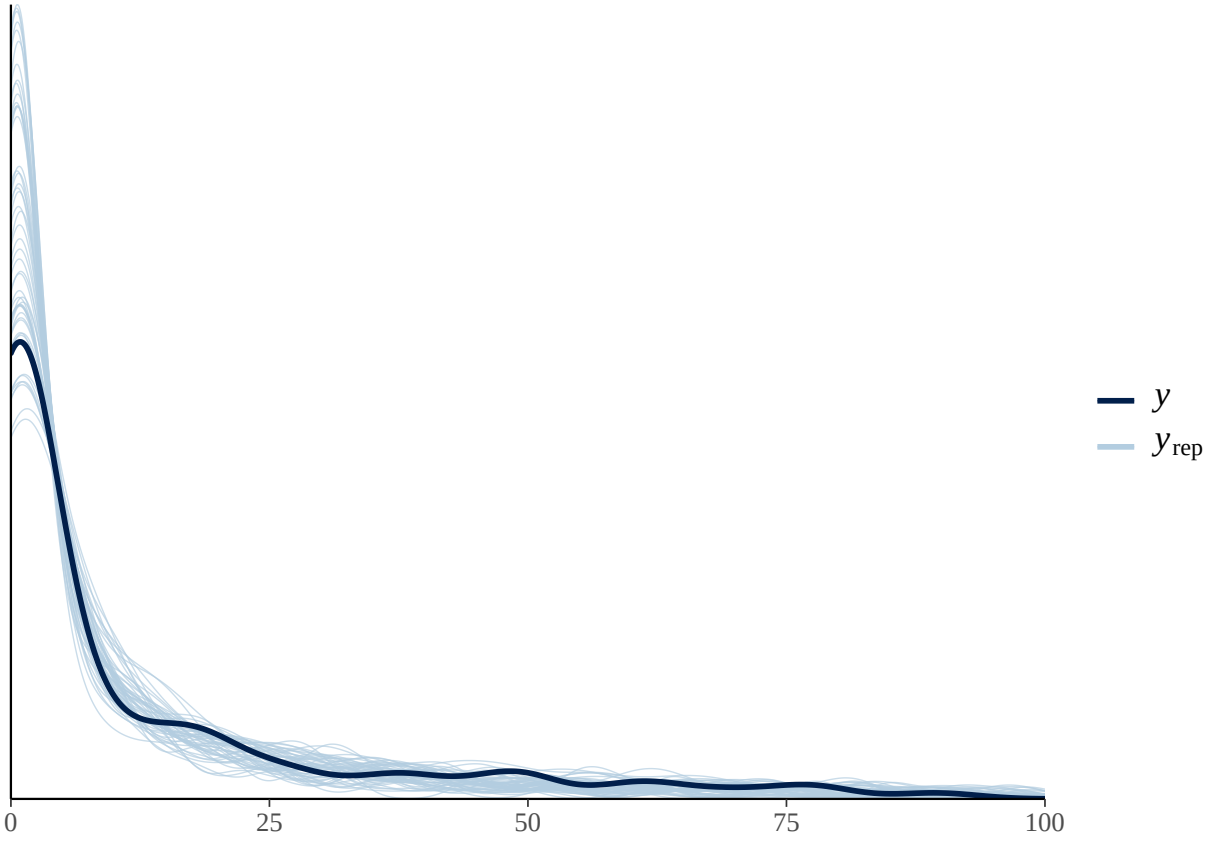



```
yrep_nb <- rstan::extract(fit_nb, pars = "y_rep")$y_rep

ppc_dens_overlay(y = d$y, yrep = yrep_nb[1:50, ]) + xlim(0, 100)
```

```
## Warning: Removed 933 rows containing non-finite values (stat_density).
```

```
## Warning: Removed 23 rows containing non-finite values (stat_density).
```



This is in line to what we have seen in the homeworks. You can see the second model has a much better model fit compared to the first one, due to the flexibility in capturing model variance overdispersion.

Goodness-of-fit Indicators: WAIC and LOO-CV

WAIC: Information Criteria:

WAIC is called widely-applicable information criterion or Watanabe-Akaike information criterion. It is mostly minus two times the log pointwise predictive density, so in general, the lower WAIC the better the model fit. There is an extra penalizing term on the effective number of parameters that penalizes the complexity of the model, so the more effective parameters, higher the WAIC.

We will be using functions in stan as a tool to calculate them.

```
ll_poisson <- extract_log_lik(fit_poisson, merge_chains = FALSE)
ll_neg_binomial <- extract_log_lik(fit_nb, merge_chains = FALSE)
waic_poisson <- waic(ll_poisson)
```

```
## Warning:
## 50 (19.1%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
waic_neg_binomial <- waic(ll_neg_binomial)
```

```
## Warning:
## 2 (0.8%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
loo_compare(waic_poisson, waic_neg_binomial)
```

```
##        elpd_diff se_diff
## model2     0.0       0.0
```

```
## model1 -5412.2     726.6
```

LOO-CV: leave one out cross validation

We can also look at the leave-one-out cross validation estimate of the log likelihood, as an indicator of the models' predictive performance. The higher, the better.

```
loo_poisson <- loo(fit_poisson)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
loo_neg_binomial <- loo(fit_nb)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
loo_compare(loo_poisson, loo_neg_binomial)
```

```
##        elpd_diff se_diff
## model2     0.0       0.0
## model1 -5332.7     704.0
```