DAY 1: RSTAN, STATISTICAL MODELS

STAN WORKSHOP

GOALS

- RStan / ShinyStan usage
- Process
- Fitting and writing models

RSTAN, SHINYSTAN

- RStan
  - Stan interface for R
- ShinyStan
  - posterior analysis for Stan fit objects

RSTAN EXAMPLE

RSTAN EXAMPLE

1. Open an R / RStudio session.
   Change working directory to a place you'll remember.
   Example: `~/stan_workshop_2016/`

2. Download `bernoulli.stan` to your working directory.
   Open the file in a text editor.

3. From R:

```
library(rstan)
options(mc.cores = parallel::detectCores())
library(shinystan)

N <- 10
y <- rbinom(N, 1, 0.3)

fit <- stan("bernoulli.stan")
```

SUMMARY OF THE FIT

- `fit`
- `print(fit)`

```
Inference for Stan model: bernoulli.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

      mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
p     0.17    0.00 0.10  0.02  0.09  0.15  0.22  0.40  1251    1
lp__ -5.95    0.02 0.76 -8.08 -6.13 -5.66 -5.46 -5.41  1358    1

Samples were drawn using NUTS(diag_e) at Mon May 30 12:28:21 2016.
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).
```

PLOTTING

- `traceplot(fit)`

- ```
  p <- extract(fit)
  hist(p$p, xlim = c(0, 1), freq = FALSE)
  curve(dbeta(x, sum(y) + 1, N - sum(y) + 1),
        0, 1, add=TRUE)
  ```

- `launch_shinystan(fit)`
  ```

RERUN WITHOUT RECOMPILING

- ```
  N <- 100
  y <- rbinom(N, 1, 0.3)
  ```
- ```
  fit <- stan(fit = fit)
  ```
- Examine fit

3 STEPS OF BAYESIAN DATA ANALYSIS

3 STEPS OF BAYESIAN DATA ANALYSIS (FROM BDA3)

1. Set up a full probability model.
   Joint probability distribution of all observables and unobservables.

2. Condition on observed data.
   Calculate and interpret the posterior distribution.

3. Evaluate fit of the model and the implications.
   Are the conclusions reasonable?

3 STEPS OF BAYESIAN DATA ANALYSIS (FROM BDA3)

1. Set up a full probability model.
   Write a Stan program.

2. Condition on observed data.
   Run the Stan program using RStan.
   Check to see if everything went ok in RStan and ShinyStan.

3. Evaluate fit of the model and the implications.
   Evaluate in RStan and ShinyStan.
   Question assumptions.
   Go back to 1.

LINEAR MODEL

LINEAR MODEL

- $y = a + b * x + \epsilon$
- $\epsilon \sim \text{Normal}(0, \sigma)$

- Equivalent to:

$$y \sim \text{Normal}(\text{a} + \text{b} * \text{x}, \sigma)$$

SIMULATE DATA IN R

- Pick `a, b,` and `sigma`
  Bonus: draw `a, b,` and `sigma` from distributions

```
a <- 10
b <- 5
sigma <- 5
```

- Simulate data

```
N <- 1000
x <- rnorm(N, 0, 5)
y <- a + b * x + rnorm(N, 0, sigma)
plot(x, y)
```

LINEAR MODEL

- Stan program: linear_model.stan

```
data {
  int N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real a;
  real b;
  real<lower = 0> sigma;
}
model {
  // bonus: priors
  y ~ normal(a + b * x, sigma);
}
```

- Run Stan program:

```
fit <- stan("linear_model.stan")
```

- Compare to lm in R:

```
summary(lm(y ~ x))
```

- Run again with less data points

```
N <- 1
x <- as.array(rnorm(N, 0, 5))
y <- as.array(a + b * x + rnorm(N, 0, sigma))

fit <- stan(fit = fit)
print(fit)
summary(lm(y ~ x))
```

LINEAR MODEL

- Posterior distribution determined by **statistical model** and **data**
- Let's add priors. Fit again. Compare.
  - `a ~ normal(0, 10);`
  - `b ~ normal(0, 10);`
  - `sigma ~ normal(0, 10);`
  - (We're human. We don't usually think of numbers like 1e20.)
- Bonus
  - generate data from priors and fit
  - log-normal error errors
  - implement glm: add link function

POISSON

POISSON REGRESSION

- Discrete distribution for count data
- Example: study of cockroaches in city apartments (From Gelman and Hill. Ch 8)
- We'll start simple and build up.

```
source("roaches.data.R")
```

$$y \sim \text{Poisson}(\lambda)$$

POISSON

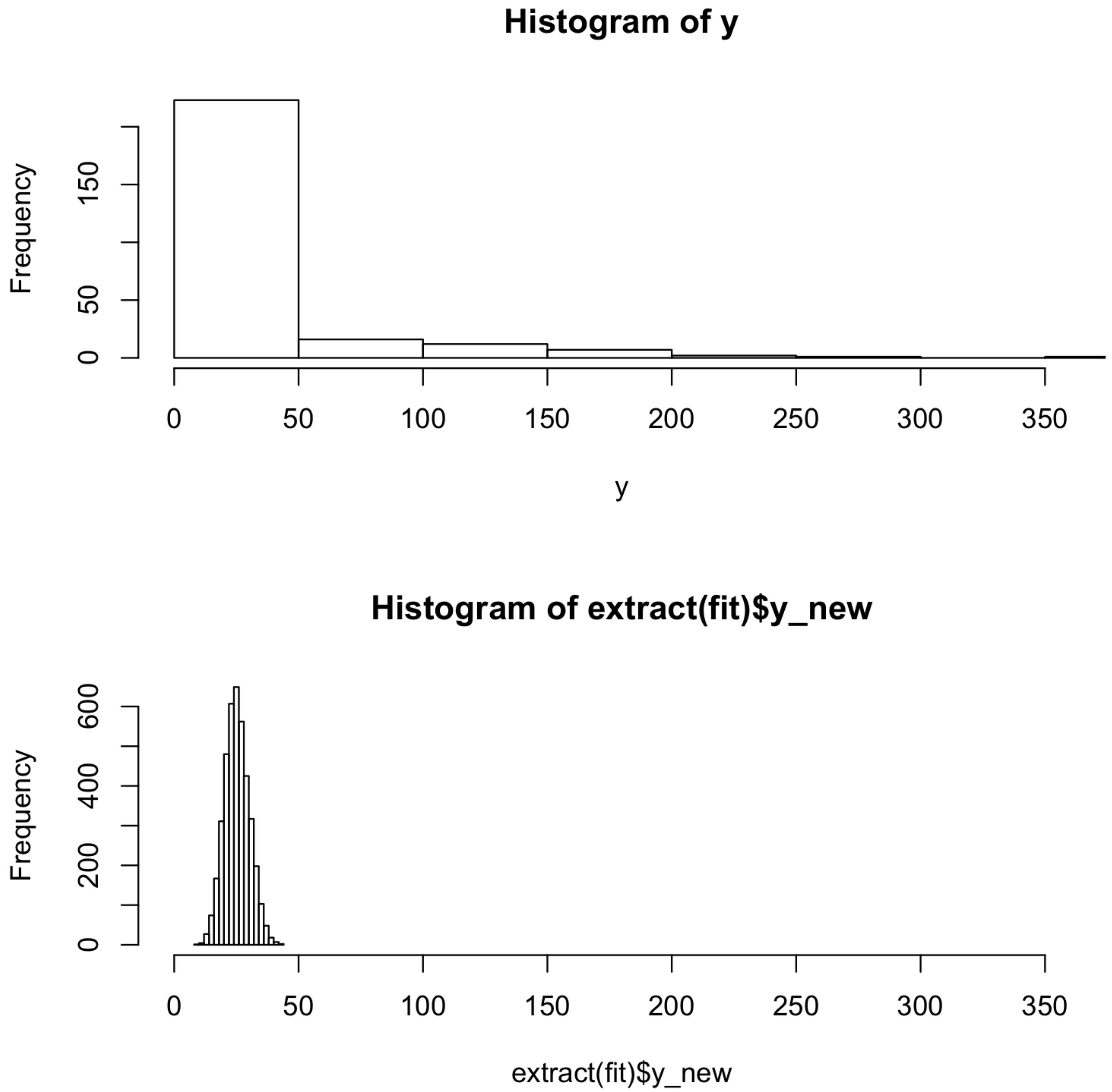
```
data {
  int N;
  int<lower = 0> y[N];
}
parameters {
  real<lower = 0> lambda;
}
model {
  lambda ~ normal(20, 20);
  y ~ poisson(lambda);
}
generated quantities {
  int y_new;
  y_new <- poisson_rng(lambda);
}
```

LOOK RIGHT?

```
par(mfrow=c(2, 1))

hist(y, xlim=c(0, 360))

hist(extract(fit)$y_new, xlim=c(0, 360))
```



ADD PREDICTORS

- $y_n \sim \text{Poisson}(\mu_n \exp^{X_n \beta})$
- In Stan, we have the `poisson_log` distribution
- Steps
  1. Add data:
     ```
     vector<lower = 0>[N] exposure2;
     vector[N] roach1;
     vector[N] treatment;
     vector[N] senior;
     ```
  2. Add predictors: `vector[4] beta;`
  3. Change likelihood:
     ```
     y[n] ~ poisson_log(log(exposure[n])
                        + beta[1]
                        + beta[2] * roach1[n]
                        + beta[3] * treatment[n]
                        + beta[4] * senior[n]));
     ```
- Bonus: negative binomial with overdispersion

ADD PREDICTORS

```
data {
  int N;
  int<lower = 0> y[N];
  vector<lower = 0>[N] exposure2;
  vector[N] roach1;
  vector[N] treatment;
  vector[N] senior;
}
transformed data {
   vector[N] log_exposure;
   log_exposure <- log(exposure2);
}
parameters {
  vector[4] beta;
}
model {
  beta[1] ~ normal(3, 1);
  beta[2:4] ~ normal(0, 1);
  y ~ poisson_log(log_exposure + beta[1] + beta[2] * roach1
                  + beta[3] * treatment + beta[4] * senior);
}
generated quantities {
  int y_new[N];
  for (n in 1:N)
    y_new[n] <- poisson_log_rng(log_exposure[n] + beta[1] + beta[2] * roach1[n]
                                + beta[3] * treatment[n] + beta[4] * senior[n]);
}
```

CLOSER

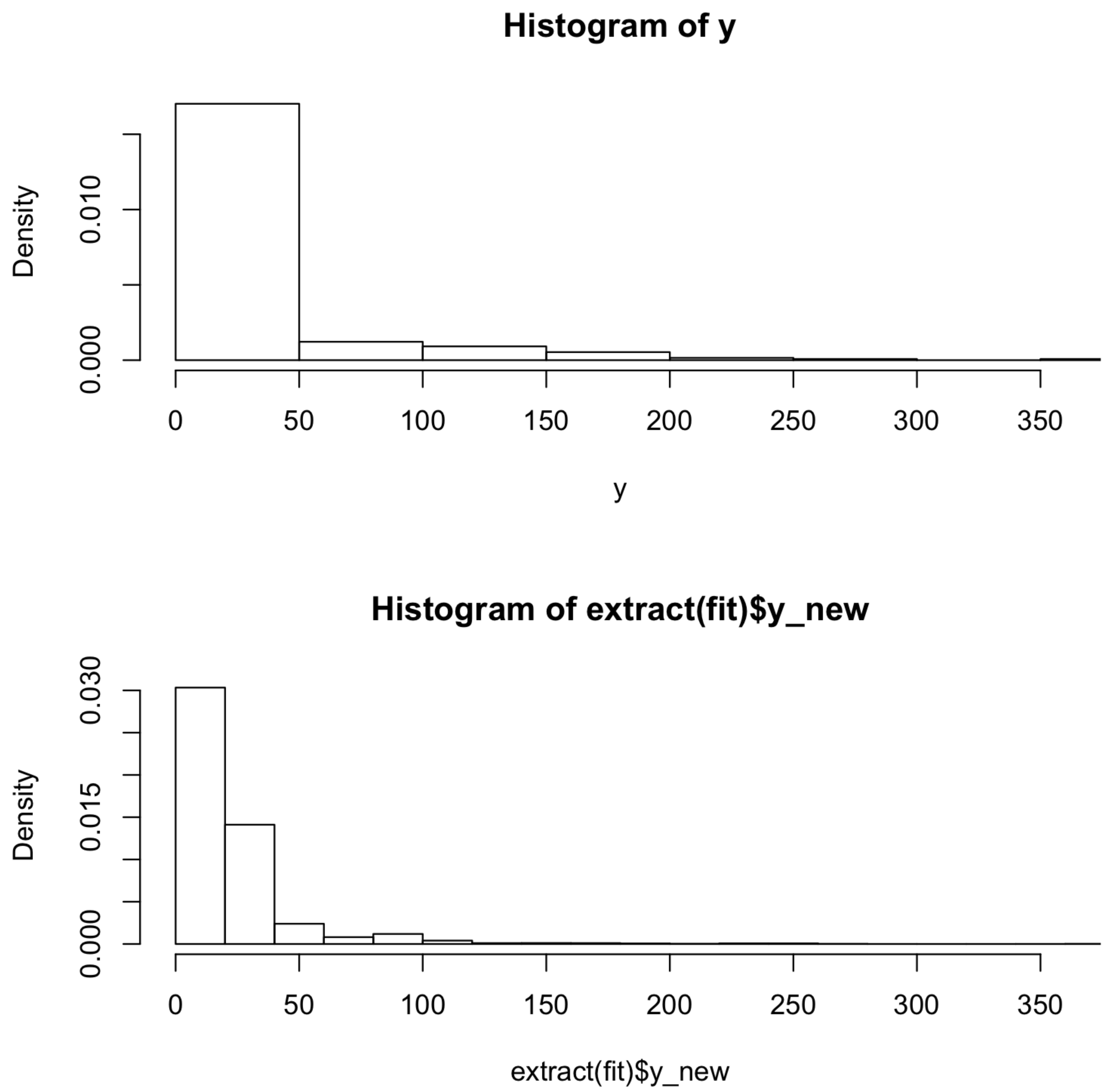
```
> print(fit, “beta")

Inference for Stan model: poisson_2.
2 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=2000.

         mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
beta[1]  3.09       0 0.02  3.05  3.07  3.09  3.10  3.13   641 1.00
beta[2]  0.01       0 0.00  0.01  0.01  0.01  0.01  0.01  1146 1.00
beta[3] -0.52       0 0.03 -0.57 -0.54 -0.52 -0.50 -0.47   421 1.01
beta[4] -0.38       0 0.03 -0.44 -0.40 -0.38 -0.36 -0.31   702 1.00

Samples were drawn using NUTS(diag_e) at Mon May 30 18:35:39 2016.
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).
```

CLOSER



Histogram of y
Density
0.000
0.010
0
50
100
150
200
250
300
350
y
Histogram of extract(fit)$y_new
Density
0.000
0.015
0.030
0
50
100
150
200
250
300
350
extract(fit)$y_new

ZERO-INFLATION

ZERO-INFLATION

- Too many zeros in data.
  Another process that inserts 0s into data.

- For every data point,
  it either comes from a Poisson draw OR it was 0

- Need to use `increment_log_prob()`

- $p(y_n = 0) = \theta \times 1 + (1 - \theta) \times \text{Poisson}(\lambda_n)$

- $p(y_n \neq 1) = \theta \times 0 + (1 - \theta) \times \text{Poisson}(\lambda_n)$

STAN PROGRAM

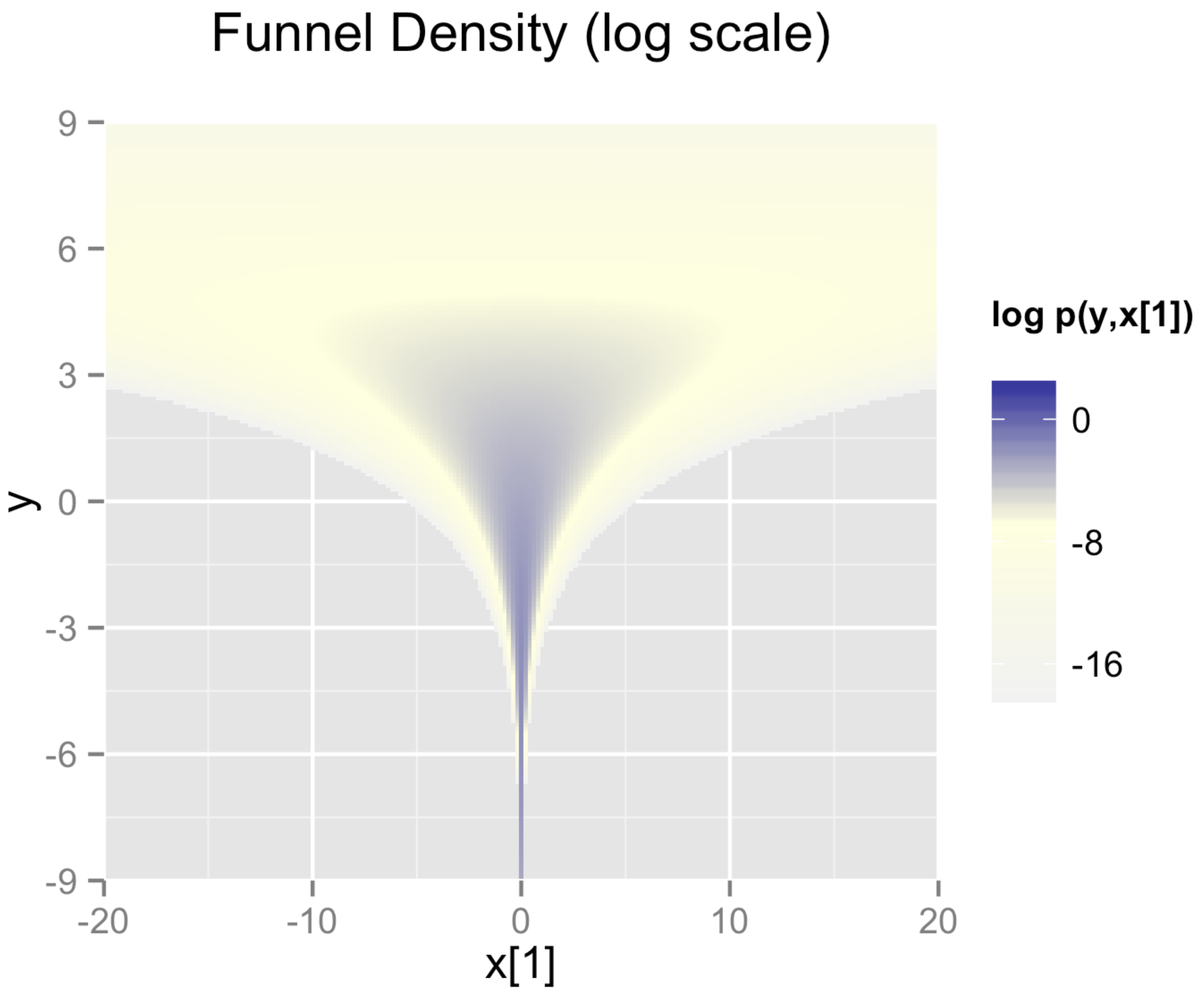
```
data {
  ...
}
transformed data {
  ...
}
parameters {
  ...
}
model {
  beta[1] ~ normal(3, 1);
  beta[2:4] ~ normal(0, 1);
  for (n in 1:N)
    if (y[n] == 0)
      increment_log_prob(log_sum_exp(log(theta),
                                     log1m(theta)
                                     + poisson_log_log(y,
                                                       log_exposure + beta[1] + beta[2] * roach1
                                                       + beta[3] * treatment + beta[4] * senior)));
    else
      increment_log_prob(log1m(theta)
                         + poisson_log_log(y,
                                           log_exposure + beta[1] + beta[2] * roach1
                                           + beta[3] * treatment + beta[4] * senior));
}
generated quantities {
  int y_new[N];
```

NON-CENTERED REPARAMETERIZATION

FUNNEL

- $y \in \mathbb{R}$
- $x \in \mathbb{R}^9$

$$p(y, x) = \text{Normal}(y|0, 3) \times \prod_{n=1}^{9} \text{Normal}(x_n|0, \exp(y/2))$$



WHEN DO YOU SEE THIS?

- Hierarchical models
- Variance parameters go to 0, all parameters shrink
  Variance parameters get large, all parameters spread
- Trick to handle low data situations
- Called non-centered parameterization aka the Matt trick …

STEPS

1. Add new parameter, *_raw.
2. Move original parameter to transformed parameters block.
3. Assign transformation of *_raw to original parameter.
4. Put Normal(0, 1) prior on *_raw.

CENTERED FUNNEL

- Easy to write in Stan
- Run. See any problems?

```
parameters {
  real y;
  vector[9] x;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```

NON-CENTERED FUNNEL: STEP 1

- Add new parameter, *_raw.

```
parameters {
  real y;
  vector[9] x;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```

NON-CENTERED FUNNEL: STEP 1

- Add new parameter, *_raw.

```
parameters {
  real y;
  vector[9] x;
  real y_raw;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```

NON-CENTERED FUNNEL: STEP 2

- Move original parameter to transformed parameters block.

```
parameters {
  real y;
  vector[9] x;
  real y_raw;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```

NON-CENTERED FUNNEL: STEP 2

- Move original parameter to transformed parameters block.

```
parameters {
  vector[9] x;
  real y_raw;
}
transformed parameters {
  real y;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```

NON-CENTERED FUNNEL: STEP 3

- Assign transformation of *_raw to original parameter.

```
parameters {
  vector[9] x;
  real y_raw;
}
transformed parameters {
  real y;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```

NON-CENTERED FUNNEL: STEP 3

- Assign transformation of *_raw to original parameter.

```
parameters {
  vector[9] x;
  real y_raw;
}
transformed parameters {
  real y;
  y <- 3 * y_raw;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```

NON-CENTERED FUNNEL: STEP 4

- Put Normal(0, 1) prior on *_raw.

```
parameters {
  vector[9] x;
  real y_raw;
}
transformed parameters {
  real y;
  y <- 3 * y_raw;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```

NON-CENTERED FUNNEL: STEP 4

- Put Normal(0, 1) prior on *_raw.

```
parameters {
  vector[9] x;
  real y_raw;
}
transformed parameters {
  real y;
  y <- 3 * y_raw;
}
model {
  y_raw ~ normal(0, 1);
  x ~ normal(0, exp(y/2));
}
```

NON-CENTERED FUNNEL

- Repeat for xs.
- Steps:
  1. Add new parameter, *_raw.
  2. Move original parameter to transformed parameters block.
  3. Assign transformation of *_raw to original parameter.
  4. Put Normal(0, 1) prior on *_raw.

NON-CENTERED FUNNEL

```
parameters {
  real y_raw;
  vector[9] x_raw;
}
transformed parameters {
  real y;
  vector[9] x;

  y <- 3.0 * y_raw;
  x <- exp(y/2) * x_raw;
}
model {
  y_raw ~ normal(0, 1);
  x_raw ~ normal(0, 1);
}
```

CENTERED VS NON-CENTERED

```
parameters {
  real y;
  vector[9] x;
}
model {
  y ~ normal(0, 3);
  x ~ normal(0, exp(y/2));
}
```

```
parameters {
  real y_raw;
  vector[9] x_raw;
}
transformed parameters {
  real y;
  vector[9] x;

  y <- 3.0 * y_raw;
  x <- exp(y/2) * x_raw;
}
model {
  y_raw ~ normal(0, 1);
  x_raw ~ normal(0, 1);
}
```