3.3 Practical: nonlinear model - functional response

Benjamin Rosenbaum

October 26, 2022

Model

In feeding interactions between a predator and a prey species, the functional response F describes the density-dependent feeding rate of the consumer.

The number of prey individuals a predator can consume in a given time does not grow proportionally with increasing prey availability, but reaches a saturation.

(A predator can only consume a limited number of prey in a given time.)

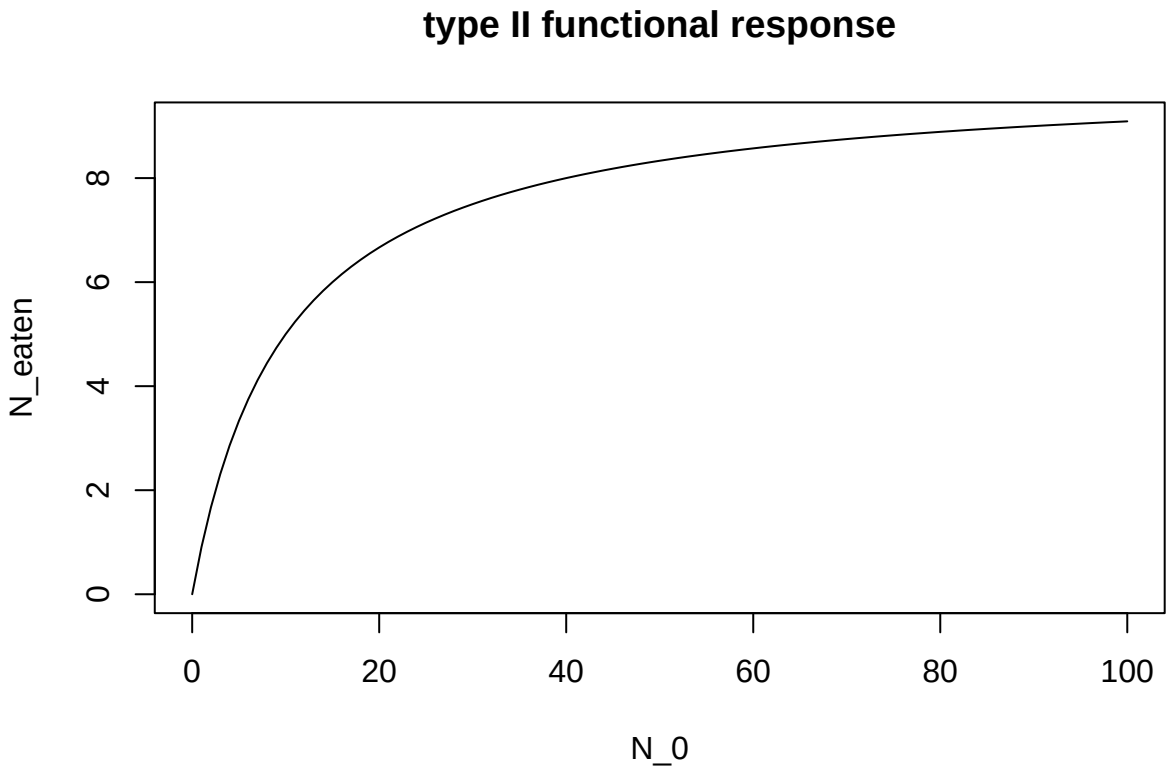
Mathematically, $F(N)$ is parameterised by attack rate a and handling time h

$$F(N) = \frac{aN}{1 + ahN}$$

a descibes the slope of the curve in the origin, while the saturation (maximum feeding rate) is given by $\frac{1}{h}$.

This is a nonlinear function (in the parameters) and there is no link function that converts it to a linear model. So can't use `lm()` or `glm()` to fit it.

```
a=1
h=0.1
curve(a*x/(1+a*h*x), from=0, to=100,
      xlab="N_0", ylab="N_eaten", main="type II functional response")
```



Setup

```
rm(list=ls())
library(rstan)
library(coda)
library(BayesianTools)
library(brms)

setwd("~/Nextcloud/teaching Bayes 2021/")

rstan_options(auto_write = TRUE)
options(mc.cores = 4)
```

Read dataset

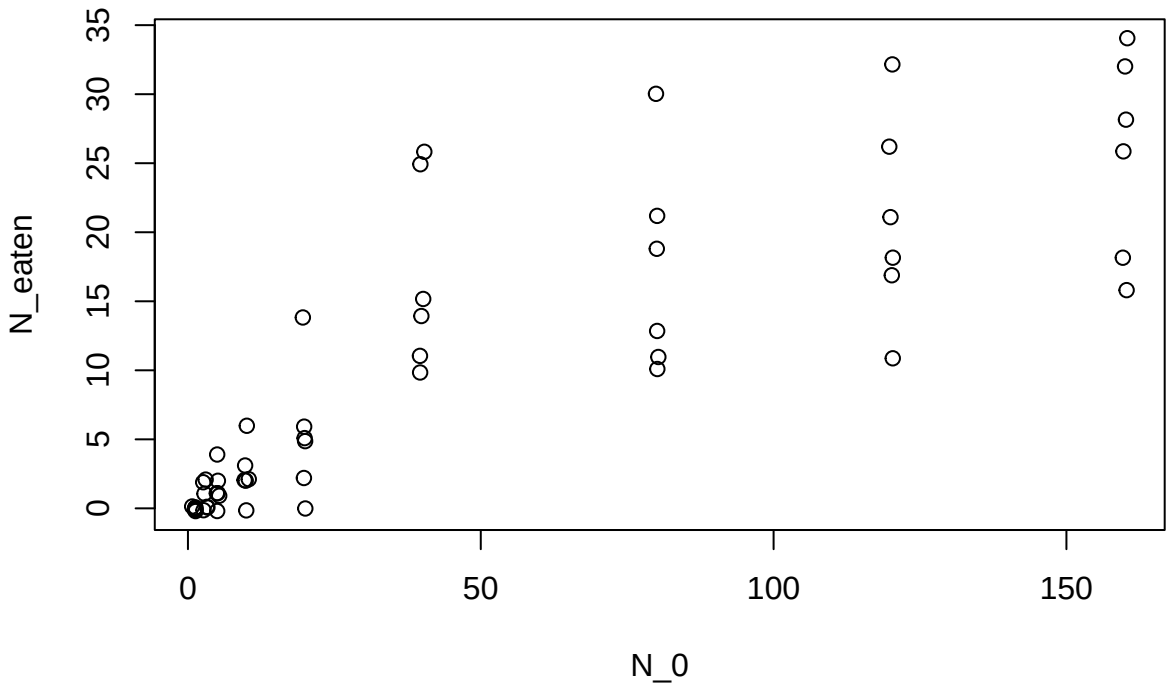
We read data from a feeding experiment.

In repeated feeding trials with varying number of initial prey, the number of eaten prey was counted

```
df = read.csv("data/feeding_experiment.csv")
head(df)
```

```
##   N0 Neaten
## 1  1      0
## 2  1      0
## 3  1      0
## 4  1      0
## 5  1      0
## 6  1      0
```

```
plot(jitter(df[,1]), jitter(df[,2]),xlab="N_0", ylab="N_eaten")
```



Let's assume that the experiment was supervised and each eaten prey item was replaced immediately, such that the number of available prey was constant during the course of the experiment.

(Otherwise we would have to correct for prey depletion, see Rosenbaum & Rall, 2018).

The deterministic model is defined by the functional response $F(N)$.

This is count data, so we use a poisson distribution for the stochastic part.

$$y_i \sim \text{poisson}(F_i)$$
$$F_i = \frac{aN_i}{1 + ahN_i}$$

Stan code and fitting

```
data = list(n = nrow(df),
            x = df$N0,
            y = df$Neaten)

stan_code = '
data {
  int n;
  int x[n];
  int y[n];
}
parameters {
  real<lower=0> a;
  real<lower=0> h;
}
model {
```

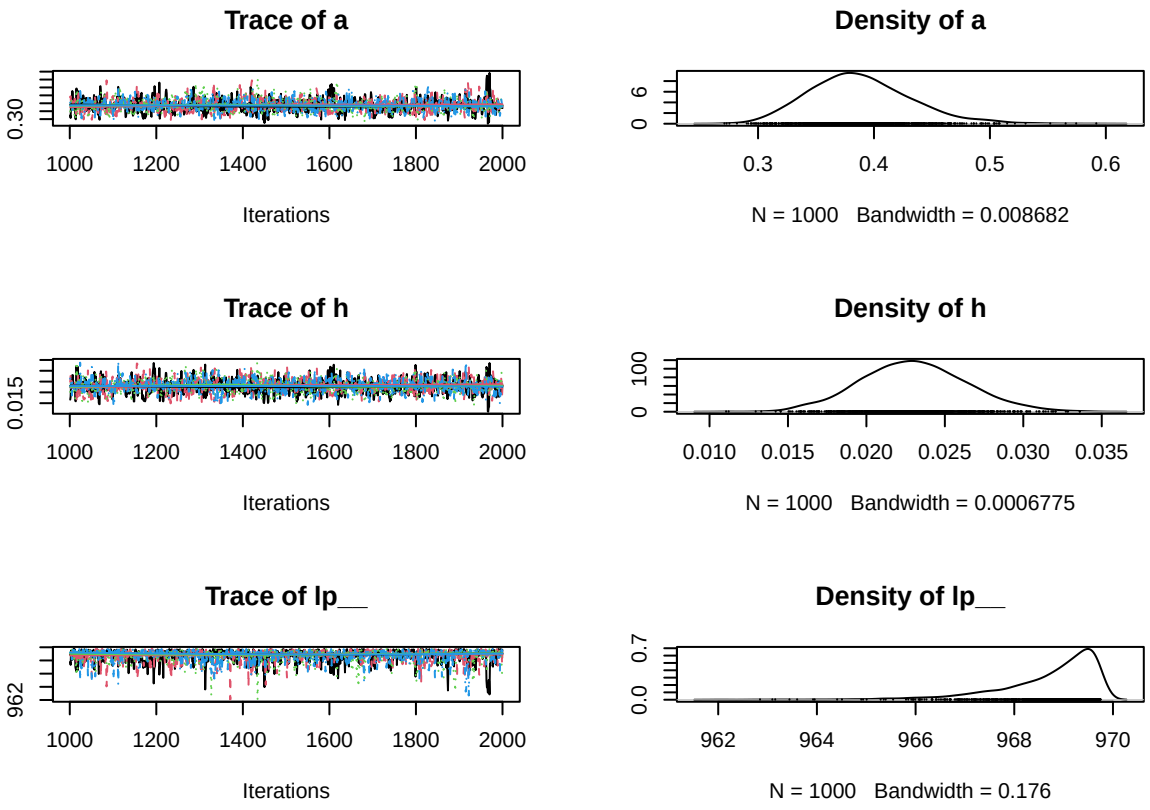
```
  // priors
  a ~ normal(0, 10);
  h ~ normal(0, 1);
  // likelihood
  for(i in 1:n){
    y[i] ~ poisson( a*x[i]/(1.0+a*h*x[i]) );
  }
}
'
```

```
stan_model = stan_model(model_code=stan_code)
fit = sampling(stan_model, data=data)
```

```
print(fit, digits=3, probs=c(0.025, 0.975))
```

```
## Inference for Stan model: 9851d626b23d5eb574cf1138e3094434.
## 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%   97.5% n_eff  Rhat
## a      0.388   0.001 0.043   0.313   0.486  1266 1.002
## h      0.023   0.000 0.003   0.016   0.030  1411 1.003
## lp__ 968.717   0.030 1.023 965.964 969.724  1199 1.002
##
## Samples were drawn using NUTS(diag_e) at Thu Oct  7 11:25:45 2021.
## 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(As.mcmc.list(fit))
```



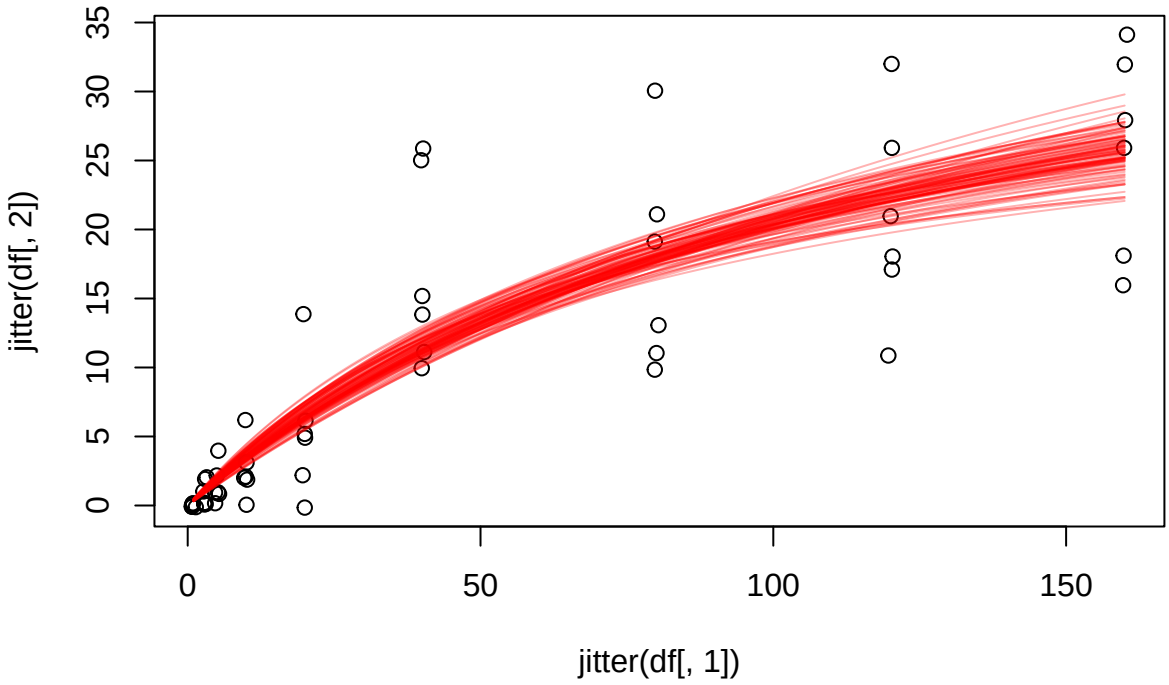
Predictions

90% credible intervals for the deterministic part of the model

```
posterior=as.matrix(fit)

x.pred = seq(from=1, to=max(data$x), by=1)
y.cred = matrix(0, nrow=nrow(posterior), ncol=length(x.pred))

for(i in 1:nrow(posterior)){
  y.cred[i, ] = posterior[i,"a"]*x.pred / (1+posterior[i,"a"]*posterior[i,"h"]*x.pred)
}
```

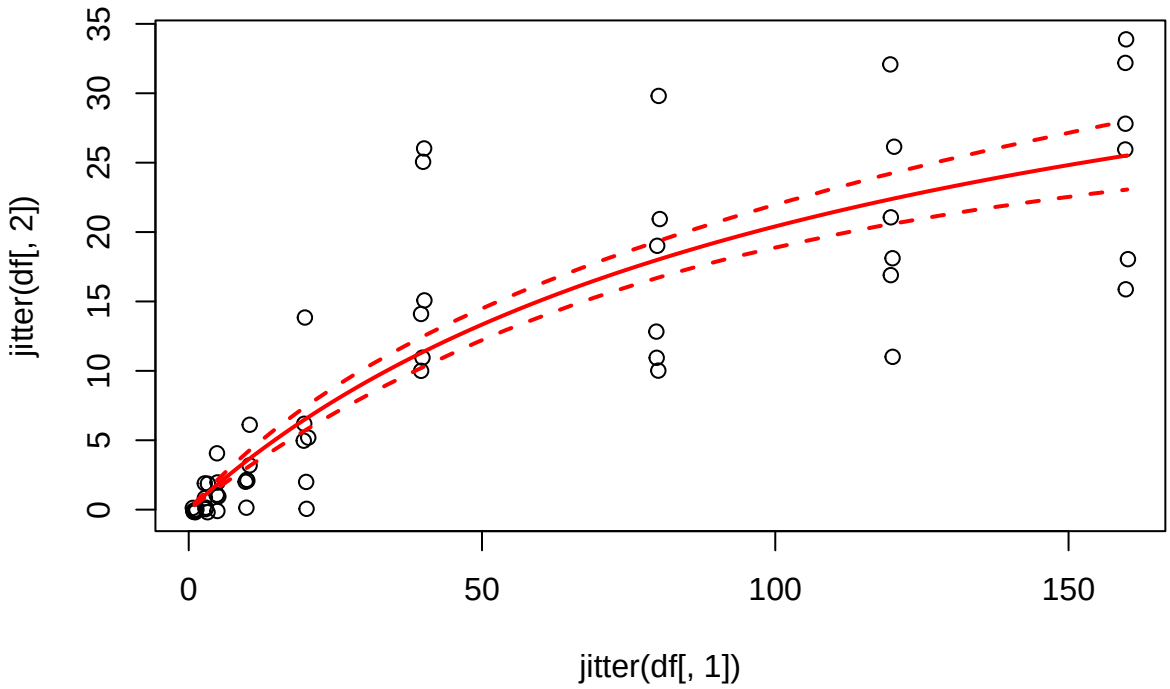
Each sample of the posterior corresponds to a posterior regression line.

```
plot(jitter(df[,1]), jitter(df[,2]))

for(i in 1:100){
  lines(x.pred, y.cred[i, ], col=adjustcolor("red", alpha.f=0.3))
}
```



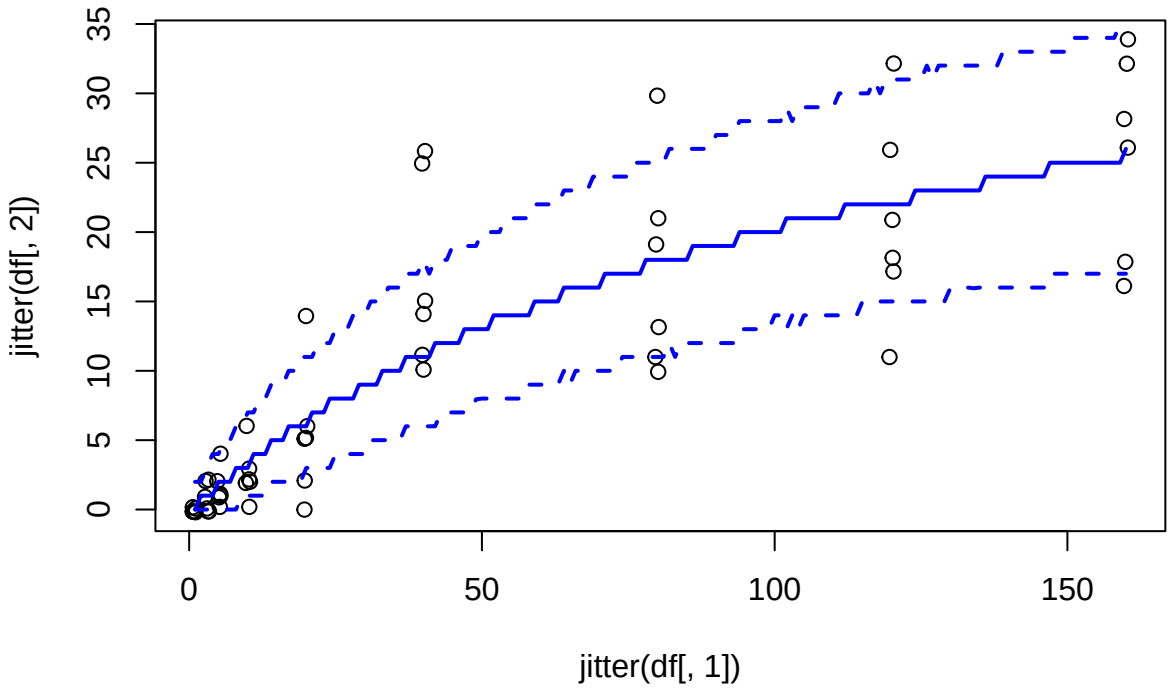
Mean prediction and quantiles (credible intervals) are plotted.

```
plot(jitter(df[,1]), jitter(df[,2]))

y.cred.mean = apply(y.cred, 2, function(x) mean(x))
lines(x.pred, y.cred.mean, col="red", lwd=2)

y.cred.q05 = apply(y.cred, 2, function(x) quantile(x, probs=0.05))
lines(x.pred, y.cred.q05, col="red", lwd=2, lty=2)

y.cred.q95 = apply(y.cred, 2, function(x) quantile(x, probs=0.95))
lines(x.pred, y.cred.q95, col="red", lwd=2, lty=2)
```



90% predictions intervals for the data including stochastic part of the model. Quantiles are integer numbers because prediction model draws from a poisson distribution. If we use median (=50% quantile) instead of mean, this is also integer-valued.

```
y.pred = matrix(0, nrow=nrow(posterior), ncol=length(x.pred))
for(i in 1:nrow(posterior)){
  y.pred[i, ] = rpois(n=length(x.pred), lambda=y.cred[i, ])
}

plot(jitter(df[,1]), jitter(df[,2]))

y.pred.med = apply(y.pred, 2, function(x) median(x))
lines(x.pred, y.pred.med, col="blue", lwd=2)

y.pred.q05 = apply(y.pred, 2, function(x) quantile(x, probs=0.05))
lines(x.pred, y.pred.q05, col="blue", lwd=2, lty=2)

y.pred.q95 = apply(y.pred, 2, function(x) quantile(x, probs=0.95))
lines(x.pred, y.pred.q95, col="blue", lwd=2, lty=2)
```



Variation in observed data increases with N, this variation is also captured in the predicted values.

brms fit

For fitting a nonlinear model in brms, a formula has to be specified with `bf()`. Predictor, response and variable names are arbitrary, but parameters are specified with `a~1`, `h~1`. This means that they are estimated directly and are not depending on other linear predictors. `nl=TRUE` specifies a nonlinear model.

```
formula.1 = bf(y ~ a*x/(1.0+a*h*x),
               nl=TRUE,
               a~1,
               h~1)
```

Priors can be defined with `nlpar` (nonlinear parameter). We enforce lower boundaries `lb=0` to define positive parameters.

```
priors = c(prior(normal(0,10), nlpar="a", lb=0),
           prior(normal(0,1),  nlpar="h", lb=0) )
```

Then, the model is fitted with `brm()` as before, handing over the formula. Without further specifications, Gaussian family (normally distributed residuals) is chosen by default.

```
fit.b1 = brm(formula.1,
             data=data,
             prior=priors)
```

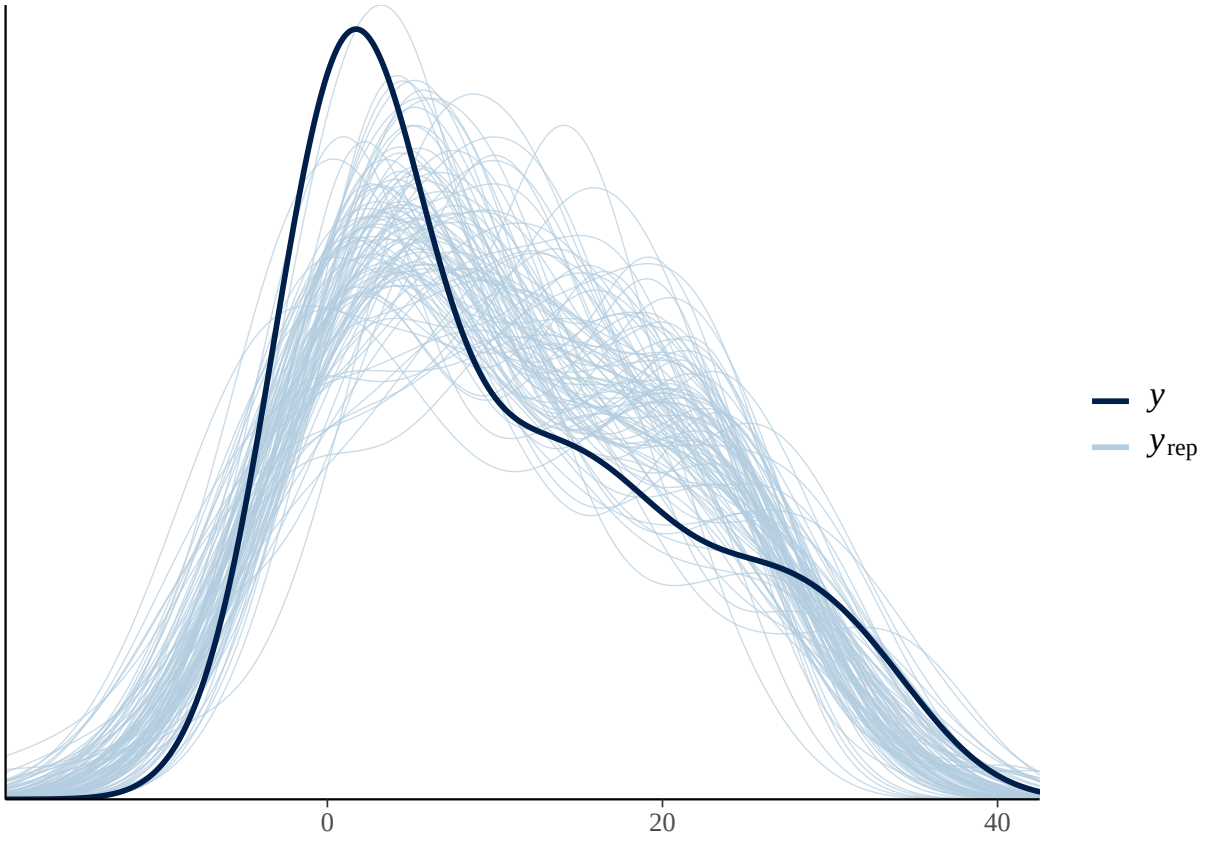
```
fit.b1
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ a * x/(1 + a * h * x)
##          a ~ 1
##          h ~ 1
##    Data: data (Number of observations: 54)
```

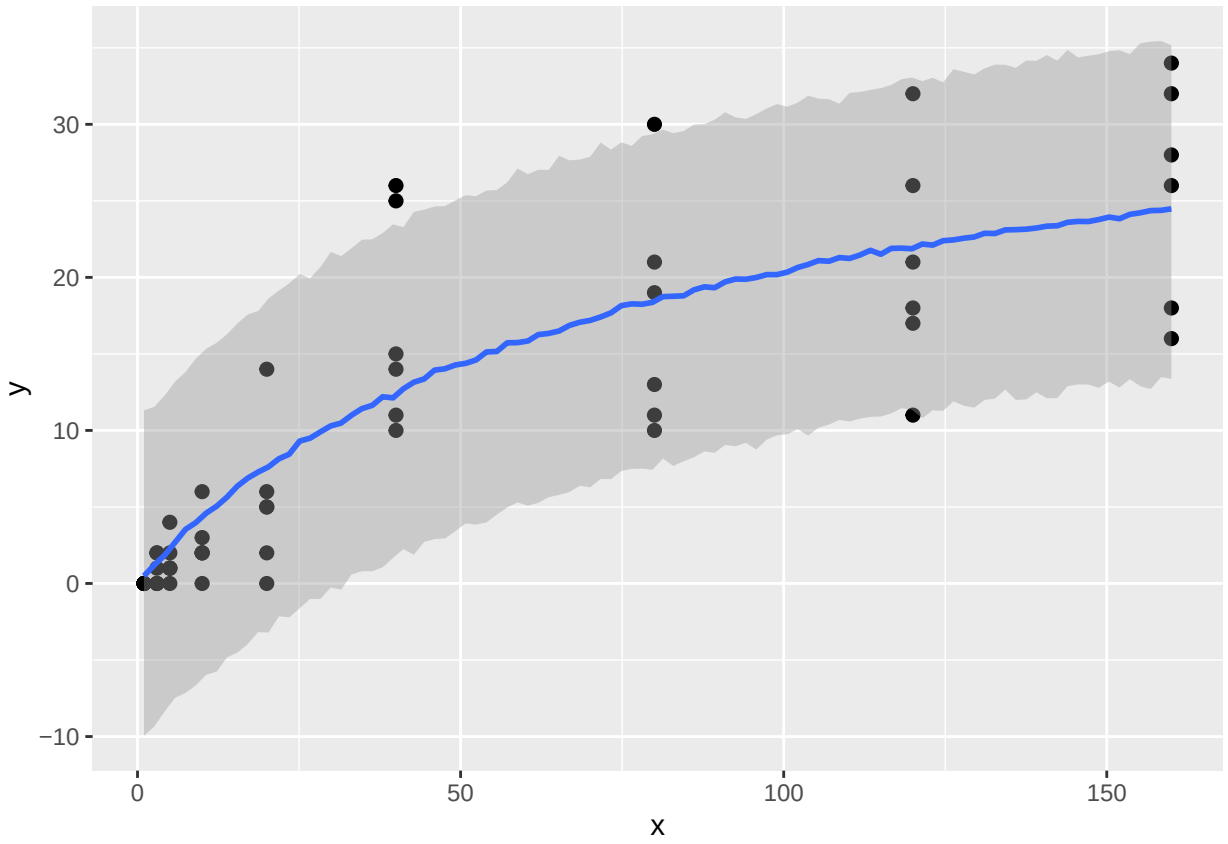
```
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Population-Level Effects:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## a_Intercept     0.50      0.12     0.30     0.78 1.00     1522     1389
## h_Intercept     0.03      0.01     0.02     0.04 1.00     1413     1228
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     5.32      0.52     4.41     6.43 1.00     1806     1894
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

When checking posterior predictions, we see that prediction intervals estimate negative data! Data is purely positive (number of eaten individuals cannot be negative), so we have to choose a better model.

```
pp_check(fit.b1, ndraws=100)
```




```
plot(conditional_effects(fit.b1, method="posterior_predict"),
     points=TRUE,
     ask=FALSE)
```



Specifying `family=poisson()` (integer-valued and non-negative as the data) is a better choice!

```
fit.b2 = brm(formula.1,
             data=data,
             prior=priors,
             family=poisson()
)
```

But now, parameter estimates are completely different! What went wrong? The second line reads `Links: mu = log`, so `family=poisson` has a log link function as default, which means $\log^{-1}(formula) = \exp(formula)$ is fitted to the observations. This biases the estimates. We want to fit formula to the data directly, without link funcion.

```
##  Family: poisson
##   Links: mu = log
## Formula: y ~ a * x/(1 + a * h * x)
##          a ~ 1
##          h ~ 1
##    Data: data (Number of observations: 54)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Population-Level Effects:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## a_Intercept     0.20      0.02     0.15     0.25 1.00     1301     1244
## h_Intercept     0.28      0.01     0.26     0.29 1.00     1329     1505
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
```

```
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

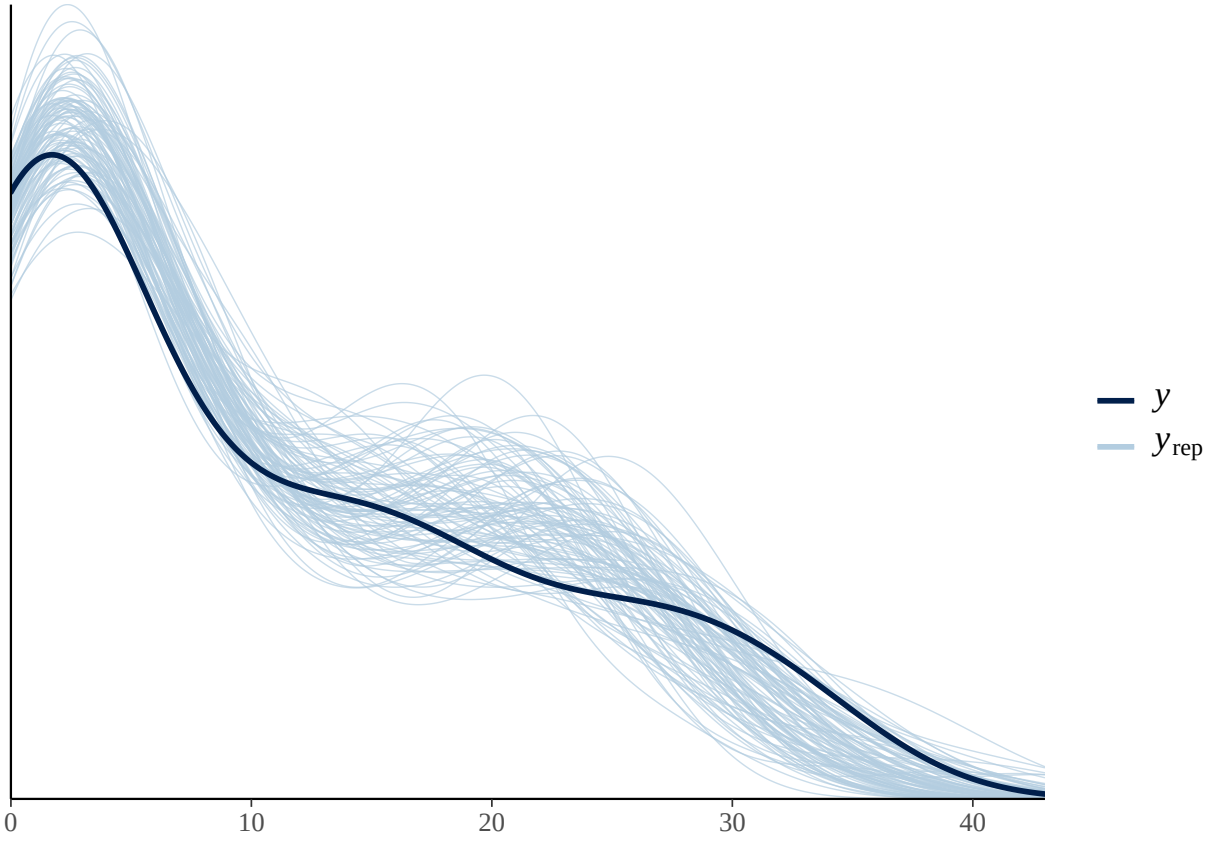
Specifying `link="identity"` overwrites the poisson's default choice of log link, meaning no link function is used.

```
fit.b3 = brm(formula.1,
             data=data,
             prior=priors,
             family=poisson(link="identity")
)
```

```
fit.b3
```

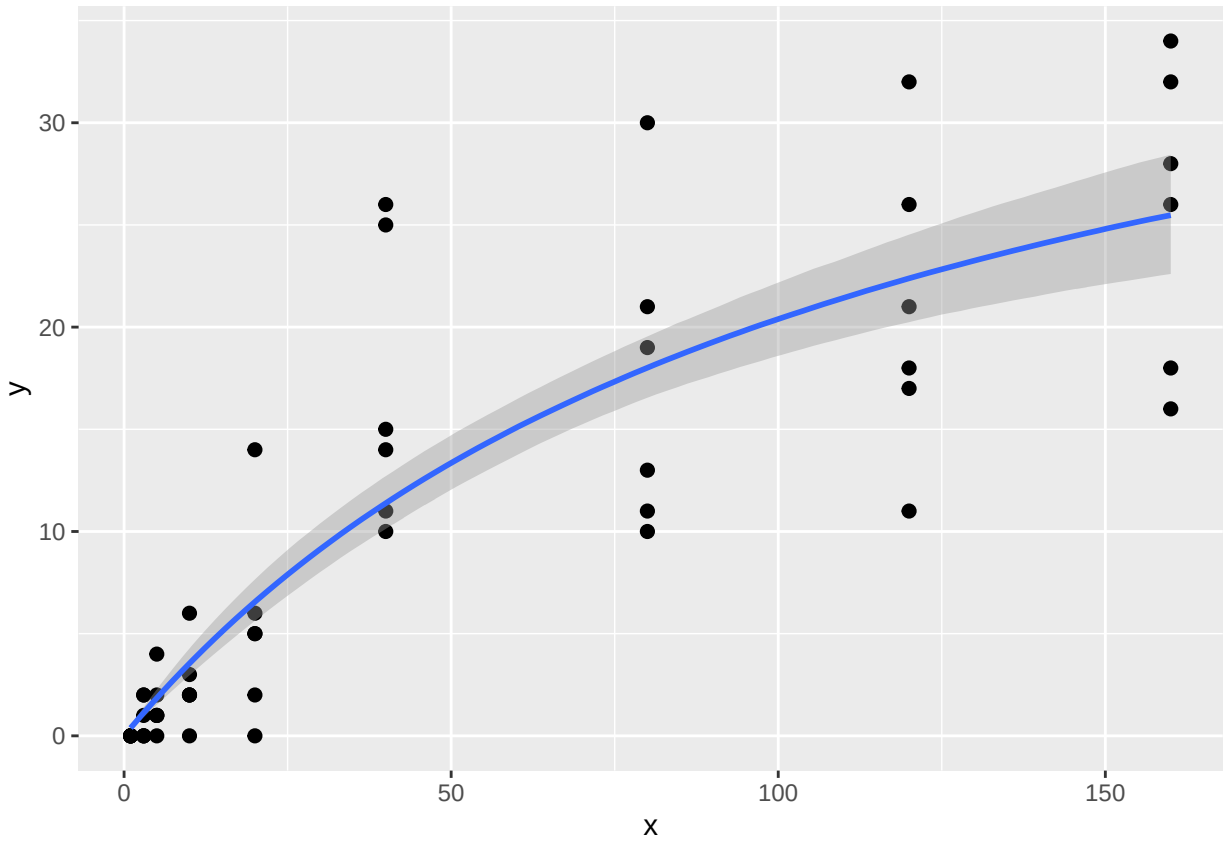
```
##  Family: poisson
##   Links: mu = identity
## Formula: y ~ a * x/(1 + a * h * x)
##          a ~ 1
##          h ~ 1
##    Data: data (Number of observations: 54)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
##
## Population-Level Effects:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## a_Intercept     0.39      0.04     0.31     0.49 1.00     1264     1538
## h_Intercept     0.02      0.00     0.02     0.03 1.00     1350     1884
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
pp_check(fit.b3, ndraws=100)
```

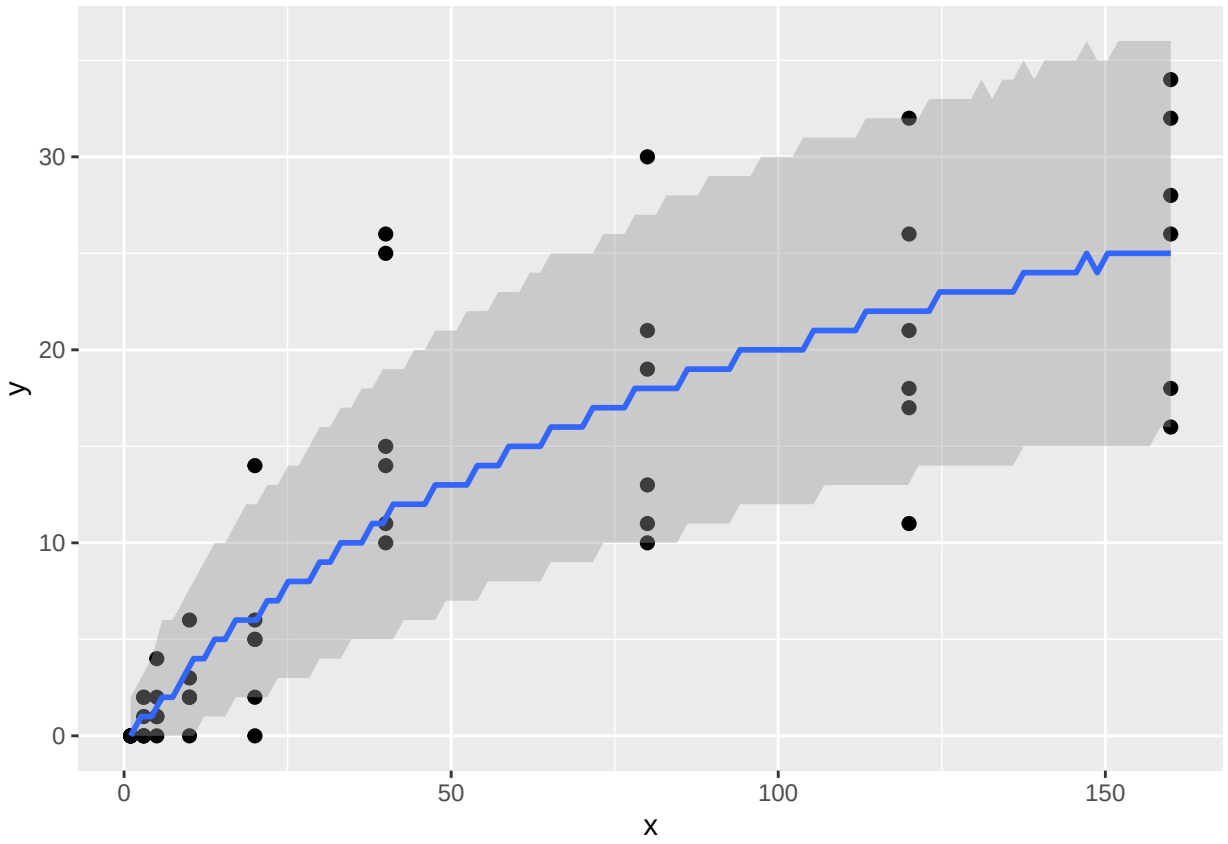


Posterior predictions with conditional_effects look identical to the Stan model above.

```
plot(conditional_effects(fit.b3),
     points=TRUE,
     ask=FALSE)
```




```
plot(conditional_effects(fit.b3,method="posterior_predict") ,
     points=TRUE,
     ask=FALSE)
```


30
20
10
0
y
0
50
100
150
x