Hierarchical modeling course notes

December 13, 2015

Contents

Preface

This document was made to complement a graduate level course in hierarchical Bayesian models aimed at biology graduate students who have some familiarity with R and statistics. While there are many great textbooks out there to go along with such a class, it seemed potentially useful to compile some concise notes with the code and concepts together. These notes are intended to be used along with other resources, which are listed at the end of each chapter. Many people have contributed to the development of these notes including Max Joseph, Will Stutz, Tim Szewczyk, Helen McCreery, Topher Weiss-Lehman, Lauren Shoemaker, and Piet Johnson. All source code to compile this document is available on GitHub at https://github.com/hmods/notes.

Chapter 1: Linear models

Big picture

This course builds on an understanding of the mechanics of linear models. Here we introduce some key topics that will facilitate future understanding hierarchical models.

Learning goals

- linear regression with `lm`
- intercepts, "categorical" effects
- varying model structure to estimate effects and standard errors
- interactions as variation in slope estimates for different groups
- centering input variables and intepreting resulting parameters
- assumptions and unarticulated priors
- understanding residual variance (Gaussian)
- understanding all of the above graphically
- understanding and plotting output of lm
- notation and linear algebra review: $X\beta$

Linear regression, ANOVA, ANCOVA, and multiple regression models are all species cases of general linear models (hereafter "linear models"). In all of these cases, we have observed some response variable y, which is potentially modeled as a function of some covariate(s) $x_1, x_2, ..., x_p$.

Model of the mean

If we have no covariates of interest, then we may be interested in estimating the population mean and variance of the random variable Y based on n observations, corresponding to the values $y_1, ..., y_n$. Here, capital letters indicate the random variable, and lowercase corresponds to realizations of that variable. This model is sometimes referred to as the "model of the mean".

```
# simulating a sample of y values from a normal distribution
y <- rnorm(20)
plot(y)
```

We have two parameters to estimate: the mean of Y, which we'll refer to as μ, and the variance of Y, which we'll refer to as σ^2. Here, and in general, we will use greek letters to refer to parameters. If Y is normally distributed, then we can assume that the realizations or samples y that we observe are also normally distributed: $y \sim N(\mu, \sigma^2)$. Here and elsewhere, the $\sim$ symbol represents that some quantity "is distributed as" something else (usually a probability distribution). You can also think of $\sim$ as meaning "is sampled from". A key concept here is that we are performing statistical inference, meaning we are trying to learn about (estimate) population-level parameters with sample data. In other words, we are not trying to learn about the sample mean $\bar{y}$ or sample variance of y. These can be calculated and treated as known once we have observed a particular collection of y values. The unknown quantities μ and σ^2 are the targets of inference.

Fitting this model (and linear models in general) is possible in R with the `lm` function. For this rather simple model, we can estimate the parameters as follows:

```
# fitting a model of the mean with lm
m <- lm(y ~ 1)
summary(m)
```

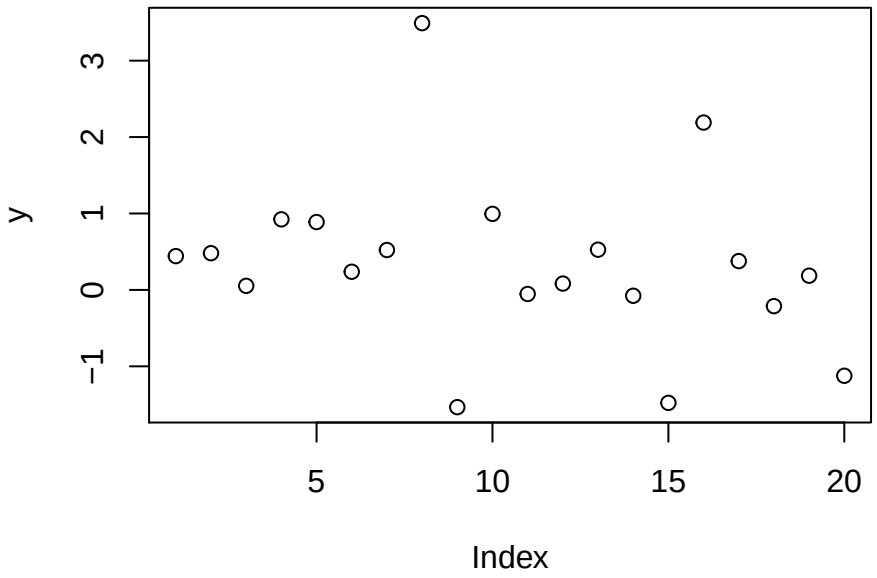


Figure 1: A set of observed y values, $n = 20$.

```
##
## Call:
## lm(formula = y ~ 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.88003 -0.40485 -0.03858  0.27088  3.14504
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.3459     0.2528   1.368    0.187
##
## Residual standard error: 1.131 on 19 degrees of freedom
```

The summary of our model object `m` provides a lot of information. For reasons that will become clear shortly, the estimated population mean is referred to as the "Intercept". Here, we get a point estimate for the population mean μ: 0.346 and an estimate of the residual standard deviation σ: 1.131, which we can square to get an estimate of the residual variance σ^2: 1.279.

Linear regression

Often, we are interested in estimating the mean of Y as a function of some other variable, say X. Simple linear regression assumes that y is again sampled from a normal distribution, but this time the mean or expected value of y is a function of x:

$$y_i \sim N(\mu_i, \sigma^2)$$

$$\mu_i = \alpha + \beta x_i$$

Here, subscripts indicate which particular value of y and x we're talking about. Specifically, we observe n pairs of values: $(x_i, y_i), ..., (x_n, y_n)$, with all x values known exactly. Linear regression models can equivalently be written as follows:

$$y_i = \alpha + \beta x_i + \epsilon_i$$

$$\epsilon_i \sim N(0, \sigma^2)$$

Key assumptions here are that each of the error terms $\epsilon_1, ..., \epsilon_n$ are normally distributed around zero with some variance (i.e., the error terms are identically distributed), and that the value of ϵ_1 does not affect the value of any other ϵ (i.e., the errors are independent). This combination of assumptions is often referred to as "independent and identically distributed" or i.i.d. Equivalently, given some particular x_i and a set of linear regression parameters, the distribution of y_i is normal. A common misconception is that linear regression assumes the distribution of y is normal. This is wrong - linear regression assumes that the error terms are normally distributed. The assumption that the variance σ^2 is constant for all values of x is referred to as homoskedasticity. Rural readers may find it useful to think of skedasticity as the amount of "skedaddle" away from the regression line in the y values. If the variance is changing across values of x, then the assumption of homoskedasticity is violated and you've got a heteroskedasticity problem.

```
# simulate and plot x and y values
n <- 50
x <- runif(n)
alpha <- -2
beta <- 3
sigma <- .4
y <- rnorm(n, mean = alpha + beta * x, sd = sigma)
plot(x, y)

# add known mean function
lines(x = x, y = alpha + beta * x, col='blue')
legend('topleft',
       pch = c(1, NA), lty = c(NA, 1),
       col = c('black', 'blue'),
       legend = c('Observed data', 'E(y | x)'),
       bty = 'n')
```

The normality assumption means that the probability density of y is highest at the value $\alpha + \beta x$, where the regression line is, and falls off away from the line according to the normal probablity density. This graphically looks like a bell 'tube' along the regression line, adding a dimension along x to the classic bell 'curve'.

Model fitting

Linear regression parameters α, β, and σ^2 can be estimated with `lm`. The syntax is very similar to the previous model, except now we need to include our covariate `x` in the formula (the first argument to the `lm` function).

```
m <- lm(y ~ x)
summary(m)
```

```
##
## Call:
```

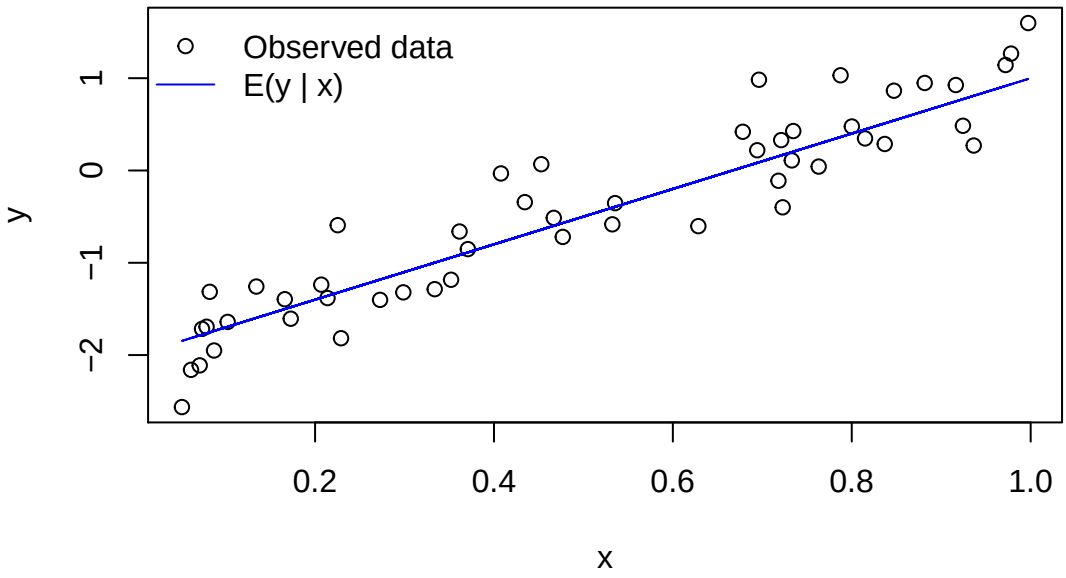


Figure 2: Simulated data from a linear regression model. The true expected value of y given x, E(y | x), is shown as a line.

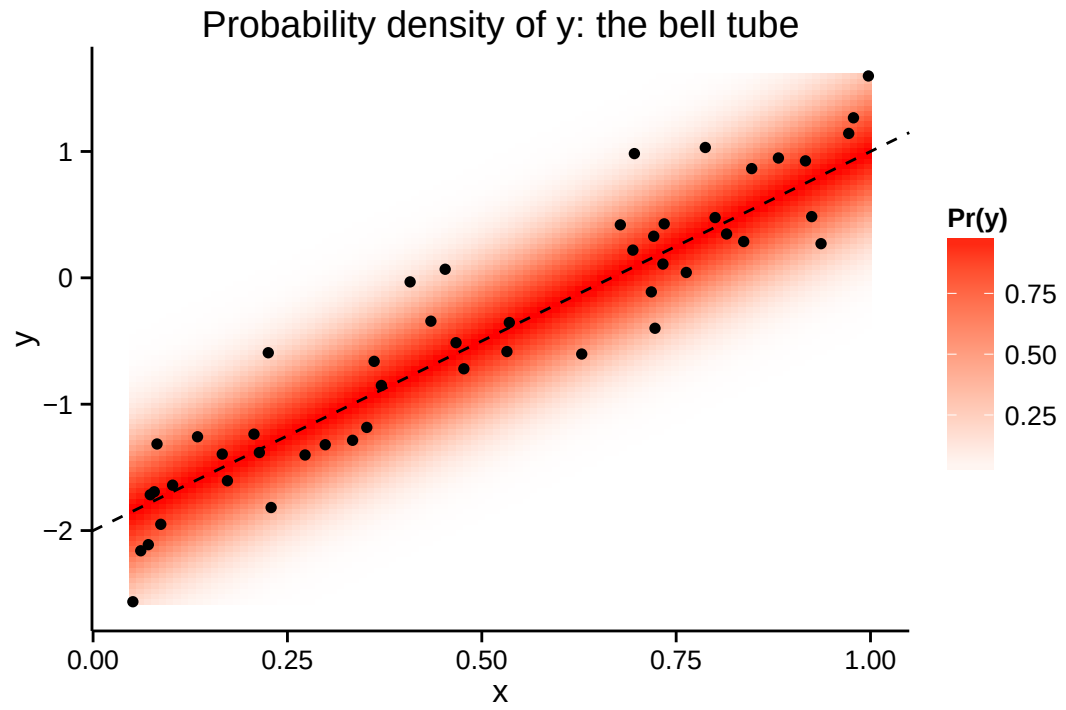


Figure 3: Graphical depiction of the linear regression normality assumption. The probability density of y is shown in color. Higher probabilities are shown as more intense colors, and regions with low probabilities are lighter.

```
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.6789 -0.2523  0.0295  0.1931  0.8110
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2.0475     0.1008  -20.31   <2e-16 ***
## x             3.1882     0.1720   18.54   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3708 on 48 degrees of freedom
## Multiple R-squared:  0.8774, Adjusted R-squared:  0.8749
## F-statistic: 343.7 on 1 and 48 DF,  p-value: < 2.2e-16
```

The point estimate for the parameter α is called "(Intercept)". This is because our estimate for α is the y-intercept of the estimated regression line when $x = 0$ (recall that $y_i = \alpha + \beta x_i + \epsilon_i$). The estimate for β is called "x", because it is a coefficient associated with the variable "x" in this model. This parameter is often referred to as the "slope", because it represents the increase in the expected value of y for a one unit increase in x (the rise in y over run in x). Point estimates for the standard deviation and variance of ϵ can be extracted as before (`summary(m)$sigma` and `summary(m)$sigma^2`).

Centering and scaling covariates

Often, it's a good idea to "center" covariates so that they have a mean of zero ($\bar{x} = 0$). This is achieved by subtracting the sample mean of a covariate from the vector of covariate values ($x - \bar{x}$). It's also useful to additionally scale covariates so that they are all on a common and unitless scale. While many will divide each covariate by its standard deviation, Gelman and Hill (pg. 57) recommend dividing by twice the standard deviation (s_x) so that binary covariates are transformed from $x \in \{0, 1\}$ to $x_t \in \{-0.5, 0.5\}$, where x_t is the transformed covariate: $x_t = \frac{x - \bar{x}}{2s_x}$. If covariates are not centered and scaled, then it is common to observe correlations between estimated slopes and intercepts.

So, we expect that in this case, the estimates for the intercept and slope must be negatively correlated. This is borne out in the confidence region for our estimates of α and β. Usually, people inspect univariate confidence intervals for parameters, e.g.,

```
confint(m)
```

```
##                  2.5 %    97.5 %
## (Intercept) -4.1374240 -2.683891
## x            0.8961927  1.314310
```

This is misleading because our estimates for these parameters are correlated. For any given value of the intercept, there are only certain values of the slope that are supported. To assess this possibility, we might also be interested in the bivariate confidence ellipse for these two parameters. We can evaluate this quantity graphically as follows with some help from the `car` package:

```
library(car)
confidenceEllipse(m)
```

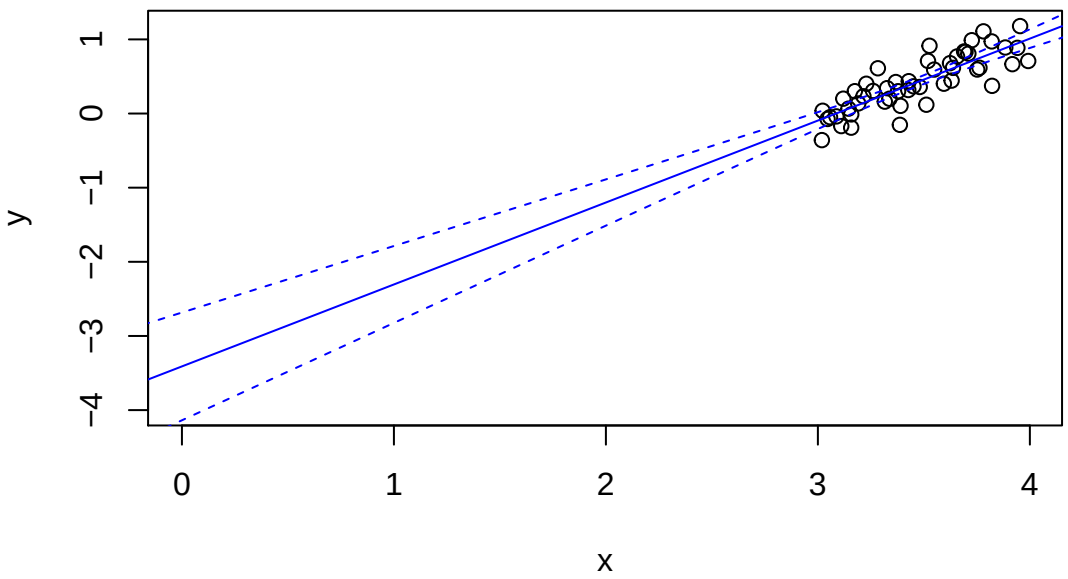


Figure 4: Linear regression line of best fit with 95% confidence intervals for the line. Notice that if the slope is lower (the upper dashed line) then the intercept necessarily goes up, and if the slope is higher (the lower dashed line), the intercept must decrease.

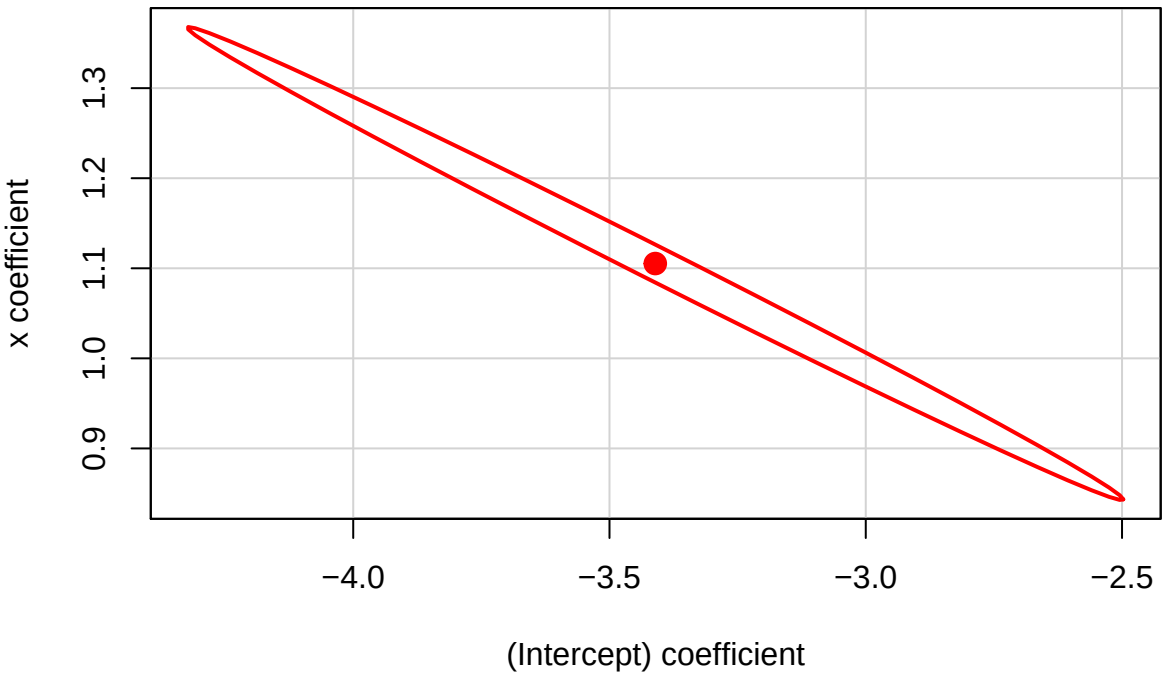


Figure 5: The bivariate confidence region for the slope and intercept. Correlation implies that for any particular value of alpha or beta, only a small subset of the values of the other parameter are supported. This information is not available by considering the univariate confidence intervals alone.

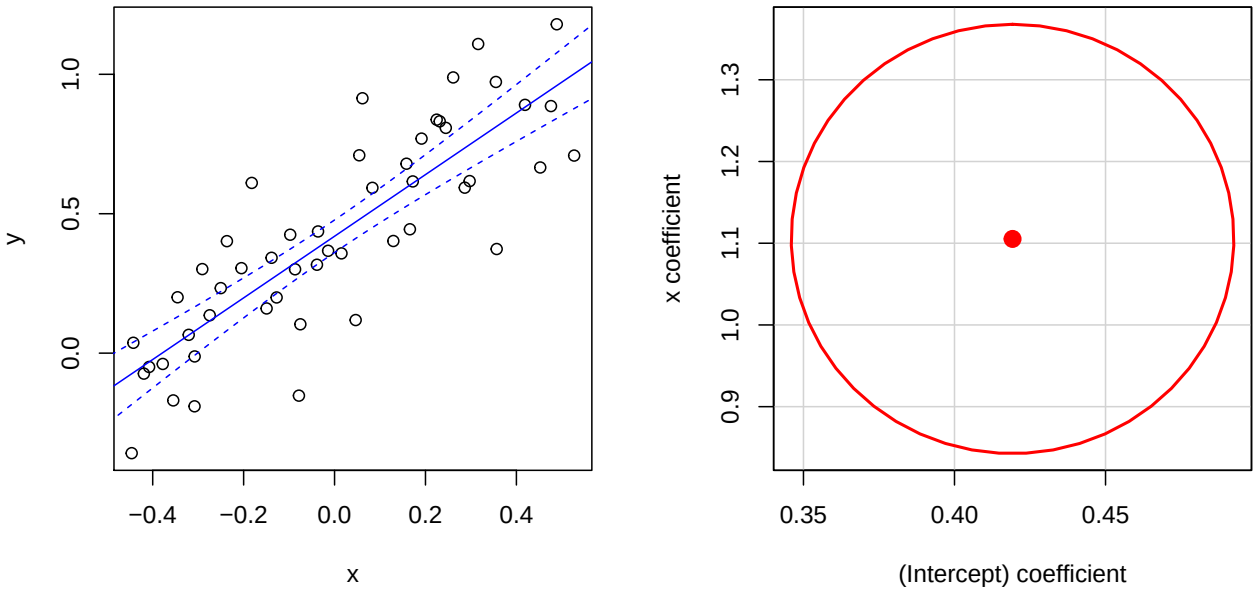


Figure 6: New line of best fit with confidence ellipses for the slope and intercept after centering the covariate x

This is not great. We want to be able to directly use the univariate confidence intervals. Our problem can be solved by centering x:

Now there is no serious correlation in the estimates and we are free to use the univariate confidence intervals without needing to consider the joint distribution of the slope and intercept. This trick helps with interpretation, but it will also prove useful later in the course in the context of Markov chain Monte Carlo (MCMC) sampling.

Checking assumptions

We have assumed that the distribution of error terms is normally distributed, and this assumption is worth checking. Below, we plot a histogram of the residuals (another name for the ϵ parameters) along with a superimposed normal probability density so that we can check normality.

```
hist(resid(m), breaks = 20, freq = F,
     main = 'Histogram of model residuals')
curve_x <- seq(min(resid(m)), max(resid(m)), .01)
lines(curve_x, dnorm(curve_x, 0, summary(m)$sigma))
```

Even when the assumption of normality is correct, it is not always obvious that the residuals are normally distributed. Another useful plot for assessing normality of errors is a quantile-quantile or Q-Q plot. If the residuals do not deviate much from normality, then the points in a Q-Q plot won't deviate much from the dashed one-to-one line. If points lie above or below the line, then the residual is larger or smaller, respectively, than expected based on a normal distribution.

```
plot(m, 2)
```

To assess heteroskedasticity, it is useful to inspect a plot of the residuals vs. fitted values, e.g. `plot(m, 1)`. If it seems as though the spread or variance of residuals varies across the range of fitted values, then it may be worth worrying about homoskedasticity and trying some transformations to fix the problem.

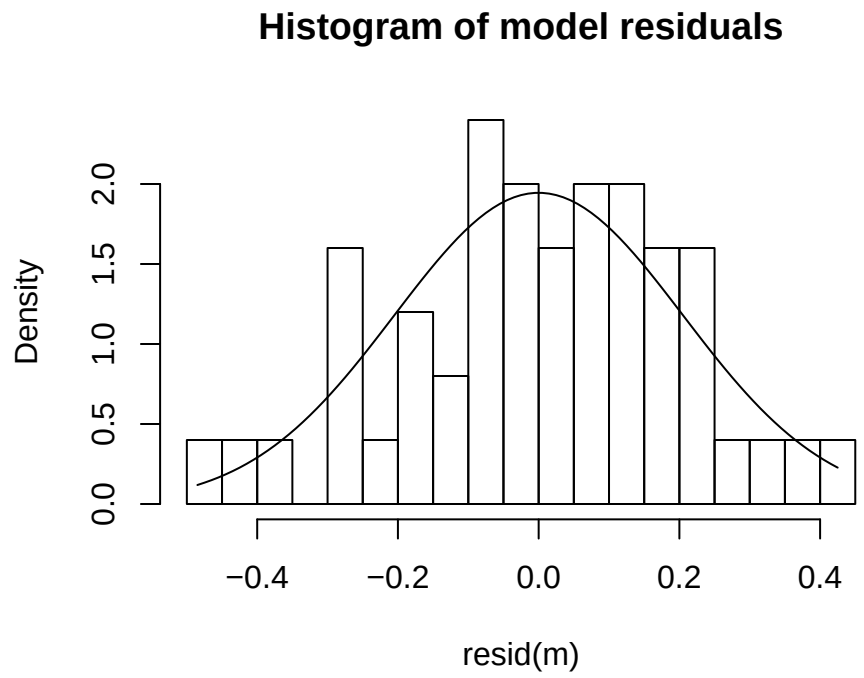


Figure 7: Simulated data from a linear regression model.

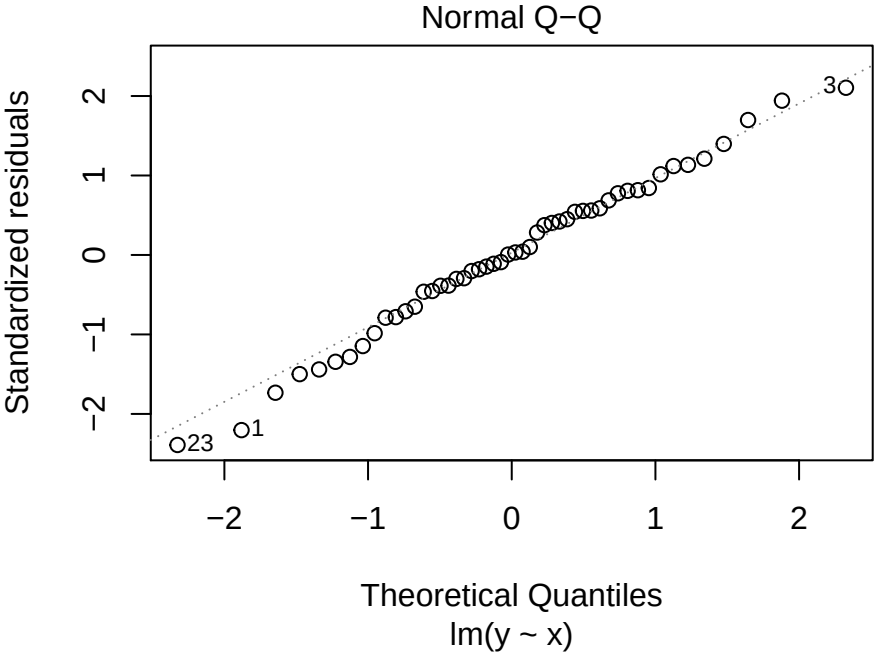


Figure 8: A quantile-quantile plot to assess normality of residuals.

Analysis of variance

Sometimes, the covariate of interest is not continuous but instead categorical (e.g., "chocolate", "strawberry", or "vanilla"). We might again wonder whether the mean of a random variable Y depends on the value of this covariate. However, we cannot really estimate a meaningful "slope" parameter, because in this case x is not continuous. Instead, we might formulate the model as follows:

$$y_i \sim N(\alpha_{j[i]}, \sigma^2)$$

Where α_j is the mean of group j, and we have J groups total. The notation $\alpha_{j[i]}$ represents the notion that the i^{th} observation corresponds to group j, and we are going to assume that all observations in the j^{th} group have the same mean, α_j. The above model is perfectly legitimate, and our parameters to estimate are the group means $\alpha_1, ..., \alpha_J$ and the residual variance σ^2. This parameterization is called the "means" parameterization, and though it is perhaps easier to understand than the following alternative, it is less often used.

This model is usually parameterized not in terms of the group means, but rather in terms of an intercept (corresponding to the mean of one "reference" group), and deviations from the intercept (differences between a group of interest and the intercept). For instance, in R, the group whose mean is the intercept (the "reference" group) will be the group whose name comes first alphabetically. Either way, we will estimate the same number of parameters. So if our groups are "chocolate", "strawberry", and "vanilla", R will assign the group "chocolate" to be the intercept, and provide 2 more coefficient estimates for the difference between the estimated group mean of strawberry vs. chocoloate, and vanilla vs. chocolate.

This parameterization can be written as

$$y_i \overset{iid}{\sim} N(\mu_0 + \beta_{j[i]}, \sigma^2)$$

where μ_0 is the "intercept" or mean of the reference group, and β_j represents the difference in the population mean of group j compared to the reference group (if j is the reference group, the $\beta_j = 0$). Traditionally this model is called simple one-way analysis of variance, but we view it simply as another special case of a linear model.

The following example illustrates some data simulation, visualization, and parameter estimation in this context. Specifically, we assess 60 humans for their taste response to three flavors of ice cream. We want to extrapolate from our sample to the broader population of all ice cream eating humans to learn whether in general people think ice cream tastiness varies as a function of flavor.

```r
# simulate and visualize data
n <- 60
x <- rep(c("chocolate", "strawberry", "vanilla"), length.out = n)
x <- factor(x)
sigma <- 1
mu_y <- c(chocolate = 3.352, strawberry = .93, vanilla = 1.5)
y <- rnorm(n, mu_y[x], sigma)

library(ggplot2)
ggplot(data.frame(x, y), aes(x, y)) +
  geom_jitter(position = position_jitter(width=.1)) +
  xlab('Group') +
  ylab('Tastiness')
```

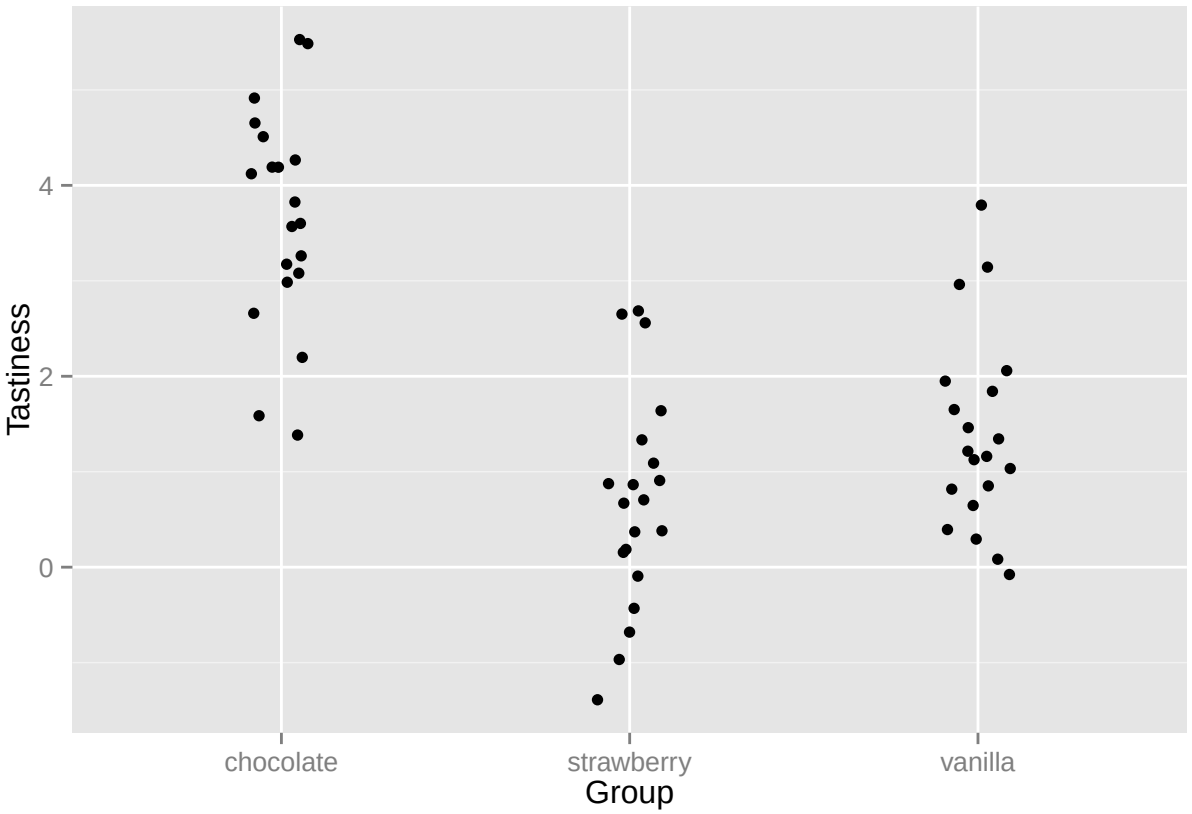


Figure 9: Simulated data on tastiness across three groups. Each point is an observation

Model fitting

We can estimate our parameters with the `lm` function (this should be a strong hint that there are not huge differences between linear regression and ANOVA). The syntax is exactly the same as with linear regression. The only difference is that our input `x` is not numeric, but a character vector.

```
m <- lm(y ~ x)
summary(m)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.27506 -0.60318 -0.05057  0.57339  2.40576
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.6592     0.2468  14.828  < 2e-16 ***
## xstrawberry  -2.9836     0.3490  -8.549 8.47e-12 ***
## xvanilla     -2.2717     0.3490  -6.510 2.08e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.104 on 57 degrees of freedom
## Multiple R-squared:  0.5832, Adjusted R-squared:  0.5686
## F-statistic: 39.88 on 2 and 57 DF,  p-value: 1.473e-11
```

Because chocolate comes first alphabetically, it is the reference group and the "(Intercept)" estimate corresponds to the estimate of the group-level mean for chocolate. The other two estimates are contrasts between the other groups and this reference group, i.e. "xstrawberry" is the estimated difference between the group mean for strawberry and the reference group.

If we wish instead to use a means paramaterization, we need to supress the intercept term in our model as follows:

```
m <- lm(y ~ 0 + x)
summary(m)
```

```
##
## Call:
## lm(formula = y ~ 0 + x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.27506 -0.60318 -0.05057  0.57339  2.40576
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## xchocolate    3.6592     0.2468  14.828  < 2e-16 ***
## xstrawberry   0.6756     0.2468   2.738  0.00824 **
## xvanilla      1.3874     0.2468   5.622 5.92e-07 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.104 on 57 degrees of freedom
## Multiple R-squared:  0.8196, Adjusted R-squared:  0.8101
## F-statistic: 86.33 on 3 and 57 DF,  p-value: < 2.2e-16
```

Arguably, this approach is more useful because it simplifies the construction of confidence intervals for the group means:

```
confint(m)
```

```
##                 2.5 %   97.5 %
## xchocolate  3.1650288 4.153310
## xstrawberry 0.1814615 1.169742
## xvanilla    0.8932899 1.881571
```

Checking assumptions

Our assumptions in this simple one way ANOVA context are identical to our assumptions with linear regression. Specifically, we assumed that our errors are independently and identically distributed, and that the variance is constant for each group (homoskedasticity). The built in `plot` method for `lm` objects is designed to return diagnostic plots that help to check these assumptions.

```
par(mfrow=c(2, 2))
plot(m)
```

General linear models

We have covered a few special cases of general linear models, which are usually written as follows:

$$y \overset{iid}{\sim} N(X\beta, \sigma^2)$$

Where y is a vector consisting of n observations, X is a "design" matrix with n rows and p columns, and β is a vector of p parameters. There are multivariate general linear models (e.g., MANOVA) where the response variable is a matrix and a covariance matrix is used in place of a scalar variance parameter, but we'll stick to univariate models for simplicity. The key point here is that the product of X and β provides the mean of the normal distribution from which y is drawn. From this perspective, the difference between the model of the mean, linear regression, ANOVA, etc., lies in the structure of X and subsequent interpretation of the parameters β. This is a very powerful idea that unites many superficially disparate approaches. It also is the reason that these models are considered "linear", even though a regression line might by quite non-linear (e.g., polynomial regression). These models are linear in their parameters, meaning that our expected value for the response y is a **linear combination** (formal notion) of the parameters. If a vector of expected values for y in some model cannot be represented as $X\beta$, then it is not a linear model.

In the model of the mean, X is an n by 1 matrix, with each element equal to 1 (i.e. a vector of ones). With linear regression, X's first column is all ones (corresponding to the intercept parameter), and the second column contains the values of the covariate x. In ANOVA, the design matrix X will differ between the means and effects parameterizations. With a means parameterization, the entries in column j will equal one if observation (row) i is in group j, and entries are zero otherwise. If you are not comfortable with matrix multiplication, it's worth investing some effort so that you can understand why $X\beta$ is such a powerful construct.

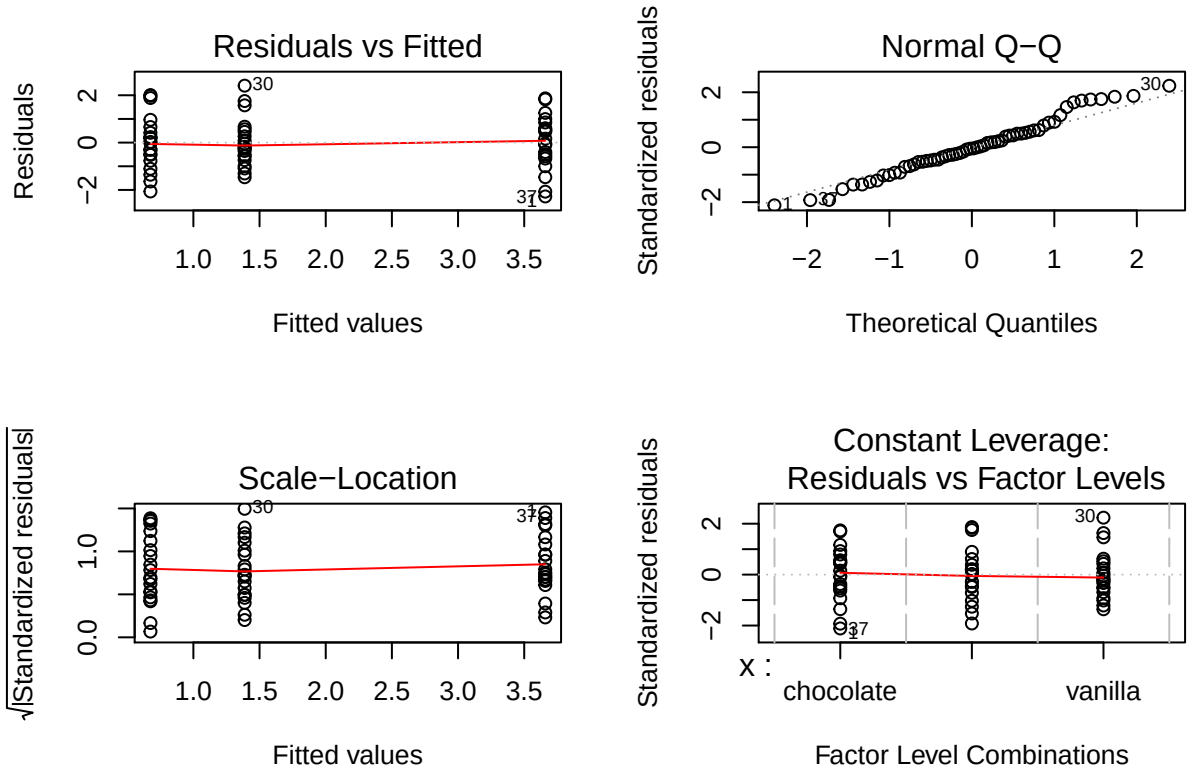


Figure 10: Default diagnostic plots for `lm` objects.

Can you figure out the structure of X with R's default effects parameterization? You can check your work with `model.matrix(m)`, where `m` is a model that you've fitted with `lm`.

Interactions between covariates

Often, the effect of one covariate depends on the value of another covariate. This is referred to as "interaction" between the covariates. Interactions can exist between two or more continuous and/or nominal covariates. These situations have special names in the classical statistics literature. For example, models with interactions between nominal covariates fall under "factorial ANOVA", while those with interactions between a continuous and a nominal covariate are referred to as "analysis of covariance (ANCOVA)". Here we prefer to consider these all as special cases of general linear models.

Interactions between two continuous covariates

Here we demonstrate simulation and estimation for a model with an interaction between two continuous covariates. Notice that in the simulation, we have exploited the $X\beta$ construct to generate a vector of expected values for y.

```
n <- 50
x1 <- rnorm(n)
x2 <- rnorm(n)
beta <- c(.5, 1, -1, 2)
sigma <- 1
X <- matrix(c(rep(1, n), x1, x2, x1 * x2), nrow=n)
mu_y <- X %*% beta
```

```r
y <- rnorm(n, mu_y, sigma)
m <- lm(y ~ x1 * x2)
summary(m)
```

```
##
## Call:
## lm(formula = y ~ x1 * x2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1792 -0.8670 -0.0096  0.7014  2.4582
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.1921     0.1810   1.061 0.294043
## x1            1.1598     0.1880   6.168 1.62e-07 ***
## x2           -0.6731     0.1858  -3.623 0.000724 ***
## x1:x2         2.1391     0.2244   9.534 1.82e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.246 on 46 degrees of freedom
## Multiple R-squared:  0.7953, Adjusted R-squared:  0.782
## F-statistic: 59.59 on 3 and 46 DF,  p-value: 7.027e-16
```

Visualizing these models is tricky, because we are in 3d space (with dimensions x_1, x_2, and y), but contour plots can be effective and leverage peoples' understanding of topographic maps.

```r
# visualizing the results in terms of the linear predictor
lo <- 40
x1seq <- seq(min(x1), max(x1), length.out = lo)
x2seq <- seq(min(x2), max(x2), length.out = lo)
g <- expand.grid(x1=x1seq, x2=x2seq)
g$e_y <- beta[1] + beta[2] * g$x1 + beta[3] * g$x2 + beta[4] * g$x1 * g$x2
ggplot(g, aes(x=x1, y=x2)) +
  geom_tile(aes(fill=e_y)) +
  stat_contour(aes(z=e_y), col='grey') +
  scale_fill_gradient2() +
  geom_point(data=data.frame(x1, x2))
```

Alternatively, you might check out the `effects` package:

```r
library(effects)
plot(allEffects(m))
```

Interactions between two categorical covariates

Here we demonstrate interaction between two categorical covariates, using the `diamonds` dataset which is in the `ggplot2` package. We are interested in the relationship between diamond price, cut quality, and color.

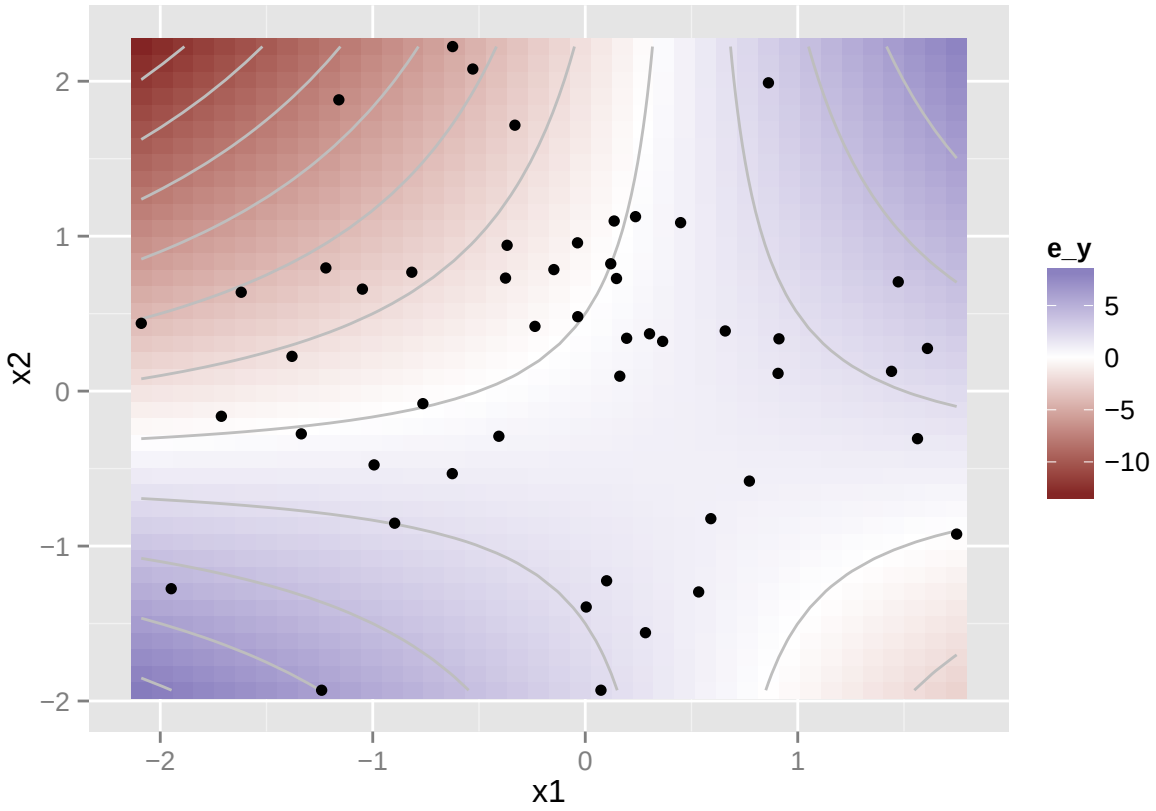


Figure 11: Contour plot of the linear predictor. Lines represent E(y) isoclines, where the expected value of y is unchanged.

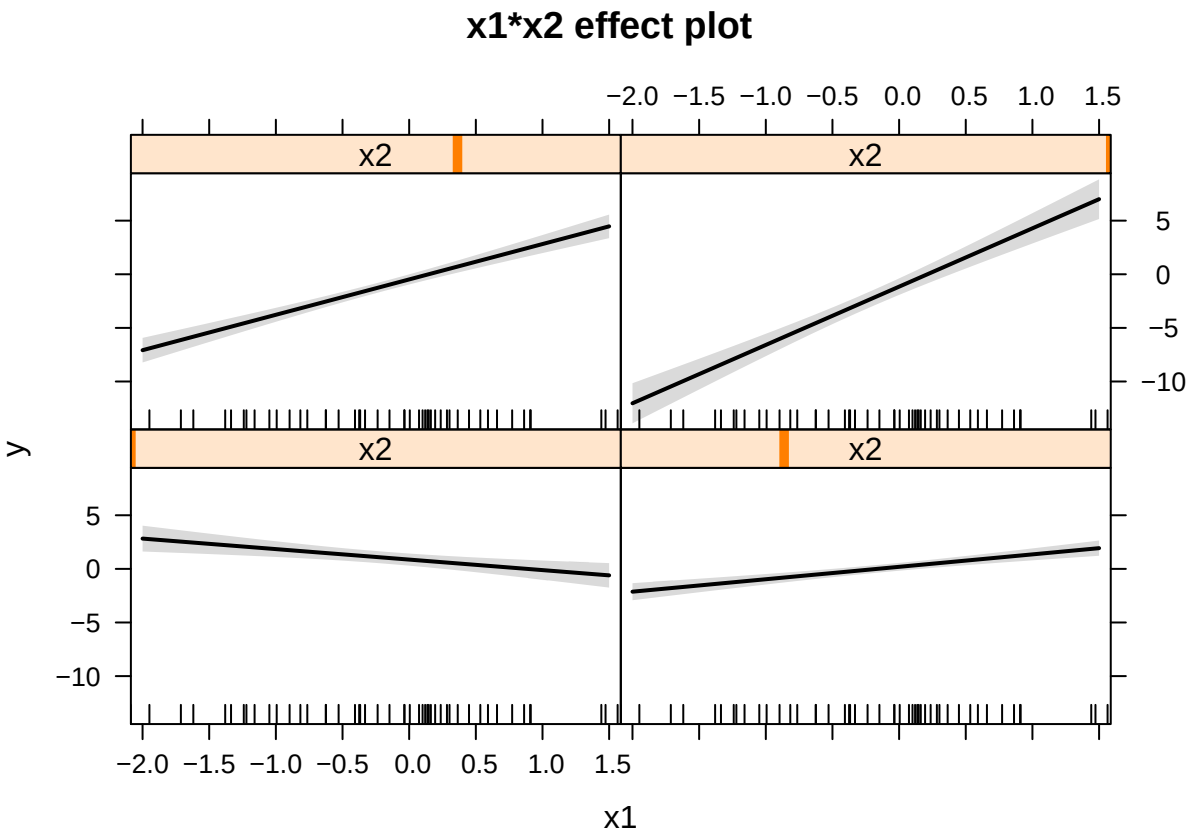


Figure 12: Default effects plot from the `effects` package.

```
str(ToothGrowth)
```

```
## 'data.frame':    60 obs. of  3 variables:
##  $ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...
##  $ supp: Factor w/ 2 levels "OJ","VC": 2 2 2 2 2 2 2 2 2 2 ...
##  $ dose: num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
```

```
ToothGrowth$dose <- factor(ToothGrowth$dose)
ggplot(ToothGrowth, aes(x=interaction(dose, supp), y=len)) +
  geom_point()
```

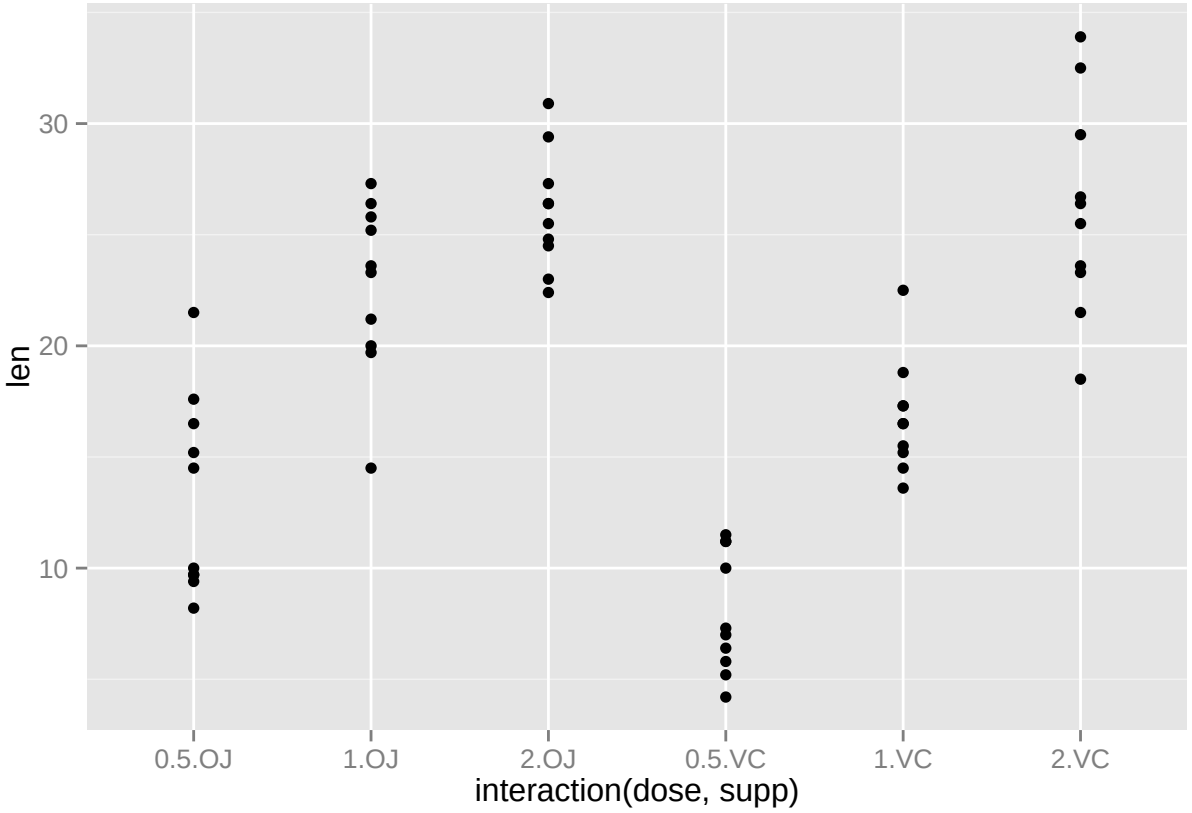


Figure 13: Stripchart of tooth growth across combinations of dose and supplements.

In general, visualizing the raw data is a good idea. However, we might also be interested in a table with group-wise summaries, such as the sample means, standard deviations, and sample sizes.

```
library(dplyr)
ToothGrowth %>%
  group_by(dose, supp) %>%
  summarize(mean = mean(len),
            sd = sd(len),
            n = n())
```

```
## Source: local data frame [6 x 5]
## Groups: dose [?]
```

```
##
##     dose   supp  mean       sd     n
##   (fctr) (fctr) (dbl)    (dbl) (int)
## 1    0.5     OJ 13.23 4.459709    10
## 2    0.5     VC  7.98 2.746634    10
## 3      1     OJ 22.70 3.910953    10
## 4      1     VC 16.77 2.515309    10
## 5      2     OJ 26.06 2.655058    10
## 6      2     VC 26.14 4.797731    10
```

We can construct a model to estimate the effect of dose, supplement, and their interaction.

```
m <- lm(len ~ dose * supp, data = ToothGrowth)
summary(m)
```

```
##
## Call:
## lm(formula = len ~ dose * supp, data = ToothGrowth)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -8.20  -2.72  -0.27   2.65   8.27
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    13.230      1.148  11.521 3.60e-16 ***
## dose1           9.470      1.624   5.831 3.18e-07 ***
## dose2          12.830      1.624   7.900 1.43e-10 ***
## suppVC         -5.250      1.624  -3.233  0.00209 **
## dose1:suppVC   -0.680      2.297  -0.296  0.76831
## dose2:suppVC    5.330      2.297   2.321  0.02411 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.631 on 54 degrees of freedom
## Multiple R-squared:  0.7937, Adjusted R-squared:  0.7746
## F-statistic: 41.56 on 5 and 54 DF,  p-value: < 2.2e-16
```

This summary gives the effects-parameterization version of the summary. The "(Intercept)" refers to the combination of factor levels that occur first alphabetically: in this case, a dose of 0.5 with the "OJ" supplement. The coefficients for `dose1` and `dose2` represent estimated contrasts for these two groups relative to the intercept. The coefficient for `suppVC` represents the contrast between the "VC" and "OJ" levels of supplement when the dose is 0.5. The interaction terms represent the difference in the effect of VC for `dose1` and `dose2` relative to a dose of 0.5. None of this is particularly intuitive, but this information can be gleaned by inspecting the design matrix X produced by `lm` in the process of fitting the model. Inspecting the design matrix along with the dataset to gives a better sense for how X relates to the factor levels:

```
cbind(model.matrix(m), ToothGrowth)
```

```
##    (Intercept) dose1 dose2 suppVC dose1:suppVC dose2:suppVC  len supp dose
## 1            1     0     0      1            0            0  4.2   VC  0.5
## 2            1     0     0      1            0            0 11.5   VC  0.5
```

```
## 3          1     0     0     1          0          0  7.3   VC  0.5
## 4          1     0     0     1          0          0  5.8   VC  0.5
## 5          1     0     0     1          0          0  6.4   VC  0.5
## 6          1     0     0     1          0          0 10.0   VC  0.5
## 7          1     0     0     1          0          0 11.2   VC  0.5
## 8          1     0     0     1          0          0 11.2   VC  0.5
## 9          1     0     0     1          0          0  5.2   VC  0.5
## 10         1     0     0     1          0          0  7.0   VC  0.5
## 11         1     1     0     1          1          0 16.5   VC    1
## 12         1     1     0     1          1          0 16.5   VC    1
## 13         1     1     0     1          1          0 15.2   VC    1
## 14         1     1     0     1          1          0 17.3   VC    1
## 15         1     1     0     1          1          0 22.5   VC    1
## 16         1     1     0     1          1          0 17.3   VC    1
## 17         1     1     0     1          1          0 13.6   VC    1
## 18         1     1     0     1          1          0 14.5   VC    1
## 19         1     1     0     1          1          0 18.8   VC    1
## 20         1     1     0     1          1          0 15.5   VC    1
## 21         1     0     1     1          0          1 23.6   VC    2
## 22         1     0     1     1          0          1 18.5   VC    2
## 23         1     0     1     1          0          1 33.9   VC    2
## 24         1     0     1     1          0          1 25.5   VC    2
## 25         1     0     1     1          0          1 26.4   VC    2
## 26         1     0     1     1          0          1 32.5   VC    2
## 27         1     0     1     1          0          1 26.7   VC    2
## 28         1     0     1     1          0          1 21.5   VC    2
## 29         1     0     1     1          0          1 23.3   VC    2
## 30         1     0     1     1          0          1 29.5   VC    2
## 31         1     0     0     0          0          0 15.2   OJ  0.5
## 32         1     0     0     0          0          0 21.5   OJ  0.5
## 33         1     0     0     0          0          0 17.6   OJ  0.5
## 34         1     0     0     0          0          0  9.7   OJ  0.5
## 35         1     0     0     0          0          0 14.5   OJ  0.5
## 36         1     0     0     0          0          0 10.0   OJ  0.5
## 37         1     0     0     0          0          0  8.2   OJ  0.5
## 38         1     0     0     0          0          0  9.4   OJ  0.5
## 39         1     0     0     0          0          0 16.5   OJ  0.5
## 40         1     0     0     0          0          0  9.7   OJ  0.5
## 41         1     1     0     0          0          0 19.7   OJ    1
## 42         1     1     0     0          0          0 23.3   OJ    1
## 43         1     1     0     0          0          0 23.6   OJ    1
## 44         1     1     0     0          0          0 26.4   OJ    1
## 45         1     1     0     0          0          0 20.0   OJ    1
## 46         1     1     0     0          0          0 25.2   OJ    1
## 47         1     1     0     0          0          0 25.8   OJ    1
## 48         1     1     0     0          0          0 21.2   OJ    1
## 49         1     1     0     0          0          0 14.5   OJ    1
## 50         1     1     0     0          0          0 27.3   OJ    1
## 51         1     0     1     0          0          0 25.5   OJ    2
## 52         1     0     1     0          0          0 26.4   OJ    2
## 53         1     0     1     0          0          0 22.4   OJ    2
## 54         1     0     1     0          0          0 24.5   OJ    2
## 55         1     0     1     0          0          0 24.8   OJ    2
## 56         1     0     1     0          0          0 30.9   OJ    2
```

```
## 57           1     0     1      0            0            0 26.4   OJ    2
## 58           1     0     1      0            0            0 27.3   OJ    2
## 59           1     0     1      0            0            0 29.4   OJ    2
## 60           1     0     1      0            0            0 23.0   OJ    2
```

Often, researchers want to know if interactions need to be included in the model. From a null hypothesis significance testing perspective, we can evaluate the 'significance' of the interaction term as follows:

```
anova(m)
```

```
## Analysis of Variance Table
##
## Response: len
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## dose       2 2426.43 1213.22  92.000 < 2.2e-16 ***
## supp       1  205.35  205.35  15.572 0.0002312 ***
## dose:supp  2  108.32   54.16   4.107 0.0218603 *
## Residuals 54  712.11   13.19
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We find that the interaction between dose and supplement is statistically significant, meaning that if we assume that there is no interaction, it is unlikely to observe data that are as or more extreme as what we have observed over the course of infinitely many replicated experiments that will probably never occur. Although this is far from intuitive, this approach has been widely used. We will introduce a more streamlined procedure in chapter 3 that 1) does not assume that the effect is zero to begin with, and 2) does not necessarily invoke a hypothetical infinite number of replicated realizations of the data, conditional on one particular parameter value. An alternative approach would be to use information theoretics to decide whether the interaction is warranted:

```
m2 <- lm(len ~ dose + supp, data = ToothGrowth)
AIC(m, m2)
```

```
##    df      AIC
## m   7 332.7056
## m2  5 337.2013
```

In the past decade following Burnham and Anderson's book on the topic, ecologists have leaned heavily on Akaike's information criterion (AIC), which is a relative measure of model quality (balancing goodness of fit with model complexity). Here we see that the original model `m` with interaction has a lower AIC value, and is therefore better supported. AIC can be considered to be similar to cross validation, approximating the ability of a model to predict future data.

Being somewhat lazy, we might again choose to plot the results of this model using the `effects` package.

```
plot(allEffects(m))
```

This is less than satisfying, as it does not show any data. All we see is model output. If the model is crap, then the output and these plots are also crap. But, evaluating the crappiness of the model is difficult when there are no data shown. Ideally, the data can be shown along with the estimated group means and some indication of uncertainty. If we weren't quite so lazy, we could use the `predict` function to obtain confidence intervals for the means of each group.

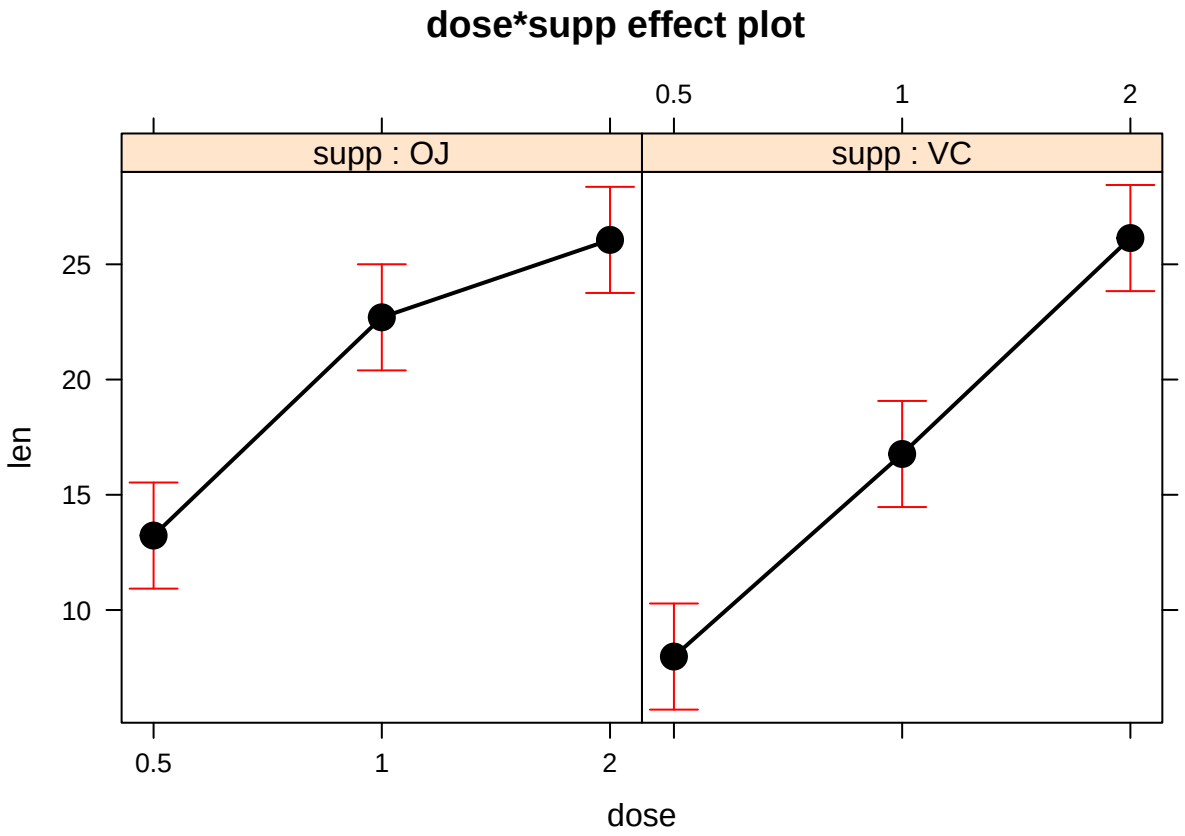


Figure 14: Default effects plot from the `effects` package.

```
# construct a new data frame for predictions
g <- expand.grid(supp = levels(ToothGrowth$supp),
                 dose = levels(ToothGrowth$dose))
p <- predict(m, g, interval = 'confidence', type='response')
predictions <- cbind(g, data.frame(p))

ggplot(ToothGrowth, aes(x=interaction(dose, supp), y=len)) +
  geom_segment(data=predictions,
               aes(y=lwr, yend=upr,
                   xend=interaction(dose, supp)), col='red') +
  geom_point(data=predictions, aes(y=fit), color='red', size=2, shape=2) +
  geom_jitter(position = position_jitter(width=.1), shape=1) +
  ylab("Length")
```

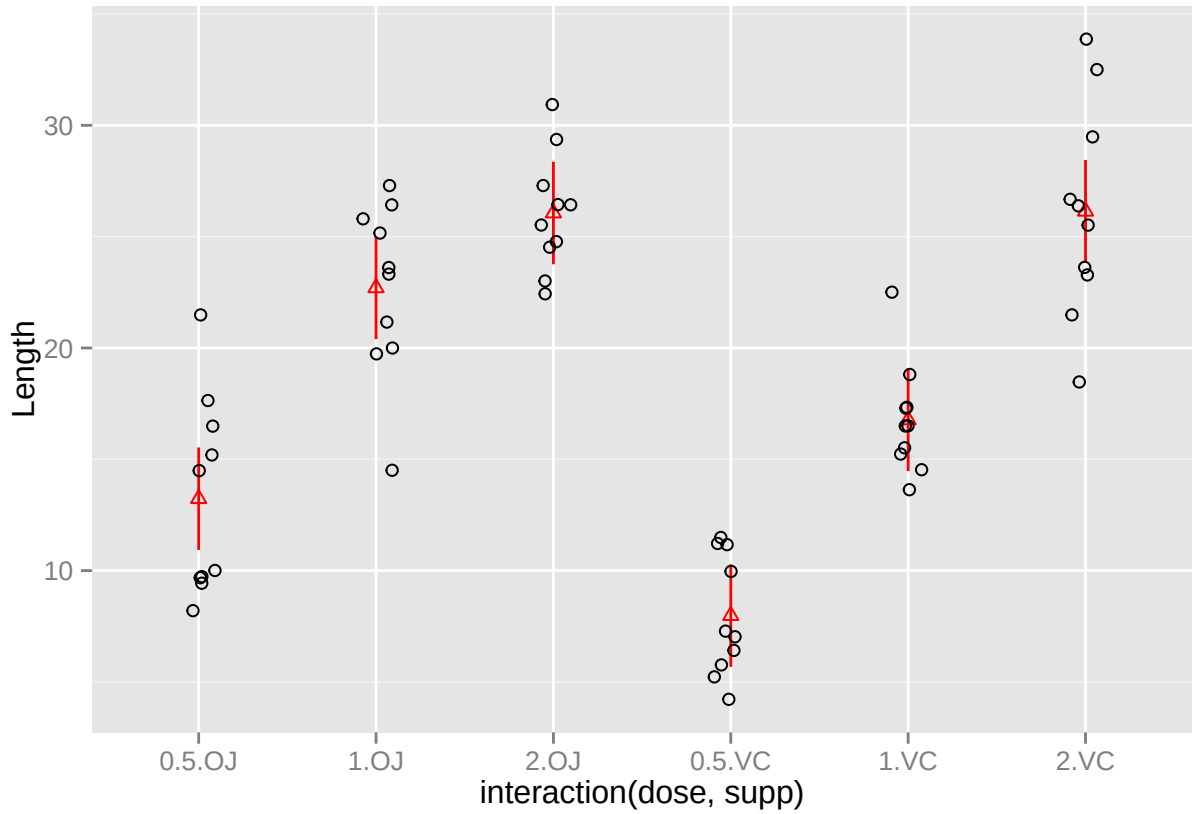


Figure 15: Raw data along with confidence intervals for the means of each group. Notice how much mor information is shown here compared to a bar chart with error bars.

This plot is nice because we can observe the data along with the model output. This makes it easier for readers to understand how the model relates to, fits, and does not fit the data. If you wish to obscure the data, you could make a bar plot with error pars to represent the standard errors. Although "dynamite" plots are common, we shall not include one here and we strongly recommend that you never produce such a plot (more here).

Interactions between continuous and categorical covariates

Sometimes, we're interested in interactions between continuous or numeric covariates and another covariate with discrete categorical levels. Again, this falls under the broad class of models used in analysis of covariance (ANCOVA).

```
x1 <- rnorm(n)
x2 <- factor(sample(c('A', 'B'), n, replace=TRUE))

# generate slopes and intercepts for the first and second groups
a <- rnorm(2)
b <- rnorm(2)
sigma <- .4

X <- matrix(c(ifelse(x2 == 'A', 1, 0),
              ifelse(x2 == 'B', 1, 0),
              ifelse(x2 == 'A', x1, 0),
              ifelse(x2 == 'B', x1, 0)
            ), nrow=n)

mu_y <- X %*% c(a, b)
y <- rnorm(n, mu_y, sigma)
```

Here the intercepts and slopes are allowed to vary for two groups. We can fit a model with an interaction between these covariates. The intercepts and slopes are estimated separately for the two groups.

```
m <- lm(y ~ x1 * x2)
summary(m)
```

```
##
## Call:
## lm(formula = y ~ x1 * x2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.99485 -0.24622  0.04654  0.26037  0.63571
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.96358    0.07611  12.661  < 2e-16 ***
## x1          -0.65639    0.08141  -8.063 2.39e-10 ***
## x2B         -0.35241    0.10687  -3.298  0.00189 **
## x1:x2B       0.37313    0.11879   3.141  0.00294 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3727 on 46 degrees of freedom
## Multiple R-squared:  0.6607, Adjusted R-squared:  0.6385
## F-statistic: 29.85 on 3 and 46 DF,  p-value: 7.235e-11
```

Let's plot the lines of best fit along with the data.

```
plot(x1, y, col=x2, pch=19)
legend('topright', col=1:2, legend=c('Group A', 'Group B'), pch=19)
abline(coef(m)[1], coef(m)[2])
abline(coef(m)[1] + coef(m)[3], coef(m)[2] + coef(m)[4], col='red')
```

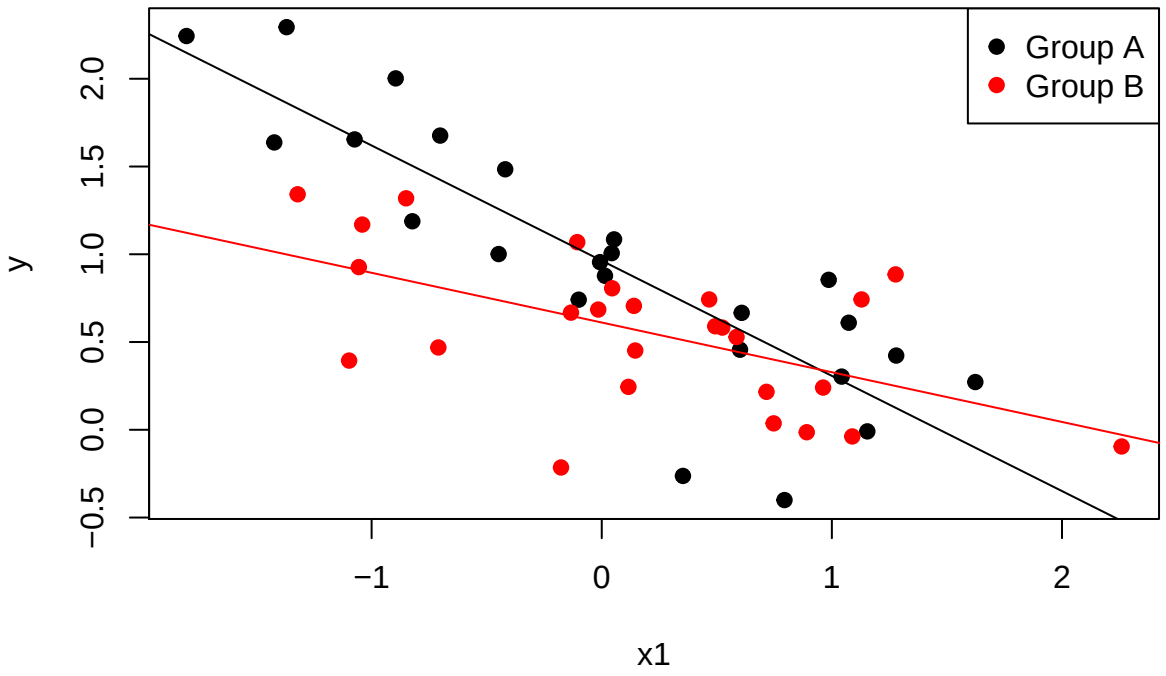


Figure 16: Lines of best fit from the ANCOVA model, along with the raw data.

The `abline` function, used above, adds lines to plots based on a y-intercept (first argument) and a slope (second argument). Do you understand why the particular coefficients that we used as inputs provide the desired intercepts and slopes for each group? If not, evaluate the design matrix X via `model.matrix(m)` and interpret the coefficients β corresponding to each column via `coef(m)`.

The general strategy of building a design matrix X is useful in many other contexts, such as when the errors are not normally distributed. Specifically, we will soon explore situations where our data may consist of binary observations or integer counts, and the $X\beta$ formulation still plays a central role. Indeed, later on in the course when we begin discussing random effects, it will be possible to represent specific types of model structures with two design matrices: one for the fixed effects X, and one for the random effects Z.

Further reading

Schielzeth, H. 2010. Simple means to improve the interpretability of regression coefficients. Methods in Ecology and Evolution 1:103–113.

Enqvist, L. 2005. The mistreatment of covariate interaction terms in linear model analyses of behavioural and evolutionary ecology studies. Animal Behaviour 70:967–971.

Gelman and Hill. 2009. *Data analysis using regression and multilevel/hierarchical models.* Chapter 3-4.

Chapter 2: Maximum likelihood estimation

Big picture

The likelihood is defined as the probability of the data, conditional on some parameter(s). Having observed some data, we often want to know which particular parameter values maximize the probability of those data. These parameter values are referred to as the maximum likelihood estimates.

The goal here is to connect the notion of a likelihood to probability distributions and models. We can obtain maximum likelihood estimates (MLEs) in a few ways: analytically, with brute force (direct search), and via optimization (e.g., the `optim` function).

Learning goals

- definition of likelihood
- single parameter models: obtaining a MLE with optim
- model of the mean with unknown variance
- fitting simple linear models with likelihood
- assumptions (especially related to independence)
- inference (what probability does the likelihood function represent?)

What is likelihood?

The likelihood function represents the probability of the data y, conditioned on the parameter(s) θ. Mathematically, the likelihood is $p(\boldsymbol{y}|\theta) = \mathcal{L}(\theta|\boldsymbol{y})$, where $\boldsymbol{y}$ is a (possibly) vector-valued sample of observations from the random variable $\boldsymbol{Y} = (Y_1, Y_2, ..., Y_n)$. More casually, the likelihood function tells us the probability of having observed the sample that we did under different values of the parameter(s) θ. It is important to recognize that θ is not treated as a random variable in the likelihood function (the data are treated as random variables). The likelihood is not the probability of θ conditional on the data $\boldsymbol{y}$; $p(y|\theta) \neq p(\theta|y)$. To calculate $p(\theta|y)$, we'll need to invert the above logic, and we can do so later with Bayes' theorem (also known as the law of inverse probability).

Joint probabilities of independent events

You may recall that if we have two events A and B, and we want to know the joint probability that both events A and B occur, we can generally obtain the joint probability as: $P(A, B) = P(A|B)P(B)$ or $P(A, B) = P(B|A)P(A)$. However, if the events A and B are independent, then $P(A|B) = P(A)$ and $P(B|A) = P(B)$. In other words, having observed that one event has happened, the probability of the other event is unchanged. In this case, we obtain the joint probability of two independent events as $P(A, B) = P(A)P(B)$. This logic extends to more than two independent events: $P(E_1, E_2, ..., E_n) = \prod_{i=1}^{n} E_i$, where E_i is the i^{th} event.

Why does this matter? Recall the independence assumption that we made in the construction of our linear models in the previous chapters: the error terms $\epsilon_i \sim N(0, \sigma^2)$, or equivalently the conditional distribution of y values y_i, $[y_i|\beta, \sigma^2] \sim N(X\beta, \sigma^2)$ are independent. Here the square brackets are used as a more compact version of probability notation, we could have also written $P(Y_i = y_i|\beta, \sigma^2)$, the probability that the random variable Y_i equals a particular value y_i conditional on the parameters. The residual error term of observation $i = 1$ tells us nothing about the error term for $i = 2$, and conditional on a particular β and σ^2, y_1 tells us nothing about y_2. If we assume that our observations are conditionally independent (conditioning on our parameter vector $\theta = (\beta, \sigma^2)$), then we can simply multiply all of the point-wise likelihoods together to find the joint probability of our sample $\boldsymbol{y}$ conditional on the parameters (the likelihood of our sample):

$$p(y_1, y_2, ..., y_n|\theta) = p(y_1|\theta)p(y_2|\theta)...p(y_n|\theta)$$

$$p(\boldsymbol{y}|\theta) = \prod_{i=1}^{n} p(y_i|\theta)$$

$$\mathcal{L}(\theta|\boldsymbol{y}) = \prod_{i=1}^{n} p(y_i|\theta)$$

If the observations $y_1, ..., y_n$ are not conditionally independent (or if you like, if the error terms are not independent), then a likelihood function that multiplies the point-wise probabilities together as if they are independent events is no longer valid. This is the problem underlying many discussions of non-independence, psuedoreplication, and autocorrelation (spatial, temporal, phylogenetic): all of these lead to violations of this independence assumption, meaning that it is not correct to work with the product of all the point-wise likelihoods unless terms are added to the model (e.g., blocking factors, autoregressive terms, spatial random effects) so that the observations are conditionally indepenent.

Obtaining maximum likelihood estimates

We have already obtained quite a few maximum likelihood estimates (MLEs) in the previous chapter with the `lm()` function. Here, we provide a more general treatment of estimation.

Assuming that we have a valid likelihood function $\mathcal{L}(\theta|\boldsymbol{y})$, we often seek to find the parameter values that maximize the probability of having observed our sample $\boldsymbol{y}$. We can proceed in a few different ways, analytically, by direct search, and by optimization for example. Usually the likelihood function is computationally and analytically more tractable on a log scale, so that we often end up working with the log-likelihood rather than the likelihood directly. This is fine, because any parameter(s) θ that maximize the likelihood will also maximize the log-likelihood and vice versa, because the log function is strictly increasing. Mathematically, we might refer to a maximum likelihood estimate as the value of θ that maximizes $p(\boldsymbol{y}|\theta)$. Recalling some calculus, it is reasonable to think that we might attempt to differentiate $p(\boldsymbol{y}|\theta)$ with respect to θ, and find the points at which the derivative equal zero to identify candidate maxima. The first derivative will be zero at a maximum, but also at any minima or inflection points, so in practice first-order differentiation alone is not sufficient to identify MLEs. In this class, we won't worry about analytically deriving MLEs, but for those who are interested and have some multivariate calculus chops, see Casella and Berger's 2002 book *Statistical Inference*.

So, we've established that the likelihood is: $p(y|\theta) = \prod_{i=1}^{n} p(y_i|\theta)$. Computationally, this is challenging because we are working with really small numbers (products of small numbers) - so small that our computers have a hard time keeping track of them with much precision. Summing logs of small numbers is more computationally stable than multiplying many small numbers together. So, let's instead work with the log likelihood by taking the log of both sides of the previous equation.

$$log(p(y|\theta)) = log\big(\prod_{i=1}^{n} p(y_i|\theta)\big)$$

Because $log(ab) = log(a) + log(b)$, we can sum up the log likelihoods on the right side of the equation:

$$log(p(y|\theta)) = \sum_{i=1}^{n} log(p(y_i|\theta))$$

Direct search

Here we'll illustrate two methods to find MLEs for normal models: direct search and optimization. Returning to our simplest normal model (the model of the mean), we have two parameters: $\theta = (\mu, \sigma^2)$ and $y \sim N(\mu, \sigma^2)$. As an aside, maximizing the likelihood is equivalent to minimizing the sum of squared error with the normal distribution. Below, we simulate a small dataset with known parameters, and then use a direct search over a bivariate grid of parameters (μ and σ).

```
# set parameters
mu <- 6
sigma <- 3

# simulate observations
n <- 200
y <- rnorm(n, mu, sigma)

# generate a grid of parameter values to search over
g <- expand.grid(mu = seq(4, 8, length.out=100),
                 sigma=seq(2, 7, length.out=100))

# evaluate the log-likelihood of the data for each parameter combination
g$loglik <- rep(NA, nrow(g))
for (i in 1:nrow(g)){
  g$loglik[i] <- sum(dnorm(y, g$mu[i], g$sigma[i], log = TRUE))
}

# plot results
library(ggplot2)
ggplot(g, aes(x = mu, y = sigma)) +
  geom_tile(aes(fill = loglik)) +
  stat_contour(aes(z = loglik), bins=40) +
  scale_fill_gradient(low="white", high="red")
```

This is a contour plot of the log-likelihood surface. The black lines are log-likelihood isoclines, corresponding to particular values of the log-likelihood. If we are lucky, there is only one global maximum on the surface (this can be assessed analytically), and we've found it. Here our best estimate for our parameters $\theta = (\mu, \sigma^2)$ will be the pair of paramters that has the greatest log-likelihood:

```
# find the approximate MLE
MLE_dsearch <- g[which.max(g$loglik), ]
MLE_dsearch
```

```
##            mu    sigma    loglik
## 1750 5.979798 2.858586 -493.7844
```

Optimization

Finding maxima and minima of functions is a common operation, and there are many algorithms that have been developed to accomplish these tasks. Some of these algorithms are included in the base R function `optim()`.

Optimization routines have an easier time optimizing in unconstrained space, where parameters can be anywhere between $-\infty$ and ∞. However, we are trying to optimize a parameter that must be positive, σ. We

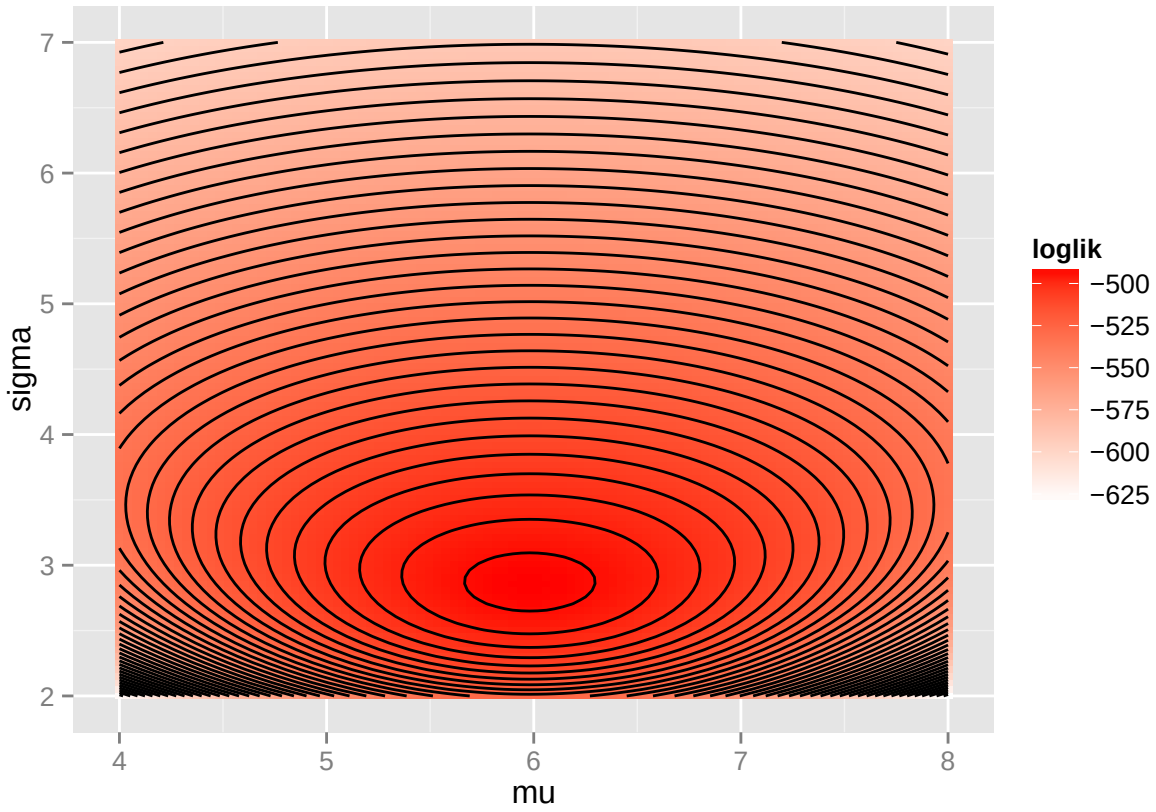


Figure 17: Contour plot for the log-likelihood surface across a 2d grid of parameter space.

can transform σ so that we can optimize over unconstrained space: `log` maps sigma from its constrained space $(0, \infty)$ to unconstrained space $(-\infty, \infty)$, and the `exp` function maps from the unconstrained space back to the constrained space (and the scale of the parameter). This trick shows up later in the context of link-functions for generalized linear models, where we transform a constrained linear predictor to unconstrained space while estimating parameters.

We need to provide some initial values for the parameters and a function to minimize. If we find the minimum of the negative log-likelihood, then we have found the MLE.

```
# writing a negative log-likelihood function
nll <- function(theta, y){
  mu <- theta[1]
  sigma <- exp(theta[2])
  # theta[2] is the log of sigma, which is unconstrained
  # this simplifies optimization because we have no boundaries
  -sum(dnorm(y, mu, sigma, log=TRUE))
}

# initial guesses
theta_init <- c(mu = 4, log_sigma = 1)

# optimize
res <- optim(theta_init, nll, y=y)
res
```

```
## $par
```

```
##        mu log_sigma
##  5.980766  1.050048
##
## $value
## [1] 493.7844
##
## $counts
## function gradient
##       89       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

If the algorithm has converged (check to see if `res$convergence` is zero), and if there is only one minimum in the negative log-likelihood surface (we know this is true based on analytical results in this case), then we have identified the MLEs of μ and $ln(\sigma)$. How do these estimates compare to our first estimates found via direct search?

```
MLE_optim <- c(res$par[1], exp(res$par[2]))
rbind(unlist(MLE_dsearch[c('mu', 'sigma')]),
      unlist(MLE_optim))
```

```
##            mu    sigma
## [1,] 5.979798 2.858586
## [2,] 5.980766 2.857788
```

This approach is quite general, and can be modified to be used for instance in a linear regression context:

```
n <- 20
x <- runif(n)
y <- rnorm(n, 3 + 10 * x, sd = 1)

nll <- function(theta, y, x){
  alpha <- theta[1]
  beta <- theta[2]
  sigma <- exp(theta[3])
  mu <- alpha + beta * x
  -sum(dnorm(y, mu, sigma, log=TRUE))
}

# initial guesses
theta_init <- c(alpha = 4, beta = 1, log_sigma = 1)

# optimize
res <- optim(theta_init, nll, y = y, x = x)
```

Next we can plot the line of best fit that minimized the negative log likelihood of our data.

```r
plot(x, y)
abline(a = res$par['alpha'], b = res$par['beta'])
```

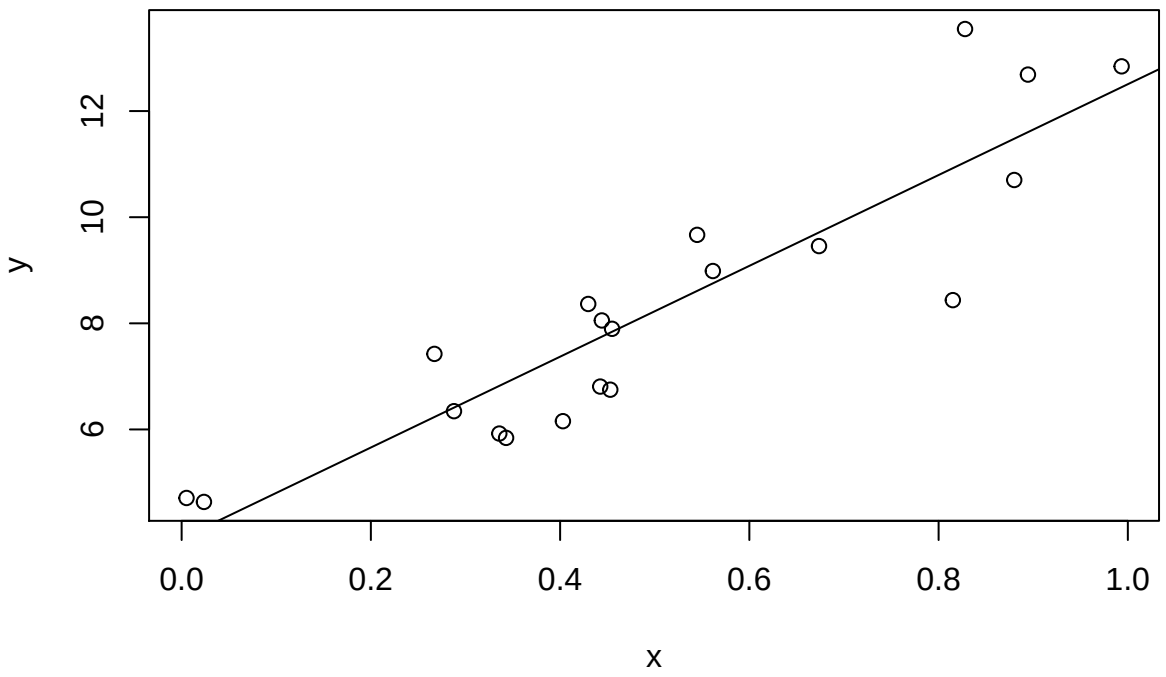


Figure 18: Raw data and line of best fit from a linear regression that used `optim` to find the maximum likelihood estimates.

We may wish to evaluate the output from `optim` so that we can see the maximum likelihood estimates:

```r
res
```

```
## $par
##      alpha       beta  log_sigma
## 3.94813935 8.55814295 0.08963467
##
## $value
## [1] 30.17068
##
## $counts
## function gradient
##      196       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

We should note that `optim` can fail in many ways. For instance, it may fail to find a minimum value. If `res$convergence` is not equal to zero, then the algorithm has not converged at all. See the help file for optim for details (`?optim`), but there are specific convergence codes to indicate that the maximum number of iterations in the algorithm has been reached, that the optimization routine has become "degenerate" in

some way, and so on. Worse, there may be multiple minima and it may converge to a local minimum, but not *the* global minimum. Unfortunately there are no warning or error messages for this. It is a good idea to experiment with different starting values for the parameters so that you can more reliably find the global minimum (if it exists). This concept will show up again later in the context of Markov chain Monte Carlo methods, where we are trying to characterize a probability surface.

Analytic, direct search, and optimization approaches for maximum likelihood estimation can all be useful, but in this class we will rarely make use of direct search and optimization. However, the likelihood function and maximum likelihood estimation will continue to play a central role, as it is involved in the estimation of parameters in Bayesian inference. In a Bayesian context, the likelihood provides the key link between observed data and unknown parameters, but we provide additional probabilistic structure for the parameters in the form of prior distributions for the unknown parameters. The key philosophical difference however remains in the consideration of the data vs. the parameters as fixed vs. unknown quantities.

Further reading

Casella and Berger. 2002. *Statistical Inference*, Chapter 7.

Scholz FW. 2004. Maximum likelihood estimation, in *Encyclopedia of Statistical Sciences*.

Gelman and Hill. 2009. *Data analysis using regression and multilevel/hierarchical models*. Chapter 18.

Chapter 3: Bayesian inference

Big picture

In recent decades Bayesian inference has increasingly been used in ecology. A key difference between Bayesian and maximum likelihood approaches lies in which quantities are treated as random variables. For any likelihood function, the parameters θ are not random variables because there are no probability distributions associated with θ. In contrast, Bayesian approaches treat parameters and future data (actually all unobserved quantities) as random variables, meaning that each unknown is associated with a probability distribution $p(\theta)$. In both Bayesian and maximum likelihood approaches, the observations y are treated as a random variable, and the probability distribution for this random variable $p(y \mid \theta)$ plays a central role in both approaches. See Hobbs and Hooten (Ch. 5) for a more detailed treatment on differences between Bayesian and maximum likelihood based inferential approaches. Some authors also point out differences between Bayesian and frequentist definitions of probability. In a frequentist framework, probabilities are defined in term of long run frequencies of events, often relying on hypothetical realizations. Bayesian probability definitions do not rely on the long-run frequency of events.

Philosophy aside, Bayesian approaches have become more popular because of intuitive appeal and practical advantages. Intuitively, it can seem backwards to focus on the probability of the data given the parameters $p(y \mid \theta)$. What we really want to know is the probability of the parameters, having observed some data $p(\theta \mid y)$. As we will see, Bayes' theorem allows us to calculate this probability. Second, Bayesian approaches are often easier to implement than maximum likelihood or frequentist approaches, particularly for complex models. Finally, we find that in many applications, Bayesian approaches facilitate a better understanding of model structure and assumptions.

Learning goals

- Bayes' theorem and Bayesian probability
- relationship between likelihood and Bayesian inference
- priors (generally, informative vs. non-informative)
- proper vs. improper priors
- intro to Bayesian computation and MCMC
- posterior summaries and comparisons
- single parameter models: MLE vs. Bayesian treatments
- Bayesian linear regression: intro to Stan

Bayes' theorem

Bayes' theorem is an incredibly powerful theorem that follows from the rules of probability. To prove the theorem, we need only a few ingredients: 1) the definition of joint probabilities $p(A, B) = p(A \mid B)p(B)$ or $p(A, B) = p(B \mid A)p(A)$ (both are valid) and 2) a bit of algebra.

$$p(A, B) = p(A \mid B)p(B)$$

$$p(B \mid A)p(A) = p(A \mid B)p(B)$$

$$p(B \mid A) = \frac{p(A \mid B)p(B)}{p(A)}$$

This is Bayes' theorem. In modern applications, we typically substitute unknown parameters θ for B, and data y for A:

$$p(\theta \mid y) = \frac{p(y \mid \theta)p(\theta)}{p(y)}$$

The terms can verbally be described as follows:

- $p(\theta \mid y)$: the *posterior* distribution of the parameters. This tells us what the parameters probably are (and are not), conditioned on having observed some data y.
- $p(y \mid \theta)$: the likelihood of the data y.
- $p(\theta)$: the *prior* distribution of the parameters. This should represent our prior knowledge about the values of the parameters. Prior knowledge comes from similar studies and/or first principles.
- $p(y)$: the marginal distribution of the data. This quantity can be difficult or even impossible to compute, will always be a constant after the data have been observed, and is often ignored.

Because $p(y)$ is a constant, it is valid and common to consider the posterior distribution up to this proportionality constant:

$$p(\theta \mid y) \propto p(y \mid \theta)p(\theta)$$

Prior distributions

We have already learned about likelihood, but the introduction of a prior distribution for the parameters requires some attention. The inclusion of prior information is not unique to Bayesian inference. When selecting study systems, designing experiments, cleaning or subsetting data, and choosing a likelihood function, we inevitably draw upon our previous knowledge of a system.

From a Bayesian perspective, prior distributions $p(\theta)$ represent our knowledge/beliefs about parameters before having observed the data y, and the posterior distribution represents our updated knowledge/beliefs about parameters after having observed our data. This is similar to the way many scientists operate: we think we know something about a system, and then we do experiments and conduct surveys to update our knowledge about the system. But, the observations generated from the experiments and surveys are not considered in isolation. They are considered in the context of our previous knowledge. In this way, the posterior distribution represents a compromise between our prior beliefs and the likelihood.

Analytical posterior with conjugate priors: Bernoulli case*

* *this section is a bit math-heavy for illustration, but most of the time we won't find the posterior analytically*

The Bernoulli distribution is a probability distribution for binary random variables (e.g., those that take one of two values: dead or alive, male or female, heads or tails, 0 or 1, success or failure, and so on). The Bernoulli distribution has one parameter, p : the probability of "success" (or more generally, the probability of one of the two outcomes) in one particular event. A Bernoulli random variable takes one of these two values. The choice of which of the two possible outcomes is considered "success" is often arbitrary - we could consider either "heads" or "tails" to be a success if we wanted to. If p is the probability of success in one particular trial, then the probability of failure is just the complement of $p : 1 - p$, sometimes referred to as q, such that $p + q = 1$. For those familiar with the Binomial distribution, the Bernoulli distribution is a special case of the Binomial, with one trial $k = 1$. We can use the Bernoulli distribution as a likelihood function, where y is either zero or one: $y \in \{0, 1\}$, and p is our only parameter. Because p is a probability, we know that $0 \leq p \leq 1$.

We can express the likelihood for a Bernoulli random variable as follows.

$$p(y \mid p) = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{if } y = 0 \end{cases}$$

Equivalently and more generally:

$$[y \mid p] = p^y (1-p)^{1-y}$$

If we have n independent Bernoulli random variables, $y_1, ..., y_n$, each with probability p, then the joint likelihood can be written as the product of the point-wise likelihoods:

$$[y_1, ..., y_n \mid p] = p^{y_1}(1-p)^{1-y_1} ... p^{y_n}(1-p)^{1-y_n}$$

$$[\boldsymbol{y} \mid p] = \prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_1}$$

Recalling from algebra that $x^a x^b = x^{a+b}$, this implies:

$$[\boldsymbol{y} \mid p] = p^{\sum_i y_i}(1-p)^{n-\sum_i y_i}$$

Having obtained the likelihood, we now must specify a prior to complete our specification of the joint distribution of data and parameters $[y \mid p][p]$, the numerator in Bayes' theorem. A natural choice is the Beta distribution, which has two parameters α and β, with support on the interval $(0, 1)$. This is a good choice because its bounds are similar to those for probabilities, and the posterior induced by the prior is also a Beta distribution. When a prior distribution for a parameter induces a posterior distribution that is of the same form (same probability distribution) as the prior, the prior is said to be a "conjugate prior" for the likelihood. The density of the beta distribution for parameter p has two parameters α and β and is defined as:

$$[p] = cp^{\alpha-1}(1-p)^{\beta-1}$$

Where c is a constant that ensures that $[p \mid \alpha, \beta]$ integrates to one over the interval $(0, 1)$ (i.e., it is a true probability distribution), with $c = \dfrac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}$, and $\Gamma(x) = (x-1)!$ That's a factorial symbol (!), not a punctuation mark! To give a bit of intuition for what the beta distribution looks like, here are some plots of the beta density with different values of α and β:

```
alpha <- rep(c(1, 5, 10))
beta <- rep(c(1, 5, 10))
g <- expand.grid(alpha=alpha, beta=beta)
x <- seq(0, 1, .005)

par(mfrow=c(3, 3))
for (i in 1:nrow(g)){
  plot(x, dbeta(x, g$alpha[i], g$beta[i]),
       type='l', ylim=c(0, 10),
       ylab="[x]", lwd=3)
  title(bquote(alpha == .(g$alpha[i]) ~ ", " ~ beta == .(g$beta[i])))
}
```

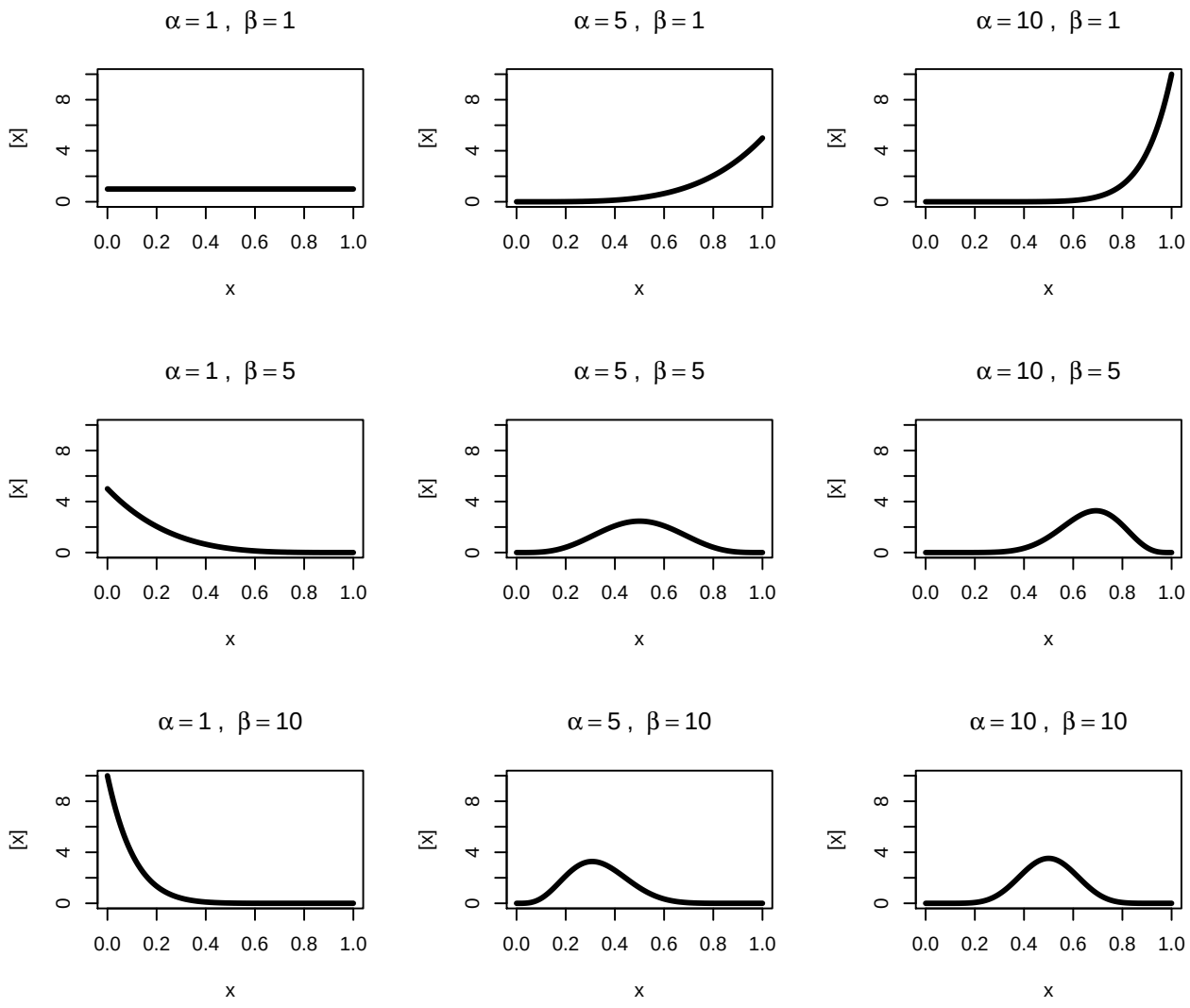


Figure 19: A collection of beta priors with varying parameters.

One commonly used prior is the beta(1, 1), because it corresponds to a uniform prior for p (shown in the top left corner). Now we have all of the ingredients to embark on our first Bayesian analysis for a Bernoulli random variable. We'll proceed by finding the posterior distribution up to some proportionality constant, then we'll use our knowledge of the beta distribution to recover the correct proportionality constant:

$$[p|y] = \frac{[y \mid p][p]}{[y]}$$

$$[p|y] \propto [y \mid p][p]$$

Plugging in the likelihood and prior that we described above:

$$[p|y] \propto p^{\sum_i y_i}(1-p)^{n-\sum_i y_i}[p]$$

$$[p|y] \propto p^{\sum_i y_i}(1-p)^{n-\sum_i y_i} c p^{\alpha-1}(1-p)^{\beta-1}$$

Dropping c, because we're only working up to some proportionality constant:

$$[p|y] \propto p^{\sum_i y_i}(1-p)^{n-\sum_i y_i} p^{\alpha-1}(1-p)^{\beta-1}$$

Then, again recalling that $x^a x^b = x^{a+b}$, we find that

$$[p|y] \propto p^{\alpha-1+\sum_i y_i}(1-p)^{\beta-1+n-\sum_i y_i}$$

Notice that this is of the same form as the beta prior for p, with updated parameters: $\alpha_{post} = \alpha + \sum_i y_i$ and $\beta_{post} = \beta + n - \sum_i y_i$. In this sense, the parameters of the beta prior α and β can be interpreted as the previous number of successes and failures, respectively. Future studies can simply use the updated values α_{post} and β_{post} as priors. We have found a quantity that is proportional to the posterior distribution, which often is enough, but here we can easily derive the proportionality constant that will ensure that the posterior integrates to one (i.e., it is a true probability distribution).

Recall the proportionality constant c from the beta distribution prior that we used, which is $c = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}$. Updating this proportionality constant gives us the correct value for our posterior distribution, ensuring that it integrates to one, so that we now can write down the posterior distribution in closed form:

$$[p|y] = \frac{\Gamma(\alpha_{post}+\beta_{post})}{\Gamma(\alpha_{post})\Gamma(\beta_{post})} p^{\alpha_{post}-1}(1-p)^{\beta_{post}-1}$$

Now we can explore the effect of our prior distributions on the posterior. Suppose that we have observed $n = 8$ data points, $y_1, y_2, ..., y_{10}$, with $y_i \in \{0, 1\}$ and 2 successes, $\sum_i y_i = 2$. Let's graph the posterior distributions that are implied by the priors plotted above and the likelihood resulting from these observations.

```
g$alpha_post <- g$alpha + 2
g$beta_post <- g$beta + 6

par(mfrow=c(3, 3))
for (i in 1:nrow(g)){
  plot(x, dbeta(x, g$alpha[i], g$beta[i]),
       type='l', ylim=c(0, 10),
       xlab="p", ylab="[p | y]", lwd=3, col='grey')
```

```
  lines(x, dbeta(x, g$alpha_post[i], g$beta_post[i]),
        lwd=3, col='blue')
  lines(x, 8 * dbinom(x=2, size=8, prob=x), col='red')
  title(bquote(alpha == .(g$alpha[i]) ~ ", " ~ beta == .(g$beta[i])))
}
```

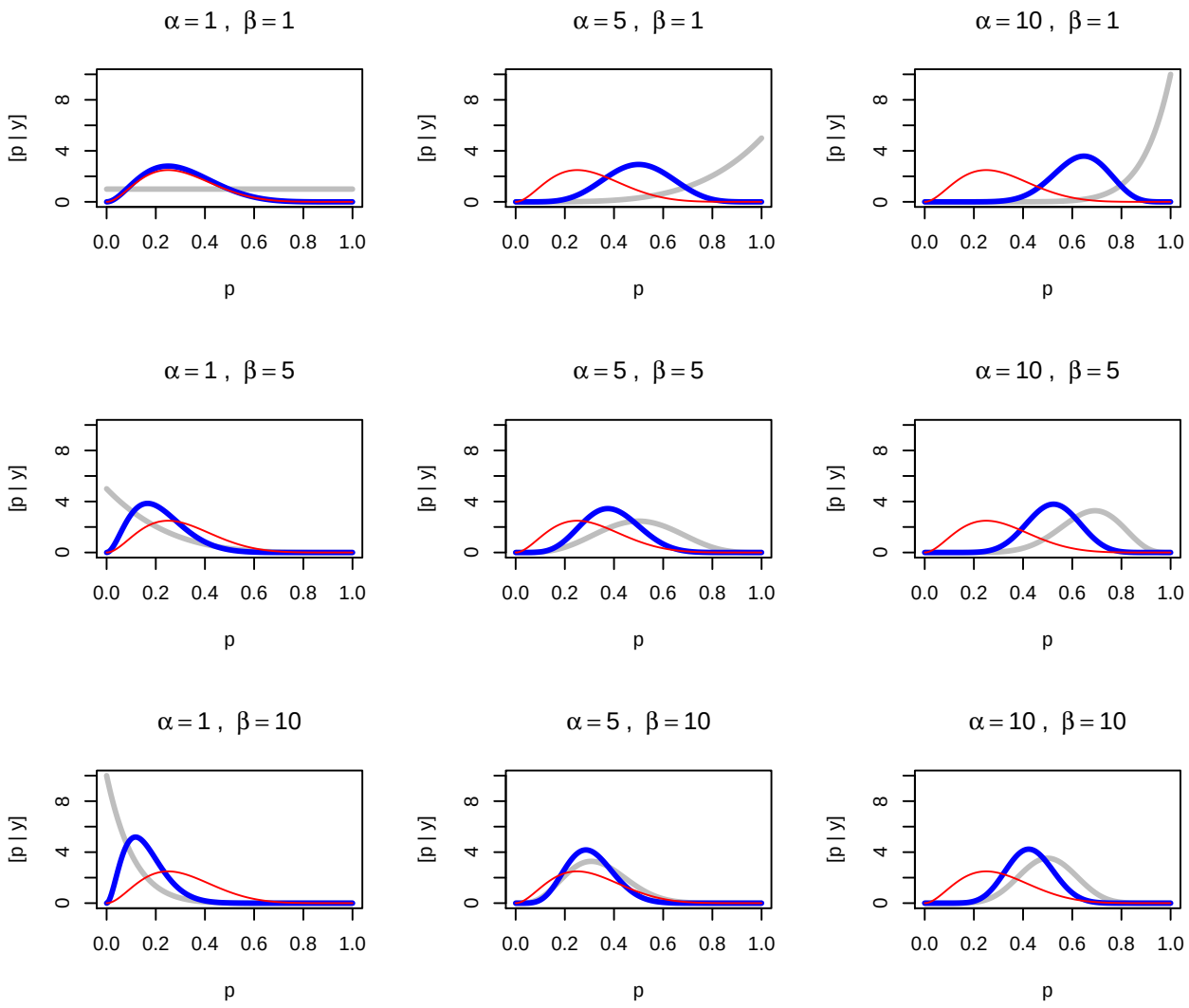


Figure 20: The same prior distributions as before, but now with the liklihood and posterior distributions. The prior is shown in grey, the likelihood is shown in red, and the posterior distribution is shown in blue.

Notice how strong priors pull the posterior distribution toward them, and weak priors result in posteriors that are mostly affected by the data. Some Bayesian statisticians nowadays advocate for the inclusion of reasonable prior distributions rather than prior distributions that feign ignorance. One nice feature of Bayesian approaches is that, given enough data, the prior tends to have less of an influence on the posterior. This is consistent with the notion that when reasonable people are presented with strong evidence, they tend to more or less agree even if they may have disagreed ahead of time (though at times the empirical evidence for this phenomenon may seem somewhat equivocal). To show this, let's increase the amount of information in our data, so that we have $n = 800$ and $\sum y = 200$ successes.

```r
g$alpha_post <- g$alpha + 200
g$beta_post <- g$beta + 600

par(mfrow=c(3, 3))
for (i in 1:nrow(g)){
  plot(x, dbeta(x, g$alpha[i], g$beta[i]),
       type='l', ylim=c(0, 32),
       xlab="p", ylab="[p | y]", col='grey', lwd=2)
  lines(x, dbeta(x, g$alpha_post[i], g$beta_post[i]),
        col='blue', lwd=2)
    lines(x, 800 * dbinom(x=200, size=800, prob=x), col='red')
  title(bquote(alpha == .(g$alpha[i]) ~ ", " ~ beta == .(g$beta[i])))
}
```

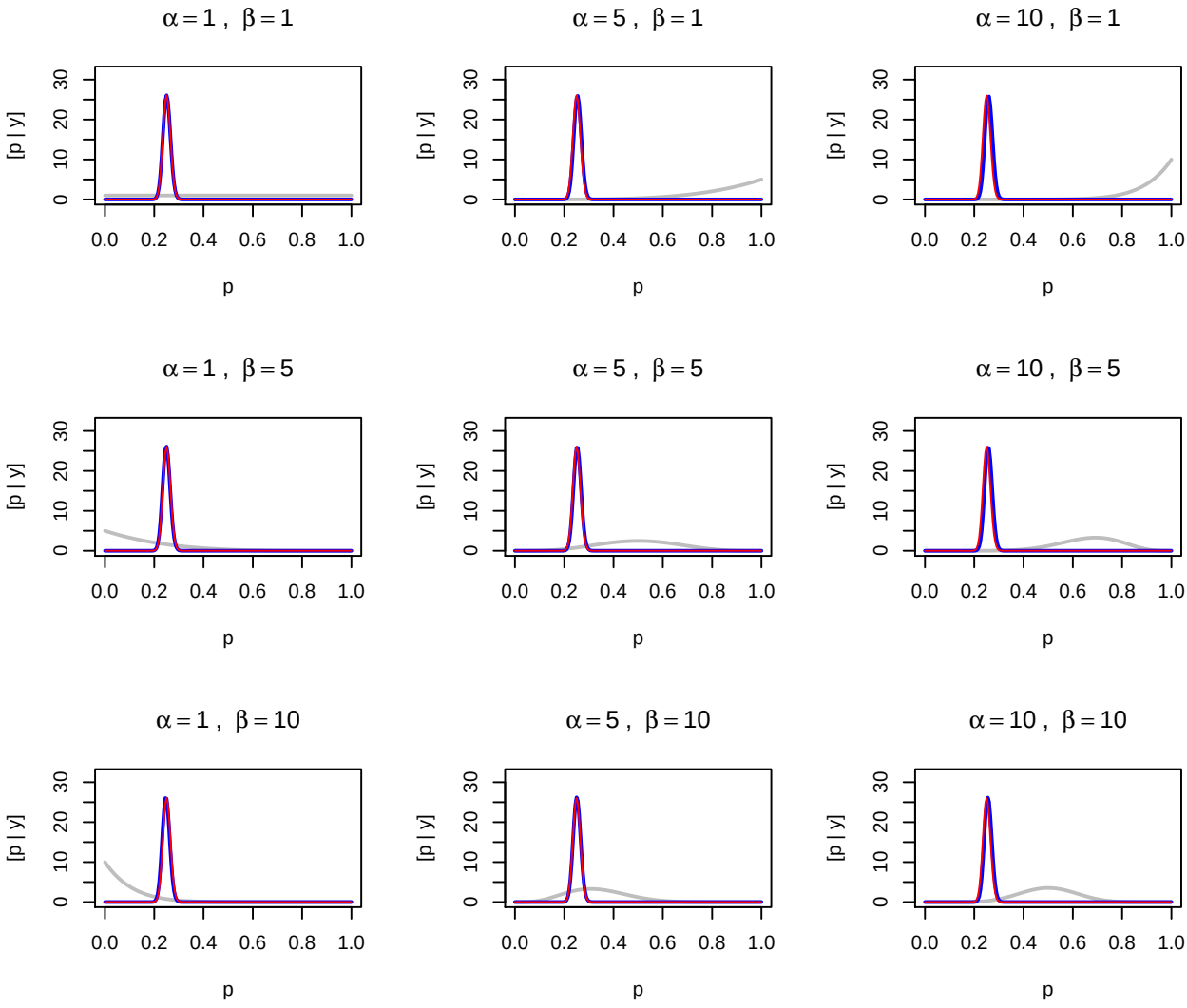


Figure 21: Updated example with same parameters but larger sample sizes. Here again the prior is shown in grey, the likelihood is shown in red, and the posterior distribution is shown in blue. The posterior distribution has essentially the same shape as the likelihood because we have much more information in this larger data set relative to our priors.

Improper priors

Improper priors do not integrate to one (they do not define proper probability distributions). For instance, a normal distribution with infinite variance will not integrate to one. Sometimes improper priors can still lead to proper posteriors, but this must be checked analytically. Unless you're willing to prove that an improper prior leads to a proper posterior distribution, we recommend using proper priors.

Posterior computation the easy way

In reality, most of the time we don't analytically derive posterior distributions. Mostly, we can express our models in specialized programming languages that are designed to make Bayesian inference easier. Here is a Stan model statement from the above example.

```
data {
  int n;
  int<lower=0, upper=1> y[n];
}

parameters {
  real<lower=0, upper=1> p;
}

model {
  p ~ beta(1, 1);
  y ~ bernoulli(p);
}
```

The above model statement has three "blocks". The data block specifies that we have two fixed inputs: the sample size n and a vector of integer values with n elements, and these have to be either zero or one. The parameter block specifies that we have one parameter p, which is a real number between 0 and 1. Last, the model block contains our beta prior for p and our Bernoulli likelihood for the data y.

Stan does not find the analytic expression for the posterior. Rather, Stan translates the above program into a Markov chain Monte Carlo (MCMC) algorithm to simulate samples from the posterior distribution. In practice, this is sufficient to learn about the model parameters.

What is MCMC?

MCMC is used to sample from probability distributions generally, and from posterior probability distributions in the context of Bayesian inference. This is a huge topic, and involves a fair bit of theory that we will not dwell on here, but it is worth reading Chapter 18 in Gelman and Hill for a taste of some of the algorithms that are used. In this class, we will rely on specialized software to conduct MCMC simulations, so that many of the details are left "under the hood". However, it is still important to know what MCMC is (supposed to be) doing, and how to identify when MCMC algorithms fail.

In a Bayesian context, we often run multiple Markov chain simulations, where we iteratively update our parameter vector θ. If all goes well, then eventually each Markov chain converges to the posterior distribution of the parameters, so that every draw can be considered a simulated sample from the posterior distribution. Typically, we initialize the chains at different (often random) points in parameter space. If all goes well, then after some number of iterations, every chain has converged to the posterior distribution, and we run the chains a bit longer to generate a representative sample from the posterior, then perform inference on our sample. Chains start dispersed in parameter space and eventually all converge to the same region and stay there. After some large number of iterations, the estimated posterior density stabilizes. At this point, usually

the early draws from the Markov chains are discarded as "warmup" or "burnin", as these do not represent draws from the posterior, because the chains had not converged.

It is always necessary to run diagnostics on MCMC output to ensure convergence. Some of these are graphical. Traceplots show the parameter values that a Markov chain has taken on the y-axis, and iteration number on the x-axis. For instance, if we run our Stan model from before:

```
library(rstan)
m <- '
data {
  int n;
  int<lower=0, upper=1> y[n];
}

parameters {
  real<lower=0, upper=1> p;
}

model {
  p ~ beta(1, 1);
  y ~ bernoulli(p);
}
'

y <- c(1, 1, 0, 0, 0, 0, 0, 0)
n <- length(y)

out <- stan(model_code=m)
```

```
##
## SAMPLING FOR MODEL '8dbabac1e0f3c5dfe4a2c8e37a6f8ed1' NOW (CHAIN 1).
##
## Chain 1, Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1, Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1, Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1, Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1, Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1, Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1, Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1, Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1, Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1, Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1, Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1, Iteration: 2000 / 2000 [100%]  (Sampling)
## #  Elapsed Time: 0.005473 seconds (Warm-up)
## #                0.005552 seconds (Sampling)
## #                0.011025 seconds (Total)
##
##
## SAMPLING FOR MODEL '8dbabac1e0f3c5dfe4a2c8e37a6f8ed1' NOW (CHAIN 2).
##
## Chain 2, Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2, Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2, Iteration:  400 / 2000 [ 20%]  (Warmup)
```

```
## Chain 2, Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2, Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2, Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2, Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2, Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2, Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2, Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2, Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2, Iteration: 2000 / 2000 [100%]  (Sampling)
## #  Elapsed Time: 0.005593 seconds (Warm-up)
## #                0.005203 seconds (Sampling)
## #                0.010796 seconds (Total)
##
##
## SAMPLING FOR MODEL '8dbabac1e0f3c5dfe4a2c8e37a6f8ed1' NOW (CHAIN 3).
##
## Chain 3, Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3, Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3, Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3, Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3, Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3, Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3, Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3, Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3, Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3, Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3, Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3, Iteration: 2000 / 2000 [100%]  (Sampling)
## #  Elapsed Time: 0.005595 seconds (Warm-up)
## #                0.005066 seconds (Sampling)
## #                0.010661 seconds (Total)
##
##
## SAMPLING FOR MODEL '8dbabac1e0f3c5dfe4a2c8e37a6f8ed1' NOW (CHAIN 4).
##
## Chain 4, Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4, Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4, Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4, Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4, Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4, Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4, Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4, Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4, Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4, Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4, Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4, Iteration: 2000 / 2000 [100%]  (Sampling)
## #  Elapsed Time: 0.005664 seconds (Warm-up)
## #                0.006362 seconds (Sampling)
## #                0.012026 seconds (Total)
```

```
traceplot(out, inc_warmup=TRUE)
```

This traceplot is useful for verifying convergence: all of the chains appear to be sampling from the same

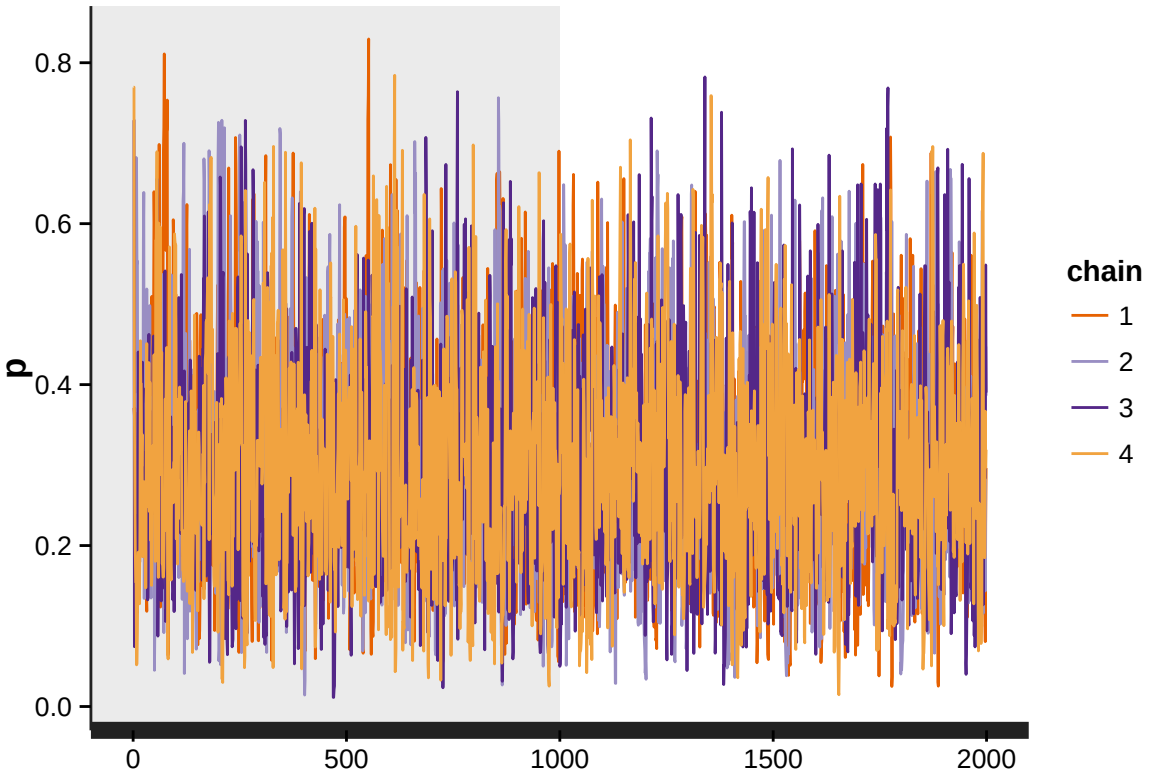


Figure 22: Traceplots that show the sequence of parameter values taken in the Markov chains.

region. We can also inspect some numerical summaries that are used to detect non-convergence. Specifically, we can look at the $\hat{R}$ statistic. If $\hat{R} > 1.1$, then we ought to be worried about convergence of our chains. In addition, plotting autocorrelation function plots can help to diagnose autocorrelation and inefficient MCMC algorithms, but won't necessarily detect non-convergence. This is important because we need a decent number of independent samples to make reliable inference. Printing our model output returns this statistic as well as some other summary statistics for the posterior draws.

```
out
```

```
## Inference for Stan model: 8dbabac1e0f3c5dfe4a2c8e37a6f8ed1.
## 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     0.30    0.00 0.14  0.08  0.19  0.29  0.38  0.60  1490    1
## lp__ -6.62    0.02 0.71 -8.64 -6.79 -6.34 -6.16 -6.11  1393    1
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:42:26 2015.
## 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).
```

This output also tells us that we ran four MCMC chains, each for 2000 iterations, and the first 1000 iterations were discarded as warmup (the shaded region in the traceplot). For each parameter (and for the log probability up to a proportionality constant `lp__`), we get the posterior mean, the MCMC standard error of the mean, the posterior standard deviation, some quantiles, and an estimate for the number of effective samples from the posterior `n_eff`, which should typically be at least in the hundreds.

Example: normal linear models

Recall that all normal linear models can be expressed as:

$$y \sim N(X\beta, \sigma^2)$$

To complete a Bayesian analysis, we need to select prior distributions for the unknown parameters β and σ^2. For instance:

$$\beta \sim N(0, 5)$$

$$\sigma \sim halfNormal(0, 5)$$

,

where the half-Normal with mean zero is a folded Gaussian probability density function with only positive mass:

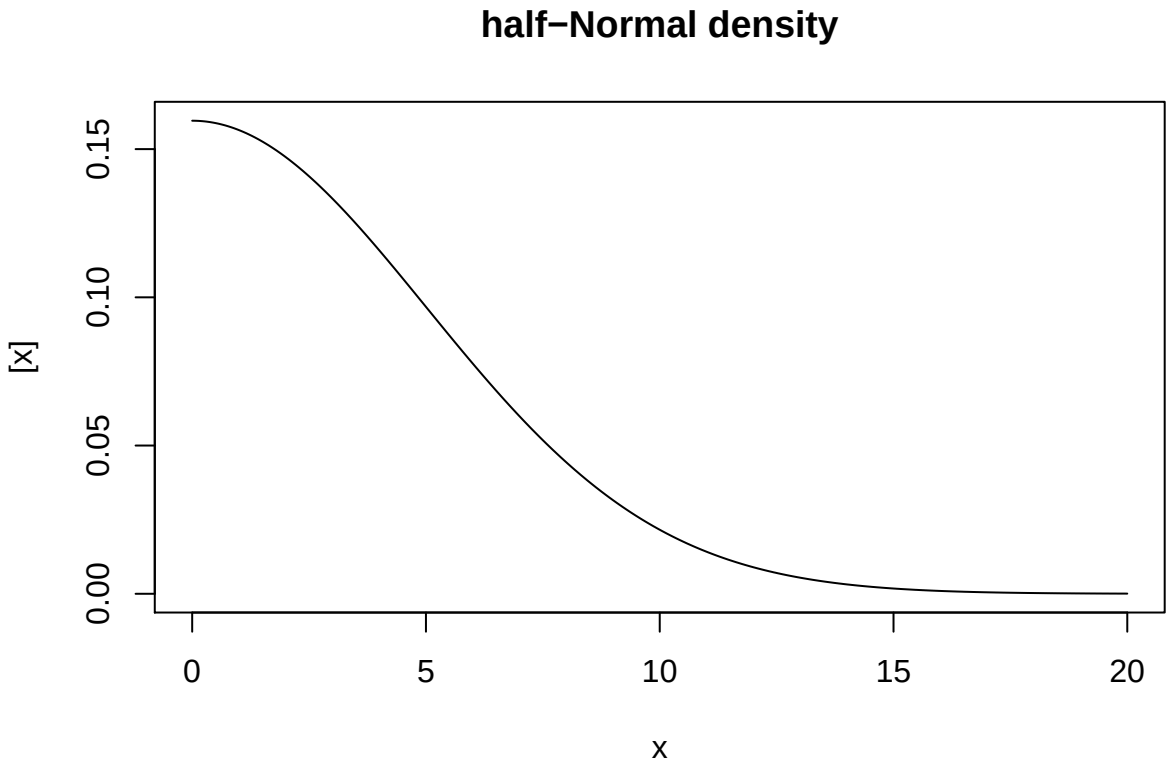


Figure 23: Half-normal prior distribution with mean 0 and standard deviation 5.

Let's do a quick linear regression model and summarize/plot the results.

```
n <- 20
x <- runif(n, 0, 3)
y <- rnorm(n, -3 + .75 * x, 1)
plot(x, y)
```

We'll use Stan again, but this time instead of specifying an object (`m` above) that is a character string containing our model statement, we'll save the model file somewhere else with a `.stan` file extension. For instance, maybe we have a general purpose Stan model that can be used for linear models called `lm.stan`:

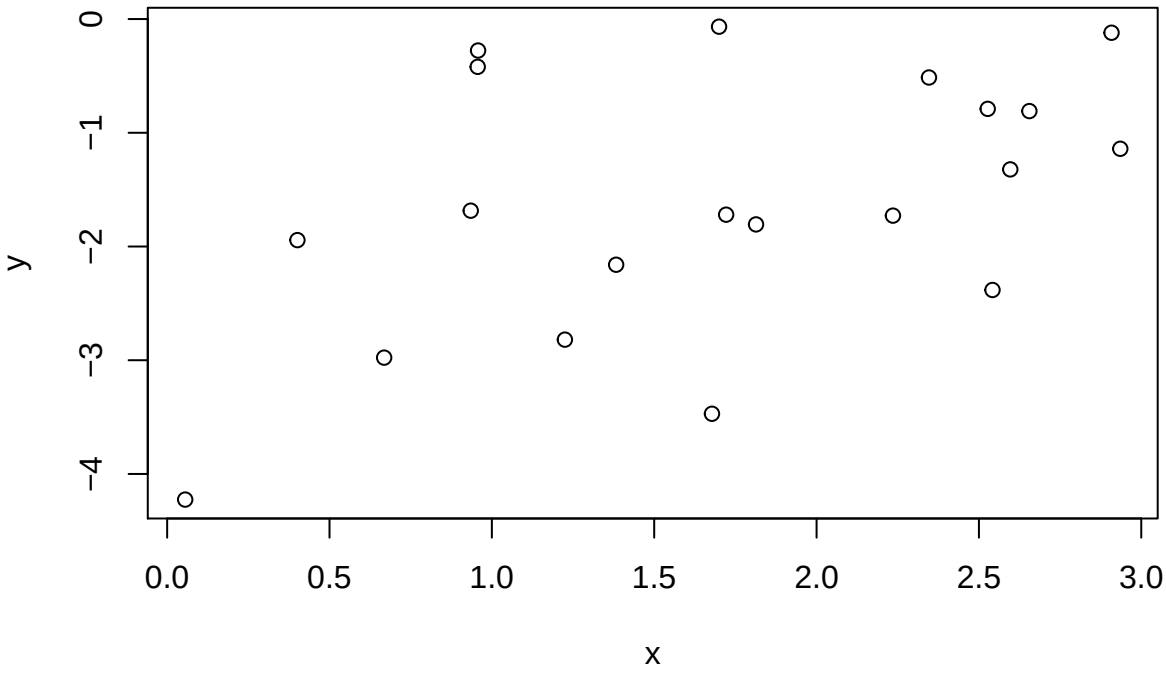


Figure 24: Plot of simulated linear regression data.

```
data {
  int n; // sample size
  int p; // number of coefficients
  matrix[n, p] X;
  vector[n] y;
}

parameters {
  vector[p] beta;
  real<lower=0> sigma;
}

model {
  beta ~ normal(0, 5);
  sigma ~ normal(0, 5);
  y ~ normal(X * beta, sigma);
}
```

So we have this file saved somewhere as `lm.stan`, and it can fit any of the linear models that we covered in Chapter 1 by changing the design matrix, but now we can include information to improve our estimates. Because this is a simulated example, we'll use somewhat vague priors. Fitting the model:

```
library(rstan)
X <- matrix(c(rep(1, n), x), ncol = 2)
stan_d <- list(n = nrow(X), p = ncol(X), X = X, y = y)
out <- stan('lm.stan', data = stan_d)
```

There are also some other default plots which are nice:

```
traceplot(out)
```

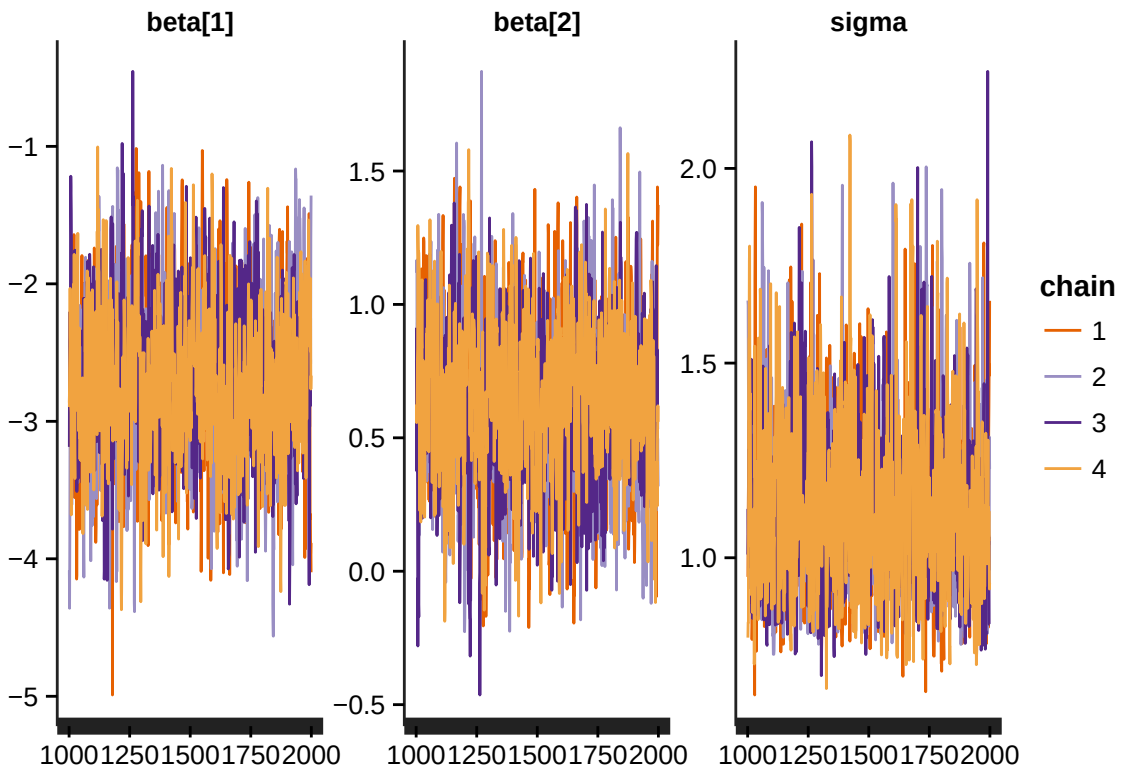


Figure 25: Traceplot of Markov chains.

```
plot(out)
```

```
## ci_level: 0.8 (80% intervals)
## outer_level: 0.95 (95% intervals)
```

```
pairs(out)
```

Notice that the slopes and intercepts are correlated in the posterior (do you recall why?). Also, `lp__` is tracked automatically, and this is proportional to the log probability of the posterior distribution.

Let's inspect the output in table form:

```
out
```

```
## Inference for Stan model: lm.
## 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
## beta[1]  -2.68    0.02 0.55  -3.78  -3.04  -2.69  -2.32  -1.60  1187    1
## beta[2]   0.62    0.01 0.29   0.04   0.43   0.62   0.81   1.20  1132    1
## sigma     1.12    0.01 0.21   0.80   0.97   1.08   1.23   1.64  1250    1
```

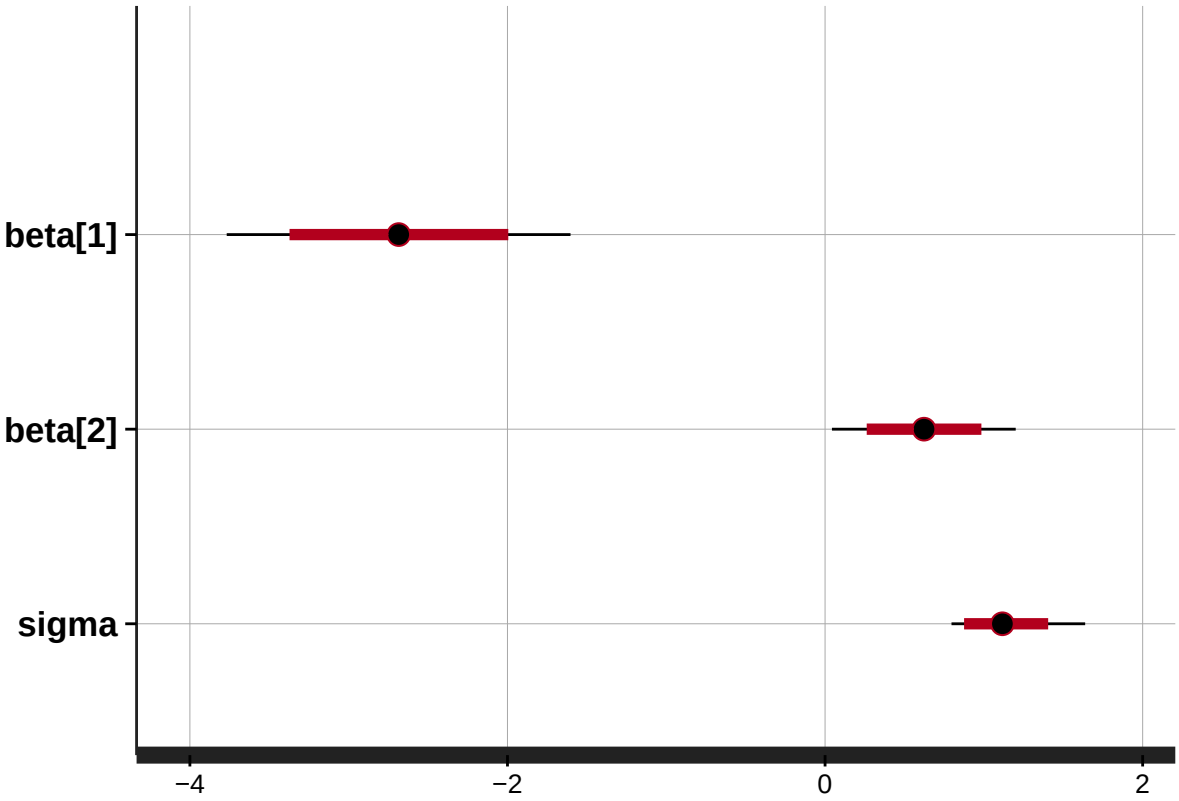


Figure 26: Default plot output for `stanfit` objects.

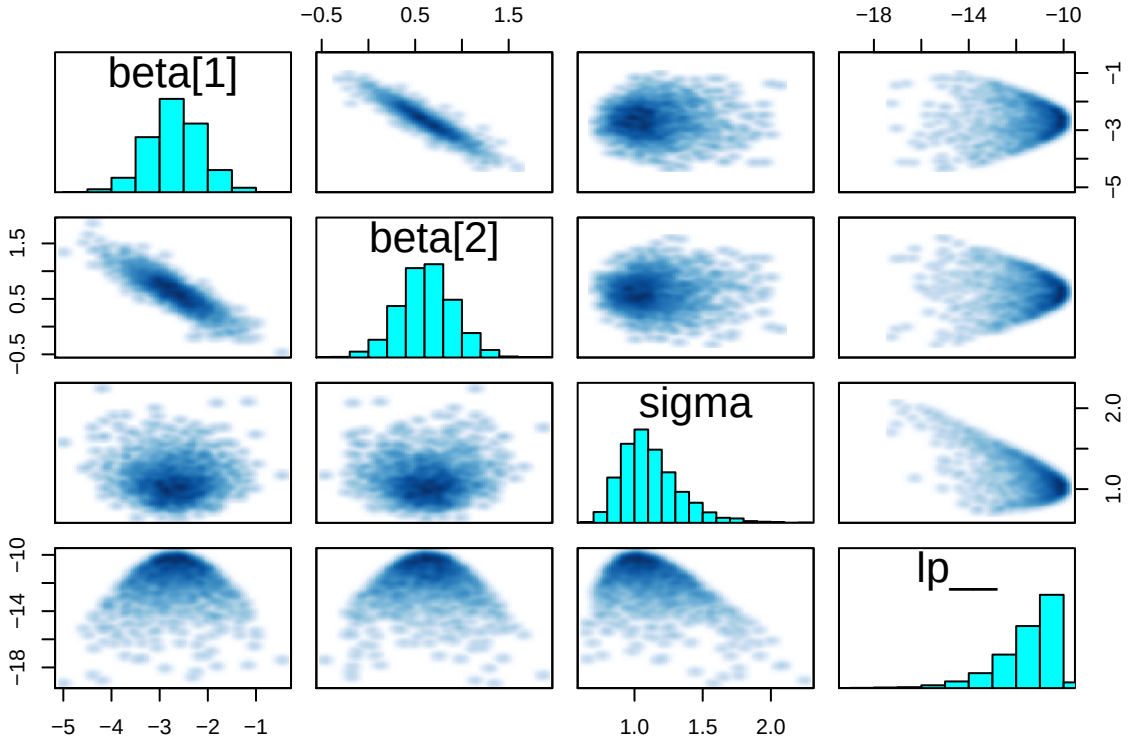


Figure 27: A pairs plot for `stanfit` objects, useful for detecting correlated posterior distributions.

```
## lp__    -11.48    0.04 1.29 -14.81 -12.09 -11.15 -10.52 -10.00  1079    1
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:42:28 2015.
## 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).
```

And finally, let's extract our samples from the posterior and plot our estimated line of best fit.

```
library(scales)
post <- extract(out)

# draw points
plot(x, y)

# add a line for each draw from the posterior
n_iter <- length(post$lp__)
for (i in 1:n_iter){
  abline(post$beta[i, 1], post$beta[i, 2], col=alpha('dodgerblue', .05))
}

# add points again so that they are visible over the line
points(x, y, pch=19)
```

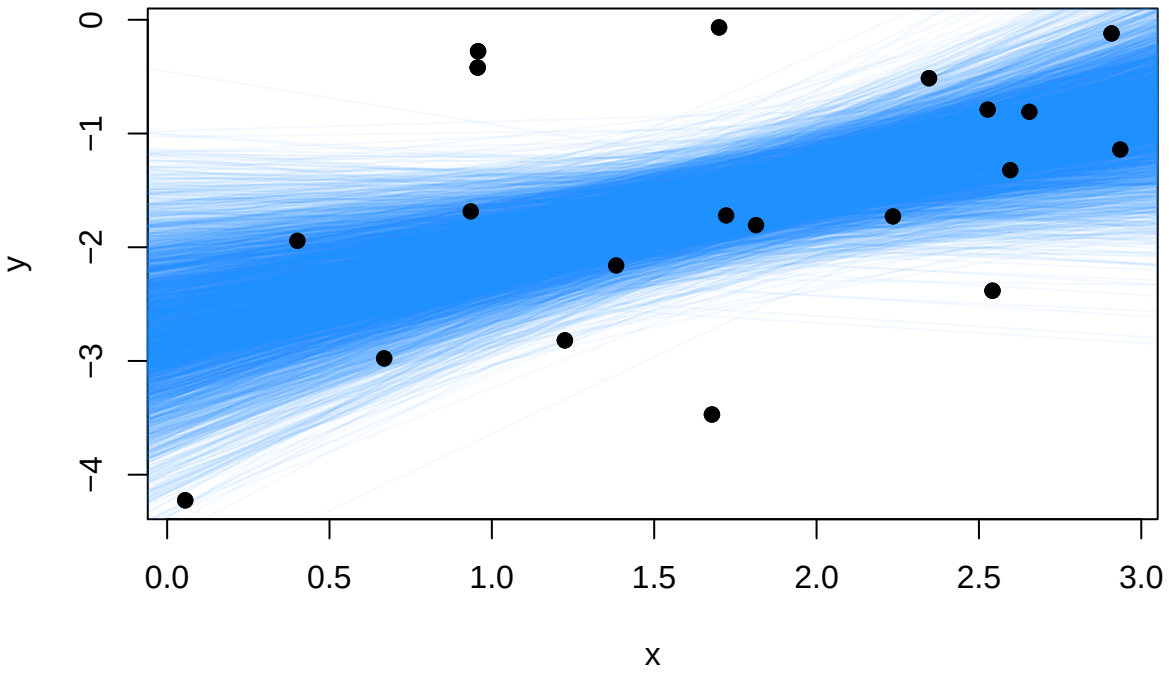


Figure 28: Raw data along with the posterior distribution of regression lines. Each draw from the posterior is shown as a line. Collectively, these lines represent the most probable regression lines conditional on the data.

We might be particularly interested in the effect of x on y. If we have abandoned frequentism, then how might we think about this? One way **not** to think of it is asking what the probability is that the slope is exactly equal to zero. If we think of a posterior probability density of our slope, then the true probability of any one particular value is zero because this is a probability density function, not a probability mass function (this is true for $\beta = 0$ and for $\beta = .0112351351$, for instance).

```
plot(density(post$beta[, 2]))
```

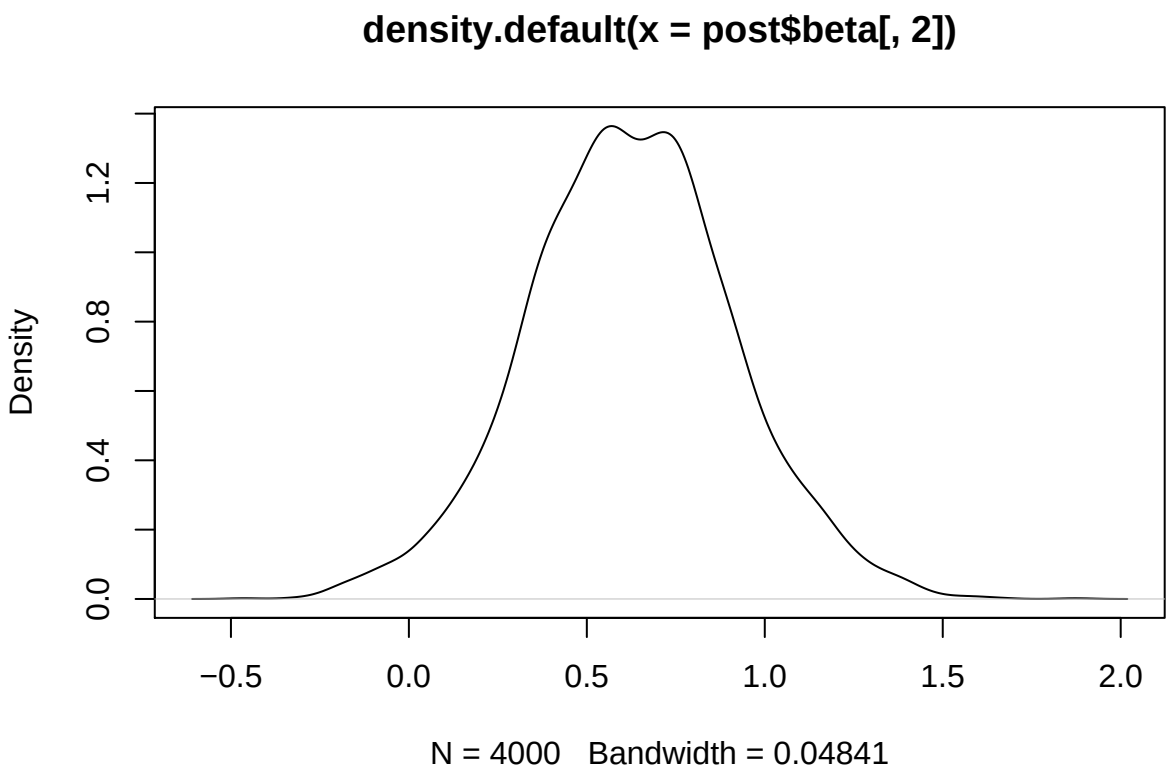


Figure 29: Density plot of the posterior distribution of the slope.

A better question (answerable with a probability density function) might be: given the data, what is the probability that x has a positive effect on y? This is equivalent to asking about the area to the right of zero in the posterior distribution of β, and the number is approximated simply the proportion of posterior draws greater than zero.

```
mean(post$beta[, 2] > 0)
```

```
## [1] 0.98125
```

```
ord <- order(post$beta[, 2])
plot(post$beta[ord, 2],
     col=ifelse(post$beta[ord, 2] > 0, 'red', 'black'),
     ylab='Sorted posterior draws for the slope')
```

We might also construct a 95% credible interval for our estimate to communicate uncertainty in our estimate of β.

```
quantile(post$beta[, 2], probs=c(0.025, 0.975))
```

```
##       2.5%      97.5%
## 0.04373813 1.20033285
```

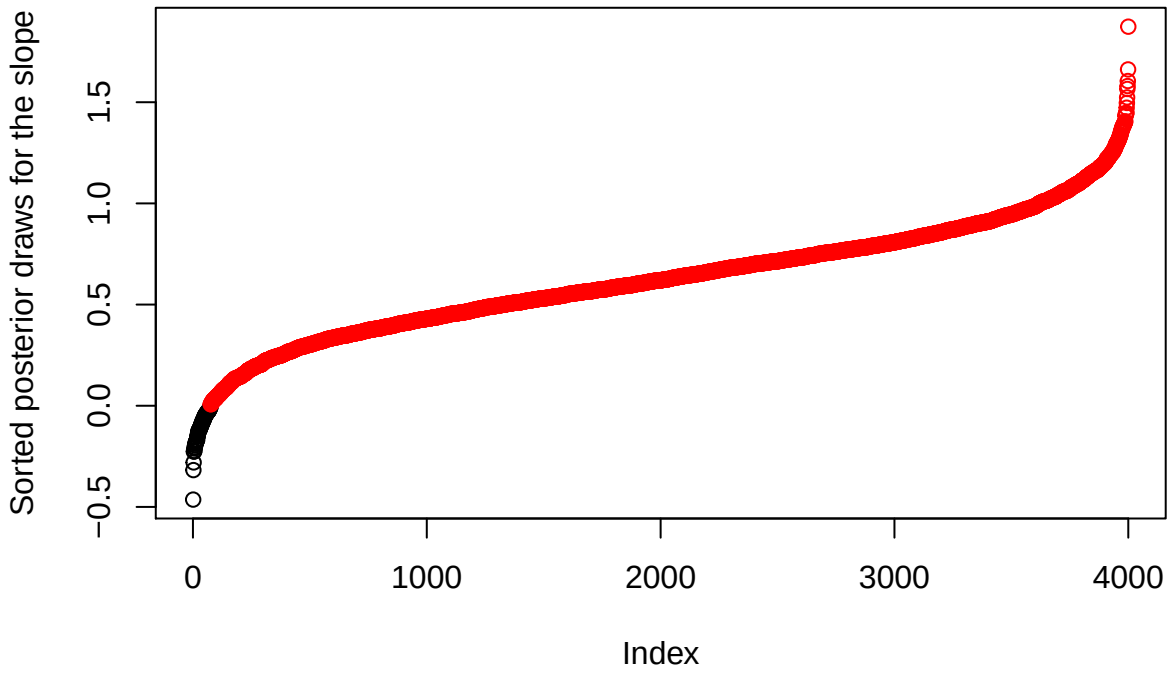


Figure 30: Sorted posterior draws for the estimated slope. These plots are useful for visually inspecting distributions.

Conditional on the data, there is a 95% probability that the true parameter value is in the credible interval. This is different from the interpretation of frequentist confidence intervals, which relies on long-run frequencies for imaginary realizations of the data collection and interval estimation procedure. Our confidence in confidence intervals is in the procedure of creating confidence intervals - in the hypothetical long run, 95% of the intervals that we construct will contain the true value. As pointed out here, this is the right answer to the wrong question.

The credible interval constructed above has equal probability density on either side of the interval because we based our interval end-points on quantiles. Sometimes, we may wish to instead construct an interval that is as narrow as possible while encapsulating some proportion of the probability mass. These intervals are called highest density posterior intervals, and can be constructed using the following function:

```
HDI <- function(values, percent=0.95){
  sorted <- sort(values)
  index <- floor(percent * length(sorted))
  nCI <- length(sorted) - index

  width <- rep(0, nCI)
  for (i in 1:nCI){
    width[i] <- sorted[i + index] - sorted[i]
  }

  HDImin <- sorted[which.min(width)]
  HDImax <- sorted[which.min(width) + index]
  HDIlim <- c(HDImin, HDImax)
  return(HDIlim)
}

HDI(post$beta[, 2])
```

```
## [1] 0.0438477 1.2015686
```

In this instance, our posterior is fairly symmetric and the two types of credible intervals will not be much different. However, if we were working with posterior distributions that were highly asymmetric, then the highest density posterior intervals and the quantile based credible intervals will tend to differ. The main consideration is that the intervals or other summary statistics reported align with the research question of interest, rather than blindly picking one summary statistic that we assume will work well all of the time.

Last, we mention that we can calculate the posterior distribution of any derived parameters immediately from the simulated posterior draws. For instance, if we were interested in the posterior distribution of the correlation between the intercept and slope, we can evaluate the correlation for each posterior draw to derive the full posterior distribution for this correlation. We could do the same if we were interested in the posterior distribution for the ratio between the slope and intercept, the difference between the slope and intercept, and so forth. This is one major advantage of the Bayesian framework in many situations, because often the targets of inference are functions of other parameters.

Further reading

Gelman and Hill. 2009. *Data analysis using regression and multilevel/hierarchical models.* Chapter 18.

Hobbs and Hooten. 2015. *Bayesian models: a statistical primer for ecologists.* Chapter 7.

Gelman et al. 2014. *Bayesian data analysis. Third edition.* Chapter 1-3.

Ellison AM. 2004. Bayesian inference in Ecology. Ecology Letters 7: 509-520.

Chapter 4: Poisson models

Big picture

The Poisson distribution is often used for discrete counts, as it has integer support. In this chapter, we show how to extend the methods of Chapter 3 to models that use Poisson likelihoods, and we point out connections to Poisson-family generalized linear models. Also, we make use of link functions as maps between constrained and unconstrained spaces, and show how to include overdispersion in Poisson models, which is often necessary in real-world applications.

Learning goals

- parameters and behavior of the Poisson distribution
- log link as a map
- effects sizes in Poisson models
- dependence of mean and variance in non-Gaussian models
- overdispersion: Poisson and negative-binomial
- implementation with `glm` and Stan
- graphical displays
- model checking
- simulation of data & parameter recovery

Poisson generalized linear models

We have covered general linear models, which by definition assume normally distributed response variables, and in this chapter we cover our first generalized linear model. Generalized linear models (glms) allow for other response distributions, including the Poisson distribution. This is admittedly a subtle distinction.

Poisson glms are defined as follows:

$$y \sim Poisson(\lambda)$$

$$log(\lambda) = X\beta$$

Where the log-link is used to map the Poisson distribution's only parameter λ from it's constrained space $[0, \infty)$ to the unconstrained space $(-\infty, \infty)$, and X is a design matrix as before. Recall that λ is the mean and variance of the Poisson distribution, in contrast for example to the Gaussian distribution which has separate parameters for the first and second moments.

Sometimes, it's desirable to include an offset. For instance, perhaps y is the number of events per unit time, or per square-meter, and each observation has different time intervals or areas sampled. Then, we might be interested in modeling the rate of events per unit (time or meters square), e.g., $\frac{\lambda}{t}$, which is in units events per time.

$$log(\lambda/t) = X\beta$$

$$log(\lambda) - log(t) = X\beta$$

$$log(\lambda) = X\beta + log(t)$$

So that now, we can still rely on $y \sim Poisson(\lambda)$, even with varying t.

The class of models that utilize the Poisson distribution is far broader than what we can do with glms, but glms provide a common and simple starting point.

Simulation and estimation

We will demonstrate with a Poisson regression, but maintain the generality of our notation by continuing to make use of the design matrix X, so that the applicability to ANOVA, ANCOVA, etc. types of applications remains apparent.

```
n <- 50
x <- runif(n, -1, 1)
X <- matrix(c(rep(1, n), x), ncol=2)
beta <- c(.5, -1)
lambda <- exp(X %*% beta)
y <- rpois(n, lambda)
```

Estimation with `glm`

We can obtain maximum likelihood estimates for Poisson glm coefficients, along with frequentist p-values and confidence intervals with the `glm` function.

```
m <- glm(y ~ x, family=poisson)
summary(m)
```

```
##
## Call:
## glm(formula = y ~ x, family = poisson)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5389  -1.0767  -0.1323   0.5973   2.5412
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.6672     0.1083   6.163 7.16e-10 ***
## x            -0.8465     0.1764  -4.798 1.61e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 94.829  on 49  degrees of freedom
## Residual deviance: 69.211  on 48  degrees of freedom
## AIC: 185.4
##
## Number of Fisher Scoring iterations: 5
```

```r
confint(m)
```

```
## Waiting for profiling to be done...
```

```
##                  2.5 %     97.5 %
## (Intercept)  0.4454474  0.8708271
## x           -1.2018638 -0.5088015
```

Plotting the line of best fit with the data:

```r
plot(x, y)

xnew <- seq(min(x), max(x), length.out = 100)
lines(xnew, exp(coef(m)[1] + coef(m)[2] * xnew))
```

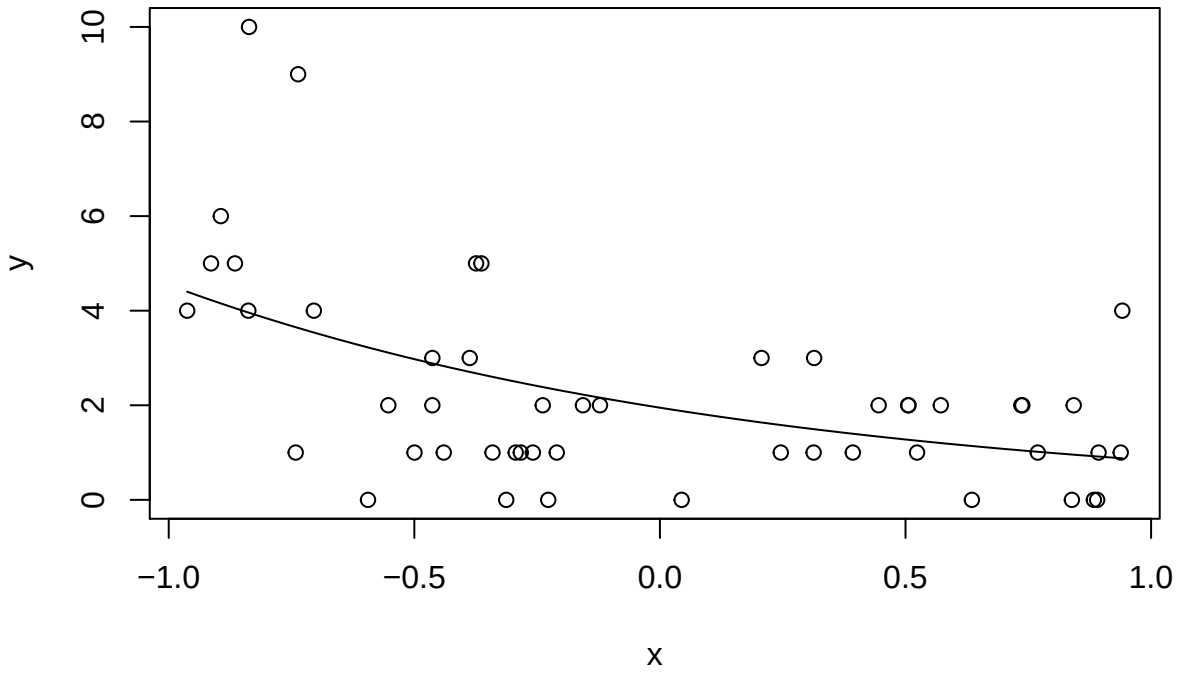


Figure 31: Line of best fit from the Poisson glm along with the raw data.

Estimation with Stan

Even though Poisson glms are often taught after Gaussian glms, they are in a way simpler, requiring one less parameter. Here is a Stan file `poisson_glm.stan` that can be used generally to fit any Poisson glm without an offset.

```
data {
  int n; // sample size
  int p; // number of coefficients
  matrix[n, p] X;
  int y[n];
```

```
}

parameters {
  vector[p] beta;
}

model {
  beta ~ normal(0, 3);
  y ~ poisson_log(X * beta);
}
```

Fitting the model is the same as before:

```
library(rstan)
X <- matrix(c(rep(1, n), x), ncol = 2)
stan_d <- list(n = nrow(X), p = ncol(X), X = X, y = y)
out <- stan('poisson_glm.stan', data = stan_d)
```

As always, it behooves us evaluate convergence and see whether there may be issues with the MCMC algorithm.

```
out
```

```
## Inference for Stan model: poisson_glm.
## 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
## beta[1]  0.66    0.00 0.11   0.44   0.59  0.66  0.73  0.86  1698    1
## beta[2] -0.85    0.00 0.18  -1.20  -0.97 -0.84 -0.73 -0.49  1537    1
## lp__    -9.94    0.03 1.02 -12.81 -10.34 -9.62 -9.21 -8.96  1426    1
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:42:35 2015.
## 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).
```

```
traceplot(out)
```



```
plot(out)
```



```
## ci_level: 0.8 (80% intervals)
## outer_level: 0.95 (95% intervals)
```

```
pairs(out)
```


Let's plot the lines of best fit from the posterior distribution:

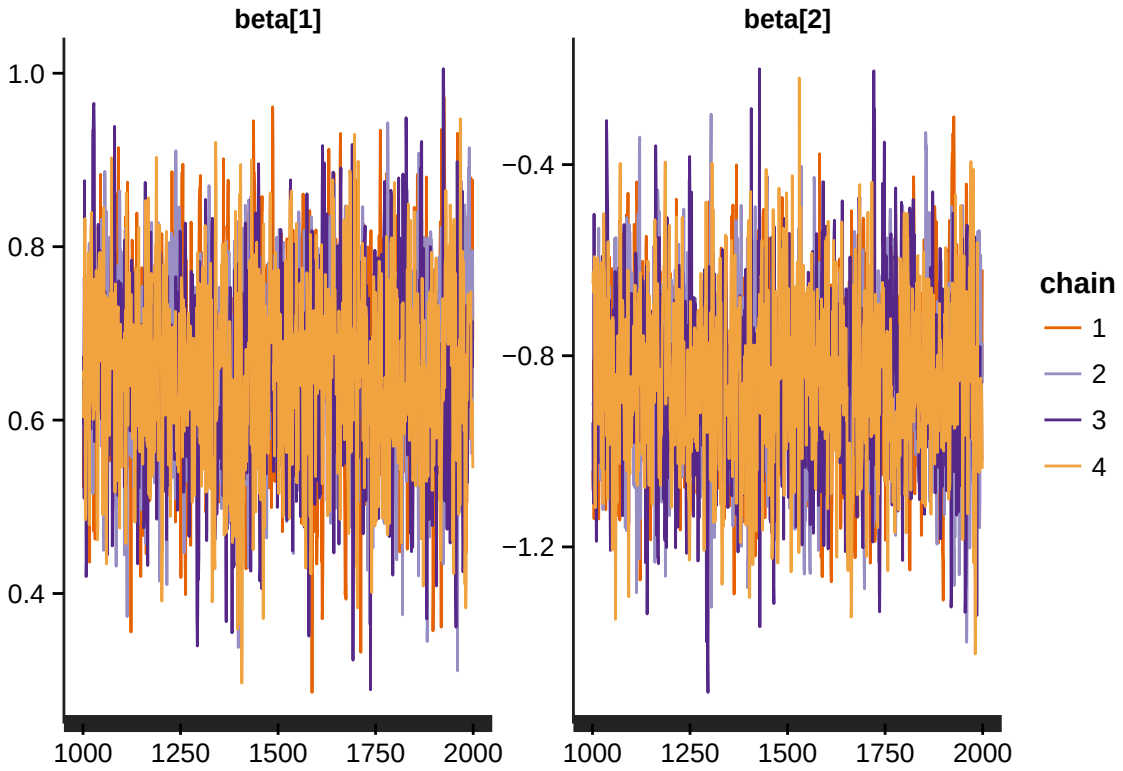


Figure 32: Traceplot of Markov chains for the Poisson glm.

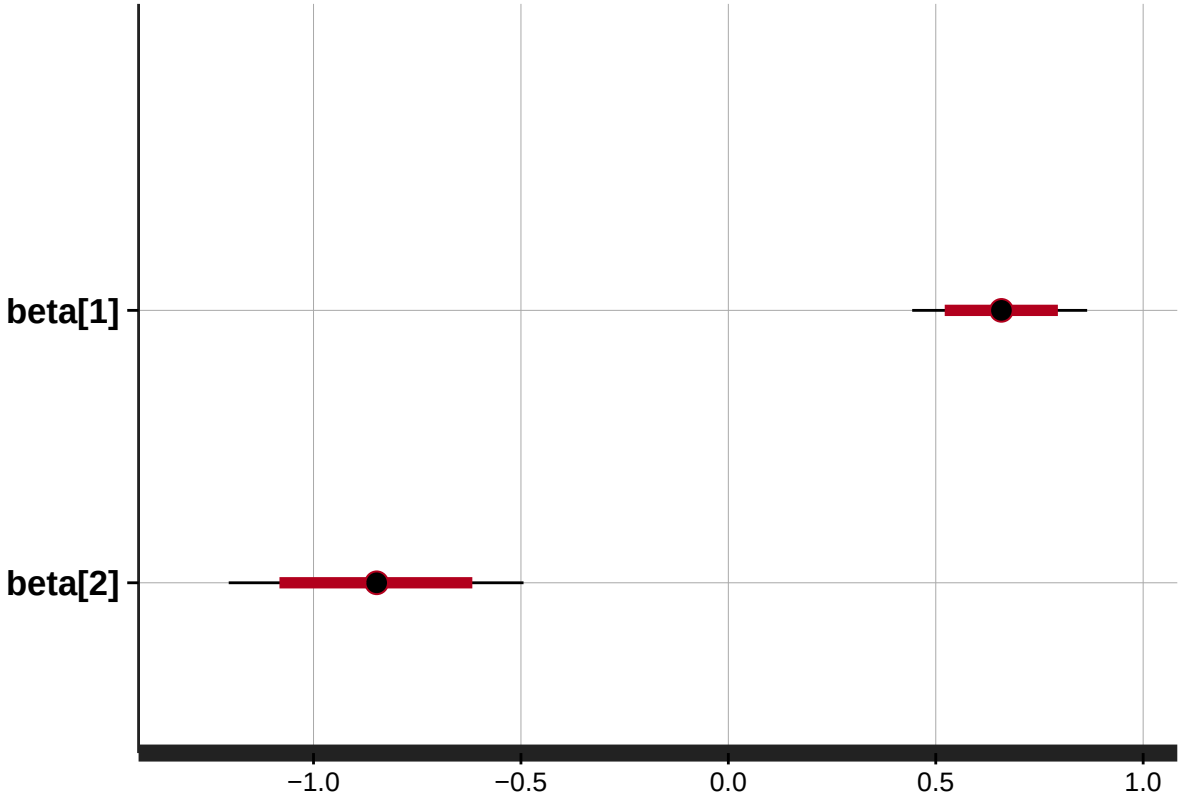


Figure 33: Default `stanfit` plot output for the Poisson glm.

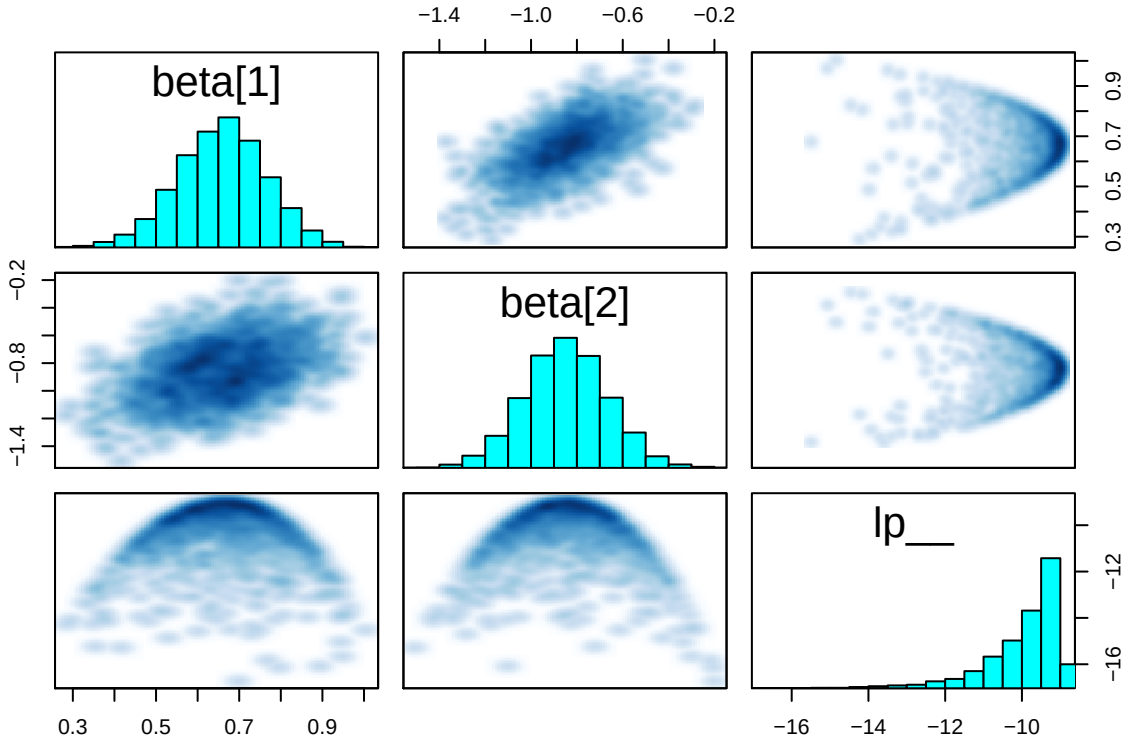


Figure 34: A pairs plot for the posterior of the Poisson glm.

```
library(scales)
post <- extract(out)
plot(x, y)
n_iter <- length(post$lp__)
for (i in 1:n_iter){
  lines(xnew, exp(post$beta[i, 1] + post$beta[i, 2] * xnew),
        col = alpha('purple3', .01))
}
points(x, y, pch=19)
```

At times, people get confused about why the fits from Poisson glms are not linear, even though we didn't include any polynomial terms. The model is linear on the link-scale only, so that if we were to plot x vs. $log(\lambda)$, we would see a straight line. There is a lot of good material in Gelman and Hill, Chapter 6 to help with the interpretation of glm parameter estimates. For many applications, visualization is an excellent step towards interpreting model output.

Overdispersion

For various reasons, the variance of counts in nature tends to be greater than the mean. This can be considered to be extra-Poisson variance. Various strategies exist to account for overdispersion, and here we cover the inclusion of lognormal random effects and also derive the negative binomial distribution as a Poisson distribution with a gamma-distributed λ.

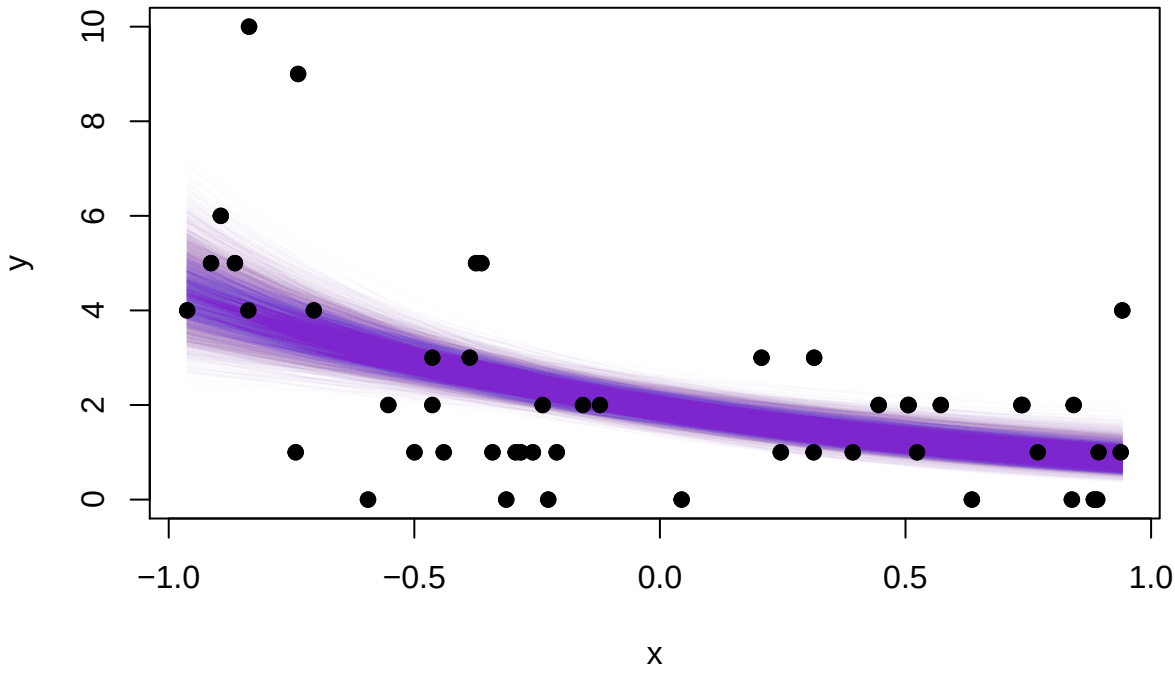


Figure 35: Raw data along with lines of best fit for the Poisson glm.

Checking for overdispersion

How do you know whether the variance in your data exceeds the variance you would expect in the absence of overdispersion? One general strategy that falls under the class of model diagnostics known as **posterior predictive checks** does the following:

1. For each posterior draw, simulate a new vector of observations.
2. For each simulated vector of observations, calculate some test statistic of interest.
3. Compare the distribution of simulated test statistics to the empirical test statistic.

In this case, it is reasonable to choose the variance of y as the test statistic.

```
# simulate new observations and store variances
y_new <- array(dim=c(n_iter, n))
var_new <- rep(NA, n_iter)
for (i in 1:n_iter){
  y_new[i, ] <- rpois(n, exp(X %*% post$beta[i, ]))
  var_new[i] <- var(y_new[i, ])
}

# compare distribution of simulated values to the empirical values
hist(var_new, breaks=40, xlab='Var(y)',
     main='Posterior predictive check \n for overdispersion')
abline(v = var(y), col=2, lwd=3)
```

In this case, we can say that the variance in y shown by the red line is perfectly compatible with the model, which makes sense because we generated the data from a Poisson distribution.

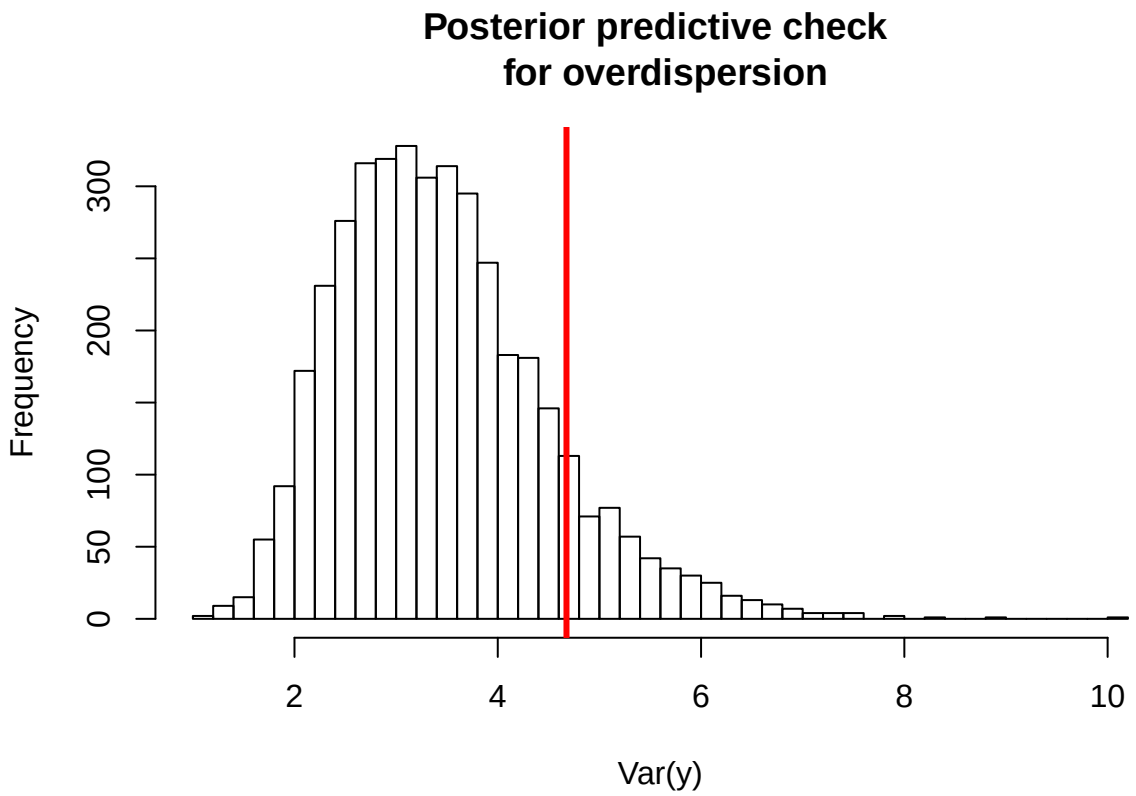


Figure 36: Histogram of simulated variances from the posterior predictive distribution along with the observed variance, comprising a posterior predictive check for the variance of y.

Here's a real world application with overdispersion. The `vegan` package has a dataset of the number of trees in a bunch of 1-hectare plots on Barro Colorado Island. Let's look at the mean and variance of each species to get a sense for how often Poisson distribution might work:

```
library(vegan)
library(dplyr)
data(BCI)

# coerce into long form
d <- stack(BCI)
str(d)
```

```
## 'data.frame':    11250 obs. of  2 variables:
##  $ values: int  0 0 0 0 0 0 0 0 0 1 ...
##  $ ind   : Factor w/ 225 levels "Abarema.macradenia",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
# calculate means and variances
summ_d <- d %>%
  group_by(ind) %>%
  summarize(lmean=log(mean(values)),
            lvar=log(var(values)))
ggplot(summ_d, aes(x=lmean, y=lvar)) +
  geom_point() +
  geom_abline(intecept=0, slope=1, linetype='dashed') +
  xlab("log mean") +
  ylab("log variance")
```

Darn. Looks like the variance exceeds the mean for most of these species. Let's put together a simple model for the abundance of the species *Trichilia pallida*, where we seek to estimate the mean density for the island based on the sampled plots.

```
# subset data
species_d <- subset(d, ind == 'Trichilia.pallida')

# visualize abundance data
plot(sort(species_d$values), ylab="Sorted abundance")
```

```
# collect data for estimation
stan_d <- list(n = nrow(species_d), p = 1,
               X = matrix(1, nrow = nrow(species_d)),
               y = species_d$values)
out <- stan('poisson_glm.stan', data = stan_d)
```

Assessing convergence:

```
out
```

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

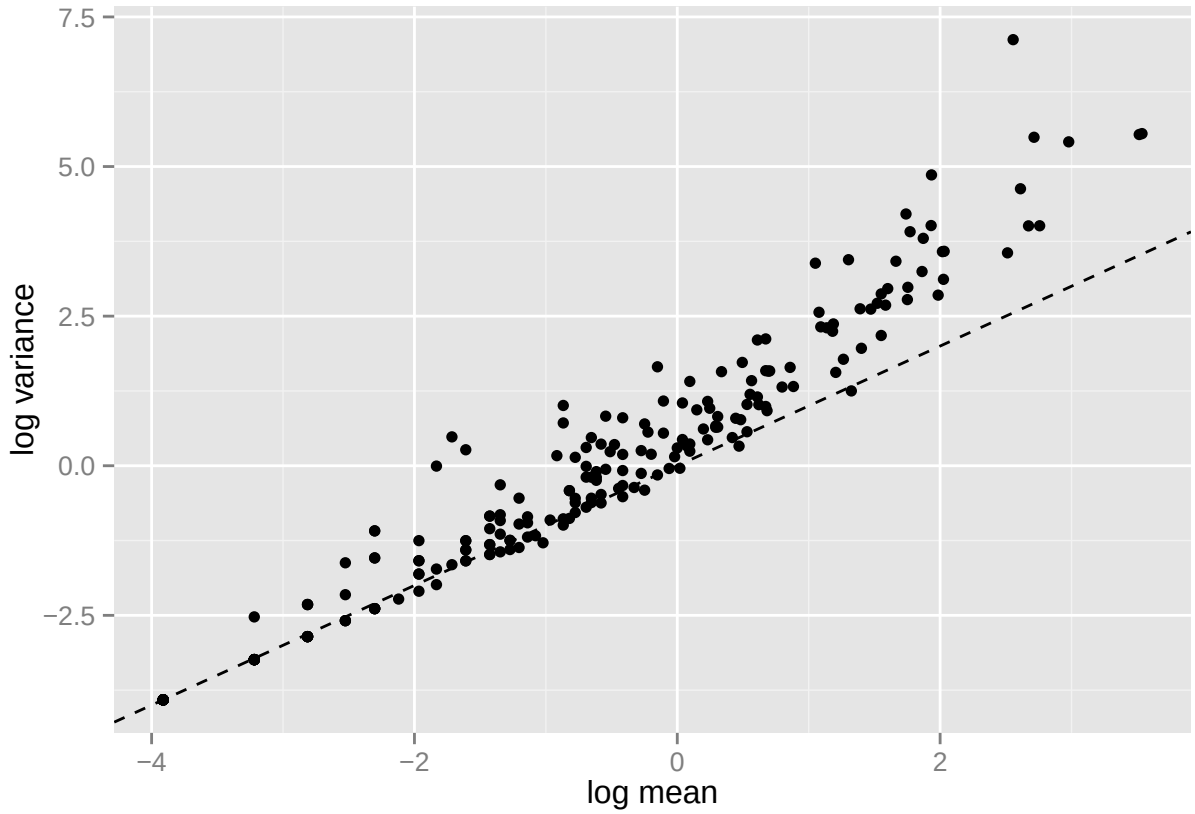


Figure 37: Sample mean and variances for abundance on Barro Colorado Island, with each point representing a different species.

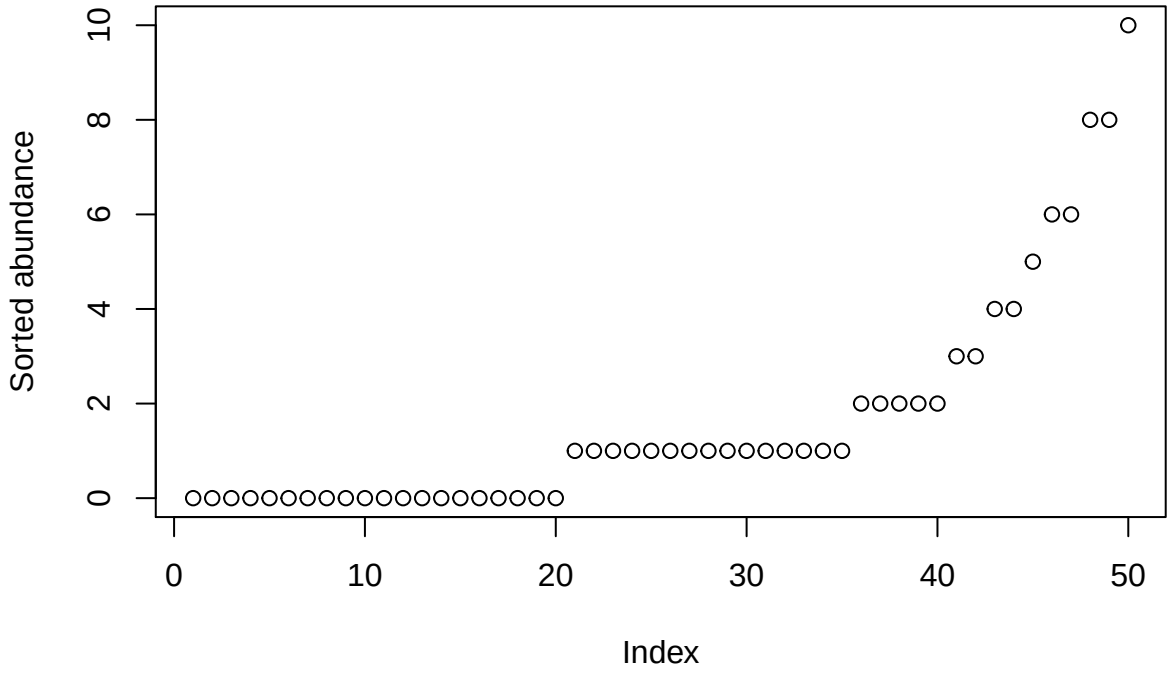


Figure 38: Sorted abundance values for *Trichilia pallida*, with each point representing the abundance of the species in a sample plot.

```
##           mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## beta[1]   0.49    0.00 0.11   0.27   0.42   0.49   0.56   0.70  1509 1.00
## lp__    -41.92    0.02 0.69 -43.86 -42.09 -41.65 -41.49 -41.45  1832 1.01
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:42:42 2015.
## 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).
```

```
traceplot(out)
```

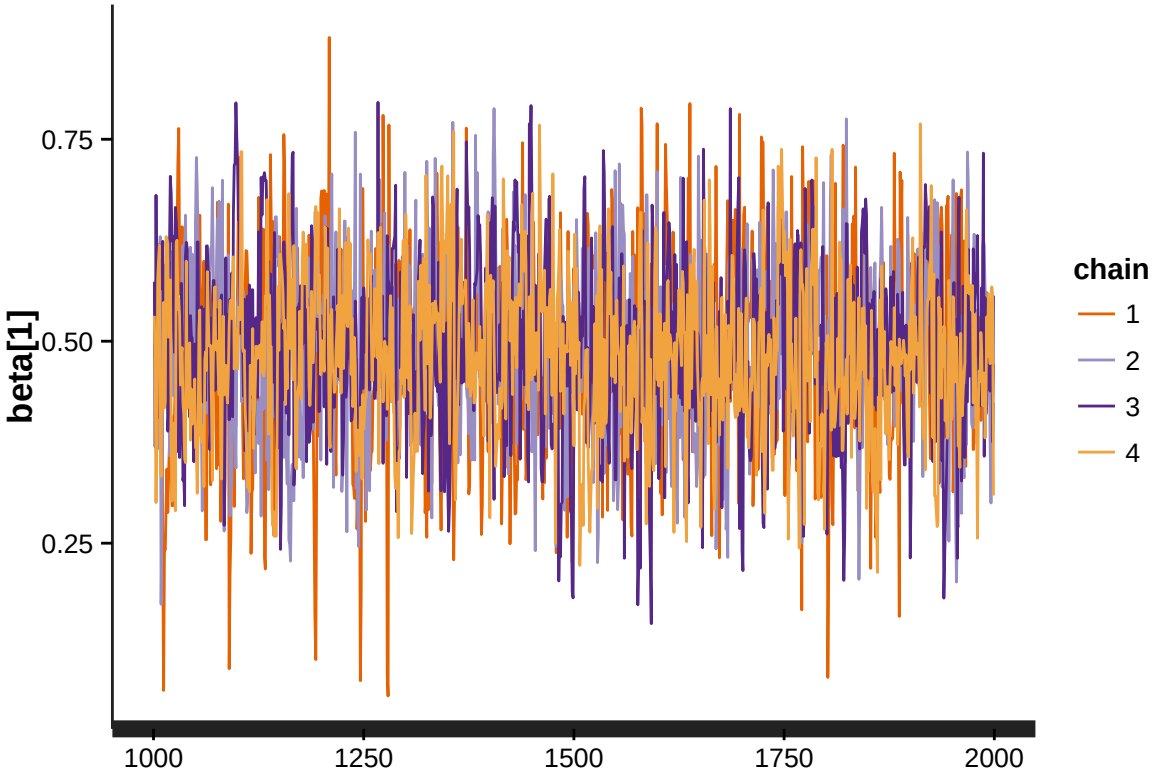


Figure 39: Traceplot of the Markov chain for the single species Poisson glm.

Using a posterior predictive check for the variance of y:

```
post <- extract(out)

# simulate new observations and store variances
y_new <- array(dim=c(n_iter, n))
var_new <- rep(NA, n_iter)
for (i in 1:n_iter){
  y_new[i, ] <- rpois(n, exp(post$beta[i, 1]))
  var_new[i] <- var(y_new[i, ])
}

# compare distribution of simulated values to the empirical values
hist(var_new, breaks=40, xlab='Var(y)', xlim=c(0, 7),
```

```
     main='Posterior predictive check \n for overdispersion')
abline(v = var(stan_d$y), col=2, lwd=3)
```

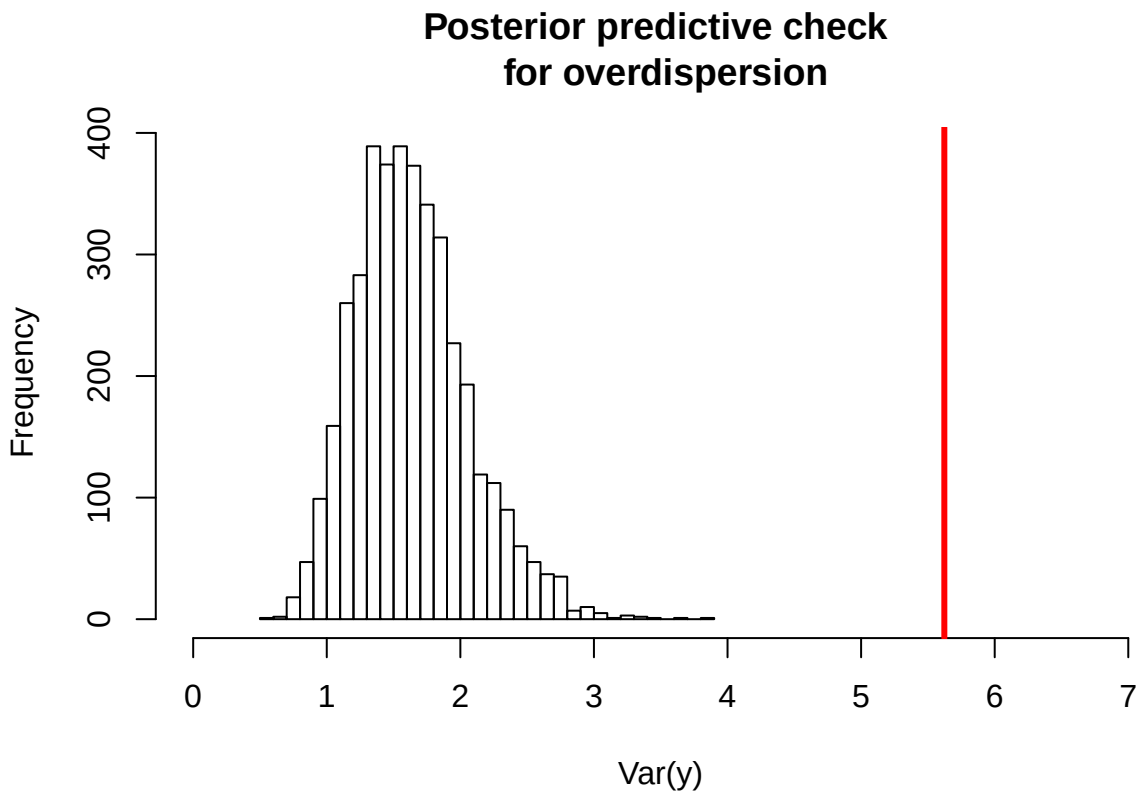


Figure 40: Posterior predictive check for the variance of plot abundance from the single species Poisson glm.

As we can see, the observed variance is more than twice the expected variance under a Poisson model.

Lognormal overdispersion

We will expand our model so that we can include some overdispersion. First, we will allow for additional plot-level variance by adding a term to our linear predictor:

$$y_i \sim Poisson(\lambda_i)$$

$$log(\lambda_i) = X_i'\beta + \epsilon_i$$

$$\epsilon_i \sim Normal(0, \sigma)$$

$$\sigma \sim halfCauchy(0, 2)$$

Here, each observation y_i has an associated parameter ϵ_i that has a normal prior, with a variance hyperparameter that determines how much extra-Poisson variance we have. If there is no overdispersion, then the posterior probability mass for σ should be near zero. Our Stan model `poisson_od.stan` might be:

```
data {
  int n; // sample size
  int p; // number of coefficients
  matrix[n, p] X;
  int y[n];
}

parameters {
  vector[p] beta;
  vector[n] epsilon;
  real<lower=0> sigma;
}

model {
  // priors
  beta ~ normal(0, 3);
  sigma ~ normal(0, 2);
  epsilon ~ normal(0, sigma);

  // likelihood
  y ~ poisson_log(X * beta + epsilon);
}
```

Our data haven't changed, but we do need to specify the path to this updated model:

```
od_out <- stan('poisson_od.stan', data = stan_d)
```

```
print(od_out, pars=c('sigma', 'beta', 'lp__'))
```

```
## Inference for Stan model: poisson_od.
## 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
## sigma     1.23    0.01 0.24   0.81   1.06   1.20   1.38   1.79   602    1
## beta[1]  -0.18    0.01 0.27  -0.76  -0.35  -0.17   0.01   0.30   685    1
## lp__    -30.23    0.31 7.33 -45.40 -34.95 -29.94 -25.11 -16.55   544    1
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:42:45 2015.
## 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).
```

```
traceplot(od_out, pars=c('sigma', 'beta'))
```

Using an updated posterior predictive check for the variance of y:

```
post <- extract(od_out)

# simulate new observations and store variances
y_new <- array(dim=c(n_iter, n))
var_new <- rep(NA, n_iter)
```

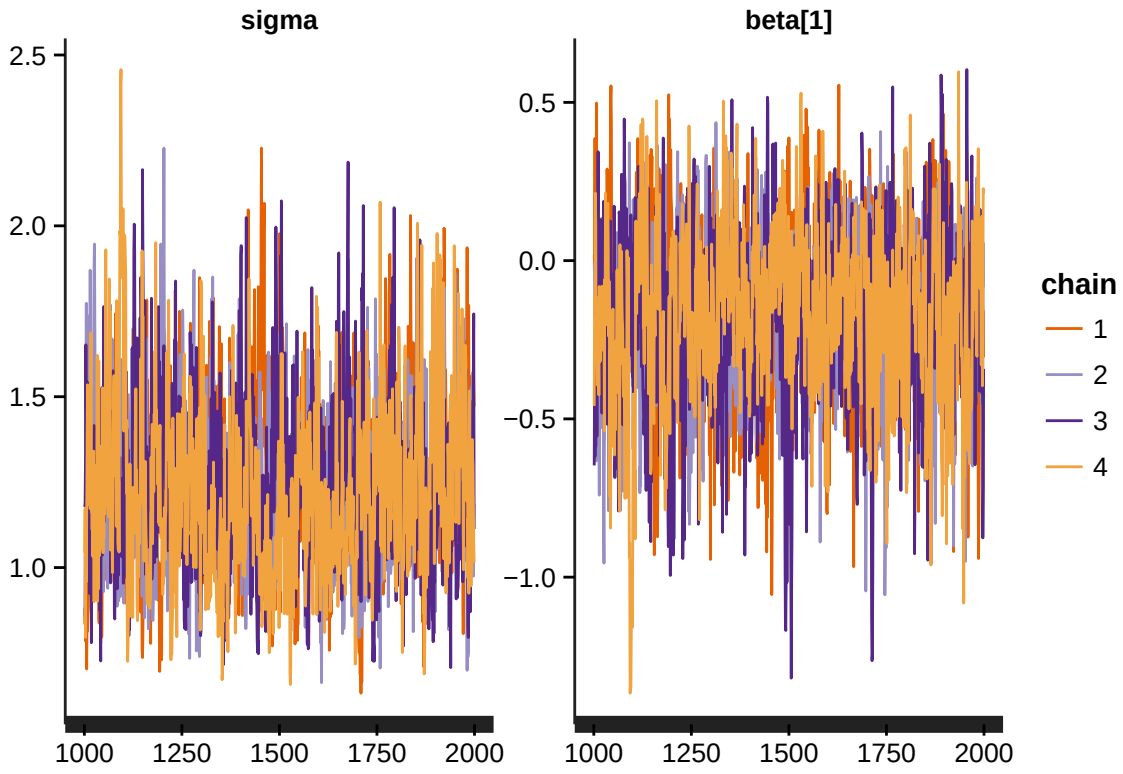


Figure 41: Traceplot of the Markov chain for the overdispersed Poisson model.

```
for (i in 1:n_iter){
  y_new[i, ] <- rpois(n, exp(post$beta[i, 1] + post$epsilon[i, ]))
  var_new[i] <- var(y_new[i, ])
}

# compare distribution of simulated values to the empirical values
hist(var_new, breaks=40, xlab='Var(y)',
     main='Posterior predictive check \n for overdispersion')
abline(v = var(stan_d$y), col=2, lwd=3)
```

That looks much better. The variance that we get from our replicated simulations is entirely consistent with the variance that we actually observed.

Poisson-gamma overdispersion and the negative binomial

The negative binomial distribution can be thought of as a Poisson distribution with a gamma prior on λ.

$$y_i \sim Poisson(\lambda_i)$$

$$\lambda_i \sim Gamma(\alpha, \beta)$$

To give a sense of what this looks like, here are some gamma distributions with varying parameters:

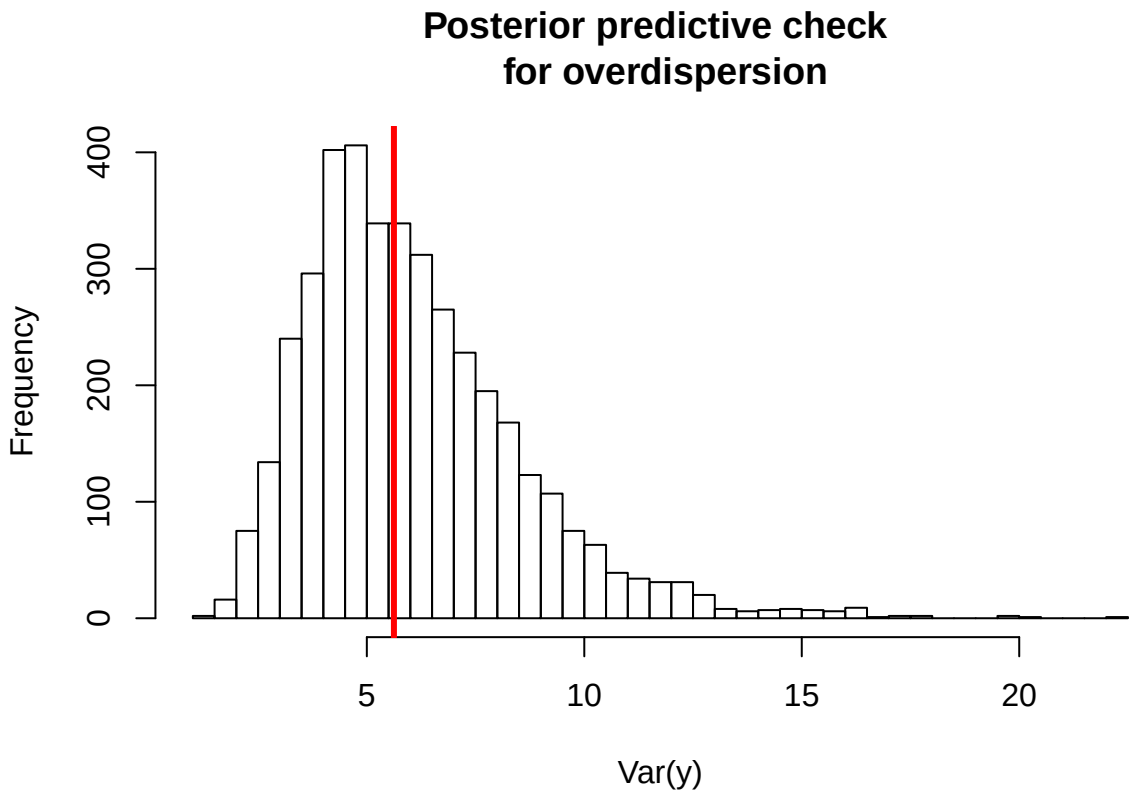


Figure 42: Updated posterior predictive check for the variance of y with the overdispersed Poisson model.

```
alpha <- c(.1, 1, 10)
beta <- c(.1, 1, 10)
g <- expand.grid(alpha = alpha, beta = beta)
x <- seq(0, 20, .1)
par(mfrow=c(3, 3))
for (i in 1:9){
  plot(x, dgamma(x, g$alpha[i], g$beta[i]), type='l',
       ylab='[x]')
}
```

We could implement this directly in Stan, but it would be easier to use the built-in and optimized negative binomial distribution function, which is parameterized in terms of its mean μ and precision ϕ:

$$y \sim NegBinom(\mu, \phi)$$

$$log(\mu) = X\beta$$

$$\phi \sim halfCauchy(0, 2)$$

```
data {
  int n; // sample size
  int p; // number of coefficients
```

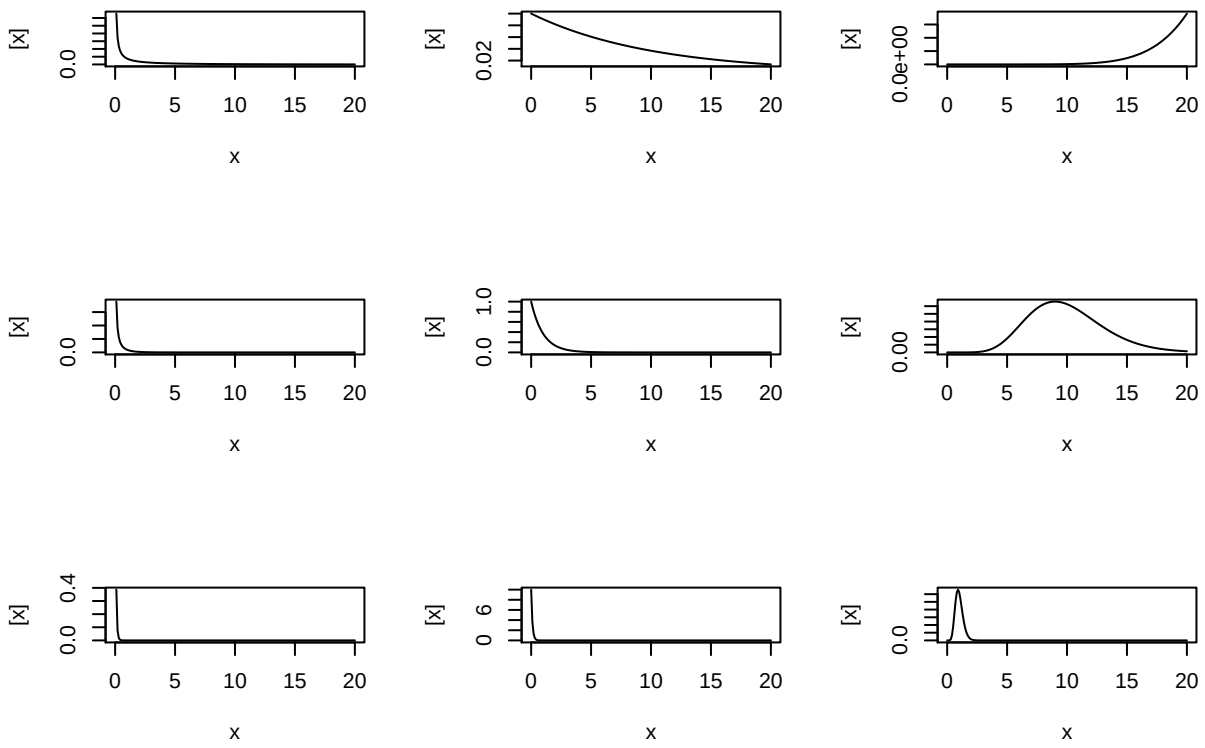


Figure 43: Probability densities of Gamma distributions with different parameters.

```
  matrix[n, p] X;
  int y[n];
}

parameters {
  vector[p] beta;
  real<lower=0> phi;
}

model {
  beta ~ normal(0, 3);
  phi ~ cauchy(0, 5);
  y ~ neg_binomial_2_log(X * beta, phi);
}
```

With this `nb_glm.stan` file in hand, we can fit the model as before:

```
nb_out <- stan('nb_glm.stan', data = stan_d)
```

```
nb_out
```

```
## Inference for Stan model: nb_glm.
## 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%
```

```
## beta[1] 5.500000e-01 5.000000e-02 1.800000e-01   0.15   0.41   0.59
## phi     3.412716e+15 4.179184e+15 5.911737e+15   0.42   0.68   0.91
## lp__    3.599100e+02 4.662200e+02 6.594900e+02 -23.15 -20.92 -20.20
##                 75%        97.5% n_eff         Rhat
## beta[1] 6.600000e-01 8.800000e-01    12 1.090000e+00
## phi     3.412716e+15 1.365086e+16     2 4.854954e+14
## lp__    3.606800e+02 1.502040e+03     2 8.261600e+02
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:44:03 2015.
## 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).
```

```
traceplot(nb_out)
```

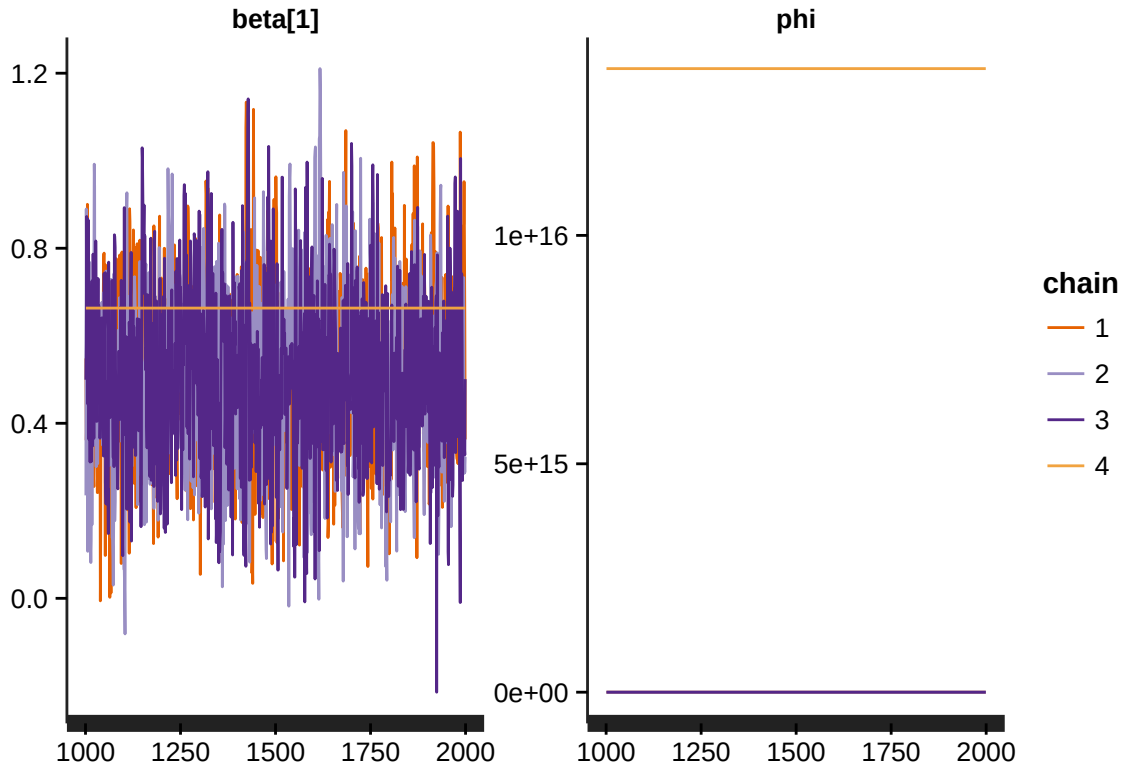


Figure 44: Traceplot of the Markov chains for the negative binomial model.

Conducting our posterior predictive check:

```
post <- extract(nb_out)

# simulate new observations and store variances
y_new <- array(dim=c(n_iter, n))
var_new <- rep(NA, n_iter)
for (i in 1:n_iter){
  y_new[i, ] <- rnbinom(n, mu = exp(post$beta[i, 1]), size = post$phi[i])
  var_new[i] <- var(y_new[i, ])
}
```

```
# compare distribution of simulated values to the empirical values
par(mfrow=c(1, 1))
hist(var_new, breaks=40, xlab='Var(y)',
     main='Posterior predictive check \n for overdispersion')
abline(v = var(stan_d$y), col=2, lwd=3)
```

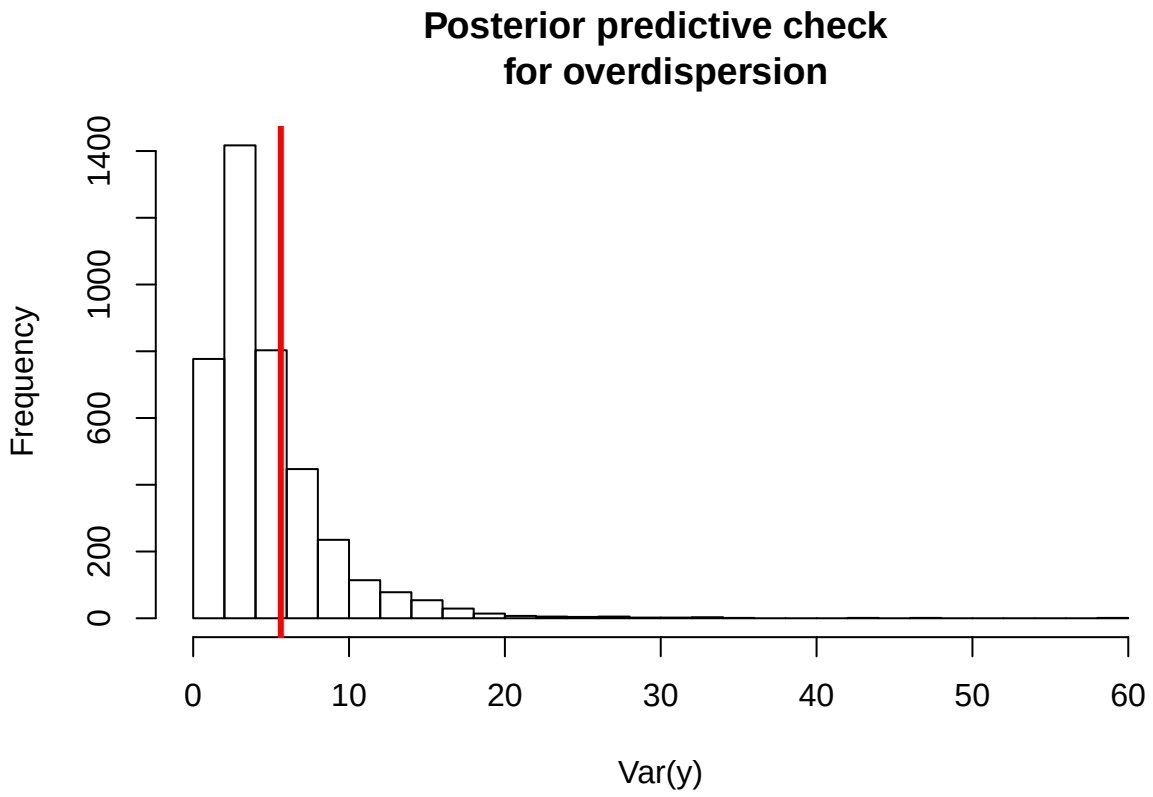


Figure 45: Posterior predictive check for the variance in y with the negative binomial likelihood.

Again, the replicated datasets that are simulated from the posterior predictive distribution now have variance consistent with the observed data.

Further reading

Gelman and Hill. 2009. *Data analysis using regression and multilevel/hierarchical models.* Chapter 6.

Gelman et al. 2014. *Bayesian data analysis. Third edition.* Chapter 16.

Chapter 5: Binomial models

Big picture

The binomial distribution is used all over the place in ecology, so it's important to familiarize yourself with the properties/behavior of the bionomial distribution. In this section we will cover the binomial in the context of glms, but we will also point out some useful hierarchical models that include the binomial distribution, including occupancy models, N-mixture models, and binomial-Poisson hierarchical models.

Learning goals

- properties of the binomial distribution
- link functions for the binomial
- implementation with `glm`
- binomial overdispersion
- implementation with Stan
- occupancy models
- graphical displays
- model checking
- simulation of data & parameter recovery

Binomial generalized linear models

The binomial distribution is used for integer valued random variables that represent the number of successes in k independent trials, each with probability p. For instance, the number of chicks surviving would be a binomial random variable, with the number of eggs laid k and a survival probability p. Usually, k is known and p is estimated, but there are some useful hierarchical models that treat k as a parameter that we will cover later (e.g., N-mixture models, and binomial-Poisson hierarchy models). When $k = 1$, the binomial distribution is also called the Bernoulli distribution. In binomial glms, a link function is needed to map p from the constrained probability space to an unconstrained space. Usually either the logit or probit link functions are used, and both are inverse cumulative distribution functions for other distributions (the logistic and the Gaussian, respectively). We can write a binomial glm as follows:

$$y \sim Binomial(p, k)$$

$$logit(p) = X\beta$$

Where $logit(p) = log(p/(1-p))$, and β is a parameter vector to be estimated.

Simulation and estimation

Imagine that we're interested in whether different egg companies are more or less likely to have broken eggs in our neighborhood grocery store, and we peek into 20 cartons from each of 4 companies, each with 12 eggs per carton. We know $k = 12$ for each carton, and we want to know $p_1, ..., p_4$, the probability of broken eggs for the 4 companies. This is analogous to ANOVA, but with a binomial response.

```r
m <- 4
n_each <- 20
company <- rep(LETTERS[1:m], each = n_each)
p <- rep(c(.02, .01, .05, .1), each = n_each)
n <- length(p)
k <- rep(12, n)
broken <- rbinom(n, size = k, prob = p)
not_broken <- 12 - broken
boxplot(broken ~ company)
```

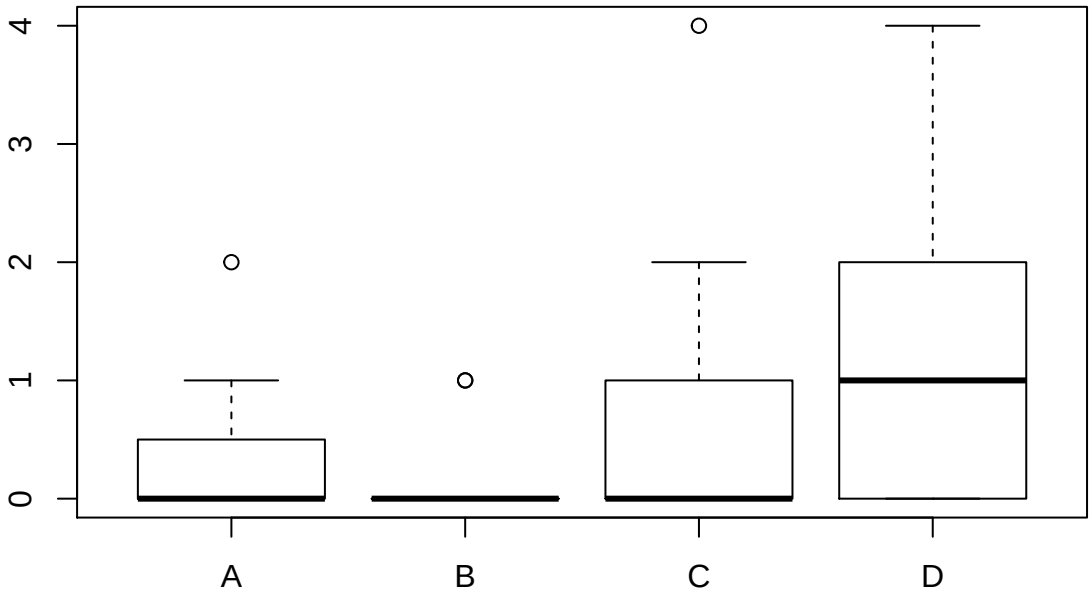


Figure 46: Boxplots for the number of eggs broken in dozen egg cartons across different companies.

Estimation with glm

We can estimate the probabilities for each company as follows:

```r
m <- glm(cbind(broken, not_broken) ~ 0 + company, family = binomial)
summary(m)
```

```
##
## Call:
## glm(formula = cbind(broken, not_broken) ~ 0 + company, family = binomial)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.51903  -0.77950  -0.44815   0.02773   2.99002
##
## Coefficients:
##          Estimate Std. Error z value Pr(>|z|)
## companyA  -3.6636     0.4134  -8.861  < 2e-16 ***
## companyB  -4.7791     0.7101  -6.730 1.69e-11 ***
## companyC  -2.8600     0.2852 -10.029  < 2e-16 ***
```

```
## companyD  -2.2935     0.2237 -10.252  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1085.940  on 80  degrees of freedom
## Residual deviance:   82.497  on 76  degrees of freedom
## AIC: 150.68
##
## Number of Fisher Scoring iterations: 6
```

By default the coefficients are returned on the logit scale. We can back-transform with the cumulative distribution function for the logistic distribution, `plogis()`:

```
plogis(coef(m))
```

```
##    companyA    companyB    companyC    companyD
## 0.025000000 0.008333333 0.054166667 0.091666667
```

```
plogis(confint(m))
```

```
## Waiting for profiling to be done...
```

```
##                2.5 %     97.5 %
## companyA 0.010011145 0.05000587
## companyB 0.001390244 0.02549424
## companyC 0.030166671 0.08760214
## companyD 0.059504555 0.13248575
```

Estimation with Stan

To do a Bayesian analysis we need priors for β, for instance so that our model is:

$$y \sim Binomial(p, k)$$

$$logit(p) = X\beta$$

$$\beta \; Normal(0, 2)$$

Where the normal(0, 2) prior for beta is fairly uninformative on the logit-scale. Note that in this example, this basically communicates the idea that we have never bought eggs in a carton before. Realistically, we would select priors that are concentrated toward small probabilities!

Our Stan model for a binomial glm (`binomial_glm.stan`) might look like this:

```
data {
  int n; // sample size
  int p; // number of coefficients
```

```
  matrix[n, p] X;
  int y[n];
  int k[n];
}

parameters {
  vector[p] beta;
}

model {
  beta ~ normal(0, 2);
  y ~ binomial_logit(k, X * beta);
}
```

We can recycle the design matrix made with glm:

```
library(rstan)
X <- model.matrix(m)
stan_d <- list(n = nrow(X),
               p = ncol(X),
               X = X,
               y = broken,
               k = k)
out <- stan('binomial_glm.stan', data = stan_d)
```

```
out
```

```
## Inference for Stan model: binomial_glm.
## 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
## beta[1]   -3.59    0.01 0.40   -4.45   -3.83   -3.56   -3.32   -2.87  2241
## beta[2]   -4.47    0.01 0.58   -5.73   -4.82   -4.43   -4.05   -3.48  2290
## beta[3]   -2.85    0.01 0.28   -3.45   -3.03   -2.83   -2.65   -2.33  2677
## beta[4]   -2.29    0.00 0.22   -2.76   -2.43   -2.28   -2.13   -1.86  2907
## lp__    -171.60    0.04 1.48 -175.32 -172.33 -171.27 -170.52 -169.80  1536
##         Rhat
## beta[1]    1
## beta[2]    1
## beta[3]    1
## beta[4]    1
## lp__       1
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:44:06 2015.
## 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).
```

```
traceplot(out)
```

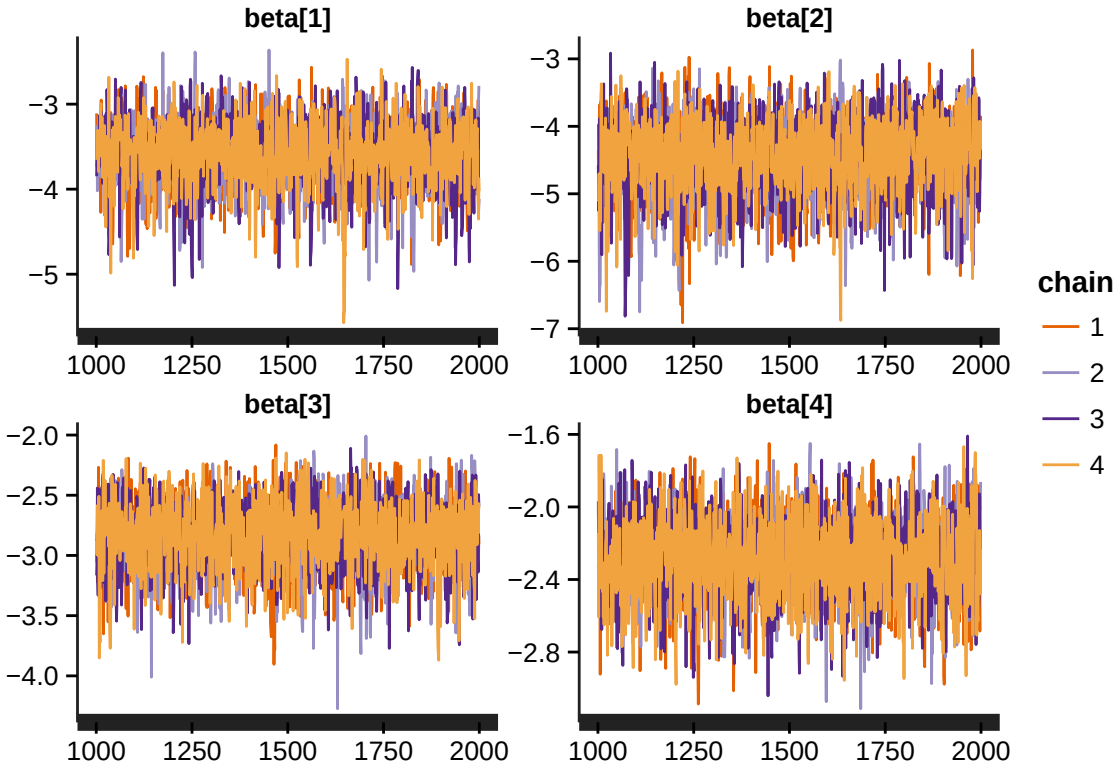


Figure 47: Traceplots for the Markov chains from the simple binomial glm.

```
plot(out)
```

```
## ci_level: 0.8 (80% intervals)
## outer_level: 0.95 (95% intervals)
```

How might we graph the results? Ideally our plot should show the data and our model output (in this case, the expected number of broken eggs). In this case, we might try something like this:

```
# put data into a data frame
d <- data.frame(broken, company)

# create data frame for posterior estimates
library(dplyr)
library(reshape2)
library(ggplot2)
library(grid)

# extract posterior samples and get summaries
post <- out %>%
  extract() %>%
  melt() %>%
  subset(L1 != 'lp__') %>%
  group_by(Var2) %>%
  summarize(median = 12 * plogis(median(value)),
            lo = 12 * plogis(quantile(value, .025)),
```

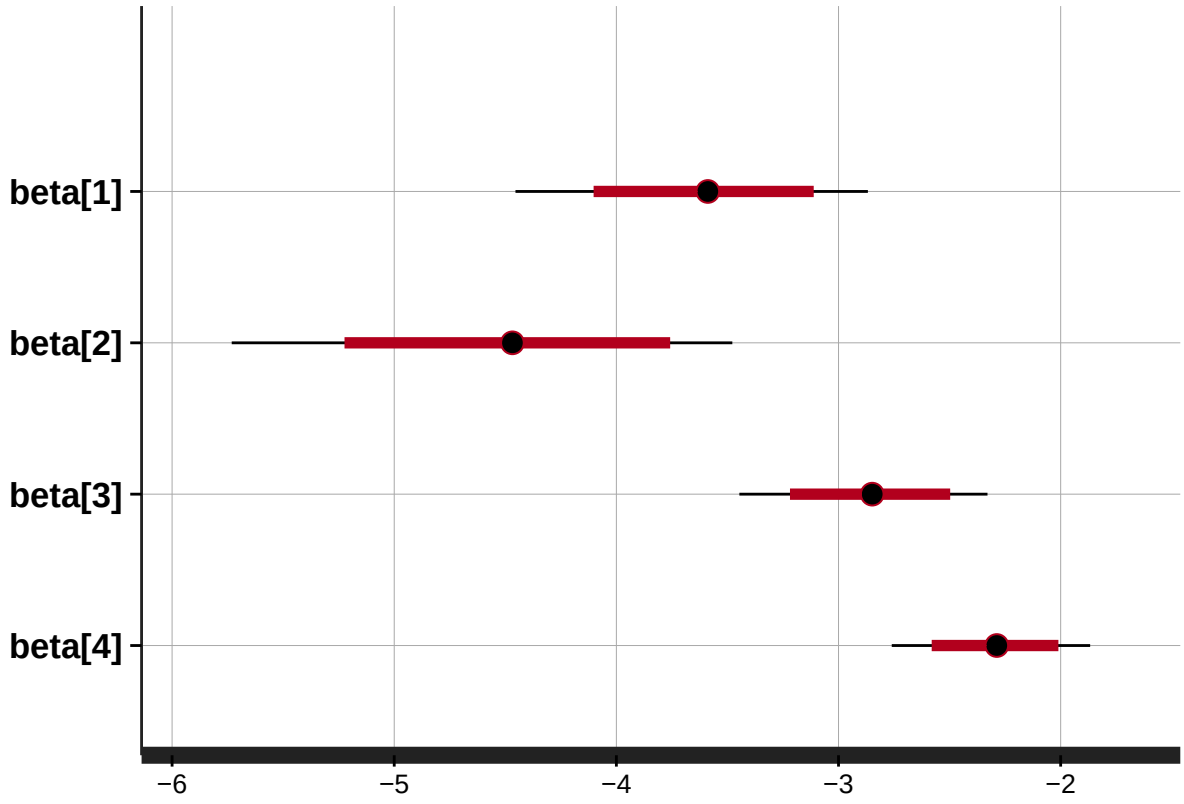


Figure 48: Default plot output for the binomial glm.

```
            hi = 12 * plogis(quantile(value, .975)))
post$company <- unique(company)

# plot results
ggplot(d, aes(x=company, y=broken)) +
  geom_segment(aes(xend=company, y=lo, yend=hi), data=post,
               size=3, col='blue', alpha=.3) +
  geom_point(aes(x=company, y=median), data=post,
             col='blue', size=3) +
  geom_jitter(position=position_jitter(width=.1, height=.1),
              shape=1) +
  ylab("Number of broken eggs")
```

Overdispersion

The binomial distribution has mean np and variance $np(1-p)$, but sometimes there is more variation than we would expect from the binomial distribution. For instance, in the previous example, we should expect sources of variation other than the companies that produce and distribute the eggs. Any particular carton may have a different history than other cartons due to chance - maybe the stocker set one carton down hard, or an unattended child opened a carton and broke an egg then closed the carton, etc. Though binomial overdispersion receives little attention in ecology compared to Poisson overdispersion, it is a common problem in real-world data. We should note, however, that if $k = 1$ (the Bernoulli case), binomial overdispersion is not identifiable. If Bernoulli random variables can be converted to binomial random variables with $k > 1$,

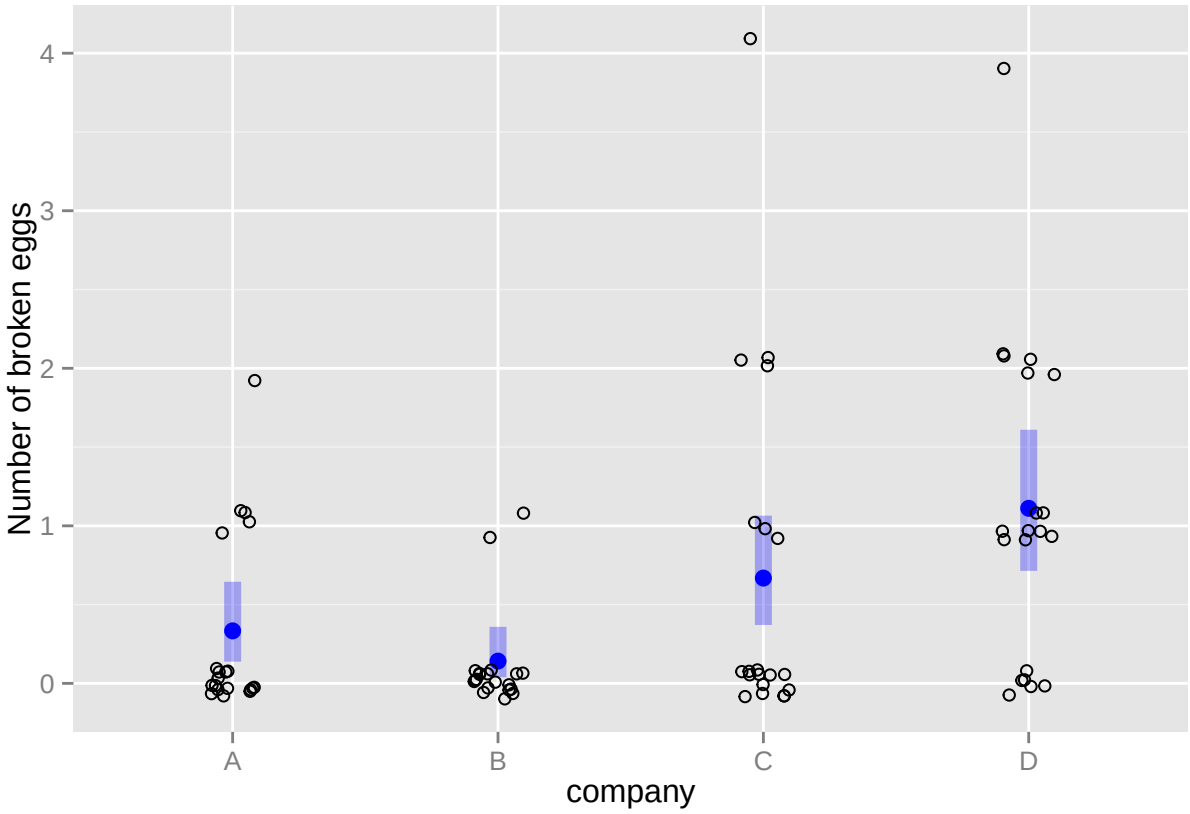


Figure 49: Raw data along with posterior HDIs for the probability of an egg being broken across the different companies.

overdispersion may be identifiable. As with the Poisson example, you can use posterior predictive checks to evaluate whether your model is underrepresenting the variance in binomial observations.

Binomial-normal model

One common solution to overdispersion is the addition of a normally distributed random effect to the linear predictor. This represents variation at the level of individual observations.

$$y_i \sim Binomial(p_i, k_i)$$

$$logit(p_i) = X_i'\beta + \epsilon_i$$

$$\beta \sim Normal(0, 2)$$

$$\epsilon_i \sim Normal(0, \sigma)$$

$$\sigma \sim halfCauchy(0, 2)$$

This is very similar to the lognormal overdispersion strategy for Poisson models covered in the previous chapter.

Beta-binomial model

Another option is to use a beta distribution as a prior for the probability of success parameter p:

$$y \sim Binomial(p, k)$$

$$p \sim beta(\alpha, \beta)$$

This strategy tends to be used infrequently in ecology, and we do not cover it in depth here, but there are good resources on the web for implementing beta-binomial models in Stan here, and here.

Further reading

Gelman and Hill. 2009. *Data analysis using regression and multilevel/hierarchical models.* Chapter 5, 6.

Gelman et al. 2014. *Bayesian data analysis. Third edition.* Chapter 16.

Chapter 6: Partial pooling and likelihood

Partial pooling is one of the primary motivations behind conventional hierarchical models. Also termed "borrowing information", partial pooling improves estimates of group-level parameters, particularly when we have > 3 groups and/or varying sample sizes among groups. In this chapter, we illustrate partial pooling and link it to prior distributions in a maximum-likelihood context.

Learning goals

- motivation for and definition of partial pooling
- simple hierarchical models with likelihood
- hyperparameters
- varying intercepts (NBA freethrow example) with `lme4`
- partial pooling
- clearing up confusion about nestedness
- predictors for multiple levels
- plotting estimates for different levels from lme4 models

Partial pooling: free throw example

Often, data are structured hierarchically. For instance, maybe we sample individual animals within sites, with multiple sites. Or, perhaps we sequence multiple genes across multiple individuals. There may be more than two levels, for instance if we sample parasites of different species within individuals of different species of hosts across multiple sites in a landscape. Commonly, sample sizes are not equal across units at various levels. The following example demonstrates how to use partial pooling to generate reliable estimates in the context of wildly varying sample sizes.

Suppose we are interested in knowing who the best free throw shooter was in the 2014-2015 NBA season. We can pull the data from the web, and plot the proportion of free-throws made by player.

```
rawd <- read.csv("leagues_NBA_2015_totals_totals.csv")
str(rawd)
```

```
## 'data.frame':    651 obs. of  30 variables:
##  $ Rk    : int  1 2 3 4 5 5 5 6 7 8 ...
##  $ Player: Factor w/ 492 levels "Aaron Brooks",..: 384 249 438 212 36 36 36 8 152 85 ...
##  $ Pos   : Factor w/ 11 levels "C","PF","PF-SF",..: 2 9 1 2 9 9 9 1 2 1 ...
##  $ Age   : int  24 20 21 28 29 29 29 26 23 26 ...
##  $ Tm    : Factor w/ 31 levels "ATL","BOS","BRK",..: 20 15 21 18 29 8 25 19 23 20 ...
##  $ G     : int  68 30 70 17 78 53 25 68 41 61 ...
##  $ GS    : int  22 0 67 0 72 53 19 8 9 16 ...
##  $ MP    : int  1287 248 1771 215 2502 1750 752 957 540 976 ...
##  $ FG    : int  152 35 217 19 375 281 94 181 40 144 ...
##  $ FGA   : int  331 86 399 44 884 657 227 329 78 301 ...
##  $ FG.   : num  0.459 0.407 0.544 0.432 0.424 0.428 0.414 0.55 0.513 0.478 ...
##  $ X3P   : int  18 10 0 0 118 82 36 0 0 0 ...
##  $ X3PA  : int  60 25 2 0 333 243 90 0 5 0 ...
##  $ X3P.1 : num  0.3 0.4 0 NA 0.354 0.337 0.4 NA 0 NA ...
##  $ X2P   : int  134 25 217 19 257 199 58 181 40 144 ...
##  $ X2PA  : int  271 61 397 44 551 414 137 329 73 301 ...
##  $ X2P.1 : num  0.494 0.41 0.547 0.432 0.466 0.481 0.423 0.55 0.548 0.478 ...
##  $ eFG.  : num  0.486 0.465 0.544 0.432 0.491 0.49 0.493 0.55 0.513 0.478 ...
```

```
##  $ FT    : int  76 14 103 22 167 127 40 81 13 50 ...
##  $ FTA   : int  97 23 205 38 198 151 47 99 27 64 ...
##  $ FT.   : num  0.784 0.609 0.502 0.579 0.843 0.841 0.851 0.818 0.481 0.781 ...
##  $ ORB   : int  79 9 199 23 27 21 6 104 78 101 ...
##  $ DRB   : int  222 19 324 54 220 159 61 211 98 237 ...
##  $ TRB   : int  301 28 523 77 247 180 67 315 176 338 ...
##  $ AST   : int  68 16 66 15 129 101 28 47 28 75 ...
##  $ STL   : int  27 16 38 4 41 32 9 21 17 37 ...
##  $ BLK   : int  22 7 86 9 7 5 2 51 16 65 ...
##  $ TOV   : int  60 14 99 9 116 83 33 69 17 59 ...
##  $ PF    : int  147 24 222 30 167 108 59 151 96 122 ...
##  $ PTS   : int  398 94 537 60 1035 771 264 443 93 338 ...
```

Some players switched teams mid-season, and they appear on separate rows, one for each team they played on. We need to aggregate across the teams, so that we end up with only one row per player:

```
library(dplyr)
library(ggplot2)

# clean the data
d <- rawd %>%
  group_by(Player) %>%
  summarize(ft_made = sum(FT),
            ft_miss = sum(FTA) - sum(FT),
            ft_shot = sum(FTA),
            ft_pct = sum(FT) / sum(FTA)) %>%
  subset(ft_shot != 0) %>%
  arrange(-ft_pct) %>%
  droplevels()
d
```

```
## Source: local data frame [475 x 5]
##
##                   Player ft_made ft_miss ft_shot ft_pct
##                   (fctr)   (int)   (int)   (int)  (dbl)
## 1              Alex Kirk       2       0       2      1
## 2  Chris Douglas-Roberts       8       0       8      1
## 3            C.J. Wilcox       2       0       2      1
## 4          Grant Jerrett       2       0       2      1
## 5              Ian Clark      18       0      18      1
## 6          Jannero Pargo       2       0       2      1
## 7           Jerel McNeal       2       0       2      1
## 8         John Lucas III       5       0       5      1
## 9          Kenyon Martin       2       0       2      1
## 10          Luigi Datome       2       0       2      1
## ..                   ...     ...     ...     ...    ...
```

```
# order factor levels by ft_pct and plot
levels(d$Player) <- levels(d$Player)[order(-d$ft_pct)]
d$shooter <- factor(d$Player, levels = d$Player[order(d$ft_pct)])
ggplot(d, aes(x=ft_pct, y=shooter)) +
  geom_point(stat='identity') +
  theme(text = element_text(size=8))
```

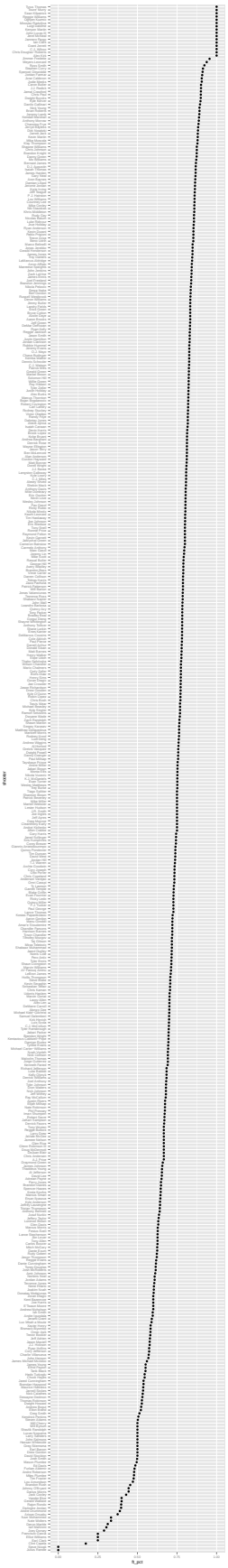


Figure 50: Empirical free throw probabilities for NBA players in the 2014-2015 season.

Wow! Looks like we have some really good (100% accuracy) and really bad (0% accuracy) free throw shooters in the NBA. We can verify this by calculating maximum likelihood estimates for the probability of making a free throw for each player. We'll assume that the number of free throws made is a binomial random variable, with p_i to be estimated for the i^{th} player, and k equal to the number of free throw attempts.

$$y_i \sim Binom(p_i, k_i)$$

```
# fit binomial glm
m <- glm(cbind(ft_made, ft_miss) ~ 0 + Player,
         family=binomial, data=d)

# print estimated probabilities
probs <- m %>%
  coef() %>%
  plogis() %>%
  round(digits=4) %>%
  sort(decreasing=TRUE)
```

It turns out that the maximum likelihood estimates are equal to the proportion of free throws made.

```
plot(d$ft_pct, probs,
     xlab="Empirial proportion FT made",
     ylab="MLE: Pr(make FT)")
```

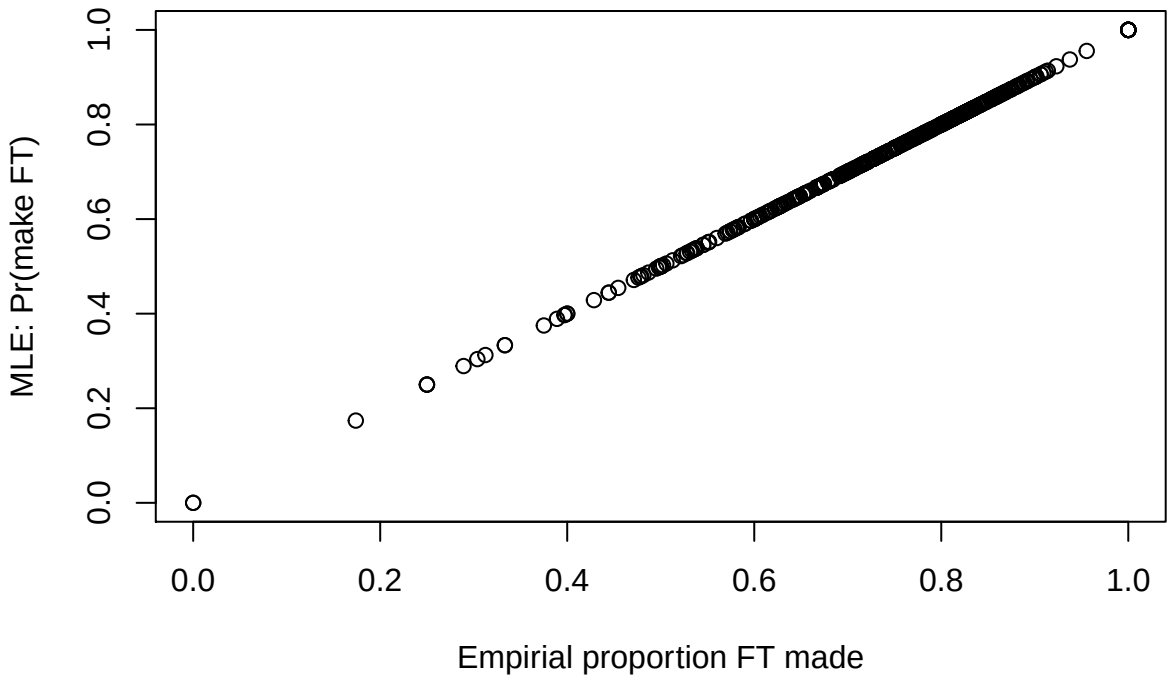


Figure 51: Plot showing 1:1 correspondence between the empirical proportion of free throws made and the maximum likelihood estimates for making a free throw.

But, can we really trust these estimates? What if we plot the maximum likelihood estimates along with the number of free throw attempts?

```
ggplot(d, aes(x=ft_shot, y=ft_pct)) +
  geom_point(alpha=.6) +
  xlab('Free throw attempts') +
  ylab('Proportion of free throws made')
```

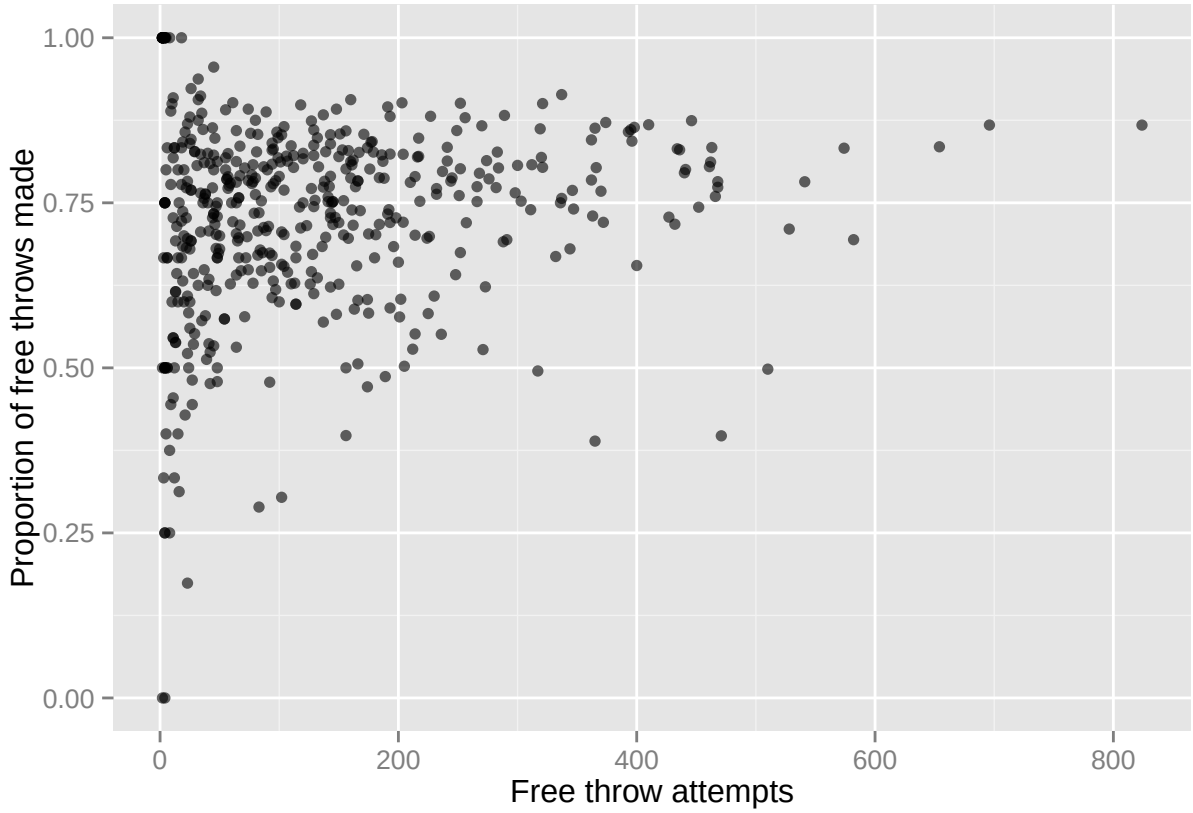


Figure 52: Maximum likelihood estimates along with the sample size for each player. Notice how there is much more variation in the MLEs for players who took few shots.

Well that's not good. It looks like the players with the highest and lowest shooting percentages took the fewest shots. We should be skeptical of these estimates. One solution is to select some minimum number of attempts, and only believe estimates for players who have taken at least that number. This is what the NBA does, and the limit is 125 shots.

```
d %>%
  filter(ft_shot >= 125) %>%
  ggplot(aes(x=ft_shot, y=ft_pct)) +
  geom_point(alpha=.6) +
  xlab('Free throw attempts') +
  ylab('Proportion of free throws made')
```

This seems somewhat arbitrary - what's special about the number 125? Is there a better way to decide which estimates to trust? What if instead of tossing out the data from players that haven't taken at least 125 shots, we tried to somehow improve those estimates? For instance, we might instead pull these estimates towards the average, and place increasingly more trust in proportions from players with more information (shot attempts). This is the intuition behind partial pooling, and the secret to implementation lies in placing a prior distribution on p_i, the probability that player i makes a free throw, so that:

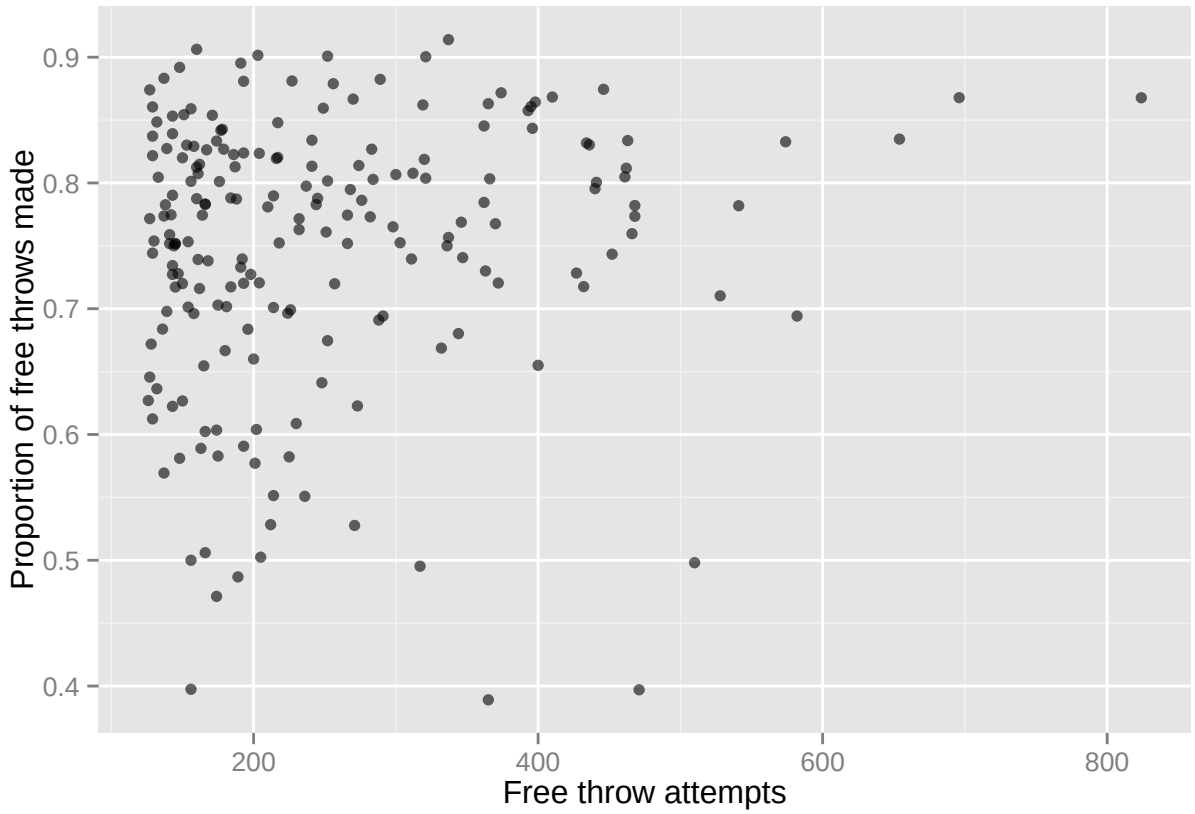


Figure 53: Plot of the truncated collection of NBA players, restricting consideration to those with 125 or more shots as the NBA does. The empirical shooting percentages of these players are considered valid for naming the best shooter(s) in the league.

$$y_i \sim Binom(p_i, k_i)$$

$$logit(p_i) \sim N(\mu_p, \sigma_p)$$

such that the likelihood to maximize is:

$$[y_i \mid p_i][p_i \mid \mu_p, \sigma_p]$$

where μ_p represents the overall league (among player) mean on a logit scale for the probability of making a free thrower, and σ_p represents the variability among players in the logit probability of making a free throw. This type of model is sometimes called a varying-intercept or random-intercept model. Because μ_p and σ_p determine the distribution of the parameter p, they are known as **hyperparameters**. This model approaches the previous model with no hyperparameters when σ_p approaches ∞. A similar strategy would be to place a beta prior directly on p_i, rather than placing a prior on $logit(p_i)$.

This model is hierarchical in the sense that we have a within player-level model (each player gets their own p_i) and an among-player model (with σ_p controlling the among-player variaiton). We will implement the logit-normal model in a maximum likelihood context to begin with, where we find maximum likelihood estimates for the hyperparameter σ_p.

```
library(lme4)
m2 <- glmer(cbind(ft_made, ft_miss) ~ (1|Player),
        family=binomial, data=d)
summary(m2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: cbind(ft_made, ft_miss) ~ (1 | Player)
##    Data: d
##
##      AIC      BIC   logLik deviance df.resid
##   3394.0   3402.3  -1695.0   3390.0      473
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.41341 -0.28396  0.01969  0.28122  1.70985
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  Player (Intercept) 0.2951   0.5432
## Number of obs: 475, groups:  Player, 475
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.07548    0.02891    37.2   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We used the `glmer` function in the `lme4` package to implement the above model. We obtain an estimate for $logit(\mu_p)$ with the "(Intercept)" parameter, and it is 0.7456375 on the probability scale. We also see a maximum likelihood estimate for σ_p under the random effects section: 0.5431875.

Let's visualize the new estimates:

```
# get estimated probabilities for each player from m2
shrunken_probs <- plogis(fixef(m2) + unlist(ranef(m2)))

# match these to the player names,
# from the row names of the ranef design matrix
shrunken_names <- m2@pp$Zt@Dimnames[[1]]

ranef_preds <- data.frame(Player = shrunken_names,
                          p_shrink = shrunken_probs)

# join the raw data with the model output
joined <- full_join(d, ranef_preds)
```

```
## Joining by: "Player"
```

```
# difference between naive & shrunken MLEs
joined$diff <- joined$ft_pct - joined$p_shrink

# plot naive MLE vs. shrunken MLE
ggplot(joined, aes(x=ft_pct, y=p_shrink, color=ft_shot)) +
  geom_point(shape=1) +
  scale_color_gradientn(colours=rainbow(4)) +
  geom_abline(yintercept=0, slope=1, linetype='dashed') +
  xlab('Naive MLE') +
  ylab('Shrunken MLE')
```

```
# using facets instead of colors
joined$ft_group <- cut(joined$ft_shot, 9)
ggplot(joined, aes(x=ft_pct, y=p_shrink)) +
  geom_abline(yintercept=0, slope=1, linetype='dashed', alpha=.7) +
  geom_point(shape=1, alpha=.5, size=1) +
  facet_wrap(~ ft_group) +
  xlab('Naive MLE') +
  ylab('Shrunken MLE')
```

```
# view difference in estimates as a function of freethrows shot
ggplot(joined, aes(x=ft_shot, y=diff)) +
  geom_point(shape=1) +
  xlab("Free throw attempts") +
  ylab("Naive MLE - Shrunken MLE")
```

The estimates from the hierarchical model differ from the MLE estimates obtained in our first model. In particular, the estimates that are imprecise (e.g., players with few attempted shots) are shrunken towards the grand mean. This **shrinkage** is highly desirable, and is consistent with the idea that we have increasing trust in estimates that are informed by more data.

What about the NBA's magic number of 125? Do we find that estimates are still undergoing shrinkage beyond this range, or are the empirical free throw percentages reliable if players have taken 125 or more shots?

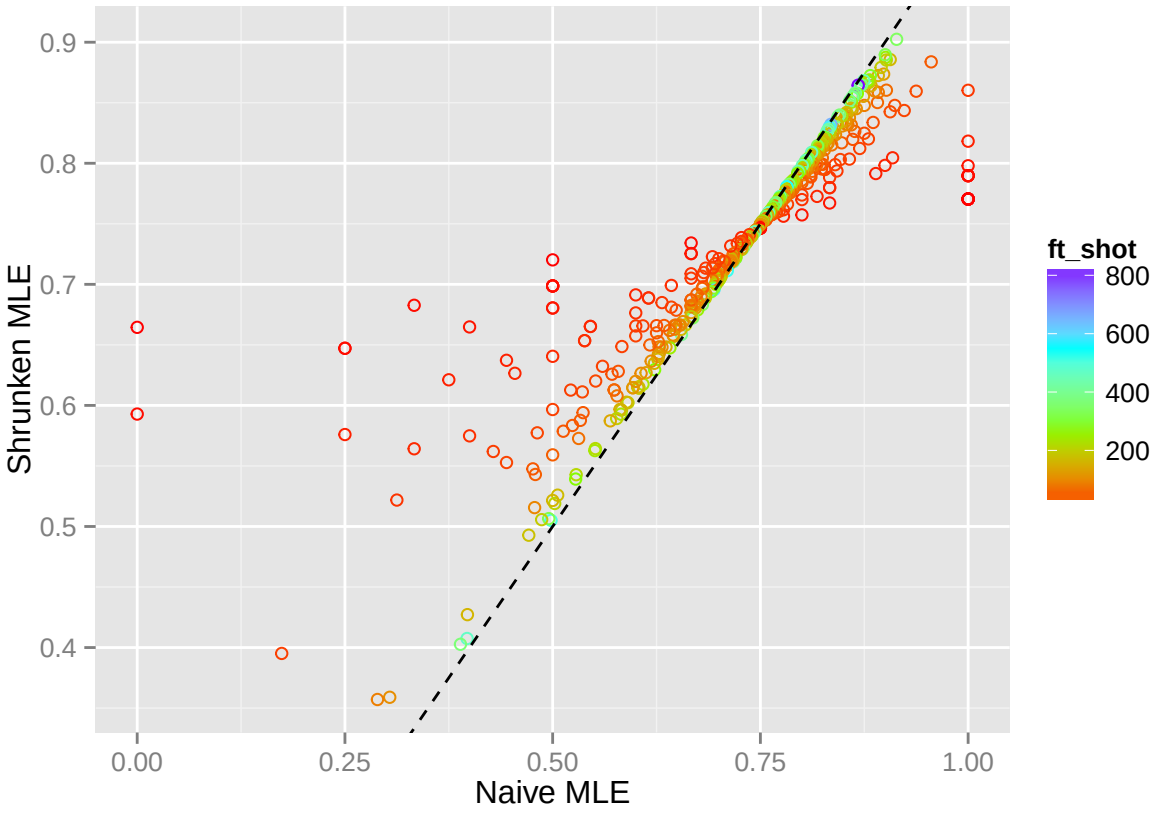


Figure 54: Plot of our shrunken estimates vs. the naive estimates. Notice that players who took fewer shots had more strongly shrunked estimates.

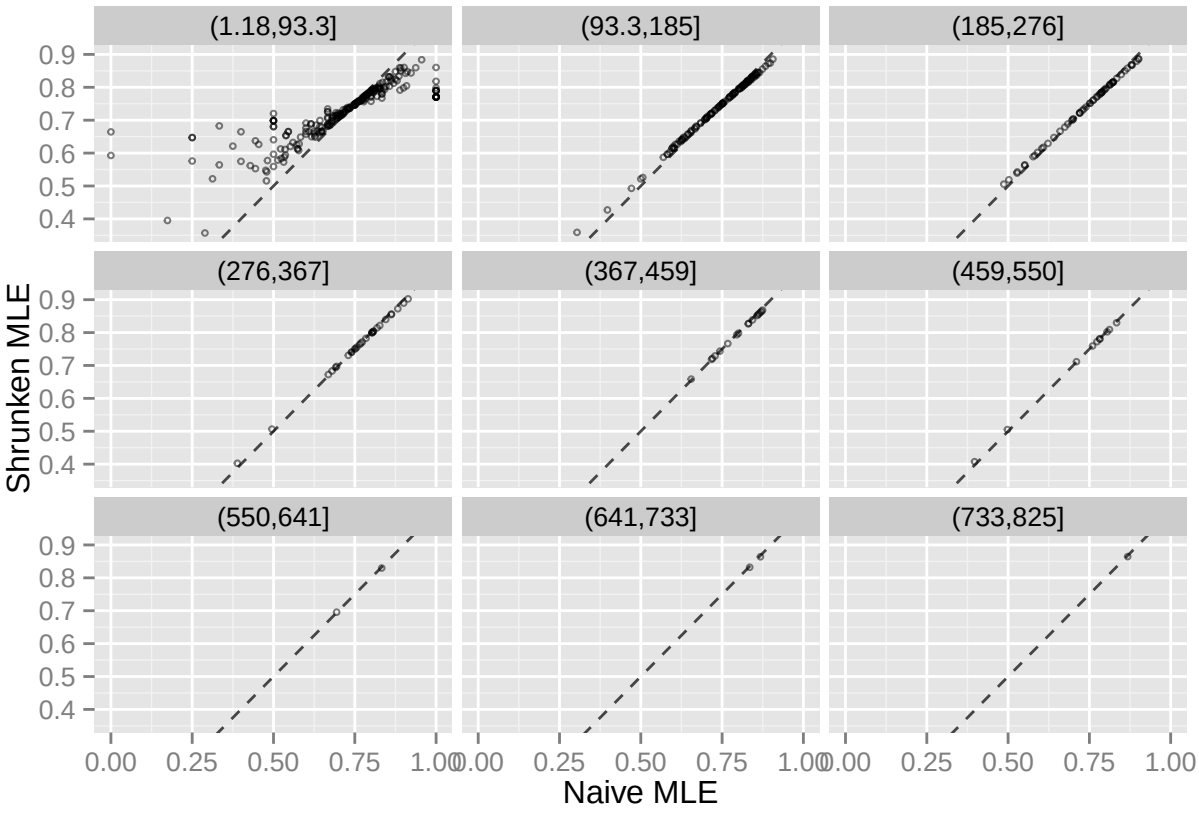


Figure 55: Plot of our shrunken estimates vs. the naive estimates. This time, instead of coloring the points to indicate how many shots a player took, this plot uses facets.

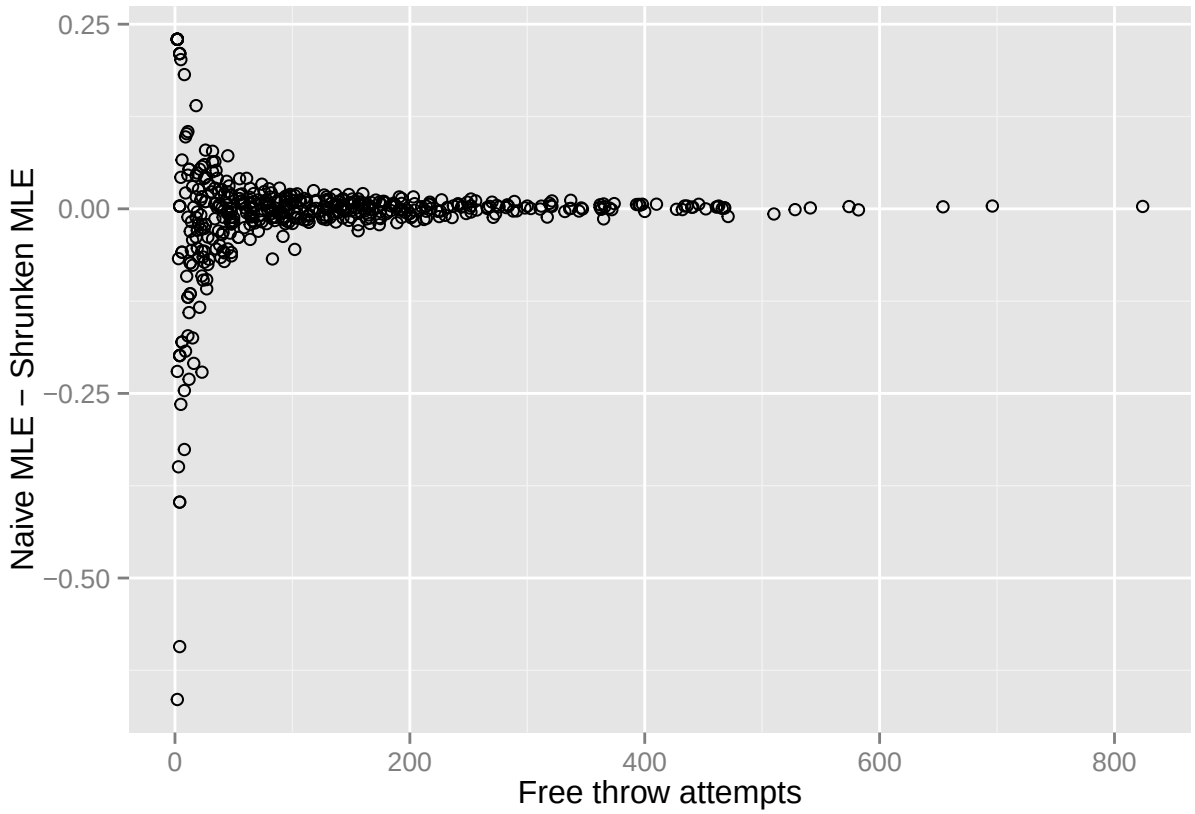


Figure 56: Another way to visualize shrinkage, this time as the difference between the naive and shrunken estimates as a function of sample size.

```
joined %>%
  filter(ft_shot >= 125) %>%
  ggplot(aes(x=ft_shot, y=diff)) +
  geom_point(shape=1) +
  xlab("Free throw attempts") +
  ylab("Naive MLE - Shrunken MLE")
```

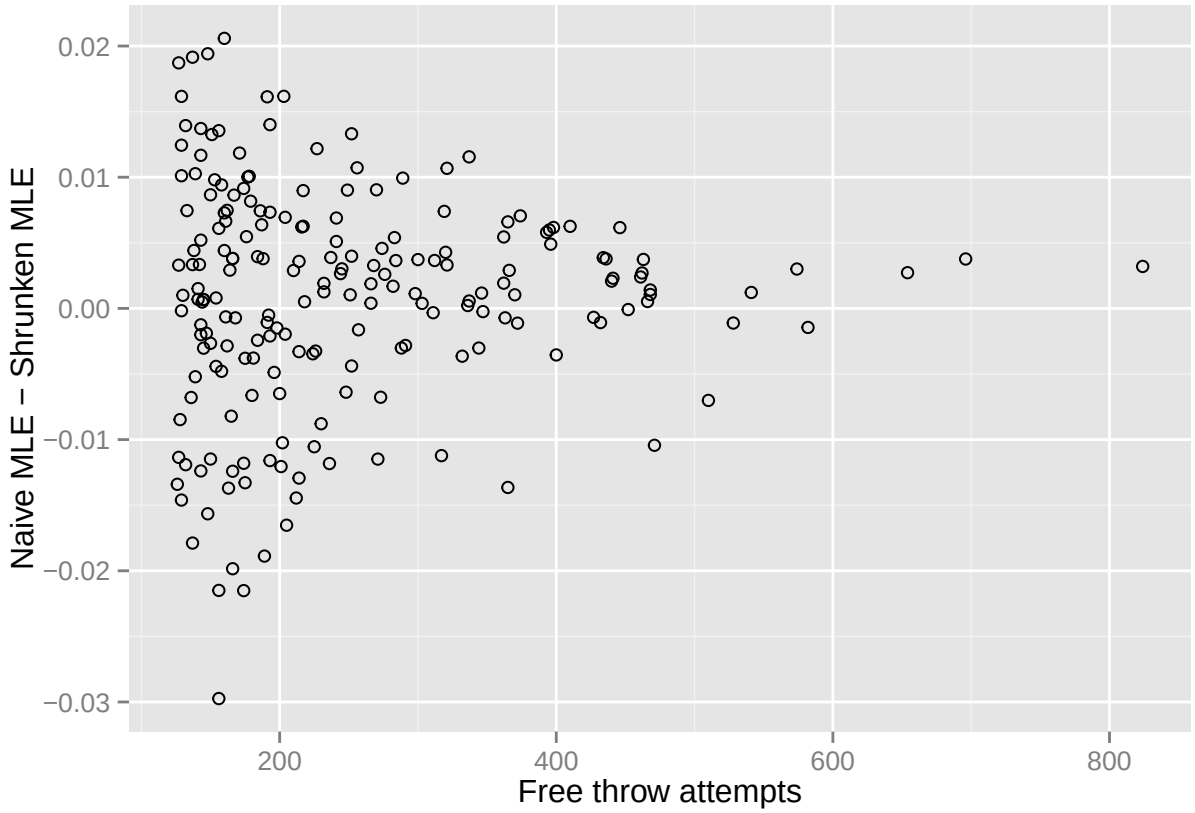


Figure 57: Subset of players eligible to be the best in the NBA based on empirical proportions. Notice that some of these estimates are still undergoing shrinkage, although to a lesser degree.

It looks like there are slight differences between the naive estimates (empirical proportions) and the probabilities that we estimate with partial pooling. We might conclude that the 125 shot threshold could give estimates that are reliable to plus or minus 2 or 3 percentage points based on the above graph.

So, which player do we think is the best and the worst?

```
joined[which.max(joined$p_shrink), ]
```

```
## Source: local data frame [1 x 9]
##
##          Player ft_made ft_miss ft_shot    ft_pct       shooter  p_shrink
##          (fctr)   (int)   (int)   (int)     (dbl)        (fctr)     (dbl)
## 1 Stephen Curry     308      29     337 0.9139466 Stephen Curry 0.9023947
## Variables not shown: diff (dbl), ft_group (fctr)
```

```
joined[which.min(joined$p_shrink), ]
```

```
## Source: local data frame [1 x 9]
##
##        Player ft_made ft_miss ft_shot    ft_pct     shooter  p_shrink
##        (fctr)   (int)   (int)   (int)     (dbl)      (fctr)     (dbl)
## 1 Joey Dorsey      24      59      83 0.2891566 Joey Dorsey 0.3570848
## Variables not shown: diff (dbl), ft_group (fctr)
```

Congrats Steph Curry (for this accomplishment and winning the NBA title), and our condolences to Joey Dorsey, who as of 2015 is playing in the Turkish Basketball League.

Partial, complete, and no pooling

Partial pooling is often presented as a compromise between complete pooling (in the above example, combining data from all players and estimating one NBA-wide p), and no pooling (using the observed proportion of free-throws made). This can be formalized by considering what happens to the parameter-level model in these three cases. With no pooling, the among-group (e.g., player) variance parameter approaches infinity, such that $p_i \sim N(\mu_p, \sigma_p) : \sigma_p \to \infty$. With complete pooling, the among-group variance parameter approaches zero, so that all groups recieve the group-level mean. With partial pooling, the estimation of σ_p leads to a data-informed amount of shrinkage.

Multiple levels, nestedness, and crossedness

In the previous example we had two levels: within and among players. Many hierarchical models have more than two levels. For instance, at the among-player level, we know that players played on different teams. Perhaps there are systematic differences among teams in free-throw shooting ability, for instance because they have a coaching staff that consistently emphasize free-throw shooting, or scouts that recruit players who are particularly good at shooting free-throws. We can expand the previous model to include team effects as follows:

$$y_i \sim Binom(p_i, k_i)$$

$$logit(p_i) = p_0 + \pi_i + \tau_{j[i]}$$

$$\pi_i \sim N(0, \sigma_\pi)$$

$$\tau_j \sim N(0, \sigma_\tau)$$

so that the likelihood to maximize is:

$$[y \mid p][p \mid p_0, \pi, \tau][\pi \mid \sigma_\pi][\tau \mid \sigma_\tau]$$

Here, p_0 is an intercept parameter that represents the mean logit probability of making a free throw across all players and teams. Player effects are represented by π_i, and team effects are represented by $\tau_{j[i]}$, with subscript indexing to represent that player i belongs to team j.

Note that not all players play for all teams, that is, players are not "crossed" with teams. Most players, specifically those that only played for one team, are nested within teams. However, because some players switched teams partway through the season, these players will show up on different teams. There is often a lot of confusion around nestedness in these types of models. We point out here that **nestedness is a feature of the data**, not a modeling decision. There are cases when the data are nested but structured poorly so that extra work is required to adequately represent the nestedness of the data. For instance, if we had data with measurements from 5 regions, each with 3 sites, and the sites were always labeled 1, 2, or 3, then our data might look like this:

```
##    region site
## 1       1    1
## 2       1    2
## 3       1    3
## 4       2    1
## 5       2    2
## 6       2    3
## 7       3    1
## 8       3    2
## 9       3    3
## 10      4    1
## 11      4    2
## 12      4    3
## 13      5    1
## 14      5    2
## 15      5    3
```

This would indicate that sites are crossed with region, with each site occurring in each region. But, this is misleading. The observations corresponding to site 1 are actually 5 different sites, occuring in 5 different regions. A more accurate data structure would be:

```
##    region site
## 1       1    1
## 2       1    2
## 3       1    3
## 4       2    4
## 5       2    5
## 6       2    6
## 7       3    7
## 8       3    8
## 9       3    9
## 10      4   10
## 11      4   11
## 12      4   12
## 13      5   13
## 14      5   14
## 15      5   15
```

This numbering scheme accurately captures the notion that each site occurs in only one region.

Fitting a 3 level model with `lme4`

Returning to the example with player and team effects, both drawn from a prior distribution with hyperparameters σ_π and σ_τ to be estimated:

```
d <- rawd %>%
  group_by(Player, Tm, Age, Pos) %>%
  summarize(ft_made = sum(FT),
            ft_miss = sum(FTA) - sum(FT),
            ft_shot = sum(FTA),
            ft_pct = sum(FT) / sum(FTA)) %>%
  subset(ft_shot != 0) %>%
  arrange(-ft_pct) %>%
  droplevels()
d
```

```
## Source: local data frame [626 x 8]
## Groups: Player, Tm, Age [626]
##
##           Player     Tm   Age    Pos ft_made ft_miss ft_shot    ft_pct
##           (fctr) (fctr) (int) (fctr)   (int)   (int)   (int)     (dbl)
## 1   Aaron Brooks    CHI    30     PG     145      29     174 0.8333333
## 2   Aaron Gordon    ORL    19     PF      44      17      61 0.7213115
## 3  Adreian Payne    ATL    23     PF       1       1       2 0.5000000
## 4  Adreian Payne    MIN    23     PF      29      15      44 0.6590909
## 5  Adreian Payne    TOT    23     PF      30      16      46 0.6521739
## 6     A.J. Price    CLE    28     PG       4       2       6 0.6666667
## 7     A.J. Price    IND    28     PG      12       6      18 0.6666667
## 8     A.J. Price    TOT    28     PG      16       8      24 0.6666667
## 9  Alan Anderson    BRK    32     SG      82      19     101 0.8118812
## 10    Alec Burks    UTA    23     SG     106      23     129 0.8217054
## ..           ...    ...   ...    ...     ...     ...     ...       ...
```

```
m <- glmer(cbind(ft_made, ft_miss) ~ (1|Player) + (1|Tm),
        family=binomial, data=d)
summary(m)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: cbind(ft_made, ft_miss) ~ (1 | Player) + (1 | Tm)
##    Data: d
##
##      AIC      BIC   logLik deviance df.resid
##   3899.1   3912.5  -1946.6   3893.1      623
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.17015 -0.30554  0.01832  0.30808  1.91060
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  Player (Intercept) 0.2951   0.5432
##  Tm     (Intercept) 0.0000   0.0000
## Number of obs: 626, groups:  Player, 475; Tm, 31
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept)  1.07548    0.02891    37.2   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
pr <- profile(m)
confint(pr)
```

```
##                 2.5 %     97.5 %
## .sig01      0.5002122 0.59097299
## .sig02      0.0000000 0.06070247
## (Intercept) 1.0187033 1.13228412
```

So, it appears that the MLE for the variance attributable to teams is zero. Profiling the likelihood, we see that there may be a little bit of variance attributable to teams, but we may be better off discarding the team information and instead including information about the player's positions. The logic here is that we expect players to vary in their shooting percentages based on whether they are guards, forwards, centers, etc.

```
m2 <- glmer(cbind(ft_made, ft_miss) ~ (1|Player) + (1|Pos),
            family=binomial, data=d)
summary(m2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: cbind(ft_made, ft_miss) ~ (1 | Player) + (1 | Pos)
##    Data: d
##
##      AIC      BIC   logLik deviance df.resid
##   3851.9   3865.2  -1923.0   3845.9      623
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.16535 -0.32463  0.02557  0.32095  1.87141
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  Player (Intercept) 0.23943  0.4893
##  Pos    (Intercept) 0.03869  0.1967
## Number of obs: 626, groups:  Player, 475; Pos, 11
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.09854    0.08041   13.66   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
AIC(m, m2)
```

```
##    df      AIC
## m   3 3899.147
## m2  3 3851.925
```

So, m2 seems to be better in terms of AIC, but one could argue that m2 will also be more useful for predictive applications. For instance, if we wanted to predict the free throw shooting ability of a new player, we could get a more precise estimate with m2 because we could use information about their position (if it was known). In contrast, in model m, we would predict the same p regardless of position, because position is not in the model.

Level-specific covariates

In hierarchical models, covariates can be included at specific levels. For instance, at the player level, we might expect that age has an impact on free throw shooting ability. Possibly, players peak at some age, and then start to go downhill as they approach retirement. We can represent this with a second degree polynomial effect of age. Trying this:

```
m3 <- glmer(cbind(ft_made, ft_miss) ~ Age + I(Age^2) + (1|Player) + (1|Pos),
            family=binomial, data=d)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control
## $checkConv, : Model failed to converge with max|grad| = 0.0280989 (tol =
## 0.001, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, : Model is nearly unide
##  - Rescale variables?;Model is nearly unidentifiable: large eigenvalue ratio
##  - Rescale variables?
```

```
summary(m3)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: cbind(ft_made, ft_miss) ~ Age + I(Age^2) + (1 | Player) + (1 |
##     Pos)
##    Data: d
##
##      AIC      BIC   logLik deviance df.resid
##   3838.1   3860.3  -1914.0   3828.1      621
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -1.99299 -0.33689  0.02006  0.33940  1.88029
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  Player (Intercept) 0.22661  0.4760
##  Pos    (Intercept) 0.04262  0.2065
## Number of obs: 626, groups:  Player, 475; Pos, 11
##
## Fixed effects:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.956747   0.952742  -2.054  0.03999 *
## Age          0.206198   0.069517   2.966  0.00302 **
## I(Age^2)    -0.003347   0.001252  -2.674  0.00749 **
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##          (Intr) Age
## Age      -0.993
## I(Age^2)  0.981 -0.996
## convergence code: 0
## Model failed to converge with max|grad| = 0.0280989 (tol = 0.001, component 1)
## Model is nearly unidentifiable: very large eigenvalue
##  - Rescale variables?
## Model is nearly unidentifiable: large eigenvalue ratio
##  - Rescale variables?
```

We get a warning about convergence resulting from numeric issues, with the recommendation to rescale our variables. Recall that unscaled continuous covariates tend to cause correlations between the estimates of slopes and intercepts, which is apparent in the Correlation of Fixed Effects section. Rescaling age does the trick in this example:

```
d$age <- (d$Age - mean(d$Age)) / sd(d$Age)
m3 <- glmer(cbind(ft_made, ft_miss) ~ age + I(age^2) + (1|Player) + (1|Pos),
            family=binomial, data=d)
summary(m3)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: cbind(ft_made, ft_miss) ~ age + I(age^2) + (1 | Player) + (1 |
##     Pos)
##    Data: d
##
##      AIC      BIC   logLik deviance df.resid
##   3838.1   3860.3  -1914.0   3828.1      621
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -1.99300 -0.33690  0.02006  0.33940  1.88029
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  Player (Intercept) 0.22661  0.4760
##  Pos    (Intercept) 0.04263  0.2065
## Number of obs: 626, groups:  Player, 475; Pos, 11
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.15916    0.08597  13.484  < 2e-16 ***
## age          0.11720    0.02843   4.122 3.76e-05 ***
## I(age^2)    -0.05722    0.02143  -2.671  0.00757 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##          (Intr) age
```

```
## age       0.109
## I(age^2) -0.251 -0.396
```

```
AIC(m, m2, m3)
```

```
##    df      AIC
## m   3 3899.147
## m2  3 3851.925
## m3  5 3838.078
```

This model does seem to receive more support than the other two models. We can visualise the age result as follows:

```
lo <- 100
new_age <- seq(min(d$age), max(d$age), length.out=lo)
X <- matrix(c(rep(1, 100), new_age, new_age^2), nrow=lo)
logit_p <- X %*% fixef(m3)
scaled_age <- new_age * sd(d$Age) + mean(d$Age)
plot(scaled_age, plogis(logit_p), type='l',
     xlab="Age", ylab="p")
```

Figure 58: Plot showing the quadratic effect of age on the logit probability of making a freethrow.

There are also packages that are designed to aid in visualization of the output of lmer and glmer objects. Handy plots here include a sorted caterpillar plot of the random effects:

```
library(sjPlot)
library(arm)
sjp.glmer(m3, ri.nr = 1, sort = "(Intercept)")
```

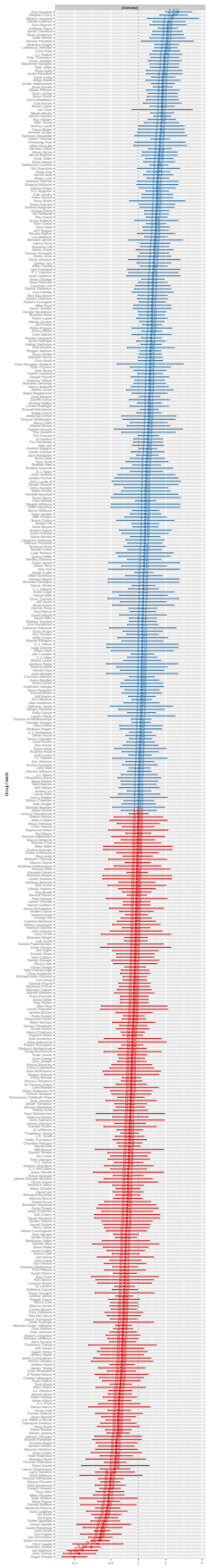


Figure 59: Random effects and confidence intervals as generated by the `sjPlot` package. We note here that these confidence intervals are probably unreliable due to restrictive assumptions.

```
sjp.glmer(m3, ri.nr=2, sort = "(Intercept)")
```

```
## Plotting random effects...
```

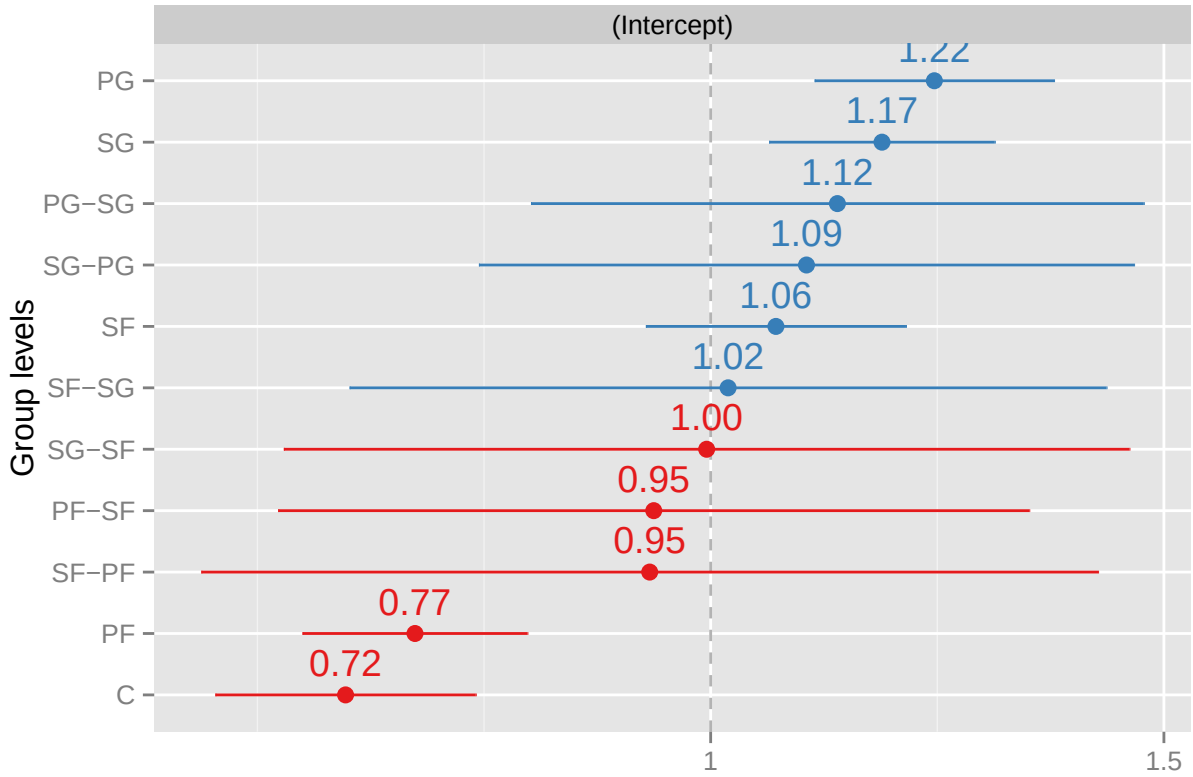


Figure 60: Player position random effects along with Wald confidence intervals.

These confidence intervals should be taken with a grain of salt, as they are calculated based on a normal approximation with Wald standard errors, which assume a quadratic log-likelihood profile. Later, we will get a more reliable estimate of the error for random effects using Bayesian methods.

We assumed that the random effects are normally distributed on a logit scale. This assumption can be checked with a q-q plot:

```
sjp.glmer(m3, type = "re.qq", facet.grid=F)
```

```
## Testing for normal distribution. Dots should be plotted along the line.
```

Last, we should do a sanity check for our estimated probabilities. One approach is to visually check the estimated probabilities vs. the naive empirical proportions.

```
d$estimated_p <- fitted(m3)
d$diff <- d$ft_pct - d$estimated_p
ggplot(d, aes(x=ft_shot, y=diff)) +
  geom_point(shape=1) +
  xlab("Free throw attempts") +
  ylab("Naive MLE - Shrunken MLE")
```

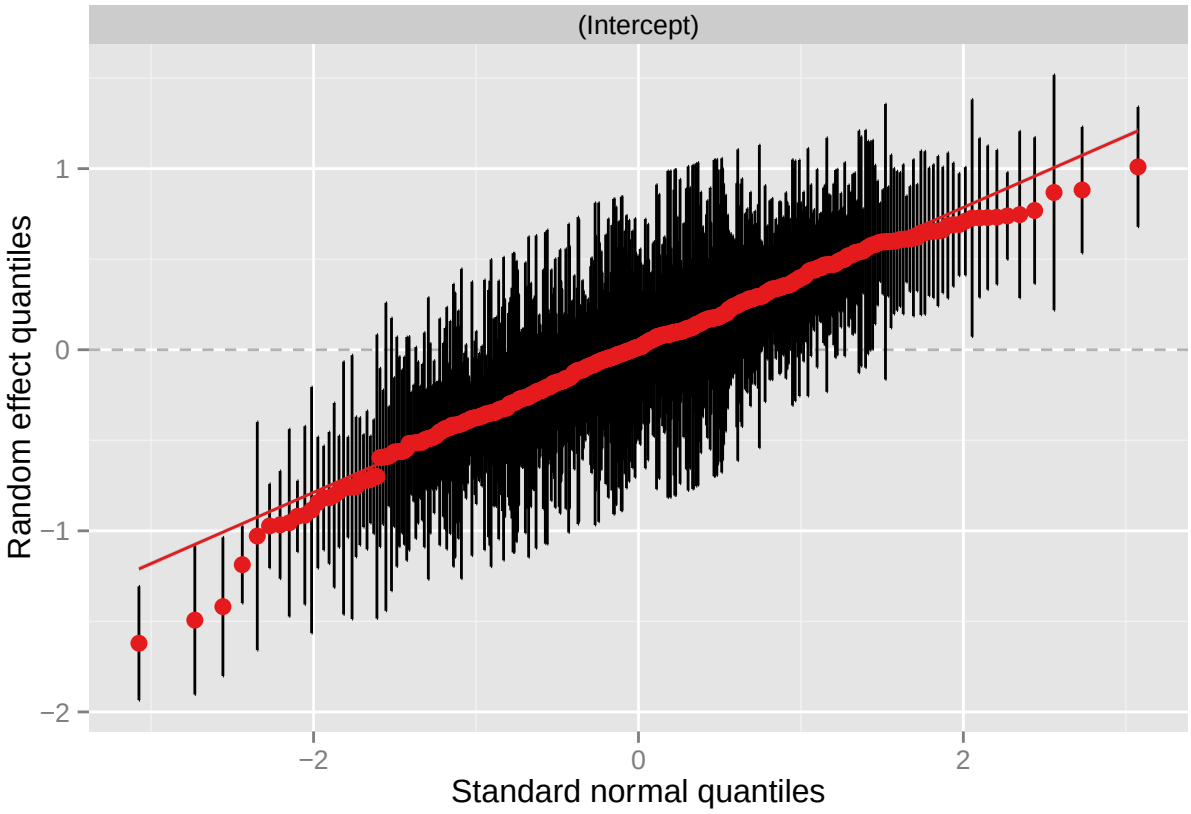


Figure 61: Q-Q plot for the random effects, which are assumed to be distributed normally.

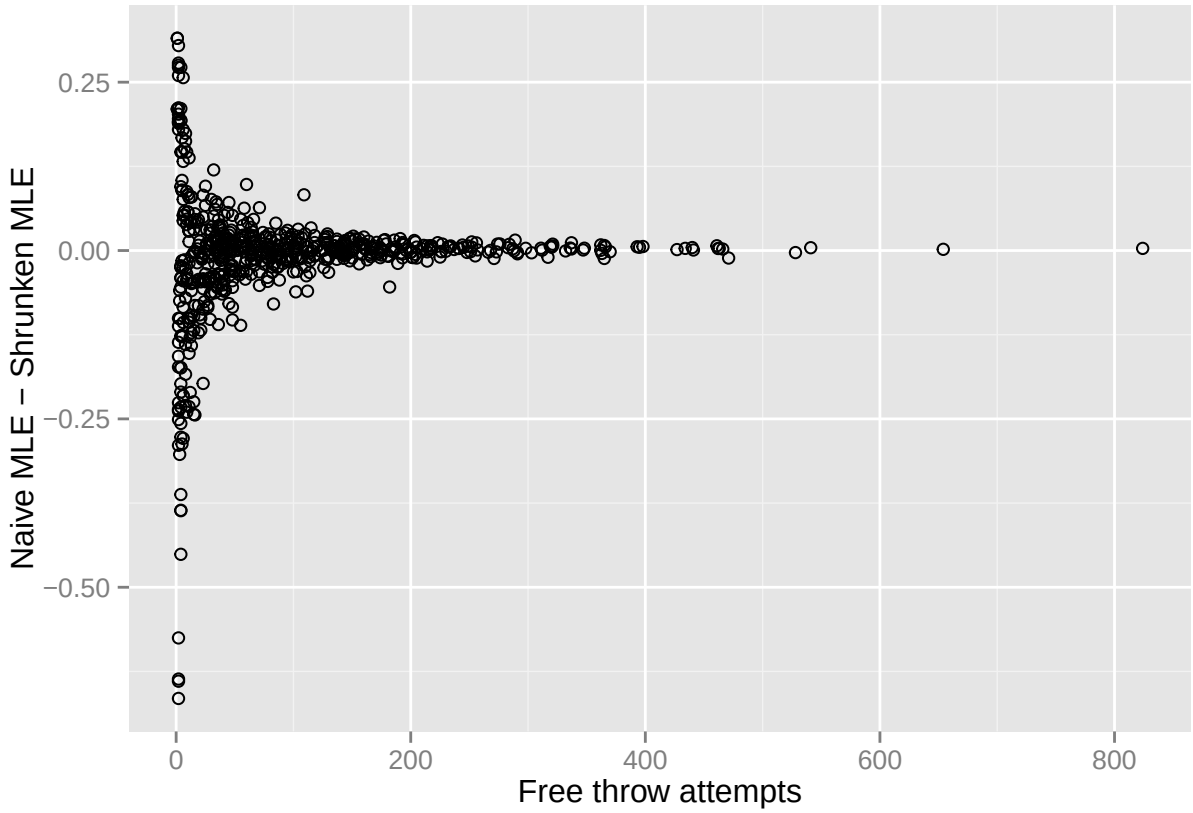


Figure 62: Shrinkage plot with the updated model.

```
ggplot(d) +
  geom_segment(aes(x="Naive MLE", xend="Shrunken MLE",
                   y=ft_pct, yend=estimated_p, group=Player),
               alpha=.3) +
  xlab('Estimate') +
  ylab('Pr(make freethrow)')
```

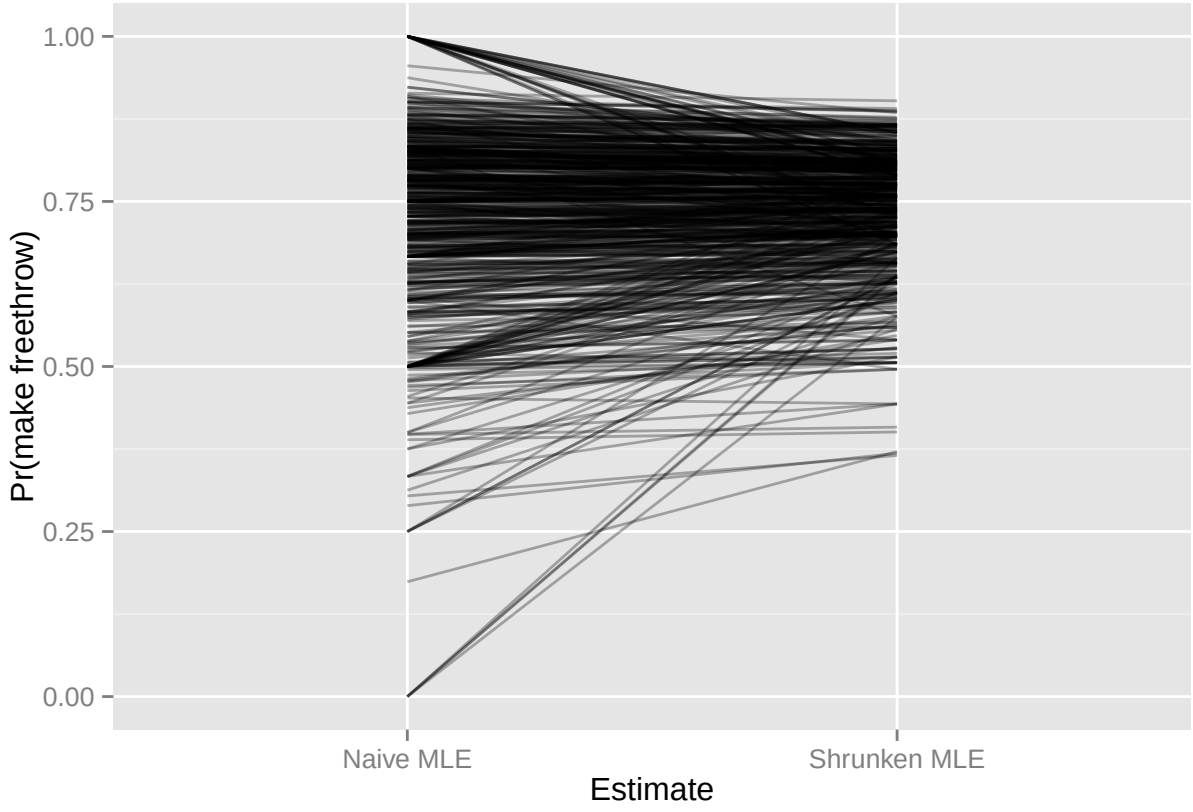


Figure 63: An alternative shrinkage plot to compare naive and shrunken estimates.

```
ggplot(d, aes(x=ft_shot, y=estimated_p)) +
  geom_point(shape=1) +
  xlab("Free throw attempts") +
  ylab("Shrunken MLE")
```

Who does this model identify as the worst?

```
d[which.min(d$estimated_p),
  c("Player", "Age", "Pos", "ft_shot", "ft_pct", "estimated_p")]
```

```
## Source: local data frame [1 x 6]
##
##         Player   Age    Pos ft_shot    ft_pct estimated_p
##        (fctr) (int) (fctr)   (int)     (dbl)       (dbl)
## 1 Ian Mahinmi    28      C     102 0.3039216   0.3652981
```

Which player might be best?

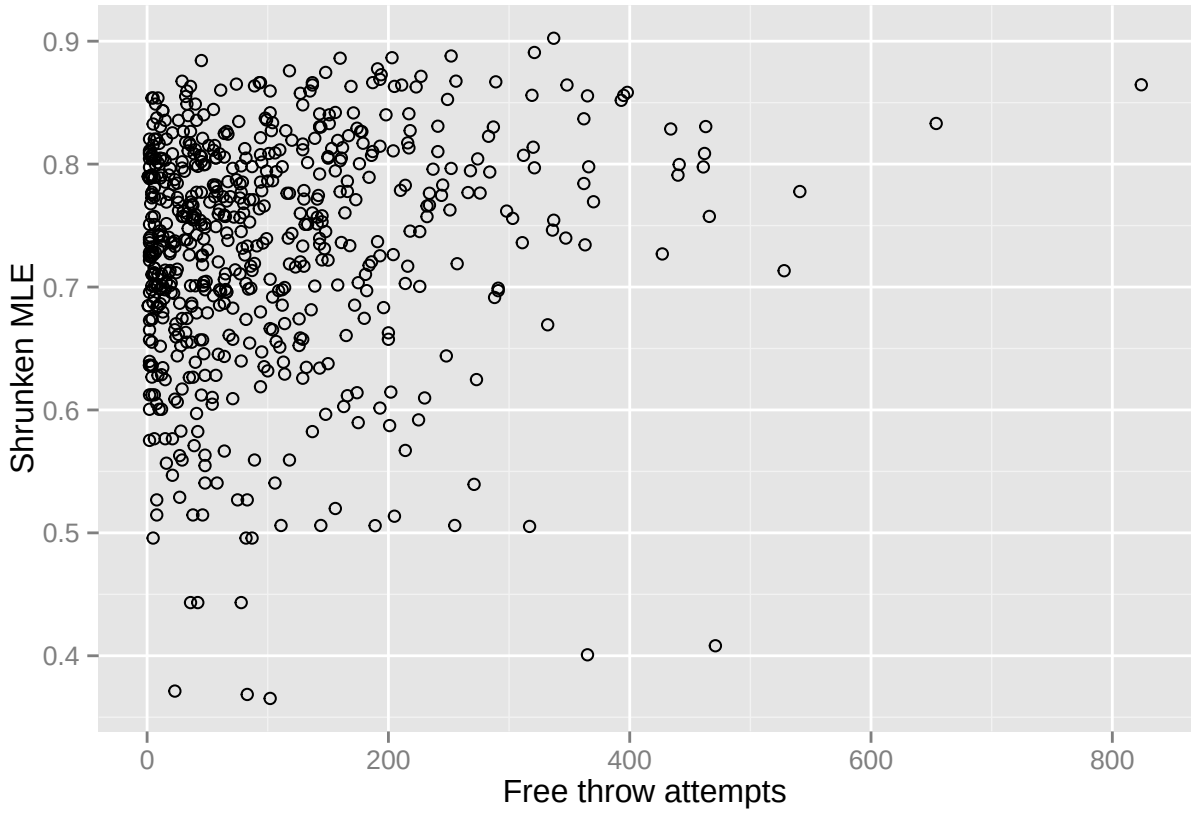


Figure 64: Final sanity check to ensure that the estimated probabilities from the final model are reasonable.

```
d[which.max(d$estimated_p),
  c("Player", "Age", "Pos", "ft_shot","ft_pct", "estimated_p")]
```

```
## Source: local data frame [1 x 6]
##
##          Player   Age    Pos ft_shot    ft_pct estimated_p
##          (fctr) (int) (fctr)   (int)     (dbl)       (dbl)
## 1 Stephen Curry    26     PG     337 0.9139466   0.9023958
```

Further reading

Gelman and Hill. 2009. *Data analysis using regression and multilevel/hierarchical models.* Chapter 11, 12.

Gelman et al. 2014. *Bayesian data analysis, 3rd edition.* Chapter 5.

Efron, Bradley, and Carl N. Morris. Stein's paradox in statistics. WH Freeman, 1977.

Gelman, Andrew, Jennifer Hill, and Masanao Yajima. "Why we (usually) don't have to worry about multiple comparisons." Journal of Research on Educational Effectiveness 5.2 (2012): 189-211.

Chapter 7: Bayesian hierarchical models

Big picture

Everything that we've done so far in this course has laid a foundation to understand the main course: Bayesian hierarchical models.

Learning goals

- varying intercepts (NBA freethrow example) with `Stan`
- Bayesian vs. MLE approaches
- binomial-Poisson hierarchy (e.g. # eggs laid & survival)
- non-centered parameterizations
- multivariate normal distribution
- priors for hierarchical variance parameters
- prediction (new vs. observed groups)
- connections to random, fixed, & mixed effects

Bayesian hierarchical models

The main difference between the hierarchical models of the previous chapter and Bayesian hierarchical models is the inclusion of a prior distribution on the hyperparameters (leading to each parameter being represented as a random variable with a probability distribution, consistent with the Bayesian philosophy). The non-Bayesian hierarchical models of the previous chapter are semi-Bayesian in that they induce a prior distribution on some but not all parameters. Fully Bayesian hierarchical models incorporate uncertainty in the hyperparameters, which is typically considerable. Bayesian approaches tend to be much easier than frequentist or MLE approaches in terms of estimation, interval construction, and prediction for all but the simplest models.

Bayesian hierarchical models are an incredibly powerful and flexible tool for learning about the world. Their usefulness is derived from the ability to combine simple model components to make models that are sufficient to represent even complex processes. For instance, consider the following example:

Binomial-Poisson hierarchy

Suppose you study birds, and you'd like to estimate fitness as measured by egg output and egg survival to fledging. There are two response variables and they are connected. On the one hand, you might be interested in a model that uses the number of eggs laid by individual i as a response:

$$y_i \sim Poisson(\lambda)$$

where λ is the expected number of eggs laid.

Further, the number of surviving fledglings is a function of Y_i. If each egg survives indepdently (or with the addtion of covariates, conditionally independently) with probability p, then the number of fledglings Z_i can be considered a binomial random variable:

$$z_i \sim Binomial(y_i, p)$$

The posterior distribution of the parameters can be written as:

$$[p, \lambda \mid y, z] = [z \mid y, p][y \mid \lambda][\lambda, p]$$

This is a complicated model that arises from fairly simple parts. We haven't yet specified the form of our prior distribution for λ and p. The simplest prior might be something that assumes that all individuals $i = 1, ..., n$ have the same values. For instance, suppose that we expected ahead of time that each bird would lay up to 7 eggs, with an expected value of about 2.5. We might then choose a gamma prior for λ that is consistent with these expectations:

```
x <- seq(0, 20, .01)
dx <- dgamma(x, 2.5, scale=1.1)
plot(x, dx, type='l',
     xlab=expression(lambda),
     ylab=expression(paste("[", lambda, "]")))
```

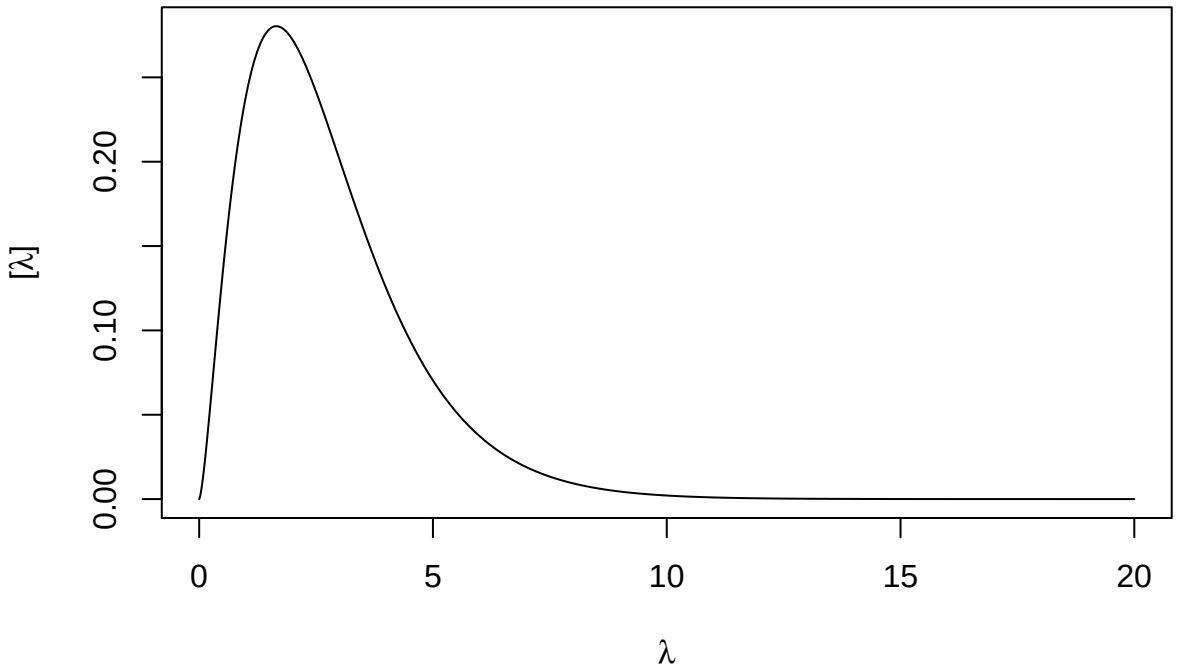


Figure 65: Probability density of the Gamma prior on lambda.

```
# What is the 97.5% quantile?
qgamma(.975, 2.5, scale=1.1)
```

```
## [1] 7.057876
```

If we thought that mean survival probability was about .375, but as low as zero and as high as .8, we could choose a beta prior for p

```
x <- seq(0, 1, .01)
dx <- dbeta(x, 3, 5)
plot(x, dx, type='l', xlab='p', ylab='[p]')
```

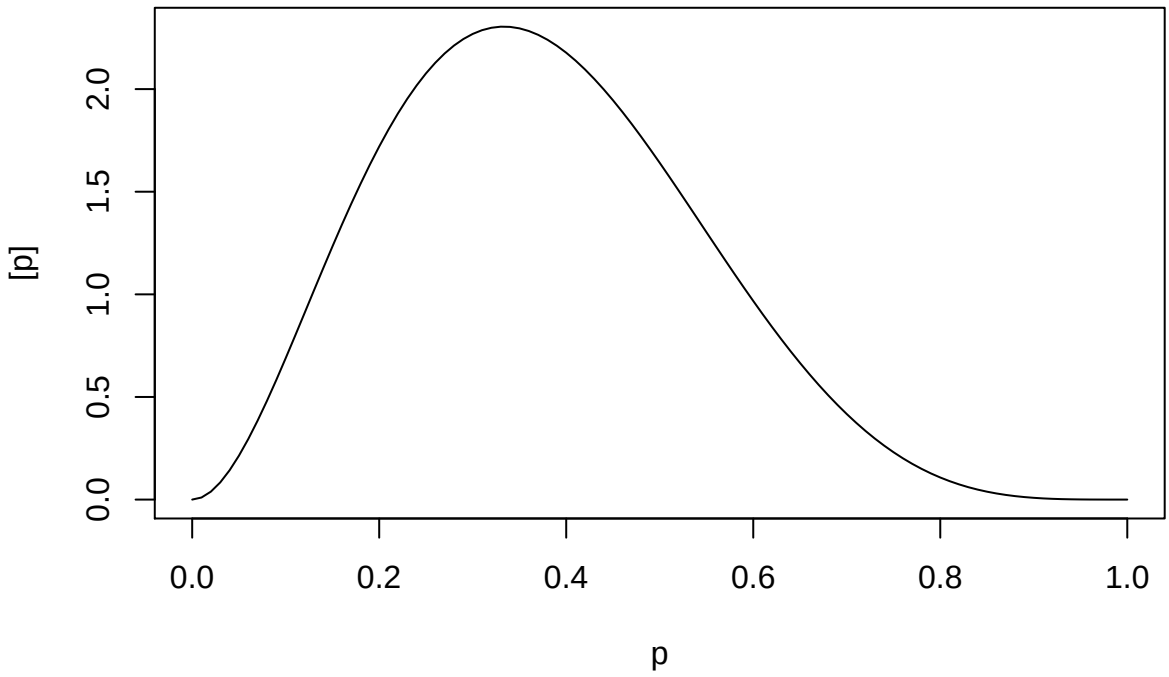


Figure 66: Probability density of the beta prior on p.

```
# check to see the CDF of the beta prior at 0.8
pbeta(.8, 3, 5)
```

```
## [1] 0.995328
```

These priors would complete our specification of our Bayesian hierarchical model.

$$y_i \sim Poisson(\lambda)$$

$$z_i \sim Binomial(y_i, p)$$

$$\lambda \sim gamma(2.5, 1.1)$$

$$p \sim beta(3, 5)$$

Suppose we have data from 100 females, and we'd like to update our priors given this new information. First, visualizing the data a bit:

```
library(dplyr)
library(ggplot2)
d <- read.csv('eggs.csv')

# calculate proportion of eggs surviving
d <- d %>%
  mutate(p_survive = fledglings / eggs)
tbl_df(d)
```

```
## Source: local data frame [200 x 5]
##
##     eggs fledglings bird_mass population p_survive
##    (int)      (int)     (dbl)     (fctr)     (dbl)
## 1      1          0 0.2278972          A      0.00
## 2      2          1 0.7182951          A      0.50
## 3      2          2 1.4752230          A      1.00
## 4      3          3 0.7623340          A      1.00
## 5      4          3 2.7034917          A      0.75
## 6      4          3 3.7814117          A      0.75
## 7      2          1 8.4822438          A      0.50
## 8      0          0 0.1786772          A       NaN
## 9      3          0 4.6517436          A      0.00
## 10     0          0 4.9095815          A       NaN
## ..   ...        ...       ...        ...       ...
```

```
ggplot(d, aes(x=eggs)) +
  geom_histogram()
```

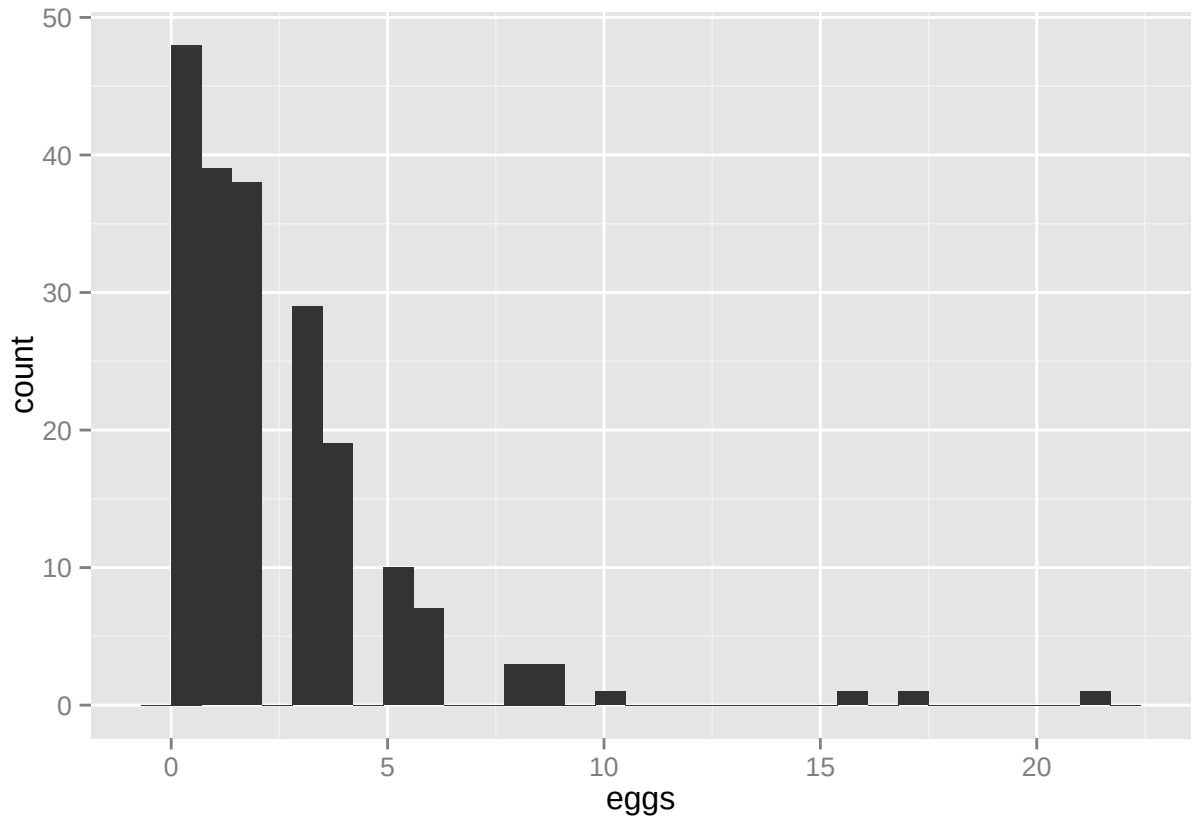


Figure 67: Empirical distribution of number of eggs laid by each bird.

```
ggplot(d, aes(x=p_survive)) +
  geom_histogram()
```

```
## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```

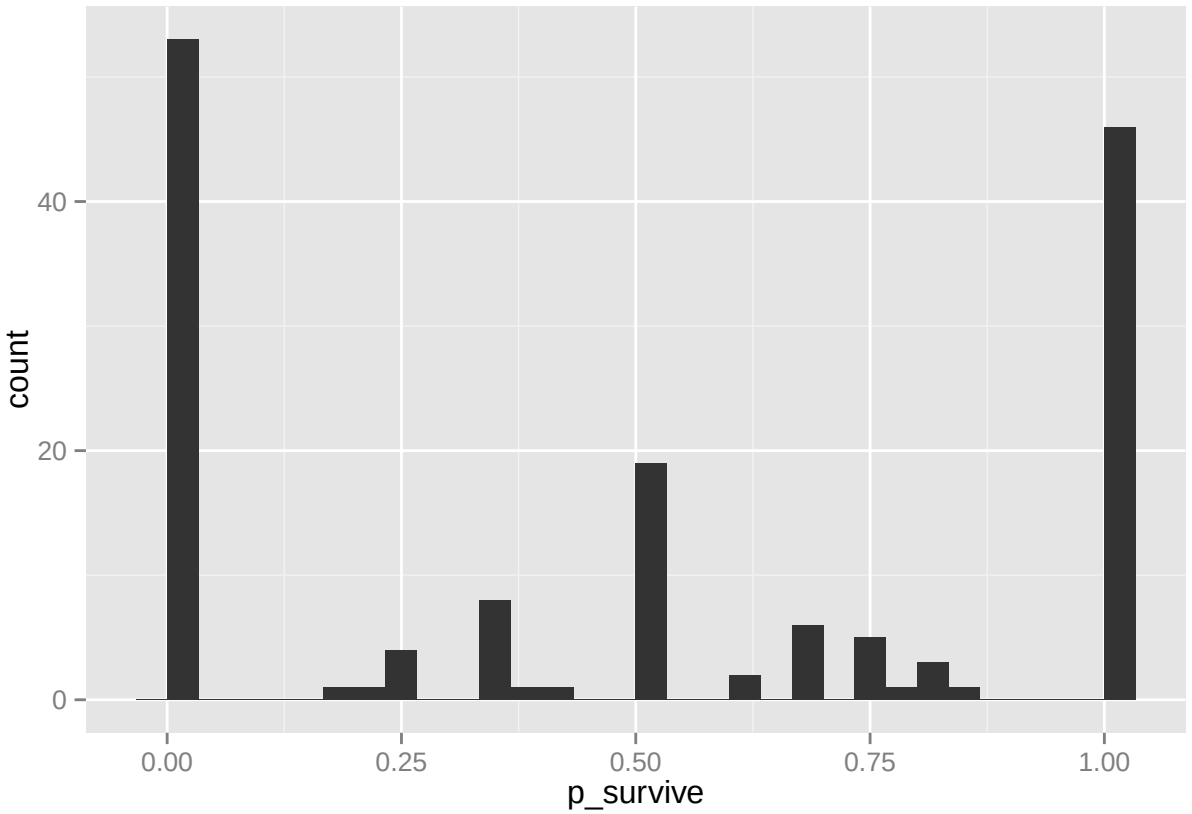


Figure 68: Empirical distribution of the proportion of eggs surviving to fledging for each bird.

Notice that for birds that did not lay eggs, we have NA values in the proportion of eggs surviving. As a result, these individuals will not provide information on p in the above model, because they cannot contribute to a binomial likelihood with $k = 0$.

We can translate the model outlined above to Stan as follows to produce the file `eggs.stan`:

```
data {
  // poisson data
  int n;
  int y[n];

  // binomial data
  int n_b;
  int k[n_b];
  int z[n_b];
}

parameters {
  real<lower=0> lambda;
  real<lower=0, upper=1> p;
}

model {
  // priors
  lambda ~ gamma(2.5, 1.1);
  p ~ beta(3, 5);

  // likelihood
  y ~ poisson(lambda);
  z ~ binomial(k, p);
}
```

Now we can bundle our data to work with the model and estimate the parameters:

```
stan_d <- list(
  n = nrow(d),
  y = d$eggs,
  n_b = sum(d$eggs > 0),
  k = d$eggs[d$eggs > 0],
  z = d$fledglings[d$eggs > 0]
)

library(rstan)
m <- stan('eggs.stan', data=stan_d)
```

```
m
```

```
## Inference for Stan model: eggs.
## 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
## lambda    2.42    0.00 0.10    2.22    2.35    2.42    2.49    2.63  2731
```

```
## p          0.62    0.00 0.02    0.57    0.60    0.62    0.63    0.66  2385
## lp__    -384.48    0.03 0.93 -386.87 -384.86 -384.19 -383.82 -383.56  1371
##         Rhat
## lambda     1
## p          1
## lp__       1
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:44:28 2015.
## 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).
```

```
traceplot(m)
```

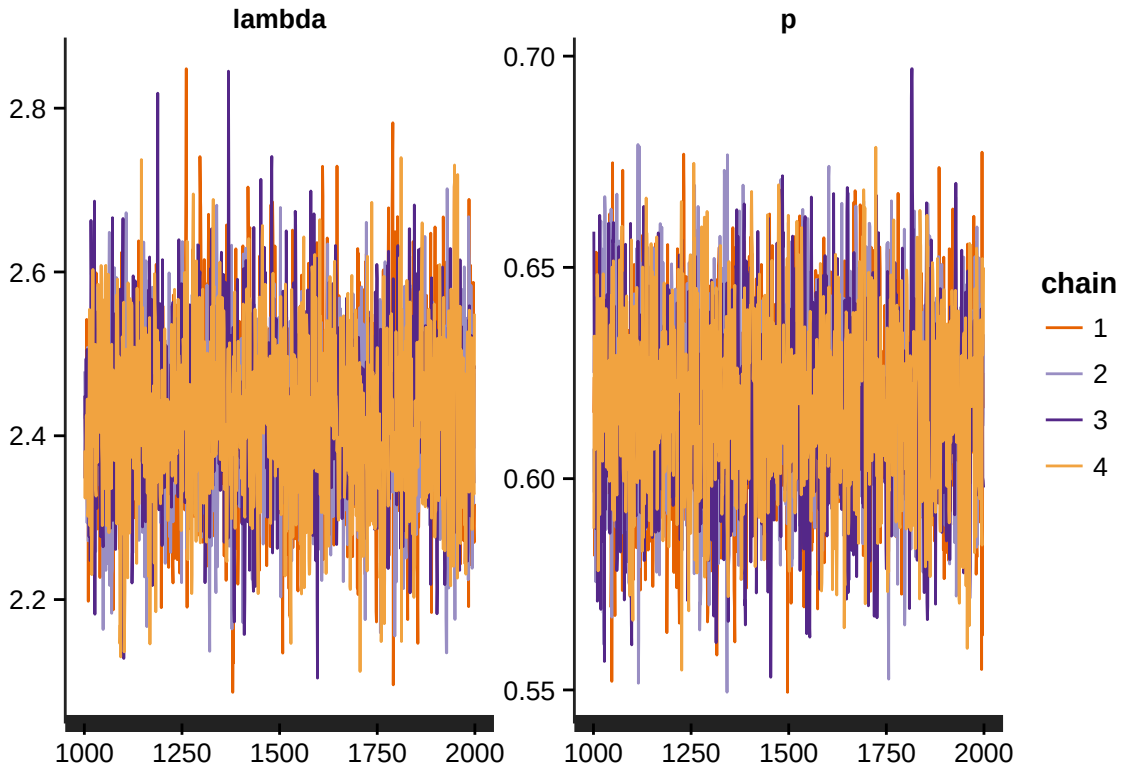


Figure 69: Traceplot of the Markov chains for the `egg.stan` model.

Non-centered parameterizations for random effects

The previous model is fairly unsatisfactory in that it assumes that all individuals have the same fecundity and egg to fledgling survival. Expecting that this assumption is probably false, we may wish to allow individuals to vary in these quantities. One way to do this would be with a normal (on the link scale) random effect, exactly like log-normal overdispersion and logit-normal overdispersion for the Poisson and binomial examples covered earlier. For instance, we might write such a model as:

$$y_i \sim Poisson(\lambda_i)$$

$$z_i \sim Binomial(y_i, p_i)$$

$$log(\lambda_i) \sim N(\mu_\lambda, \sigma_\lambda)$$

$$logit(p_i) \sim N(\mu_p, \sigma_p)$$

$$\mu_\lambda \sim N(0, 1)$$

$$\sigma_\lambda \sim halfNormal(0, 2)$$

$$\mu_p \sim N(0, 2)$$

$$\sigma_p \sim halfNormal(0, 1.5)$$

For the purposes of illustration, we've provided somewhat vague priors, but one could adapt these to reflect the priors that we expressed in the simpler model. Much has been written on how to choose priors for hierarchical variance parameters. The main take home is to avoid hard upper limits and instead to use priors that reflect your previous beliefs with soft constraints, such as half-Cauchy, half-t, or half-normal. See the further reading section for a good reference on selecting good priors for these hyperparameters. In this model, the log fecundity and logit survival probabilities are drawn from independent normal distributions, allowing for individual variation around the population means.

Updating our model and calling it `egg_ranef.stan`, we might get:

```
data {
  // poisson data
  int n;
  int y[n];

  // binomial data
  int n_b;
  int k[n_b];
  int z[n_b];
}

parameters {
  vector[n] log_lambda;
  vector[n_b] logit_p;
  real mu_lambda;
  real mu_p;
  real<lower=0> sigma_lambda;
  real<lower=0> sigma_p;
}

model {
  // priors
  mu_lambda ~ normal(0, 1);
  mu_p ~ normal(0, 2);
```

```
  sigma_lambda ~ normal(0, 2);
  sigma_p ~ normal(0, 1.5);

  log_lambda ~ normal(mu_lambda, sigma_lambda);
  logit_p ~ normal(mu_p, sigma_p);

  // likelihood
  y ~ poisson_log(log_lambda);
  z ~ binomial_logit(k, logit_p);
}
```

Fitting the new model:

```
m <- stan('egg_ranef.stan', data=stan_d,
          pars=c('mu_lambda', 'mu_p', 'sigma_lambda', 'sigma_p', 'lp__'))
```

```
## The following numerical problems occured the indicated number of times after warmup on chain 1
##                                                                                       count
## Exception thrown at line 28: normal_log: Scale parameter is 0, but must be > 0!     1
## When a numerical problem occurs, the Metropolis proposal gets rejected.
## However, by design Metropolis proposals sometimes get rejected even when there are no numerical prob
## Thus, if the number in the 'count' column is small, do not ask about this message on stan-users.
## The following numerical problems occured the indicated number of times after warmup on chain 3
##                                                                                       count
## Exception thrown at line 28: normal_log: Scale parameter is 0, but must be > 0!     1
## When a numerical problem occurs, the Metropolis proposal gets rejected.
## However, by design Metropolis proposals sometimes get rejected even when there are no numerical prob
## Thus, if the number in the 'count' column is small, do not ask about this message on stan-users.
```

```
traceplot(m)
```

```
m
```

```
## Inference for Stan model: egg_ranef.
## 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%
## mu_lambda       0.56    0.00  0.09    0.38    0.50    0.56    0.62    0.72
## mu_p            0.14    0.00  0.24   -0.33   -0.02    0.14    0.30    0.62
## sigma_lambda    0.81    0.00  0.08    0.66    0.75    0.80    0.86    0.97
## sigma_p         2.28    0.02  0.33    1.70    2.05    2.25    2.48    3.00
## lp__         -349.52    1.09 22.02 -393.85 -364.20 -349.13 -334.08 -308.17
##              n_eff Rhat
## mu_lambda     1594 1.00
## mu_p          4000 1.00
## sigma_lambda   904 1.00
## sigma_p        472 1.01
## lp__           410 1.01
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:44:37 2015.
## 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).
```

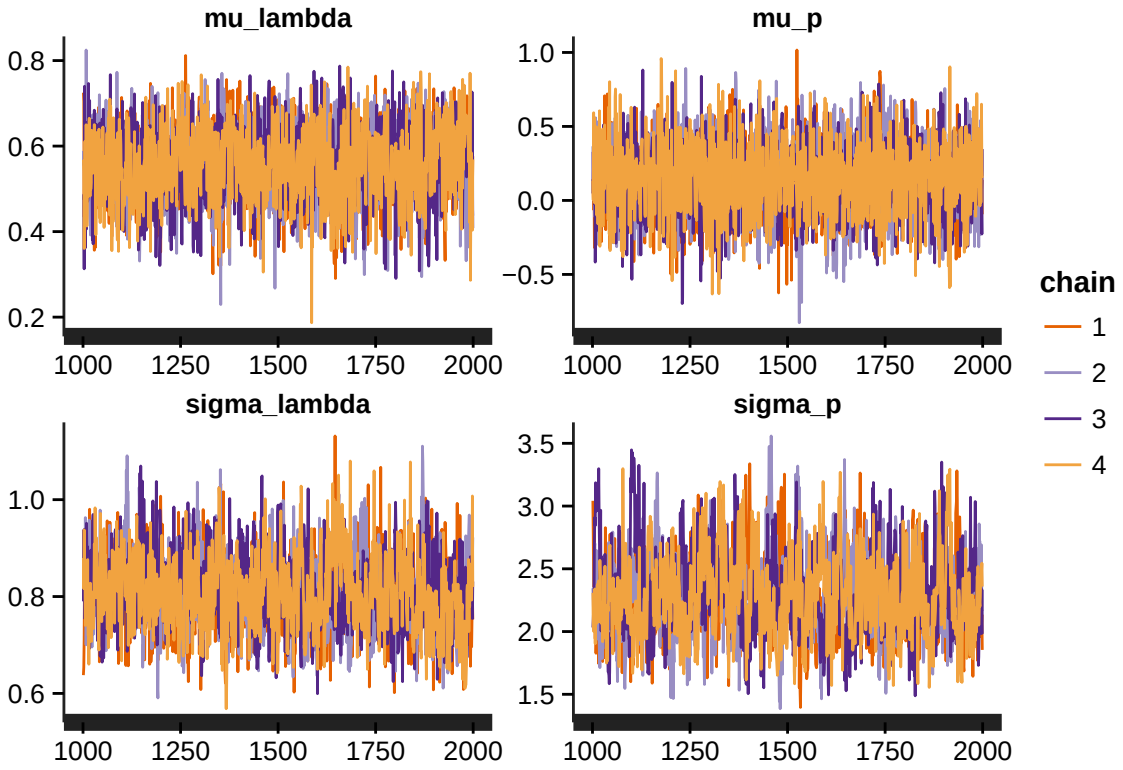


Figure 70: Traceplot of the Markov chains for the `egg_ranef.stan` model, which introduces hyperparameters to account for among-individual variation.

It turns out that this parameterization is not optimal. We can greatly increase the efficiency of our model by using a "non-centered" parameterization for the normal random effects. The basis for this lies in the fact that we can recover a random vector $y \sim N(\mu, \sigma)$ by first generating a vector of standard normal variates: $y_{raw} \sim N(0, 1)$, and then translating the sample to the mean and rescaling all values by σ:

$$y = \mu + y_{raw}\sigma$$

This is more efficient because the MCMC algorithm used with Stan is highly optimized to sample from posteriors with geometry corresponding to N(0, 1) distributions. This trick is incredibly useful for nearly all hierarchical models in Stan that use normal random effects. We can do this translation and scaling in the transformed parameters block, generating the following file called `egg_ncp.Stan`:

```
data {
  // poisson data
  int n;
  int y[n];

  // binomial data
  int n_b;
  int k[n_b];
  int z[n_b];
}

parameters {
  vector[n] log_lambdaR;
```

```
  vector[n_b] logit_pR;
  real mu_lambda;
  real mu_p;
  real<lower=0> sigma_lambda;
  real<lower=0> sigma_p;
}

transformed parameters {
  vector[n] log_lambda;
  vector[n_b] logit_p;

  log_lambda <- mu_lambda + log_lambdaR * sigma_lambda;
  logit_p <- mu_p + logit_pR * sigma_p;
}

model {
  // priors
  mu_lambda ~ normal(0, 1);
  mu_p ~ normal(0, 2);
  sigma_lambda ~ normal(0, 2);
  sigma_p ~ normal(0, 1.5);

  log_lambdaR ~ normal(0, 1);
  logit_pR ~ normal(0, 1);

  // likelihood
  y ~ poisson_log(log_lambda);
  z ~ binomial_logit(k, logit_p);
}
```

Fitting our new model:

```
m <- stan('egg_ncp.stan', data=stan_d)
```

```
traceplot(m, pars=c('mu_lambda', 'mu_p', 'sigma_lambda', 'sigma_p', 'lp__'))
```

```
print(m, pars=c('mu_lambda', 'mu_p', 'sigma_lambda', 'sigma_p', 'lp__'))
```

```
## Inference for Stan model: egg_ncp.
## 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%
## mu_lambda       0.56    0.00  0.09    0.38    0.51    0.56    0.62    0.72
## mu_p            0.15    0.01  0.24   -0.34   -0.01    0.15    0.31    0.61
## sigma_lambda    0.80    0.00  0.08    0.66    0.75    0.80    0.85    0.96
## sigma_p         2.24    0.01  0.33    1.66    2.02    2.22    2.44    2.94
## lp__         -271.02    0.66 18.35 -308.26 -283.18 -270.78 -258.45 -236.62
##              n_eff Rhat
## mu_lambda     2225    1
## mu_p          1510    1
## sigma_lambda  1357    1
```

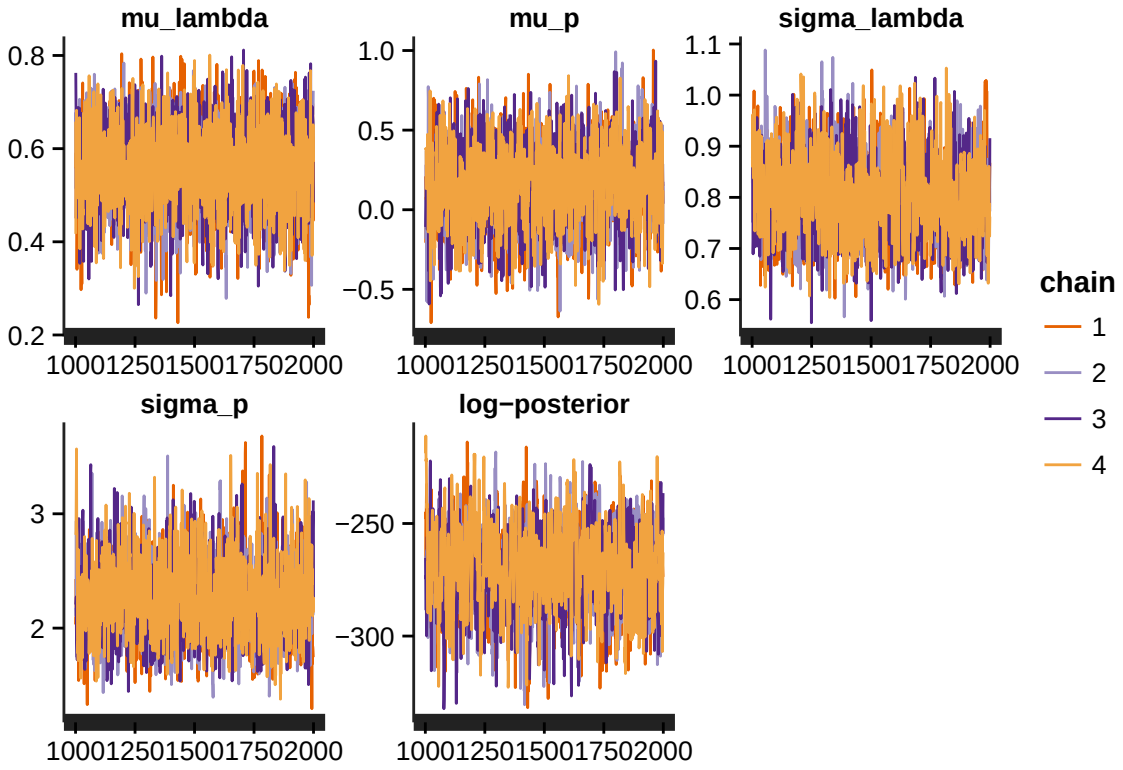


Figure 71: Traceplot from the `egg_ncp.stan` model, which uses a non-centered parameterization for the univariate random effects.

```
## sigma_p       1479    1
## lp__           784    1
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:44:48 2015.
## 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).
```

We might be interested in whether there is any correlation between egg output and egg survival. To explore this, we can plot the random effects on the link scale:

```
post <- extract(m)
ll_meds <- apply(post$log_lambda, 2, median)
lp_meds <- apply(post$logit_p, 2, median)
plot(exp(ll_meds[d$eggs > 0]), plogis(lp_meds),
     xlab=expression(lambda[i]), ylab=expression(p[i]))
```

That plot is not so informative, but if we are interested in the correlation, we can simply calculate the correlation for each draw to get the posterior distribution for the correlation. This is one of the huge advantages of Bayesian inference: we can calculate the posterior distribution for any derived parameters using posterior draws.

```
n_iter <- length(post$lp__)
cor_post <- rep(NA, n_iter)
for (i in 1:n_iter){
```

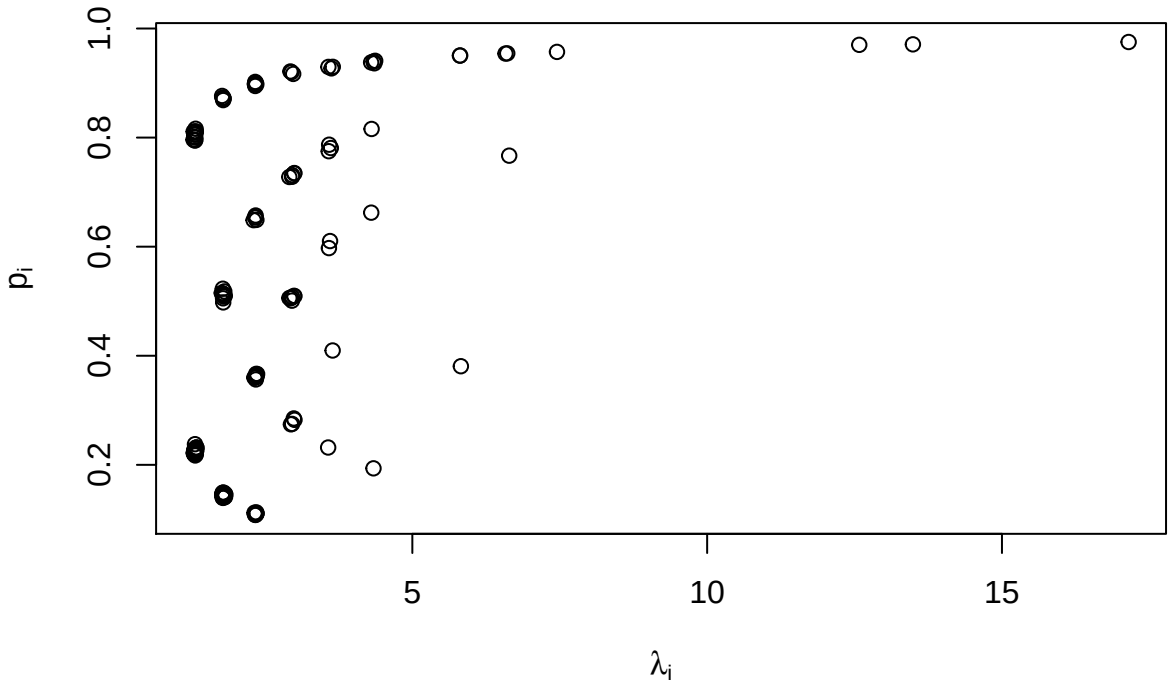


Figure 72: Plot of the posterior medians for the expected number of eggs laid and probability of egg survival for birds that laid eggs.

```
  cor_post[i] <- cor(post$log_lambda[i, d$eggs > 0],
                     post$logit_p[i, ])
}
hist(cor_post, breaks=seq(-1, 1, .02))
```

It appears that birds that produce many eggs tend to have higher per-egg survival. But, we haven't included this correlation in the model explicitly. Generally speaking correlations between two random variables A and B can result from three causal scenarios:

- A or B have a causal effect on eachother, directly or indirectly
- A and B are both affected by some other quantity or quantities
- we have conditioned on a variable that is influenced by A and B (also known as Berkson's paradox)

In this case, we can model correlation between these two latent quantities by way of multivariate normal random effects rather than two independent univariate normal random effects.

Multivariate normal random effects

In many cases, we have multiple random effects which may be correlated. In these instances, many turn to the multivariate normal distribution, which generalizes the univariate normal distribution to N dimensions. The multivariate normal distribution has two parameters: μ, which is a vector with N elements, each describing the mean of the distribution in each dimension, and Σ, an N by N covariance matrix that encodes the variance in each dimension and correlation among dimensions. Any multivariate normal random vector will be a point in N dimensional space.

Consider the bivariate normal distribution, a multivariate normal with $N = 2$ dimensions. The mean vector will have two elements μ_1 and μ_2, that provide the center of mass in the first and second dimension. The covariance matrix will have two rows and columns. We might write these parameters as follows:

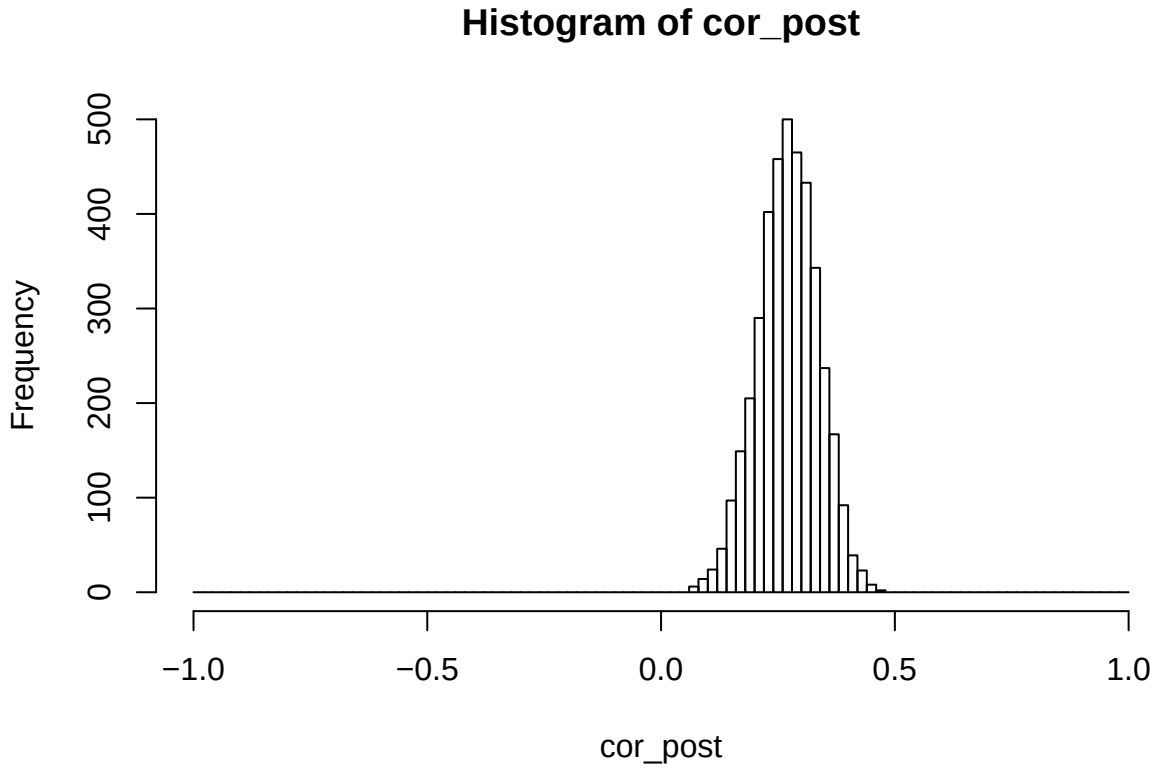


Figure 73: Histogram of the posterior distribution for the correlation between the log expeted number of chicks laid and the logit probability of egg survival.

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \boldsymbol{\Sigma} = \begin{bmatrix} Cov[X_1, X_1] & Cov[X_1, X_2] \\ Cov[X_2, X_1] & Cov[X_2, X_2] \end{bmatrix}$$

The element of $Sigma$ in the i^{th} row and j^{th} column describes the covariance between the i^{th} and j^{th} dimension. By definition, the covariance between one random variable and itself (e.g., $Cov[X_1, X_1]$ and $Cov[X_2, X_2]$) is the variance of the random variable, $\sigma^2_{X_1}$ and $\sigma^2_{X_2}$.

For concreteness, suppose that we're considering the following multivariate normal distribution:

$$\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

We can visualize the density of this bivariate normal as with a heatmap:

Non-centered parameterization: multivariate normal

As with univariate normal random effects, a noncentered parameterization can greatly improve MCMC convergence and efficiency. To acheive this, we first define the Cholesky factor of a matrix L, which is lower triangular, and which equals sigma when multiplied by it's own transpose: $\Sigma = LL^T$. Given L, which is a lower triangular d by d matrix, μ, which is the mean vector with length d, and z, which is a vector of d standard normal N(0, 1) deviates, we can generate a draw from d dimensional multivariate normal distribution $MvN(\mu, \Sigma)$ as follows:

$$y = \mu + Lz$$

Sometimes, it is convenient to parameterize the multivariate normal in terms of a Cholesky decomposed correlation matrix L_R such that $L_R L_R^T = R$ and a vector of standard deviations σ, which can be coerced into

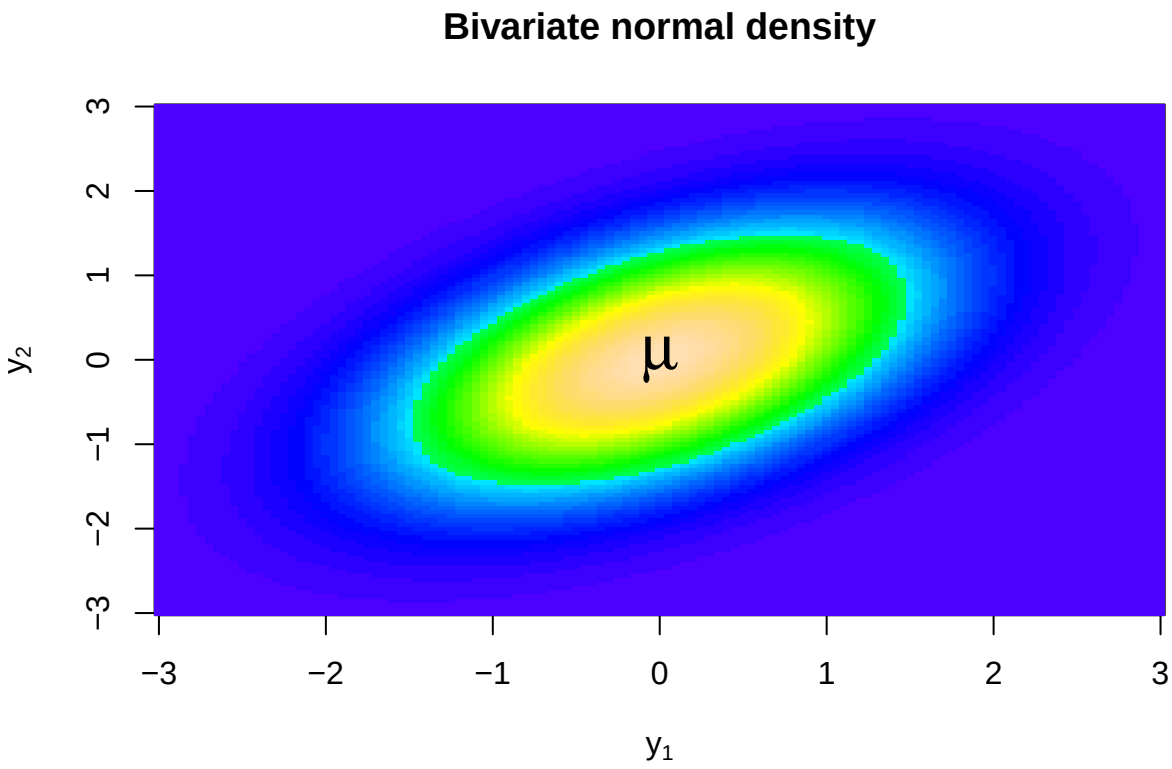


Figure 74: Bivariate normal density heatmap.

a diagonal matrix that has the same dimensions as the desired covariance matrix. If we have these, then we can adapt the above equation to obtain:

$$y = \mu + diag(\sigma) L_R z$$

This parameterization is most useful for hierarchical models, because we can place separate priors on correlation matrices and on the standard deviations. For correlation matrices, it is currently recommended to use LKJ priors, which can be specified on the cholesky decomposed matrix (obviating the need for Cholesky decompositions at each MCMC iteration). The LKJ correlation distribution has one parameter that specifies how concentrated the correlations are around a uniform distribution $\eta = 1$, or the identity matrix with all correlations (non-diagonal elements) equal to zero when η is very large. An LKJ correlation with $\eta = 2$ implies a prior in which correlations are somewhat concentrated on zero. Below are the LKJ prior correlations implied by different values of η.

Let's expand the above model to explicitly allow for correlation between egg survival and egg output. This tends to be useful computationally when parameters are correlated, but it also may be of practical use if egg output or survival are incompletely observed and we'd like to predict the missing data using information on correlated quantities. The main difference will be that instead of two separate univariate normal random effects, we instead have one bivariate normal distribution, and we're modeling correlation between the two dimensions.

$$y_i \sim Poisson(\lambda)$$

$$z_i \sim Binomial(y_i, p)$$

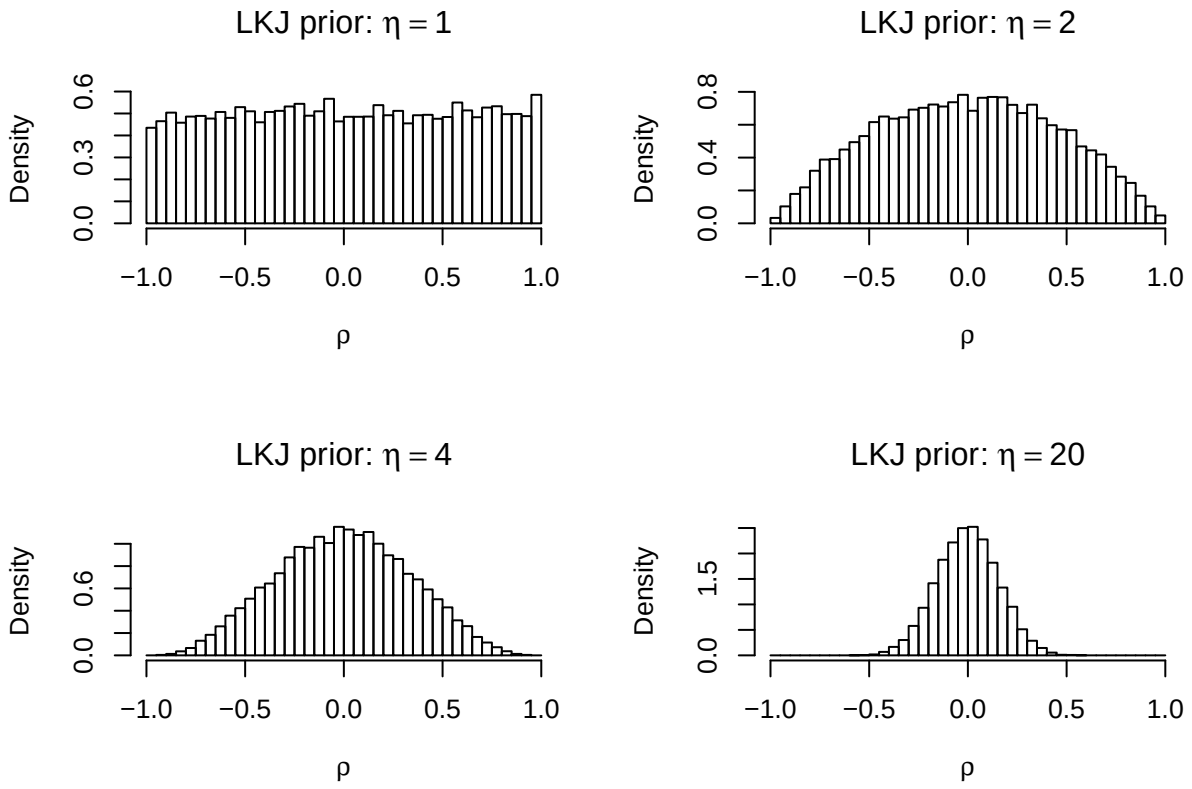


Figure 75: LKJ priors for correlation parameters with varying values of eta.

$$log(\lambda_i) = \alpha_{1i}$$

$$logit(p) = \alpha_{2i}$$

$$\alpha_i \sim N(\mu, \Sigma)$$

$$\mu \sim N(0, 2)$$

$$\Sigma = (diag(\sigma)L_R)(diag(\sigma)L_R)^T$$

$$\sigma \sim halfNormal(0, 2)$$

$$L_R \sim LKJcorr(2)$$

With this new parameterization, we can estimate a random effect vector α_i of length 2 for each individual $i = 1, ..., N$, with elements corresponding to the log expected number of eggs and logit probability of survival for each egg. However, recall that not all individual contribute to the likelihood for egg survival. In particular, we have no survival data from birds that laid zero eggs. This bivariate approach allows us to combine information so that the number of eggs laid informs our estimates of survival probabilities. In this way, we will be able to predict the survival probability of eggs from individuals that did not lay eggs. Here is a Stan model statement, saved in the file `egg_lkj.stan`:

```
data {
  // poisson data
  int n;
  int y[n];

  // binomial data
  int n_b;
  int p_index[n_b];
  int k[n_b];
  int x[n_b];
}

parameters {
  matrix[2, n] z;
  vector[2] mu;
  cholesky_factor_corr[2] L;
  vector<lower=0>[2] sigma;
}

transformed parameters {
  matrix[n, 2] alpha;
  vector[n] log_lambda;
  vector[n_b] logit_p;
  alpha <- (diag_pre_multiply(sigma, L) * z)';

  for (i in 1:n) log_lambda[i] <- alpha[i, 1];
  log_lambda <- log_lambda + mu[1];

  for (i in 1:n_b){
    logit_p[i] <- alpha[p_index[i], 2];
  }
  logit_p <- logit_p + mu[2];
}

model {
  // priors
  mu ~ normal(0, 2);
  sigma ~ normal(0, 2);
  L ~ lkj_corr_cholesky(2);
  to_vector(z) ~ normal(0, 1);

  // likelihood
  y ~ poisson_log(log_lambda);
  x ~ binomial_logit(k, logit_p);
}

generated quantities {
  // recover the correlation matrix
  matrix[2, 2] Rho;

  Rho <- multiply_lower_tri_self_transpose(L);
}
```

Again, here the idea is to not directly sample from the multivariate normal distribution, but instead to sample

from a simpler distribution (univariate standard normal), and transform these values using the cholesky factor of the correlation matrix, vector of standard deviations, and vector of means to generate multivariate normal parameters. It is possible to sample directly from the multivariate normal distribution, but this approach is much more computationally efficient. We have to generate the indexes for the survival observations, bundle the data, and then we can fit the model:

```
p_ind <- which(d$eggs > 0)
stan_d <- list(
  n = nrow(d),
  y = d$eggs,
  n_b = sum(d$eggs > 0),
  p_index = p_ind,
  k = d$eggs[d$eggs > 0],
  x = d$fledglings[d$eggs > 0]
)
m <- stan('egg_lkj.stan', data=stan_d,
          pars=c('Rho', 'alpha', 'sigma', 'mu'))
```

```
## The following numerical problems occured the indicated number of times after warmup on chain 2
##
## Exception thrown at line 44: binomial_logit_log: Probability parameter[1] is -nan, but must be finit
## When a numerical problem occurs, the Metropolis proposal gets rejected.
## However, by design Metropolis proposals sometimes get rejected even when there are no numerical prob
## Thus, if the number in the 'count' column is small, do not ask about this message on stan-users.
```

Let's check convergence for the hyperparameters, which usually implies convergence of the child parameters:

```
print(m, pars=c('Rho', 'sigma', 'mu', 'lp__'))
```

```
## Inference for Stan model: egg_lkj.
## 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%
## Rho[1,1]    1.00    0.00  0.00    1.00    1.00    1.00    1.00    1.00
## Rho[1,2]    0.85    0.00  0.08    0.66    0.80    0.86    0.91    0.97
## Rho[2,1]    0.85    0.00  0.08    0.66    0.80    0.86    0.91    0.97
## Rho[2,2]    1.00    0.00  0.00    1.00    1.00    1.00    1.00    1.00
## sigma[1]    0.78    0.00  0.07    0.64    0.73    0.78    0.83    0.93
## sigma[2]    2.34    0.01  0.34    1.74    2.10    2.32    2.54    3.11
## mu[1]       0.57    0.00  0.09    0.40    0.51    0.57    0.63    0.73
## mu[2]      -0.63    0.01  0.28   -1.21   -0.81   -0.63   -0.44   -0.12
## lp__     -297.18    0.74 19.01 -335.48 -309.85 -297.16 -284.45 -259.57
##          n_eff Rhat
## Rho[1,1]  4000  NaN
## Rho[1,2]   489 1.01
## Rho[2,1]   489 1.01
## Rho[2,2]  4000 1.00
## sigma[1]  1223 1.00
## sigma[2]  1092 1.00
## mu[1]     1294 1.00
## mu[2]      964 1.00
## lp__       661 1.01
```

```
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:46:24 2015.
## 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).
```

```
traceplot(m, pars=c('Rho', 'sigma', 'mu', 'lp__'))
```

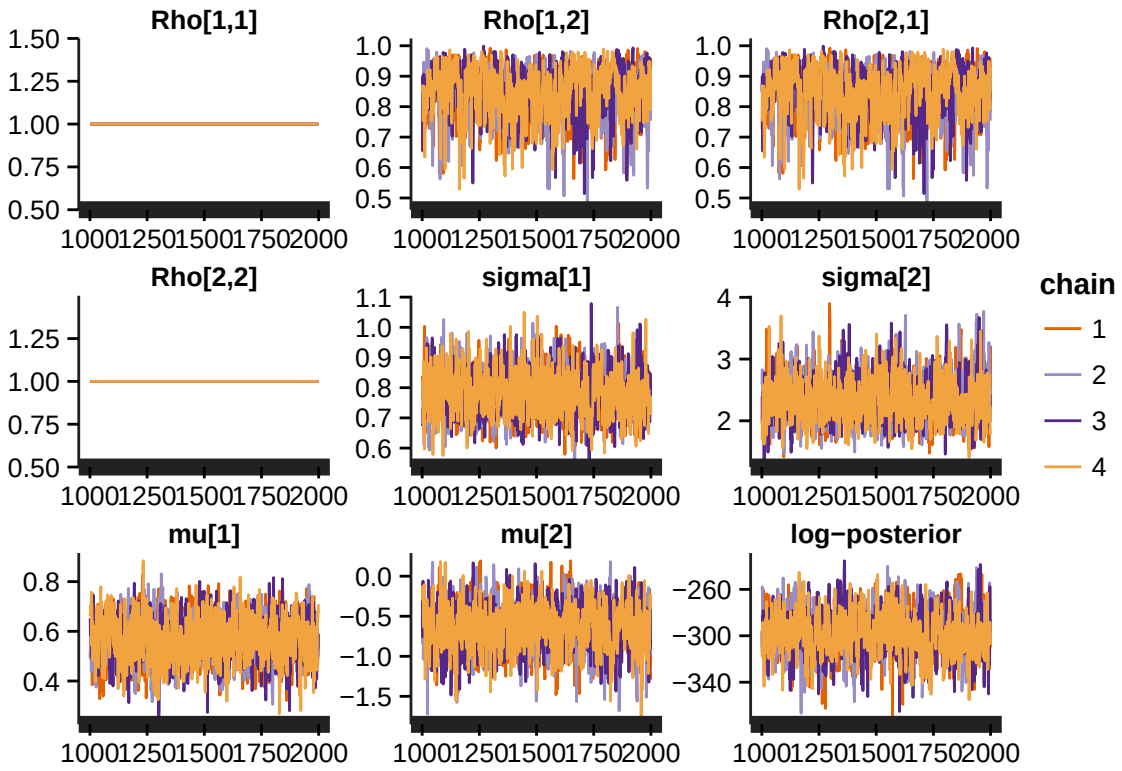


Figure 76: Traceplot of the Markov chains for the multivariate normal noncentered model.

In this example, we might proceed by adding "fixed" effects of body mass, so that we can evaluate how much of the correlation between clutch size and survival may be related to body size. We leave this as an exercise, but point out that this can be accomplished using design matrices or for-loops in the Stan file.

Varying intercepts and slopes

Some of the most common applications of Bayesian hierarchical models involve intercept and slope parameters that vary among groups. In these cases, it is often wise to allow the intercepts and slopes to correlate, and this is mostly accomplished via multivariate normal random effects, where one dimension corresponds to intercepts, and the other to slopes. To demonstrate this, we will use a classic example from a sleep study in which the reaction times of 18 subjects was measured daily with increasing levels of sleep deprivation.

```
library(lme4)
ggplot(sleepstudy, aes(x=Days, y=Reaction)) +
  geom_point() +
  stat_smooth(method='lm') +
  facet_wrap(~ Subject)
```

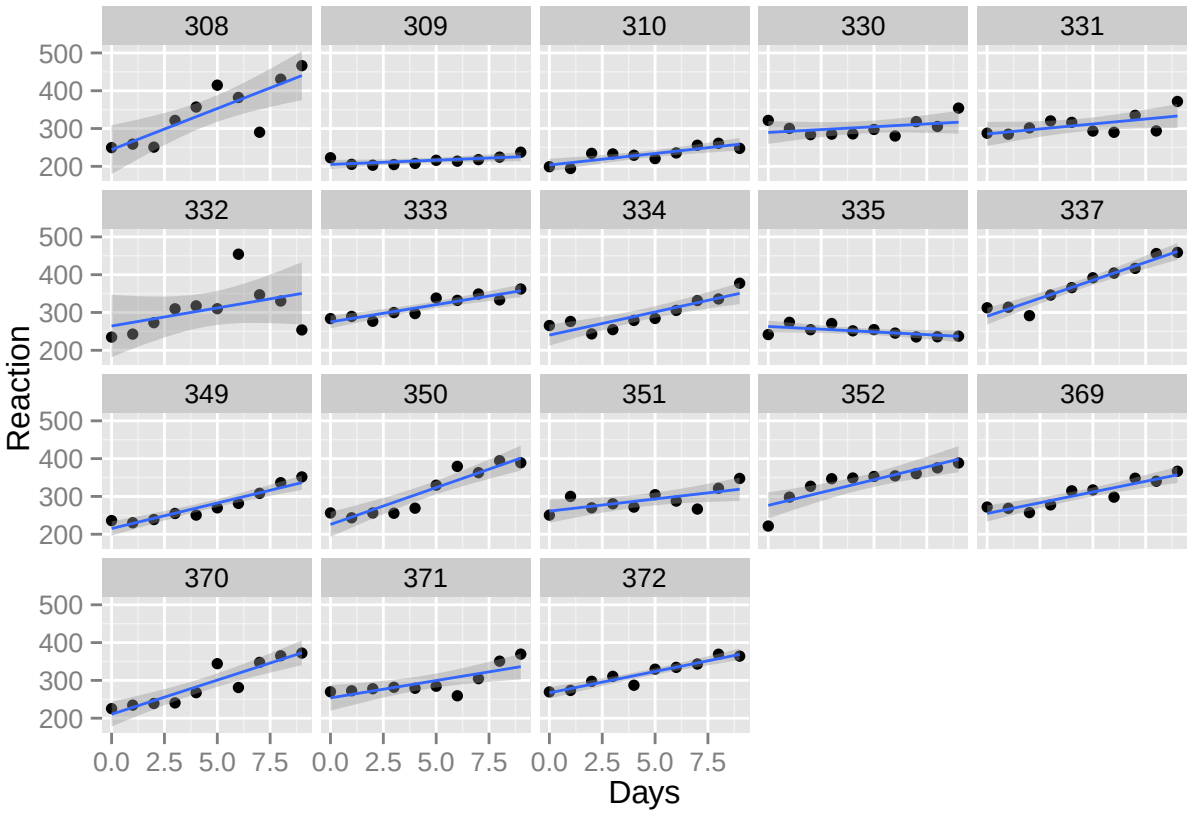


Figure 77: Results from the sleep study experiment. Each panel is a different patient.

We might envision the following model that allows the intercepts (reaction on day 0) and slope (daily change in expected reaction time) to vary among subjects, with normally distributed error, indexing subjects by i and days by t:

$$y_{it} \sim N(\mu_{it}, \sigma_y)$$

$$\mu_{it} = \alpha_i + \beta_i t$$

$$\begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_\alpha \\ \mu_\beta \end{bmatrix}, \Sigma \right)$$

We can implement this model in `lme4` if we want:

```
mle <- lmer(Reaction ~ Days + (Days | Subject), data=sleepstudy)
```

Or, we could translate the model to Stan:

```
data {
  int n;
  vector[n] y;
  int n_subject;
  int n_t;

  // indices
  int<lower=1, upper=n_subject> subject[n];
  int<lower=1, upper=n_t> t[n];
}

parameters {
  matrix[2, n_subject] z;
  vector[2] mu;
  cholesky_factor_corr[2] L;
  vector<lower=0>[2] sigma;
  real<lower=0> sigma_y;
}

model {
  to_vector(z) ~ normal(0, 1);
}
```

```
stan_d <- list(n = nrow(sleepstudy),
               y = sleepstudy$Reaction,
               tmax = max(sleepstudy$Days),
               t = sleepstudy$Days,
               n_subject = max(as.numeric(sleepstudy$Subject)),
               subject = as.numeric(sleepstudy$Subject))
m <- stan('sleep.stan', data=stan_d,
          pars = c('mu', 'sigma', 'sigma_y', 'alpha', 'Rho'))
```

Checking convergence:

```
traceplot(m, pars = c('mu', 'sigma', 'sigma_y', 'Rho'))
```

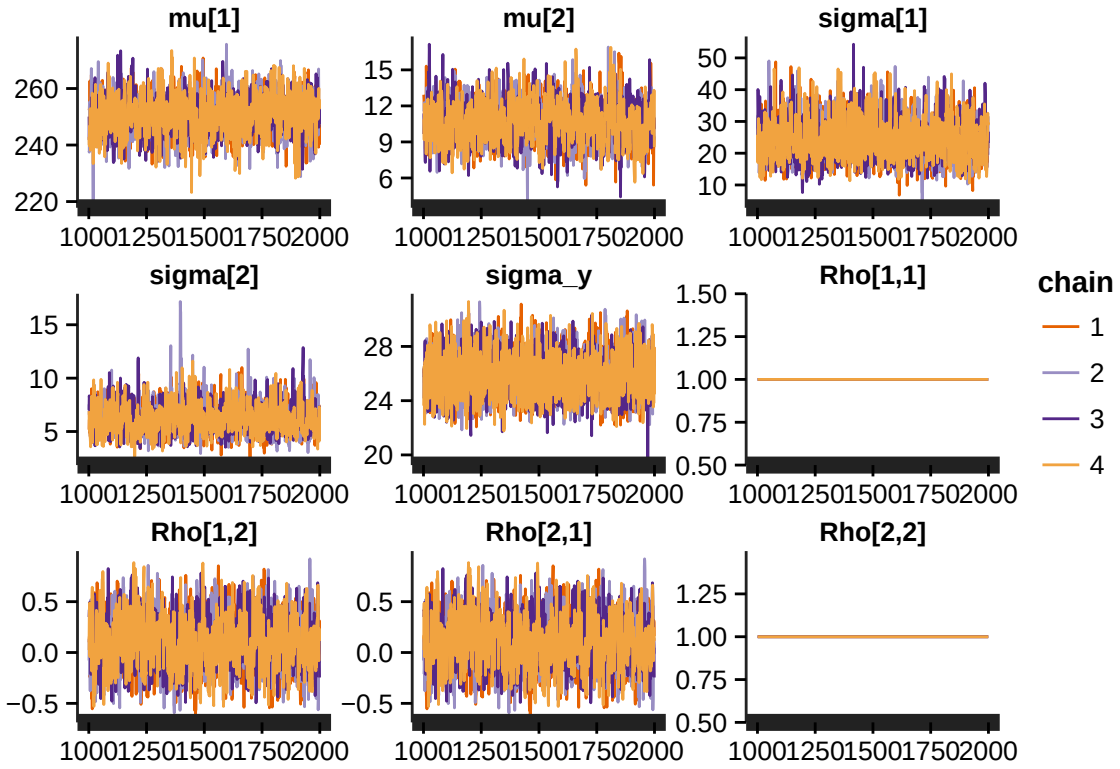


Figure 78: Traceplot for the Markov chains from the sleep study example, which uses multivariate normal random effects to allow for correlation between the intercepts and slopes.

```
print(m, pars = c('mu', 'sigma', 'sigma_y', 'Rho'))
```

```
## Inference for Stan model: sleep.
## 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
## mu[1]    250.45    0.18 6.74 237.48 245.89 250.50 254.89 263.84  1377    1
## mu[2]     10.72    0.05 1.66   7.50   9.63  10.74  11.80  14.05  1233    1
## sigma[1]  24.09    0.16 6.20  13.89  19.73  23.47  27.81  37.65  1581    1
## sigma[2]   6.17    0.04 1.37   3.93   5.22   6.06   6.91   9.29  1197    1
## sigma_y   25.89    0.02 1.56  23.06  24.80  25.82  26.90  29.10  4000    1
## Rho[1,1]   1.00    0.00 0.00   1.00   1.00   1.00   1.00   1.00  4000  NaN
## Rho[1,2]   0.11    0.01 0.27  -0.39  -0.08   0.11   0.30   0.64   896    1
## Rho[2,1]   0.11    0.01 0.27  -0.39  -0.08   0.11   0.30   0.64   896    1
## Rho[2,2]   1.00    0.00 0.00   1.00   1.00   1.00   1.00   1.00  4000    1
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:47:11 2015.
## 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).
```

These results are not very different than those of the `lmer` implementation, but with a Bayesian implementation we immediately have the full posterior distribution for every parameter, which is a huge advantage.

Further reading

Hobbs and Hooten. 2015. *Bayesian models: a statistical primer for ecologists.* Chapter 6.

Gelman and Hill. 2009. *Data analysis using regression and multilevel/hierarchical models.* Chapter 13.

Gelman, A., J. Hill, and M. Yajima. 2012. Why We (Usually) Don't Have to Worry About Multiple Comparisons. Journal of Research on Educational Effectiveness 5:189–211.

Gelman, Andrew. Prior distributions for variance parameters in hierarchical models (comment on article by Browne and Draper). Bayesian Analysis. 1 (2006), no. 3, 515–534.

Chapter 8: Hierarchical model construction

Big picture

Translating problems to models is a key skill, and it may take a fair bit of practice. In this chapter we introduce tools to help with this step: parameter, process, and observation models, and a few examples that make use of familiar probability distributions.

Learning goals

- building models from simple submodels
- parameter, process, and observation models
- examples:
  - occupancy model
  - N-mixture model
  - error in variables models

Parameter, process, and observation models

One incredibly powerful perspective in model construction is the distinction between observation, process, and parameter models. The observation model includes the likelihood, and relates the latent process of interest to the quantities that we have actually observed. The process model is typically what we're most interested in scientifically, and typically is not fully observed. The parameter model includes any hierarchical structures on parameters and priors for hyperparameters.

Depending on the nature of any particular problem, it may be advantageous to begin model formulation at different levels. For instance, if there is a specific biological process that one has in mind, but it is not clear what sources of data will be necessary, it might be useful to first construct the process model and then think about what sources of information are needed to estimate the process parameters. In other situations, the process based model may be simple or known ahead of time, but a model must make use of vastly different sources of data. In these cases, it may be more useful to begin by developing an observation model.

Example: occupancy model

Occupancy models represent the presence or absence of a species from a location as a Bernoulli random variable Z, with $z = 0$ corresponding to absence and $z = 1$ corresponding to presence. If the species is present $z = 1$, it will be detected on a survey with probability p. So, species may be present but undetected. If a species is absent, it will not be detected (we assume no false detections), but species may be present and undetected.

We can write this as a hierarchical model with occurrence state z treated as a binary latent (hidden) variable:

$$[z \mid \psi] \sim Bernoulli(\psi)$$

$$[y \mid z, p] \sim Bernoulli(zp)$$

The observation model is $[y \mid z, p]$, and the process model is $[z \mid \psi]$.

If we wish to avoid the use of a discrete parameter, we can marginalize z out of the posterior distribution by summing over all possible values (in this case, just 0 and 1):

$$[\psi, p \mid y] \propto \sum_z [y \mid z, p][z \mid \psi][p, \psi]$$

$$[\psi, p \mid y] \propto [p, \psi] \sum_z [y \mid z, p][z \mid \psi]$$

$$[\psi, p \mid y] \propto [p, \psi]\big([y \mid z = 1, p][z = 1 \mid \psi] + [y \mid z = 0, p][z = 0 \mid \psi]\big)$$

$$[\psi, p \mid y] \propto [p, \psi]\big(\psi Bernoulli(p) + (1 - \psi)I(y = 0)\big)$$

where $I(y = 0)$ is an identity function that sets y to zero because we assumed that there are no false detections when the species is absent. Multiple surveys are necessary to identify ψ and p, so that we can expand the likelihood to account of binomial observation histories, with k surveys conducted per site, and observation history vectors y_i for the i^{th} site:

$$[y_i \mid \psi_i, p_i] \begin{cases} \psi_i Binom(y_i, k) & \text{if } \sum y_i > 0 \\ \psi_i Binom(0, k) + (1 - \psi) & otherwise \end{cases}$$

If the organism was observed, then we know that any non-detections represent false absences. If the organism was not observed, then it was either there and not observed (with probability $\psi_i Binom(0, k)$) or it was not there with probability $1 - \psi$. This if-else structure can be exploited in Stan to implement this likelihood:

```
data {
   int<lower=0> nsite;
   int<lower=0> nsurvey;
   int<lower=0,upper=1> y[nsite,nsurvey];
}
parameters {
   real<lower=0,upper=1> psi;
   real<lower=0,upper=1> p;
}
model {
   for (i in 1:nsite) {
     if (sum(y[i]) > 0)
       // species was observed: it is there
       increment_log_prob(log(psi) + bernoulli_log(y[i],p));
     else
       // it may or may not be there
       increment_log_prob(log_sum_exp(log(psi) + bernoulli_log(y[i],p),
                                      log1m(psi)));
   }
}
```

Now let's simulate some occupancy data and fit the model:

```
nsite <- 50
nsurvey <- 3
psi <- .4
p <- .8
z <- rbinom(nsite, 1, psi)
```

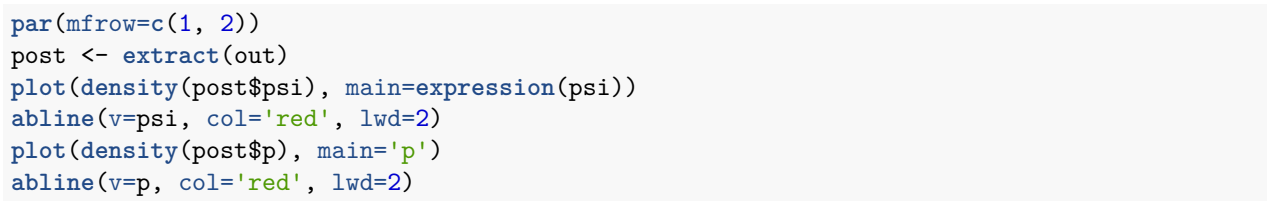
```
y <- matrix(rbinom(nsite * nsurvey, 1, z * p),
            nrow=nsite)

stan_d <- list(nsite = nsite,
               nsurvey = nsurvey,
               y = y)
out <- stan('occupancy.stan', data = stan_d)
```

How did we do?

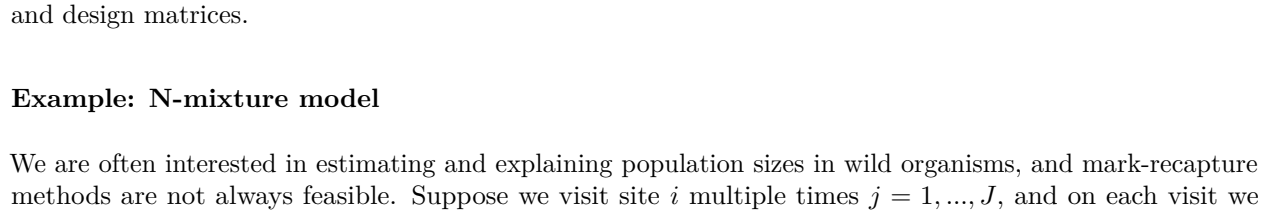
```
out
```

```
## Inference for Stan model: occupancy.
## 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
## psi    0.35    0.00 0.06   0.23   0.30   0.34   0.39   0.47  2151    1
## p      0.89    0.00 0.04   0.78   0.86   0.89   0.92   0.96  2466    1
## lp__ -53.22    0.03 0.99 -55.96 -53.56 -52.91 -52.54 -52.26  1121    1
##
## Samples were drawn using NUTS(diag_e) at Sun Dec 13 19:47:13 2015.
## 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).
```

```
traceplot(out)
```



```
par(mfrow=c(1, 2))
post <- extract(out)
plot(density(post$psi), main=expression(psi))
abline(v=psi, col='red', lwd=2)
plot(density(post$p), main='p')
abline(v=p, col='red', lwd=2)
```



The parameter-level model can be extended to include covariates for ψ and p by making use of a link function and design matrices.

Example: N-mixture model

We are often interested in estimating and explaining population sizes in wild organisms, and mark-recapture methods are not always feasible. Suppose we visit site i multiple times $j = 1, ..., J$, and on each visit we conduct a survey, recording the number of individuals observed. If we can assume that across visits, the true abundance N_i is unchanged, and that the detection of each individual is imperfect but independent, then we might construct an N-mixture model to estimate abundance.

Observation model

We observe a subset of the individuals at the site, and each individual is detected with probability p_i:

$$y_{ij} \sim Binom(N_i, p_i)$$

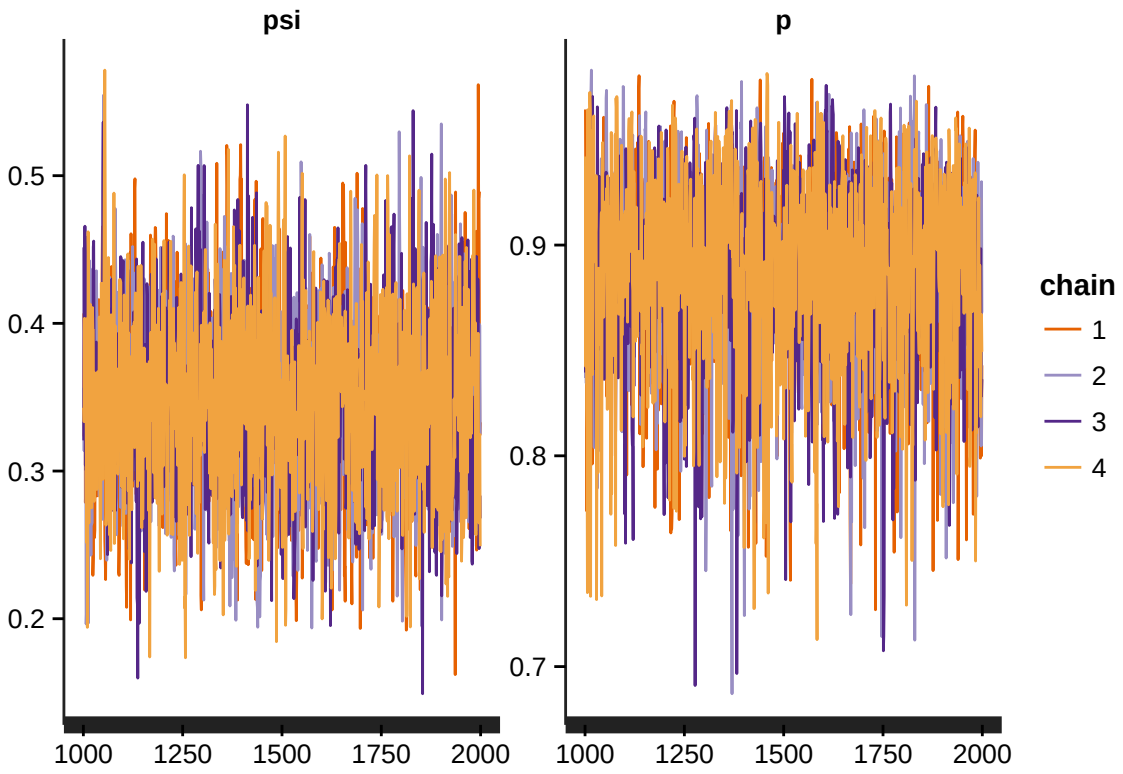


Figure 79: Traceplot of the Markov chains from the occupancy model.

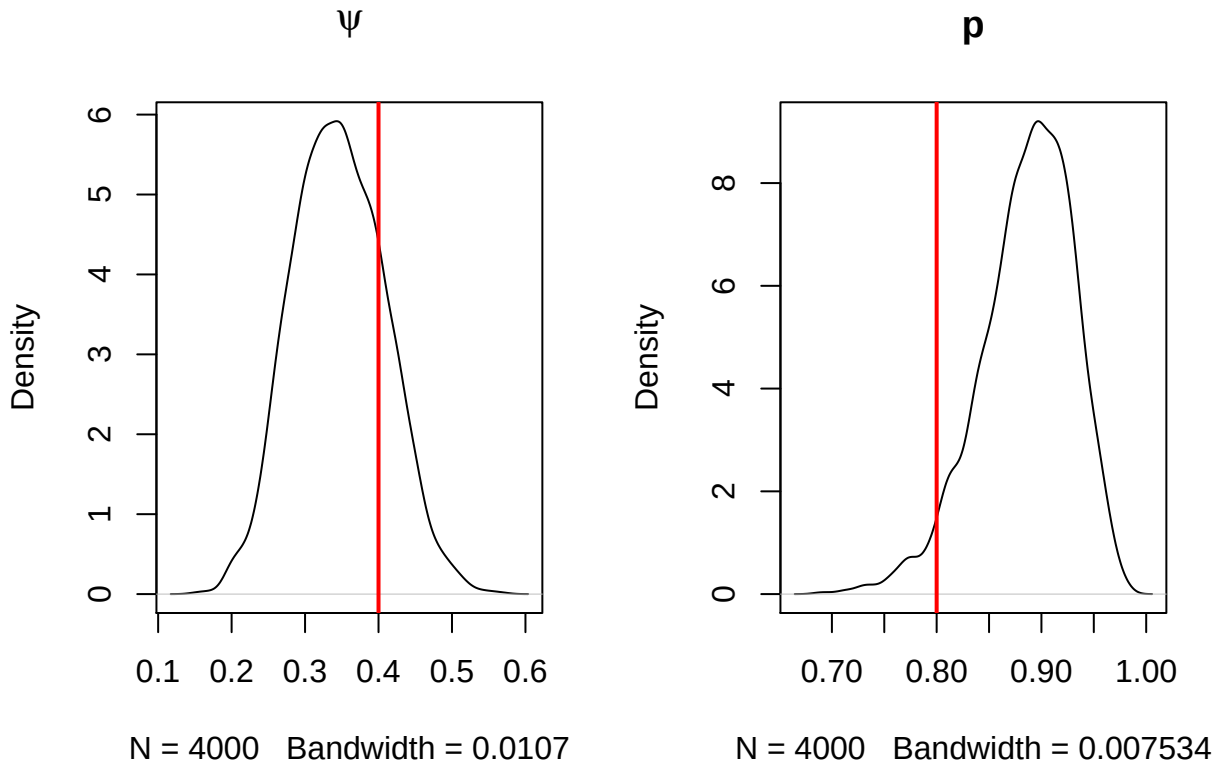


Figure 80: Posterior densities for occupancy and detection probabilities along with their known true values.

Notice that both parameters of the binomial observation model are unknown. True abundance N_i is a latent quantity.

Process model

We need some discrete probability distribution for N, such as the Poisson or negative binomial.

$$N_i \sim Pois(\lambda_i)$$

Parameter model

Finally, we need priors for p and λ.

$$p \sim ...$$

$$\lambda \sim ...$$

We note that this model has integer parameters, and as such cannot be fitted (presently) with Stan, but this is easy to implement in JAGS.

Example: error in variables models

Most of the time in ecology, covariates are observed with error and subsequently assumed to be known exactly. This causes bias in slope estimates, and despite the fact that this has been known for over a century, most people carry on assuming that the covariate is fixed.

For what follows, we'll assume a simple linear regression, in which continuous covariates are measured with error. But, this approach can be applied to any of the models that we've covered in this class.

True covariate values are considered latent variables, with repeated measurements of covariate values $x_{i=1}, ..., x_{i=n}$ arising from a normal distribution with a mean equal to the true value, and some measurement error σ_x. Again, this is a special case, but in principle the covariate could have other distributions (e.g., you're interested in the effect of sex (M or F), but this is measured imperfectly).

Observation model

We assume that for sample unit i and repeat measurement j:

$$x_{ij}^{obs} \sim Normal(x_i, \sigma_x)$$

$$y \sim Normal(\mu_y, \sigma_y)$$

The trick here is to use repeated measurements of the covariates to estimate and correct for measurement error. In order for this to be valid, the true covariate values cannot vary across repeat measurements. If the covariate was individual weight, you would have to ensure that the true weight did not vary across repeat measurements (for me, frogs urinating during handling would violate this assumption).

Process model

This is what we're typically interested in: how the expected value of y varies with x. In this case, x is a parameter (it will require a prior).

$$\mu_y = \alpha + \beta x_i$$

Parameter model

In the parameter model, we would specify priors for x, α, β, and the variance parameters. More on this example can be found here.

General strategies for model building

Verify your model

It is always a good idea to verify your model, meaning to ensure that the estimation procedure is reliable. One way to do this is with simulated data. In particular, simulating data from a known generative model, and then trying to recover the parameters can give you insights into a) whether your model even works, b) the behavior of the MCMC algorithm, and c) the frequentist properties of your estimate (long-run interval coverage, bias). In addition, this procedure gives you piece of mind. It is also often useful to investigate the implications of and sensitivity to model misspecification in this context, where there are known differences between the generative model and the statistical model. In my experience this excercise nearly always proves to be incredibly useful.

Start simple

Too often, people charge forward and build incredibly elaborate hierarchical Bayesian models, only to find that they don't work in some way (either they don't converge, recover parameters, take too long, etc.). Instead, start with simple models. Add complexity incrementally. That way, when something breaks, you will be more likely to know what model component is causing the problem. This also helps to clarify thinking about what complexity can be safely ignored, and what must be included in the model.

Stay simple

It's easy to build incredibly complex models, but simple models are sometimes more powerful. Also, simple models are more likely to be broadly applicable to other systems, whereas an incredibly complicated model might only apply to your system. In papers, simple models are much easier to explain as well.

Further reading

Gelman and Hill. 2009. *Data analysis using regression and multilevel/hierarchical models.* Chapter 14 & 15.

Royle JA. N-mixture models for estimating population size from spatially replicated counts. Biometrics 60.