Lab 8:

Likelihood & Bootstrapping Confidence Intervals

Gregor Thomas

Monday, February 29, 2016

We'll use the Radon data from G&H.

The response, `y`, is a (logged) measurement of Radon gas inside people's homes. The predictors we'll use are `x`, the floor on which the measurement was taken (0 for basement, 1 for first floor—Radon comes out of the ground), and `county`, the county the house is in (all the data is from Minnesota).

Glance at the data:

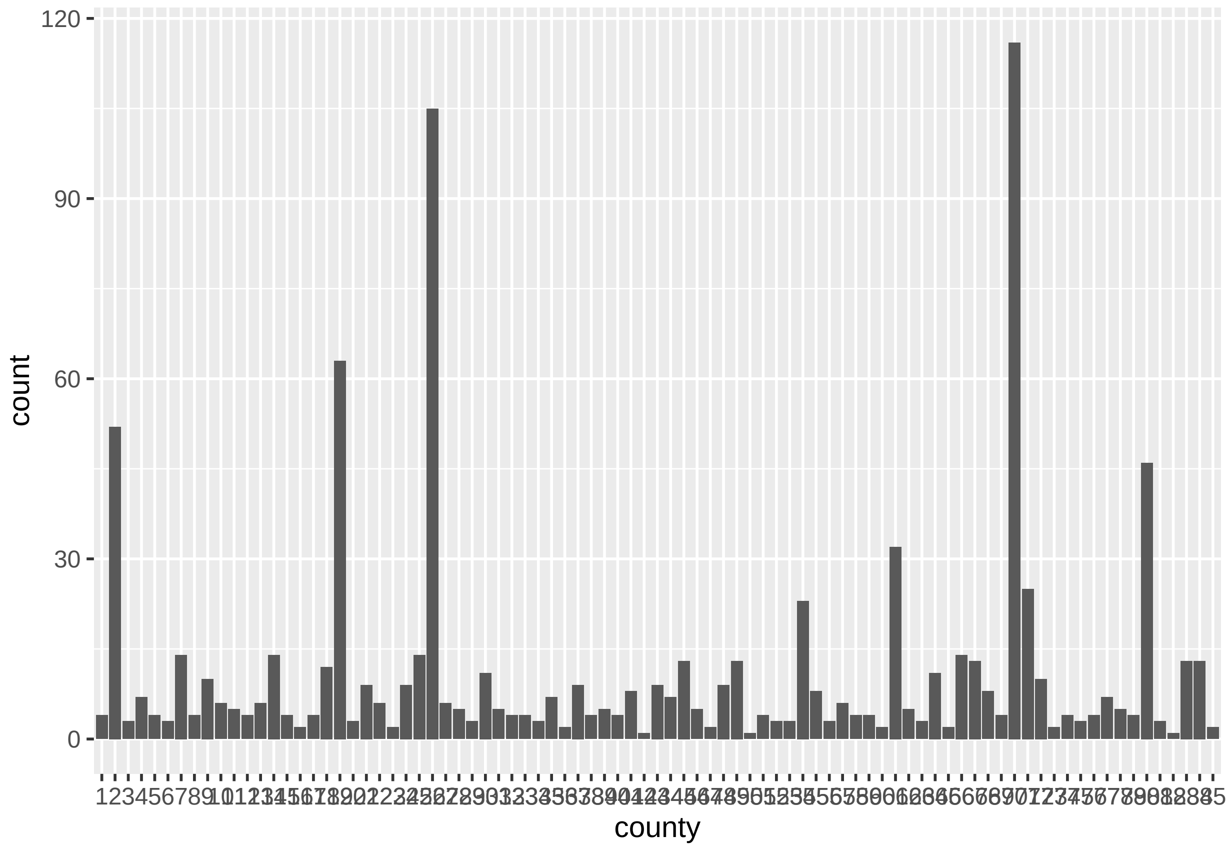
```
load("radon.RData")
str(radon)
```

```
## 'data.frame':    919 obs. of  4 variables:
##  $ x     : num  1 0 0 0 0 0 0 0 0 0 ...
##  $ y     : num  0.788 0.788 1.065 0 1.131 ...
##  $ county: Factor w/ 85 levels "1","2","3","4",..: 1 1 1 1 2 2 2 2 2 2 ...
##  $ u     : num  -0.689 -0.689 -0.689 -0.689 -0.847 ...
```

```
summary(radon)
```

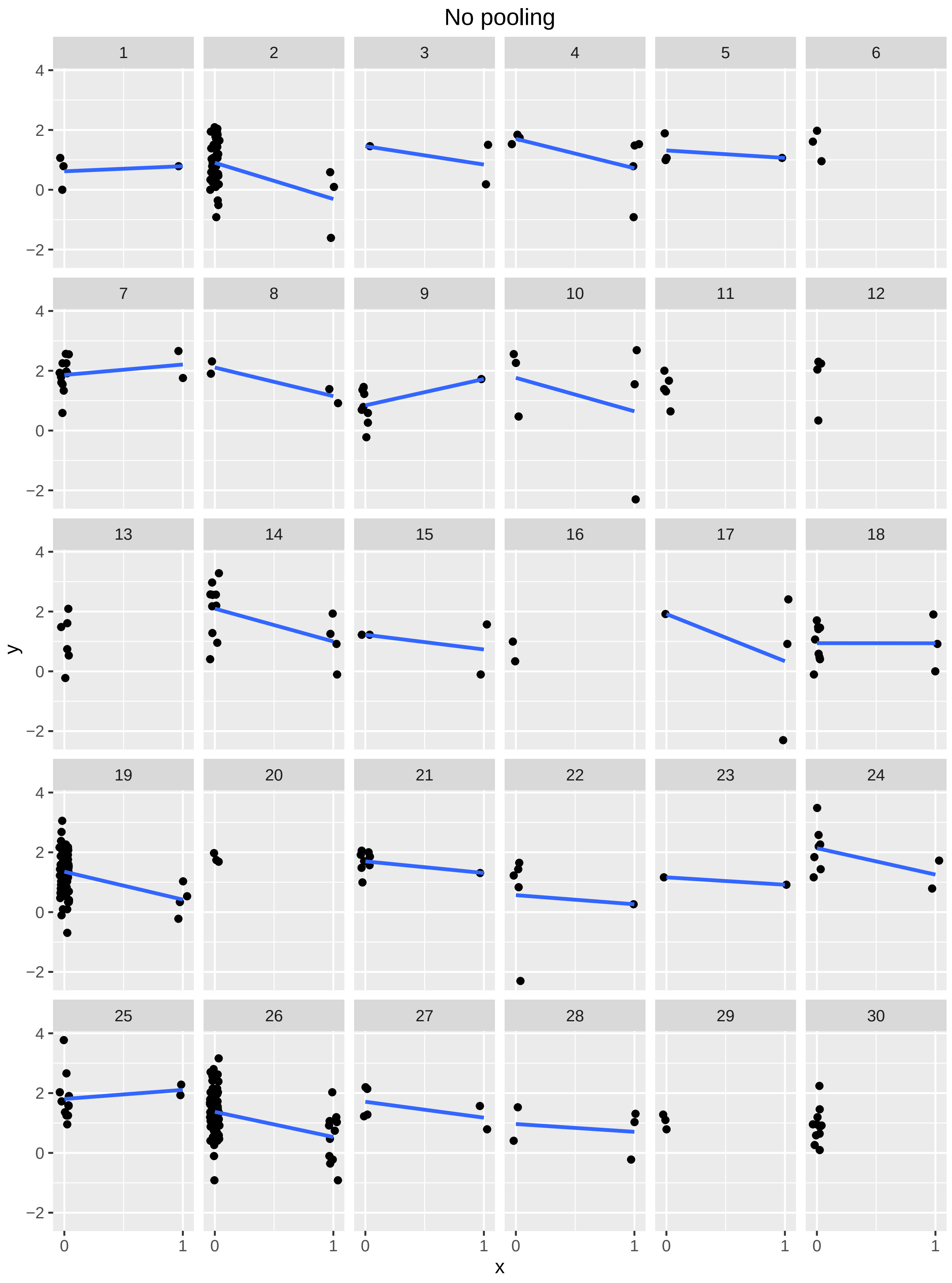
```
##        x                y                county          u
##  Min.   :0.0000   Min.   :-2.3026   70     :116   Min.   :-0.88183
##  1st Qu.:0.0000   1st Qu.: 0.6419   26     :105   1st Qu.:-0.47467
##  Median :0.0000   Median : 1.2809   19     : 63   Median :-0.09652
##  Mean   :0.1665   Mean   : 1.2246   2      : 52   Mean   :-0.13171
##  3rd Qu.:0.0000   3rd Qu.: 1.7918   80     : 46   3rd Qu.: 0.18324
##  Max.   :1.0000   Max.   : 3.8754   61     : 32   Max.   : 0.52802
##                                     (Other):505
```

```
ggplot(radon, aes(x = county)) +
    geom_bar()
```



Graph with response:

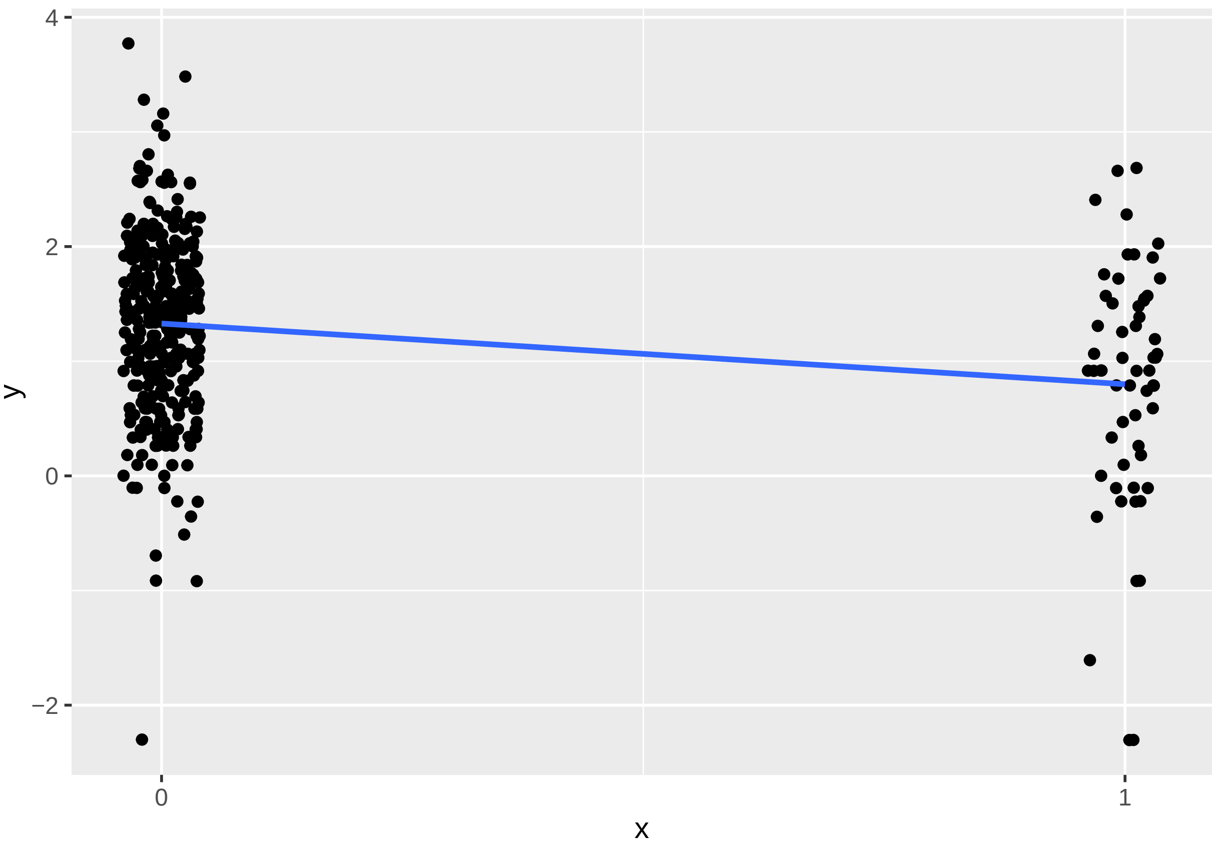
```r
ggplot(filter(radon, county %in% 1:30),
       aes(x = x, y = y)) +
    geom_point(position = position_jitter(width = 0.1, height = 0)) +
    scale_x_continuous(breaks = 0:1) +
    geom_smooth(method = "lm", se = FALSE) +
    facet_wrap(~ county, ncol = 6) +
    labs(title = "No pooling")
```



The above graph shows no-pooling estimates. Every county considered individually, with no information shared county to county.

By contrast, complete pooling throws every observation in together (in the same big pool) and ignores the county differences.

```r
ggplot(filter(radon, county %in% 1:30),
       aes(x = x, y = y)) +
    geom_point(position = position_jitter(width = 0.1)) +
    scale_x_continuous(breaks = 0:1) +
    geom_smooth(method = "lm", fill = NA)
```

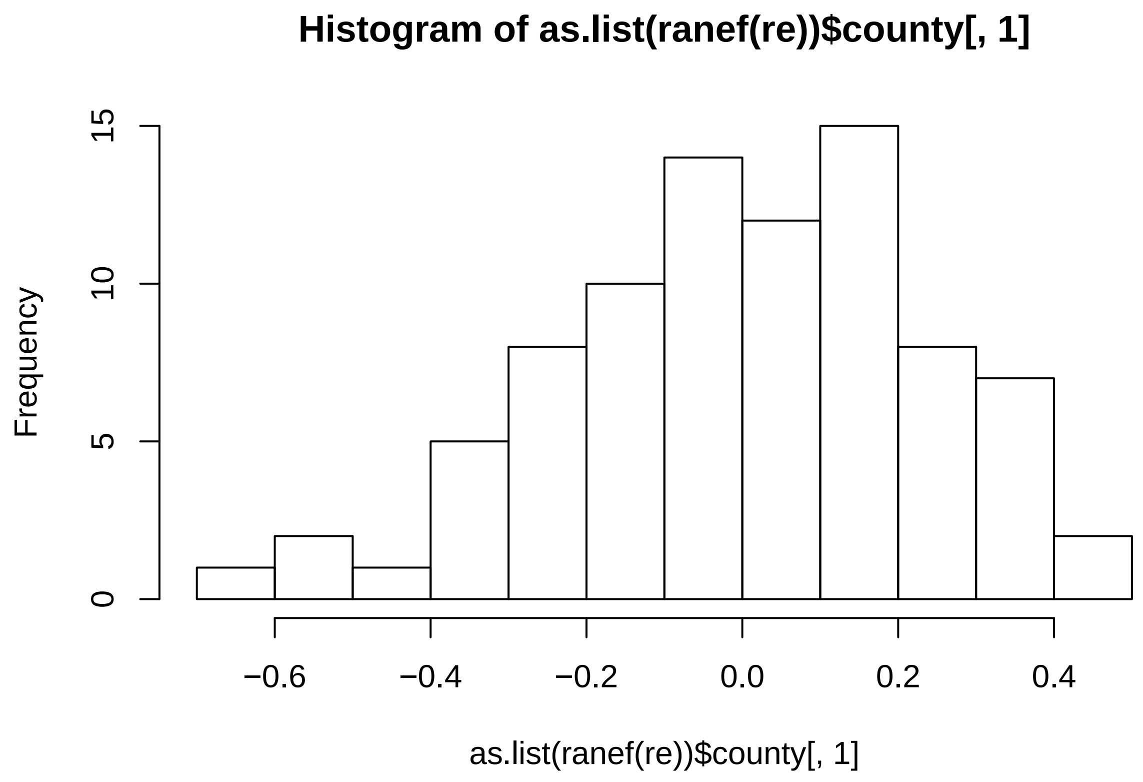


Let's fit a *partial-pooling* model where the slope is the same in every county, but the base level of radon in the soil can vary, and is expected to follow a normal distribution among the counties.

```r
re = lmer(y ~ x + (1 | county), data = radon)
```

Histogram of county intercepts: (as an optional exercise, you could fit a model with county as a fixed effect, `y ~ x + county` and look at a histogram of the county intercepts estimated with no pooling).

```r
hist(as.list(ranef(re))$county[, 1])
```



`lmer` Comparison

The `lme4` vignette (http://cran.r-project.org/web/packages/lme4/vignettes/lmer.pdf) gives a more detailed description of everything to follow.

Guidelines

- AIC/ANOVA can be used to compare models where the random effects are the same (e.g., for parameter selection of the fixed effects).
- When inference is the goal, the choice of whether to use fixed or random effects depends more *on the analytic questions* than on the model fits.
- However, if prediction is the goal, then cross-validation should be used to guide your choice.

For standard errors / p-values of individual coefficients,

- Calculated p-values (no longer included in output), are based on a Chi-Squared distribution. Asymptotically, this may be true, but in practice they **overstate significance.**

Confidence Intervals

Three ways of generating confidence intervals

1. Generate confidence intervals using *strong assumptions* about the likelihood **at the MLE**. Use the derivatives of the Likelihood at the MLE to guess confidence intervals.

- Computationally very easy
- Heavy on assumptions (valid asymptotically)

2. Likelihood profiles: Vary the parameters systematically around your estimate (MLE) and record how the likelihood changes.

- More computationally intensive
- Less reliant on assumptions

3. Bootstrap - resample data according to bootstrap rules, and re-fit the model. Do that a bunch, and construct CIs based on the distribution of parameter estimates observed across the bootstrap resamples.

- Very slow computationally
- Nice and accurate
- Added bonus: resampling takes care of outliers pretty well too

Topological metaphor for likelihoods

We can imagine that finding a MLE of a parameter is like trying to find the highest elevation point of a landscape. If your landscape looks like the Matterhorn, with a nice sharp peak, you can be very sure of exactly where the MLE is. However, if your landscape is more like gently rolling hills, there will be a much wider confidence interval where the highest point *might* be. Even worse are ridged surfaces…

Examples:

```
(ci_wald = confint(re, method = "Wald"))
```

```
##                  2.5 %     97.5 %
## .sig01              NA         NA
## .sigma              NA         NA
## (Intercept)  1.3605103  1.5626854
## x           -0.8310356 -0.5549519
```

```
# only works on fixed effects terms

(ci_prof = confint(re, method = "profile"))
```

```
## Computing profile confidence intervals ...
```

```
##                  2.5 %     97.5 %
## .sig01       0.2435081  0.4222524
## .sigma       0.7205403  0.7926003
## (Intercept)  1.3605862  1.5647622
## x           -0.8309670 -0.5542531
```

```r
# works on fixed and random effects

# requires the boot package (loaded up top)
(ci_boot = confint(re, method = "boot"))
```

```
## Computing bootstrap confidence intervals ...
```

```
##                   2.5 %     97.5 %
## .sig01       0.2289936  0.4097047
## .sigma       0.7166880  0.7922557
## (Intercept)  1.3582823  1.5823991
## x           -0.8339749 -0.5648346
```

```r
# slowest, most accurate
```

We can visually examine likelihood profiles:

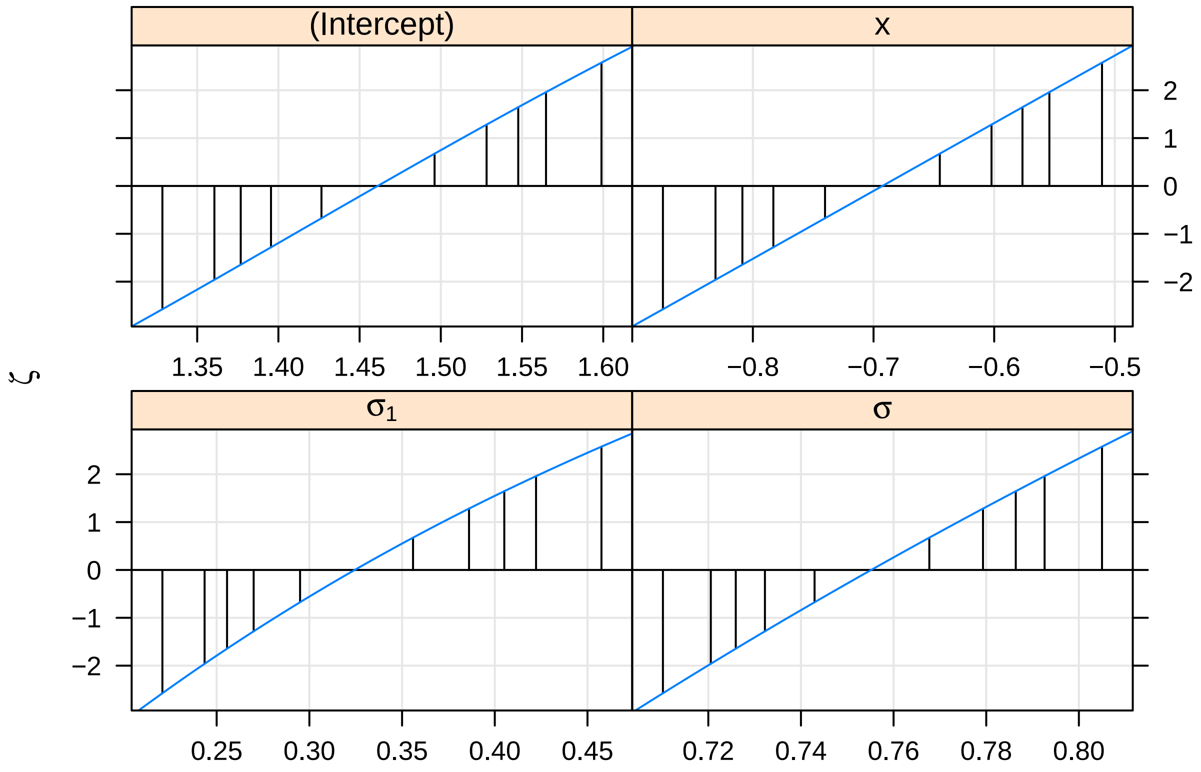
```r
re_prof = profile(re)
```

This is a *profile zeta plot* for each parameter. The vertical axis is a signed square root of the deviance (signed so that the sign of zeta corresponds to the sign of the difference from the fitted parameter value). Linearity is nice but not absolutely necessary.

The vertical lines indicate confidence bands—50%, 80%, 90%, 95%, 99% (customizable, see `?xyplot.thpr`). The y-axis should correspond to standard deviations, and yew can see that the 4th vertical line from the center corresponds to a y-value just under 2, 1.96 to be exact.

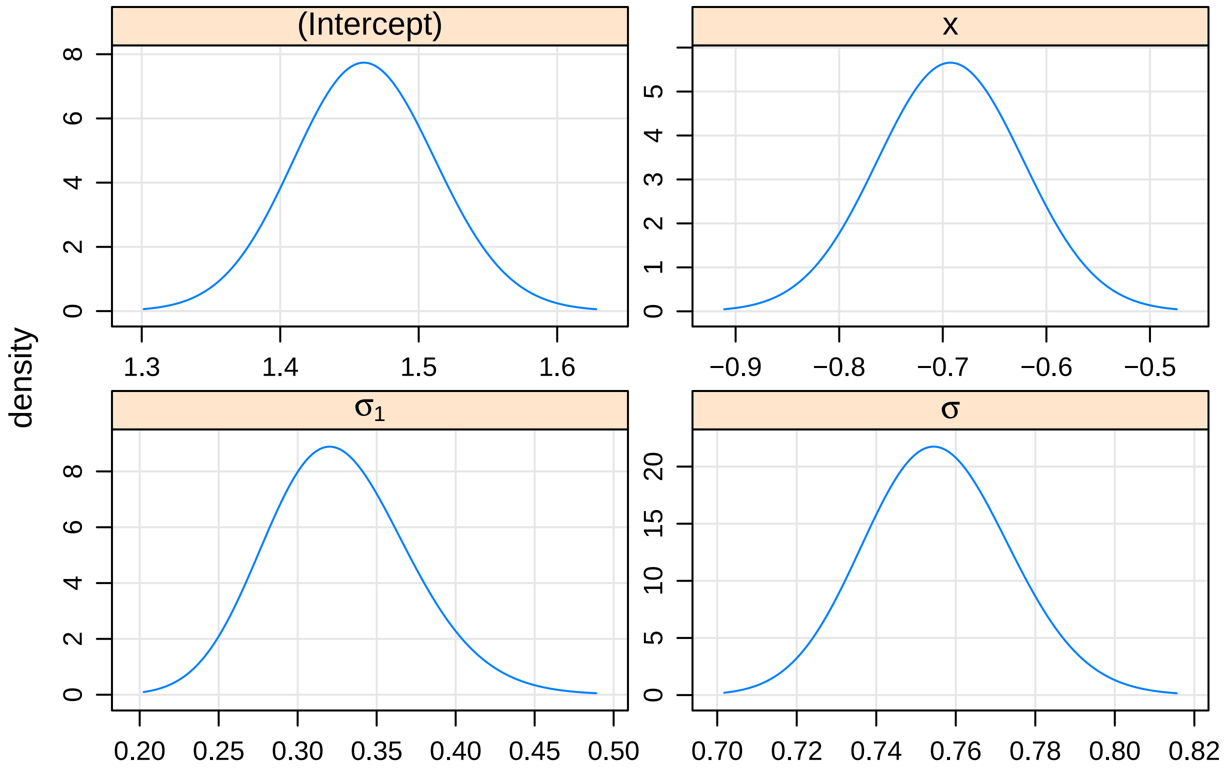
(The Wald confidence intervals we did above assume perfect linearity (which is really an assumption that the coefficient estimate is perfectly normally distributed), so when the likelihood profile is linear the Wald estimate should be accurate.)

```r
xyplot(re_prof)
```



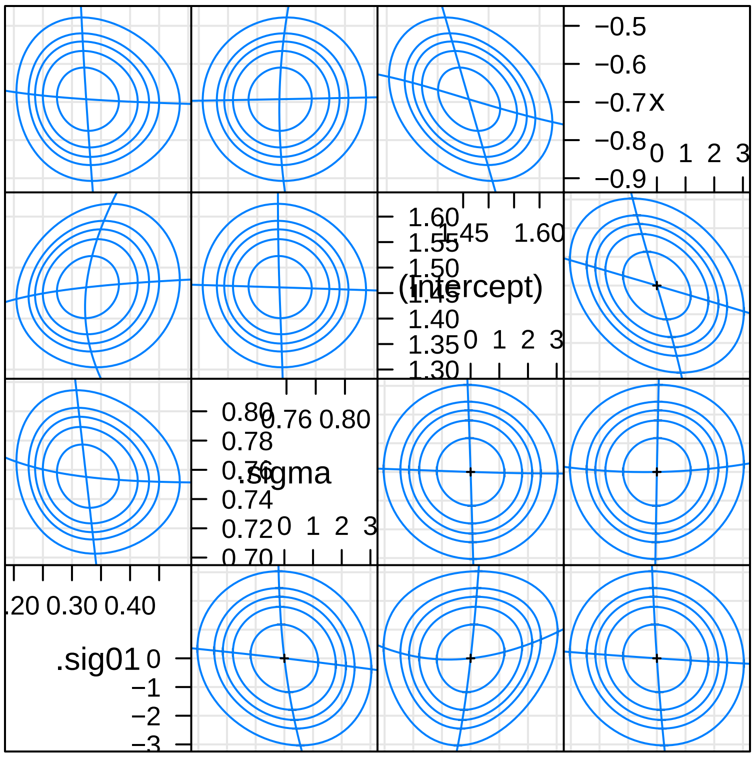
The density plot is a distributional way of looking at things. I like these better, but they can actually be harder to read. Deviations from a straight line in the `xyplot` are more obvious than skew in this density plot.

```
densityplot(re_prof)
```



Finally, looking at pairs of variables in a scatter plot matrix we can see the correlations between parameters. Ideally nothing would be correlated and everything would be perfect circles. That's never the case though, ellipses show correlation between parameter estimates, which is also fine up to a point. What we *don't* want to see are weird shapes: funnels, ridges, extreme ellipses, etc.

```
splom(re_prof)
```



Scatter Plot Matrix

Generally, if any of the likelihood profiles look strange, that's a good indication that you should be using a bootstrap estimation technique instead.

Parametric bootstrap

A statistical *bootstrap* technique is a nice way to leverage your data into inferences when calculations could be difficult or distributional assumptions may not be completely true. It's a good way to get the sense of the variability in possible estimates that could have come from your data. You treat your actual-data estimate as true, and then your resample your data (drawing the same total number of observations, with replacement) and treat that as the sample. You do it many times, and get a p-value (much like our other simulation methods).

This is nicely implemented in the `boot` package for `merMods` (and many other classes). This is about the only way to get accurate p-values for the parameters of these models, but it can be slow. (Aside: this is why `lme4` *suggests* `boot` in it's DESCRIPTION file.)

Missing data

Briefly, a nice way to deal with missing data is to *impute* it - try to fill in the holes. This can work very well when data is *MCAR*, missing completely at random. This usually involves bootstrap estimates. Nice implementations in R include the packages `Amelia`, `mi` and `missForest`.