Stanislaw

Contents

1 Goal

What is the goal of statistical analysis in science?

One way to think about it: we want to build models with parameters that inform our theories.

We can use probability to evaluate and express uncertainty about possible values of these parameters, and to compare and criticize the models themselves.

What are the plausible values of some parameters θ after we have observed our data?

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

For writing out models more explicitly, we'll need to specify how the likelihood ($p(y \mid \theta)$: the probability of our data given some specific set of parameters) for each data point contributes to the overall probability.

$$p(\theta \mid y) \propto p(\theta) \prod_{n=1}^{N} p(y_n \mid \theta)$$

For coding models in stan, it might be helpful to think also about adding up the log probability for each observation to the overall log probability.

$$\log p(\theta \mid y) \propto \log p(\theta) + \sum_{n=1}^{N} \log p(y_n \mid \theta)$$

[We won't cover it, but think about how you could extend this joint probability if you had N observations for each of S subjects, with θ split up into θ_{group} and $\theta_{subject}$]

2 Why Stan?

So, the functions above let us evaluate the probability of any parameter values given our data. If we plug in all the possible (binned) values our parameters could take on, we have a posterior probability distribution!

Many interesting models have too many parameters to do this...

Maybe we just search for the best combination of parameters and use those? (Maximum likelihood or Maximum a posteriori)

Not as informative, and we can get weird answers for some types of parameters (see literally every lmer error)

Stan and other MCMC techniques allow us to approximate very high dimensional probability distributions without trying out every combination of parameters.

3 What is Stan?

Stan is a probabilistic programming language.

Stan uses Hamiltonian MCMC to approximate $p(\theta \mid y)$. See https://observablehq.com/@herbps10/hamiltonian-monte-carlo for an animated explaination.

We can write out (almost) any probabilistic model and get full probability distributions to express our uncertainty about model parameters!

Stan can be used through R with the **rstan** package:

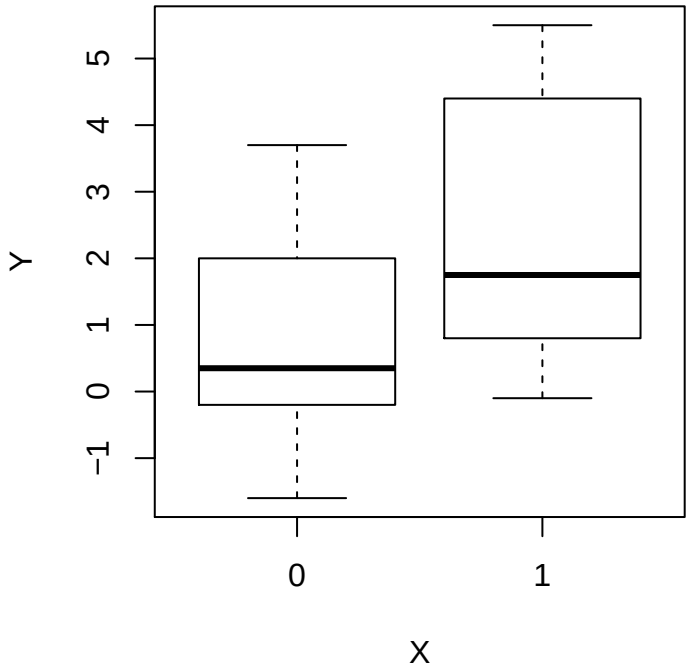
```
library(rstan)
```

4 Models

4.1 Example dataset

Suppose it's 1905 and we are evaluating the effectiveness of two soporific drugs. (R: `?sleep`)

We measured 20 volunteers' normal sleep durations. We then gave 10 volunteers drug A and 10 volunteers drug B, and measured how much longer they slept compared to their usual sleep duration. Let's keep things general and call the extra duration of sleep Y, and the drug variable X.



We will then start building statistical models to help us evaluate the potentially different effects of these two drugs. We consider three gaussian models of the extra sleep duration Y:

1. 2-parameter model: One mean and one standard deviation.
2. 3-parameter model: Different means for the two drugs, common SD.
3. 4-parameter model: Different means and SDs for the two drugs.

4.2 Model 1

We assume that the extra sleep durations y_n in $1, \ldots, N$ are normally distributed, with mean μ and standard deviation σ.

You have seen this model before written out as

$$y_n = \mu + \varepsilon_n, \text{where}$$
$$\varepsilon_n \sim N(0, \sigma^2)$$

But we prefer the following notation for its clarity and emphasis on data rather than errors.

$$y_n \sim N(\mu, \sigma^2)$$

To complete the model, we specify vague priors on both parameters.

$$\mu \sim \text{N}(0, 10)$$
$$\sigma \sim \text{HalfCauchy}(0, 10)$$

We specify this mathematical model in Stan language in three "blocks" of code: `data`, `parameters`, and `model`. Before looking at the code, let's look at our data to make sure we understand what data is going into our model (Stan requires data to be input as a list, not a data.frame)

```
datalist
```

```
$N
[1] 20

$y
 [1]  0.7 -1.6 -0.2 -1.2 -0.1  3.4  3.7  0.8  0.0  2.0  1.9  0.8  1.1  0.1
[15] -0.1  4.4  5.5  1.6  4.6  3.4

$x
 [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
```

Then we write out our Stan code. Note how the data block corresponds to the list of data in R, and how the model block corresponds to the mathematical statements above. We also show in the model block the many equivalent ways of writing the data model (these are commented out with `//`)

```
data {
  int N;
  vector[N] y;
  vector[N] x;
}

parameters {
  real mu;
  real<lower=0> sigma;
}

model {
  // Priors count once towards log posterior
  mu ~ normal(0, 10);
```

```
  sigma ~ cauchy(0, 10);

  // Likelihood counts N times
  for (n in 1:N) {
    y[n] ~ normal(mu, sigma);
    // Equivalent statements of whats going on here:
    // target += normal_lpdf( y[n] | mu, sigma);
    // increment_log_prob(normal_log(y[n], mu, sigma));

  }
  // Vectorized version
  // y ~ normal(mu, sigma);
}
```

We have saved this model into `m1.stan`. To sample from the model's posterior, we pass the filename to `stan()` with a variable indicating the data list in current R environment.

```
fit1 <- stan(
  "m1.stan",
  data = datalist
)
```

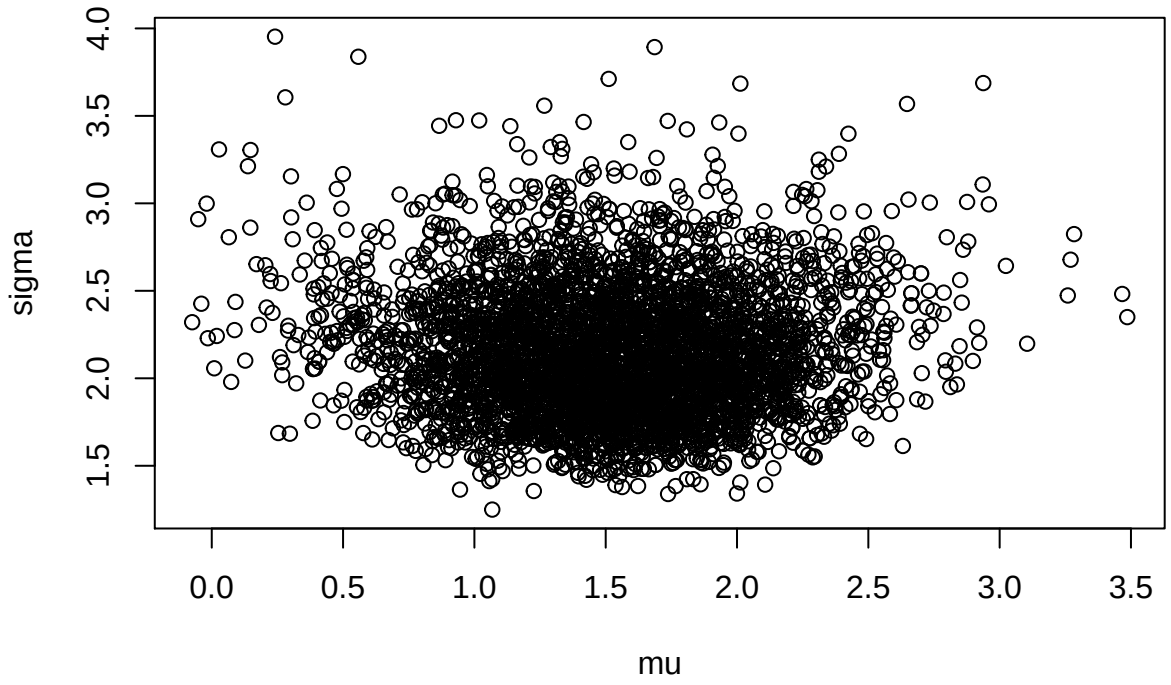
```
recompiling to avoid crashing R session
```

After Stan has drawn samples, we can summarize the parameters' posterior draws numerically, graphically, or transform them to give posteriors of other interesting unknown quantities.

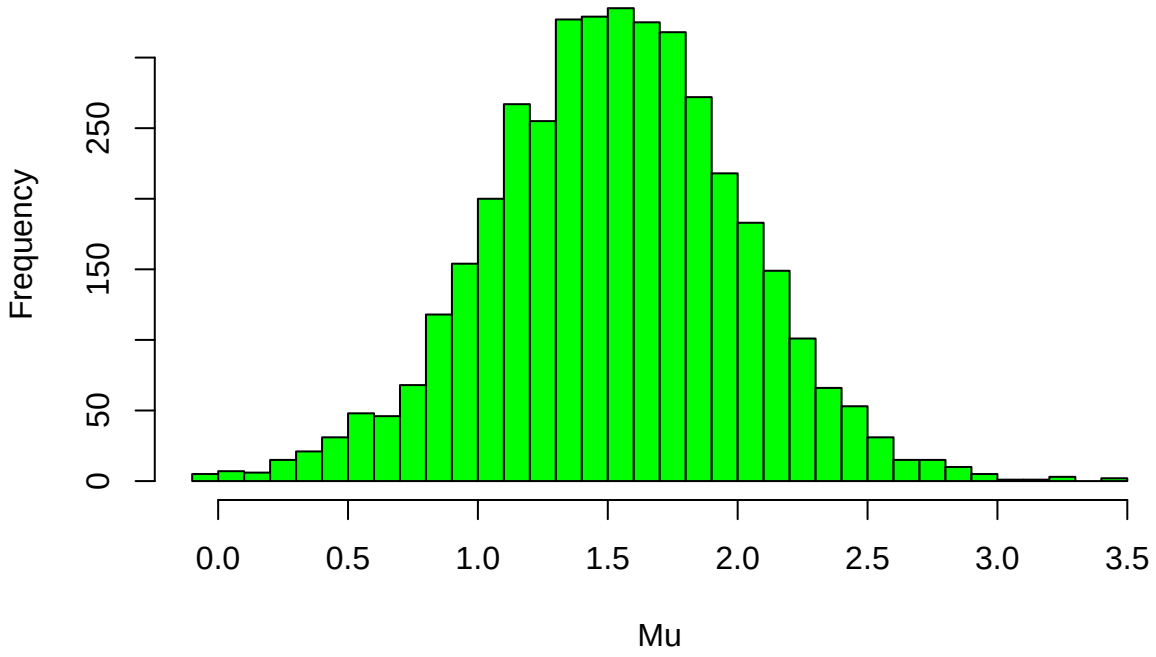
```
print(fit1, probs = c(.05, .5, .95))
```

```
Inference for Stan model: m1.
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     5%    50%    95% n_eff Rhat
mu      1.53    0.01 0.48   0.74   1.53   2.30  2240    1
sigma   2.15    0.01 0.37   1.63   2.10   2.81  2441    1
lp__  -23.92    0.03 0.99 -25.86 -23.64 -22.96  1281    1

Samples were drawn using NUTS(diag_e) at Wed Apr 10 13:39:44 2019.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

```
plot(as.data.frame(fit1)[,1:2])
```




```
hist(as.data.frame(fit1)$mu,main=NULL,xlab="Mu",breaks=50,col='green')
```



If you are familiar with R's formula syntax, the above model is similar to

```
lm(y ~ 1, data = dat)
```

4.3 Model 2

Our previous model was not that interesting because we didn't have any predictors of μ or σ. We first ask if the mean of extra sleep durations varies between the two drugs (e.g. perhaps drug A led to greater sleep duration increases than drug B or vice versa.)

We do this by writing out a linear model for μ, predicting it from an intercept and effect of drug B.

$$y_n \sim N(\mu_n, \sigma^2)$$
$$\mu_n = b_0 + b_1 D_n$$

Priors

$$\mu \sim \text{N}(0, 10)$$
$$b_1 \sim \text{N}(0, 5)$$
$$\sigma \sim \text{HalfCauchy}(0, 10)$$

Note that the Drug variable is binary here (making this a two sample t-test), but it could also be continuous (making this a univariate regression), or a factor with > 2 levels (making this an ANOVA)... The Stan code is very similar to above

```
data {
  int N;
  vector[N] y;
  vector[N] x;
}

parameters {
  real b0;
  real b1;
  real<lower=0> sigma;
}

model {
  vector[N] mu;

  b0 ~ normal(0, 10);
  b1 ~ normal(0, 5);
  sigma ~ cauchy(0, 10);

  for (n in 1:N) {
    mu[n] = b0 + b1*x[n]; // mu changes for each obs
    y[n] ~ normal(mu[n], sigma); // likelihood for single obs
  }
}
```

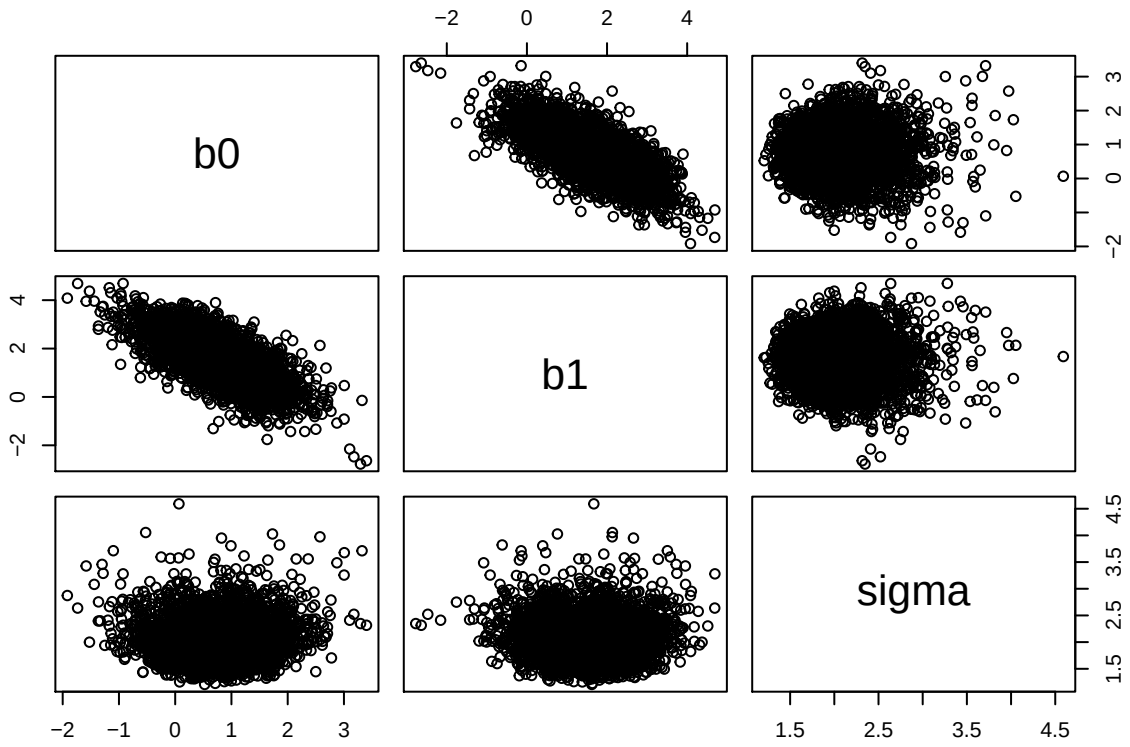
```
fit2 <- stan(
  "m2.stan",
  data = datalist
)
```

```
print(fit2, probs = c(.05, .5, .95))
```

```
Inference for Stan model: m2.
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     5%    50%    95% n_eff Rhat
b0      0.77    0.01 0.67  -0.30   0.77   1.85  2067    1
```

```
b1      1.54    0.02 0.93  -0.02   1.55   3.07  1941    1
sigma   2.04    0.01 0.38   1.51   1.98   2.74  2267    1
lp__  -22.96    0.03 1.36 -25.59 -22.63 -21.47  1560    1

Samples were drawn using NUTS(diag_e) at Wed Apr 10 13:39:46 2019.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

```
plot(as.data.frame(fit2)[,1:3])
```



You may recognize this model in R's formula syntax as

```
lm(y ~ x, data = dat)
```

4.4 Model 3

Hold on, why are we only modeling the location of the assumed gaussian of extra sleep durations as possibly differing between drugs? It is reasonable to expect that the spread may also vary between drugs. Previously, we had a linear model of μ. It may not be obvious but we can also write models for σ.

The only complication is that standard deviations must be strictly positive. Therefore, it is useful to model it through a link function that makes it positive. Most common choice is a log-link, which we use here.

$$
\begin{aligned}
y_n &\sim N(\mu_n, \sigma_n^2) \\
\mu_n &= b_0 + b_1 D_n \\
\sigma_n &= \exp(t_0 + t_1 D_n)
\end{aligned}
$$

Priors

$$
\begin{aligned}
\mu &\sim \mathrm{N}(0, 10) \\
b_1 &\sim \mathrm{N}(0, 5) \\
t0 &\sim \mathrm{Cauchy}(0, 10) \\
t1 &\sim \mathrm{Cauchy}(0, 5)
\end{aligned}
$$

```
data {
  int N;
  vector[N] y;
  vector[N] x;
}

parameters {
  real b0;
  real b1;
  real t0;
  real t1;
}

model {
  vector[N] mu;
  vector[N] sigma;

  b0 ~ normal(0, 10);
  b1 ~ normal(0, 5);
  t0 ~ cauchy(0, 10);
  t1 ~ cauchy(0, 5);

  for (n in 1:N) {
    mu[n] = b0 + b1*x[n];
    sigma[n] = exp(t0 + t1*x[n]);
    y[n] ~ normal(mu[n], sigma[n]);
  }
}

generated quantities {
  // Transform t0 and t1 to standard deviations of each drug
  real sigma_Da = exp(t0);
  real sigma_Db = exp(t0 + t1);
}
```

```
fit3 <- stan(
  "m3.stan",
  data = datalist
)
```

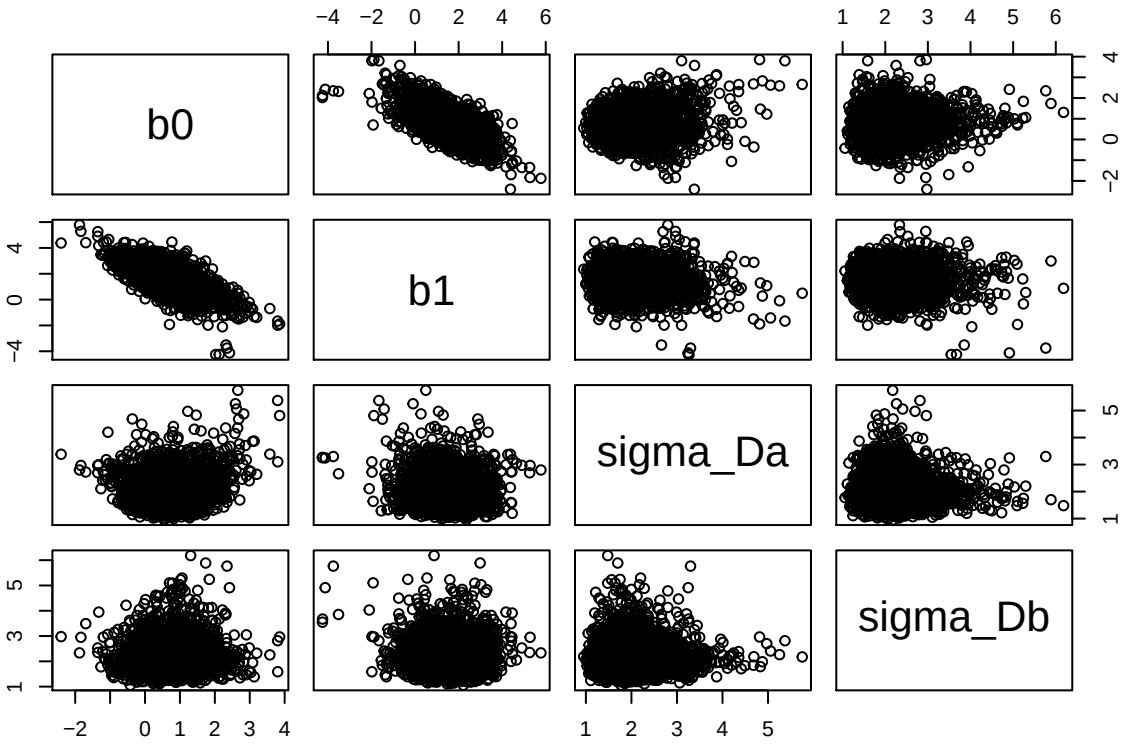
```
recompiling to avoid crashing R session
```

```
print(fit3, probs = c(.05, .5, .95))
```

```
Inference for Stan model: m3.
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     5%    50%    95% n_eff Rhat
```

```
b0          0.78    0.02 0.63  -0.19   0.77   1.83  1522    1
b1          1.51    0.02 0.94  -0.02   1.51   3.02  1503    1
t0          0.64    0.01 0.25   0.26   0.62   1.09  1767    1
t1          0.11    0.01 0.35  -0.47   0.11   0.67  1966    1
sigma_Da    1.95    0.01 0.53   1.30   1.85   2.97  1548    1
sigma_Db    2.17    0.01 0.57   1.46   2.06   3.23  3020    1
lp__      -23.92    0.04 1.57 -26.95 -23.52 -22.14  1235    1

Samples were drawn using NUTS(diag_e) at Wed Apr 10 13:40:15 2019.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

```
plot(as.data.frame(fit3)[,c("b0", "b1", "sigma_Da", "sigma_Db")])
```



Previously, we were able to point out the equivalent `lm()` formulas. Is there one for this? What classical "test" does it roughly correspond to?

4.5 Posteriors of other quantities

Consider the standardized effect size metric of the difference of two means

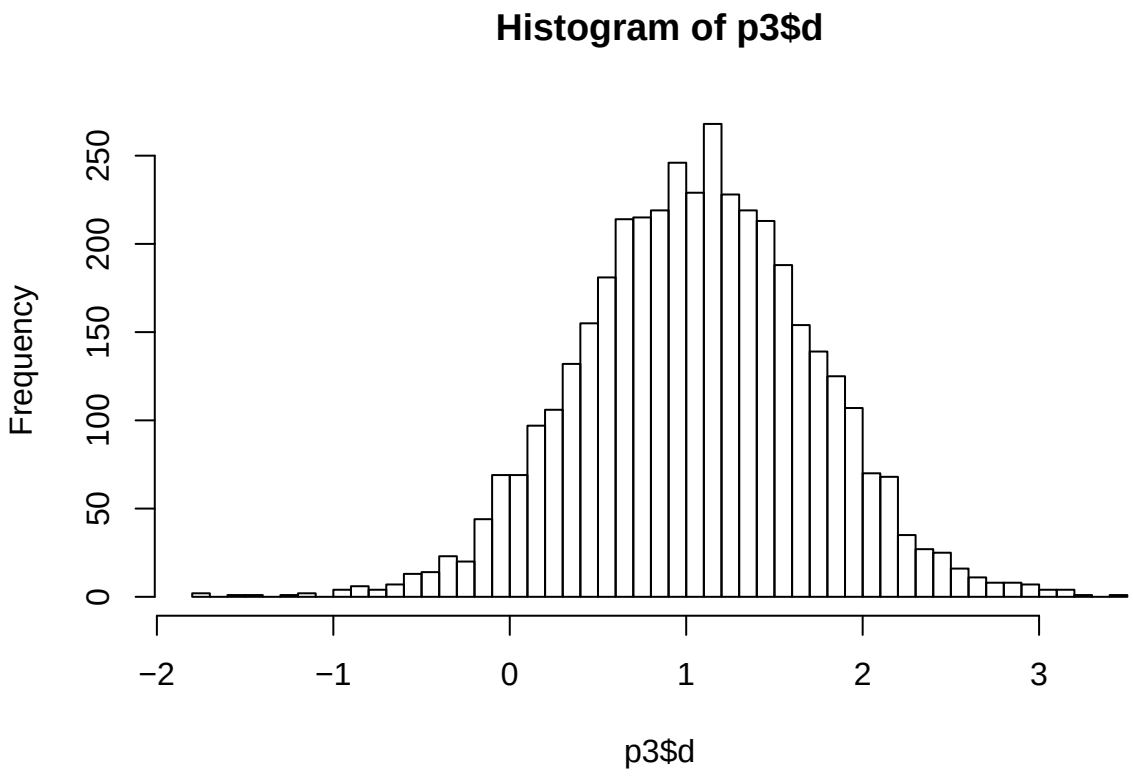
$$d = \frac{\mu_a - \mu_b}{\sqrt{(\sigma_a^2 + \sigma_b^2)/2}}$$

A point estimate of this is typically reported. But as is with means, raw effects, etc, we should not accept reporting of quantities of interest without some representation of their corresponding uncertainty.

It is easy to calculate posteriors for QOIs, like d, given our posterior samples:

```
p3 <- as.data.frame(fit3)
p3$mua <- p3$b0
```

```r
p3$mub <- p3$b0 + p3$b1

p3$d <- with(p3, (mub - mua) / (sqrt(sigma_Da^2 + sigma_Db^2) / 2))
hist(p3$d, breaks = 50)
```



5 Cool things we couldn't cover

5.1 Multilevel models

Can extend this approach to hierarchically structured data

No more Hessian errors or variances of 0!

5.2 Custom models

(Almost) no limit to the types of models and parameters available to you.

E.g. Reinforcement Learning models can be very easily evaluated in Stan.

Out of luck in (g)lm(er)

5.3 Model comparison

Can generate distributions of your favorite Information Criteria!

Express uncertainty in our model comparisons.

5.4 Model checking

Everything we've done so far assumes the truth of the underlying model.

Models are not true!

With Stan, we can generate new data, examine the ways our actual data don't match it, and hopefully build a less-untrue model.