PGS Analyses

Thomas Ward

2021-08-29

```r
library(magrittr)
library(dplyr)
library(readr)
library(forcats)
library(tidyr)
library(stringr)
library(ggplot2)
library(rethinking)
library(ggdist)
library(showtext)
font_add_google("Lato")
set_ulam_cmdstan(TRUE)
theme_set(theme_light(base_family = "Lato"))
showtext_auto()
histo_color = "#4C0099"
```

Prepare modeling data

Load

```r
dat <- readr::read_csv("../data/chole_pgs.csv", col_types = "iiiddddddli")
skimr::skim_without_charts(dat)
```

Table 1: Data summary

Name	dat
Number of rows	179
Number of columns	10
Column type frequency:	
logical	1
numeric	9
Group variables	None

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
gb_hole	0	1	0.27	FAL: 131, TRU: 48

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
videoid	0	1.00	90.00	51.82	1.00	45.50	90.00	134.50	179.00
surgid	0	1.00	9.61	7.07	1.00	4.00	8.00	17.00	23.00
pgs	0	1.00	2.46	1.21	1.00	1.00	2.00	3.00	5.00
time_until_1st_clip	0	1.00	15.53	12.42	1.07	5.57	13.00	21.15	60.92
time_cvs_attained	117	0.35	17.51	10.78	3.02	9.59	14.70	23.31	52.95
laparascopic_duration	0	1.00	42.87	25.52	5.62	24.50	38.53	60.90	131.60
dissection_duration	0	1.00	16.08	13.20	1.07	6.15	13.00	21.05	63.22
gb_removal_duration	1	0.99	6.87	5.16	0.82	3.16	5.64	9.31	27.22
gb_holes	0	1.00	0.35	0.66	0.00	0.00	0.00	1.00	4.00

Keep surgeons with 5 or more cases

Arrange rows by surgeons number of cases (surgeons with most cases will be at the top, fewest cases at the bottom). Then replace the `videoid` with a new sequential integer id based on this order. This will allow us to drop surgeons with few cases and then still have a nice sequential id to join back on to during analysis of results:

```
dat <- dat %>%
    mutate(surgid = as.integer(fct_infreq(as.character(surgid)))) %>%
    arrange(surgid) %>%
    mutate(videoid = 1:nrow(.))
```

Keep surgeons with 5 or more cases:

```
dat <- dat %>%
    group_by(surgid) %>%
    filter(n() >= 5) %>%
    ungroup()
```

```
nsurgs <- max(dat$surgid)
nsurgs
```

```
## [1] 10
```

Prepare data for ulam/stan

Duration analyses

We performed analyses on the log scale given we were concerned about the magnitude of duration and that factors that increase duration tend to do so exponentially. For example, an inflamed gallbladder is harder to grasp, but also harder to dissect, and combining the two together, you have an exponential increase in time.

We also standardize our variables, as it assists with making weakly regularizing priors:

```
log_duration <- log(dat$laparascopic_duration)
mean_log_duration <- mean(log_duration)
centered_log_duration <- log_duration - mean_log_duration
sd_log_duration <- sd(centered_log_duration)
std_log_duration <- centered_log_duration / sd_log_duration
```

Put list of data for modeling together. Note the `alpha`, which is our dirichlet prior (prior of 2 for PGS2, PGS3, PGS4, and PGS5):

```
ddat <- list(
    duration = std_log_duration,
    sid = dat$surgid,
    pgs = dat$pgs,
    alpha = rep(2, 4)
)
```

GB hole analyses

```
gdat <- list(
    hole = as.integer(dat$gb_hole),
    sid = dat$surgid,
    pgs = dat$pgs,
    alpha = rep(2, 4)
)
```

CVS attainment analyses

Analyses data is nearly ready, except for `dat$time_cvs_attained`.

We want to look at the binomial outcome, of whether or not CVS was achieved. To do so, just need to transform the data, so if it's `NA` then the CVS was never achieved.

```
cvs <- as.integer(!is.na(dat$time_cvs_attained))
```

Put the data together for stan:

```
cdat <- list(
    cvs = cvs,
    sid = dat$surgid,
    pgs = dat$pgs,
    alpha = rep(2, 4)
)
```

Utility functions

standard error

```
std_err  <- function(xs) {
    sd(xs) / sqrt(length(xs))
}
```

De-standardize (back to original scale)

Used to move from the standardized/centered log scale to normal scale to facilitate analyses:

```
unstd <- function(x) {
    exp(x * sd_log_duration + mean_log_duration)
}
```

Find rows with high pareto k

```
high_k_rows <- function(results) {
    results %>%
```

```r
        as_tibble(rownames = "videoid") %>%
        mutate(videoid = as.integer(videoid)) %>%
        inner_join(dat, by = "videoid") %>%
        filter(k >= 0.5)
}
```

samples from stan model

This is a thin wrapper around `rethinking` packages `extract.samples()` that returns a tibble with `janitor` fixing column names and a sample number given to each sample.

```r
extract_samples <- function(..., seed = 1234) {
    set.seed(seed)
    extract.samples(...) %>%
        as_tibble() %>%
        # case = "none" to not mess up capitalization of our parameters
        # otherwise fixes brackets, commas, and other problem chars
        janitor::clean_names(case = "none") %>%
        mutate(sample_num = paste0("sample", 1:nrow(.)), .before = 1L)
}
```

tidy surgeons

Each sample has the info for one or more surgeons on each row. We want this info "tidy", that is, one per row, so make a function to do that:

```r
tidy_surgeons <- function(df) {
    df %>%
        pivot_longer(
            starts_with(c("a_", "bP")),
            names_to = c(".value", "surgeon"),
            names_sep = "_"
        )
}
```

tidy pgs

Each sample, when using pgs levels, has 5 tidy rows of info, 1 for each PGS. So need to pivot these longer:

```r
tidy_pgs <- function(df) {
    df %>%
        pivot_longer(
            starts_with("PGS"),
            names_to = c(NA, "PGS"),
            names_pattern = "(PGS)([1-5])",
            names_transform = list("PGS" = as.integer),
            # to keep consistent with var name in model definition
            values_to = "sum_delta_j"
        )
}
```

sum deltas

We treat PGS as an ordered categorical predictor following McElreath's strategy in "Statistical Rethinking", Chapter 12, section 4. This assigns a proportion of the maximum value (PGS5) to each of the other PGS's.

PGS1 is absorbed into the intercept.

The below function just calculates this proportion to make later calculations easier.

```
sum_deltas <- function(df) {
    df %>% mutate(
        PGS1 = 0,
        PGS2 = delta_1,
        PGS3 = PGS2 + delta_2,
        PGS4 = 1 - delta_4,
        PGS5 = 1
    )
}
```

Plot half eye

This is a wrapper over `stat_halfeye()` that will provide a fill for both the distribution and the scale with labels for percents. Color palette is purple gradients.

```
halfeye <- function(df, xcol, ycol) {
    ggplot(df, aes(y = {{ ycol }}, x = {{ xcol }})) +
    stat_halfeye(
        aes(
            fill = stat(
                cut_cdf_qi(cdf, .width = c(0.5, 0.8, 0.95), labels = scales::percent_format())
            )
        ),
        .width = c(0.5, 0.8, 0.95)
    ) +
    scale_fill_brewer(direction = -1, palette = "Purples", na.translate = FALSE) +
    labs(fill = "Compatibility\nIntervals")
}
```

Compatibility intervals

Quickly calculate a bunch of compatibility intervals and pretty format them:

```
ci_ints  <- function(df, variable) {
    summarise(df,
        mean = mean({{ variable }}),
        low_50 = quantile({{ variable }}, probs = 0.25),
        high_50 = quantile({{ variable }}, probs = 0.75),
        low_66 = quantile({{ variable }}, probs = 0.17),
        high_66 = quantile({{ variable }}, probs = 0.83),
        low_80 = quantile({{ variable }}, probs = 0.20),
        high_80 = quantile({{ variable }}, probs = 0.80),
        low_89 = quantile({{ variable }}, probs = 0.055),
        high_89 = quantile({{ variable }}, probs = 0.945),
        low_95 = quantile({{ variable }}, probs = 0.025),
        high_95 = quantile({{ variable }}, probs = 0.975)
    ) %>%
    ungroup() %>%
    mutate(across(where(is.numeric), ~ round(., 2))) %>%
    unite(int_50, contains("_50"), sep = ", ") %>%
    unite(int_66, contains("_66"), sep = ", ") %>%
    unite(int_80, contains("_80"), sep = ", ") %>%
```

```r
    unite(int_89, contains("_89"), sep = ", ") %>%
    unite(int_95, contains("_95"), sep = ", ")
}
```

Abbreviations in formulas

- `sid`: Deidentified surgeon id
- `PGS`: Parkland Grading Scale for gallbladder inflammation
- `MVNormal`: Multivariate normal distribution
- `LKJCorr`: Lewandowski, Kurowicka, and Joe Correlation Distribution
- `CVS`: Critical View of Safety

Duration Analysis

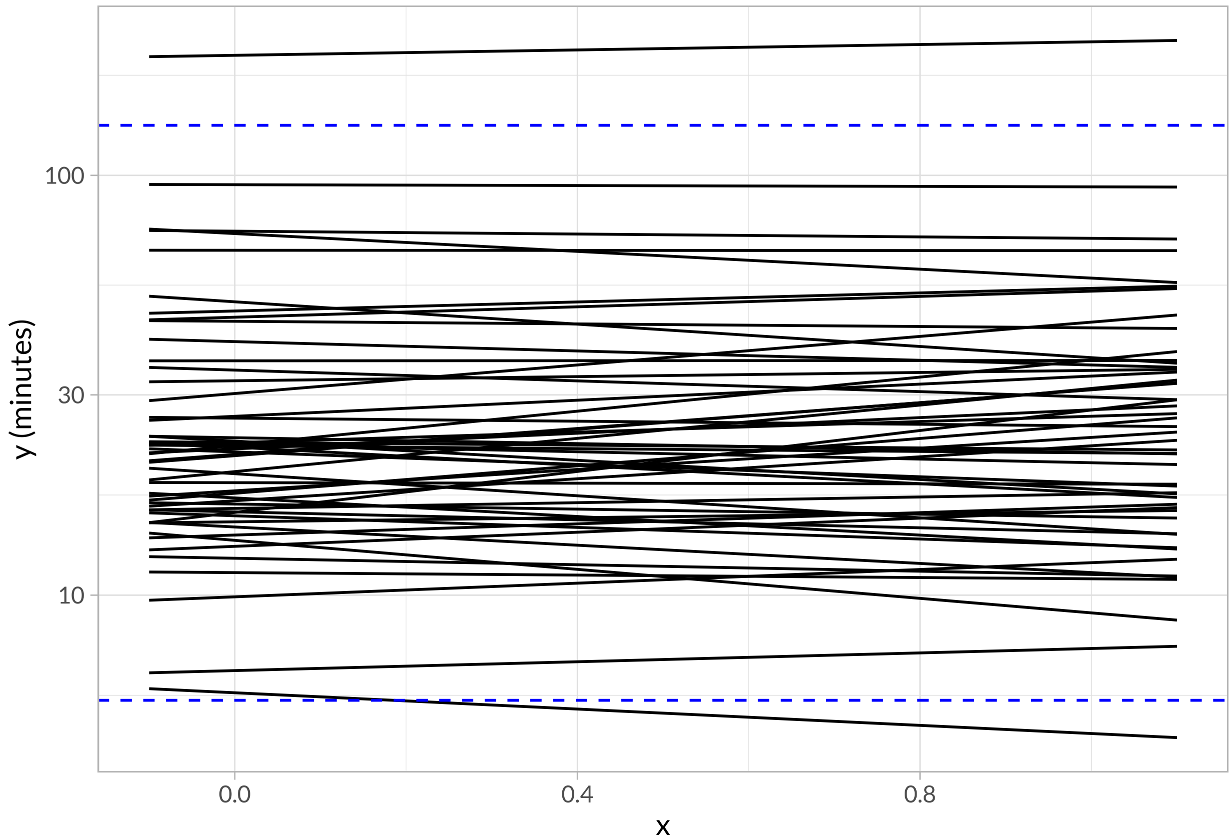
Priors determination

We will use weakly regularizing priors, that is, those that constrain parameters to those that are logically possible, while still allowing for some implausibly strong relationships if that is what the data determines.

I will graph simulated values of the priors to help determine good ones to use:

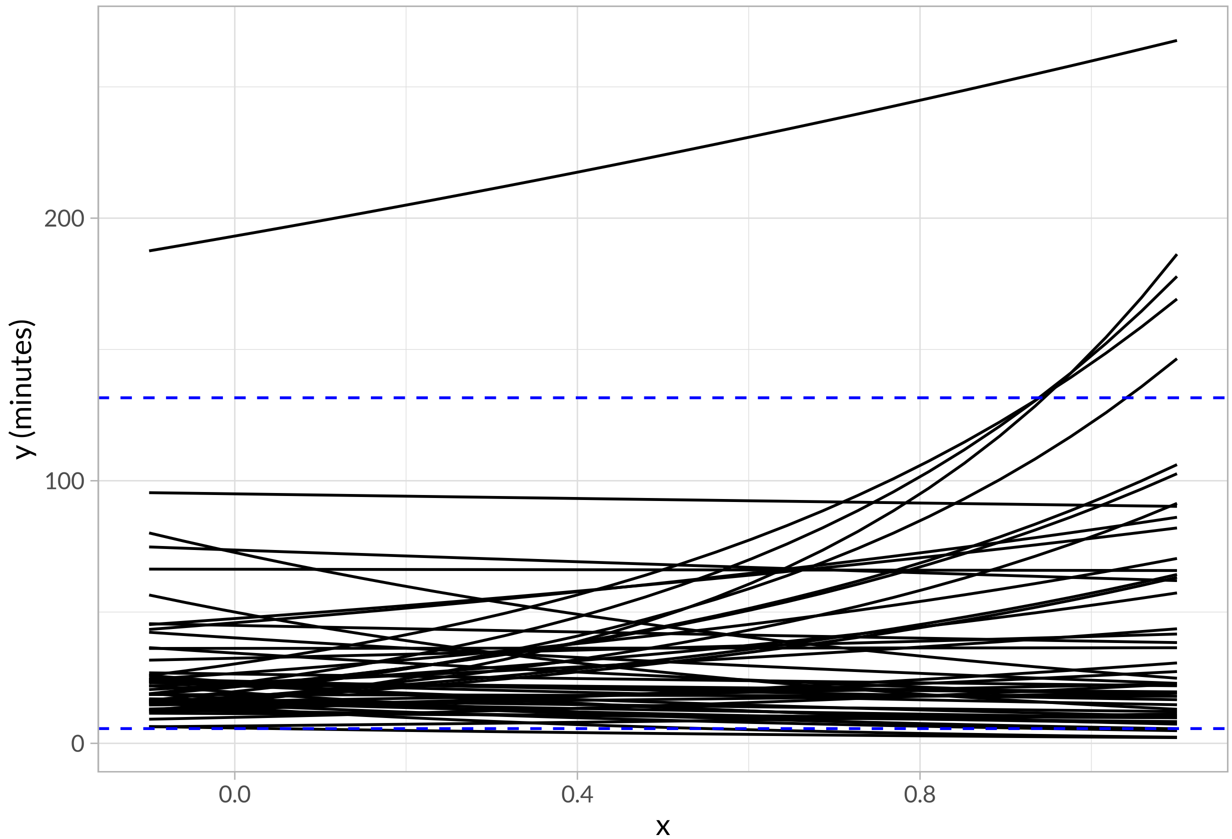
Try a prior of N(0, 1) for the intercept (`a`) and N(0, 0.3) for the slope for PGS5, `bp`:

```r
set.seed(1234)
max_dur <- max(ddat$duration)
min_dur <- min(ddat$duration)
tibble(
    sample_num = paste0("sample", 1:50),
    a = rnorm(50, mean = 0, sd = 1),
    bp = rnorm(50, mean = 0, sd = 0.3)
) %>%
    mutate(x = list(seq(from = -0.1, to = 1.1, length.out = 30))) %>%
    tidyr::unnest(x) %>%
    mutate(y = a + bp * x, og_y = unstd(y)) %>%
    ggplot(aes(x, og_y, group = sample_num)) +
    geom_line() +
    scale_y_log10() +
    labs(y = "y (minutes)") +
    geom_hline(yintercept = unstd(c(min_dur, max_dur)), linetype = 2, color = "blue")
```



The intercept prior looks acceptable. The slope prior is much too tight (remember slope is for a PGS5, so the maximum effect). The min/max of data never even reached. Let's try making it a bit bigger:

```
set.seed(1234)
max_dur <- max(ddat$duration)
min_dur <- min(ddat$duration)
tibble(
    sample_num = paste0("sample", 1:50),
    a = rnorm(50, mean = 0, sd = 1),
    bp = rnorm(50, mean = 0, sd = 1.2)
) %>%
    mutate(x = list(seq(from = -0.1, to = 1.1, length.out = 30))) %>%
    tidyr::unnest(x) %>%
    mutate(y = a + bp * x, og_y = unstd(y)) %>%
    ggplot(aes(x, og_y, group = sample_num)) +
    geom_line() +
    #scale_y_log10() +
    #labs(y = "y (log scale, minutes)") +
    labs(y = "y (minutes)") +
    geom_hline(yintercept = unstd(c(min_dur, max_dur)), linetype = 2, color = "blue")
```



The numbers are now constrained to mostly realistic values.

Other priors will be the usual weakly regularizing ones, including Dirichlet of 2, Exponential 1, and LKJCorr of 4.

Formula

Below is the centered version. The model given to Stan is the non-centered version that is mathematically equivalent but dramatically improves sampling.

$$
\begin{aligned}
log(Duration_i) &\sim Normal(\mu_i, \sigma) \\
\mu_i &= \alpha_{sid[i]} + \beta_{sid[i]} * \sum_{j=0}^{PGS_i - 1} \delta_j \\
\begin{bmatrix} \alpha_{sid} \\ \beta_{sid} \end{bmatrix} &\sim MVNormal(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \mathbf{S}) \\
\alpha &\sim Normal(0, 1) \\
\beta &\sim Normal(0, 1.2) \\
\delta &\sim Dirichlet(2) \\
\mathbf{S} &= \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \mathbf{R} \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \\
\mathbf{R} &= \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \\
\mathbf{R} &\sim LKJCorr(4) \\
\sigma, \sigma_\alpha, \sigma_\beta &\sim Exponential(1)
\end{aligned} \tag{1}
$$

Code

```
set.seed(1234)
dur_mod <- ulam(
    alist(
        duration ~ normal(mu, sigma),
        mu <- a_bar + ab_sid[sid, 1] + (bP_bar + ab_sid[sid, 2]) * sum(delta_j[1:pgs]),
        transpars> matrix[sid, 2]:ab_sid <-
             compose_noncentered(sigma_sid, L_Rho_sid, z_sid),
        matrix[2, sid]:z_sid ~ normal(0, 1),
        a_bar ~ normal(0, 1),
        bP_bar ~ normal(0, 1.2),
        vector[2]:sigma_sid ~ exponential(1),
        sigma ~ exponential(1),
        cholesky_factor_corr[2]:L_Rho_sid ~ lkj_corr_cholesky(4),
        vector[5]:delta_j <<- append_row(0, delta),
        simplex[4]:delta ~ dirichlet(alpha),
        # compute correlation matrix from Cholesky matrix
        gq> matrix[2, 2]: Rho_sid <<- Chol_to_Corr(L_Rho_sid),
        # for our analysis sake, compute a[sid] and b[sid]
        gq> vector[sid]:a <<- a_bar + ab_sid[, 1],
        gq> vector[sid]:bP <<- bP_bar + ab_sid[, 2]
    ),
    data = ddat,
    cores = 4,
    chains = 4,
    iter = 5000,
    log_lik = TRUE
)
```

```
## This is cmdstanr version 0.3.0

## - Online documentation and vignettes at mc-stan.org/cmdstanr

## - CmdStan path set to: /home/thomas/.cmdstanr/cmdstan-2.26.1
```

```
## - Use set_cmdstan_path() to change the path
##
## A newer version of CmdStan is available. See ?install_cmdstan() to install it.
## To disable this check set option or environment variable CMDSTANR_NO_VER_CHECK=TRUE.
## Compiling Stan program...
## -\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/- Ru
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 3 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 4 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 3 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 4 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 3 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 4 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 3 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 4 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 3 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 4 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 3 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 4 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 4 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 3 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 4 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 3 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 3 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 4 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 3 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 4 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Warmup)
```

```
## Chain 3 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 4 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 4 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 3 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 4 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 3 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 3 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 4 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 3 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 4 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 4 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 3 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 4 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 3 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 4 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 3 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 4 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 4 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 3 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 3 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 4 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 3 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 4 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 3 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 4 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 1 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 3 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 3 Iteration: 2300 / 5000 [ 46%]  (Warmup)
```

```
## Chain 4 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 4 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 2 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 3 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 4 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 4 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 3 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 3 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 4 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 3 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 3 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 4 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 3 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 4 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 3 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 4 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 4 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 3 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 4 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 3 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 3 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 4 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 3 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 4 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 3 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 4 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 3 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 3 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 4 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 4 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
```

```
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 3 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 4 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 3 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 4 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 3 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 4 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 3 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 3 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 4 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 3 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 4 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 4 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 3 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 4 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 3 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 4 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 3 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 3 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 4 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 3 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 4 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 3 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 4 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 4 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 3 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 3 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 4 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 finished in 4.2 seconds.
## Chain 3 finished in 4.2 seconds.
```

```
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 4 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 4 finished in 4.3 seconds.
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 finished in 4.4 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 4.3 seconds.
## Total execution time: 4.5 seconds.
```

Diagnostic Evaluation of Markov Chains

Rhat4 and effective sampling size

```r
precis(dur_mod, depth = 3)
```

```
##                        mean         sd        5.5%      94.5%     n_eff
## z_sid[1,1]     -2.384040295 0.64845693 -3.46047155 -1.3887519  4361.301
## z_sid[1,2]      0.526393679 0.37258315 -0.05185201  1.1411565  3233.090
## z_sid[1,3]     -0.196054682 0.34787152 -0.74475582  0.3629053  3283.831
## z_sid[1,4]      0.743951570 0.40404040  0.11449700  1.4060161  3517.962
## z_sid[1,5]     -0.121013920 0.38091790 -0.72746068  0.4919521  3743.610
## z_sid[1,6]     -0.230778330 0.39064474 -0.86389002  0.3867039  4171.463
## z_sid[1,7]      0.282479789 0.43019798 -0.39529988  0.9795907  4550.611
## z_sid[1,8]      0.648260842 0.44872450 -0.04838142  1.3932141  4447.475
## z_sid[1,9]      0.012495293 0.45714921 -0.71428326  0.7310225  4835.161
## z_sid[1,10]     0.736366008 0.45693052  0.04007889  1.4858520  4238.050
## z_sid[2,1]     -0.188884195 0.91739099 -1.64864895  1.2756814 11575.454
## z_sid[2,2]      0.343427499 0.86451672 -1.07709830  1.6953510  9601.130
## z_sid[2,3]     -0.103317120 0.88550879 -1.51337615  1.3156256 12563.761
## z_sid[2,4]      0.366442393 0.91754956 -1.16928100  1.8118154 11928.553
## z_sid[2,5]     -0.570701866 0.98628381 -2.10178525  1.0583522 10010.370
## z_sid[2,6]      0.119371030 0.91779743 -1.32280940  1.5911848 13768.783
## z_sid[2,7]     -0.029551626 0.95685800 -1.55359445  1.5153637 12634.192
## z_sid[2,8]     -0.006542448 0.99020980 -1.60811240  1.5483340 13602.784
## z_sid[2,9]      0.092697914 0.94128684 -1.42897290  1.5773290 12587.717
## z_sid[2,10]     0.100754564 0.95920028 -1.45515890  1.6221550 13120.354
## a_bar           0.078315842 0.24502259 -0.30479893  0.4641564  3025.395
## bP_bar          0.525659390 0.18897720  0.23208460  0.8236595  8272.162
## sigma_sid[1]    0.740994701 0.20399831  0.48811979  1.0992365  4714.022
## sigma_sid[2]    0.230295577 0.19106328  0.01804434  0.5787149  5171.965
## sigma           0.492707031 0.02953328  0.44816751  0.5413854 11622.999
## L_Rho_sid[1,1]  1.000000000 0.00000000  1.00000000  1.0000000       NaN
## L_Rho_sid[1,2]  0.000000000 0.00000000  0.00000000  0.0000000       NaN
## L_Rho_sid[2,1]  0.060022547 0.31384906 -0.44226001  0.5565246 12951.443
## L_Rho_sid[2,2]  0.944709241 0.07369466  0.80007415  0.9997231  3846.376
## delta[1]        0.196725932 0.10723971  0.04812869  0.3827334 11843.450
## delta[2]        0.258804358 0.12415547  0.08142931  0.4770108 11882.380
## delta[3]        0.302800703 0.13683920  0.09792895  0.5322698 12468.829
## delta[4]        0.241669012 0.12747727  0.05962633  0.4685847 13088.522
## bP[1]           0.446410234 0.26111763  0.01871364  0.8462773 10784.322
## bP[2]           0.620028919 0.20316337  0.31698957  0.9640038 10063.920
## bP[3]           0.498760160 0.22936905  0.13005026  0.8536246 12062.550
## bP[4]           0.639706544 0.25936135  0.26889784  1.0847010  9940.633
```

```
## bP[5]           0.347814625 0.34991123 -0.28751296  0.7852630  6497.632
## bP[6]           0.557224206 0.26923184  0.15688216  0.9945579  9822.049
## bP[7]           0.519682475 0.29183425  0.05966005  0.9573360  9073.491
## bP[8]           0.529620630 0.32998362  0.03607356  1.0077398  8659.421
## bP[9]           0.557250023 0.29183105  0.13178997  1.0110399  8494.040
## bP[10]          0.566535486 0.30895248  0.10406067  1.0517094 10190.231
## a[1]           -1.583054846 0.11740468 -1.76952275 -1.3955567 11001.569
## a[2]            0.444448537 0.12686933  0.23652517  0.6421926 10553.103
## a[3]           -0.059114236 0.10674821 -0.23049965  0.1137655 12680.428
## a[4]            0.597097399 0.14475824  0.35701904  0.8206876 10545.440
## a[5]           -0.006984282 0.16152566 -0.26077277  0.2542622 11058.199
## a[6]           -0.084064937 0.17829539 -0.36779120  0.1984511 13050.039
## a[7]            0.276363836 0.23111632 -0.08897297  0.6377499 11705.923
## a[8]            0.535090894 0.24211732  0.15167483  0.9162728 11661.807
## a[9]            0.087360977 0.26259003 -0.33214508  0.4956021 10718.759
## a[10]           0.596570234 0.23151126  0.22871259  0.9644537 13036.584
## Rho_sid[1,1]    1.000000000 0.00000000  1.00000000  1.0000000       NaN
## Rho_sid[1,2]    0.060022547 0.31384906 -0.44226001  0.5565246 12951.443
## Rho_sid[2,1]    0.060022547 0.31384906 -0.44226001  0.5565246 12951.443
## Rho_sid[2,2]    1.000000000 0.00000000  1.00000000  1.0000000       NaN
## ab_sid[1,1]    -1.661370708 0.25887463 -2.07702200 -1.2573134  3262.760
## ab_sid[1,2]    -0.079249148 0.23262053 -0.48389211  0.2463126 11377.196
## ab_sid[2,1]     0.366132700 0.25997624 -0.04014196  0.7845014  3350.392
## ab_sid[2,2]     0.094369531 0.19892059 -0.15844152  0.4576320  7184.016
## ab_sid[3,1]    -0.137430073 0.25730204 -0.54035649  0.2642420  3228.425
## ab_sid[3,2]    -0.026899230 0.20543773 -0.36813397  0.2818660 10592.281
## ab_sid[4,1]     0.518781563 0.27092806  0.08806932  0.9460820  3566.606
## ab_sid[4,2]     0.114047162 0.23811935 -0.16820897  0.5595589  8248.674
## ab_sid[5,1]    -0.085300116 0.27842923 -0.52605934  0.3578331  3606.977
## ab_sid[5,2]    -0.177844761 0.30259357 -0.76474788  0.1274416  6720.732
## ab_sid[6,1]    -0.162380778 0.28241711 -0.61705793  0.2837209  3843.780
## ab_sid[6,2]     0.031564815 0.22866791 -0.30383671  0.4197998 10104.012
## ab_sid[7,1]     0.198048005 0.30921125 -0.29264917  0.6896935  4362.857
## ab_sid[7,2]    -0.005976903 0.24165363 -0.39010508  0.3577238  9685.792
## ab_sid[8,1]     0.456775033 0.31616834 -0.03566903  0.9716484  4650.984
## ab_sid[8,2]     0.003961250 0.27642614 -0.39676551  0.4074769  9475.762
## ab_sid[9,1]     0.009045131 0.33133023 -0.51664892  0.5314908  4665.404
## ab_sid[9,2]     0.031590649 0.24464680 -0.30414098  0.4361623  9120.277
## ab_sid[10,1]    0.518254401 0.31394342  0.02934193  1.0175772  4378.663
## ab_sid[10,2]    0.040876102 0.25994092 -0.33061215  0.4617071 10584.545
##                    Rhat4
## z_sid[1,1]     1.0009547
## z_sid[1,2]     1.0000485
## z_sid[1,3]     0.9999519
## z_sid[1,4]     1.0003597
## z_sid[1,5]     1.0000571
## z_sid[1,6]     0.9999865
## z_sid[1,7]     0.9998480
## z_sid[1,8]     1.0000571
## z_sid[1,9]     0.9997443
## z_sid[1,10]    1.0002478
## z_sid[2,1]     0.9998427
## z_sid[2,2]     0.9997749
## z_sid[2,3]     0.9998908
```

```
## z_sid[2,4]      0.9997793
## z_sid[2,5]      1.0002381
## z_sid[2,6]      0.9996765
## z_sid[2,7]      0.9997916
## z_sid[2,8]      1.0000862
## z_sid[2,9]      1.0000990
## z_sid[2,10]     0.9997940
## a_bar           1.0000781
## bP_bar          1.0000207
## sigma_sid[1]    1.0012668
## sigma_sid[2]    1.0007773
## sigma           0.9998212
## L_Rho_sid[1,1]        NaN
## L_Rho_sid[1,2]        NaN
## L_Rho_sid[2,1] 1.0002752
## L_Rho_sid[2,2] 1.0010717
## delta[1]        0.9998563
## delta[2]        0.9998311
## delta[3]        1.0000443
## delta[4]        0.9998103
## bP[1]           1.0000889
## bP[2]           0.9999250
## bP[3]           0.9999165
## bP[4]           0.9997010
## bP[5]           1.0005906
## bP[6]           0.9998104
## bP[7]           1.0001544
## bP[8]           1.0005158
## bP[9]           1.0001844
## bP[10]          1.0001504
## a[1]            0.9999226
## a[2]            0.9999620
## a[3]            1.0004889
## a[4]            0.9998053
## a[5]            1.0000256
## a[6]            0.9998460
## a[7]            1.0000033
## a[8]            1.0000659
## a[9]            0.9997889
## a[10]           1.0000340
## Rho_sid[1,1]          NaN
## Rho_sid[1,2]    1.0002752
## Rho_sid[2,1]    1.0002752
## Rho_sid[2,2]          NaN
## ab_sid[1,1]     1.0001677
## ab_sid[1,2]     1.0000533
## ab_sid[2,1]     0.9999082
## ab_sid[2,2]     1.0004184
## ab_sid[3,1]     1.0000298
## ab_sid[3,2]     0.9999735
## ab_sid[4,1]     0.9999604
## ab_sid[4,2]     0.9999812
## ab_sid[5,1]     1.0001224
## ab_sid[5,2]     1.0003042
```

```
## ab_sid[6,1]    1.0000758
## ab_sid[6,2]    0.9998335
## ab_sid[7,1]    1.0000718
## ab_sid[7,2]    1.0000265
## ab_sid[8,1]    0.9997919
## ab_sid[8,2]    1.0002419
## ab_sid[9,1]    0.9997953
## ab_sid[9,2]    1.0002789
## ab_sid[10,1]   0.9999335
## ab_sid[10,2]   1.0006268
```

All Rhat4 values are 1.

Each parameter also sampled well.

PSIS/WAIC

```r
PSIS(dur_mod)
```

```
## Some Pareto k values are high (>0.5). Set pointwise=TRUE to inspect individual points.
```

```
##       PSIS      lppd  penalty  std_err
## 1 232.3422 -116.1711 13.90847 17.11978
```

```r
WAIC(dur_mod)
```

```
##       WAIC      lppd  penalty  std_err
## 1 231.7685 -102.2626 13.62158 16.99175
```

The are some Pareto k values > 0.5. As long as < 0.7, not an issue. Are they?

```r
PSIS(dur_mod, pointwise = TRUE) %>%
    high_k_rows()
```

```
## Some Pareto k values are high (>0.5). Set pointwise=TRUE to inspect individual points.
```

```
## # A tibble: 2 x 15
##   videoid  PSIS  lppd penalty std_err     k surgid   pgs time_until_1st_c~
##     <int> <dbl> <dbl>   <dbl>   <dbl> <dbl>  <int> <int>             <dbl>
## 1     146  5.93 -2.96   0.949    17.1 0.579      9     4              7.85
## 2     151  2.48 -1.24   0.327    17.1 0.563     10     2             12.6
## # ... with 6 more variables: time_cvs_attained <dbl>,
## #   laparascopic_duration <dbl>, dissection_duration <dbl>,
## #   gb_removal_duration <dbl>, gb_hole <lgl>, gb_holes <int>
```

Only 2 rows total, and both have k < 0.6, so minimal issue with outliers.

Trace rank plot (trankplot)

```r
trankplot(dur_mod)
```

Good mixing as shown with large amount of overlap.

Trace plot

```r
traceplot(dur_mod)
```

Chains are stationary with a visible central tendency, have good mixing, and converge. Excellent.

Evaluate model results

List of the parameters I care about from the model:

1. Intercepts (`a[N]` and average `a_bar`)
2. beta for pgs5 (`bP[N]` and average `bP_bar`)
3. Correlation (`Rho_sid[1, 2]`)
4. sigmas (overall, for intercept, for beta)
5. deltas (for incremental effect of each PGS)

```
pars <- c(
    "a_bar",
    paste0("a[", 1:nsurgs, "]"),
    "bP_bar",
    paste0("bP[", 1:nsurgs, "]"),
    # sigma for entire model
    "sigma",
    # sigma_a
    "sigma_sid[1]",
    # sigma_b
    "sigma_sid[2]",
    paste0("Rho_sid[1,2]"),
    paste0("delta[", 1:4, "]")
)
```

Now time to extract samples from stan:

```
dur_samples <- extract_samples(dur_mod, pars = pars) %>%
    # change names for ease of remembering rather than having to
    # remember their position in the matrices
    rename(sigma_a = sigma_sid_1, sigma_bP = sigma_sid_2, rho = Rho_sid_1_2) %>%
    sum_deltas()
```

Now, let us generate a "new" unseen surgeon to see what the model will predict for a new surgeon, make the data tidy (one row per surgeon per PGS per sample), then calculate the outcome from the samples to see the model's predictions.

```
tidy_dur_samples <- dur_samples %>%
    # generates a "new" unseen surgeon
    mutate(
        a_new = rnorm(nrow(.), a_bar, sigma_a),
        bP_new = rnorm(nrow(.), bP_bar, sigma_bP)
    ) %>%
    select(sample_num, starts_with(c("a_", "bP", "PGS")), sigma) %>%
    tidy_surgeons() %>%
    tidy_pgs() %>%
    mutate(mu = a + bP * sum_delta_j, mu_og = unstd(mu)) %>%
    select(sample_num, surgeon, PGS, mu, mu_og, sigma)
```

Plot results

Was duration correlated with starting time?

```
extract_samples(dur_mod, pars = c("Rho_sid[1,2]")) %>%
    rename(rho = Rho_sid_1_2) %>%
    ci_ints(rho) %>%
    knitr::kable(
```

```
        caption = "Correlation between time to perform PGS1 case and the effect of incrementing PGS on l
    )
```

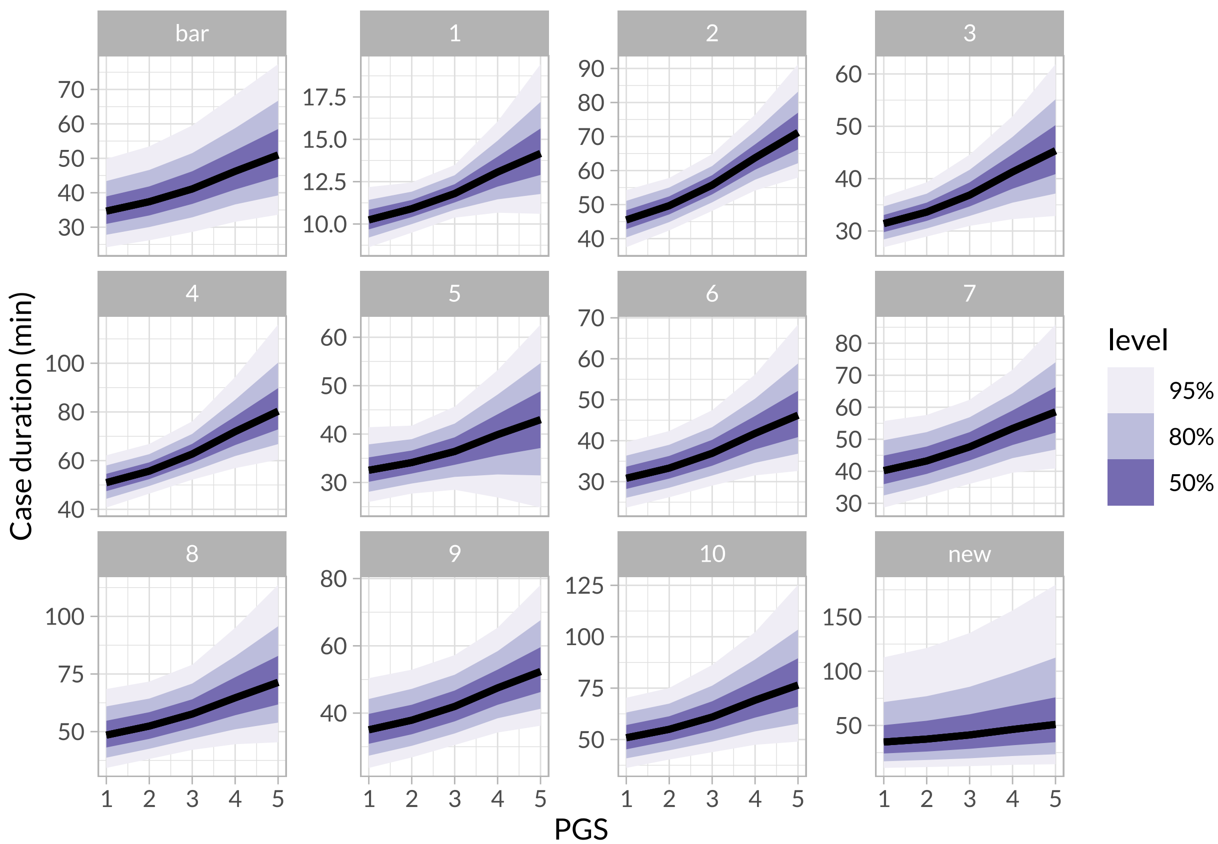
Table 4: Correlation between time to perform PGS1 case and the effect of incrementing PGS on laparoscopic duration

mean	int_50	int_66	int_80	int_89	int_95
0.06	-0.16, 0.29	-0.26, 0.37	-0.22, 0.34	-0.44, 0.56	-0.54, 0.65

For every surgeon

Visualization of the effects PGS has on each surgeon. Surgeon `bar` is the average surgeon, and surgeon `new` is for a future never seen surgeon.

```
tidy_dur_samples %>%
    mutate(surgeon = parse_factor(surgeon, levels = c("bar", as.character(1:nsurgs), "new"))) %>%
    ggplot(aes(PGS, mu_og)) +
    stat_lineribbon() +
    facet_wrap(~ surgeon, scales = "free_y") +
    scale_fill_brewer(labels = c("95%", "80%", "50%"), palette = "Purples") +
    labs(y = "Case duration (min)")
```



Calculate deltas

Calculate the change from a PGS1 each PGS level has.

First, who is affected the most and the least by PGS? This will manifest by the smallest and the largest `bP`:

```
dur_bps <- precis(dur_mod, depth = 2, pars = paste0("bP[", 1:nsurgs, "]")) %>%
    as_tibble(rownames = "var")
# Least affected surgeon:
slice_min(dur_bps, mean)
```

```
## # A tibble: 1 x 7
##   var     mean    sd `5.5%` `94.5%` n_eff Rhat4
##   <chr>  <dbl> <dbl>  <dbl>   <dbl> <dbl> <dbl>
## 1 bP[5]  0.348 0.350 -0.288   0.785 6498.  1.00
```

```
# Most affected surgeon:
slice_max(dur_bps, mean)
```

```
## # A tibble: 1 x 7
##   var     mean    sd `5.5%` `94.5%` n_eff Rhat4
##   <chr>  <dbl> <dbl>  <dbl>   <dbl> <dbl> <dbl>
## 1 bP[4]  0.640 0.259  0.269    1.08 9941.  1.00
```

So the most affected surgeon is Surgeon "4" and the least affected surgeon is Surgeon "5".

Now actually calculate the deltas:

```
dur_pgs_deltas <- tidy_dur_samples %>%
    # selecting our least, average, and most affected surgeons
    filter(surgeon %in% c("bar", "4", "5")) %>%
    mutate(
        surgeon = case_when(
            surgeon == "4" ~ "Most affected",
            surgeon == "5" ~ "Least affected",
            surgeon == "bar" ~ "Average"
        ),
        surgeon = parse_factor(
            surgeon,
            levels = c("Least affected", "Average", "Most affected")
        )
    ) %>%
    # ensure correct order, with PGS1 as first row in future group
    # will rely on this ordering to calculate delta
    arrange(sample_num, surgeon, PGS) %>%
    group_by(sample_num, surgeon) %>%
    mutate(pgs_delta = mu_og - mu_og[1]) %>%
    ungroup() %>%
    # needed for ggdist to work
    mutate(PGS = as.factor(PGS))
```

On average across all surgeons

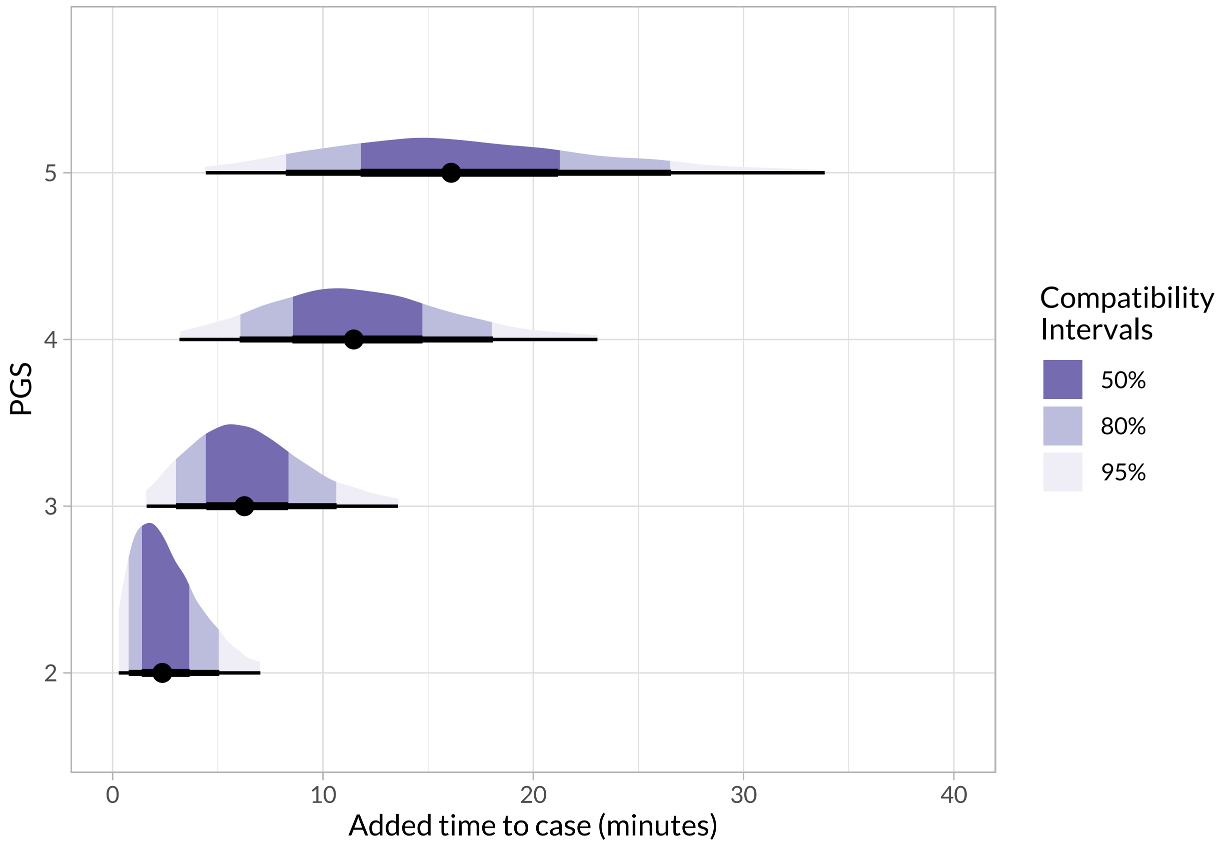
How was the average surgeon affected?

Then plot this:

```
dur_pgs_deltas_bar  <- filter(dur_pgs_deltas, surgeon == "Average", PGS != "1")
```

```
dur_pgs_deltas_bar %>%
    halfeye(pgs_delta, PGS) +
    coord_cartesian(xlim = c(0, 40)) +
```

```r
    labs(
        x = "Added time to case (minutes)",
        y = "PGS"
    )
```




```r
ggsave("../output/pgs_duration.svg", width = 6, height = 6)
ggsave("../output/pgs_duration.pdf", width = 6, height = 6)
```

What are the numbers of the compatibility intervals:

```r
dur_pgs_deltas_bar %>%
    group_by(PGS) %>%
    ci_ints(pgs_delta) %>%
    knitr::kable()
```

PGS	mean	int_50	int_66	int_80	int_89	int_95
2	2.70	1.38, 3.66	1.06, 4.3	1.18, 4.03	0.5, 5.9	0.29, 7.02
3	6.59	4.46, 8.35	3.76, 9.34	4.04, 8.93	2.38, 11.96	1.61, 13.57
4	11.87	8.55, 14.73	7.35, 16.24	7.82, 15.58	4.67, 20.37	3.17, 23.05
5	16.94	11.79, 21.18	10.06, 23.71	10.79, 22.64	6.49, 29.76	4.43, 33.86

All units are in minutes.

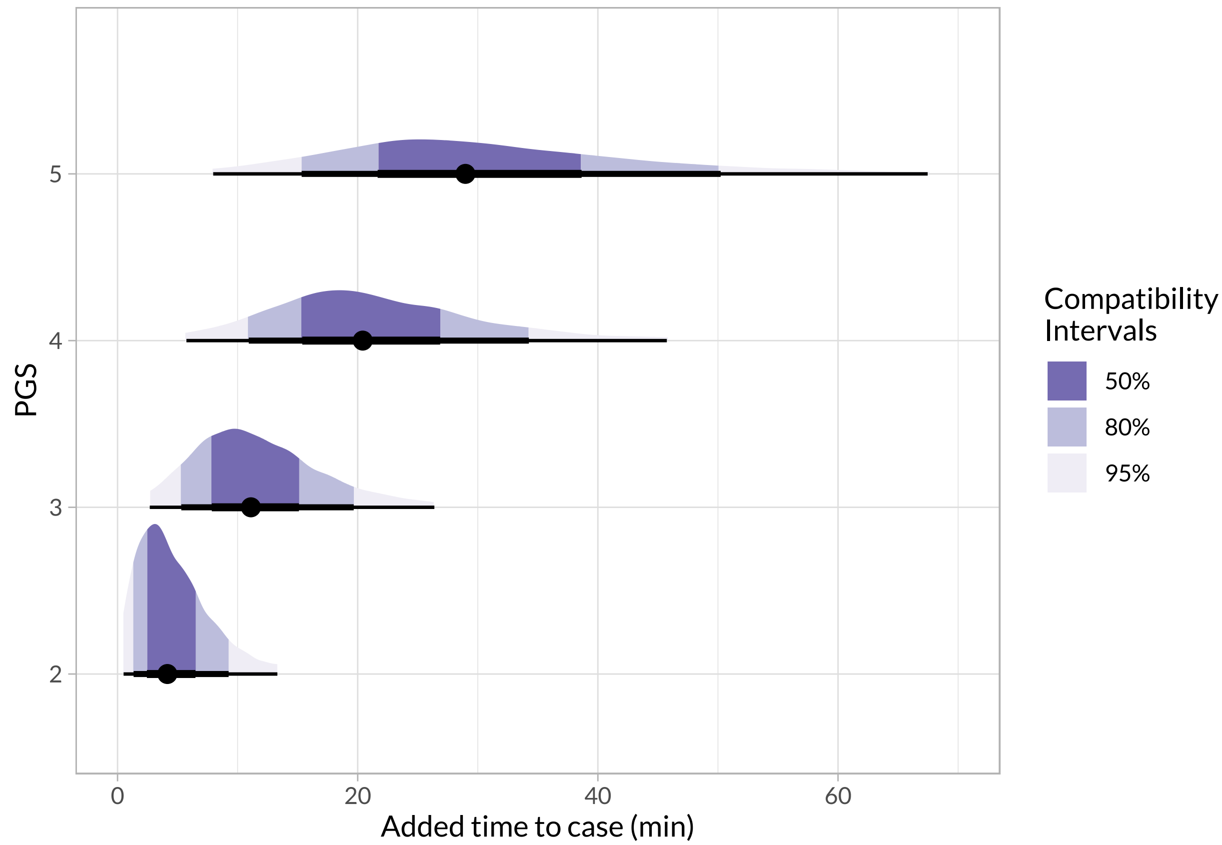
surgeon most affected

How much was the most affected surgeon, affected?

Then plot this:

```
dur_pgs_deltas_ma  <- filter(dur_pgs_deltas, surgeon == "Most affected", PGS != "1")
```

```
dur_pgs_deltas_ma %>%
    halfeye(pgs_delta, PGS) +
    coord_cartesian(xlim = c(0, 70)) +
    labs(
        x = "Added time to case (min)"
    )
```



What are the numbers of the compatibility intervals:

```
dur_pgs_deltas_ma %>%
    group_by(PGS) %>%
    ci_ints(pgs_delta) %>%
    knitr::kable()
```

PGS	mean	int_50	int_66	int_80	int_89	int_95
2	4.86	2.44, 6.49	1.88, 7.7	2.09, 7.2	0.87, 10.96	0.5, 13.31
3	11.98	7.85, 15.1	6.71, 17.17	7.17, 16.29	4.12, 22.67	2.68, 26.39
4	21.77	15.38, 26.87	13.27, 29.88	14.16, 28.6	8.45, 38.9	5.74, 45.74
5	31.30	21.66, 38.64	18.73, 43.54	19.88, 41.48	12.03, 57.46	7.97, 67.46

All units are in minutes.

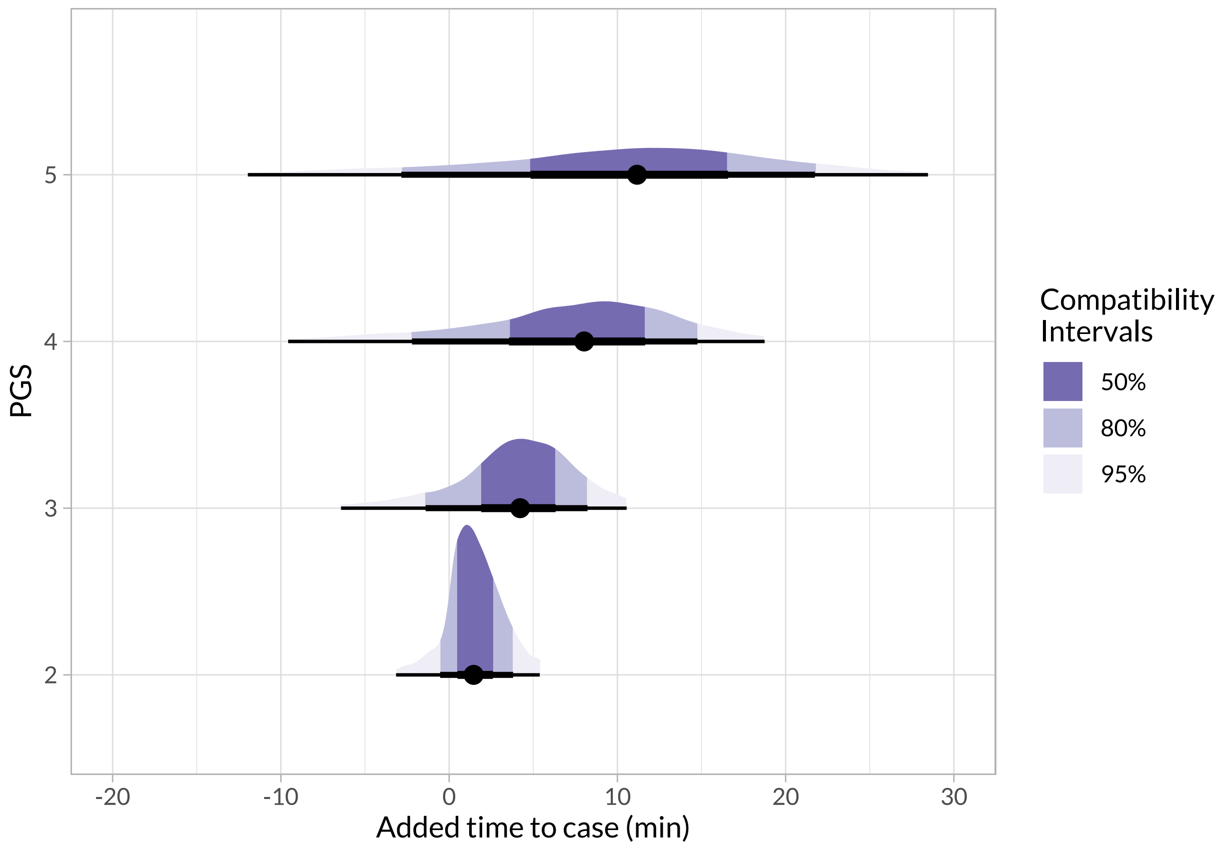
surgeon least affected

How much was the least affected surgeon, affected?

Then plot this:

```
dur_pgs_deltas_la  <- filter(dur_pgs_deltas, surgeon == "Least affected", PGS != "1")
```

```
dur_pgs_deltas_la %>%
    halfeye(pgs_delta, PGS) +
    coord_cartesian(xlim = c(-20, 30)) +
    labs(
        x = "Added time to case (min)"
    )
```



What are the numbers of the compatibility intervals:

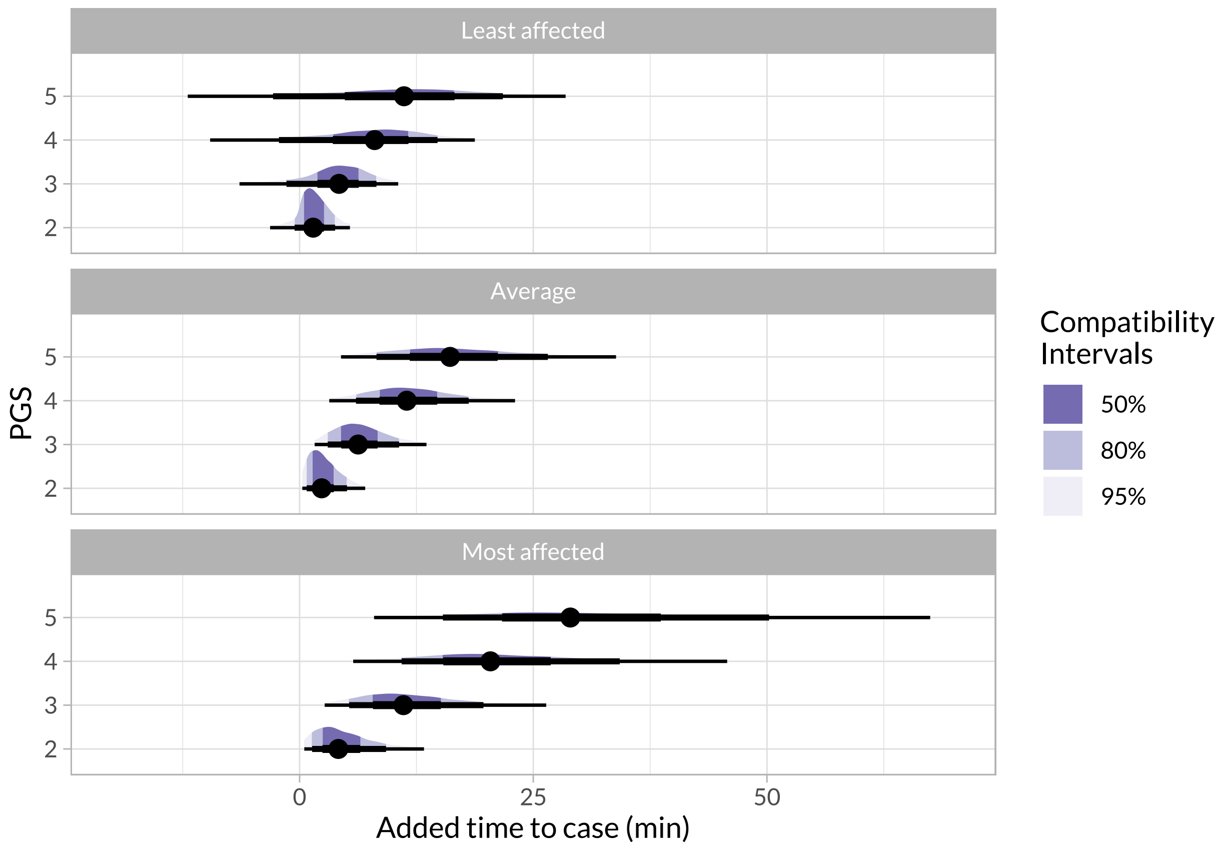
```
dur_pgs_deltas_la %>%
    group_by(PGS) %>%
    ci_ints(pgs_delta) %>%
    knitr::kable()
```

PGS	mean	int_50	int_66	int_80	int_89	int_95
2	1.46	0.5, 2.6	0.16, 3.13	0.3, 2.91	-1.55, 4.49	-3.15, 5.39
3	3.76	1.91, 6.33	0.64, 7.17	1.21, 6.83	-3.46, 9.34	-6.43, 10.55
4	7.07	3.56, 11.65	1.07, 13	2.12, 12.45	-5.4, 16.54	-9.57, 18.75
5	10.31	4.84, 16.58	1.44, 18.89	2.83, 17.89	-6.92, 24.43	-11.96, 28.46

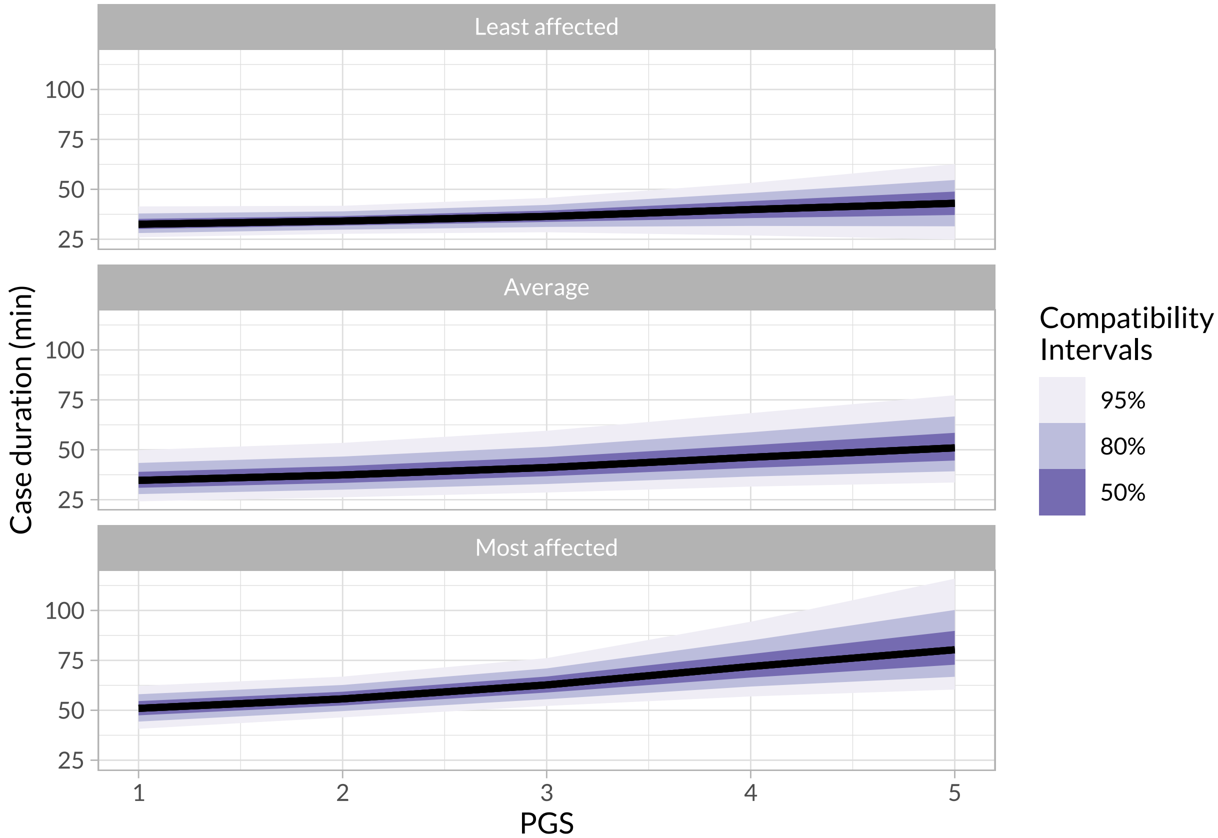
All units are in minutes.

all together now

```
dur_pgs_deltas %>%
    filter(PGS != "1") %>%
    halfeye(pgs_delta, PGS) +
    facet_wrap(~ surgeon, ncol = 1) +
    coord_cartesian(xlim = c(-20, 70)) +
    labs(
        x = "Added time to case (min)"
    )
```




```
dur_pgs_deltas %>%
    mutate(PGS = as.integer(PGS)) %>%
    ggplot(aes(PGS, mu_og)) +
    stat_lineribbon() +
    facet_wrap(~ surgeon, ncol = 1) +
    scale_fill_brewer(labels = c("95%", "80%", "50%"), palette = "Purples") +
    labs(y = "Case duration (min)", fill = "Compatibility\nIntervals")
```



Gallbladder Holes Analysis

We treated a gallbladder hole occurring during the removal of the gallbladder bed as a binomial outcome, that is, we cared if 1 or more holes occurred and treated those equal. We chose this because once a full-thickness perforation occurs, bile/stones spill. Adding further holes at this point is rather inconsequential, as what matters is going from no hole to a single hole.

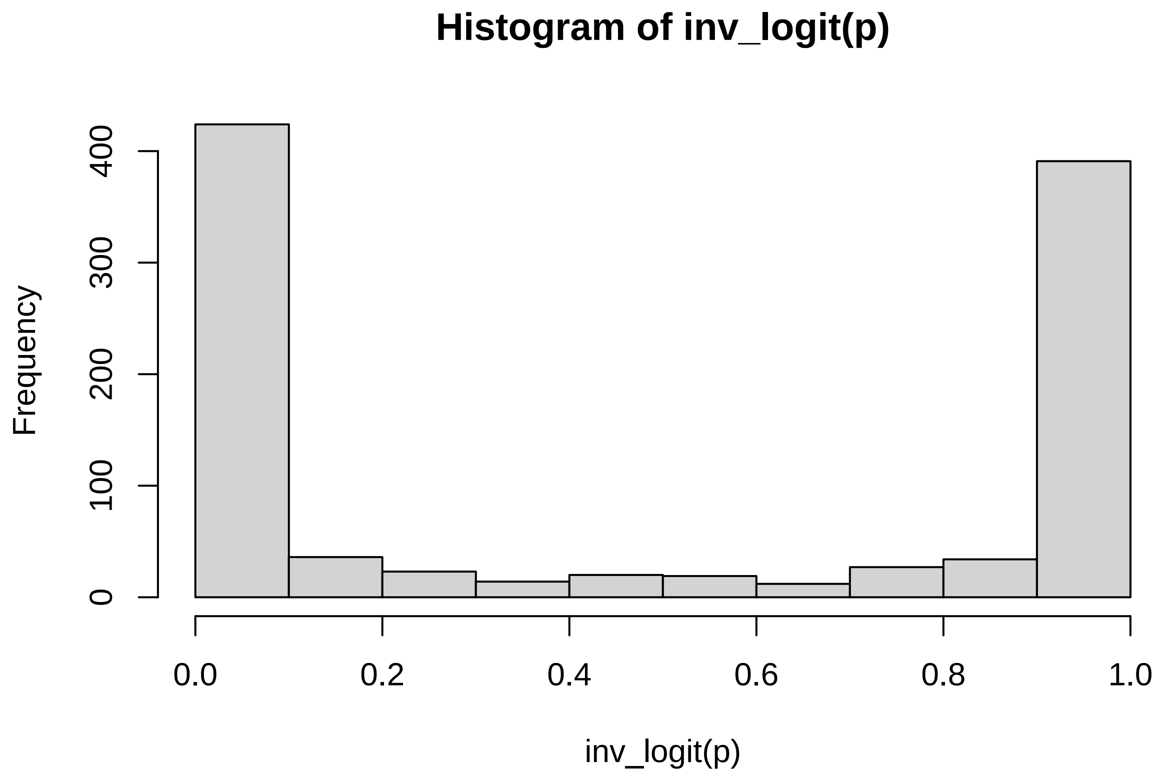
Priors determination

As before, we will use weakly regularizing priors.

Intercept

To see how `logit()` can mess up priors, look at a N(0, 10) intercept prior:

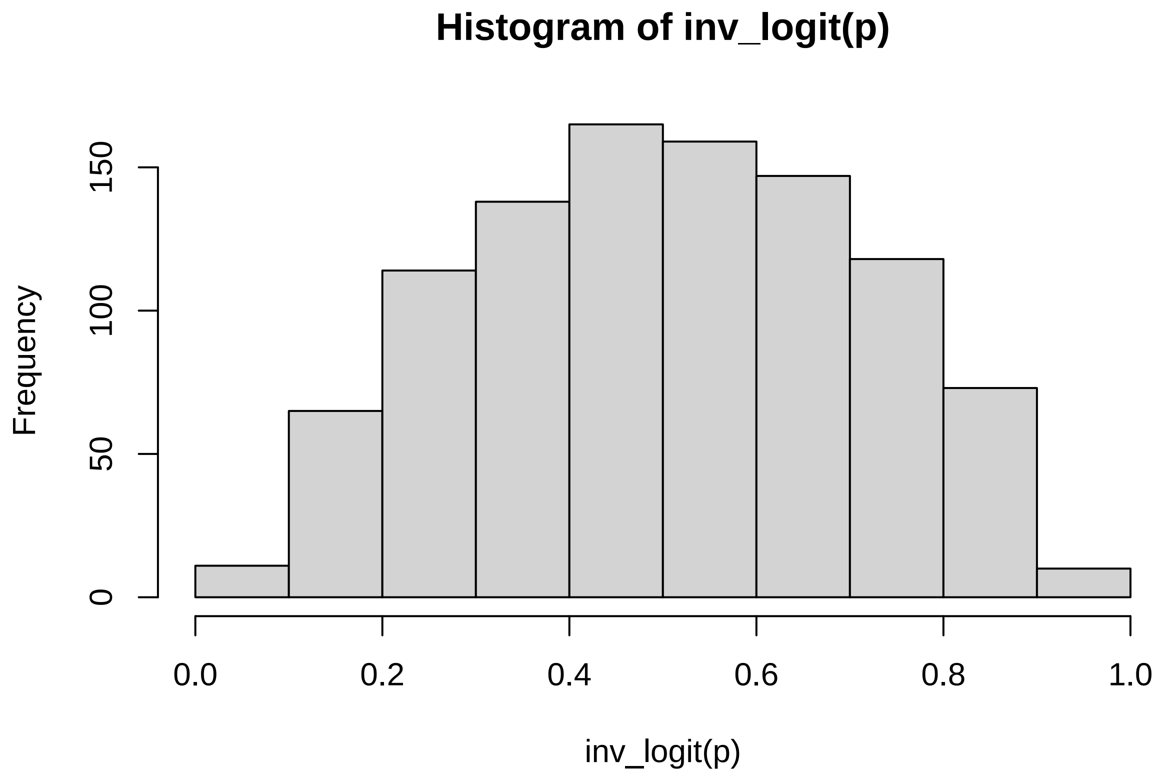
```
p <- rnorm(1000, 0, 10)
hist(inv_logit(p))
```



Nearly all the probability mass is at 0 or 1 for the prior, which is nonsensical.

Trying a N(0, 1) prior:

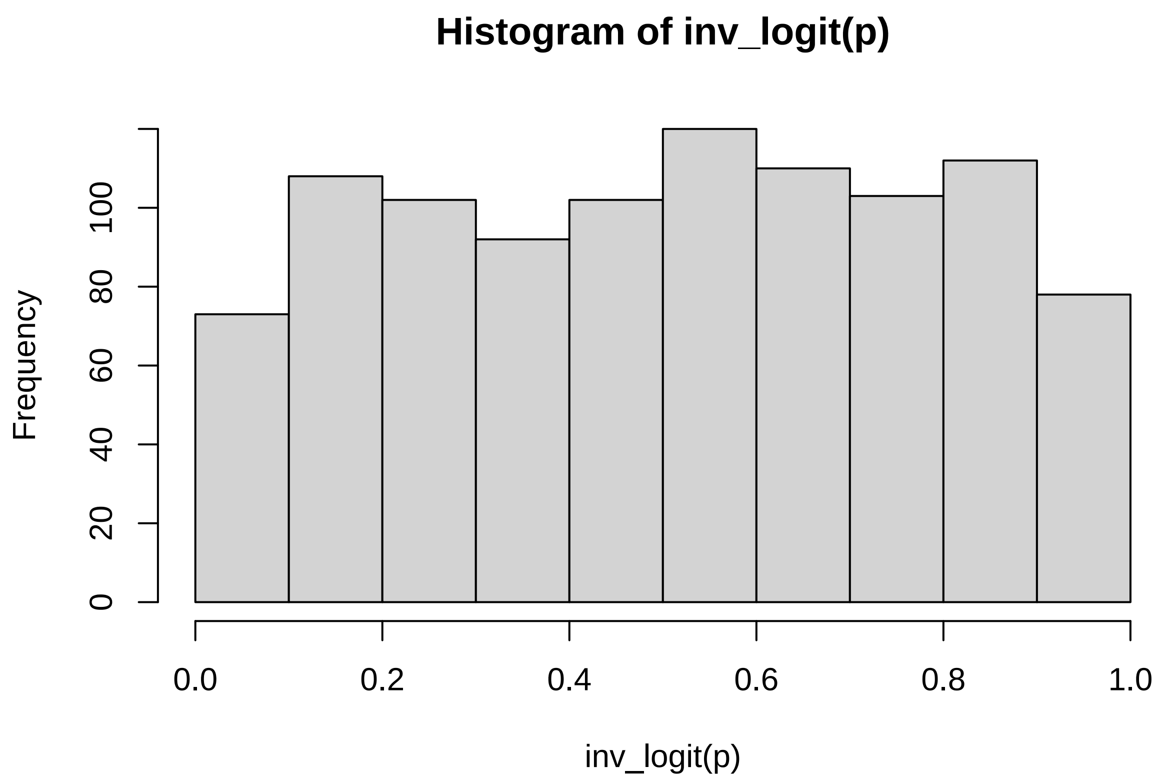
```
p <- rnorm(1000, 0, 1)
hist(inv_logit(p))
```



The bulk of the mass is too in the middle, that is, it is estimating the probability of a GB hole to be around 50%. a priori, that seems unlikely.

A N(0, 1.5) is much more even and the one we will use:

```
p <- rnorm(1000, 0, 1.5)
hist(inv_logit(p))
```

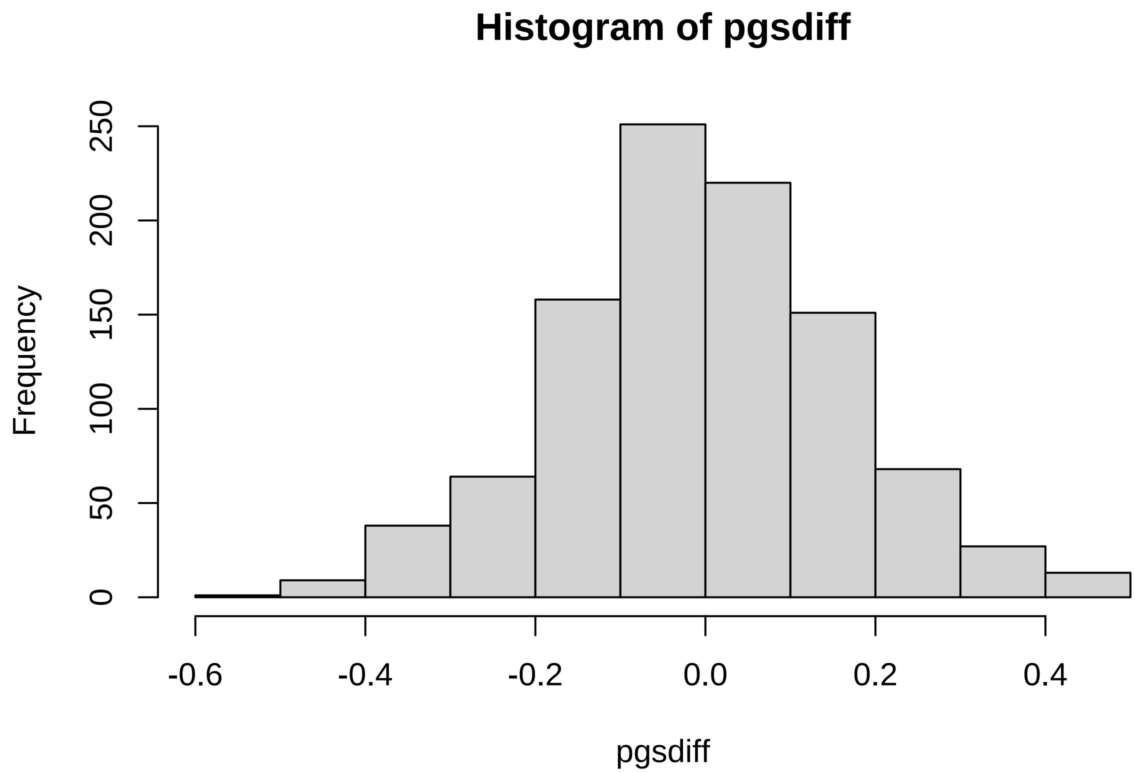


PGS coefficient

Remember, this is the effect of a PGS5. We will need to simulate the intercept then add the slope, followed by undoing the logit to see the probability:

A N(0, 1)

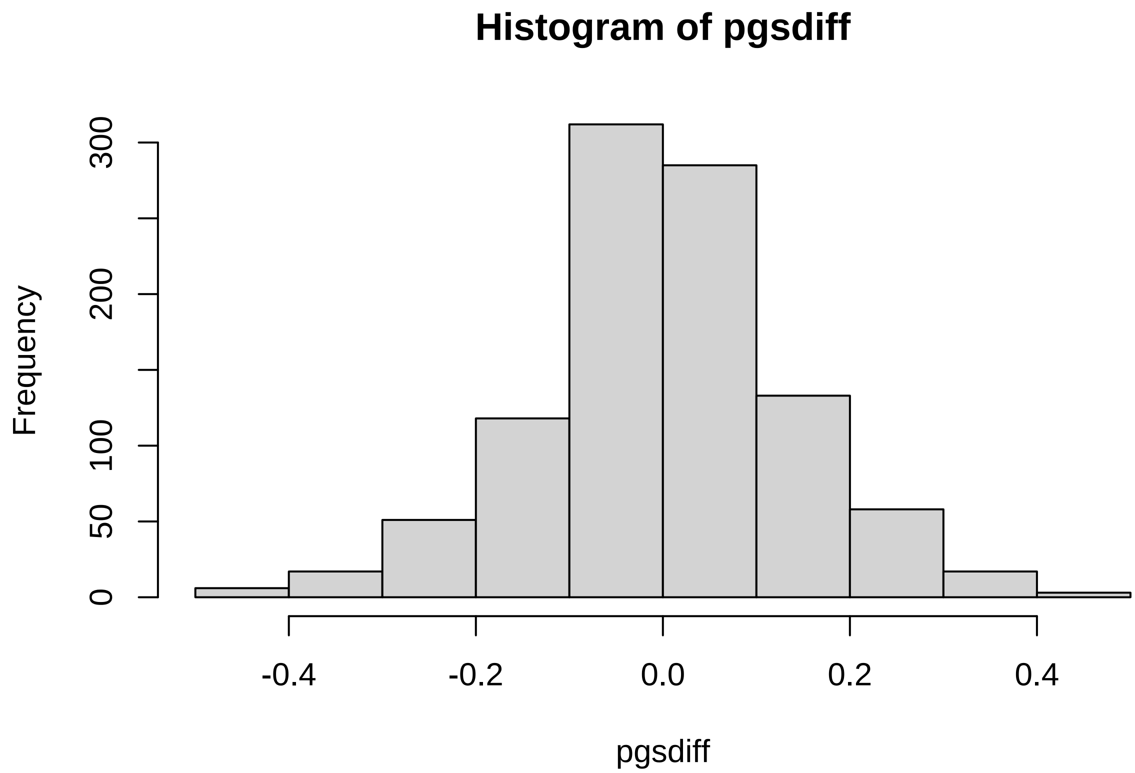
```
pint <- rnorm(1000, 0, 1.5)
ppgs <- rnorm(1000, 0, 1)
pgsdiff <- inv_logit(pint + ppgs) - inv_logit(pint)
hist(pgsdiff)
```



Is fairly reasonable (PGS5 does on average no change in getting a GB hole, but can change it by 60% probability in rare instances).

That seems a little too strong of an effect, so let's decrease with N(0, 0.75):

```
pint <- rnorm(1000, 0, 1.5)
ppgs <- rnorm(1000, 0, 0.75)
pgsdiff <- inv_logit(pint + ppgs) - inv_logit(pint)
hist(pgsdiff)
```



other priors

Other priors will be the usual weakly regularizing ones, including Dirichlet of 2, Exponential 1, and LKJCorr of 4.

Formula

Below is the centered version. The model given to Stan is the non-centered version that is mathematically equivalent but dramatically improves sampling.

$$
\begin{aligned}
Hole_i &\sim Bernoulli(p_i) \\
logit(p_i) &= \alpha_{sid[i]} + \beta_{sid[i]} * \sum_{j=0}^{PGS_i - 1} \delta_j \\
\begin{bmatrix} \alpha_{sid} \\ \beta_{sid} \end{bmatrix} &\sim MVNormal(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \mathbf{S}) \\
\alpha &\sim Normal(0, 1.5) \\
\beta &\sim Normal(0, 0.75) \\
\delta &\sim Dirichlet(2) \\
\mathbf{S} &= \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \mathbf{R} \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \\
\mathbf{R} &= \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \\
\mathbf{R} &\sim LKJCorr(4) \\
\sigma_\alpha, \sigma_\beta &\sim Exponential(1)
\end{aligned}
\tag{2}
$$

Code

```
set.seed(1234)
hole_mod <- ulam(
    alist(
        hole ~ bernoulli(p),
        logit(p) <- a_bar + ab_sid[sid, 1] + (bP_bar + ab_sid[sid, 2]) * sum(delta_j[1:pgs]),
        a_bar ~ normal(0, 1.5),
        bP_bar ~ normal(0, 0.75),
        vector[5]: delta_j <<- append_row(0, delta),
        simplex[4]: delta ~ dirichlet(alpha),
        transpars> matrix[sid, 2]: ab_sid <-
            compose_noncentered(sigma_sid, L_Rho_sid, z_sid),
        matrix[2, sid]: z_sid ~ normal(0, 1),
        vector[2]: sigma_sid ~ exponential(1),
        cholesky_factor_corr[2]: L_Rho_sid ~ lkj_corr_cholesky(4),
        gq> matrix[2, 2]: Rho_sid <<- Chol_to_Corr(L_Rho_sid),
        gq> vector[sid]:a <<- a_bar + ab_sid[, 1],
        gq> vector[sid]:bP <<- bP_bar + ab_sid[, 2]
    ),
    data = gdat,
    cores = 4,
    chains = 4,
    iter = 5000,
    log_lik = TRUE
)
```

```
## Compiling Stan program...
```

```
## -\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\| Runni
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
```

```
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 3 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 4 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 3 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 3 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 3 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 4 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 4 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 3 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 4 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 4 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 3 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 3 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 4 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 3 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 3 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 4 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 4 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 3 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 3 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 4 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 4 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 3 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 4 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 4 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 3 Iteration: 1300 / 5000 [ 26%]  (Warmup)
```

```
## Chain 3 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 4 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 4 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 3 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 3 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 4 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 4 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 3 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 3 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 4 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 4 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 3 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 3 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 4 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 3 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 4 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 4 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 3 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 3 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 4 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 4 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 1 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 2 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 3 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 3 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 3 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 4 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 3 Iteration: 2600 / 5000 [ 52%]  (Sampling)
```

```
## Chain 4 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 4 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 4 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 3 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 3 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 4 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 4 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 3 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 3 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 4 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 3 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 4 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 4 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 3 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 3 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 4 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 4 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 3 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 3 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 4 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 3 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 3 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 4 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 4 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 3 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 4 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 4 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 3 Iteration: 3900 / 5000 [ 78%]  (Sampling)
```

```
## Chain 3 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 4 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 3 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 3 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 4 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 4 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 3 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 4 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 4 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 3 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 3 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 4 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 3 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 3 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 4 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 4 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 3 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 4 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 4 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 3 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 3 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 4 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 finished in 3.0 seconds.
## Chain 2 finished in 2.9 seconds.
## Chain 3 finished in 3.0 seconds.
## Chain 4 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 4 finished in 3.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 3.0 seconds.
## Total execution time: 3.2 seconds.
```

Diagnostic Evaluation of Markov Chains

Rhat4 and effective sampling size

```r
precis(hole_mod, depth = 3)
```

```
##                        mean         sd        5.5%      94.5%     n_eff
## a_bar          -1.3826316500 0.31848139 -1.90273440 -0.8852383  8979.762
## bP_bar          0.5860863112 0.56080178 -0.32735928  1.4632966  9197.078
## delta[1]        0.1963108936 0.12032544  0.04255204  0.4174099 13942.335
## delta[2]        0.2960796980 0.15259443  0.07844509  0.5607019 12859.262
## delta[3]        0.2517295763 0.14119610  0.05996026  0.5003064 13981.086
## delta[4]        0.2558798319 0.14226006  0.05865852  0.5109675 13325.849
## z_sid[1,1]     -0.2680234545 0.84707966 -1.57988155  1.0960221 13389.775
## z_sid[1,2]      0.4428785263 0.91494280 -1.08626245  1.8223225  8565.378
## z_sid[1,3]      0.1404743506 0.86371689 -1.26426440  1.4953206 11537.128
## z_sid[1,4]     -0.4406078745 0.89892164 -1.84642425  1.0187180 12285.393
## z_sid[1,5]      0.0451315484 0.88610944 -1.38255100  1.4457191 16115.530
## z_sid[1,6]     -0.3144259405 0.93568743 -1.78824695  1.1856433 13722.441
## z_sid[1,7]      0.0026572858 0.93992277 -1.52948015  1.4780507 15456.629
## z_sid[1,8]      0.1596764919 0.93939233 -1.35336765  1.6614847 14951.069
## z_sid[1,9]      0.3262559676 0.98434474 -1.26689045  1.8541843 12670.376
## z_sid[1,10]    -0.3311636644 0.98888466 -1.88631695  1.2736729 14990.080
## z_sid[2,1]     -0.0137703779 0.86450538 -1.39452375  1.3764800 13618.729
## z_sid[2,2]      0.7225369725 0.87893525 -0.76879505  2.0615121  8494.220
## z_sid[2,3]     -0.5404047987 0.95252796 -2.03077895  1.0141886 11923.747
## z_sid[2,4]     -0.1671835743 0.91256035 -1.61525055  1.2915584 15271.088
## z_sid[2,5]     -0.0542666528 0.92549679 -1.54724060  1.4008160 15807.034
## z_sid[2,6]      0.0080325677 0.92640357 -1.48253045  1.4824769 13544.957
## z_sid[2,7]      0.0919204801 0.91020555 -1.39482765  1.5261582 14220.981
## z_sid[2,8]      0.2811958137 0.96989427 -1.29406300  1.8349766 14744.338
## z_sid[2,9]      0.5066919372 0.95003818 -1.05127485  1.9600038 12444.594
## z_sid[2,10]    -0.1933510552 0.96749125 -1.73644495  1.3588837 16637.308
## sigma_sid[1]    0.3678874557 0.29865146  0.03045718  0.9266509  5074.428
## sigma_sid[2]    0.8103188851 0.64316838  0.06166090  1.9784393  5034.385
## L_Rho_sid[1,1]  1.0000000000 0.00000000  1.00000000  1.0000000       NaN
## L_Rho_sid[1,2]  0.0000000000 0.00000000  0.00000000  0.0000000       NaN
## L_Rho_sid[2,1] -0.0256299181 0.32498198 -0.53956611  0.4973029 11833.618
## L_Rho_sid[2,2]  0.9424830887 0.07393621  0.79544456  0.9996901  5309.883
## bP[1]           0.6166678177 0.82824939 -0.69834796  1.8805989 10343.783
## bP[2]           1.2854067142 0.79149138  0.14505816  2.6461369  7764.960
## bP[3]          -0.0008982751 1.19111342 -2.16518825  1.4472370  7306.758
## bP[4]           0.4301040061 0.98189209 -1.25428810  1.7856643  9887.653
## bP[5]           0.5298460929 1.00643362 -1.12598080  1.9328967 10097.770
## bP[6]           0.6371909064 0.98080046 -0.89593169  2.0722067 10980.284
## bP[7]           0.7126906019 0.92445199 -0.72701333  2.1021274 12428.390
## bP[8]           0.9147067141 1.07995680 -0.57429527  2.6369995 10302.683
## bP[9]           1.1660275711 1.03873357 -0.21625851  2.9779732  9469.926
## bP[10]          0.3539717791 1.18717124 -1.67208630  1.8848414  9283.098
## a[1]           -1.4998808111 0.38872381 -2.15612210 -0.9283229 11298.882
## a[2]           -1.1847544972 0.41283226 -1.81172980 -0.4988825  9009.631
## a[3]           -1.3202538583 0.36199746 -1.89868430 -0.7429484 12326.175
## a[4]           -1.5954647520 0.47683464 -2.43332935 -0.9432220  7995.159
## a[5]           -1.3618487657 0.41966448 -2.04731440 -0.7180963 12922.930
## a[6]           -1.5434330775 0.50406777 -2.41351815 -0.8693691  9429.728
```

```
## a[7]           -1.3818445936 0.47788656 -2.12114660 -0.6750021 11025.689
## a[8]           -1.2960082544 0.48596934 -2.02055715 -0.5223752 11380.072
## a[9]           -1.2083199819 0.53304410 -1.95187165 -0.3068774  9219.834
## a[10]          -1.5645614981 0.56525156 -2.53296495 -0.8353756  8994.751
## Rho_sid[1,1]    1.0000000000 0.00000000  1.00000000  1.0000000       NaN
## Rho_sid[1,2]   -0.0256299181 0.32498198 -0.53956611  0.4973029 11833.618
## Rho_sid[2,1]   -0.0256299181 0.32498198 -0.53956611  0.4973029 11833.618
## Rho_sid[2,2]    1.0000000000 0.00000000  1.00000000  1.0000000       NaN
## ab_sid[1,1]    -0.1172491578 0.32711263 -0.69330797  0.3241707 10354.589
## ab_sid[1,2]     0.0305814860 0.70615294 -1.01004720  1.1915695 10798.797
## ab_sid[2,1]     0.1978771722 0.35856443 -0.24741457  0.8657138  6931.912
## ab_sid[2,2]     0.6993203966 0.81335084 -0.19849025  2.2116013  5276.416
## ab_sid[3,1]     0.0623778177 0.32044568 -0.40059408  0.6243177  9933.756
## ab_sid[3,2]    -0.5869845653 1.01341438 -2.49088730  0.4345075  7897.709
## ab_sid[4,1]    -0.2128330683 0.40330567 -0.96309798  0.2370936  7909.462
## ab_sid[4,2]    -0.1559823313 0.81901048 -1.52752265  0.9932486 11893.812
## ab_sid[5,1]     0.0207829060 0.34671726 -0.51554953  0.5833995 12662.388
## ab_sid[5,2]    -0.0562402390 0.85953494 -1.42155190  1.1727073 10685.694
## ab_sid[6,1]    -0.1608014557 0.42397913 -0.94423806  0.3492115  9204.745
## ab_sid[6,2]     0.0511046170 0.84817391 -1.20988545  1.4175699 11039.948
## ab_sid[7,1]     0.0007870842 0.39387131 -0.60889799  0.6047590 11938.298
## ab_sid[7,2]     0.1266042848 0.81509078 -1.01727775  1.4560045  9975.510
## ab_sid[8,1]     0.0866233637 0.40300116 -0.45738196  0.7835542 11373.189
## ab_sid[8,2]     0.3286203927 0.99651317 -0.85936993  2.0869241  8225.266
## ab_sid[9,1]     0.1743116289 0.45115193 -0.36216547  1.0119296  8484.258
## ab_sid[9,2]     0.5799412533 1.00453399 -0.48558835  2.4571680  7113.841
## ab_sid[10,1]   -0.1819298869 0.48537782 -1.04371630  0.3621447  9063.103
## ab_sid[10,2]   -0.2321145447 0.99522748 -1.87405685  1.0377873 10736.335
##                    Rhat4
## a_bar          0.9997749
## bP_bar         1.0001568
## delta[1]       0.9997256
## delta[2]       0.9999080
## delta[3]       0.9997599
## delta[4]       0.9999473
## z_sid[1,1]     0.9999703
## z_sid[1,2]     1.0002516
## z_sid[1,3]     0.9997692
## z_sid[1,4]     1.0000150
## z_sid[1,5]     0.9997936
## z_sid[1,6]     0.9999944
## z_sid[1,7]     0.9997764
## z_sid[1,8]     0.9998223
## z_sid[1,9]     0.9998046
## z_sid[1,10]    0.9998023
## z_sid[2,1]     1.0000344
## z_sid[2,2]     1.0003897
## z_sid[2,3]     0.9999215
## z_sid[2,4]     0.9998411
## z_sid[2,5]     0.9999645
## z_sid[2,6]     1.0000676
## z_sid[2,7]     0.9999433
## z_sid[2,8]     1.0000963
## z_sid[2,9]     0.9999750
```

```
## z_sid[2,10]    0.9996388
## sigma_sid[1]   1.0006535
## sigma_sid[2]   1.0008714
## L_Rho_sid[1,1]       NaN
## L_Rho_sid[1,2]       NaN
## L_Rho_sid[2,1] 0.9996744
## L_Rho_sid[2,2] 1.0001170
## bP[1]          1.0000179
## bP[2]          1.0003592
## bP[3]          1.0002064
## bP[4]          1.0002444
## bP[5]          1.0000277
## bP[6]          0.9999787
## bP[7]          0.9998517
## bP[8]          0.9999688
## bP[9]          1.0000127
## bP[10]         0.9999843
## a[1]           1.0000490
## a[2]           1.0002031
## a[3]           0.9998702
## a[4]           0.9999863
## a[5]           0.9999142
## a[6]           1.0003368
## a[7]           0.9999020
## a[8]           0.9999780
## a[9]           0.9999501
## a[10]          0.9997137
## Rho_sid[1,1]         NaN
## Rho_sid[1,2]   0.9996744
## Rho_sid[2,1]   0.9996744
## Rho_sid[2,2]         NaN
## ab_sid[1,1]    1.0001156
## ab_sid[1,2]    1.0002616
## ab_sid[2,1]    1.0003290
## ab_sid[2,2]    1.0005997
## ab_sid[3,1]    0.9998307
## ab_sid[3,2]    1.0002095
## ab_sid[4,1]    1.0000470
## ab_sid[4,2]    1.0001087
## ab_sid[5,1]    0.9997569
## ab_sid[5,2]    0.9998781
## ab_sid[6,1]    1.0003932
## ab_sid[6,2]    0.9997136
## ab_sid[7,1]    0.9999204
## ab_sid[7,2]    1.0002205
## ab_sid[8,1]    0.9999696
## ab_sid[8,2]    1.0000858
## ab_sid[9,1]    1.0000457
## ab_sid[9,2]    1.0003082
## ab_sid[10,1]   0.9998571
## ab_sid[10,2]   1.0000915
```

All Rhat4 values are 1.

Each parameter also sampled well.

PSIS/WAIC

```r
PSIS(hole_mod)
```

```
## Some Pareto k values are high (>0.5). Set pointwise=TRUE to inspect individual points.

##       PSIS      lppd  penalty  std_err
## 1 172.8493 -86.42464 6.148971 12.23386
```

```r
WAIC(hole_mod)
```

```
##       WAIC      lppd  penalty  std_err
## 1 172.6463 -80.27567 6.047468 12.17055
```

The are some Pareto k values > 0.5. As long as < 0.7, not an issue. Are they?

```r
PSIS(hole_mod, pointwise = TRUE) %>%
    high_k_rows()
```

```
## Some Pareto k values are high (>0.5). Set pointwise=TRUE to inspect individual points.

## # A tibble: 2 x 15
##   videoid  PSIS   lppd penalty std_err     k surgid   pgs time_until_1st_~
##     <int> <dbl>  <dbl>   <dbl>   <dbl> <dbl>  <int> <int>            <dbl>
## 1      98 0.733 -0.367  0.0530    12.2 0.503      4     5             39.5
## 2     130 3.23  -1.61   0.330     12.2 0.557      6     4             22.6
## # ... with 6 more variables: time_cvs_attained <dbl>,
## #   laparascopic_duration <dbl>, dissection_duration <dbl>,
## #   gb_removal_duration <dbl>, gb_hole <lgl>, gb_holes <int>
```

Only 2 rows total, and both have $k < 0.6$, so minimal issue with outliers.

Trace rank plot (trankplot)

```r
trankplot(hole_mod)
```

Good mixing as shown with large amount of overlap.

Trace plot

```r
traceplot(hole_mod)
```

Chains are stationary with a visible central tendency, have good mixing, and converge.

Evaluate model results

First, obtain tidy samples, as we did with the duration model. Additionally, transform the result to get the absolute probability of a hole and the OR of a hole for each PGS:

```r
tidy_hole_samples <- extract_samples(
    hole_mod,
    pars = c(
        "a_bar",
        paste0("a[", 1:nsurgs, "]"),
        "bP_bar",
        paste0("bP[", 1:nsurgs, "]"),
        # sigma_a
        "sigma_sid[1]",
```

```
        # sigma_b
        "sigma_sid[2]",
        paste0("Rho_sid[1,2]"),
        paste0("delta[", 1:4, "]")
    )
) %>%
    tidy_surgeons() %>%
    sum_deltas() %>%
    tidy_pgs() %>%
    mutate(
        p = inv_logit(a + bP * sum_delta_j),
        bOR = exp(bP * sum_delta_j)
    ) %>%
    arrange(sample_num, surgeon, PGS) %>%
    group_by(surgeon, sample_num) %>%
    # calc change in probability for each sample
    # group by surgeon as well because each sample_num
    # has the info for all 10 surgeons and bar surgeon
    mutate(
        # change from one PGS level to the other
        incr_delta_p = p - lag(p),
        # change from PGS1 to that PGS level
        delta_p = p - p[1]
    ) %>%
    ungroup()
```

Key to know, is that:

`incr_delta_p` is the probability difference from one level to the next. It will be `NA` for a PGS1.

`delta_p` is probability difference from a certain PGS level to PGS1.

`p` is the probability of a hole at that level

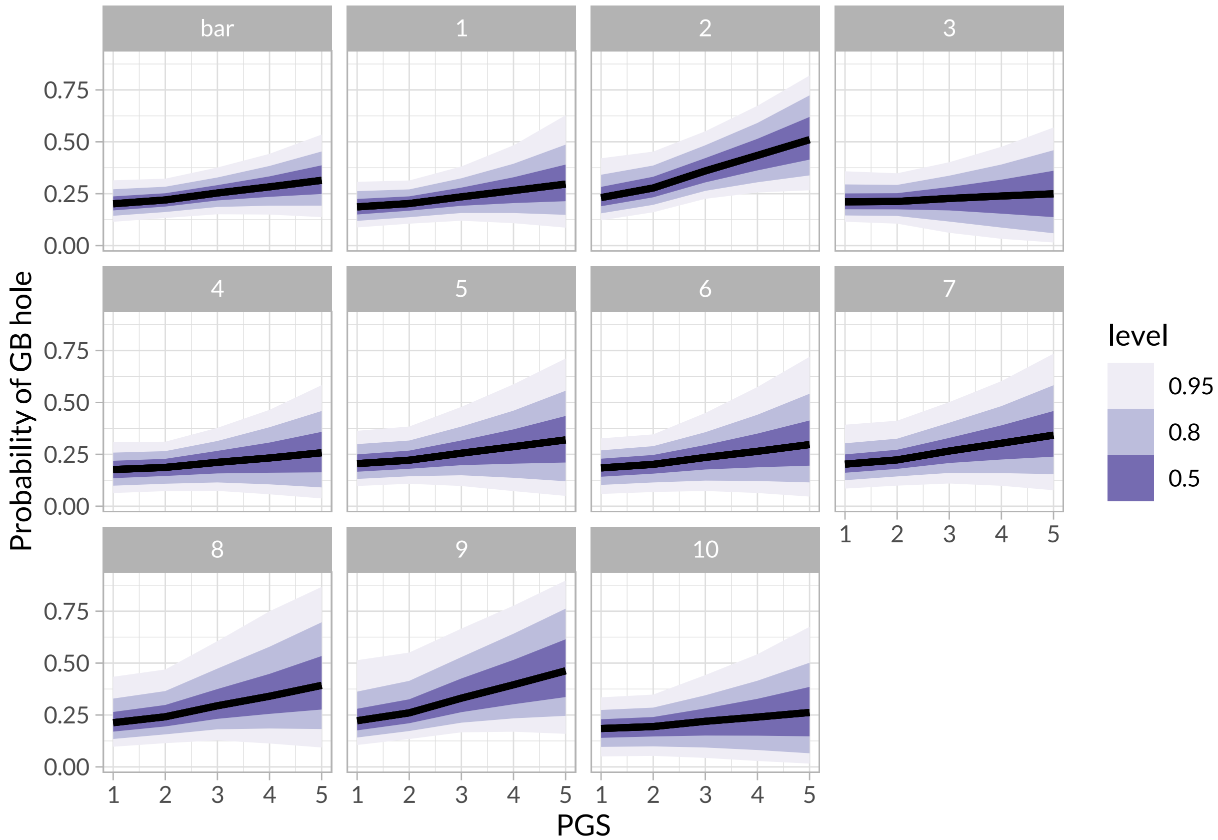
`bOR` is the increased odds, compared to a PGS of 1, of a gb hole.

Plot results

For all surgeons

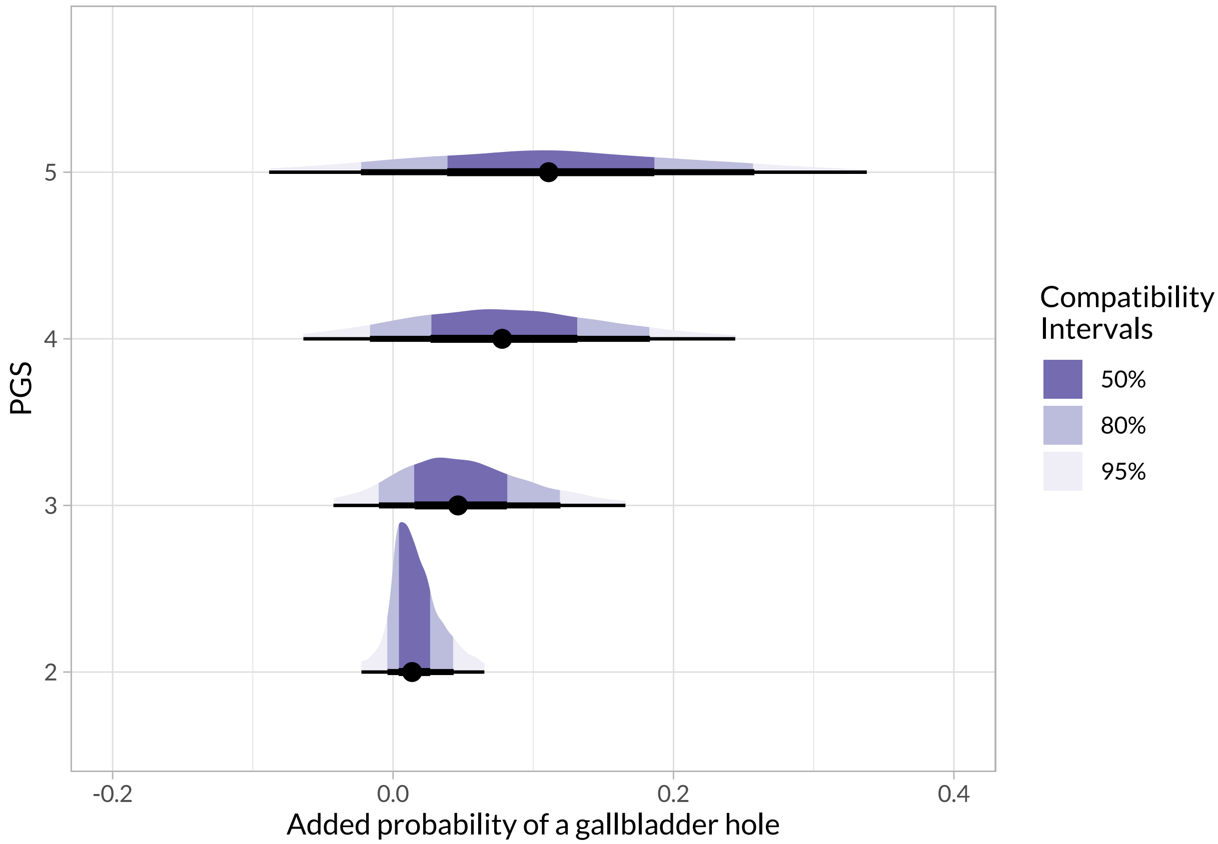
Remember that "bar" is the average of all the surgeons:

```
tidy_hole_samples %>%
    mutate(surgeon = parse_factor(surgeon, levels = c("bar", as.character(1:nsurgs)))) %>%
    ggplot(aes(PGS, p)) +
    stat_lineribbon() +
    facet_wrap(~ surgeon) +
    scale_fill_brewer(palette = "Purples") +
    labs(y = "Probability of GB hole")
```



On average across all surgeons

```r
bar_gb <- tidy_hole_samples %>%
    filter(surgeon == "bar", PGS != 1) %>%
    mutate(PGS = as.factor(PGS))
bar_gb %>%
    halfeye(delta_p, PGS) +
    coord_cartesian(xlim = c(-0.2, 0.4)) +
    labs(
        y = "PGS",
        x = "Added probability of a gallbladder hole"
    )
```



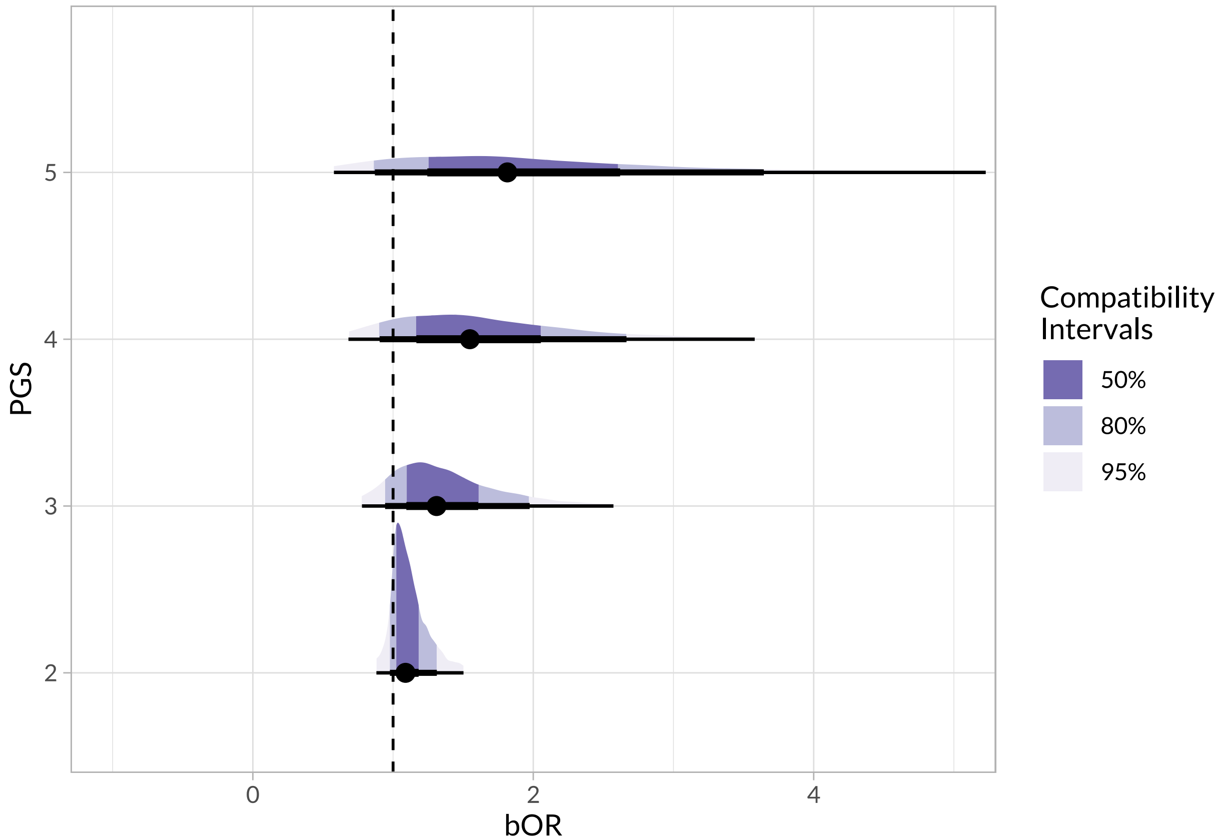
And numbers for this:

```
bar_gb %>%
    group_by(PGS) %>%
    ci_ints(delta_p)
```

```
## # A tibble: 4 x 7
##   PGS    mean int_50     int_66     int_80     int_89      int_95
##   <fct> <dbl> <chr>      <chr>      <chr>      <chr>       <chr>
## 1 2      0.02 0, 0.03    0, 0.03    0, 0.03    -0.01, 0.05 -0.02, 0.07
## 2 3      0.05 0.02, 0.08 0, 0.1     0.01, 0.09 -0.03, 0.14 -0.04, 0.17
## 3 4      0.08 0.03, 0.13 0.01, 0.15 0.01, 0.15 -0.04, 0.21 -0.06, 0.24
## 4 5      0.11 0.04, 0.19 0.01, 0.22 0.02, 0.21 -0.05, 0.3  -0.09, 0.34
```

And looking at the odd ratios:

```
bar_gb %>%
    halfeye(bOR, PGS) +
    coord_cartesian(xlim = c(-1, 5)) +
    geom_vline(xintercept = 1, linetype = 2)
```



Numbers for this:

```
bar_gb %>%
    group_by(PGS) %>%
    ci_ints(bOR)
```

```
## # A tibble: 4 x 7
##   PGS    mean int_50     int_66     int_80     int_89     int_95
##   <fct> <dbl> <chr>      <chr>      <chr>      <chr>      <chr>
## 1 2      1.12 1.02, 1.18 1.01, 1.24 1.01, 1.22 0.94, 1.4  0.88, 1.5
## 2 3      1.4  1.09, 1.61 1.02, 1.77 1.05, 1.7  0.86, 2.23 0.78, 2.57
## 3 4      1.7  1.17, 2.05 1.04, 2.3  1.09, 2.2  0.79, 3.07 0.68, 3.58
## 4 5      2.1  1.24, 2.62 1.06, 3.04 1.13, 2.85 0.72, 4.32 0.58, 5.23
```

For a surgeon most affected by PGS

```
hole_bps <- precis(hole_mod, depth = 2, pars = paste0("bP[", 1:nsurgs, "]")) %>%
    as_tibble(rownames = "var")
# Least affected surgeon:
slice_min(hole_bps, mean)
```

```
## # A tibble: 1 x 7
##   var         mean    sd `5.5%` `94.5%` n_eff Rhat4
##   <chr>      <dbl> <dbl>  <dbl>   <dbl> <dbl> <dbl>
## 1 bP[3] -0.000898  1.19  -2.17    1.45 7307.  1.00
```

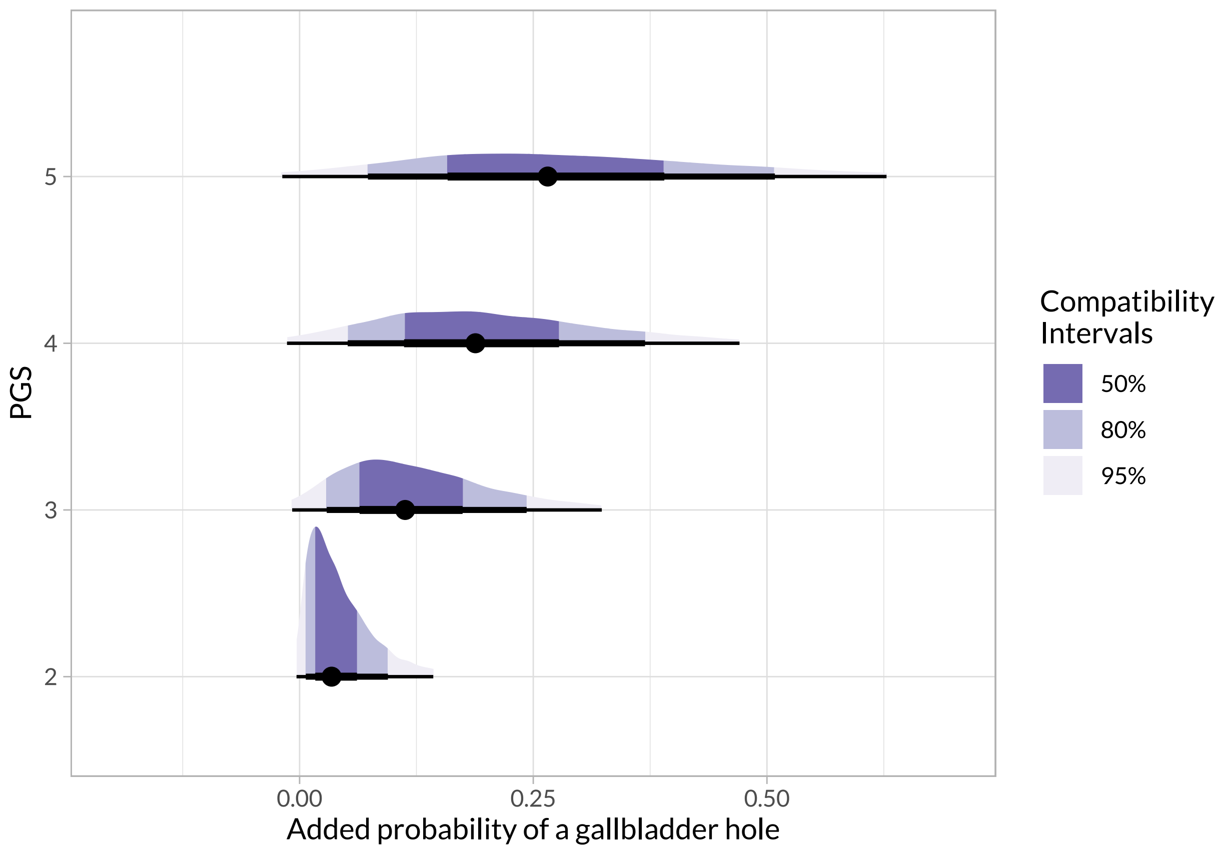
```r
# Most affected surgeon:
slice_max(hole_bps, mean)
```

```
## # A tibble: 1 x 7
##   var    mean    sd `5.5%` `94.5%` n_eff Rhat4
##   <chr> <dbl> <dbl>  <dbl>   <dbl> <dbl> <dbl>
## 1 bP[2]  1.29 0.791  0.145    2.65 7765.  1.00
```

Surgeon 2 is most affected.

```r
surg2_gb <- tidy_hole_samples %>%
    filter(surgeon == "2", PGS != 1) %>%
    mutate(PGS = as.factor(PGS))

surg2_gb %>%
    halfeye(delta_p, PGS) +
    coord_cartesian(xlim = c(-0.2, 0.7)) +
    labs(
        y = "PGS",
        x = "Added probability of a gallbladder hole"
    )
```



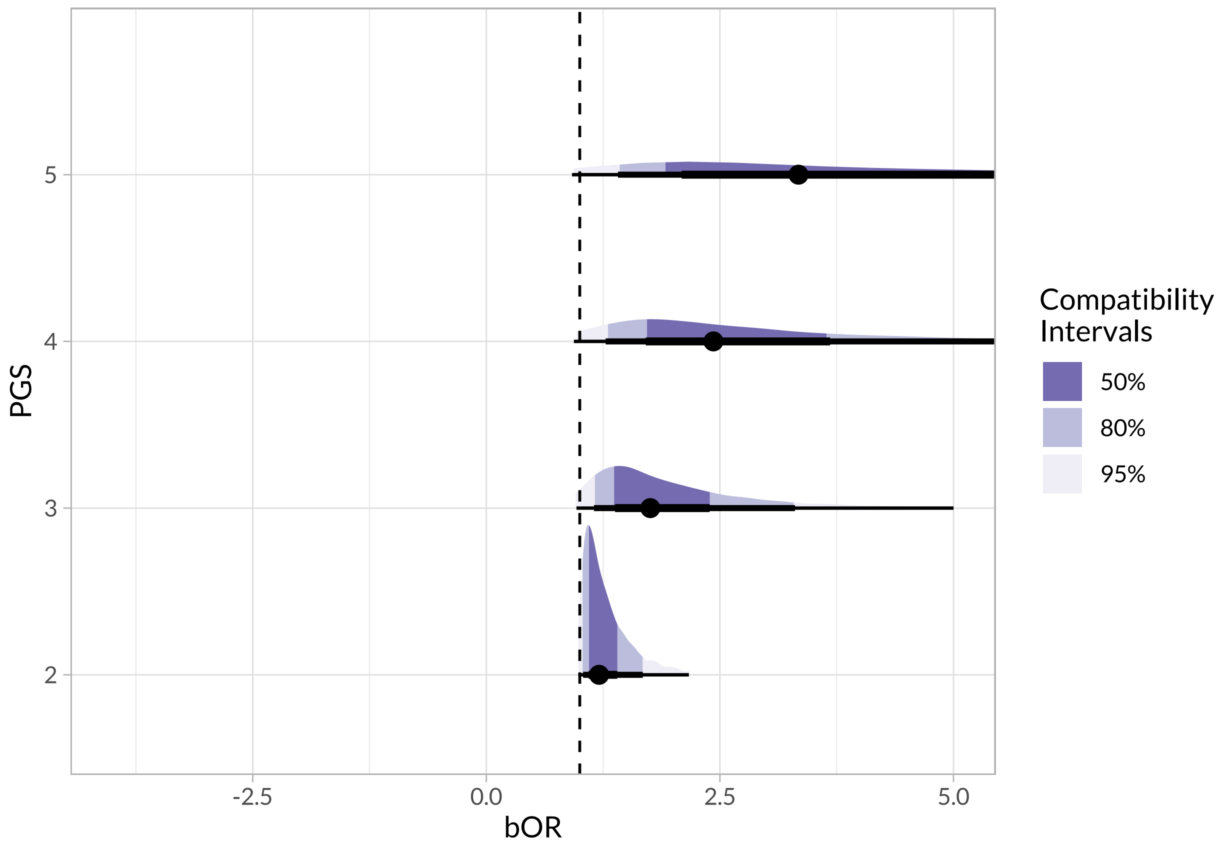
And numbers for this:

```r
surg2_gb %>%
    group_by(PGS) %>%
    ci_ints(delta_p)
```

```
## # A tibble: 4 x 7
##   PGS    mean int_50     int_66     int_80     int_89     int_95
##   <fct> <dbl> <chr>      <chr>      <chr>      <chr>      <chr>
## 1 2      0.04 0.02, 0.06 0.01, 0.08 0.01, 0.07 0, 0.12    0, 0.14
## 2 3      0.13 0.06, 0.17 0.05, 0.2  0.05, 0.19 0.01, 0.28 -0.01, 0.32
## 3 4      0.2  0.11, 0.28 0.08, 0.32 0.1, 0.3   0.02, 0.42 -0.01, 0.47
## 4 5      0.28 0.16, 0.39 0.12, 0.44 0.14, 0.42 0.03, 0.57 -0.02, 0.63
```

And looking at odds ratios:

```
surg2_gb %>%
    halfeye(bOR, PGS) +
    coord_cartesian(xlim = c(-4, 5)) +
    geom_vline(xintercept = 1, linetype = 2)
```



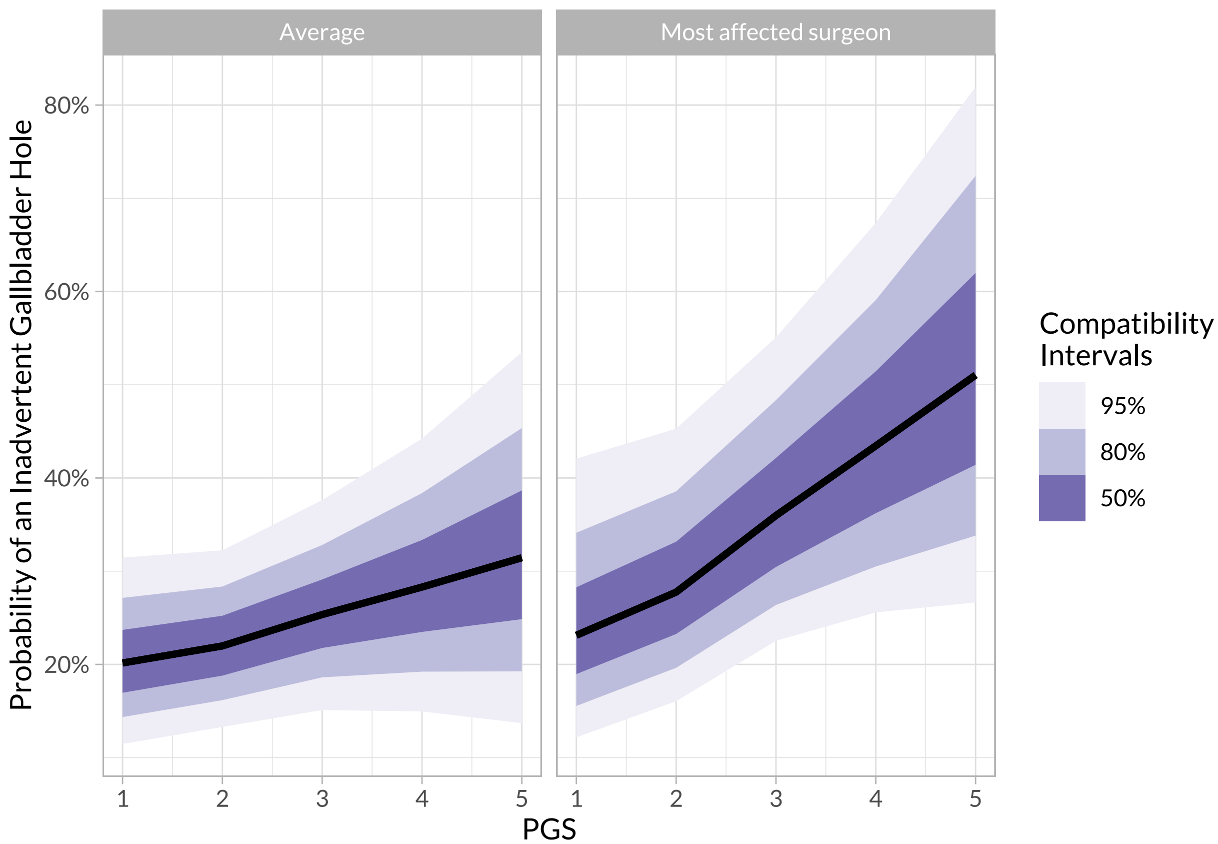
Numbers for this:

```
surg2_gb %>%
    group_by(PGS) %>%
    ci_ints(bOR)
```

```
## # A tibble: 4 x 7
##   PGS    mean int_50     int_66     int_80     int_89     int_95
##   <fct> <dbl> <chr>      <chr>      <chr>      <chr>      <chr>
## 1 2      1.3  1.09, 1.4  1.06, 1.52 1.08, 1.47 1.01, 1.87 0.99, 2.17
## 2 3      2.06 1.38, 2.39 1.26, 2.77 1.3, 2.61  1.06, 3.99 0.96, 5
## 3 4      3.09 1.71, 3.68 1.5, 4.47  1.58, 4.13 1.11, 7.06 0.94, 9.12
## 4 5      5.22 2.09, 5.83 1.74, 7.59 1.89, 6.81 1.16, 14.1 0.92, 20.45
```

And comparing to the average surgeon:

```
tidy_hole_samples %>%
    filter(surgeon %in% c("bar", "2")) %>%
    mutate(surgeon = if_else(surgeon == "bar", "Average", "Most affected surgeon")) %>%
    #mutate(surgeon = parse_factor(surgeon, levels = c("bar", as.character(1:nsurgs)))) %>%
    ggplot(aes(PGS, p)) +
    stat_lineribbon() +
    facet_wrap(~ surgeon) +
    scale_fill_brewer(labels = c("95%", "80%", "50%"), palette = "Purples") +
    scale_y_continuous(labels = scales::percent_format()) +
    labs(
        x = "PGS",
        y = "Probability of an Inadvertent Gallbladder Hole",
        fill = "Compatibility\nIntervals"
    )
```




```
ggsave("../output/pgs_hole.svg", width = 6, height = 4)
ggsave("../output/pgs_hole.pdf", width = 6, height = 4)
```

numbers for average surgeon:

```
tidy_hole_samples %>%
    filter(surgeon == "bar") %>%
    group_by(PGS) %>%
    ci_ints(p)
```

```
## # A tibble: 5 x 7
```

```
##     PGS  mean int_50     int_66     int_80     int_89     int_95
##   <dbl> <dbl> <chr>      <chr>      <chr>      <chr>      <chr>
## 1     1 0.21  0.17, 0.24 0.16, 0.25 0.16, 0.25 0.13, 0.29 0.11, 0.31
## 2     2 0.22  0.19, 0.25 0.18, 0.27 0.18, 0.26 0.15, 0.3  0.13, 0.32
## 3     3 0.26  0.22, 0.29 0.2, 0.31  0.21, 0.3  0.17, 0.35 0.15, 0.38
## 4     4 0.290 0.23, 0.33 0.21, 0.36 0.22, 0.35 0.17, 0.41 0.15, 0.44
## 5     5 0.32  0.25, 0.39 0.22, 0.42 0.23, 0.4  0.17, 0.49 0.14, 0.53
```

numbers for most affected surgeon:

```
tidy_hole_samples %>%
    filter(surgeon == "2") %>%
    group_by(sample_num, PGS) %>%
    ci_ints(p)
```

```
## `summarise()` has grouped output by 'sample_num'. You can override using the `.groups` argument.
```

```
## # A tibble: 50,000 x 8
##    sample_num   PGS  mean int_50    int_66    int_80    int_89    int_95
##    <chr>      <dbl> <dbl> <chr>     <chr>     <chr>     <chr>     <chr>
##  1 sample1        1  0.13 0.13, 0.~ 0.13, 0.~ 0.13, 0.~ 0.13, 0.~ 0.13, 0~
##  2 sample1        2  0.23 0.23, 0.~ 0.23, 0.~ 0.23, 0.~ 0.23, 0.~ 0.23, 0~
##  3 sample1        3  0.25 0.25, 0.~ 0.25, 0.~ 0.25, 0.~ 0.25, 0.~ 0.25, 0~
##  4 sample1        4  0.3  0.3, 0.3  0.3, 0.3  0.3, 0.3  0.3, 0.3  0.3, 0.3
##  5 sample1        5  0.37 0.37, 0.~ 0.37, 0.~ 0.37, 0.~ 0.37, 0.~ 0.37, 0~
##  6 sample10       1  0.14 0.14, 0.~ 0.14, 0.~ 0.14, 0.~ 0.14, 0.~ 0.14, 0~
##  7 sample10       2  0.17 0.17, 0.~ 0.17, 0.~ 0.17, 0.~ 0.17, 0.~ 0.17, 0~
##  8 sample10       3  0.26 0.26, 0.~ 0.26, 0.~ 0.26, 0.~ 0.26, 0.~ 0.26, 0~
##  9 sample10       4  0.43 0.43, 0.~ 0.43, 0.~ 0.43, 0.~ 0.43, 0.~ 0.43, 0~
## 10 sample10       5  0.47 0.47, 0.~ 0.47, 0.~ 0.47, 0.~ 0.47, 0.~ 0.47, 0~
## # ... with 49,990 more rows
```

```
tidy_hole_samples %>%
    filter(surgeon == "2")
```

```
## # A tibble: 50,000 x 17
##    sample_num sigma_sid_1 sigma_sid_2 Rho_sid_1_2 delta_1 delta_2 delta_3
##    <chr>            <dbl>       <dbl>       <dbl>   <dbl>   <dbl>   <dbl>
##  1 sample1          0.350       0.774      0.0426   0.488  0.0862   0.209
##  2 sample1          0.350       0.774      0.0426   0.488  0.0862   0.209
##  3 sample1          0.350       0.774      0.0426   0.488  0.0862   0.209
##  4 sample1          0.350       0.774      0.0426   0.488  0.0862   0.209
##  5 sample1          0.350       0.774      0.0426   0.488  0.0862   0.209
##  6 sample10         0.176       1.10       0.111    0.118  0.325    0.459
##  7 sample10         0.176       1.10       0.111    0.118  0.325    0.459
##  8 sample10         0.176       1.10       0.111    0.118  0.325    0.459
##  9 sample10         0.176       1.10       0.111    0.118  0.325    0.459
## 10 sample10         0.176       1.10       0.111    0.118  0.325    0.459
## # ... with 49,990 more rows, and 10 more variables: delta_4 <dbl>,
## #   surgeon <chr>, a <dbl>, bP <dbl>, PGS <int>, sum_delta_j <dbl>,
## #   p <dbl>, bOR <dbl>, incr_delta_p <dbl>, delta_p <dbl>
```

Correlation of PGS1 and effect of incremental PGS

```
extract_samples(hole_mod, pars = c("Rho_sid[1,2]")) %>%
    rename(rho = Rho_sid_1_2) %>%
```

```r
    ci_ints(rho) %>%
    knitr::kable(
        caption = "Correlation between GB hole in PGS1 case and the effect of incrementing PGS"
    )
```

Table 8: Correlation between GB hole in PGS1 case and the effect of incrementing PGS

mean	int_50	int_66	int_80	int_89	int_95
-0.03	-0.27, 0.21	-0.35, 0.31	-0.32, 0.27	-0.54, 0.5	-0.63, 0.6

Critical View of Safety Attainment Analysis

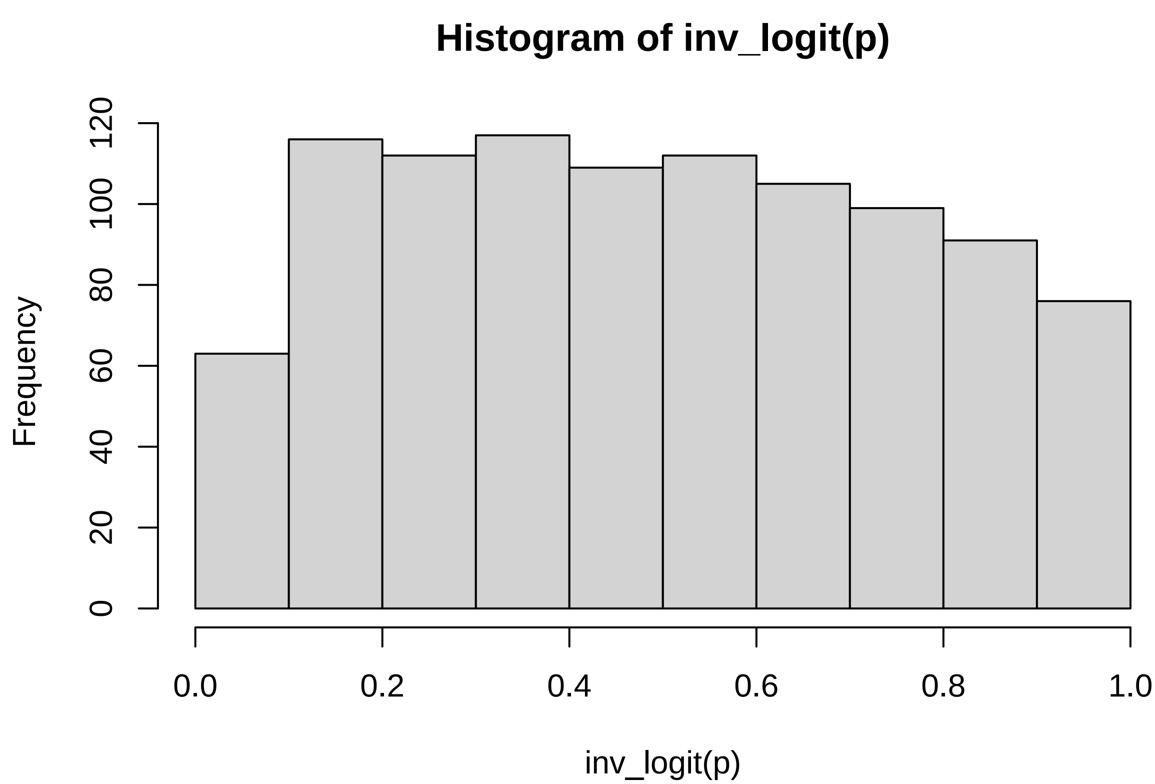
Priors determination

As before, we will use weakly regularizing priors.

Intercept

We will reuse the N(0, 1.5) prior we established for GB hole:

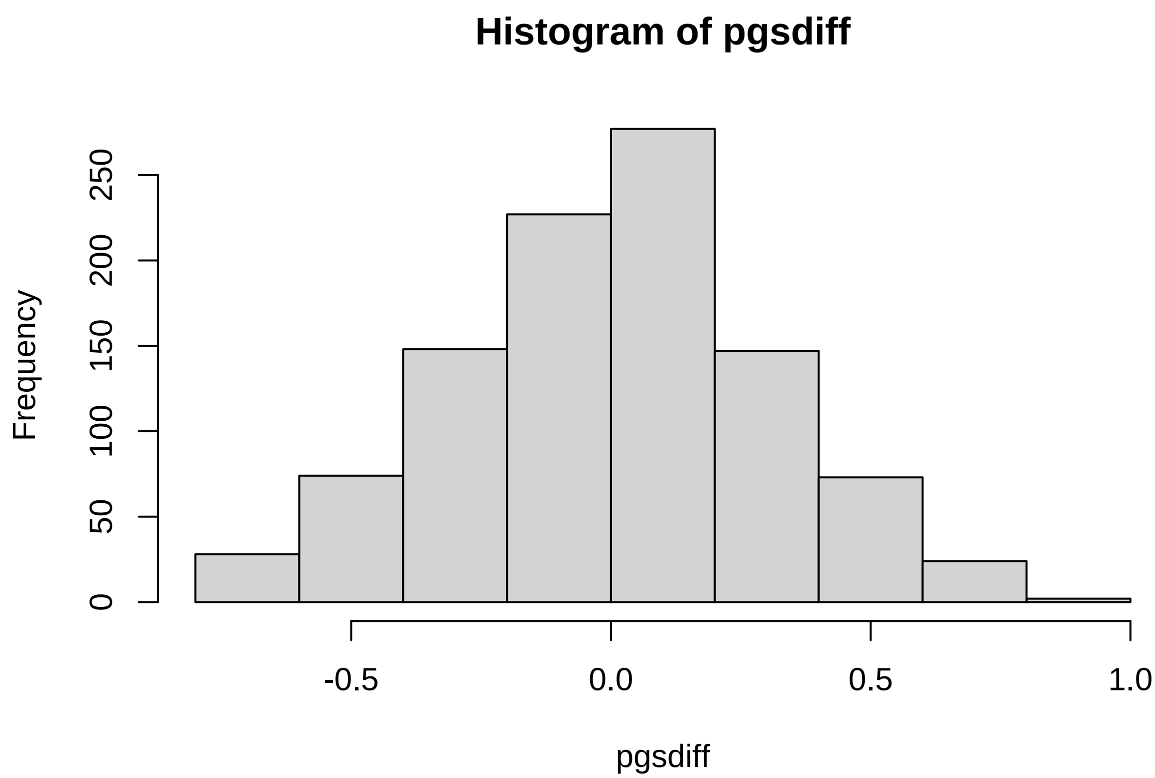
```r
p <- rnorm(1000, 0, 1.5)
hist(inv_logit(p))
```



PGS coefficient

Remember, this is the effect of a PGS5. We will need to simulate the intercept then add the slope, followed by undoing the logit to see the probability:

```
pint <- rnorm(1000, 0, 1.5)
ppgs <- rnorm(1000, 0, 2)
pgsdiff <- inv_logit(pint + ppgs) - inv_logit(pint)
hist(pgsdiff)
```



This prior, N(0, 2), is a wide and rather uninformative prior, but still fair. It allows for some surgeons to obtain CVS for a PGS1 but for PGS5 to never obtain it (due to technique preference). However, it still clusters most of the effects around zero, as we would, a priori, predict.

other priors

Other priors will be the usual weakly regularizing ones, including Dirichlet of 2, Exponential 1, and LKJCorr of 4.

Formula

Below is the centered version. The model given to Stan is the non-centered version that is mathematically equivalent but dramatically improves sampling.

$$
\begin{aligned}
CVS_i &\sim Bernoulli(p_i) \\
logit(p_i) &= \alpha_{sid[i]} + \beta_{sid[i]} * \sum_{j=0}^{PGS_i - 1} \delta_j \\
\begin{bmatrix} \alpha_{sid} \\ \beta_{sid} \end{bmatrix} &\sim MVNormal(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \mathbf{S}) \\
\alpha &\sim Normal(0, 1.5) \\
\beta &\sim Normal(0, 2) \\
\delta &\sim Dirichlet(2) \\
\mathbf{S} &= \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \mathbf{R} \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \\
\mathbf{R} &= \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \\
\mathbf{R} &\sim LKJCorr(4) \\
\sigma_\alpha, \sigma_\beta &\sim Exponential(1)
\end{aligned} \tag{3}
$$

Code

```
set.seed(1234)
cvs_mod <- ulam(
    alist(
        cvs ~ bernoulli(p),
        logit(p) <- a_bar + ab_sid[sid, 1] + (bP_bar + ab_sid[sid, 2]) * sum(delta_j[1:pgs]),
        a_bar ~ normal(0, 1.5),
        bP_bar ~ normal(0, 2),
        vector[5]: delta_j <<- append_row(0, delta),
        simplex[4]: delta ~ dirichlet(alpha),
        transpars> matrix[sid, 2]: ab_sid <-
            compose_noncentered(sigma_sid, L_Rho_sid, z_sid),
        matrix[2, sid]: z_sid ~ normal(0, 1),
        vector[2]: sigma_sid ~ exponential(1),
        cholesky_factor_corr[2]: L_Rho_sid ~ lkj_corr_cholesky(4),
        gq> matrix[2, 2]: Rho_sid <<- Chol_to_Corr(L_Rho_sid),
        gq> vector[sid]:a <<- a_bar + ab_sid[, 1],
        gq> vector[sid]:bP <<- bP_bar + ab_sid[, 2]
    ),
    data = cdat,
    cores = 4,
    chains = 4,
    iter = 5000,
    log_lik = TRUE
)
```

```
## Compiling Stan program...
```

```
## -\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\| Runni
##
## Chain 1 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 2 Iteration:    1 / 5000 [  0%]  (Warmup)
```

```
## Chain 2 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 2 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 3 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 4 Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 5000 [  2%]  (Warmup)
## Chain 1 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 2 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 2 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 3 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 4 Iteration:  200 / 5000 [  4%]  (Warmup)
## Chain 4 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 1 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 3 Iteration:  300 / 5000 [  6%]  (Warmup)
## Chain 3 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 4 Iteration:  400 / 5000 [  8%]  (Warmup)
## Chain 1 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 1 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 2 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 2 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 3 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 3 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 4 Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 4 Iteration:  600 / 5000 [ 12%]  (Warmup)
## Chain 1 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 2 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 2 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 3 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 3 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 4 Iteration:  700 / 5000 [ 14%]  (Warmup)
## Chain 4 Iteration:  800 / 5000 [ 16%]  (Warmup)
## Chain 1 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 2 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 3 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 3 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 4 Iteration:  900 / 5000 [ 18%]  (Warmup)
## Chain 4 Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 1 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 2 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 2 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 3 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 4 Iteration: 1100 / 5000 [ 22%]  (Warmup)
## Chain 1 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 1 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 2 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 2 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 3 Iteration: 1200 / 5000 [ 24%]  (Warmup)
## Chain 3 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 4 Iteration: 1200 / 5000 [ 24%]  (Warmup)
```

```
## Chain 4 Iteration: 1300 / 5000 [ 26%]  (Warmup)
## Chain 1 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 2 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 2 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 3 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 3 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 4 Iteration: 1400 / 5000 [ 28%]  (Warmup)
## Chain 4 Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 1 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 1 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 2 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 3 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 3 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 4 Iteration: 1600 / 5000 [ 32%]  (Warmup)
## Chain 4 Iteration: 1700 / 5000 [ 34%]  (Warmup)
## Chain 1 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 1 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 2 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 2 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 3 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 4 Iteration: 1800 / 5000 [ 36%]  (Warmup)
## Chain 1 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 1 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 2 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 2 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 3 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 3 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 4 Iteration: 1900 / 5000 [ 38%]  (Warmup)
## Chain 4 Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 1 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 1 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 2 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 3 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 3 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 4 Iteration: 2100 / 5000 [ 42%]  (Warmup)
## Chain 4 Iteration: 2200 / 5000 [ 44%]  (Warmup)
## Chain 1 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 1 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 2 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 3 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 3 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 4 Iteration: 2300 / 5000 [ 46%]  (Warmup)
## Chain 4 Iteration: 2400 / 5000 [ 48%]  (Warmup)
## Chain 1 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 1 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 2 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 3 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 3 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 4 Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 4 Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2800 / 5000 [ 56%]  (Sampling)
```

```
## Chain 1 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 2 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 2 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 3 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 3 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 4 Iteration: 2600 / 5000 [ 52%]  (Sampling)
## Chain 4 Iteration: 2700 / 5000 [ 54%]  (Sampling)
## Chain 1 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 2 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 3 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 3 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 4 Iteration: 2800 / 5000 [ 56%]  (Sampling)
## Chain 4 Iteration: 2900 / 5000 [ 58%]  (Sampling)
## Chain 1 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 1 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 2 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 2 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 3 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 3 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 4 Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 1 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 3 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 4 Iteration: 3100 / 5000 [ 62%]  (Sampling)
## Chain 4 Iteration: 3200 / 5000 [ 64%]  (Sampling)
## Chain 1 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 2 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 3 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 3 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 4 Iteration: 3300 / 5000 [ 66%]  (Sampling)
## Chain 4 Iteration: 3400 / 5000 [ 68%]  (Sampling)
## Chain 1 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 1 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 2 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 2 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 3 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 3 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 4 Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 1 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 2 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 2 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 3 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 4 Iteration: 3600 / 5000 [ 72%]  (Sampling)
## Chain 4 Iteration: 3700 / 5000 [ 74%]  (Sampling)
## Chain 1 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 2 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 3 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 3 Iteration: 3900 / 5000 [ 78%]  (Sampling)
```

```
## Chain 4 Iteration: 3800 / 5000 [ 76%]  (Sampling)
## Chain 4 Iteration: 3900 / 5000 [ 78%]  (Sampling)
## Chain 1 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 2 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 3 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 3 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 4 Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 4 Iteration: 4100 / 5000 [ 82%]  (Sampling)
## Chain 4 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 1 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 1 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 2 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 3 Iteration: 4200 / 5000 [ 84%]  (Sampling)
## Chain 3 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 3 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 4 Iteration: 4300 / 5000 [ 86%]  (Sampling)
## Chain 1 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 1 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 3 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 4 Iteration: 4400 / 5000 [ 88%]  (Sampling)
## Chain 4 Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 2 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 2 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 3 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 3 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 4 Iteration: 4600 / 5000 [ 92%]  (Sampling)
## Chain 4 Iteration: 4700 / 5000 [ 94%]  (Sampling)
## Chain 2 finished in 2.9 seconds.
## Chain 1 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 1 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 3 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 3 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 4 Iteration: 4800 / 5000 [ 96%]  (Sampling)
## Chain 1 finished in 3.0 seconds.
## Chain 3 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 4 Iteration: 4900 / 5000 [ 98%]  (Sampling)
## Chain 4 Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 3 finished in 3.0 seconds.
## Chain 4 finished in 3.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 3.0 seconds.
## Total execution time: 3.2 seconds.
```

Diagnostic Evaluation of Markov Chains

Rhat4 and effective sampling size

```r
precis(cvs_mod, depth = 3)
```

```
##                        mean         sd        5.5%      94.5%     n_eff
## a_bar          -0.448264277 0.61688817 -1.41086435  0.5275515  5526.114
## bP_bar         -0.889484396 0.78108251 -2.11310205  0.3496151  9201.120
## delta[1]        0.328473683 0.16433750  0.08260989  0.6057762 13496.905
## delta[2]        0.220419204 0.13195595  0.04725604  0.4609032 14146.265
## delta[3]        0.212912739 0.13240279  0.04365943  0.4563196 12076.815
## delta[4]        0.238194371 0.13766828  0.05413588  0.4869947 13531.513
## z_sid[1,1]     -1.839045735 0.65210868 -2.91555170 -0.8459421  9366.012
## z_sid[1,2]      0.618261270 0.45749453 -0.06635933  1.3862307  6149.916
## z_sid[1,3]      0.669119309 0.46456034 -0.03185255  1.4434717  5964.906
## z_sid[1,4]      0.362315778 0.47940086 -0.37027781  1.1554911  7389.905
## z_sid[1,5]     -1.380382002 0.67677304 -2.51579955 -0.3565313 11137.968
## z_sid[1,6]     -0.302936852 0.57023348 -1.22991925  0.5780491  9718.295
## z_sid[1,7]      0.582620856 0.60083384 -0.35222335  1.5378422  8832.196
## z_sid[1,8]      0.157058844 0.63259009 -0.84518418  1.1522848  9723.647
## z_sid[1,9]      0.564585927 0.65212853 -0.48601591  1.5798250  8548.197
## z_sid[1,10]     0.153927964 0.60398823 -0.82178581  1.1014727  9986.034
## z_sid[2,1]     -0.249554790 1.00763955 -1.86455245  1.3696495 13200.527
## z_sid[2,2]     -0.065789320 0.88995102 -1.49860715  1.3800517 12929.078
## z_sid[2,3]     -0.018175847 0.90890870 -1.48154160  1.4394117 13921.625
## z_sid[2,4]     -0.151404376 0.94401335 -1.64393310  1.3765873 13931.730
## z_sid[2,5]     -0.145643791 0.98203323 -1.71762960  1.4358294 14578.909
## z_sid[2,6]     -0.090901718 0.97910390 -1.65641080  1.4929421 15407.370
## z_sid[2,7]      0.199337105 0.98718006 -1.39282815  1.7434569 14347.200
## z_sid[2,8]      0.124666472 0.97829507 -1.44821740  1.6918144 14862.009
## z_sid[2,9]      0.345549564 1.02640615 -1.30792115  1.9698708 12629.499
## z_sid[2,10]    -0.150004857 0.98051893 -1.72198380  1.4395869 14374.933
## sigma_sid[1]    1.621404837 0.57342794  0.88822188  2.6459997  4731.374
## sigma_sid[2]    0.659058739 0.62805927  0.03962365  1.8287401  5749.896
## L_Rho_sid[1,1]  1.000000000 0.00000000  1.00000000  1.0000000       NaN
## L_Rho_sid[1,2]  0.000000000 0.00000000  0.00000000  0.0000000       NaN
## L_Rho_sid[2,1] -0.001936964 0.33372487 -0.53620862  0.5380030 14076.456
## L_Rho_sid[2,2]  0.939499448 0.07730767  0.78549857  0.9996702  4329.513
## bP[1]          -1.150665657 1.29460826 -3.19523830  0.5955109  9564.878
## bP[2]          -0.955404812 0.80787821 -2.23963440  0.2865176 12252.271
## bP[3]          -0.896671158 0.89001770 -2.26516665  0.5080129 10467.681
## bP[4]          -1.042222982 0.95837704 -2.57519565  0.3985956  9751.798
## bP[5]          -1.055264904 1.23202517 -2.91185735  0.7117994  9737.563
## bP[6]          -0.988572624 1.03040089 -2.57387165  0.5573008  9884.912
## bP[7]          -0.688147867 1.09174104 -2.21805555  1.1006252  8313.455
## bP[8]          -0.747784559 1.15793385 -2.35319760  1.0687032  8203.859
## bP[9]          -0.516838955 1.24193269 -2.12125605  1.6121935  6285.792
## bP[10]         -1.065238608 1.10530393 -2.78919965  0.5314901  9779.706
## a[1]           -3.322331744 1.22642001 -5.46087965 -1.7082114  9547.034
## a[2]            0.478285940 0.53969654 -0.35338086  1.3800427 12297.721
## a[3]            0.553435894 0.46637565 -0.16874549  1.3219033 12143.054
## a[4]            0.102501321 0.58940133 -0.81825293  1.0626413 11874.579
## a[5]           -2.647260416 1.29108536 -4.93535810 -0.9683662  9198.918
## a[6]           -0.905369345 0.79879007 -2.24964860  0.3317886 12565.145
```

```
## a[7]            0.463634188 0.92330483 -0.97912540  1.9642975 10672.536
## a[8]           -0.193327342 0.95623393 -1.70277165  1.3077560 11482.991
## a[9]            0.439898077 1.04467563 -1.20548275  2.0686177  8635.116
## a[10]          -0.196167550 0.89093668 -1.64701190  1.2158240 13280.297
## Rho_sid[1,1]    1.000000000 0.00000000  1.00000000  1.0000000       NaN
## Rho_sid[1,2]   -0.001936964 0.33372487 -0.53620862  0.5380030 14076.456
## Rho_sid[2,1]   -0.001936964 0.33372487 -0.53620862  0.5380030 14076.456
## Rho_sid[2,2]    1.000000000 0.00000000  1.00000000  1.0000000       NaN
## ab_sid[1,1]    -2.874067524 1.23309749 -4.97845385 -1.2337064  8000.900
## ab_sid[1,2]    -0.261181339 1.03882541 -1.98891005  0.9202769  8139.012
## ab_sid[2,1]     0.926550247 0.68617632 -0.10575894  2.0547187  6761.132
## ab_sid[2,2]    -0.065920439 0.62229765 -1.11103025  0.8080064  9042.591
## ab_sid[3,1]     1.001700163 0.68442639 -0.05153216  2.1029684  6368.713
## ab_sid[3,2]    -0.007186774 0.66777693 -1.01856935  0.9753703 10406.442
## ab_sid[4,1]     0.550765612 0.73280885 -0.58177873  1.7597441  7195.507
## ab_sid[4,2]    -0.152738645 0.70993754 -1.38853530  0.7410747  9046.928
## ab_sid[5,1]    -2.198996156 1.27711390 -4.46134320 -0.5011197  7855.024
## ab_sid[5,2]    -0.165780510 0.96096568 -1.69709050  0.9941787  8570.812
## ab_sid[6,1]    -0.457105066 0.87067648 -1.88403190  0.8954287  9200.391
## ab_sid[6,2]    -0.099088247 0.77991732 -1.32565025  0.9478919 10856.604
## ab_sid[7,1]     0.911898440 0.93759330 -0.52004717  2.4357948  8736.683
## ab_sid[7,2]     0.201336486 0.81568376 -0.78696891  1.6469500  8503.783
## ab_sid[8,1]     0.254936940 0.97391174 -1.28704465  1.7970104  9482.088
## ab_sid[8,2]     0.141699801 0.84800448 -0.86385854  1.4592503  8121.752
## ab_sid[9,1]     0.888162378 1.03249034 -0.70713162  2.5327493  7999.904
## ab_sid[9,2]     0.372645364 0.96846062 -0.60613709  2.1131399  6499.924
## ab_sid[10,1]    0.252096718 0.93648487 -1.24014530  1.7312969  9554.567
## ab_sid[10,2]   -0.175754206 0.86016709 -1.56562500  0.8249945  7986.352
##                    Rhat4
## a_bar          0.9999567
## bP_bar         1.0007505
## delta[1]       1.0002862
## delta[2]       0.9998575
## delta[3]       1.0000244
## delta[4]       0.9998040
## z_sid[1,1]     0.9997372
## z_sid[1,2]     1.0000619
## z_sid[1,3]     0.9999466
## z_sid[1,4]     1.0000051
## z_sid[1,5]     0.9999790
## z_sid[1,6]     0.9999521
## z_sid[1,7]     0.9998447
## z_sid[1,8]     0.9997917
## z_sid[1,9]     0.9998741
## z_sid[1,10]    0.9999204
## z_sid[2,1]     0.9998742
## z_sid[2,2]     1.0001594
## z_sid[2,3]     0.9999579
## z_sid[2,4]     0.9998734
## z_sid[2,5]     0.9999513
## z_sid[2,6]     0.9997571
## z_sid[2,7]     1.0000343
## z_sid[2,8]     0.9997416
## z_sid[2,9]     0.9998339
```

```
## z_sid[2,10]     1.0002454
## sigma_sid[1]    1.0003029
## sigma_sid[2]    1.0001184
## L_Rho_sid[1,1]        NaN
## L_Rho_sid[1,2]        NaN
## L_Rho_sid[2,1]  1.0003038
## L_Rho_sid[2,2]  1.0003288
## bP[1]           1.0009580
## bP[2]           0.9999730
## bP[3]           1.0002129
## bP[4]           1.0002224
## bP[5]           1.0006391
## bP[6]           1.0011100
## bP[7]           1.0009017
## bP[8]           1.0003042
## bP[9]           1.0002106
## bP[10]          0.9999089
## a[1]            0.9998172
## a[2]            1.0002256
## a[3]            1.0001210
## a[4]            0.9998188
## a[5]            0.9999735
## a[6]            1.0004143
## a[7]            1.0004496
## a[8]            0.9999813
## a[9]            1.0002249
## a[10]           0.9998948
## Rho_sid[1,1]          NaN
## Rho_sid[1,2]    1.0003038
## Rho_sid[2,1]    1.0003038
## Rho_sid[2,2]          NaN
## ab_sid[1,1]     1.0000385
## ab_sid[1,2]     1.0010492
## ab_sid[2,1]     0.9997134
## ab_sid[2,2]     1.0000161
## ab_sid[3,1]     0.9999292
## ab_sid[3,2]     0.9999158
## ab_sid[4,1]     0.9998295
## ab_sid[4,2]     0.9998102
## ab_sid[5,1]     1.0001105
## ab_sid[5,2]     1.0004355
## ab_sid[6,1]     1.0000184
## ab_sid[6,2]     0.9999682
## ab_sid[7,1]     1.0000505
## ab_sid[7,2]     1.0004745
## ab_sid[8,1]     0.9997837
## ab_sid[8,2]     0.9999249
## ab_sid[9,1]     0.9999622
## ab_sid[9,2]     1.0004938
## ab_sid[10,1]    0.9997966
## ab_sid[10,2]    0.9998074
```

All Rhat4 values are 1.

Each parameter also sampled well.

PSIS/WAIC

```r
PSIS(cvs_mod)
```

```
## Some Pareto k values are high (>0.5). Set pointwise=TRUE to inspect individual points.
```

```
##       PSIS      lppd  penalty  std_err
## 1 161.6147 -80.80735 9.814636 9.838208
```

```r
WAIC(cvs_mod)
```

```
##       WAIC      lppd  penalty  std_err
## 1 161.2069 -70.99271 9.610717 9.756552
```

The are some Pareto k values > 0.5. As long as < 0.7, not an issue. Are they?

```r
PSIS(cvs_mod, pointwise = TRUE) %>%
    high_k_rows()
```

```
## Some Pareto k values are high (>0.5). Set pointwise=TRUE to inspect individual points.
```

```
## # A tibble: 1 x 15
##   videoid  PSIS  lppd penalty std_err     k surgid   pgs time_until_1st_c~
##     <int> <dbl> <dbl>   <dbl>   <dbl> <dbl>  <int> <int>             <dbl>
## 1      98  3.07 -1.54   0.345    9.84 0.506      4     5              39.5
## # ... with 6 more variables: time_cvs_attained <dbl>,
## #   laparascopic_duration <dbl>, dissection_duration <dbl>,
## #   gb_removal_duration <dbl>, gb_hole <lgl>, gb_holes <int>
```

Only 1 row with a k of 0.506, so minimal issue with outliers.

Trace rank plot (trankplot)

```r
trankplot(cvs_mod)
```

Good mixing as shown with large amount of overlap.

Trace plot

```r
traceplot(cvs_mod)
```

Chains are stationary with a visible central tendency, have good mixing, and converge.

Evaluate model results

First, obtain tidy samples, as we did with the duration and GB hole model. Additionally, transform the result to get the absolute probability of CVS and the OR of a obtained CVS for each PGS:

```r
tidy_cvs_samples <- extract_samples(
    cvs_mod,
    pars = c(
        "a_bar",
        paste0("a[", 1:nsurgs, "]"),
        "bP_bar",
        paste0("bP[", 1:nsurgs, "]"),
        # sigma_a
        "sigma_sid[1]",
        # sigma_b
```

```
        "sigma_sid[2]",
        paste0("Rho_sid[1,2]"),
        paste0("delta[", 1:4, "]")
    )
) %>%
    tidy_surgeons() %>%
    sum_deltas() %>%
    tidy_pgs() %>%
    mutate(
        p = inv_logit(a + bP * sum_delta_j),
        bOR = exp(bP * sum_delta_j)
    ) %>%
    arrange(sample_num, surgeon, PGS) %>%
    group_by(surgeon, sample_num) %>%
    # calc change in probability for each sample
    # group by surgeon as well because each sample_num
    # has the info for all 10 surgeons and bar surgeon
    mutate(
        # change from one PGS level to the other
        incr_delta_p = p - lag(p),
        # change from PGS1 to that PGS level
        delta_p = p - p[1]
    ) %>%
    ungroup()
```

Key to know, is that:

`incr_delta_p` is the probability difference from one level to the next. It will be `NA` for a PGS1.

`delta_p` is probability difference from a certain PGS level to PGS1.

`p` is the probability of a cvs at that PGS level

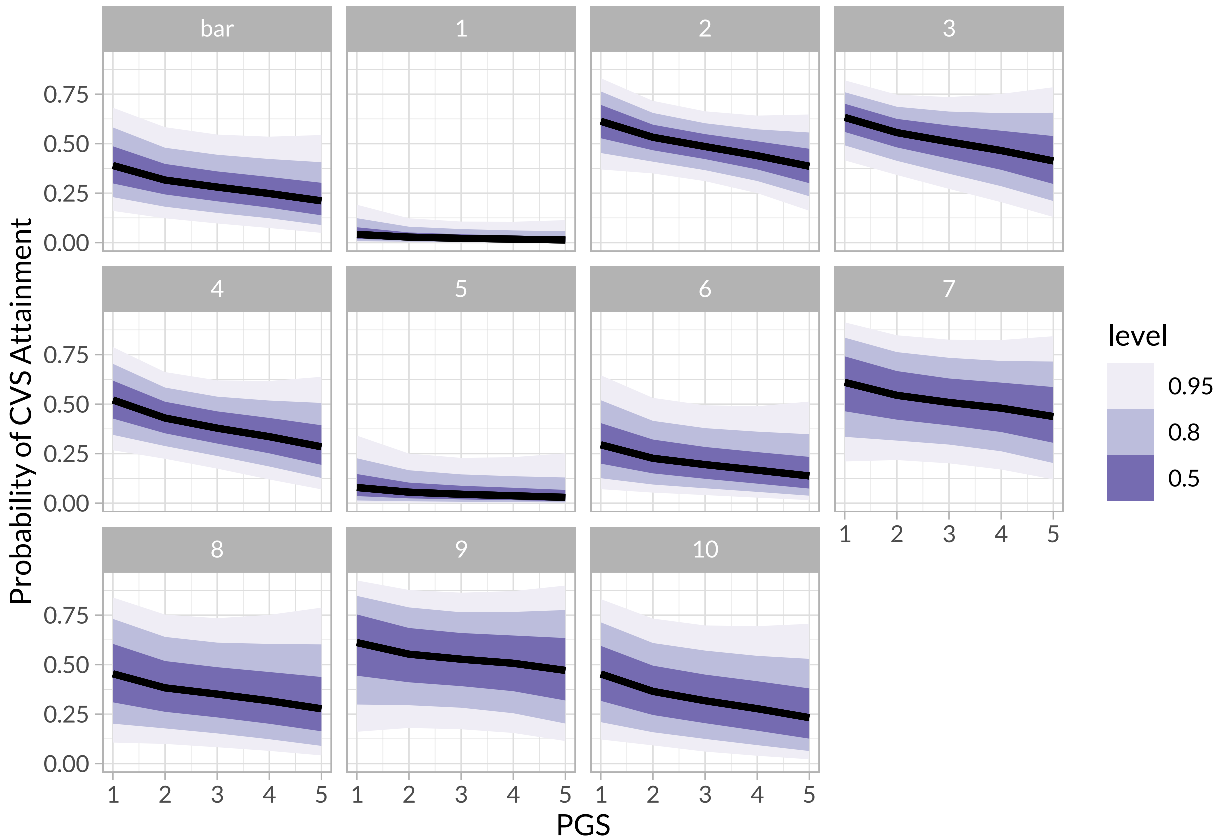
`bOR` is the increased odds, compared to a PGS of 1, of obtaining CVS

Plot results

For all surgeons

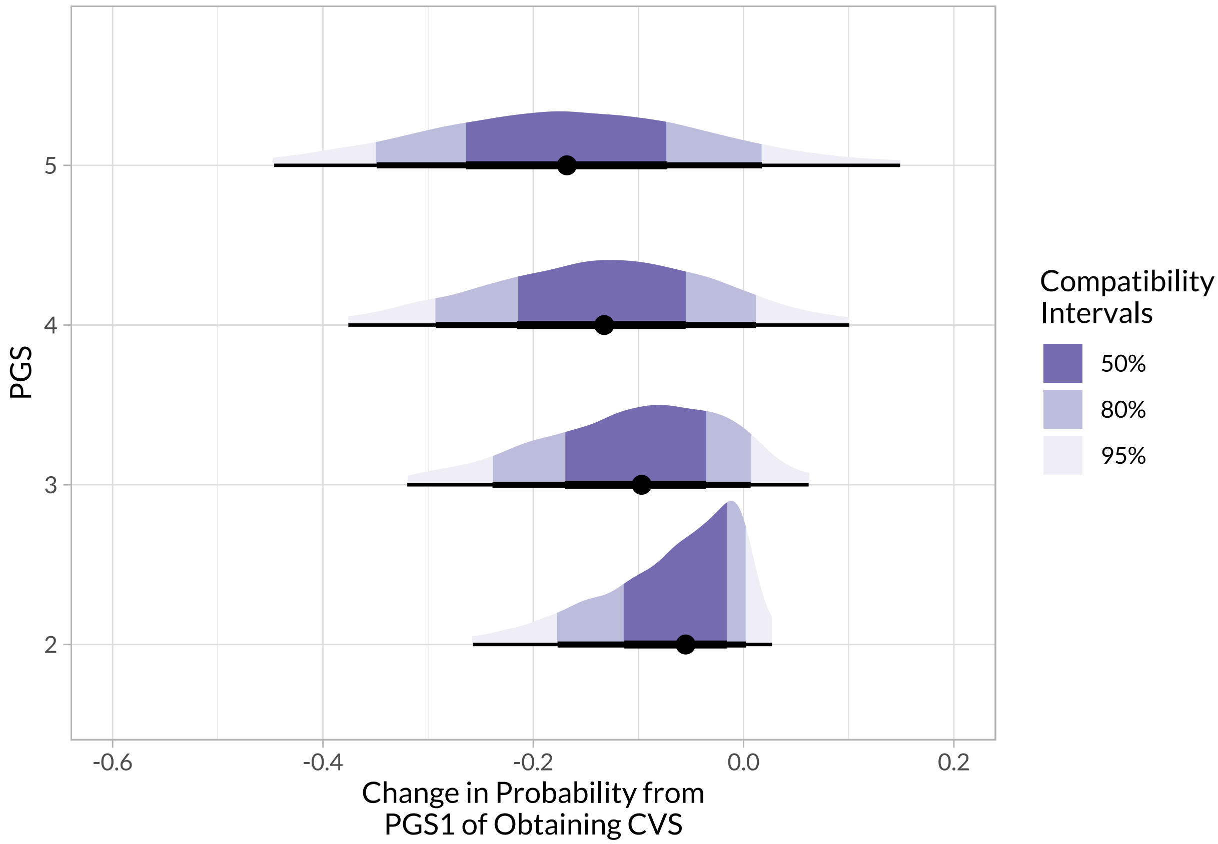
Remember that "bar" is the average of all the surgeons:

```
tidy_cvs_samples %>%
    mutate(surgeon = parse_factor(surgeon, levels = c("bar", as.character(1:nsurgs)))) %>%
    ggplot(aes(PGS, p)) +
    stat_lineribbon() +
    facet_wrap(~ surgeon) +
    scale_fill_brewer(palette = "Purples") +
    labs(y = "Probability of CVS Attainment")
```



On average across all surgeons

```r
bar_cvs <- tidy_cvs_samples %>%
    filter(surgeon == "bar", PGS != 1) %>%
    mutate(PGS = as.factor(PGS))
bar_cvs %>%
    halfeye(delta_p, PGS) +
    coord_cartesian(xlim = c(-0.6, 0.2)) +
    labs(
        y = "PGS",
        x = "Change in Probability from\nPGS1 of Obtaining CVS"
    )
```



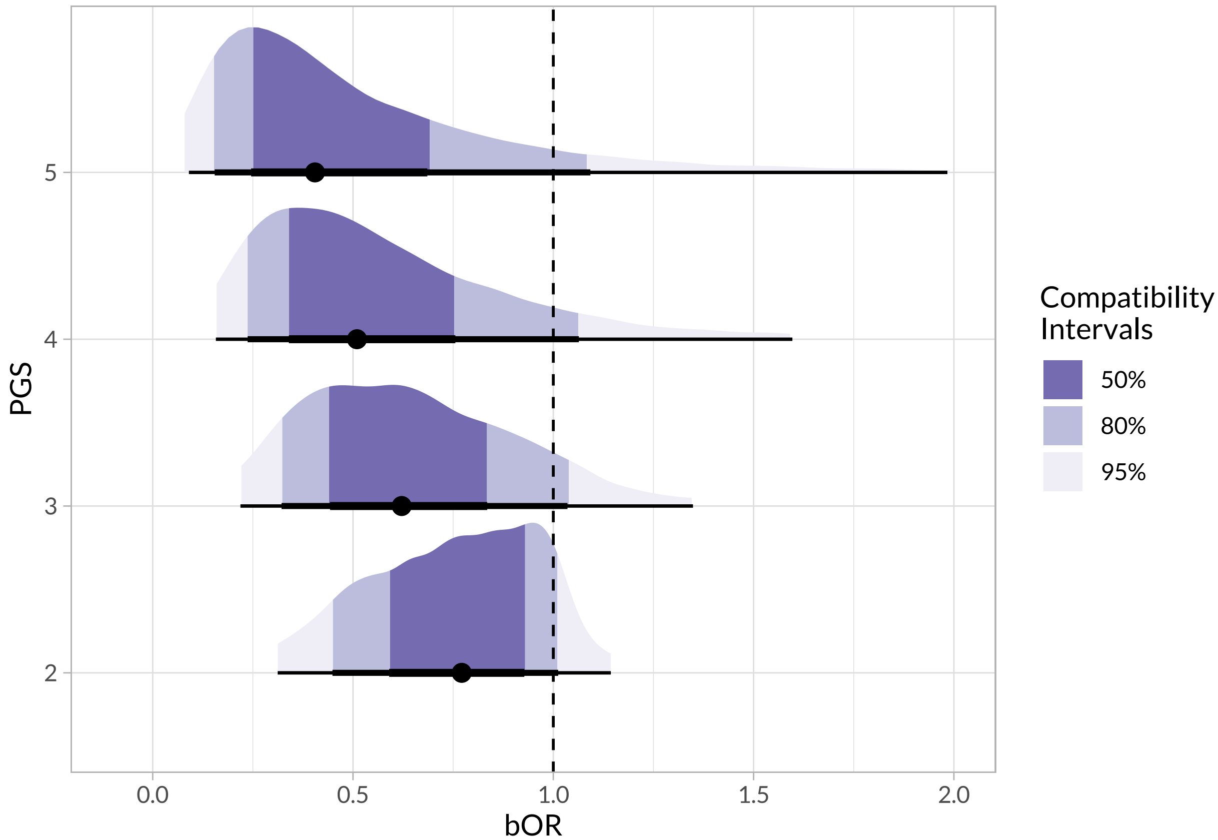
And numbers for this:

```
bar_cvs %>%
    group_by(PGS) %>%
    ci_ints(delta_p)
```

```
## # A tibble: 4 x 7
##   PGS    mean int_50       int_66       int_80       int_89     int_95
##   <fct> <dbl> <chr>        <chr>        <chr>        <chr>      <chr>
## 1 2     -0.07 -0.11, -0.02 -0.14, -0.01 -0.13, -0.01 -0.21, 0.~ -0.26, 0.~
## 2 3     -0.11 -0.17, -0.04 -0.2, -0.01  -0.19, -0.02 -0.28, 0.~ -0.32, 0.~
## 3 4     -0.14 -0.22, -0.06 -0.25, -0.02 -0.24, -0.04 -0.33, 0.~ -0.38, 0.1
## 4 5     -0.17 -0.26, -0.07 -0.3, -0.03  -0.29, -0.05 -0.39, 0.~ -0.45, 0.~
```

And looking at the odd ratios:

```
bar_cvs %>%
    halfeye(bOR, PGS) +
    coord_cartesian(xlim = c(-0.1, 2)) +
    geom_vline(xintercept = 1, linetype = 2)
```



Numbers for this:

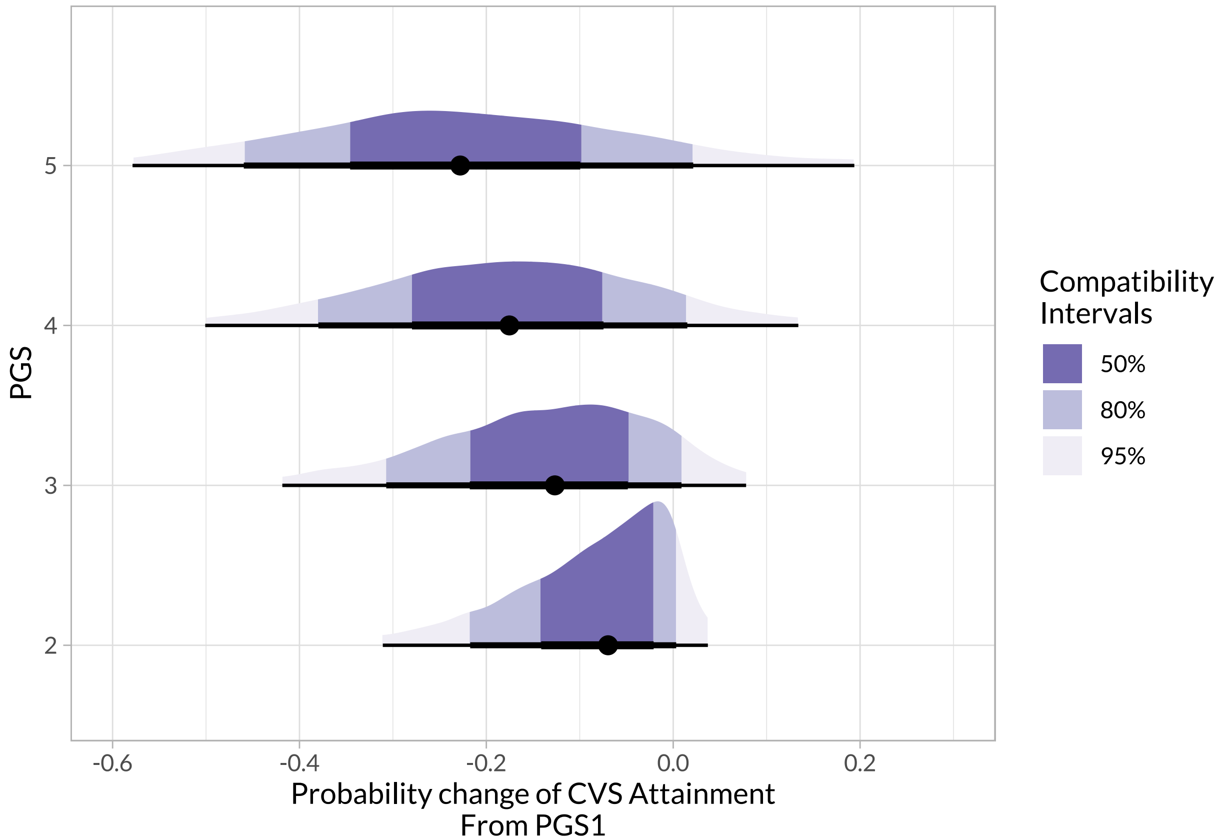
```
bar_cvs %>%
    group_by(PGS) %>%
    ci_ints(bOR)
```

```
## # A tibble: 4 x 7
##   PGS    mean int_50     int_66     int_80     int_89     int_95
##   <fct> <dbl> <chr>      <chr>      <chr>      <chr>      <chr>
## 1 2     0.75  0.59, 0.93 0.52, 0.97 0.55, 0.95 0.38, 1.06 0.31, 1.14
## 2 3     0.66  0.44, 0.84 0.38, 0.93 0.41, 0.89 0.27, 1.15 0.22, 1.35
## 3 4     0.61  0.34, 0.76 0.29, 0.88 0.31, 0.83 0.2, 1.28  0.16, 1.6
## 4 5     0.570 0.25, 0.69 0.2, 0.85  0.22, 0.78 0.12, 1.42 0.09, 1.98
```

For a surgeon particularly affected by PGS

```
surg4_cvs <- tidy_cvs_samples %>%
    filter(surgeon == "4", PGS != 1) %>%
    mutate(PGS = as.factor(PGS))

surg4_cvs %>%
    halfeye(delta_p, PGS) +
    coord_cartesian(xlim = c(-0.6, 0.3)) +
    labs(
        y = "PGS",
        x = "Probability change of CVS Attainment\nFrom PGS1"
    )
```



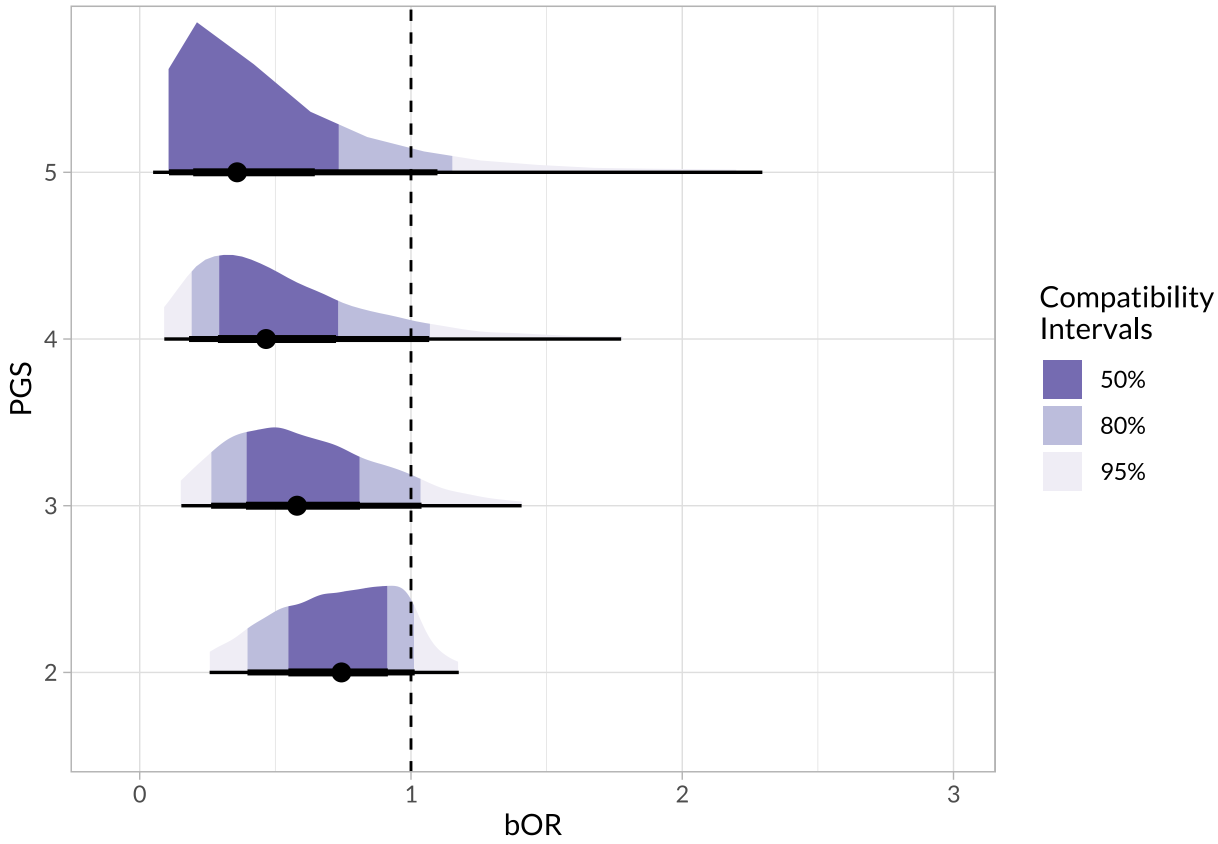
And numbers for this:

```
surg4_cvs %>%
    group_by(PGS) %>%
    ci_ints(delta_p)
```

```
## # A tibble: 4 x 7
##   PGS    mean int_50       int_66       int_80       int_89     int_95
##   <fct> <dbl> <chr>        <chr>        <chr>        <chr>      <chr>
## 1 2     -0.09 -0.14, -0.02 -0.17, -0.01 -0.16, -0.01 -0.26, 0.~ -0.31, 0.~
## 2 3     -0.14 -0.22, -0.05 -0.26, -0.02 -0.24, -0.03 -0.36, 0.~ -0.42, 0.~
## 3 4     -0.18 -0.28, -0.07 -0.33, -0.03 -0.31, -0.05 -0.43, 0.~ -0.5, 0.13
## 4 5     -0.22 -0.35, -0.1  -0.4, -0.04  -0.38, -0.07 -0.51, 0.~ -0.58, 0.~
```

And looking at odds ratios:

```
surg4_cvs %>%
    halfeye(bOR, PGS) +
    coord_cartesian(xlim = c(-0.1, 3)) +
    geom_vline(xintercept = 1, linetype = 2)
```



Numbers for this:

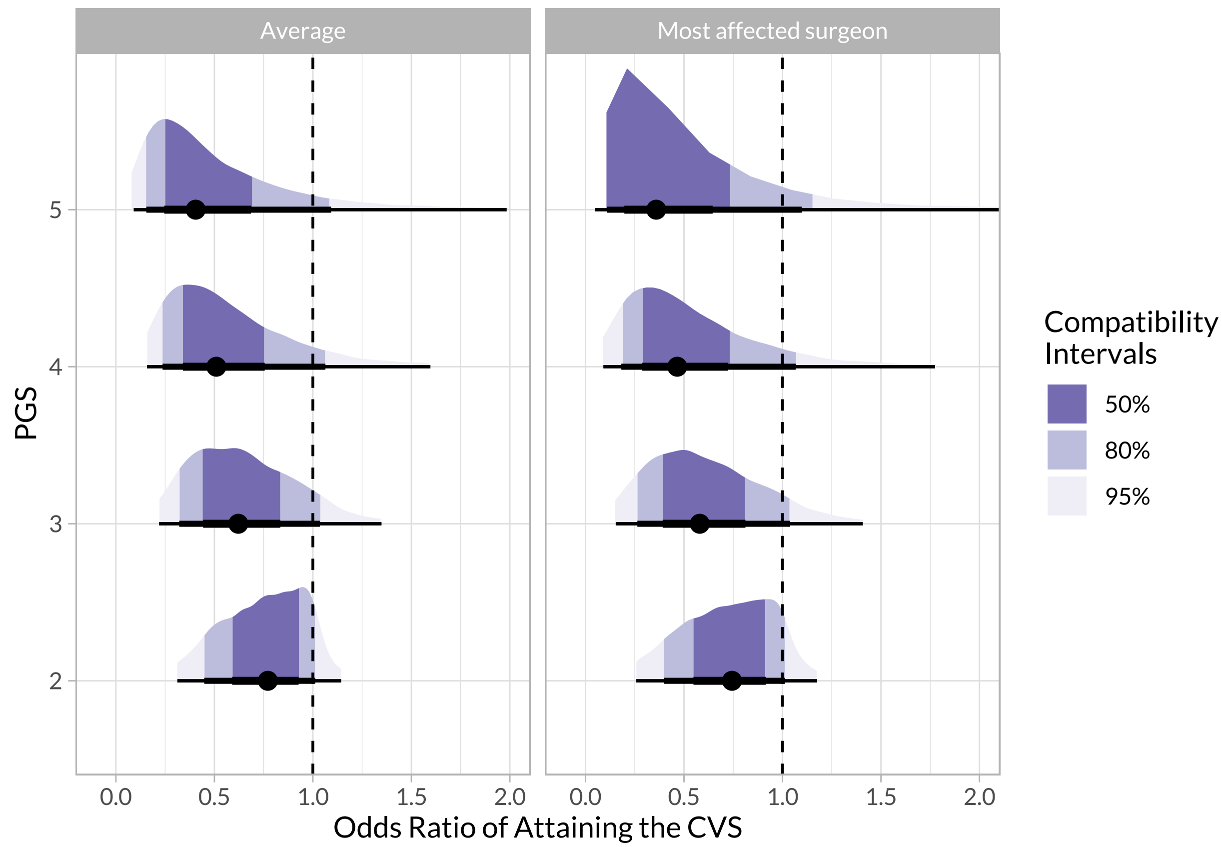
```
surg4_cvs %>%
    group_by(PGS) %>%
    ci_ints(bOR)
```

```
## # A tibble: 4 x 7
##   PGS    mean int_50     int_66     int_80     int_89     int_95
##   <fct> <dbl> <chr>      <chr>      <chr>      <chr>      <chr>
## 1 2     0.73  0.55, 0.91 0.48, 0.97 0.51, 0.95 0.32, 1.07 0.26, 1.18
## 2 3     0.63  0.39, 0.81 0.33, 0.92 0.35, 0.88 0.21, 1.18 0.15, 1.41
## 3 4     0.580 0.29, 0.72 0.23, 0.87 0.25, 0.81 0.14, 1.34 0.09, 1.77
## 4 5     0.580 0.2, 0.65  0.15, 0.83 0.17, 0.75 0.08, 1.49 0.05, 2.3
```

Average and surgeon most affected by PGS

```
tidy_cvs_samples %>%
    filter(surgeon %in% c("bar", "4"), PGS != 1) %>%
    mutate(
        surgeon = if_else(surgeon == "bar", "Average", "Most affected surgeon"),
        PGS = as.factor(PGS)
    ) %>%
    halfeye(bOR, PGS) +
    facet_wrap(~ surgeon) +
    coord_cartesian(xlim = c(-0.1, 2)) +
    geom_vline(xintercept = 1, linetype = 2) +
    labs(
```

```
        x = "Odds Ratio of Attaining the CVS",
        y = "PGS"
    )
```




```
ggsave("../output/pgs_cvs.svg", width = 6, height = 4)
ggsave("../output/pgs_cvs.pdf", width = 6, height = 4)
```

Correlation of PGS1 and effect of incremental PGS on CVS

```
extract_samples(cvs_mod, pars = c("Rho_sid[1,2]")) %>%
    rename(rho = Rho_sid_1_2) %>%
    ci_ints(rho) %>%
    knitr::kable(
        caption = "Correlation between attaining CVS in PGS1 case and the effect of incrementing PGS"
    )
```

Table 9: Correlation between attaining CVS in PGS1 case and the effect of incrementing PGS

mean	int_50	int_66	int_80	int_89	int_95
0	-0.24, 0.24	-0.34, 0.34	-0.3, 0.3	-0.54, 0.54	-0.63, 0.62

Summarise inputs/outcomes

Numbers

Number of surgeons:

```
nsurgs
```

```
## [1] 10
```

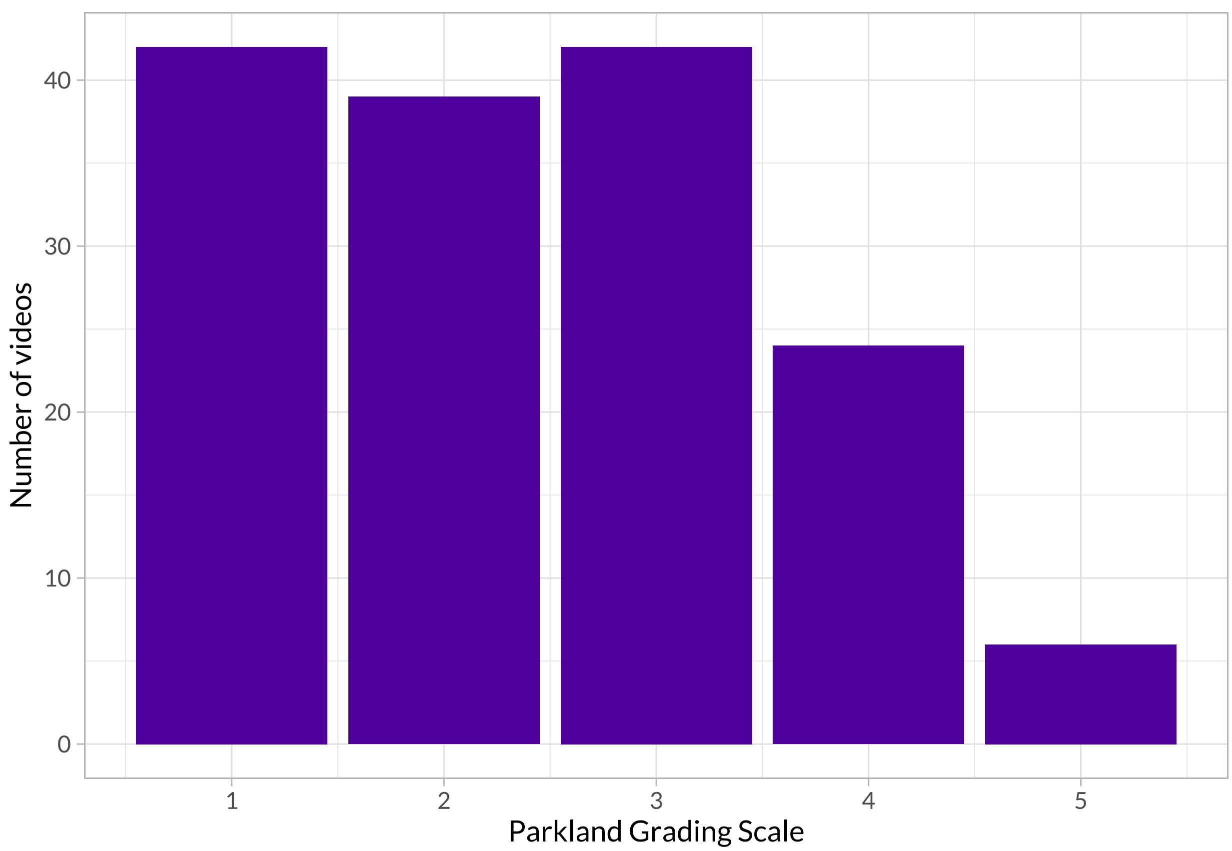
Number of videos with surgeons >= 5 cases:

```
nrow(dat)
```

```
## [1] 153
```

PGS

```
dat %>%
    ggplot(aes(pgs)) +
    geom_bar(fill = histo_color) +
    labs(
        x = "Parkland Grading Scale",
        y = "Number of videos"
    )
```



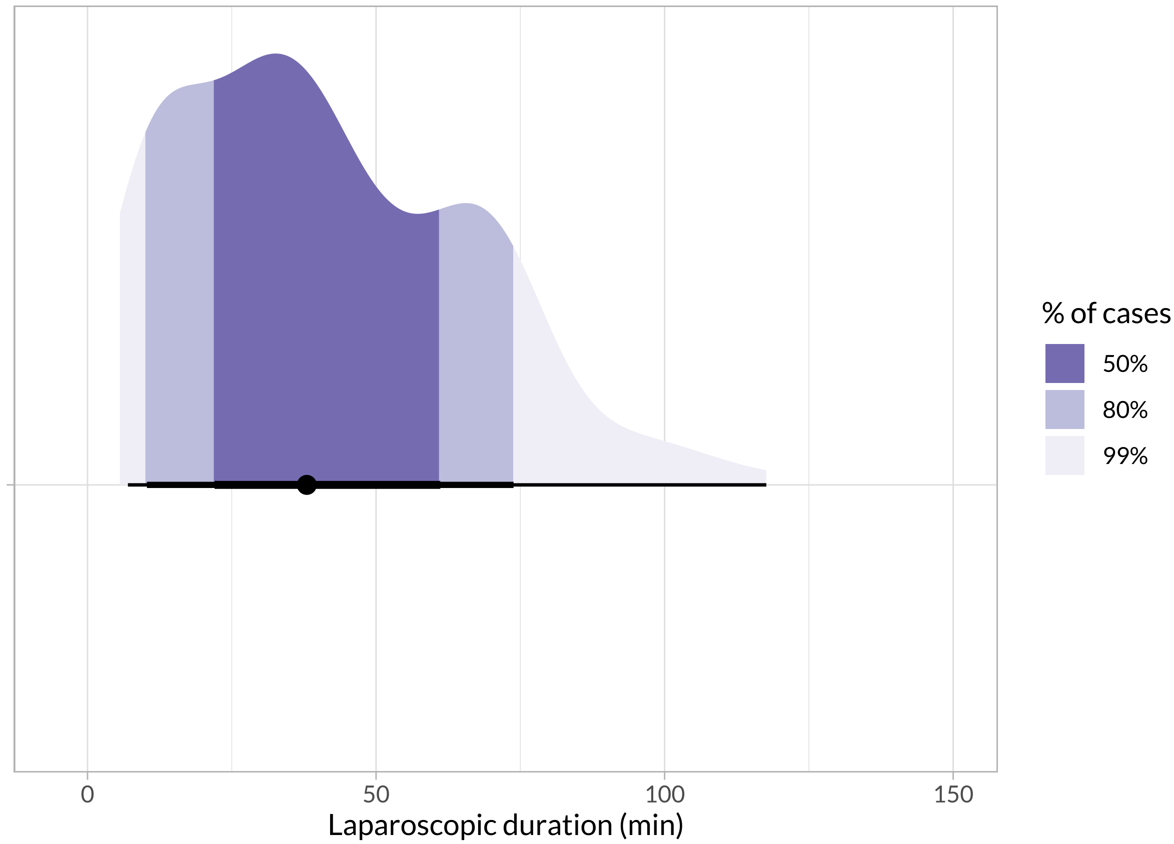

```
ggsave("../output/pgs_distribution.svg", width = 6, height = 6)
ggsave("../output/pgs_distribution.pdf", width = 6, height = 6)
```

```r
dat %>%
    count(pgs) %>%
    mutate(prop = n / sum(n)) %>%
    mutate(cum.sum = cumsum(prop))
```

```
## # A tibble: 5 x 4
##     pgs     n   prop cum.sum
##   <int> <int>  <dbl>   <dbl>
## 1     1    42 0.275    0.275
## 2     2    39 0.255    0.529
## 3     3    42 0.275    0.804
## 4     4    24 0.157    0.961
## 5     5     6 0.0392   1
```

Laparoscopic duration

```r
dat %>%
    ggplot(aes(y = "", x = laparascopic_duration)) +
    stat_halfeye(
            aes(
                fill = stat(
                    cut_cdf_qi(cdf, .width = c(0.5, 0.8, 0.99), labels = scales::percent_format())
                )
            ),
            .width = c(0.5, 0.8, 0.99)
        ) +
        scale_fill_brewer(direction = -1, palette = "Purples", na.translate = FALSE) +
    labs(fill = "% of cases", y = "", x = "Laparoscopic duration (min)") +
    coord_cartesian(xlim = c(-5, 150))
```

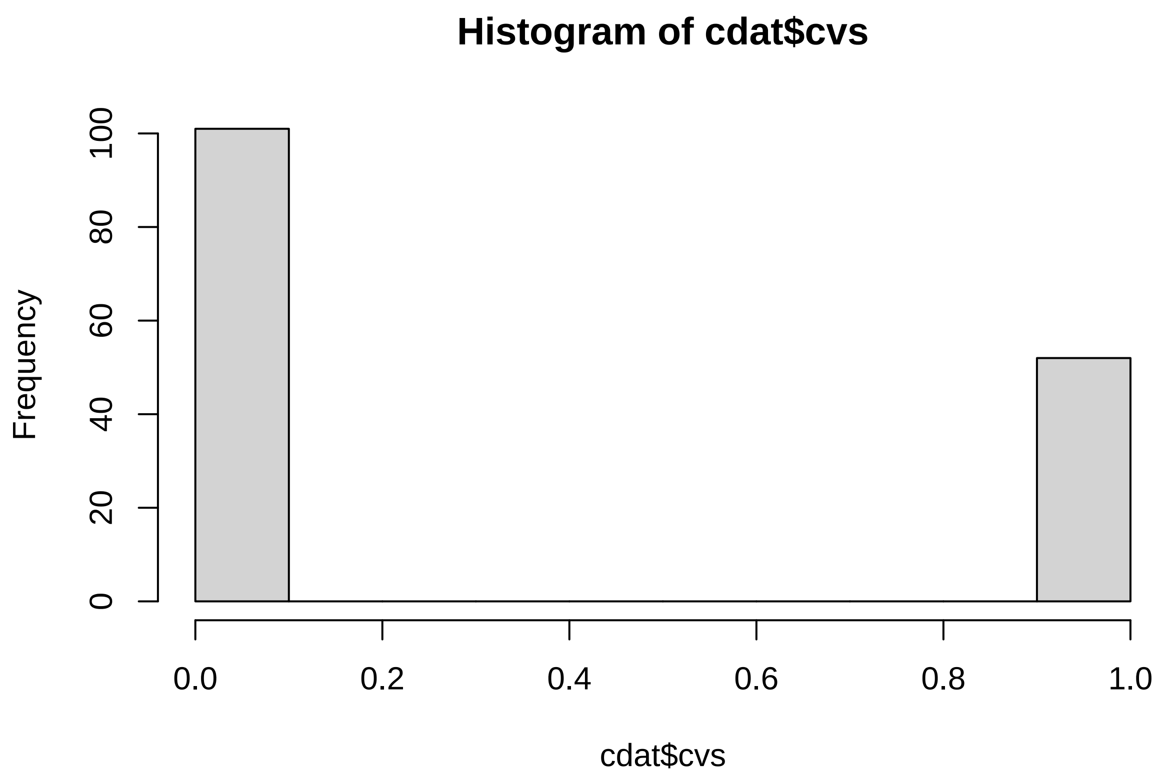



```
summary(dat$laparascopic_duration)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   5.617  22.017  38.000  41.296  61.133 131.600
```

CVS number of cases with attainment

```
hist(cdat$cvs)
```




```
mean(cdat$cvs)
```

```
## [1] 0.3398693
```

GB holes

```
hist(gdat$hole)
```


```r
mean(gdat$hole)
```

```
## [1] 0.254902
```

Environment

```r
devtools::session_info()
```

```
## - Session info ----------------------------------------------------------
##  setting  value
##  version  R version 4.0.5 (2021-03-31)
##  os       Fedora 34 (Workstation Edition)
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       America/New_York
##  date     2021-08-29
##
## - Packages --------------------------------------------------------------
##  package        * version  date       lib
##  abind            1.4-5    2016-07-21 [1]
##  assertthat       0.2.1    2019-03-21 [1]
##  backports        1.2.1    2020-12-09 [1]
##  base64enc        0.1-3    2015-07-28 [1]
```

```
##  blob             1.2.1    2020-01-20 [1]
##  callr            3.6.0    2021-03-28 [1]
##  checkmate        2.0.0    2020-02-06 [1]
##  cli              2.3.1    2021-02-23 [1]
##  cmdstanr       * 0.3.0    2021-03-29 [1]
##  coda             0.19-4   2020-09-30 [1]
##  codetools        0.2-18   2020-11-04 [2]
##  colorspace       2.0-0    2020-11-11 [1]
##  crayon           1.4.1    2021-02-08 [1]
##  curl             4.3      2019-12-02 [1]
##  data.table       1.13.2   2020-10-19 [1]
##  DBI              1.1.0    2019-12-15 [1]
##  desc             1.3.0    2021-03-05 [1]
##  devtools         2.3.2    2020-09-18 [1]
##  digest         * 0.6.27   2020-10-24 [1]
##  distributional   0.2.2    2021-02-02 [1]
##  dplyr          * 1.0.5    2021-03-05 [1]
##  ellipsis         0.3.1    2020-05-15 [1]
##  evaluate         0.14     2019-05-28 [1]
##  fansi            0.4.2    2021-01-15 [1]
##  farver           2.1.0    2021-02-28 [1]
##  forcats        * 0.5.1    2021-01-27 [1]
##  fs               1.5.0    2020-07-31 [1]
##  gdtools        * 0.2.3    2021-01-06 [1]
##  generics         0.1.0    2020-10-31 [1]
##  ggdist         * 2.4.0    2021-01-04 [1]
##  ggplot2        * 3.3.3    2020-12-30 [1]
##  glue             1.4.2    2020-08-27 [1]
##  gridExtra        2.3      2017-09-09 [1]
##  gtable           0.3.0    2019-03-25 [1]
##  highr            0.8      2019-03-20 [1]
##  hms              0.5.3    2020-01-08 [1]
##  htmltools        0.5.1.1  2021-01-22 [1]
##  inline           0.3.17   2020-12-01 [1]
##  janitor          2.0.1    2020-04-12 [1]
##  jsonlite         1.7.2    2020-12-09 [1]
##  knitr            1.30     2020-09-22 [1]
##  labeling         0.4.2    2020-10-20 [1]
##  lattice          0.20-41  2020-04-02 [2]
##  lifecycle        1.0.0    2021-02-15 [1]
##  loo              2.4.1    2020-12-09 [1]
##  lubridate        1.7.9    2020-06-08 [1]
##  magrittr       * 2.0.1    2020-11-17 [1]
##  MASS             7.3-53.1 2021-02-12 [2]
##  matrixStats      0.58.0   2021-01-29 [1]
##  memoise          1.1.0    2017-04-21 [1]
##  munsell          0.5.0    2018-06-12 [1]
##  mvtnorm          1.1-1    2020-06-09 [1]
##  nvimcom        * 0.9-102  2021-05-17 [1]
##  pillar           1.5.1    2021-03-05 [1]
##  pkgbuild         1.2.0    2020-12-15 [1]
##  pkgconfig        2.0.3    2019-09-22 [1]
##  pkgload          1.1.0    2020-05-29 [1]
##  posterior        0.1.4    2021-03-29 [1]
```

```
## prettyunits      1.1.1    2020-01-24 [1]
## processx         3.5.0    2021-03-23 [1]
## ps               1.6.0    2021-02-28 [1]
## purrr            0.3.4    2020-04-17 [1]
## R6               2.5.0    2020-10-28 [1]
## RColorBrewer     1.1-2    2014-12-07 [1]
## Rcpp             1.0.6    2021-01-15 [1]
## RcppParallel     5.0.3    2021-02-24 [1]
## readr          * 1.4.0    2020-10-05 [1]
## remotes          2.2.0    2020-07-21 [1]
## repr             1.1.0    2020-01-28 [1]
## rethinking     * 2.13     2020-10-24 [1]
## rlang            0.4.10   2020-12-30 [1]
## rmarkdown      * 2.5      2020-10-21 [1]
## rprojroot        2.0.2    2020-11-15 [1]
## rstan          * 2.21.2   2020-07-27 [1]
## rstudioapi       0.11     2020-02-07 [1]
## scales           1.1.1    2020-05-11 [1]
## sessioninfo      1.1.1    2018-11-05 [1]
## shape            1.4.5    2020-09-13 [1]
## showtext       * 0.9-2    2021-01-10 [1]
## showtextdb     * 3.0      2020-06-04 [1]
## skimr            2.1.2    2020-07-06 [1]
## snakecase        0.11.0   2019-05-25 [1]
## StanHeaders    * 2.21.0-7 2020-12-17 [1]
## stringi          1.5.3    2020-09-09 [1]
## stringr        * 1.4.0    2019-02-10 [1]
## svglite          1.2.3.2  2020-07-07 [1]
## sysfonts       * 0.8.3    2021-01-10 [1]
## systemfonts      0.3.2    2020-09-29 [1]
## tensorA          0.36.2   2020-11-19 [1]
## testthat         3.0.3    2021-06-16 [1]
## tibble           3.1.0    2021-02-25 [1]
## tidyr          * 1.1.3    2021-03-03 [1]
## tidyselect       1.1.0    2020-05-11 [1]
## usethis          1.6.3    2020-09-17 [1]
## utf8             1.2.1    2021-03-12 [1]
## V8               3.4.0    2020-11-04 [1]
## vctrs            0.3.7    2021-03-29 [1]
## withr            2.4.1    2021-01-26 [1]
## xfun             0.18     2020-09-29 [1]
## yaml             2.2.1    2020-02-01 [1]
## source
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## local
## CRAN (R 4.0.3)
## CRAN (R 4.0.5)
```

```
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.5)
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.5)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## local
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.3)
## local
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
## CRAN (R 4.0.3)
## CRAN (R 4.0.4)
```

```
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  Github (rmcelreath/rethinking@3b48ec8)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.5)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.4)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##
## [1] /home/thomas/R/x86_64-redhat-linux-gnu-library/4.0
## [2] /usr/lib64/R/library
## [3] /usr/share/R/library
```