Exam 2023

2024-01-23

Exercise 1

Let X be uniform on the unit interval. Let $Y|X = x \sim \mathcal{N}(0, x)$.

1.

We can simulate from the joint distribution of (X, Y) by first simulating from the marginal of X and then using the conditional distribution of Y given X to simulate from Y. We count the number of times out of $n = 10^5$ that $Y < 1$ to get an estimate of $P(Y < 1)$. Recall that `r` parametrizes the normal distribution with the standard deviation, so we take the square root of X below

```
set.seed(1)
n <- 10e5
X <- runif(n)
Y <- rnorm(n, 0, sqrt(X))
mean(Y < 1)
```

```
## [1] 0.924313
```

We see that $P(Y < 1) \approx 0.924313$.

2.

By the law of total expectation

$$E(Y) = E(E(Y|X)) = E(0) = 0$$

And by theorem 3.3

$$V(Y) = V(E(Y|X)) + E(V(Y|X)) = V(0) + E(X) = 1/2$$

3.

Since X and $Y|X = x$ both have densities w.r.t. the Lebesgue measure which is σ-finite, it follows from theorem 2.1 that (X, Y) has joint density h w.r.t. the 2-dimensional lebesgue measure. We have that

$$h(x, y) = q(x) \cdot g_x(y) = 1_{[0,1]}(x) \cdot \frac{1}{\sqrt{2\pi x}} \exp(-\frac{y^2}{2x})$$

It now follows from theorem 2.2 that the conditional distribution of $X|y = t$ exists and has density w.r.t. the Lebesgue measure which satisfies by

$$q_y(x) \propto h(x, y) = 1_{[0,1]}(x) \cdot \frac{1}{\sqrt{2\pi x}} \exp(-\frac{y^2}{2x}) \propto 1_{(0,1)}(x) \cdot x^{-1/2} \exp(-\frac{y^2}{2x})$$

4.

Consider the case $Y = 0$. Then

$$q_0(x) \propto 1_{(0,1)}(x) \cdot x^{-1/2} \exp(0) = 1_{(0,1)}(x) \cdot x^{-1/2}$$

Recall that the $\chi^2(1)$ has density

$$f(x) = \frac{1}{2^{1/2}\Gamma(1/2)} x^{-1/2} \exp(-x/2) 1_{(0,\infty)}(x) = Cx^{-1/2} \exp(-x/2) 1_{(0,\infty)}(x)$$

Letting $C = \frac{1}{2^{1/2}\Gamma(1/2)}$. We can clearly sample from the $\chi^2(1)$ distribution, and noting that for some constant $M > 0$

$$1_{(0,1)}(x) \cdot x^{-1/2} \leq M \cdot C \cdot x^{-1/2} \exp(-x/2) 1_{(0,\infty)}(x) \Leftrightarrow 1_{(0,1)}(x) \leq M \cdot C \cdot \exp(-x/2) 1_{(0,\infty)}(x)$$

If $x \geq 1$ the inequality trivially holds. If $0 \leq x < 1$ we have that $\exp(-1/2) < \exp(-x/2) \leq 1$, hence if we want the inequality to hold for all $x \in (0, 1)$ we have to solve

$$1 \leq M \cdot C \cdot \exp(-1/2) \Leftrightarrow \frac{\exp(1/2)}{C} \leq M$$

The value M that maximizes the acceptance probability is the M that ensures equality in the above

$$M = \frac{\exp(1/2)}{C} = 2^{1/2}\Gamma(1/2) \exp(1/2)$$

With this choice of M we obtain that

$$1_{(0,1)}(x) \cdot x^{-1/2} \leq M \cdot f(x)$$

Which shows that we can use rejecetion sampling with $\chi^2(1)$ as the proposal to sample from Q_0. The probability that a sample x^* will be accepted is

$$\alpha = \frac{1_{(0,1)}(x^*) \cdot (x^*)^{-1/2}}{Mf(x^*)}$$

Which could probably be simplifed. We simulate from Q_0 and compute $E(Y|X = 0)$.

```
set.seed(1)
C <- 1/(2^(1/2)*gamma(1/2))
M <- exp(1/2)/C
proposal <- function(x){
  dchisq(x, 1)
}
target <- function(x){
  (x>0)*(x<1)*x^(-1/2)
}
sample_dist <- function(n){
  rchisq(n=n, df = 1)
}

rejection_sample <- function(M, proposal, target, sample_dist, print_acceptance = TRUE){
  theta_star <- sample_dist(n = 1e6)
  U <- runif(1e6)
  alpha <- target(theta_star)/(M*proposal(theta_star))
  idx <- (U <= alpha)
  if(print_acceptance){
```

```
    cat("Acceptance rate is: ", sum(idx)/1e6)
  }
  return(theta_star[idx])
}
X <- rejection_sample(M, proposal = proposal, target = target, sample_dist = sample_dist)
```

```
## Acceptance rate is:  0.484515
```

```
mean(X)
```

```
## [1] 0.3334866
```

We get $E(Y|X=0) \approx 0.3329909$.

COMMENT: It is actually not necessary to carry around the constant C for the χ^2 distribution in the above. If only calculating up to proportionality it will cancel when computing α. This would mean that the sample_dist parameter above should be the unnormalized proposal instead. Accordingly M would change from $M = \exp(1/2)/C$ to $M = \exp(1/2)$.

Exercise 2

We load the data and add $\log(SOC)$ as a variable in the dataset.

```
rm(list = ls())
load("april22.Rdata")
SOCdata <- SOCdata %>% tibble() %>% mutate(log_SOC = log(SOC))
```

1.

It is most appropriate to use plant as a random factor, as we want to view the plants as a random sample from the population of plants, such that we can draw inference for the population of plants as a whole and not just about the specific plants in the dataset. Furthermore, measurements from the same plant is likely correlated, so it would be inappropriate to model observations from the same plant as independent.

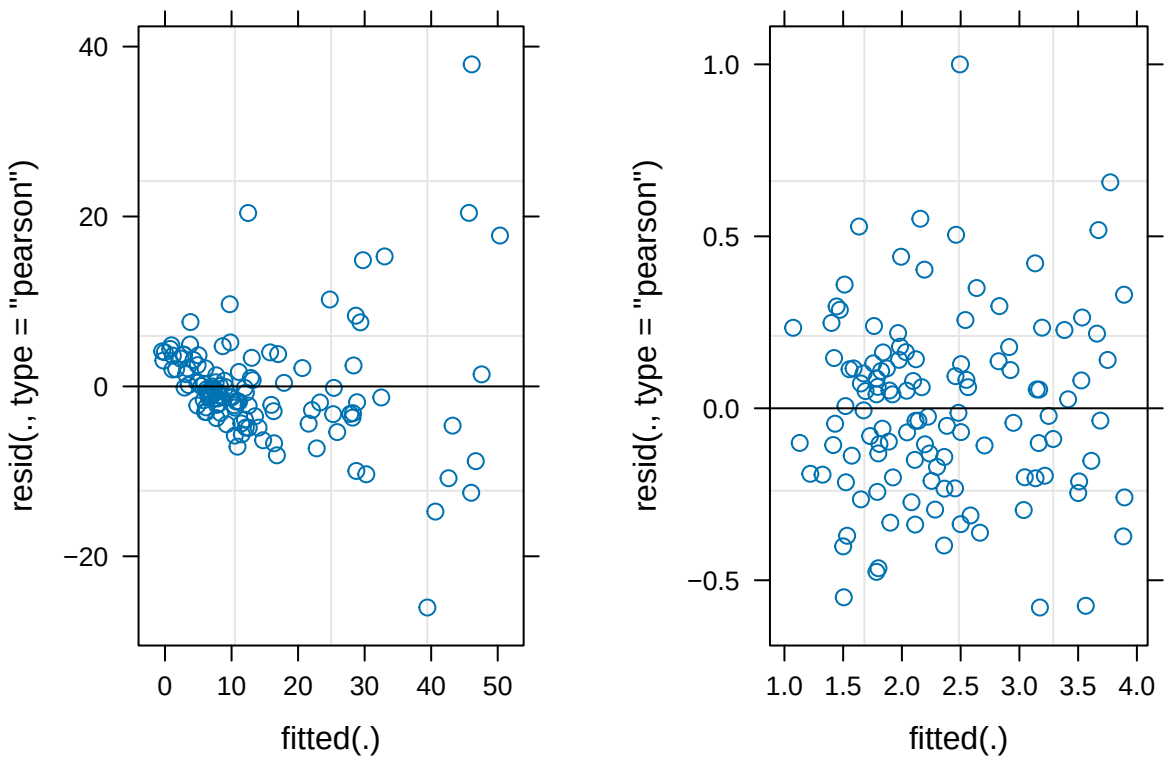
In this experiment it is most appropriate to model block as a fixed effect rather than a random effect because the 'were deliberately chosen such that they represent three different soil conditions'. This means that we are not interested in viewing blocks as a sample from the 'population of blocks', but rather we may be interested in saying something about the soil conditions effect on soil organic carbon. Hence, we should model it as a fixed effect.

We fit `M1` and `M2`. And make residual plots. Note that the plot on the left is the one were SOC is not log-transformed and the one on the right is the log-transformed version

```
M1 <- lmer(SOC ~ Treat -1 + Block*DepthFac + (1|Plant), data=SOCdata)
```

```
## boundary (singular) fit: see help('isSingular')
```

```
M2 <- lmer(log_SOC ~ Treat -1 + Block*DepthFac + (1|Plant), data=SOCdata)
gridExtra::grid.arrange(
  plot(M1),
  plot(M2),
  ncol = 2
)
```



We see a clear pattern in the residual plot from the non-logtransformed model. There is clear overdispersion/variance inhomogeneity as variance seems to increase with fitted values. The mean value structure also looks a bit off for small fitted values as there seems to be a downward trend in the residuals for small fitted values. The residual after log-transformation is much nicer. There is no sign of variance inhomogeneity and the mean structure also seems good.

2.

We print a summary of `M2`

```
summary(M2)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: log_SOC ~ Treat - 1 + Block * DepthFac + (1 | Plant)
##    Data: SOCdata
##
## REML criterion at convergence: 106.7
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.8936 -0.6336 -0.0315  0.4621  3.2692
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  Plant    (Intercept) 0.02366  0.1538
##  Residual             0.09356  0.3059
## Number of obs: 120, groups:  Plant, 30
```

```
##
## Fixed effects:
##                     Estimate Std. Error t value
## TreatA               3.19221    0.16074  19.859
## TreatB               3.50934    0.16074  21.832
## TreatC               3.49745    0.16074  21.758
## TreatD               3.21403    0.16074  19.995
## TreatE               3.37264    0.16074  20.982
## TreatF               3.47085    0.16074  21.593
## TreatG               3.07392    0.16074  19.124
## TreatH               3.23905    0.16074  20.151
## TreatI               3.08815    0.16074  19.212
## TreatControl         2.86858    0.16074  17.846
## Blockb               0.00574    0.15312   0.037
## Blockc               0.43173    0.15312   2.820
## DepthFac25          -0.58466    0.13679  -4.274
## DepthFac55          -1.05250    0.13679  -7.694
## DepthFac100         -1.61424    0.13679 -11.801
## Blockb:DepthFac25   -0.43965    0.19345  -2.273
## Blockc:DepthFac25   -0.80589    0.19345  -4.166
## Blockb:DepthFac55   -0.35552    0.19345  -1.838
## Blockc:DepthFac55   -0.79753    0.19345  -4.123
## Blockb:DepthFac100  -0.07874    0.19345  -0.407
## Blockc:DepthFac100  -0.48591    0.19345  -2.512
##
## Correlation matrix not shown by default, as p = 21 > 12.
## Use print(x, correlation=TRUE)  or
##     vcov(x)        if you need it
```

We see that the residual variance is estimated to $\hat{\sigma} = 0.09356$ and the within plant variance is estimated to $\hat{\sigma}_p = 0.02366$. This implies that

$$\hat{V}(Y_1) = \hat{\sigma}^2 + \hat{\sigma}_p^2 = 0.02366 + 0.09356 = 0.11722$$

If we have another measurement from the same plant but at different depths, an estimate for $\rho(Y_1, Y_2)$ is

$$\hat{\rho}(Y_1, Y_2) = \frac{\hat{\sigma}_p^2}{\hat{\sigma}^2 + \hat{\sigma}_p^2} = \frac{0.02366}{0.11722} = 0.2018427$$

3

Consider the models `M2` and `M3`. When we test the model reduction from `M2` to `M3` we are testing the hypothesis

$$H_0 : EY \in L_{Block \times DepthFac}$$

Against the larger model

$$EY \in L_{Treat} + L_{Block \times DepthFac}$$

We do this by computing a Likelihood ratio test statistics. Notice that we have 10 treatments, so we should compare the likelihood ratio test statistic to a $\chi^2(10-1)$ distribution, i.e. a χ^2 distribution with 9 degrees of freedom. We carry out the test with the `PBModComp` function from `pbkrtest`

```
M3 <- lmer(log_SOC ~ Block*DepthFac + (1|Plant), data=SOCdata)
PBmodcomp(M2, M3, nsim = 1000)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00265799 (tol = 0.002, component 1)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00238568 (tol = 0.002, component 1)
```

```
## Bootstrap test; time: 25.22 sec; samples: 1000; extremes: 28;
## large : log_SOC ~ Treat - 1 + Block * DepthFac + (1 | Plant)
## log_SOC ~ Block * DepthFac + (1 | Plant)
##         stat df  p.value
## LRT    26.46  9 0.001717 **
## PBtest 26.46    0.028971 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that we get a LRT statistic of 26.46, and the corresponding p-value is 0.001717. This is significant, and hence we reject H_0 - and even fairly strongly so.

4.

In question 3, we actually also simulated 1000 LRT statistics from the model. From these we get a p-value of 0.028971 This is still significant at a 5% level, but it is an order of magnitude larger than the χ^2-approximation and all in all much less convicing, even though we would still reject H_0.

5.

Consider `M2`. Let $\alpha_A, \cdots \alpha_I, \alpha_{control}$ be the first ten parameters of the model and let $\delta = \frac{1}{9}(alpha_A + \cdots + \alpha_I) - \alpha_{control}$. That is, δ is the average of the log-SOC for giving some treatment subtracted with the log-SOC level for the control group. Hence, δ can be interpreted as the average effect (change relative to control) of giving some treatment. We compute an estimate for δ.

```
(mean(fixef(M2)[1:9]) - fixef(M2)[10]) %>% unname()
```

```
## [1] 0.4267139
```

And so $\hat{\delta} = 0.4267139$. Notice that we could also have computed this as

$$A^T \begin{pmatrix} \alpha_A & \alpha_B & \cdots & \alpha_I & \alpha_{control} \end{pmatrix}$$

$$A = \begin{pmatrix} 1/9 \\ 1/9 \\ \vdots \\ -1 \end{pmatrix}$$

Let $\hat{\Sigma}$ be the estimated covariance matrix for the α estimates. Per the above, and since the estimates has a normal distribution under the model assumptions, we can compute an estimate of the variance for δ as

$$\hat{V}(\hat{\delta}) = A^T \hat{\Sigma} A$$

And so an estimate for the standard error is

```
A <- matrix(c(rep(1/9,9),-1),1,10)
sqrt(A %*% vcov(M2)[1:10,1:10] %*% t(A))[1,1]
```

```
## [1] 0.1320083
```

And so

$$\hat{SE}(\hat{\delta}) = 0.1320083$$

6.

Consider `M4`

```
M4 <- lmer(log_SOC ~ Treat2 + Block*DepthFac + (1|Treat) + (1|Plant),
data=SOCdata)
```

In this model, the estimate for `Treat2` is exactly the expected difference in log(SOC) between an average EFB treatment and control. Looking at the summary we see that

```
summary(M4)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: log_SOC ~ Treat2 + Block * DepthFac + (1 | Treat) + (1 | Plant)
##    Data: SOCdata
##
## REML criterion at convergence: 103.4
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.9879 -0.6150  0.0304  0.5284  3.3361
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  Plant    (Intercept) 0.02366  0.1538
##  Treat    (Intercept) 0.01379  0.1174
##  Residual             0.09356  0.3059
## Number of obs: 120, groups:  Plant, 30; Treat, 10
##
## Fixed effects:
##                     Estimate Std. Error t value
## (Intercept)          2.86858    0.19907  14.410
## Treat2EFB            0.42671    0.18097   2.358
## Blockb               0.00574    0.15312   0.037
## Blockc               0.43173    0.15312   2.820
## DepthFac25          -0.58466    0.13679  -4.274
## DepthFac55          -1.05250    0.13679  -7.694
## DepthFac100         -1.61424    0.13679 -11.801
## Blockb:DepthFac25   -0.43965    0.19345  -2.273
## Blockc:DepthFac25   -0.80589    0.19345  -4.166
## Blockb:DepthFac55   -0.35552    0.19345  -1.838
## Blockc:DepthFac55   -0.79753    0.19345  -4.123
## Blockb:DepthFac100  -0.07874    0.19345  -0.407
## Blockc:DepthFac100  -0.48591    0.19345  -2.512
##
## Correlation matrix not shown by default, as p = 13 > 12.
## Use print(x, correlation=TRUE)  or
##     vcov(x)        if you need it
```

Hence our estimate $\hat{\delta}'$ for the expected difference in log(SOC) between an average EFB treatment and control is thus

$$\hat{\delta}' = 0.42671$$

and the associated standard error is

$$\hat{SE}(\hat{\delta}') = 0.18097$$

Notice that the estimate is exactly the same due to the balanced design, but that the standard error is know

higher, likely due to the fact that we have included treatment as a random effect. If we fit the model with `lmerTest` we get a p-value from t-distribution for the $\hat{\delta}'$ estimate

```
lmerTest::lmer(log_SOC ~ Treat2 + Block*DepthFac + (1|Treat) + (1|Plant),
data=SOCdata) %>% summary()
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: log_SOC ~ Treat2 + Block * DepthFac + (1 | Treat) + (1 | Plant)
##    Data: SOCdata
##
## REML criterion at convergence: 103.4
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.9879 -0.6150  0.0304  0.5284  3.3361
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  Plant    (Intercept) 0.02366  0.1538
##  Treat    (Intercept) 0.01379  0.1174
##  Residual             0.09356  0.3059
## Number of obs: 120, groups:  Plant, 30; Treat, 10
##
## Fixed effects:
##                     Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)          2.86858    0.19907 14.30854  14.410 6.47e-10 ***
## Treat2EFB            0.42671    0.18097  7.99980   2.358  0.04611 *
## Blockb               0.00574    0.15312 74.77081   0.037  0.97020
## Blockc               0.43173    0.15312 74.77081   2.820  0.00615 **
## DepthFac25          -0.58466    0.13679 81.00000  -4.274 5.20e-05 ***
## DepthFac55          -1.05250    0.13679 81.00000  -7.694 3.00e-11 ***
## DepthFac100         -1.61424    0.13679 81.00000 -11.801  < 2e-16 ***
## Blockb:DepthFac25   -0.43965    0.19345 81.00000  -2.273  0.02570 *
## Blockc:DepthFac25   -0.80589    0.19345 81.00000  -4.166 7.71e-05 ***
## Blockb:DepthFac55   -0.35552    0.19345 81.00000  -1.838  0.06977 .
## Blockc:DepthFac55   -0.79753    0.19345 81.00000  -4.123 9.00e-05 ***
## Blockb:DepthFac100 -0.07874    0.19345 81.00000  -0.407  0.68506
## Blockc:DepthFac100 -0.48591    0.19345 81.00000  -2.512  0.01400 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation matrix not shown by default, as p = 13 > 12.
## Use print(x, correlation=TRUE)  or
##     vcov(x)        if you need it
```

And we see that the effect of EFB treatment is borderline significant at the 5% level. We can also do a likelihood ratio test (with both simulated and the $\chi^2(1)$ approximation) using `PBmodComp` of the same hypothesis:

```
M4_small <- lmer(log_SOC ~ Block*DepthFac + (1|Treat) + (1|Plant),
data=SOCdata)

PBmodcomp(M4, M4_small)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00207362 (tol = 0.002, component 1)
```

```
## Bootstrap test; time: 30.30 sec; samples: 1000; extremes: 36;
## large : log_SOC ~ Treat2 + Block * DepthFac + (1 | Treat) + (1 | Plant)
## log_SOC ~ Block * DepthFac + (1 | Treat) + (1 | Plant)
##          stat df p.value
## LRT    5.2767  1 0.02161 *
## PBtest 5.2767    0.03696 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that the p-value corresponding to the $\chi^2(1)$ is around the same level as with the t-test. The p-value from the simulated LRT statistics seems to agrees. All in all there is some evidence that the EFB treatment works, but it is not crystal clear.

7.

We estimate ρ the same as we did in question 2, namely

$$\hat{\rho}(Y_1, Y_2) = \frac{\hat{\sigma}_p^2}{\hat{\sigma}^2 + \hat{\sigma}_p^2}$$

We do this for all the simulated values from the posterior distribution, and compute a 95% credible interval by taking quantiles in this distribution. We compute posterior mean by taking the mean for the simulated values of ρ:

```
brmSim2 %>% tibble() %>%
  select(sd_Plant__Intercept,sigma) %>%
  mutate(rho = sd_Plant__Intercept^2/(sd_Plant__Intercept^2 + sigma^2) ) %>%
  select(rho) %>%
  summarise(post_mean = mean(rho), lower_q = quantile(rho, 0.025), upper_rho = quantile(rho, 0.975))
```

```
## # A tibble: 1 x 3
##   post_mean lower_q upper_rho
##       <dbl>   <dbl>     <dbl>
## 1     0.205 0.00528     0.472
```

Hence the posterior mean for ρ is 0.205, and a 95% credible interval for ρ is $[0.00528, 0.472]$.

Exercise 3

We write the model to a .stan file

```
library(rstan)
```

```
## Warning: package 'rstan' was built under R version 4.3.2
```

```
## Loading required package: StanHeaders
```

```
## Warning: package 'StanHeaders' was built under R version 4.3.2
```

```
##
## rstan version 2.32.3 (Stan version 2.26.1)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
```

```
## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```
to_stan <- "data {
int N;
vector[N] y;
}
parameters {
real<lower = 0, upper = 1> theta;
}
model {
y ~ normal(0, sqrt(theta));
}
"
write(to_stan, file = "model.stan")
```

Stan per default uses the uniform prior, and since we have restricted θ to be in the interval between 0 and 1, this really does corresponds to the model setup. We simulate from the posterior of θ using 4 chains, 6000 iterations and using half for burn-in, with our given data that is

```
y <- c(0.556,0.225,0.154,0.644,0.245,0.520,1.033,-0.067,0.367,-0.743)
n <- length(y)
data = list(N = n , y = y)
stan_fit <- stan("model.stan", iter = 6000, data = data, verbose = FALSE, seed = 1)
```

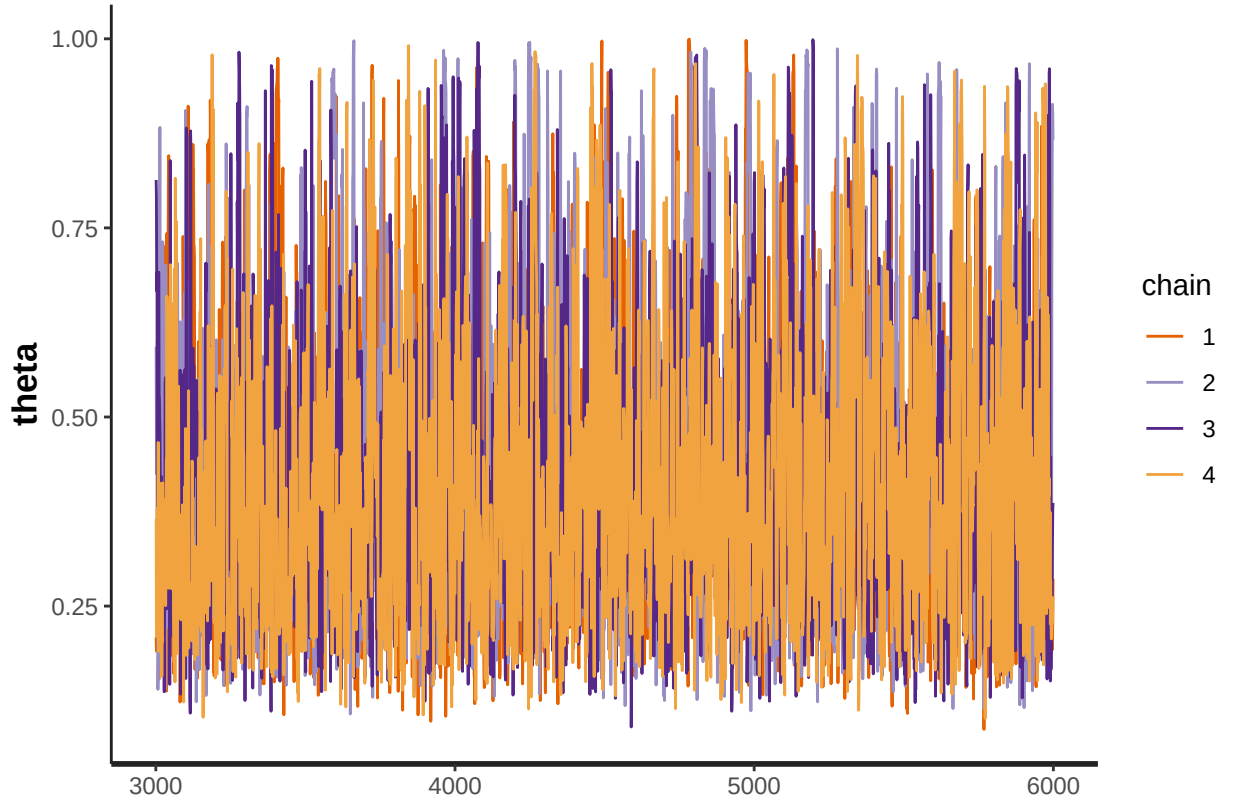
```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.3 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 6000 [  0%]  (Warmup)
## Chain 1: Iteration:  600 / 6000 [ 10%]  (Warmup)
## Chain 1: Iteration: 1200 / 6000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1800 / 6000 [ 30%]  (Warmup)
## Chain 1: Iteration: 2400 / 6000 [ 40%]  (Warmup)
## Chain 1: Iteration: 3000 / 6000 [ 50%]  (Warmup)
## Chain 1: Iteration: 3001 / 6000 [ 50%]  (Sampling)
## Chain 1: Iteration: 3600 / 6000 [ 60%]  (Sampling)
## Chain 1: Iteration: 4200 / 6000 [ 70%]  (Sampling)
## Chain 1: Iteration: 4800 / 6000 [ 80%]  (Sampling)
## Chain 1: Iteration: 5400 / 6000 [ 90%]  (Sampling)
## Chain 1: Iteration: 6000 / 6000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.045 seconds (Warm-up)
## Chain 1:                0.054 seconds (Sampling)
## Chain 1:                0.099 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
```

```
## Chain 2: Gradient evaluation took 7e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 6000 [  0%]  (Warmup)
## Chain 2: Iteration:  600 / 6000 [ 10%]  (Warmup)
## Chain 2: Iteration: 1200 / 6000 [ 20%]  (Warmup)
## Chain 2: Iteration: 1800 / 6000 [ 30%]  (Warmup)
## Chain 2: Iteration: 2400 / 6000 [ 40%]  (Warmup)
## Chain 2: Iteration: 3000 / 6000 [ 50%]  (Warmup)
## Chain 2: Iteration: 3001 / 6000 [ 50%]  (Sampling)
## Chain 2: Iteration: 3600 / 6000 [ 60%]  (Sampling)
## Chain 2: Iteration: 4200 / 6000 [ 70%]  (Sampling)
## Chain 2: Iteration: 4800 / 6000 [ 80%]  (Sampling)
## Chain 2: Iteration: 5400 / 6000 [ 90%]  (Sampling)
## Chain 2: Iteration: 6000 / 6000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.083 seconds (Warm-up)
## Chain 2:                0.08 seconds (Sampling)
## Chain 2:                0.163 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 6e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 6000 [  0%]  (Warmup)
## Chain 3: Iteration:  600 / 6000 [ 10%]  (Warmup)
## Chain 3: Iteration: 1200 / 6000 [ 20%]  (Warmup)
## Chain 3: Iteration: 1800 / 6000 [ 30%]  (Warmup)
## Chain 3: Iteration: 2400 / 6000 [ 40%]  (Warmup)
## Chain 3: Iteration: 3000 / 6000 [ 50%]  (Warmup)
## Chain 3: Iteration: 3001 / 6000 [ 50%]  (Sampling)
## Chain 3: Iteration: 3600 / 6000 [ 60%]  (Sampling)
## Chain 3: Iteration: 4200 / 6000 [ 70%]  (Sampling)
## Chain 3: Iteration: 4800 / 6000 [ 80%]  (Sampling)
## Chain 3: Iteration: 5400 / 6000 [ 90%]  (Sampling)
## Chain 3: Iteration: 6000 / 6000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.047 seconds (Warm-up)
## Chain 3:                0.05 seconds (Sampling)
## Chain 3:                0.097 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 7e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
```

```
## Chain 4:
## Chain 4: Iteration:    1 / 6000 [  0%]  (Warmup)
## Chain 4: Iteration:  600 / 6000 [ 10%]  (Warmup)
## Chain 4: Iteration: 1200 / 6000 [ 20%]  (Warmup)
## Chain 4: Iteration: 1800 / 6000 [ 30%]  (Warmup)
## Chain 4: Iteration: 2400 / 6000 [ 40%]  (Warmup)
## Chain 4: Iteration: 3000 / 6000 [ 50%]  (Warmup)
## Chain 4: Iteration: 3001 / 6000 [ 50%]  (Sampling)
## Chain 4: Iteration: 3600 / 6000 [ 60%]  (Sampling)
## Chain 4: Iteration: 4200 / 6000 [ 70%]  (Sampling)
## Chain 4: Iteration: 4800 / 6000 [ 80%]  (Sampling)
## Chain 4: Iteration: 5400 / 6000 [ 90%]  (Sampling)
## Chain 4: Iteration: 6000 / 6000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.065 seconds (Warm-up)
## Chain 4:                0.071 seconds (Sampling)
## Chain 4:                0.136 seconds (Total)
## Chain 4:
```

We show traceplots for θ below,

```
traceplot(stan_fit)
```



The chains seem to be well-mixing. We do not see anything alarming - i.e. we do not see anything systematic trends within chains over time and we do not see differences between chains. We print a summary of the model

```
stan_fit
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=6000; warmup=3000; thin=1;
## post-warmup draws per chain=3000, total post-warmup draws=12000.
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## theta  0.42    0.00 0.19  0.16  0.27  0.37  0.53  0.89  2178    1
## lp__  -0.98    0.02 0.88 -3.61 -1.18 -0.63 -0.40 -0.34  2132    1
##
## Samples were drawn using NUTS(diag_e) at Wed Jan 24 13:00:13 2024.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

We see that $\hat{R}$ for θ is 1, indicating that the chains have converged to their stationary distribution.

2.

We simulate from the posterior predictive distribution of $\tilde{Y}$ by using our simulated values of θ from θ's posterior distribution to simulate from the conditional $Y|\theta = \theta' \sim \mathcal{N}(0, \theta')$.

```
set.seed(1)
theta <- extract(stan_fit)$theta
y_post <- rnorm(length(theta) , 0, theta^2)
```

Say that researchers observe $y = 1.4$. The probability of observing something as extreme as 1.4 in the posterior predictive distribution is

```
mean(y_post > 1.4)
```

```
## [1] 0.00275
```

Which must be considered fairly extreme. This would indicate that the conditions for the experiment has indeed changed as the researchers are concerned about.