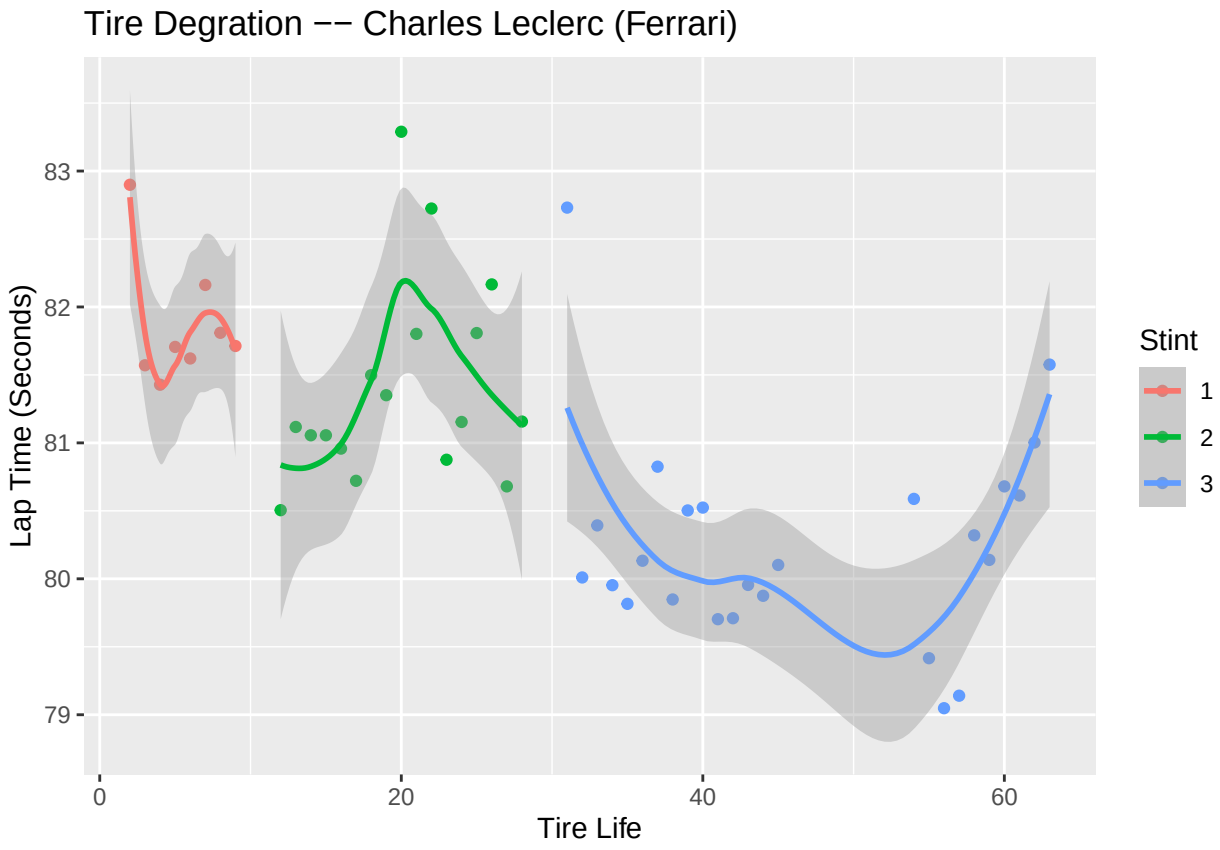
Full Race Simulation & Model for One Driver

Cole Cappello

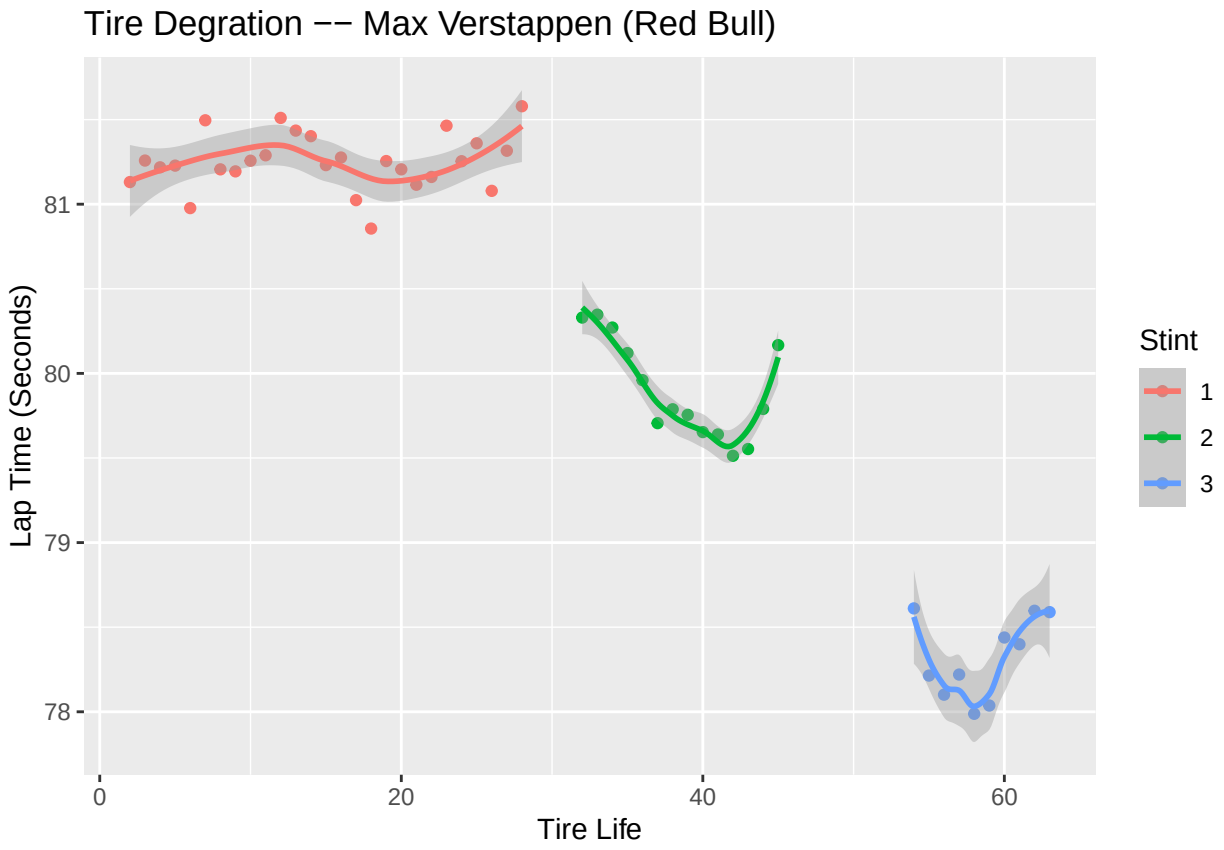
06-20-2025

```
canada_laps <- read_csv("imola_laps.csv")

canada_laps <- canada_laps %>%
    mutate(Stint = as.factor(Stint))

filter(canada_laps, Driver == "LEC", is.na(PitOutTime) & is.na(PitInTime),LapTime < 100) %>%
    ggplot(mapping = aes(x = LapNumber, y = LapTime, colour = Stint)) +
    geom_point() +
    geom_smooth() +
    labs(title = "Tire Degration -- Charles Leclerc (Ferrari)",
         x = "Tire Life",
         y = "Lap Time (Seconds)")
```

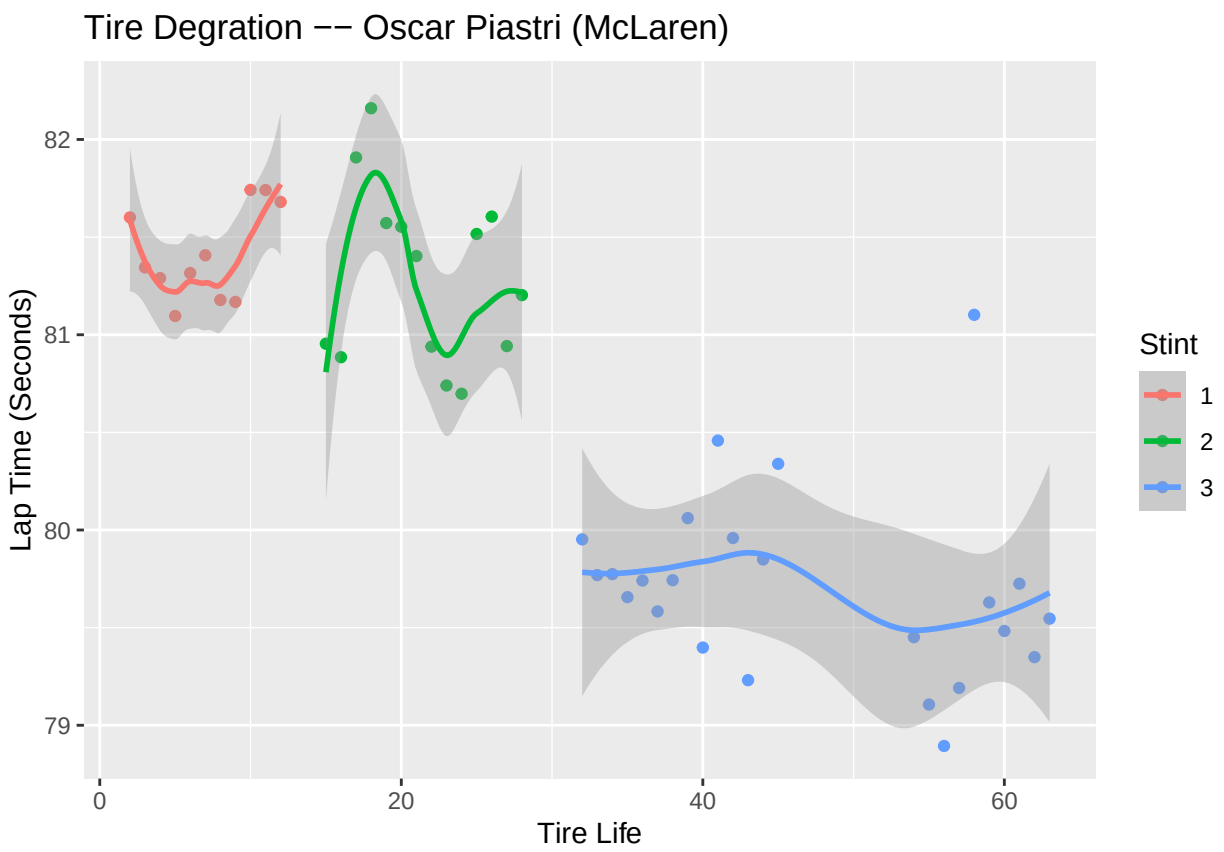



```
filter(canada_laps, Driver == "VER", is.na(PitOutTime) & is.na(PitInTime),LapTime < 90) %>%
    ggplot(mapping = aes(x = LapNumber, y = LapTime, colour = Stint)) +
    geom_point() +
```

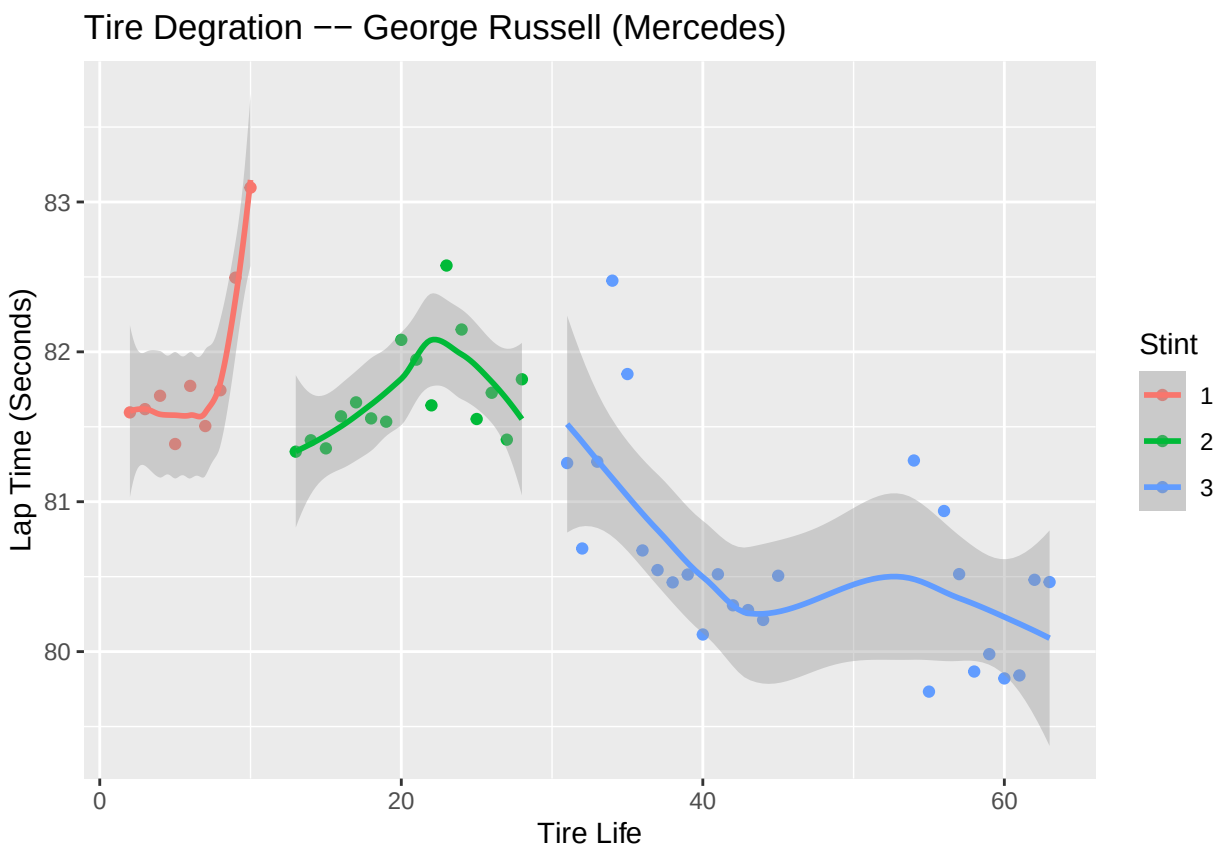
```
    geom_smooth() +
    labs(title = "Tire Degration -- Max Verstappen (Red Bull)",
         x = "Tire Life",
         y = "Lap Time (Seconds)")
```



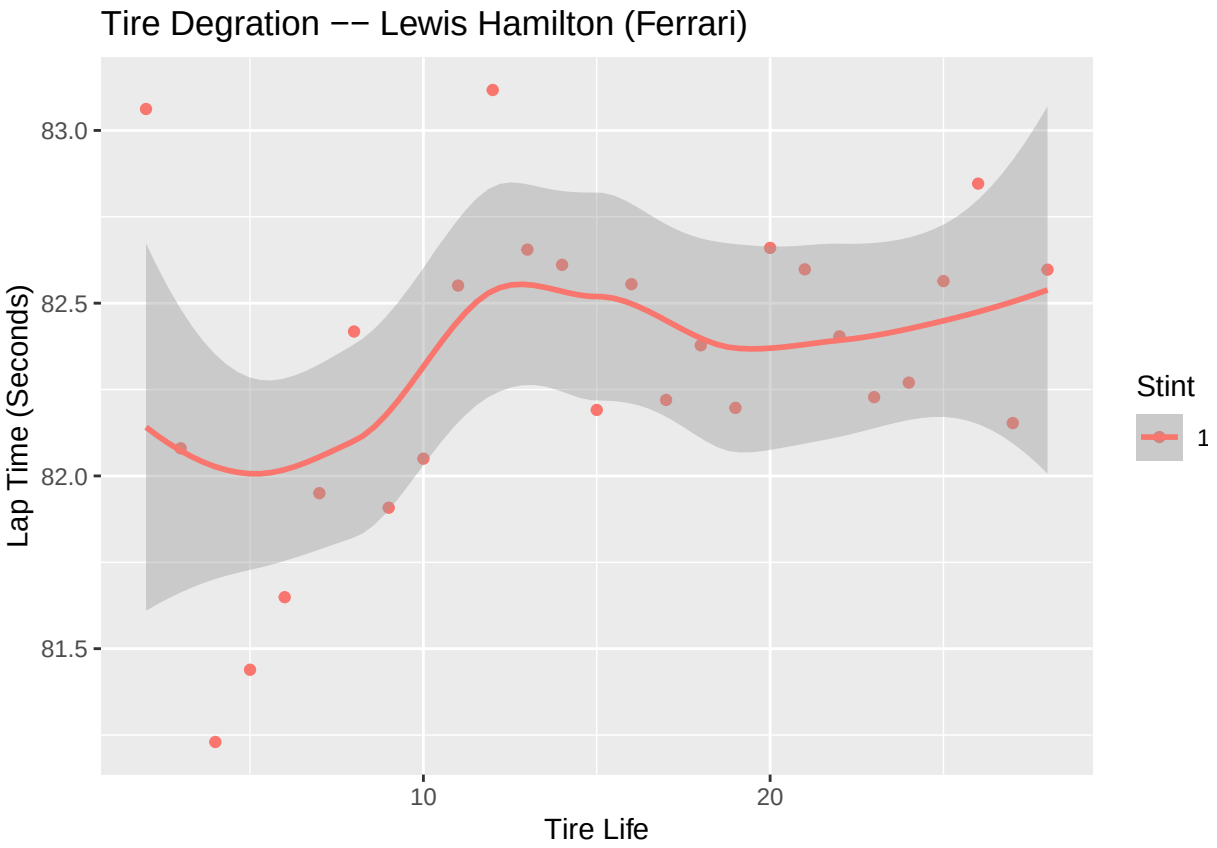

```
filter(canada_laps, Driver == "PIA", is.na(PitOutTime) & is.na(PitInTime),LapTime < 100) %>%
    ggplot(mapping = aes(x = LapNumber, y = LapTime, colour = Stint)) +
    geom_point() +
    geom_smooth() +
    labs(title = "Tire Degration -- Oscar Piastri (McLaren)",
         x = "Tire Life",
         y = "Lap Time (Seconds)")
```




```
filter(canada_laps, Driver == "RUS", is.na(PitOutTime) & is.na(PitInTime),LapTime < 100) %>%
    ggplot(mapping = aes(x = LapNumber, y = LapTime, colour = Stint)) +
    geom_point() +
    geom_smooth() +
    labs(title = "Tire Degration -- George Russell (Mercedes)",
         x = "Tire Life",
         y = "Lap Time (Seconds)")
```




```
filter(canada_laps, Driver == "HAM", is.na(PitOutTime) & is.na(PitInTime),LapTime < 100, Stint == 1) %>%
    ggplot(mapping = aes(x = LapNumber, y = LapTime, colour = Stint)) +
    geom_point() +
    geom_smooth() +
    labs(title = "Tire Degration -- Lewis Hamilton (Ferrari)",
         x = "Tire Life",
         y = "Lap Time (Seconds)")
```



In the next model we'll try implementing the degradation model for a whole race. The main difference that we'll notice is that in every one of the plots above, we see a decrease in the average lap time with each stint. This is evidence of the cars getting faster as fuel load decreases. The next step after this will be to try to find ways to estimate the decrease in laptime across stints due to fuel usage.

Linearly Increasing Slope for the full Race

The idea behind extending this model to a full race for a single drive is to allow intervention effects to account for pit stops for new tires. Specifically, we'll try to do this by creating a reset vector for the latent states that depends on compound, and a reset value for ν. The degradation rate β now depends on the tire compound being used as well.

$$y_t = \alpha_t + \epsilon_t \tag{1}$$

$$\alpha_{t+1} = \begin{cases} \alpha_t + \nu_t + \omega_t & pit_t = 0 \\ \alpha.reset[compound_t] & pit = 1 \end{cases} \tag{2}$$

$$\nu_{t+1} = \begin{cases} \nu_t + \beta[compound_t] & pit_t = 0 \\ \nu.reset & pit = 1 \end{cases} \tag{3}$$

where $\epsilon_t \sim N(0, .4)$, $\beta[Hard] = .01$,$\beta[Medium] = .03$, $\beta[Soft] = .08$, $\nu_1 = \nu.reset = 0$, $\omega_t \sim N(0, .1)$, and $\alpha_1 = \alpha.reset[compound_{t=1}]$. We'll let $\alpha.reset$ be 96, 95, and 94.5 for hard, medium, and soft tires resepectively. In this simulation, we will start on hard tires, pit on lap 30 for medium tires, then pit on lap 50 for soft tires.

```r
# Number of states/laps
N <- 60

# Tire Compound index 1 - Hard, 2 - Medium, 3 - Soft
compound <- c(rep(1,times = 30),rep(2, times = 20), rep(3, times = 10))

# Pit Index indicates with a 1 if a new set of tires is put on the next lap
pit <- rep(0, N)
pit[30] <- 1
pit[50] <- 1

# Beta is now a vector with the index representing degredation increase for each compound
beta <- rep(0,3)
beta[1] <- .01
beta[2] <- .03
beta[3] <- .08

# Reset vector stores the true pace of a new tire compound
reset.alpha <- rep(0,3)
reset.alpha[1] <- 96
reset.alpha[2] <- 95
reset.alpha[3] <- 94.5

# Reset value for nu
reset.nu <- 0

# Initialize nu vector
nu <- rep(NA, N)
nu[1] <- reset.nu

# Initialize alpha vector
alpha <- rep(NA, N)
alpha[1] <- reset.alpha[1]

# Initialize process errors
omega <- rnorm(n = N, mean = 0, sd = .1)

# Store true latent states
for(i in 2:(length(alpha))) {
    if(pit[i-1] == 1) {
        alpha[i] <- reset.alpha[compound[i]] + omega[i]
        nu[i] <- reset.nu
    } else {
    alpha[i] <- alpha[i-1] + nu[i-1] + omega[i]
    nu[i] <- nu[i-1] + beta[compound[i-1]]
    }
}

# Vector of observation errors
epsilon <- rnorm(n = N, mean = 0, sd = .4)

# Simulated observations
y <- alpha + epsilon
```

The full_racce_1driver.stan file fits the above model.

```
data_stan <- list(TT = length(y), y = y, C = 3, Compound = compound, Pit = pit, z_reset0 = c(96,95,94.5)

stan_test <- stan(file = "full_race_1driver.stan",
                  data = data_stan,
                  chains = 3, iter = 30000,
                  control = list(adapt_delta = .99))
```

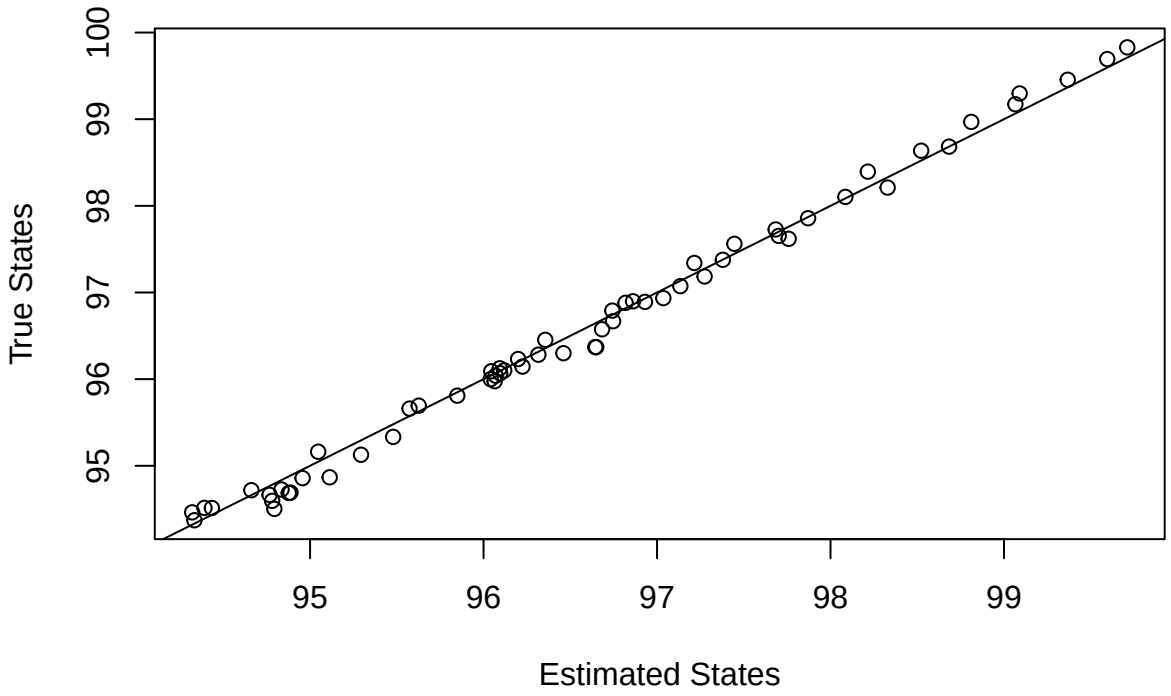
```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: 'Apple clang version 17.0.0 (clang-1700.0.13.3)'
## using SDK: 'MacOSX15.4.sdk'
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Framew
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/StanHeader
## In file included from /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/RcppEigen/
## In file included from /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/RcppEigen/
## /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/RcppEigen/include/Eigen/src/Core
##   679 | #include <cmath>
##       |          ^~~~~~~
## 1 error generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 1:
## Chain 1: Gradient evaluation took 7.7e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.77 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:     1 / 30000 [  0%]  (Warmup)
## Chain 2:
## Chain 2: Gradient evaluation took 1.8e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.18 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:     1 / 30000 [  0%]  (Warmup)
## Chain 3:
## Chain 3: Gradient evaluation took 1.7e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.17 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:     1 / 30000 [  0%]  (Warmup)
## Chain 3: Iteration:  3000 / 30000 [ 10%]  (Warmup)
## Chain 2: Iteration:  3000 / 30000 [ 10%]  (Warmup)
## Chain 1: Iteration:  3000 / 30000 [ 10%]  (Warmup)
## Chain 3: Iteration:  6000 / 30000 [ 20%]  (Warmup)
## Chain 2: Iteration:  6000 / 30000 [ 20%]  (Warmup)
## Chain 1: Iteration:  6000 / 30000 [ 20%]  (Warmup)
```

```
## Chain 3: Iteration:  9000 / 30000 [ 30%]  (Warmup)
## Chain 2: Iteration:  9000 / 30000 [ 30%]  (Warmup)
## Chain 1: Iteration:  9000 / 30000 [ 30%]  (Warmup)
## Chain 3: Iteration: 12000 / 30000 [ 40%]  (Warmup)
## Chain 2: Iteration: 12000 / 30000 [ 40%]  (Warmup)
## Chain 1: Iteration: 12000 / 30000 [ 40%]  (Warmup)
## Chain 2: Iteration: 15000 / 30000 [ 50%]  (Warmup)
## Chain 2: Iteration: 15001 / 30000 [ 50%]  (Sampling)
## Chain 3: Iteration: 15000 / 30000 [ 50%]  (Warmup)
## Chain 3: Iteration: 15001 / 30000 [ 50%]  (Sampling)
## Chain 1: Iteration: 15000 / 30000 [ 50%]  (Warmup)
## Chain 1: Iteration: 15001 / 30000 [ 50%]  (Sampling)
## Chain 2: Iteration: 18000 / 30000 [ 60%]  (Sampling)
## Chain 3: Iteration: 18000 / 30000 [ 60%]  (Sampling)
## Chain 1: Iteration: 18000 / 30000 [ 60%]  (Sampling)
## Chain 2: Iteration: 21000 / 30000 [ 70%]  (Sampling)
## Chain 1: Iteration: 21000 / 30000 [ 70%]  (Sampling)
## Chain 3: Iteration: 21000 / 30000 [ 70%]  (Sampling)
## Chain 2: Iteration: 24000 / 30000 [ 80%]  (Sampling)
## Chain 1: Iteration: 24000 / 30000 [ 80%]  (Sampling)
## Chain 3: Iteration: 24000 / 30000 [ 80%]  (Sampling)
## Chain 1: Iteration: 27000 / 30000 [ 90%]  (Sampling)
## Chain 2: Iteration: 27000 / 30000 [ 90%]  (Sampling)
## Chain 3: Iteration: 27000 / 30000 [ 90%]  (Sampling)
## Chain 1: Iteration: 30000 / 30000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 9.891 seconds (Warm-up)
## Chain 1:                8.5 seconds (Sampling)
## Chain 1:                18.391 seconds (Total)
## Chain 1:
## Chain 2: Iteration: 30000 / 30000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 9.047 seconds (Warm-up)
## Chain 2:                9.625 seconds (Sampling)
## Chain 2:                18.672 seconds (Total)
## Chain 2:
## Chain 3: Iteration: 30000 / 30000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 9.21 seconds (Warm-up)
## Chain 3:                10.683 seconds (Sampling)
## Chain 3:                19.893 seconds (Total)
## Chain 3:
```

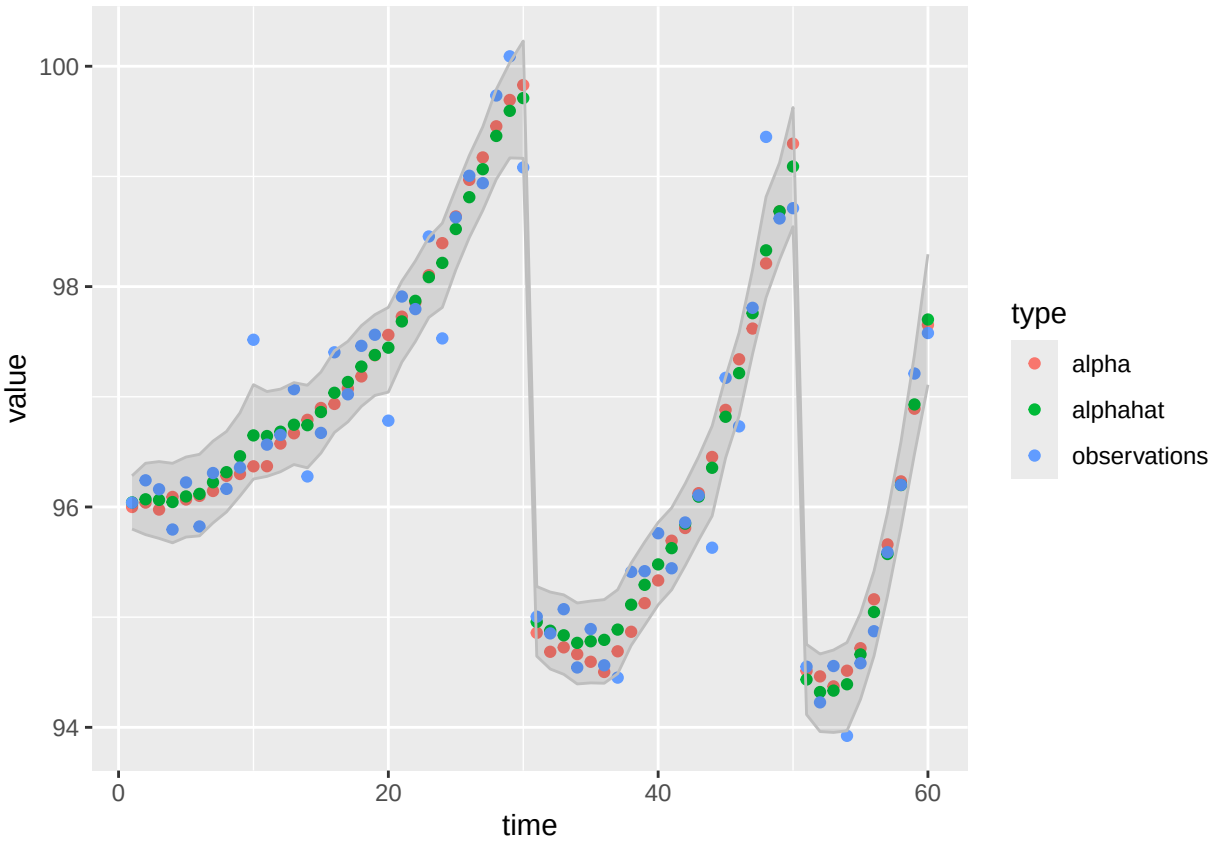
```
## Warning: There were 3 chains where the estimated Bayesian Fraction of Missing Information was low. Se
## https://mc-stan.org/misc/warnings.html#bfmi-low
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
states <- colMeans(extract(stan_test, pars = c("z"))[[1]])

plot(x = states, y = alpha,
     xlab = "Estimated States",
     ylab = "True States")
abline(0,1)
```




```
# Include a plot of time vs. estimated state, true state, and observations
time <- seq(from = 1, to = N, by = 1)

z_stan <- extract(stan_test, pars = c("z"))[[1]]
z_CIl <- apply(z_stan, 2, quantile, probs=0.025)
z_CIu <- apply(z_stan, 2, quantile, probs=0.975)

df1 <- tibble(time = time,
              alpha = alpha,
              alphahat = states,
              observations = y,
              z_CIl = z_CIl,
              z_CIu = z_CIu)


df_long <- df1 %>%
  pivot_longer(cols = c(alpha, alphahat,observations), names_to = "type", values_to = "value")

ggplot(df_long, aes(x = time, y = value, color = type)) +
    geom_point() +
    geom_ribbon(aes(ymin = z_CIl, ymax = z_CIu), fill = "black",color = "gray", alpha = .1)
```




```
# Estimated Observation Error:
mean(extract(stan_test, pars = c("sdo"))[[1]])
```

```
## [1] 0.392562
```

```
# Estimated Process Error
mean(extract(stan_test, pars = c("sdp"))[[1]])
```

```
## [1] 0.1809931
```

```
calc_rMSE(alpha,states)
```

```
## [1] 0.1220051
```

```
calc_rMSE(alpha, y)
```

```
## [1] 0.3936683
```

Posterior Predictive Checks

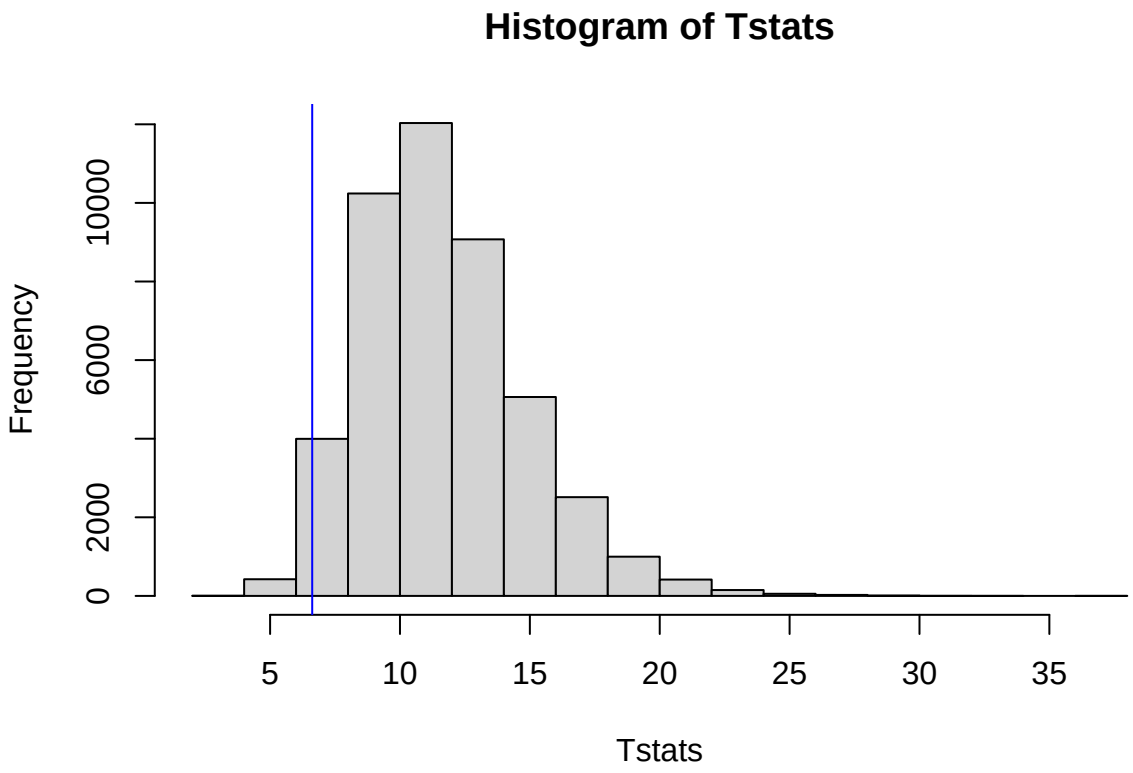
For the previous model, we will do a posterior predictive check with the test statistic:

$$T(\mathbf{y}, \theta) = \sum_{t=1}^{T} (y_t - \hat{y}_{t|1:T})^2$$

where $\hat{y}_{t|1:T}$) is the state estimate for the model at time t.

```
Tstat <- function(y) {
    return(sum((y - df1$alphahat)^2))
}
```

```
posterior <- extract(stan_test)
yrep <- posterior$y_rep

Tstats <- rep(NA,dim(yrep)[1])
for(i in 1:dim(yrep)[1]) {
    Tstats[i] <- Tstat(yrep[i,])
}

hist(Tstats)
abline(v = Tstat(df1$observations), col = 'blue')
```




```
Tobs <- Tstat(df1$observations)
count <- 0
for(i in 1:length(Tstats)) {
    if(Tstats[i] > Tobs) {
        count <- count + 1
    }
}

pval <- count/length(Tstats)
pval
```

```
## [1] 0.977
```

WAIC Calculation

```
library(loo)

log_pd <- posterior$log_pd
```

```
waic(log_pd)
```

```
## Warning:
## 8 (13.3%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

```
##
## Computed from 45000 by 60 log-likelihood matrix.
##
##           Estimate   SE
## elpd_waic    -36.9  6.8
## p_waic        14.8  3.1
## waic          73.9 13.7
##
## 8 (13.3%) p_waic estimates greater than 0.4. We recommend trying loo instead.
```

Overall the estimates look decent, but the posterior predictive check doesn't look great unfortunately. If I bump up the observation error it gets better.

In this previous example, the average lap time decreases across stints because we have set the reset time for softs to be less than mediums, and mediums less than hards. This makes sense based on knowledge of the tire compounds. However, in actual race data we see that the average laptime decreases across stints regardless of the compound used. This is due to fuel loss during the course of the race.

Lap time Decrease due to Fuel Loss

We'll create a fuel.kg vector which starts at 110 (this is the max amount of fuel allowed by regulation) and decreases linearly to 0 at the end of the race. We are going to use a similar model to the above but add a covariate to the observation equation for fuel. In order to estimate gamma correctly, it will be critical to set a proper prior for the initial latent $\alpha_1 = \alpha.reset[compound_{t=1}]$. Ultimately, it needs to be roughly 1-1.5 seconds less than the initial observation y_1, but in the stan code we'll set the prior to be centered on the true value of 95.

$$y_t = \alpha_t + \gamma * fuel.kg_t + \epsilon_t \tag{4}$$

$$\alpha_{t+1} = \begin{cases} \alpha_t + \nu_t + \omega_t & pit_t = 0 \\ \alpha.reset[compound_t] & pit = 1 \end{cases} \tag{5}$$

$$\nu_{t+1} = \begin{cases} \nu_t + \beta[compound_t] & pit_t = 0 \\ \nu.reset & pit = 1 \end{cases} \tag{6}$$

where $\epsilon_t \sim N(0, .2)$, $\beta[Hard] = .002$, $\beta[Medium] = .01$, $\beta[Soft] = .02$, $\nu_1 = \nu.reset = 0$, $\omega_t \sim N(0, .15)$, and $\alpha_1 = \alpha.reset[compound_{t=1}]$. We'll let $\alpha.reset$ be 95, 94.5, and 94 for hard, medium, and soft tires resepectively. In this simulation, we will start on hard tires, pit on lap 30 for soft tires, then pit on lap 40 for medium tires.

```
# Number of states/laps
N <- 60

# Tire Compound index 1 - Hard, 2 - Medium, 3 - Soft
compound <- c(rep(1,times = 30), rep(3, times = 10),rep(2, times = 20))

# Fuel Mass in kg
fuel.kg <- seq(from = 110, to = 0, length.out = N)
```

```r
# Gamma
gamma <- .01


# Pit Index indicates with a 1 if a new set of tires is put on the next lap
pit <- rep(0, N)
pit[30] <- 1
pit[40] <- 1

# Beta is now a vector with the index representing each compound
beta <- rep(0,3)
beta[1] <- .002
beta[2] <- .01
beta[3] <- .02

# Reset vector stores the true pace of a new tire for each compound
reset.alpha <- rep(0,3)
reset.alpha[1] <- 95
reset.alpha[2] <- 94.5
reset.alpha[3] <- 94

# Reset value for nu
reset.nu <- 0

# Initialize nu vector
nu <- rep(NA, N)
nu[1] <- .01

# Initialize alpha vector
alpha <- rep(NA, N)
alpha[1] <- reset.alpha[1]

# Initialize process errors
omega <- rnorm(n = N, mean = 0, sd = .1)

# Store true latent states
for(i in 2:(length(alpha))) {
    if(pit[i-1] == 1) {
        alpha[i] <- reset.alpha[compound[i]] + omega[i]
        nu[i] <- reset.nu
    } else {
    alpha[i] <- alpha[i-1] + nu[i-1] + omega[i]
    nu[i] <- nu[i-1] + beta[compound[i-1]]
    }
}

# Vector of observation errors
epsilon <- rnorm(n = N, mean = 0, sd = .2)

# Simulated observations
y <- alpha + fuel.kg*gamma + epsilon
```

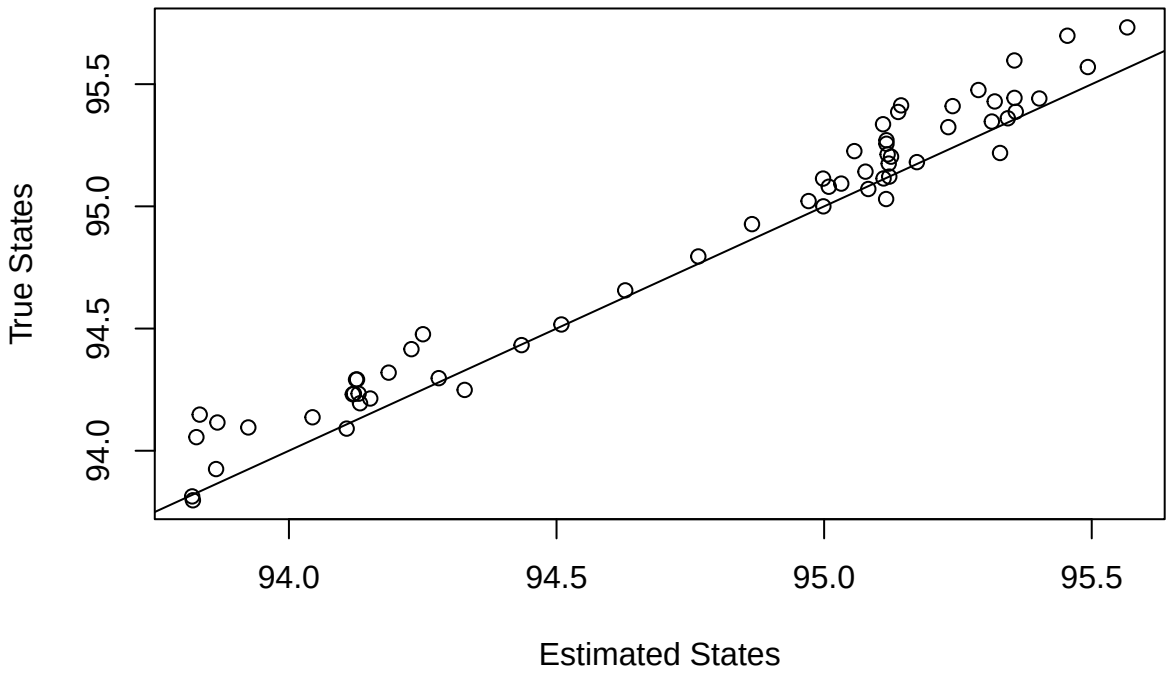
full_race_1driver_and_fuel.stan contains code to fit the model

```
data_stan <- list(TT = length(y), y = y, C = 3, Compound = compound, Pit = pit, z_reset0 = c(95,94.5,94

stan_test <- stan(file = "full_race_1driver_and_fuel.stan",
                  data = data_stan,
                  chains = 3, iter = 30000,
                  control = list(adapt_delta = .99,max_treedepth = 12))
```

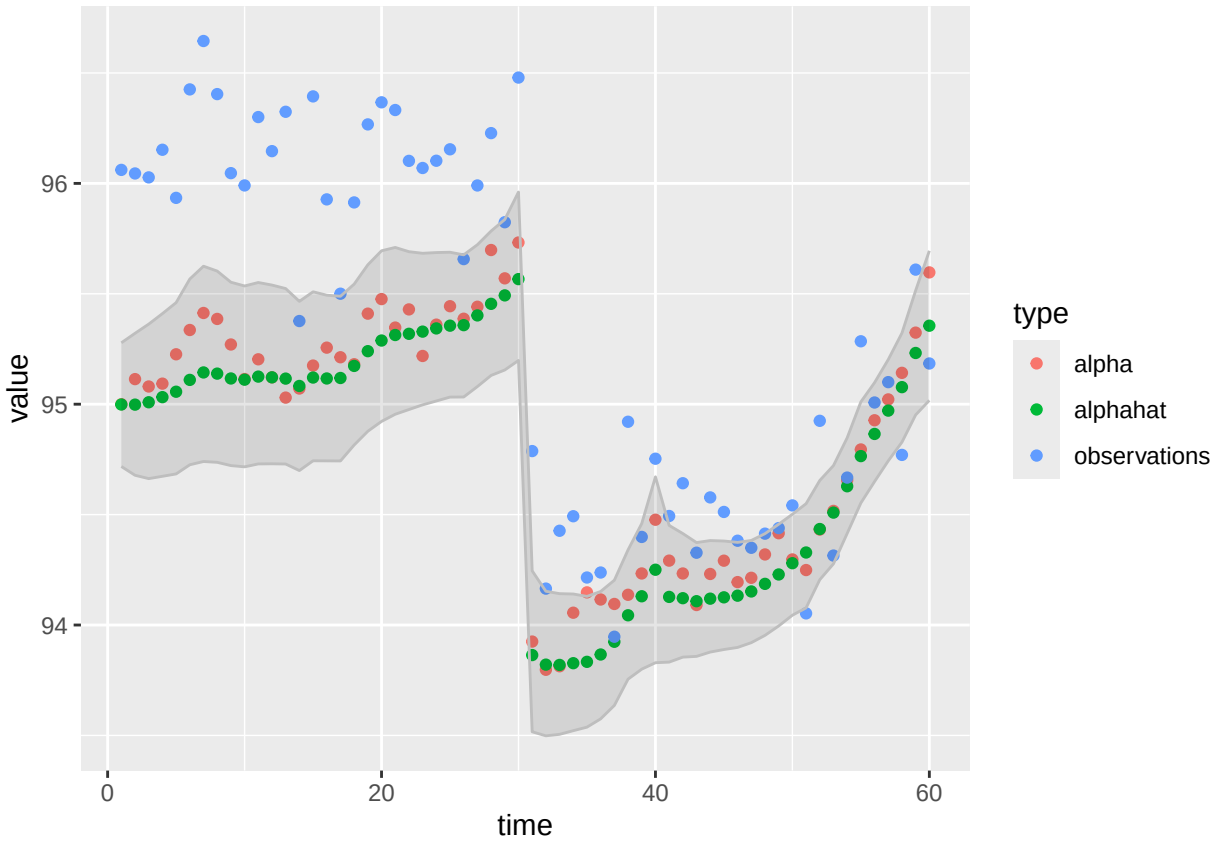
```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## using C compiler: 'Apple clang version 17.0.0 (clang-1700.0.13.3)'
## using SDK: 'MacOSX15.4.sdk'
## clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Framew
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/StanHeade
## In file included from /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/RcppEigen/
## In file included from /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/RcppEigen/
## /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/library/RcppEigen/include/Eigen/src/Core
##   679 | #include <cmath>
##       |          ^~~~~~
## 1 error generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.66 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:     1 / 30000 [  0%]  (Warmup)
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.9e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.19 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:     1 / 30000 [  0%]  (Warmup)
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 2.9e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.29 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:     1 / 30000 [  0%]  (Warmup)
## Chain 1: Iteration:  3000 / 30000 [ 10%]  (Warmup)
## Chain 1: Iteration:  6000 / 30000 [ 20%]  (Warmup)
## Chain 3: Iteration:  3000 / 30000 [ 10%]  (Warmup)
## Chain 2: Iteration:  3000 / 30000 [ 10%]  (Warmup)
## Chain 3: Iteration:  6000 / 30000 [ 20%]  (Warmup)
## Chain 2: Iteration:  6000 / 30000 [ 20%]  (Warmup)
## Chain 1: Iteration:  9000 / 30000 [ 30%]  (Warmup)
```

```
## Chain 3: Iteration:  9000 / 30000 [ 30%]  (Warmup)
## Chain 3: Iteration: 12000 / 30000 [ 40%]  (Warmup)
## Chain 2: Iteration:  9000 / 30000 [ 30%]  (Warmup)
## Chain 1: Iteration: 12000 / 30000 [ 40%]  (Warmup)
## Chain 3: Iteration: 15000 / 30000 [ 50%]  (Warmup)
## Chain 3: Iteration: 15001 / 30000 [ 50%]  (Sampling)
## Chain 3: Iteration: 18000 / 30000 [ 60%]  (Sampling)
## Chain 2: Iteration: 12000 / 30000 [ 40%]  (Warmup)
## Chain 1: Iteration: 15000 / 30000 [ 50%]  (Warmup)
## Chain 1: Iteration: 15001 / 30000 [ 50%]  (Sampling)
## Chain 3: Iteration: 21000 / 30000 [ 70%]  (Sampling)
## Chain 3: Iteration: 24000 / 30000 [ 80%]  (Sampling)
## Chain 1: Iteration: 18000 / 30000 [ 60%]  (Sampling)
## Chain 3: Iteration: 27000 / 30000 [ 90%]  (Sampling)
## Chain 2: Iteration: 15000 / 30000 [ 50%]  (Warmup)
## Chain 2: Iteration: 15001 / 30000 [ 50%]  (Sampling)
## Chain 1: Iteration: 21000 / 30000 [ 70%]  (Sampling)
## Chain 3: Iteration: 30000 / 30000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 42.809 seconds (Warm-up)
## Chain 3:                27.278 seconds (Sampling)
## Chain 3:                70.087 seconds (Total)
## Chain 3:
## Chain 2: Iteration: 18000 / 30000 [ 60%]  (Sampling)
## Chain 1: Iteration: 24000 / 30000 [ 80%]  (Sampling)
## Chain 2: Iteration: 21000 / 30000 [ 70%]  (Sampling)
## Chain 1: Iteration: 27000 / 30000 [ 90%]  (Sampling)
## Chain 2: Iteration: 24000 / 30000 [ 80%]  (Sampling)
## Chain 2: Iteration: 27000 / 30000 [ 90%]  (Sampling)
## Chain 1: Iteration: 30000 / 30000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 52.666 seconds (Warm-up)
## Chain 1:                40.275 seconds (Sampling)
## Chain 1:                92.941 seconds (Total)
## Chain 1:
## Chain 2: Iteration: 30000 / 30000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 66.815 seconds (Warm-up)
## Chain 2:                30.427 seconds (Sampling)
## Chain 2:                97.242 seconds (Total)
## Chain 2:

## Warning: There were 14 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: There were 3 chains where the estimated Bayesian Fraction of Missing Information was low. S
## https://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess
```

```r
states <- colMeans(extract(stan_test, pars = c("z"))[[1]])

plot(x = states, y = alpha,
     xlab = "Estimated States",
     ylab = "True States")
abline(0,1)
```




```r
# Include a plot of time vs. estimated state, true state, and observations
time <- seq(from = 1, to = N, by = 1)

z_stan <- extract(stan_test, pars = c("z"))[[1]]
z_CIl <- apply(z_stan, 2, quantile, probs=0.025)
z_CIu <- apply(z_stan, 2, quantile, probs=0.975)

df1 <- tibble(time = time,
              alpha = alpha,
              alphahat = states,
              observations = y,
              z_CIl = z_CIl,
              z_CIu = z_CIu)


df_long <- df1 %>%
  pivot_longer(cols = c(alpha, alphahat,observations), names_to = "type", values_to = "value")

ggplot(df_long, aes(x = time, y = value, color = type)) +
    geom_point() +
    geom_ribbon(aes(ymin = z_CIl, ymax = z_CIu), fill = "black",color = "gray", alpha = .1)
```




```
# Estimated Observation Error:
mean(extract(stan_test, pars = c("sdo"))[[1]])
```

```
## [1] 0.2692449
```

```
# Estimated Process Error
mean(extract(stan_test, pars = c("sdp"))[[1]])
```

```
## [1] 0.07541048
```

```
calc_rMSE(alpha,states)
```

```
## [1] 0.1316958
```

```
calc_rMSE(alpha, y)
```

```
## [1] 0.6575127
```

This looks pretty good! It seems similar to the pattern observed in the laptime data and we are able to infer back to the true states pretty well even when the observation error is close to the process error.

One of the things I like about this as well is that the laptime observations can look somewhat random, or like there isn't much degradation. Take the first stint for example, the observation times remain relatively stable. However, the underlying degradation process still continues and we are able to infer back to it pretty well.

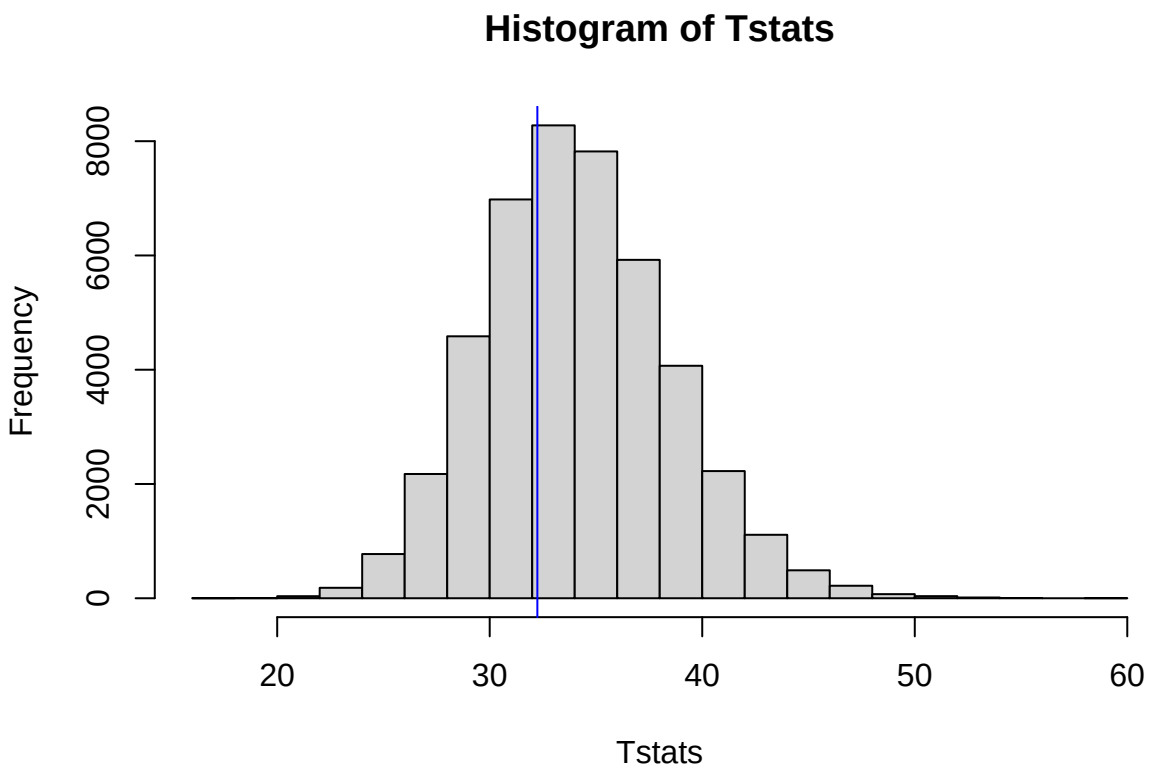
Posterior Predictive Checks

eFor the previous model, we will do a posterior predictive check with the test statistic:

$$T(\mathbf{y}, \theta) = \sum_{t=1}^{T} (y_t - \hat{y}_{t|1:T})^2$$

where $\hat{y}_{t|1:T}$) is the state estimate for the model at time t.

```
Tstat <- function(y) {
    return(sum((y - df1$alphahat)^2))
}

posterior <- extract(stan_test)
yrep <- posterior$y_rep

Tstats <- rep(NA,dim(yrep)[1])
for(i in 1:dim(yrep)[1]) {
    Tstats[i] <- Tstat(yrep[i,])
}

hist(Tstats)
abline(v = Tstat(df1$observations), col = 'blue')
```




```
Tobs <- Tstat(df1$observations)
count <- 0
for(i in 1:length(Tstats)) {
    if(Tstats[i] > Tobs) {
        count <- count + 1
    }
}

pval <- count/length(Tstats)
```

```
pval
```

```
## [1] 0.6509778
```

Here the posterior predictive checks are pretty good too.