Two Parameter Smoothing Test

Zane Billings

4/21/2020

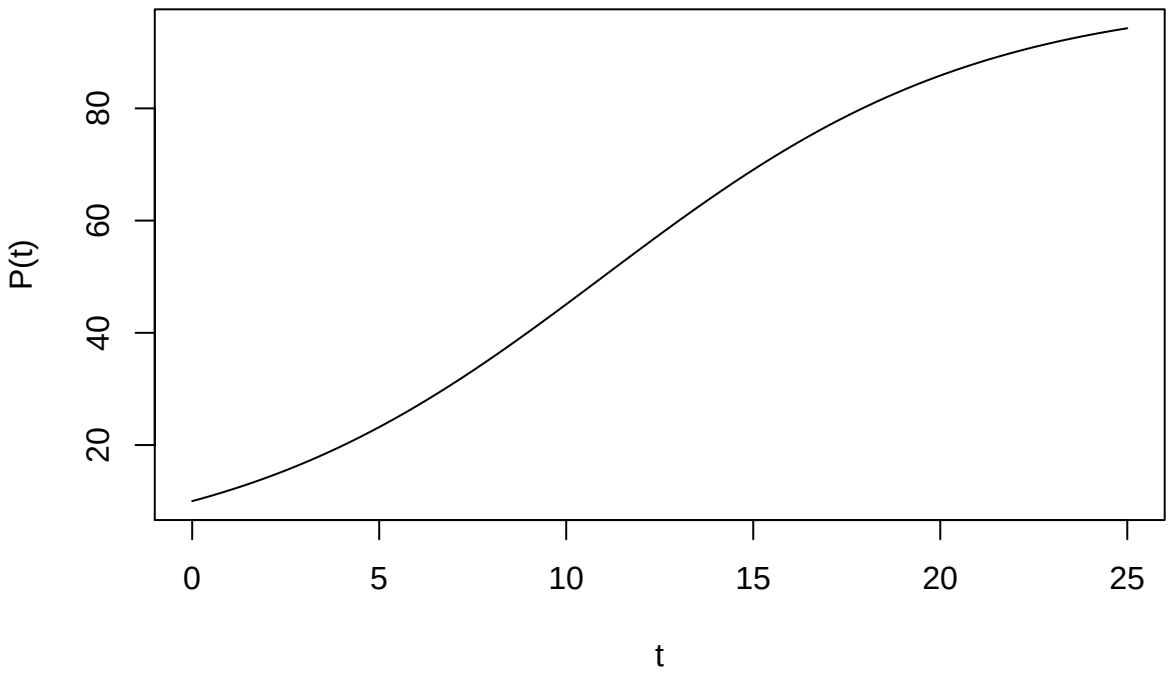
Objectives for this week:

- Test how smoothing works in the two parameter model.
- Basically, I will repeat the stuff from the dimensionless model smoothing test but I will specify a $K > 1$.
- I think we will not be able to recover the value of the carrying capacity, because before we saw that it was typically underestimated.

Generate Data

Using the fancy new `HMBGR` package available from GitHub (`devtools::install_github("wz-billings/HMBGR")`), I will generate the data with $P_0 = 10; K = 100; r = 0.2; t_{\max} = 25, \Delta t = 0.1$.

```
test <- HMBGR::generate_logistic_data(10, 100, 0.2, 25, 0.1, make_plot = TRUE)
```



First I'll try CVE with lambda is zero. This is also a bug test for the HMBGR cross validation method.

```
cve <- HMBGR::cross_validate(test, k = 3, reduced = FALSE, lambda = 0)
cve
```

```
## [1] 73367.41
```

Just repeating this a few times by hand, I got a lot of variation in the result, so I will implement Stein et al's recommendation of repeating the 3-fold CVE process 10 times.

```
cves <- replicate(10,
                  HMBGR::cross_validate(test,
                                        k = 3,
                                        reduced = FALSE,
                                        lambda = 0),
                  simplify = TRUE)
mean_cve <- mean(cves); mean_cve
```

```
## [1] 75166.25
```

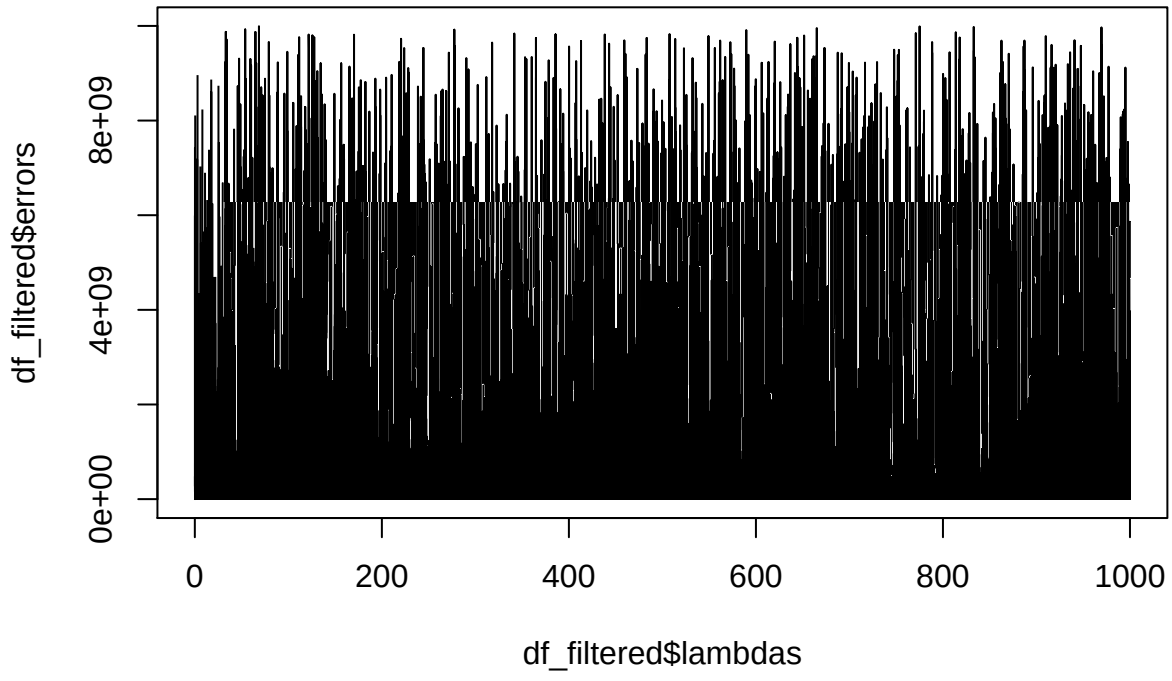
That is a pretty high average CVE. But remember we have not applied smoothing yet, so maybe that will help. Let's make a list of smoothing parameters and do this a bunch of times for different values, and see what our best result is.

```
lambdas <- seq(0.01, 1000, 0.01)
errors <- numeric(length(lambdas))
for (i in 1:length(lambdas)) {
  cves <- replicate(10,
                  HMBGR::cross_validate(test,
                                        k = 3,
                                        reduced = FALSE,
                                        lambda = lambdas[[i]]),
                  simplify = TRUE)
  errors[i] <- mean(cves)
}
```

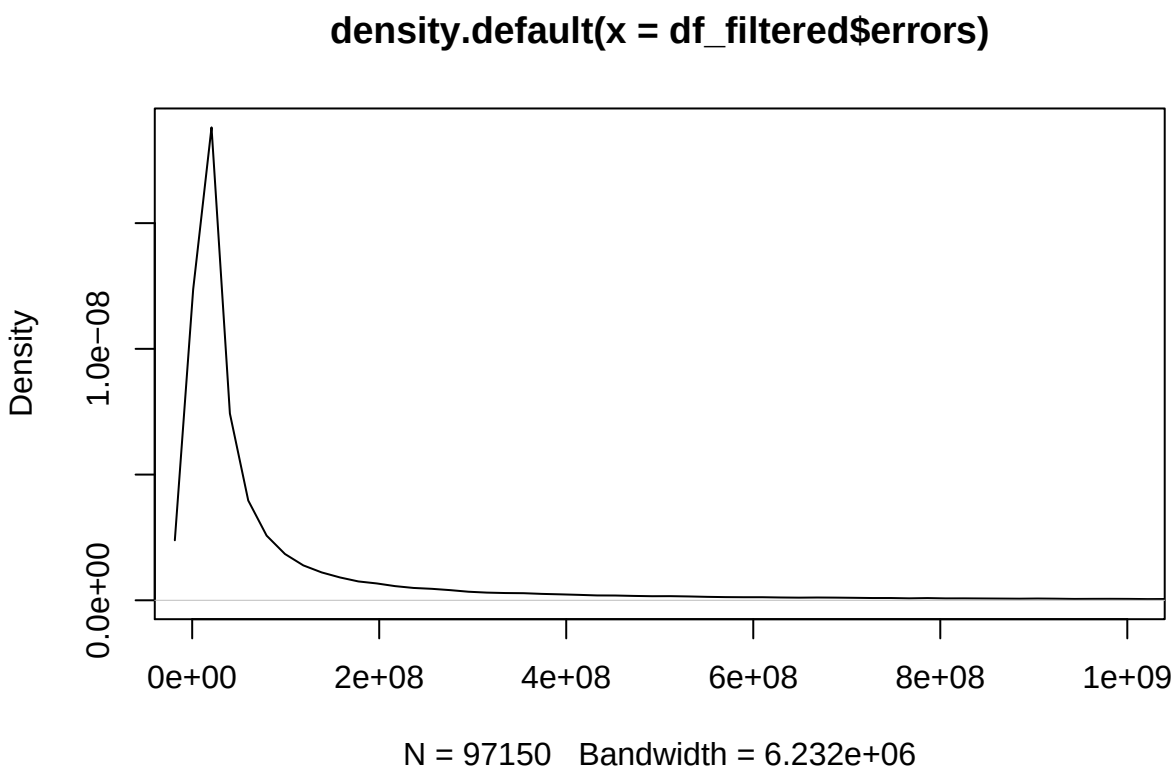
Now we can plot the results.

```
df <- data.frame(lambdas, errors)
df_filtered <- subset(df, df$errors < 10^10)

plot(df_filtered$lambdas, df_filtered$errors, type = "l")
```



```r
plot(density(df_filtered$errors), xlim = c(0, 10^9))
```



Ok so this is hard to visualize just due to the randomness, but the smallest error value I got was 1.3326204×10^5, which corresponds to a smoothing value of 0.01.

So, now I want to try normalizing the data I have and applying the one-parameter model. First, I will estimate the derivative from the data numerically.

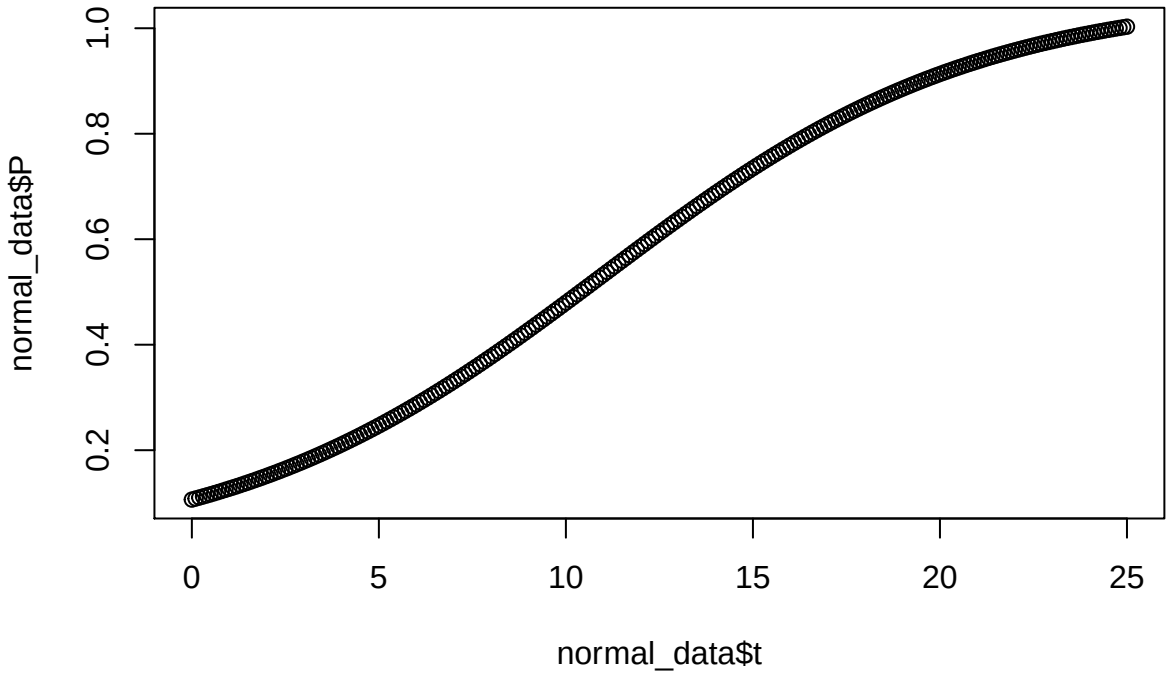
```r
# Using symmetrical quotient (centered difference) method, estimate the middle
middle <- test$P[2:1-length(test$P)]
n <- length(middle)
h <- max(test$t)/length(test$t)
mid_deriv <- (middle[2:n] - middle[1:n-1])/(2 * h)
lower_end <- (middle[2] - middle[1])/h
upper_end <- (middle[n] - middle[n-1])/h
derivative <- append(lower_end, mid_deriv, upper_end)
```

I thought this would be more useful–when the derivative is close to zero, we should be close to the steady state. But I might need to fine-tune my estimations, especially the endpoints. However, this definitely shows us that we are approaching a steady-state towards the end.

So, now I will use the highest 5 (not sure what the best number is, but this is probably fine) values to estimate the carrying capacity, and then normalize the data by the carrying capacity.

```r
n <- nrow(test)
khat <- mean(test$P[(n - 5):n])

normal_data <- test
normal_data$P <- normal_data$P / khat
plot(normal_data$t, normal_data$P)
```

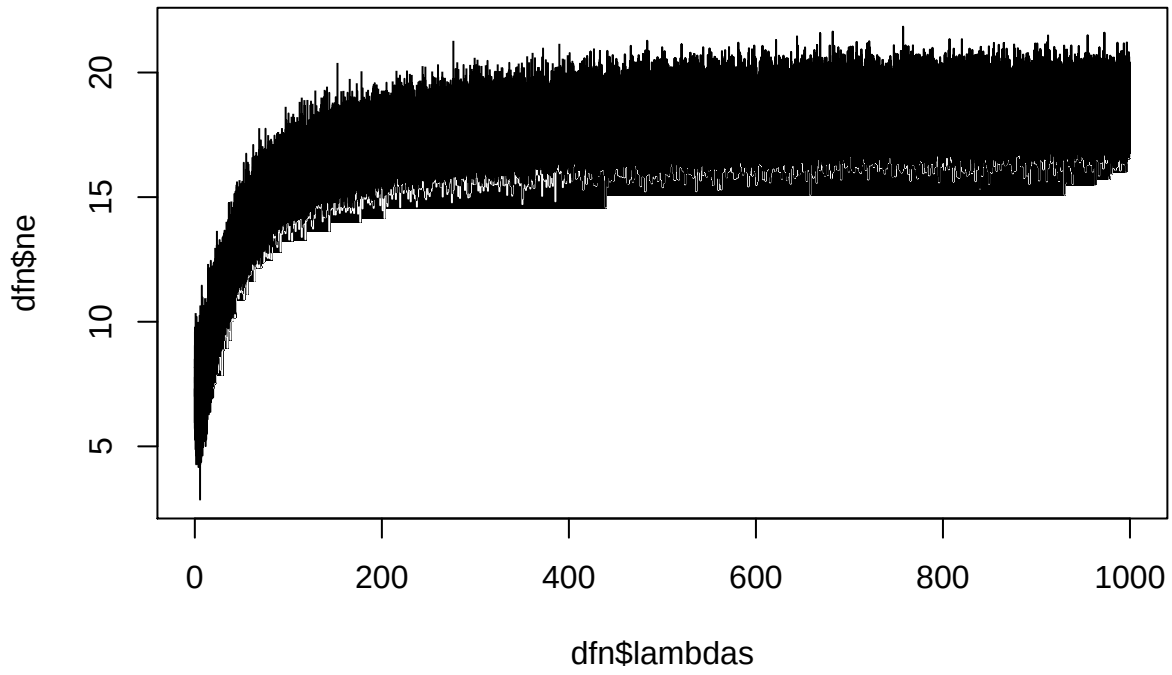


We see that this data appears to be exactly the same, except rescaled, which is what I wanted. So now I will retry the smoothing CVE stuff using the dimensionless case.

```
lambdas <- seq(0.01, 1000, 0.01)
ne <- numeric(length(lambdas))
for (i in 1:length(lambdas)) {
  cves <- replicate(10,
                    HMBGR::cross_validate(normal_data,
                                          k = 3,
                                          reduced = TRUE,
                                          lambda = lambdas[[i]]),
                    simplify = TRUE)
  ne[i] <- mean(cves)
}
```

And now we can visualize the new CVEs to see if this method works better than trying to fit the carrying capacity.

```
dfn <- data.frame(lambdas, ne)
plot(dfn$lambdas, dfn$ne, type = "l")
```



We have the highest CVE as 21.8389036 and the lowest CVE as 2.8644359, which occurs at a lambda value of 5.59. Let's zoom in.

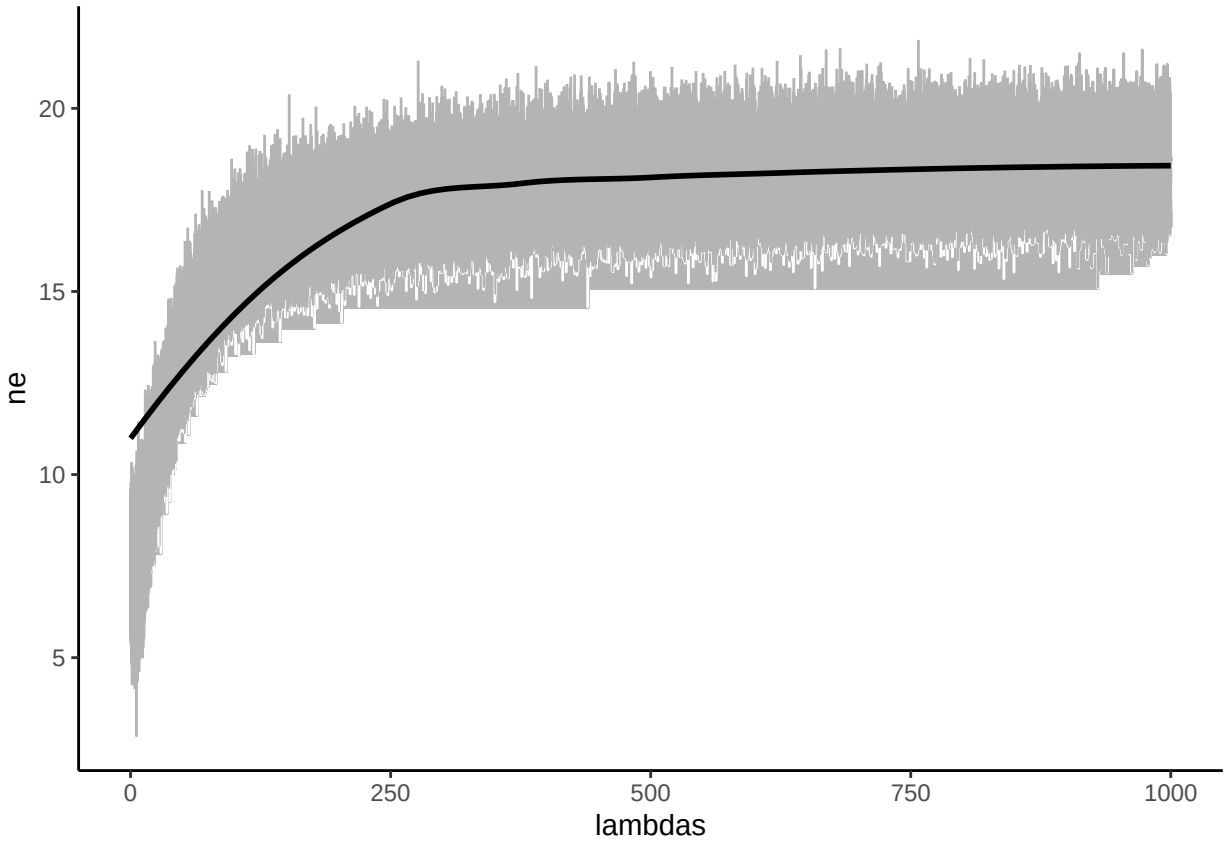
```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

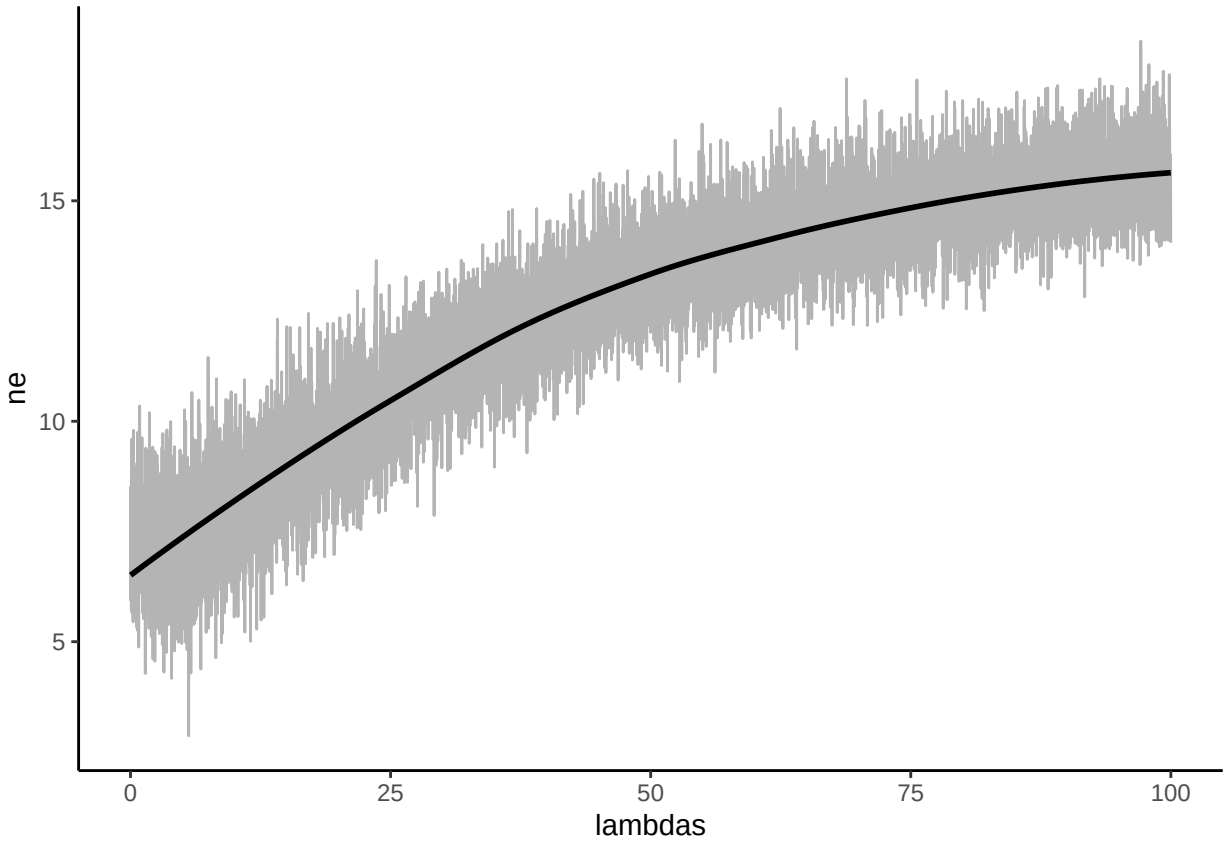
```
## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  3.0.0     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

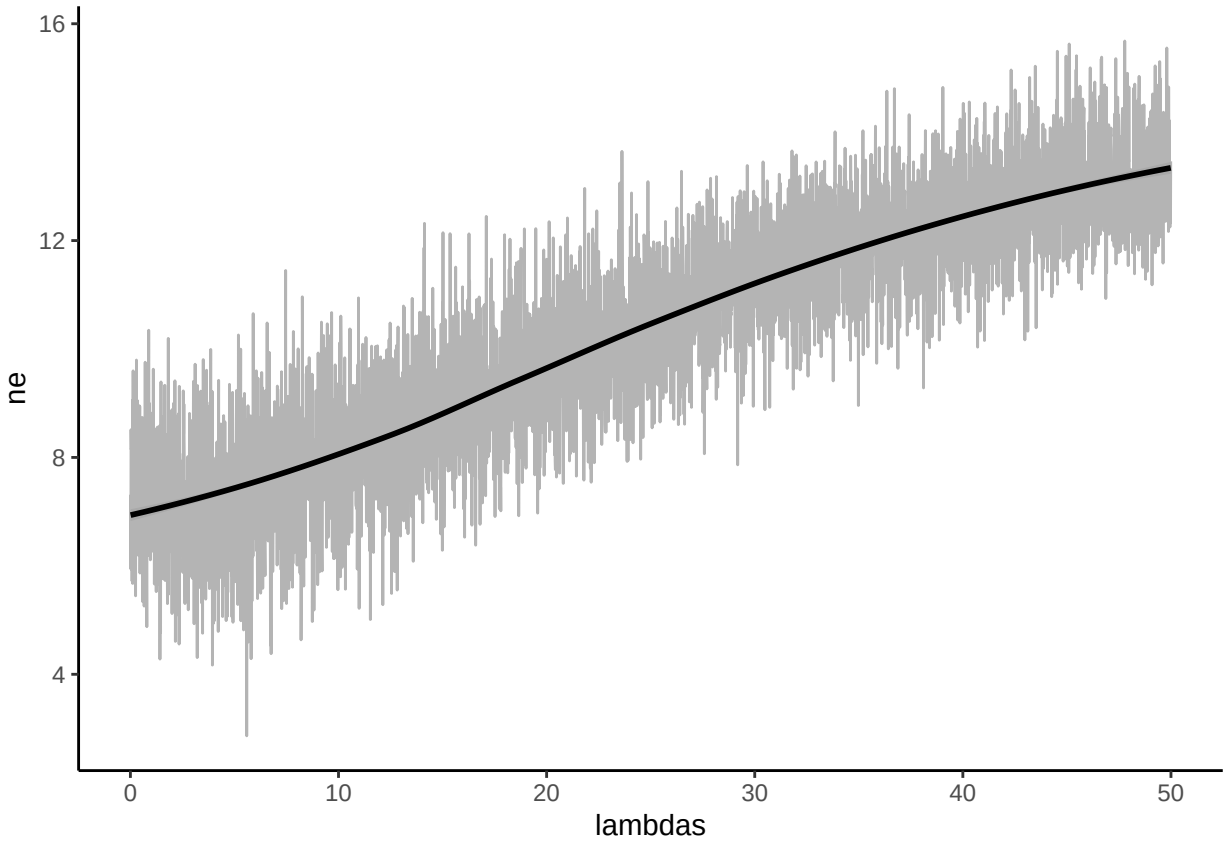
```
dfn %>%
  ggplot(aes(x = lambdas, y = ne)) +
  geom_line(alpha = 0.3) +
  geom_smooth(method = "loess", formula = y~x, col = "black", se = FALSE) +
  theme_classic()
```




```
dfn %>%
  filter(lambdas < 100) %>%
  ggplot(aes(x = lambdas, y = ne)) +
  geom_line(alpha = 0.3) +
  geom_smooth(method = "loess", formula = y~x, col = "black", se = TRUE) +
  theme_classic()
```




```
dfn %>%
  filter(lambdas < 50) %>%
  ggplot(aes(x = lambdas, y = ne)) +
  geom_line(alpha = 0.3) +
  geom_smooth(method = "loess", formula = y~x, col = "black", se = TRUE) +
  theme_classic()
```



TO-DO: Use the final optimal values of lambda in both cases to compute the actual model, then compare the results to the true values we already know using the t-test for regression parameters.