New_Algorithm

Andrew Glover

2023-11-11

Setup

Loading Package Functions

```
devtools::load_all()
```

Libraries Used

```
library(tibble)
library(ggplot2)
library(magrittr)
library(dplyr)
library(patchwork)
# loads the functions from the package
```

We are going to assume that we know p_0 exactly. That way it is either a data point in actual performance, with no cost, or we don't count it as an actual performance.

Algorithm Functions:

Helpers

```
# function inside of parameter distance
lim_func <- function(k_1, k_2, tau_1, tau_2) {
  k_1*exp(-1/tau_1)/(1-exp(-1/tau_1))-k_2*exp(-1/tau_2)/(1-exp(-1/tau_2))
}

# computes "parameter distance"
params_dist <- function(params_1, params_2) {
  # ignore params_1[[1]] since it is just the inital value
  abs(lim_func(params_1[[2]], params_1[[3]], params_1[[4]], params_1[[5]]) -
        lim_func(params_2[[2]], params_2[[3]], params_2[[4]], params_2[[5]])
      )
}
```

```r
# plotting function
plot_perf <- function(observed_performance,
                      modeled_performance) {
  data <- tibble(
  "day" = c(0:length(observed_performance)),
  "pred" = modeled_performance,
  "obs" = c(NA,observed_performance)
  )

  plot <- ggplot(data, aes(x = day)) +
  geom_point(aes(y = pred, color = "pred")) +
  geom_point(aes(y = obs, color = "obs")) +
  labs(x = "Day",
       y = "Performance",
       title = "")
  plot
}

# chooses the optimal parameters from a parameter matrix
# this is farily optimized, and is faster than an application of
# r's built in minimia function
min_params <- function(params_matrix,
                       training_load,
                       obs_perf,
                       old_params,
                       lambda) {
  min_params <- old_params
  obs_indexes <- which(!is.na(obs_perf))
  n <- length(obs_indexes)
  min_cost <- sqrt(SSE(old_params, training_load, obs_perf)/n)
  for (i in 1:nrow(params_matrix)){
    params_i <- as.numeric(params_matrix[i, ])
    params_dist_i <- params_dist(params_i, old_params)
    SSE_i <- 0; cost_i <- 0
    p_0 <- params_i[[1]]
    k_1 <- params_i[[2]]; k_2 <- params_i[[3]]
    coef_1 <- exp(-1/params_i[[4]]); coef_2 <- exp(-1/params_i[[5]])
    T_1 <- 0; T_2 <- 0
    lower_index <- 1
    j <- 0
    for (new_index in obs_indexes) {
      for (t in lower_index:new_index) {
        T_1 <- coef_1*(T_1 + k_1*training_load[[t]])
        T_2 <- coef_2*(T_2 + k_2*training_load[[t]])
      }
      SSE_i <- SSE_i + (p_0 + T_1 - T_2 - obs_perf[[new_index]])^2
      j <- j + 1
      lower_index <- new_index + 1
      if(min_cost < sqrt(SSE_i/n) + lambda*params_dist_i) {
        break # goes to next set of parameters
        # for a lot of the sets of parameters,
        # the condition should trigger on the first few data points
      }
```

```
      if (j == n && min_cost > sqrt(SSE_i/n) + lambda*params_dist_i) {
        # the second condition helps with stability, if two different sets
        # of parameters have the same cost -- namely when they both have 0 cost --
        # we will pick our previous point
        min_cost <- sqrt(SSE_i/n) + lambda*params_dist_i
        min_params <- params_i

      }
      else {
        # do nothing; go to the next loop
      }
    }
  }
  return(list("cost" = min_cost, "opt_params" = min_params))
}

# this function creates (or calls, we will see) the parameter matrix and applies
# it to the previous function. makes things tider in the algorithm funciton
update_params_opt <- function(old_params,
                          training_load,
                          sub_obs_perf,
                          bounds_type = list("test"),
                          lambda
                          ) {
  if(bounds_type == "test") {
    params_matrix <- expand.grid(p_0 = 500,
                                 k_1 = c(1,2,3,4),
                                 k_2 = c(2,4,6,8),
                                 tau_1 = c(5:35),
                                 tau_2 = c(5:35))
  }
  params_matrix <- params_matrix[params_matrix$tau_1 <= params_matrix$tau_2,]

  getting_params_ouput <- min_params(params_matrix,
                                 training_load,
                                 sub_obs_perf,
                                 old_params,
                                 lambda = lambda)

  out_list <- list()
  out_list$opt_params <- getting_params_ouput$opt_params
  out_list$cost <- getting_params_ouput$cost
  return(out_list)
}
```

The Important Function

```
new_pred_perf <- function(init_params,
                          training_load,
                          obs_perf,
                          bounds_type = list("test"),
```

```
                         lambda) {
# initializations
curr_params <- init_params
curr_cost <- 0
####
# curr_params take form c(k_1, k_2, tau_1, tau_2), to work with the time-invariant
# functions
###

matrix_params <- matrix(0, nrow = length(training_load), ncol = 5)
colnames(matrix_params) <- c("p_0", "k_1", "k_2", "tau_1", "tau_2")

days <- length(training_load)
perf_out <- c(rep(curr_params[[1]], days + 1))
cost_vec <- c(rep(0, days + 1))
T_1 <- 0
T_2 <- 0

# doing the algorithm
for (i in 1:length(obs_perf )) {
  # update params if there is new information
  if (is.na(obs_perf[[i]]) == FALSE) {
    params_output <- update_params_opt(
      old_params = init_params,
      training_load = training_load,
      sub_obs_perf = obs_perf[1:i],
      # the subset up to observation
      bounds_type = bounds_type,
      lambda = lambda
    )
    curr_params <- params_output$opt_params
    curr_cost <- params_output$cost

  } else {
  } # pass
  T_1 <- exp(-1 / curr_params[[4]]) * (T_1 + curr_params[[2]] * training_load[[i]])
  # training load index is i-1 since i starts at 2
  T_2 <- exp(-1 / curr_params[[5]]) * (T_2 + curr_params[[3]] * training_load[[i]])
  perf_out[[i+1]] <-
    curr_params[[1]] +  T_1 - T_2
  cost_vec[[i]] <- curr_cost
  ##############
  # fix cost it is wrong,
  #############
  matrix_params[i, ] <- as.numeric(curr_params)
}

#plotting
params_data <- tibble(
  "day" = 0:length(training_load),
  "k_1" = as.numeric(c(matrix_params[1, "k_1"], matrix_params[, "k_1"])),
  "k_2" = as.numeric(c(matrix_params[1, "k_2"], matrix_params[, "k_2"])),
  "tau_1" = as.numeric(c(matrix_params[1, "tau_1"], matrix_params[, "tau_1"])),
```

```
    "tau_2" = as.numeric(c(matrix_params[1, "tau_2"], matrix_params[, "tau_2"])),
    "p_0" = as.numeric(c(matrix_params[1, "p_0"], matrix_params[, "p_0"]))
  )
  k_plot <- ggplot(params_data, aes(x = day)) +
    geom_line(aes(y = k_1, color = "k_1")) +
    geom_line(aes(y = k_2, color = "k_2"))

  tau_plot <- ggplot(params_data, aes(x = day)) +
    geom_line(aes(y = tau_1, color = "tau_1")) +
    geom_line(aes(y = tau_2, color = "tau_2"))

  #outputs
  out_list <- list()
  out_list$performance <- perf_out
  out_list$cost_vec <- cost_vec
  out_list$perf_plot <- plot_perf(obs_perf, perf_out)
  out_list$params_plots <- k_plot + tau_plot
  return(out_list)
}
```

Algorithm Pseudo-code, not complete

Inputs:

- Actual performance, vector of length n
- Training load, vector of length n
- Initial guess of parameters, vector of length 4
- p_0, a number
- Ranges to search over for each variable, The length of these vectors multiply to ℓ

Outputs,

Algorithm:

(1) Accept inputs
(2) Define

- `pred_perf`, a vector of length $n+1$, initalized with p_0
- `cur_param`, a vector of length 4, used to keep track of the parameters used on each step
- `cost`, a number to keep track of the cost function

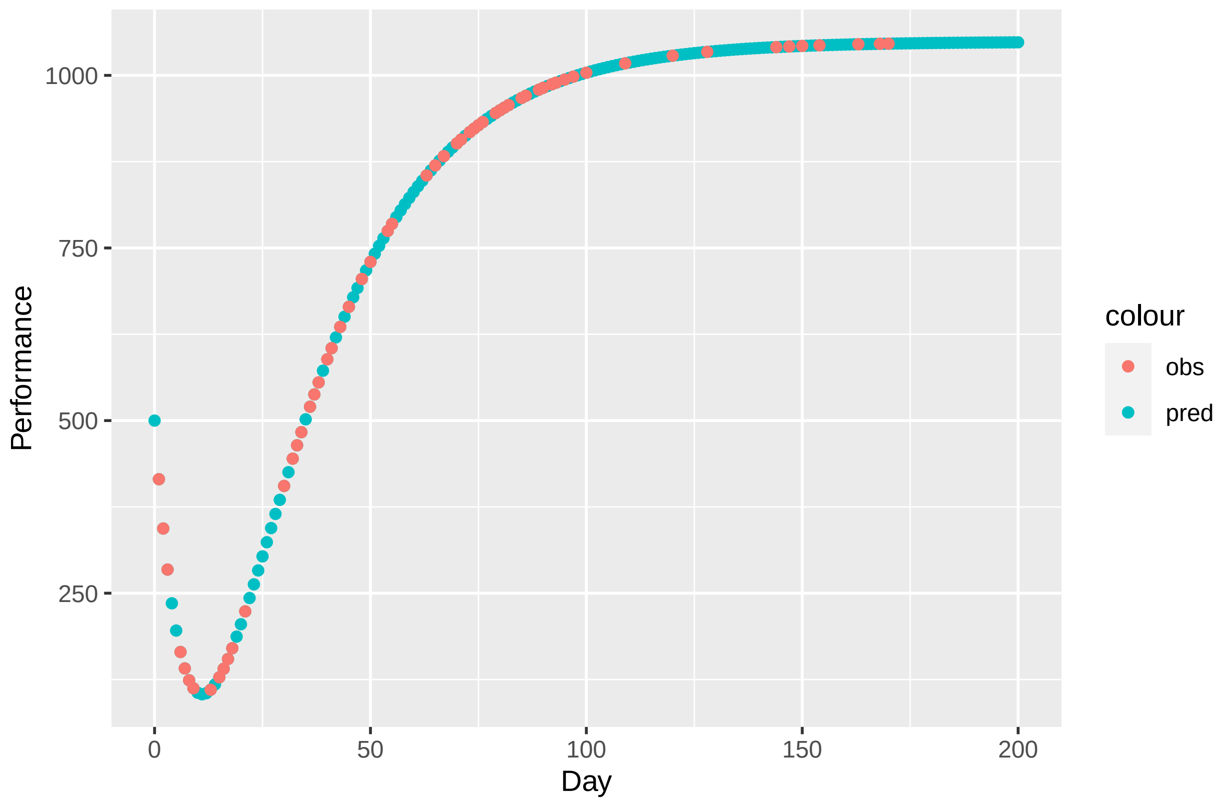
For $i = 0, 1 \ldots, n$ If Actual Performance at i is equal to NA, vvapply `curr_param` to predicted performance recursive equation to get predicted performance set `pred_perf[[i]]` to be this predicted performance

Testing New Algorthm on simulated data

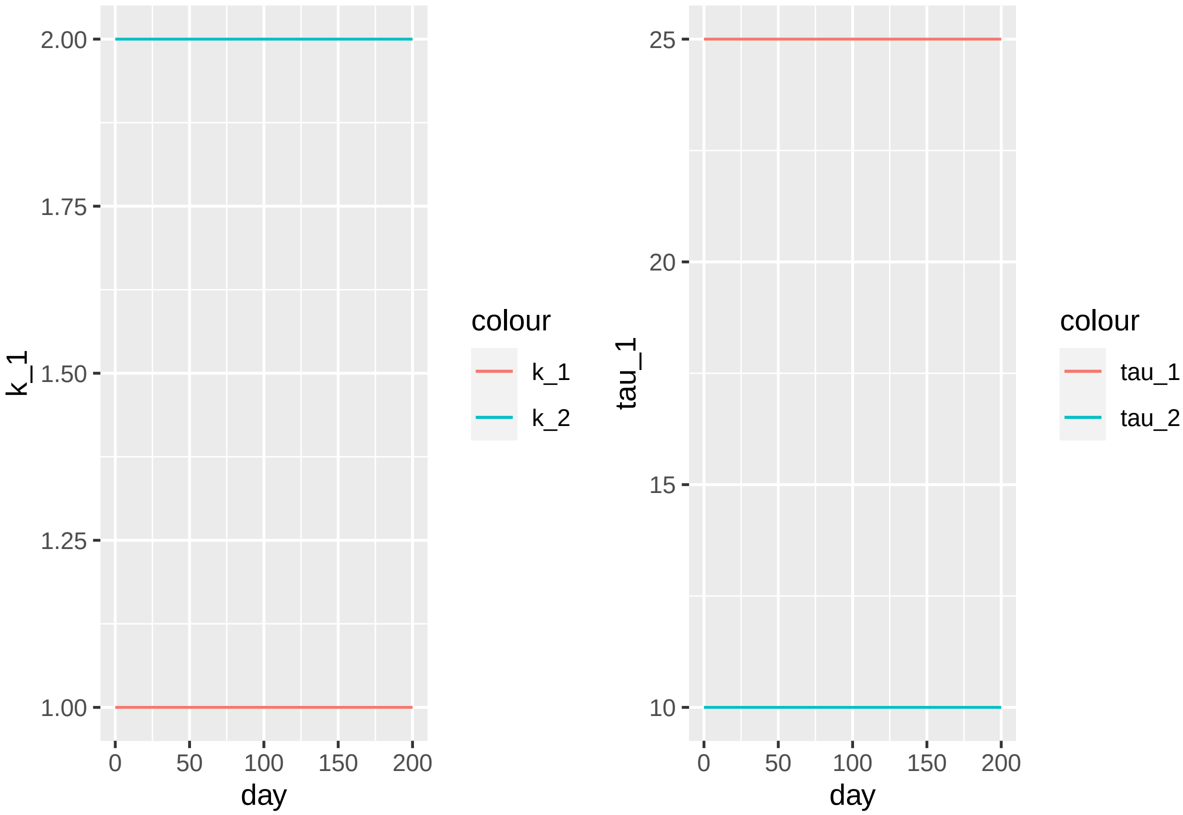
basic testing

As a sanity check, we give the algorithm the true values of the parameters as the initial parameters, tested aginst the the actual performance with no noise

```r
set.seed(2)
days_test <- 200
perf_sim_1 <- c(500, perf_tv(p_0 = 500,
          k_1 = 1,
          tau_1 = 25,
          k_2 = 2,
          tau_2 = 10,
          days = days_test,
          training_stim = list("constant", 100))$performance
)
noise_fn <- function(x,
                     sd){
  x+rnorm(1, mean = 0, sd = sd)
}

# perf_sim_1 = purrr::map_dbl(perf_sim_1, noise_fn, sd = 20)

# this is one off
samp_index <- c(sample(c(1:100), 50), sample(c(101:days_test), 10))
actual_perf <- rep(NA, days_test)
for (i in samp_index) {
  actual_perf[[i]] = perf_sim_1[[i+1]]

}

model_perf <- new_pred_perf(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = 1
)
model_perf$perf_plot
```




```
model_perf$params_plots
```



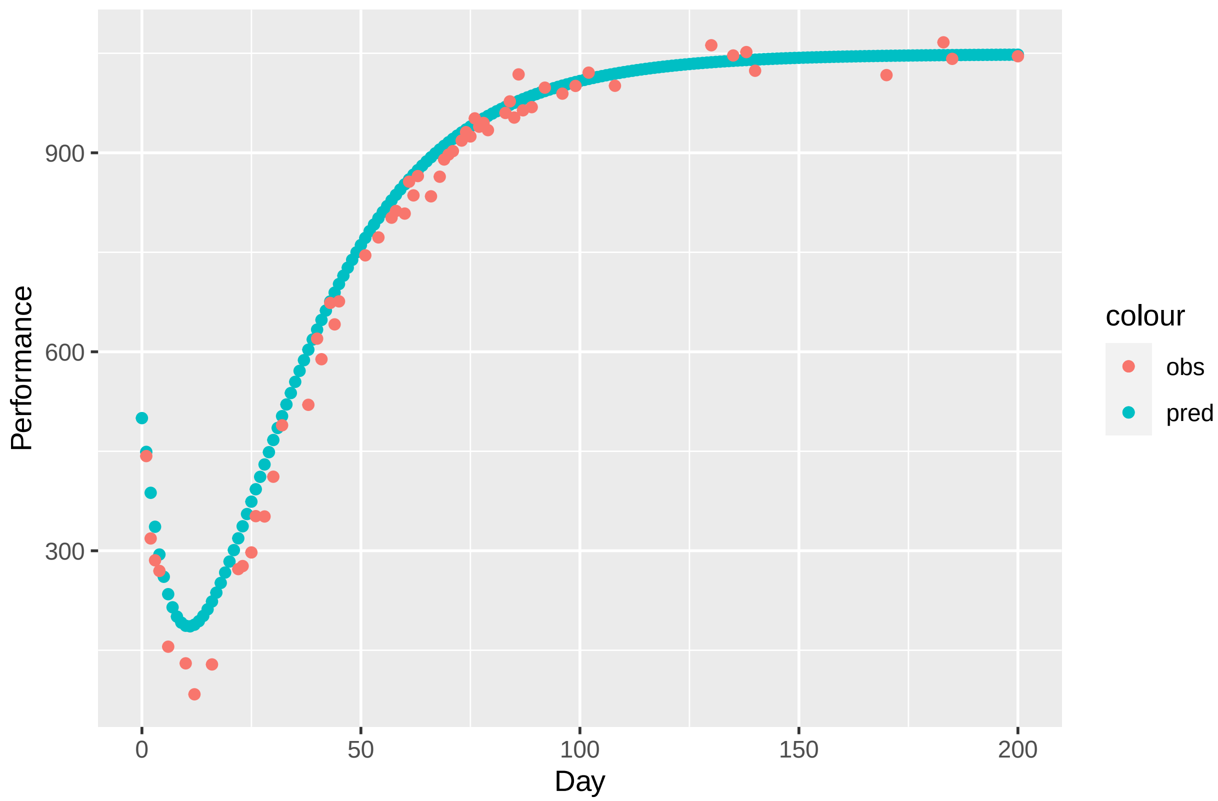
This checks out. It has passed the sanity check. The algorithm gives the same resutls no matter what the value of `lambda` is.

adding noise

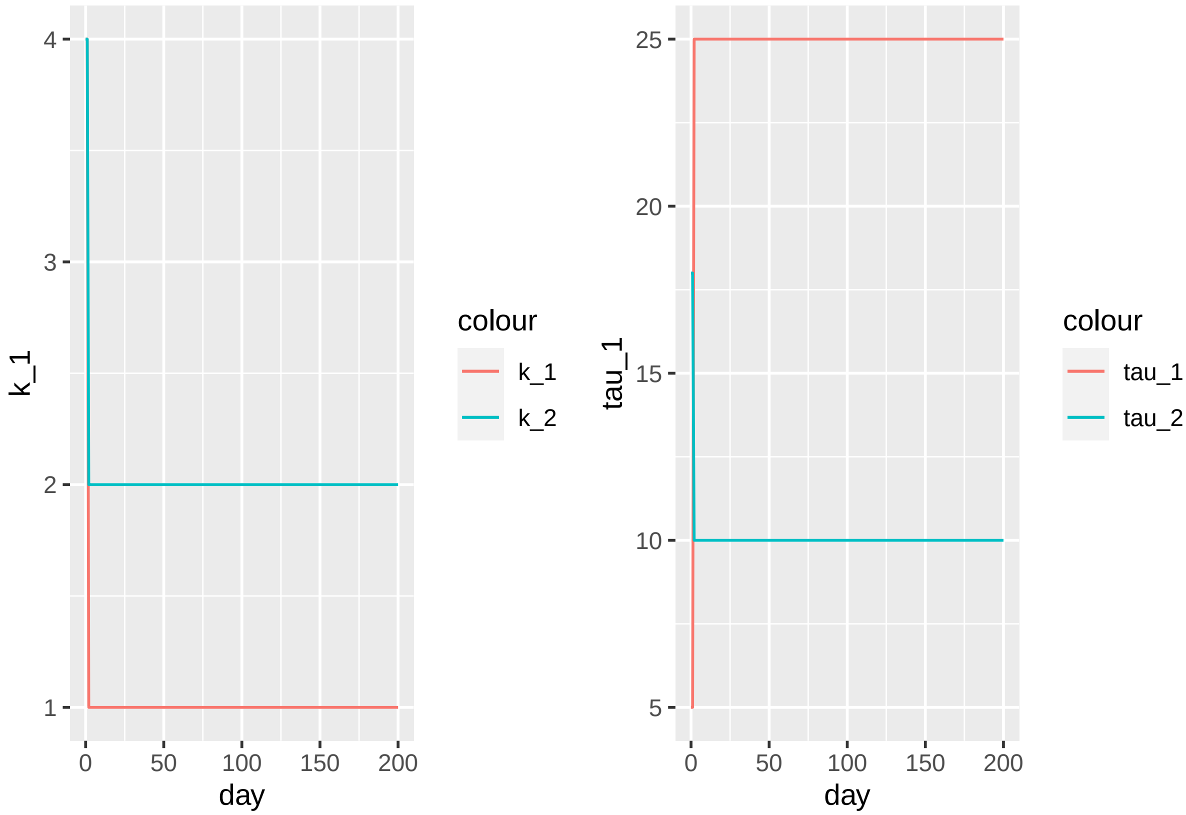
Running the same simulation with gaussian noise to the simulated performance, with standard deviation of 20. We will see that increasing lambda smoothes out the performance curves. This first simulation is with `lambda = 0` which just finds the minimum parameters with respect to SSE.

```
set.seed(5)
perf_sim_1 = purrr::map_dbl(perf_sim_1, noise_fn, sd = 20)
samp_index <- c(sample(c(1:100), 50), sample(c(101:days_test), 10))
actual_perf <- rep(NA, days_test)
for (i in samp_index) {
  actual_perf[[i]] = perf_sim_1[[i+1]]
}
model_perf <- new_pred_perf(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = .3
)
model_perf$perf_plot
```

```
## Warning: Removed 141 rows containing missing values (‘geom_point()‘).
```



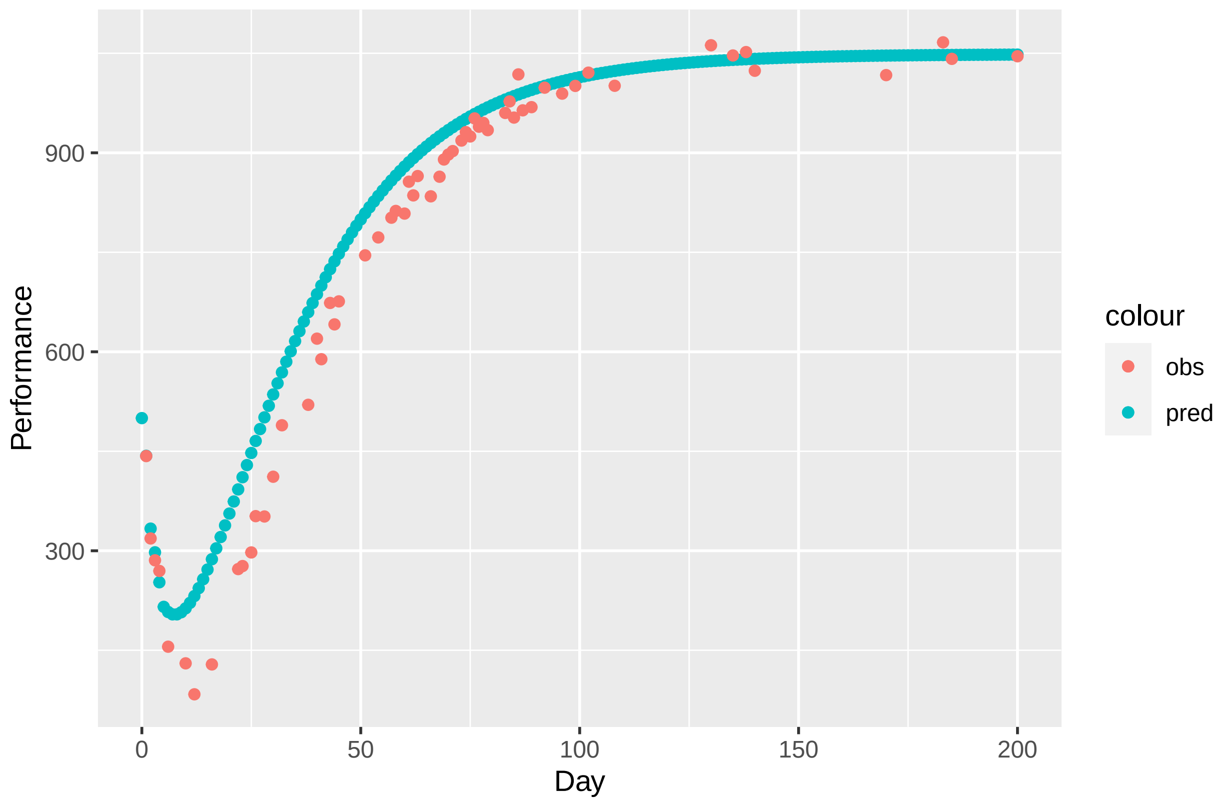

```
model_perf$params_plots
```



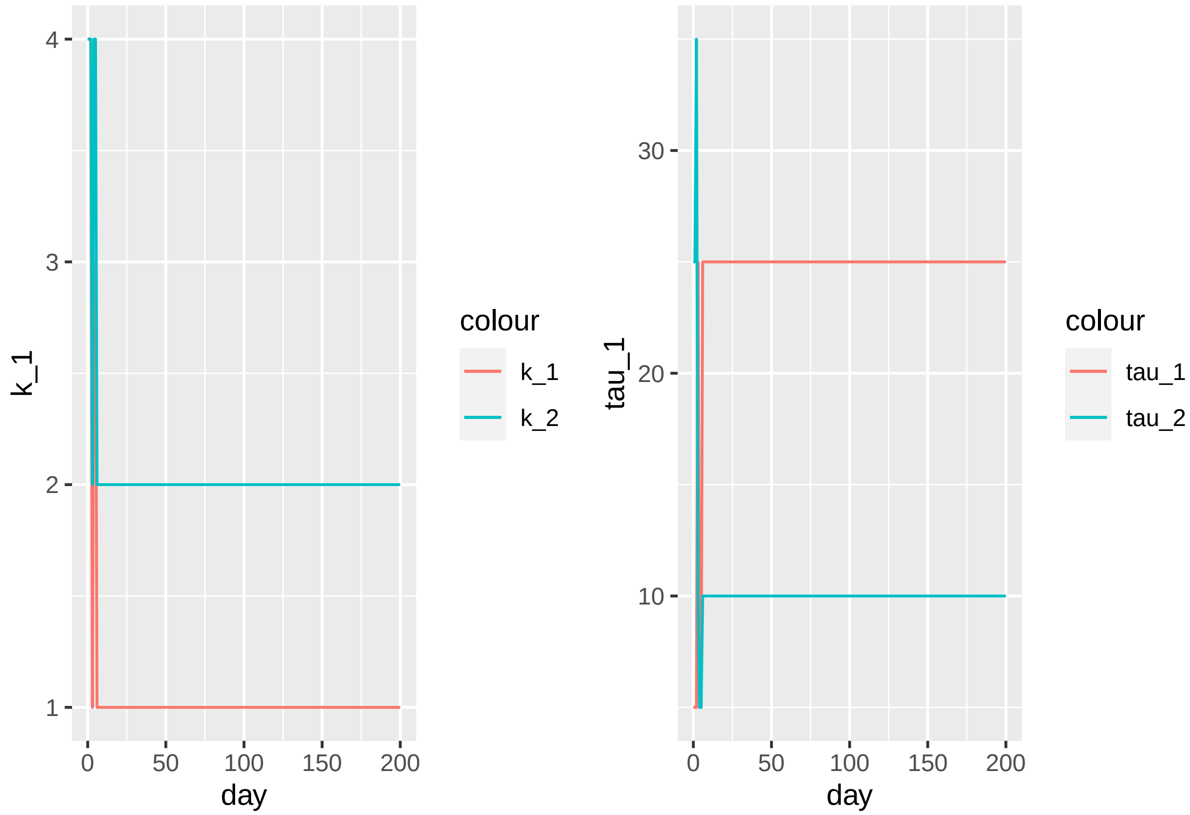
This kind of wanders off

```
model_perf <- new_pred_perf(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = .1
)
model_perf$perf_plot
```

```
## Warning: Removed 141 rows containing missing values ('geom_point()').
```



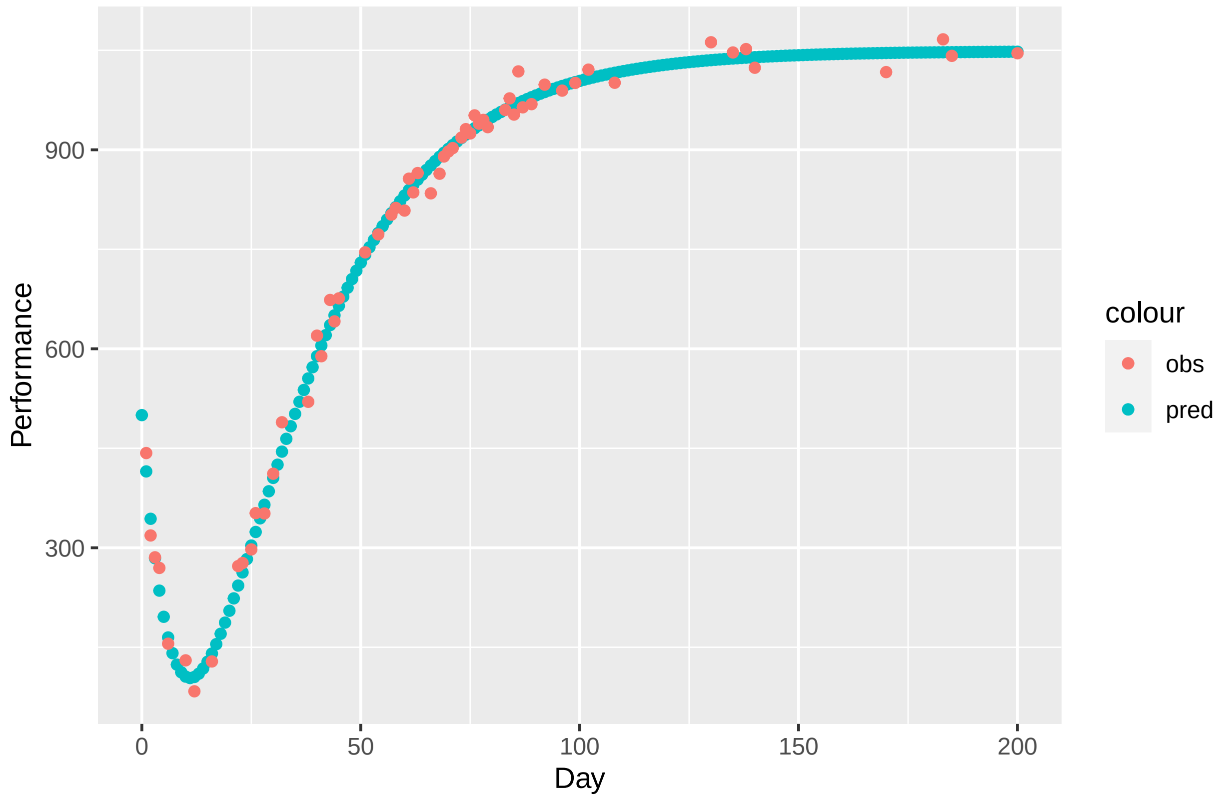

```
model_perf$params_plots
```



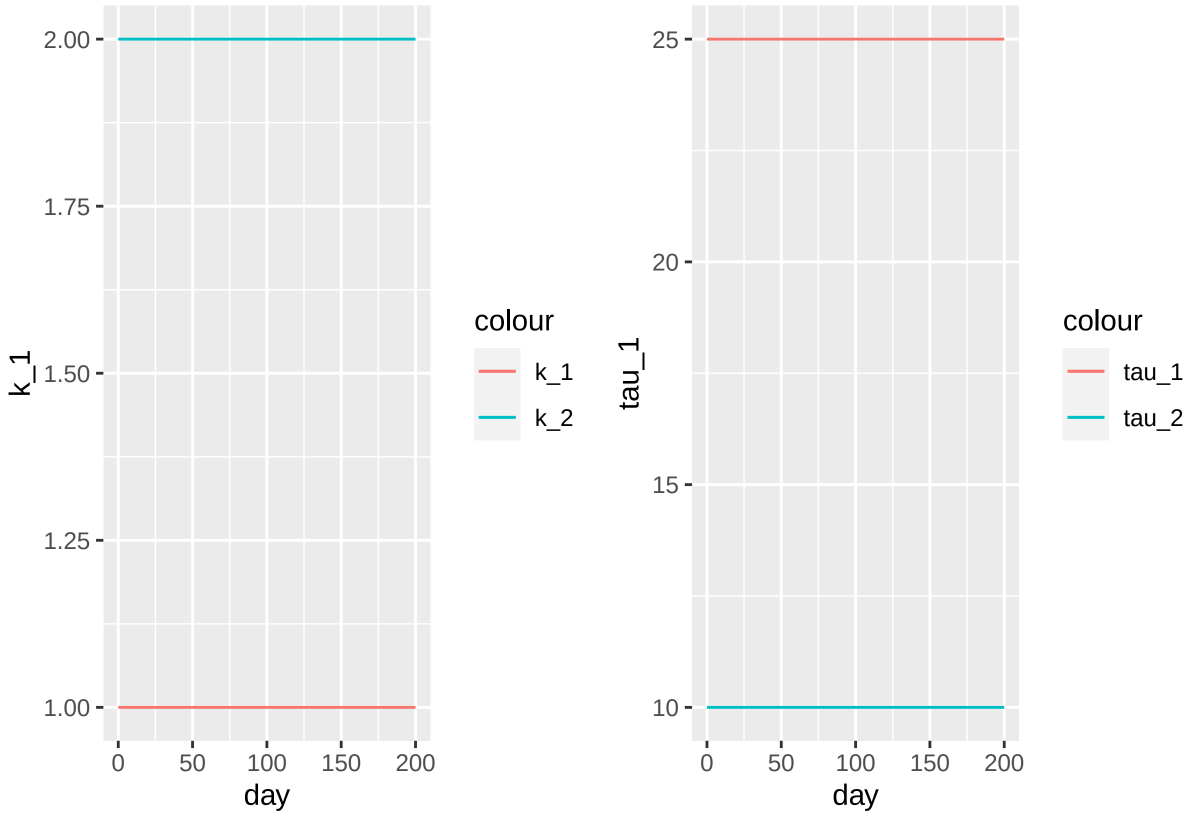
The wandering is fixed when we introduce some penalty for the distance function

```
model_perf <- new_pred_perf(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = .5
)
model_perf$perf_plot
```

```
## Warning: Removed 141 rows containing missing values ('geom_point()').
```



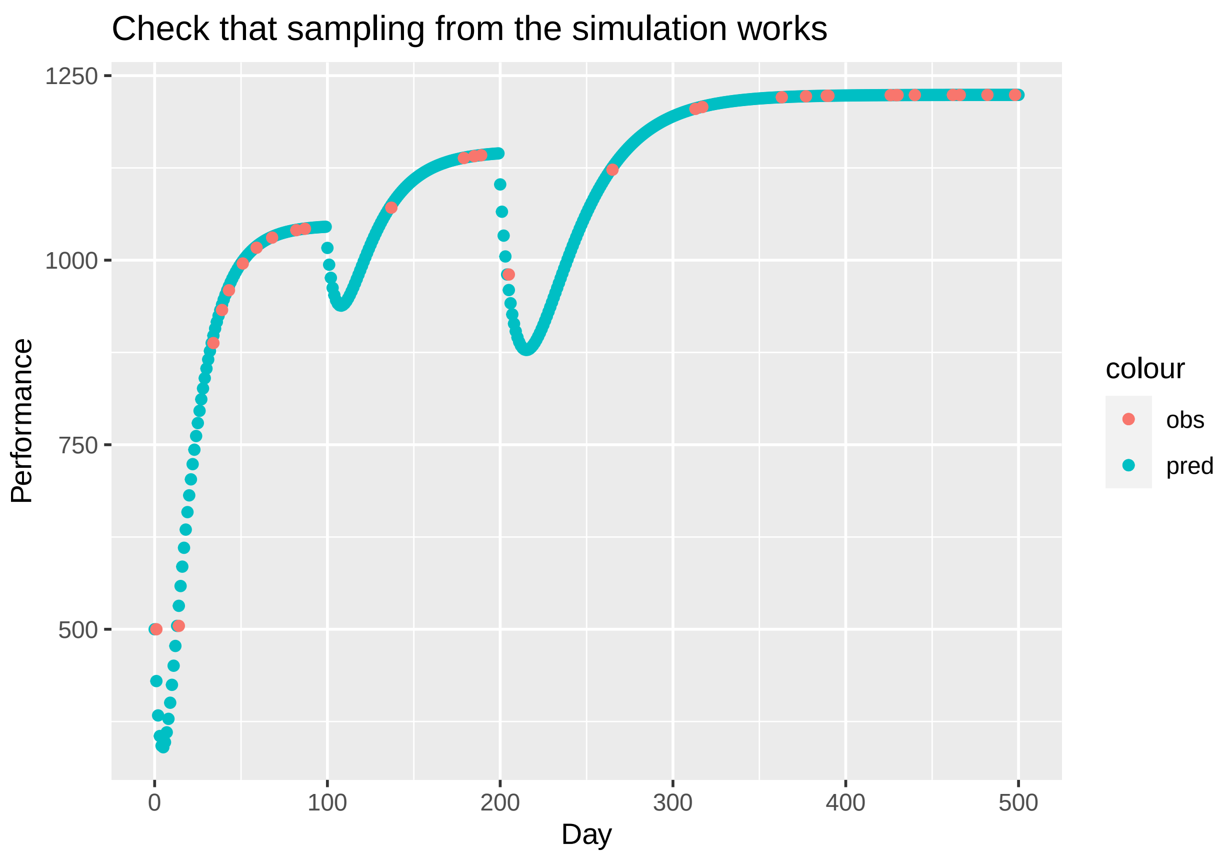

```
model_perf$params_plots
```



More complicated simulation Working on simulating a situation where the parameter change over time, but the simulated perforamance level is not realistic. Right now, the algorthm cannot handle this case.

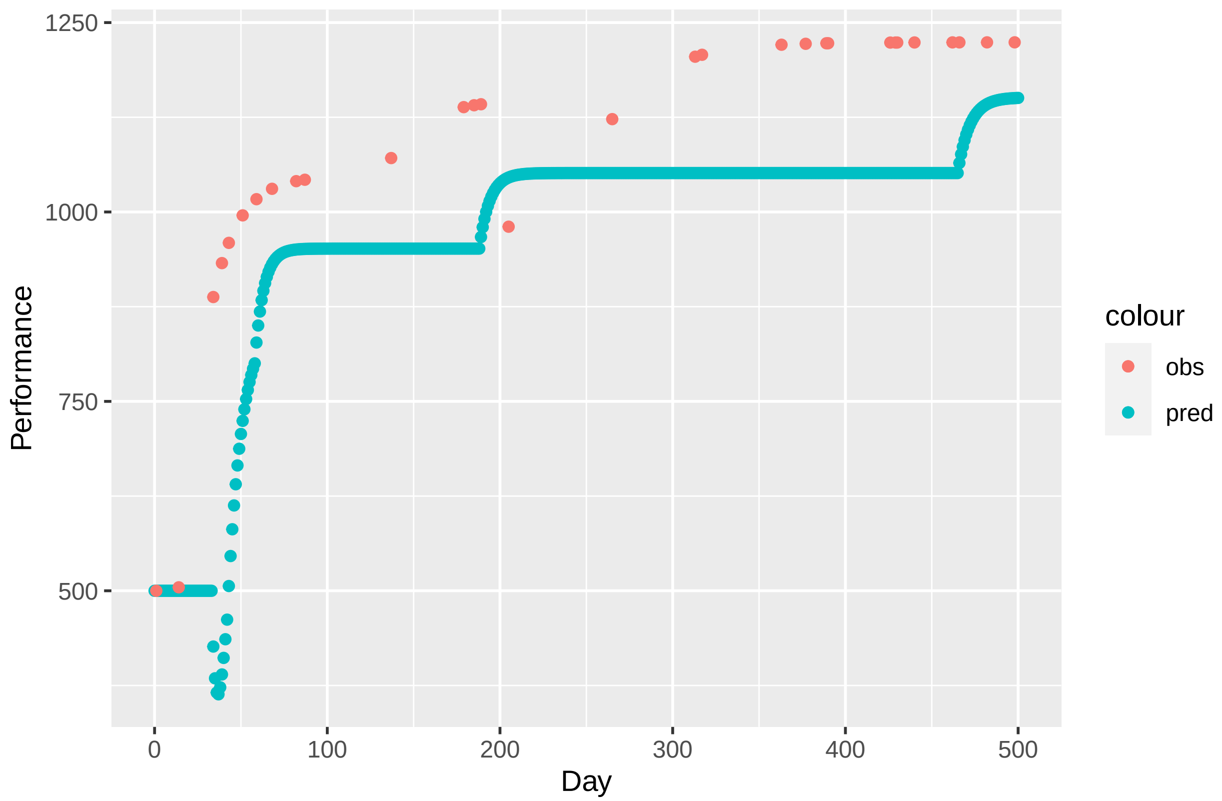
```
set.seed(1)
days_test <- 500
perf_sim_1 <- c(500, perf_tv(p_0 = 500,
          k_1 = c(1,1,1.1),
          tau_1 = c(15, 20, 28),
          k_2 = c(2, 2, 2),
          tau_2 = c(5,7, 12),
          change_days = c(100, 200),
          days = days_test,
          training_stim = list("constant", 100))$performance
)
samp_index <- c(sample(c(1:100), 10), sample(c(101:days_test), 20))
actual_perf <- rep(NA, days_test)
for (i in samp_index) {
  actual_perf[[i]] = perf_sim_1[[i]]
}

plot_check <- plot_perf(actual_perf,
                        perf_sim_1) +
  labs(title = "Check that sampling from the simulation works")
plot_check
```

```
## Warning: Removed 471 rows containing missing values (‘geom_point()‘).
```

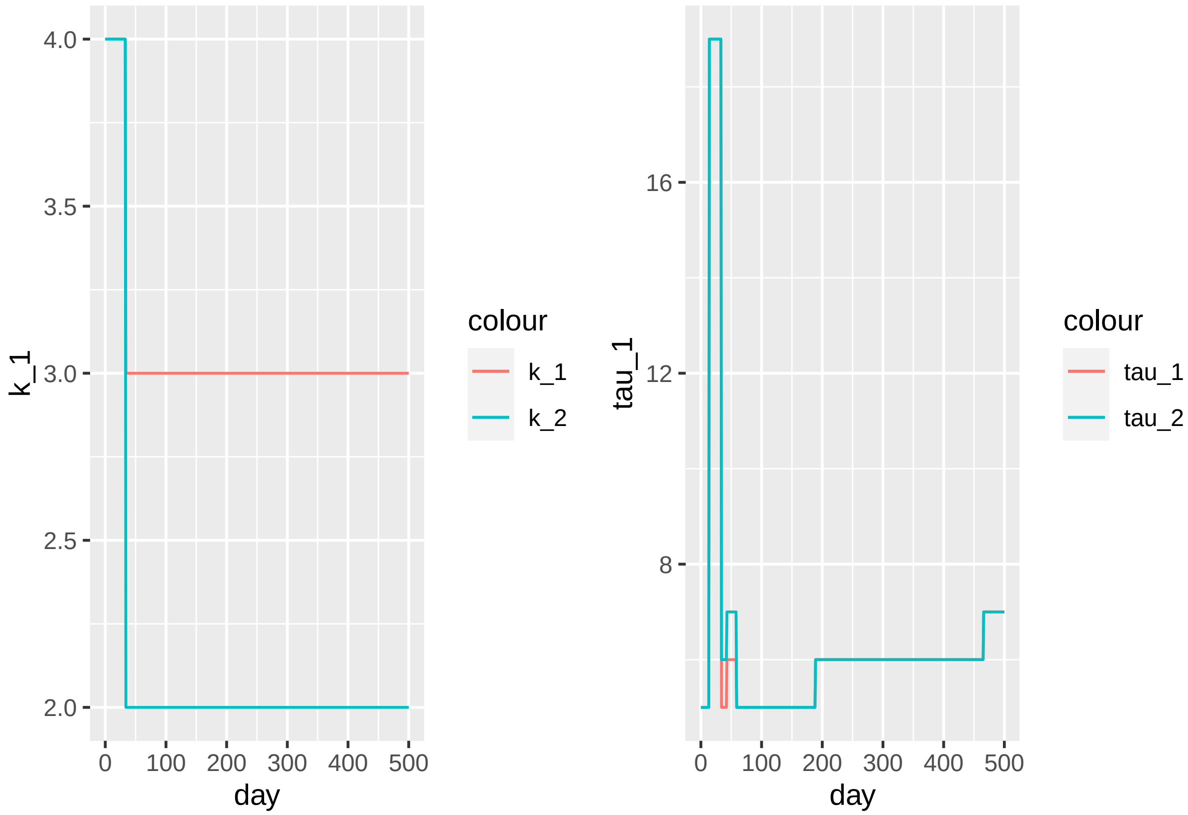



```
model_perf_2 <- new_pred_perf(
  c(500, 1,2, 25, 10),
  c(rep(100, days_test)),
  actual_perf,
  lambda = 2
)
model_perf_2$perf_plot
```

```
## Warning: Removed 471 rows containing missing values (‘geom_point()‘).
```




```
model_perf_2$params_plots
```



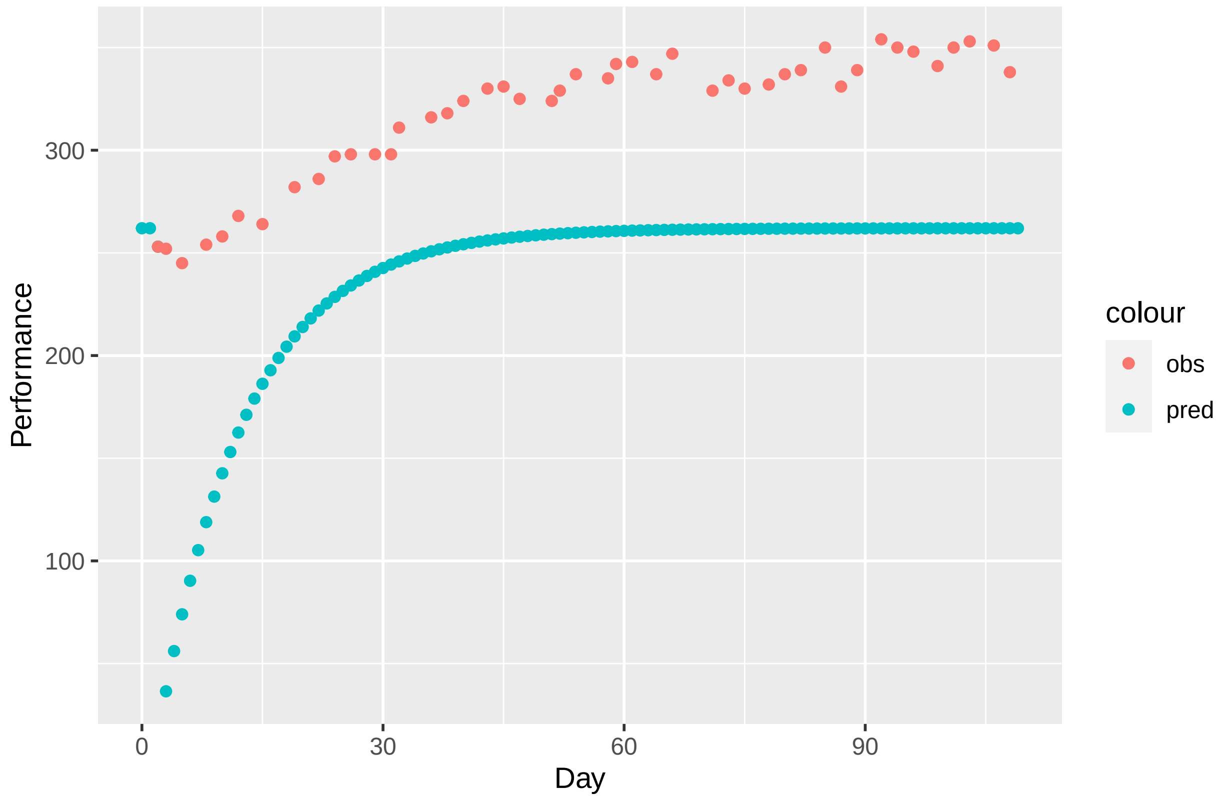
It does a pretty bad job. This is because the algorithm, at every new data point, tries to fit the best Time-*Invariant* curve. In this situation, all of the Time-Invariant functions don't fit well when the parameters first change.

Applying to Real Data

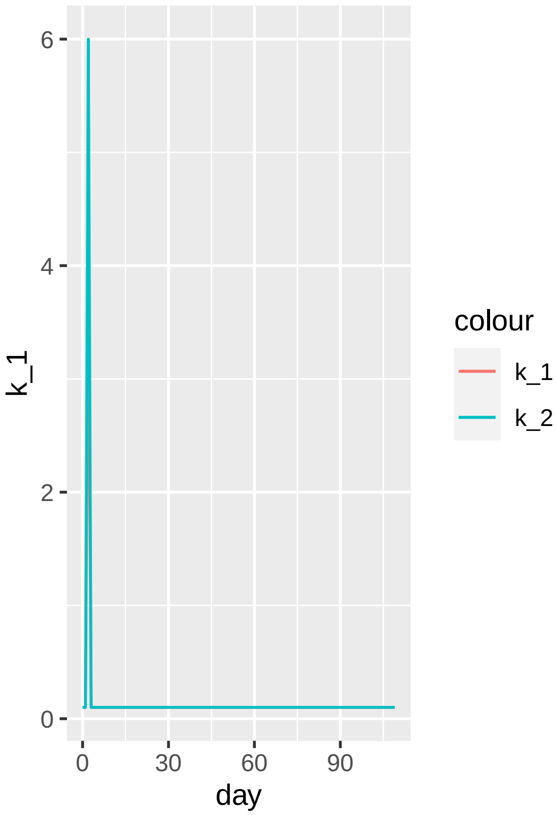
This is a work in progress. Right now, the algorithm cannot handle this case

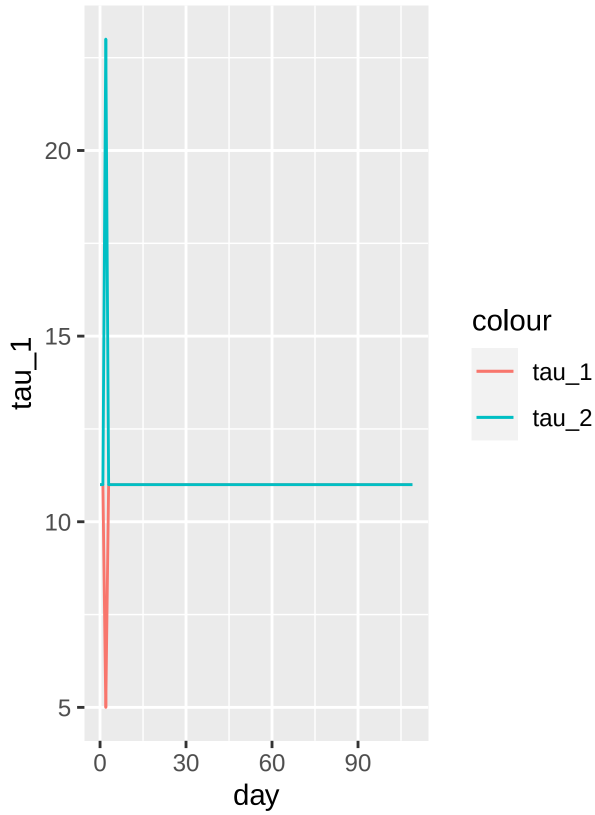
```
real_data <- data_2
day_vec_real <- real_data[[1]]
training_load_real <- real_data[[2]]
obs_perf_real <- real_data[[3]]
init_params_real <- c(262, .1, .1, 11, 11)
obs_new_alg <- new_pred_perf(
  init_params_real,
  training_load_real,
  obs_perf_real,
  lambda = .000000000005
)

obs_new_alg$perf_plot
```

```
## Warning: Removed 65 rows containing missing values (‘geom_point()‘).
```




```
obs_new_alg$params_plots
```





Here, it does not change because the algorithm has decided that all of the options that we have told it to search over are all bad so it stuck with the initial estimate which is probably a good sign.