Simulating Time-Varying Performance

Andrew Glover

2023-10-09

Libraries

```r
library(ggplot2) # for graphing
library(patchwork) # to add graphs together
library(tibble) # tibbles
```

Functions used for analysis

```r
params_mat <- function(k_1, tau_1, k_2, tau_2, change_days=NULL, days) {
  if (length(k_1) == length(tau_1) &&
      length(k_1) == length(tau_2) &&
      length(k_1) == length(k_2) &&
      length(k_1) == length(change_days)+1
  ) {}
    else {stop("check length of parameters")}
  out_matrix <- matrix(0, nrow = days, ncol = 4)
  colnames(out_matrix) <- c("k_1", "tau_1", "k_2", "tau_2")
  bound_1 <- 1; bound_2 <- days
  j <- 0 # counter for index of k_1, tau_1, etc
  for (elem in c(change_days, days)) {
    j <- j + 1
    bound_2 <- elem
    for (i in bound_1:bound_2) {
      out_matrix[i, ] <- c(k_1[[j]], tau_1[[j]], k_2[[j]], tau_2[[j]])
    }
    bound_1 <- elem
  }
  return(out_matrix)
}
```

```r
#' Title
#'
#' @param params_mat a n by 4 matrix, where n is the number of days
#' @param training_load an n dimensional vector
#'
#' @return an n? dimensional vector with the performance
#' @export
```

```r
#'
#' @examples
perf_tv <- function(p_0, params_mat, training_load) {
  days <- nrow(params_mat)
  perf_out <- c(rep(NA, days))
  T_1 <- 0; T_2 <- 0
  for (i in 1:days) {
    T_1 <- exp(-1/params_mat[i, "tau_1"])*T_1 + training_load[[i]]
    T_2 <- exp(-1/params_mat[i, "tau_2"])*T_2 + training_load[[i]]
    perf_out[[i]] <- p_0 + params_mat[i, "k_1"]*T_1 - params_mat[i, "k_2"]*T_2
  }
  return(perf_out)
}
```

```r
perf_plot <- function(p_0,
                      k_1,
                      tau_1,
                      k_2,
                      tau_2,
                      change_days = NULL,
                      days,
                      training_stim) {
  training_load <- c()
  limit <- 0
  if (training_stim[[1]] == "constant") {
    training_load <- c(rep(training_stim[[2]], days))
    k <- length(k_1)

    # See justification for this below
    limit <- p_0 + training_stim[[2]]*k_1[[k]]/(1-exp(-1/tau_1[[k]])) -
      training_stim[[2]]*k_2[[k]]/(1-exp(-1/tau_2[[k]]))
  }
  tmp_matrix <- params_mat(k_1,
                           tau_1,
                           k_2,
                           tau_2,
                           change_days,
                           days)
  modeled_performance <- perf_tv(p_0, tmp_matrix, training_load)
  tmp_data <- tibble(
    "day" = c(0:days),
    "performance" = c(p_0, modeled_performance),
    "limit" = c(rep(limit, days + 1))
  )

  plot <- ggplot(tmp_data, aes(x = day)) +
    geom_point(aes(y = performance, color = "perf")) +
    geom_line(aes(y = limit, color = "lim"))

  #   scale_color_manual("Legend",
  #                      values = c("lim" = "#e31a1c", # this color comes from the theme "Paired"
  #                                 "perf" = "black"))
  plot
}
```

Computing the limit of the model

I would like to compute the limit of the predicted performance for the time-invariant model Under the assumption of constant training load. We have

$$p(t) = p_0 + k_1 \sum_{i=1}^{t-1} e^{\frac{t-i}{\tau_1}} w(i) + k_2 \sum_{i=1}^{t-1} e^{\frac{t-i}{\tau_2}} w(i)$$

Assume that $w(i) = C$ for all i. Note that

$$\sum_{i=1}^{t-1} e^{\frac{t-i}{\tau_1}} = e^{1/\tau_1} \sum_{i=1}^{t-1} e^{\frac{(t-1)-i}{\tau_1}} = e^{1/\tau_1} \left(-1 + \sum_{i=0}^{s} \left(e^{-1/\tau_1}\right)^i \right)$$

Finding the long-run limit of $p(t)$ then amounts to computing

$$\sum_{i=0}^{\infty} \left(e^{-1/\tau_1}\right)^i = \frac{1}{1 - e^{-1/\tau_1}}$$

Notice that this is a convergent geometric series, $e^{-1/\tau_1} < 1$ when $\tau_1 > 1$ (which we have assumed). Therefore,

$$\sum_{i=1}^{t-1} e^{\frac{t-i}{\tau_1}} = e^{1/\tau_1} \left(-1 + \frac{1}{1 - e^{-1/\tau_1}} \right) = e^{1/\tau_1} \left(\frac{e^{-1/\tau_1}}{1 - e^{-1/\tau_1}} \right) = \frac{1}{1 - e^{-1/\tau_1}}$$

Therefore

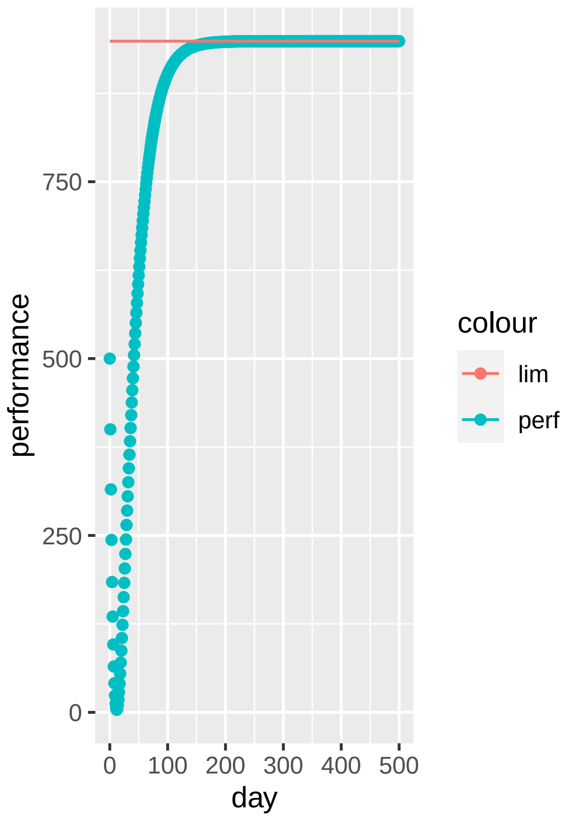
$$\lim_{t \to \infty} p(t) = p_0 + C \left(\frac{k_1}{1 - e^{-1/\tau_1}} - \frac{k_2}{1 - e^{-1/\tau_2}} \right)$$

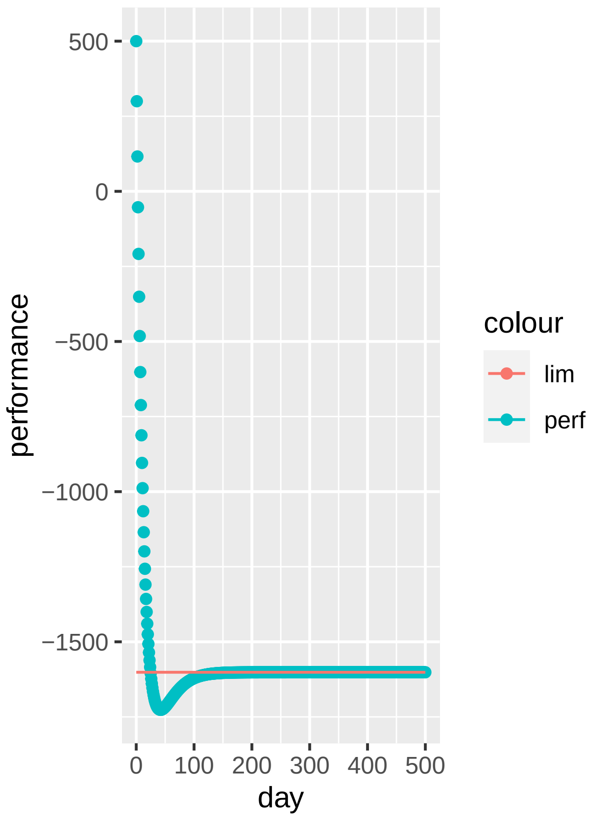
This is how we compute the red line in `perf_plot`.

Exploring the time, varying model

This is a time-invariant plot with our new function, to check that things are working correctly.

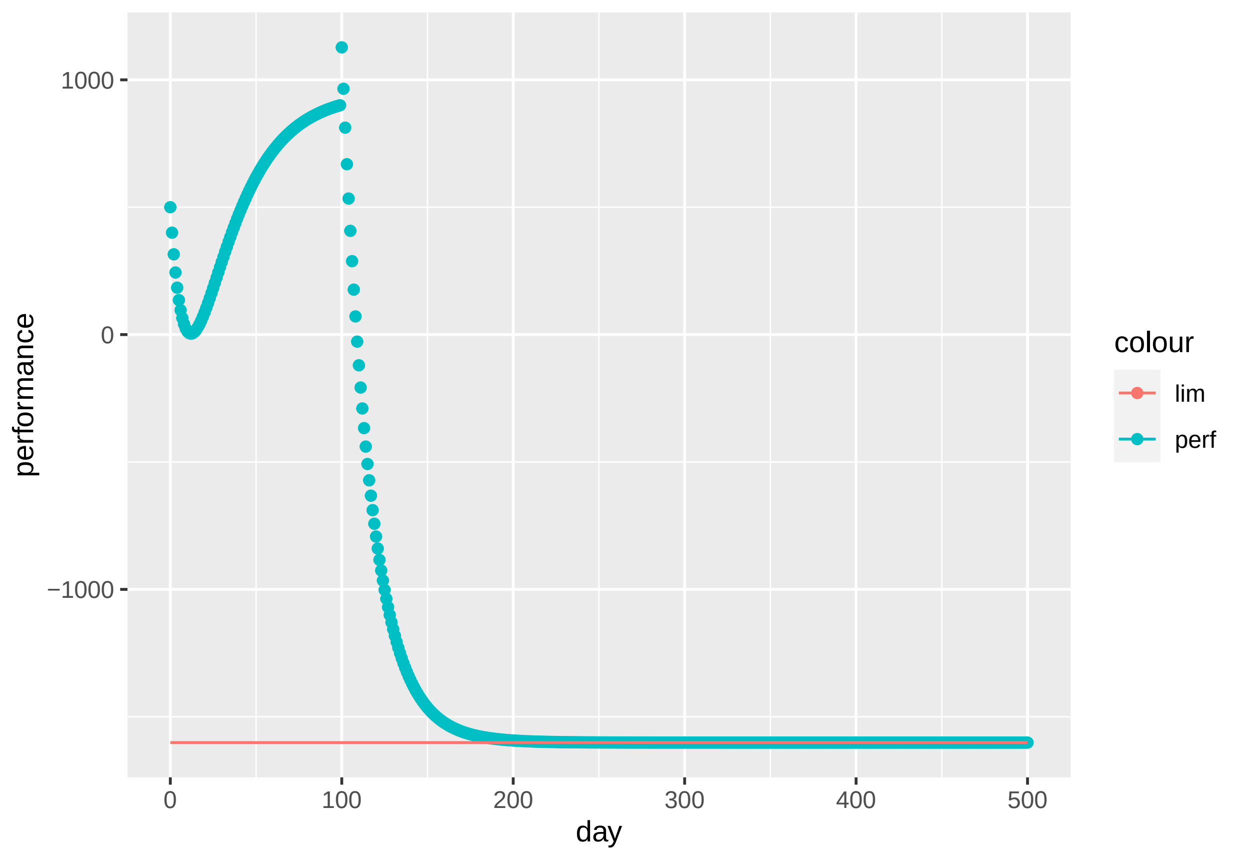
```
perf_plot(p_0 = 500,
          k_1 = 1,
          tau_1 = 25,
          k_2 = 2,
          tau_2 = 10,
          days = 500,
          training_stim = list("constant", 100)) +
perf_plot(p_0 = 500,
          k_1 = 2,
          tau_1 = 20,
          k_2 = 4,
          tau_2 = 15,
          days = 500,
          training_stim = list("constant", 100))
```





Addinng one change date

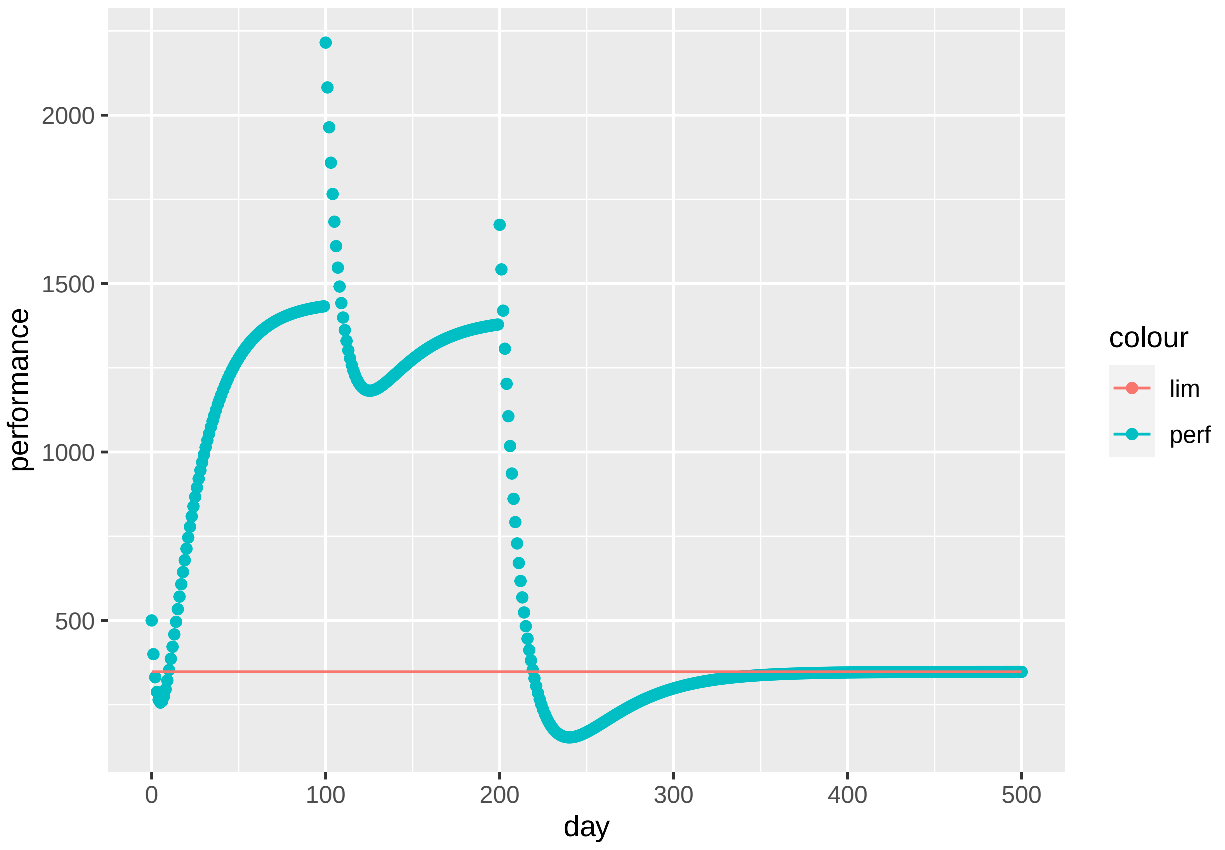
```
perf_plot(p_0 = 500,
          k_1 = c(1, 2),
          tau_1 = c(25, 20),
          k_2 = c(2, 4),
          tau_2 = c(10, 15) ,
          change_days = c(100),
          days = 500,
          training_stim = list("constant", 100))
```



Notice that this doesn't look like the first set of parameters for the first 100 days, and then the other distribution for the rest of the days. Even though it looks like a new curve at the change day, it is reflecting a change of the parameters.

Adding an additional change dage

```
perf_plot(p_0 = 500,
          k_1 = c(1,2,3),
          tau_1 = c(20, 25, 30),
          k_2 = c(2, 4, 6),
          tau_2 = c(5,10, 15),
          change_days = c(100, 200),
          days = 500,
          training_stim = list("constant", 100))
```



To have the effects of an initial negative effect, and a long-run positive effect to a constant stimulus, it seems to be necessary that $k_1 < k_2$ and τ_1 has to be much bigger than

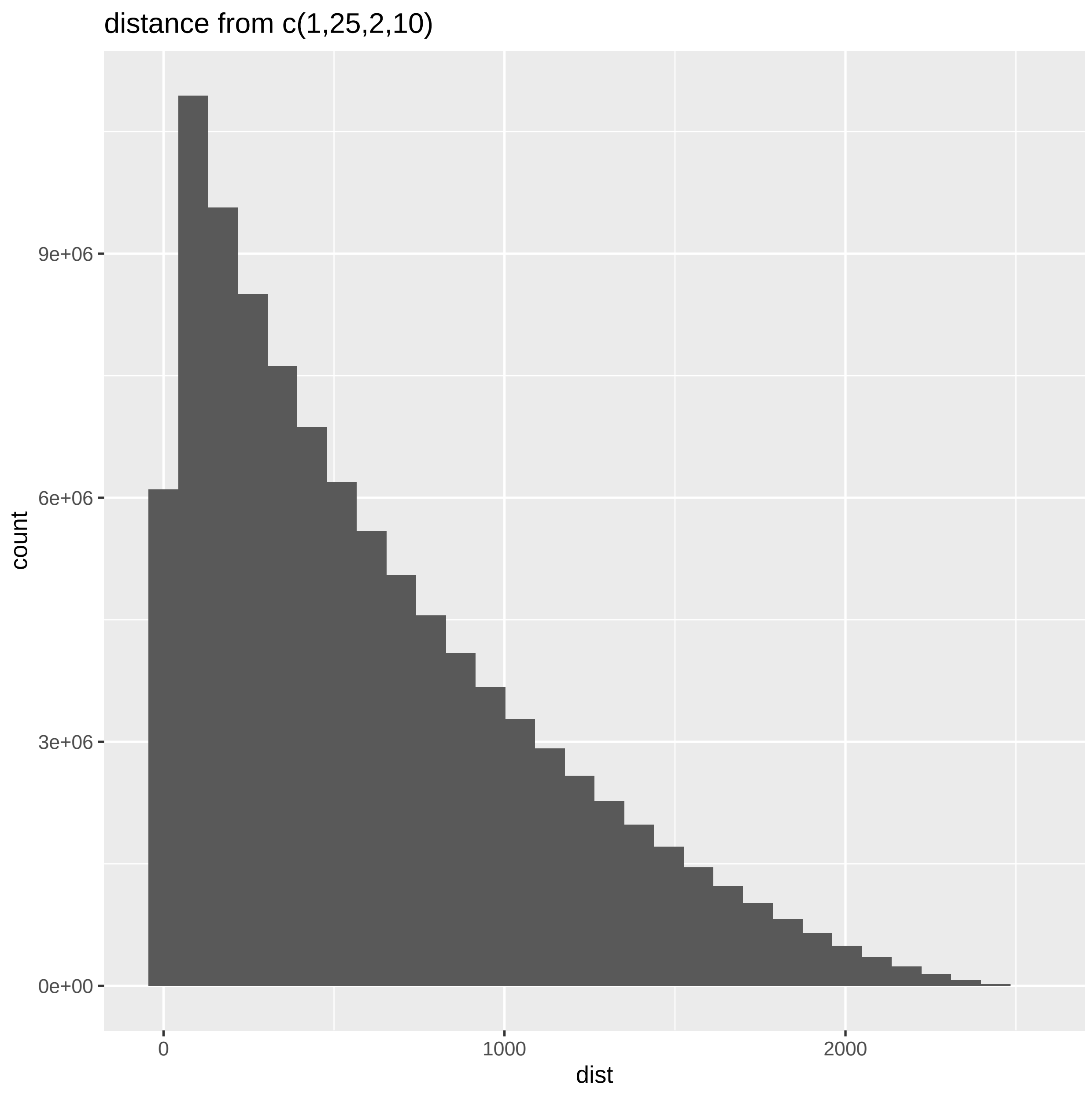
Lets try to use $\frac{k_1}{1-e^{-1/\tau_1}} - \frac{k_2}{1-e^{-1/\tau_2}}$ in the metric to evaluate the distance between parameter sets

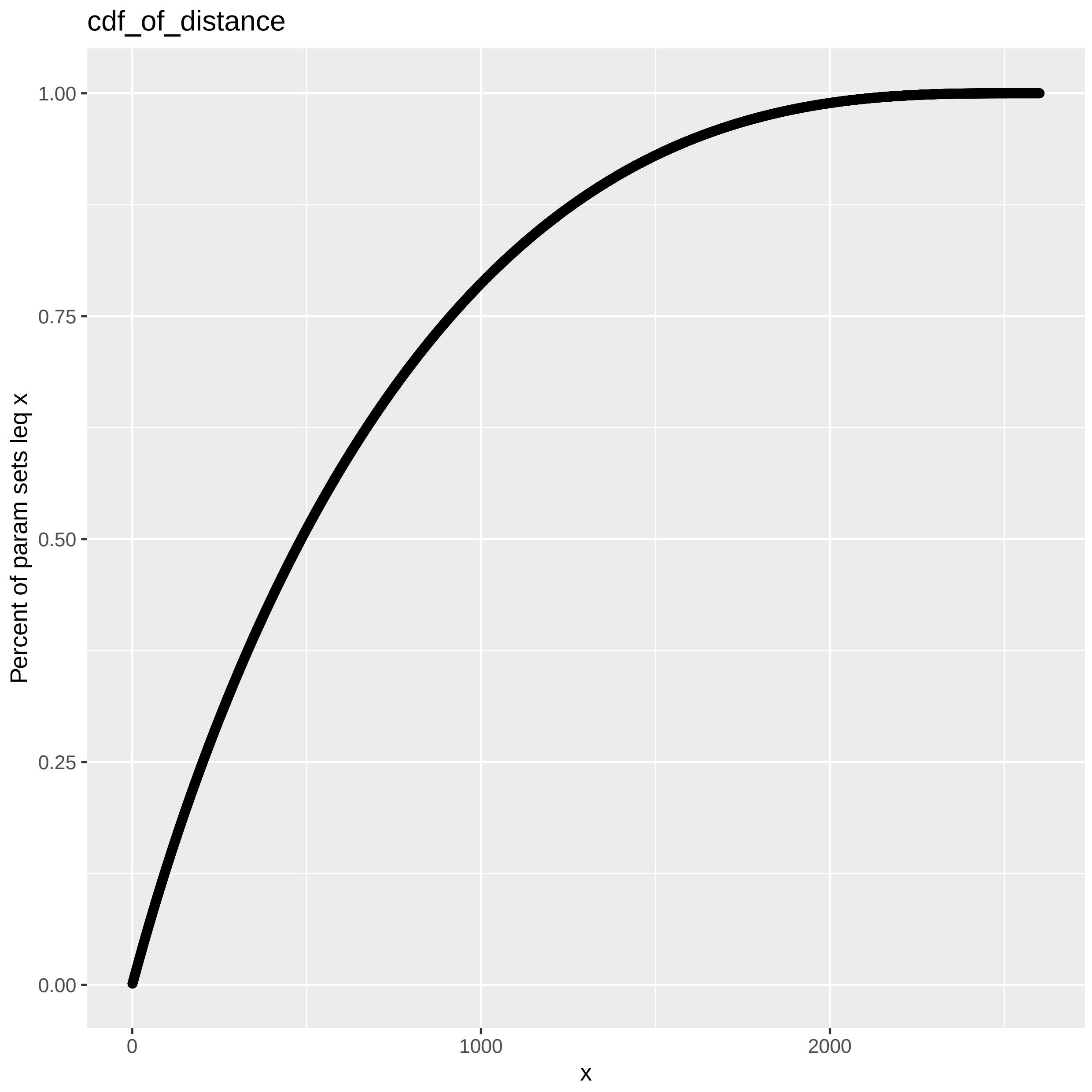
```
lim_func <- function(k_1, tau_1, k_2, tau_2) {
  k_1/(1-exp(-1/tau_1))-k_2/(1-exp(-1/tau_2))
}

params_dist <- function(params_1, params_2) {
  abs(lim_func(params_1[[1]], params_1[[2]], params_1[[3]], params_1[[4]])-
        lim_func(params_2[[1]], params_2[[2]], params_2[[3]], params_2[[4]])
      )
}

params_grid <- expand.grid(k_1 = seq(1,50, length.out=100),
            tau_1 = seq(1,50, length.out=100),
            k_2 = seq(1,50, length.out=100),
            tau_2 = seq(1,50, length.out=100)
            )
```

Applying the ''norm'' to figure out ''distance'' from a point

```r
params_matrix <- as.matrix(params_grid)
dist_vec <- c(rep(0, 100000000))
iter_fn <- function(i, params_matrix) {
  params_dist(params_matrix[i, ], c(1,25,2,10))
}

# parallel computing is a factor for this computation, took a few minuites
dist_vec <- parallel::mcmapply(
  iter_fn,
  i = c(1:100000000),
  MoreArgs = list(params_matrix = params_matrix),
  mc.cores = floor(.9 * parallel::detectCores())
)
# a 100,000,000 element, 100 MB vector
save(dist_vec, file = stringr::str_c(rprojroot::find_rstudio_root_file(),"/generated_data/dist_vec.RDat
# so we don't have to do the computation again

dist_tib <- tibble::tibble(
  "dist" = dist_vec
)

plot_hist <- ggplot(dist_tib, aes(x=dist)) +
  geom_histogram() +
  labs(title = "distance from c(1,25,2,10)" )
plot_hist


# so we can call the outputed plot in this markdownfile, to save time.
ggsave(filename = "dist from single set.pdf",
       plot_hist,
       path = stringr::str_c(rprojroot::find_rstudio_root_file(),"/plots"),
       device = "pdf")


# this took awhile
cdf_vec <- c(rep(0, 2601))
for (i in 1:2601) {
  cdf[[i]] <- length(which(dist_vec<=i))
}

cdf_data <- tibble(
  "x" = c(1:2601),
  "percent_leq_x" = cdf/100000000
)

# to save computation time later
save(cdf_vec, file = stringr::str_c(rprojroot::find_rstudio_root_file(),"/generated_data/cdf_vec.RData"

cdf_plot <- ggplot(cdf_data, aes(x=x, y= percent_leq_x)) +
  geom_point() +
  labs(x = "x",
       y = "Percent of param sets leq x",
       title = "cdf_of_distance ")
```

```r
cdf_plot

ggsave(filename = "cdf_plot.pdf",
       cdf_plot,
       path = stringr::str_c(rprojroot::find_rstudio_root_file(),"/plots"),
       device = "pdf")
```




cdf_of_distance
Percent of param sets leq x
1.00
0.75
0.50
0.25
0.00
0
1000
2000
x