Simulating Time-Varying Performance

Andrew Glover

2024-01-23

setup

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

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

```
## i Loading impulseResponse
##
## Attaching package: 'dplyr'
##
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Functions used for analysis

Generating the parameters matri

```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)
}
```

A note on recursive equations:

All of the time-invariant models here rely on the following recursion equations:

$$p(t) = p_0 + k_1 \cdot g(t) - k_2 \cdot h(t)$$

where $g(0) = h(0) = 0$ and

$$g(i) = e^{-1/\tau_1}\big(g(i-1) + w(i)\big), \quad h(i) = e^{-1/\tau_2}\big(g(i-1) + w(i)\big)$$

We want to have recursive equations that allow the parameters to change over time. Breaking down either the fitness or the fatigue term in the model, we get that

$$\text{Fitness}(t) = k_1 \sum_{i=0}^{t-1} e^{t-i} w(i) = k_1 e^{-1/\tau_1} w(t) + k_1 e^{-2/\tau_1} w(t-1) + \cdots + k_1 e^{t/\tau_1} w(0)$$

Lets say that at time t_{new}, k_1 changes to k_{new} and τ_1 changes to τ_{new}. We would expect that the new model looks something like

$$\text{Fitness}(t) = k_{new} e^{-1/\tau_{\text{new}}} w(t) + \cdots + k_{\text{new}} e^{-(t-t_{\text{new}})/\tau_{\text{new}}} w(t_{\text{new}}) + k_1 e^{-(t-t_{\text{new}})/\tau_{\text{new}} - 1/\tau_1} w(t_{\text{new}} - 1) + \cdots + k_1 e^{-(t-t_{\text{new}})/\tau_{\text{new}} - t_{\text{new}}/}$$

So, I propose a new model for performance:

$$p(t) = p_0 + \sum_{i=0}^{t-1} k_1^i \exp\left(-\sum_{j=i}^{t} (\tau_1^j)^{-1}\right) w(i) + \sum_{i=1}^{t-1} k_2^i \exp\left(-\sum_{j=i}^{t} (\tau_2^j)^{-1}\right)$$

where the superscript in the parameters indicates the parameter to be used for that day, i.e. k_1^i indicates the k_1 parameter on the ith day.

I made a naieve implementation of the time varying model, before I wrote the above model for performance down. The code below

```
mat_to_perf_old <- 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)
}
```

fails because it would instead calculates this

Fitness$(t) = k_{\text{new}}e^{-1/\tau_{\text{new}}}w(t) + \cdots + k_{\text{new}}e^{-(t-t_{\text{new}})/\tau_{\text{new}}}w(t_{\text{new}}) + k_{\text{new}}e^{-(t-t_{\text{new}})/\tau_{\text{new}}-1/\tau_1}w(t_{\text{new}}-1) + \cdots + k_{\text{new}}e^{-(t-t_{\text{new}})/\tau_{\text{new}}-}$

Insead of what we want. In works it goes back and changes the k_1 to k_{new} on the previous days before the model switched to k_new.

The old implementation relies on this recursive form of the model:

$$p(t) = p_0 + k_1^t g(t) - k_2^t h(t)$$

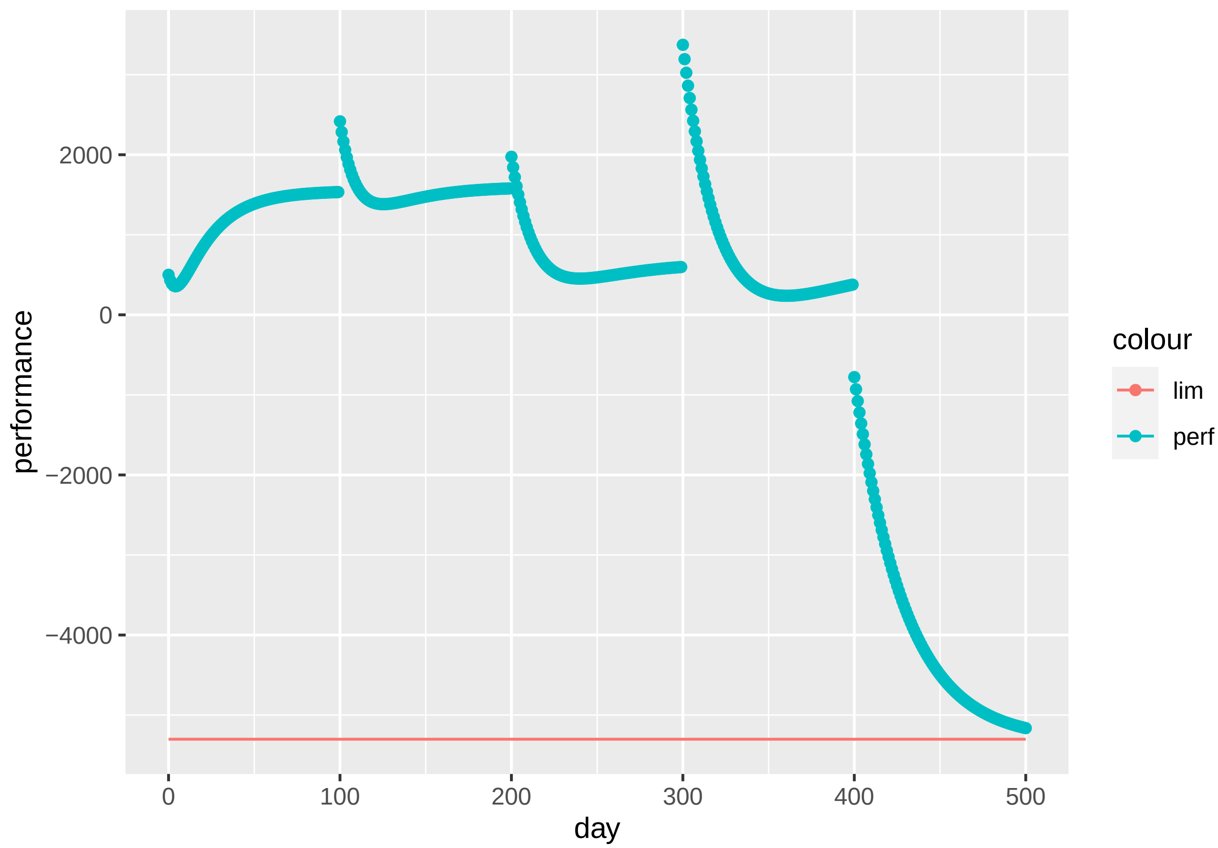
the recursive equations of $g(0) = h(0) = 0$ and

$$g(i) = e^{-1/\tau_1}(g(i-1) + w(i)) \quad h(i) = e^{-1/\tau_2}(h(i-1) + w(i))$$

What happens in practice with this old recursive equation is that the model jump very rapidly when the parameteters are changed.

If we plot the performance using this old way with the parameter sets

	1-100	101-200	201-300	301-400	401-500
k_1	1	2	3	5	6
k_2	20	25	30	40	40
tau_1	2	4	6	8	10
tau_2	5	10	15	25	30

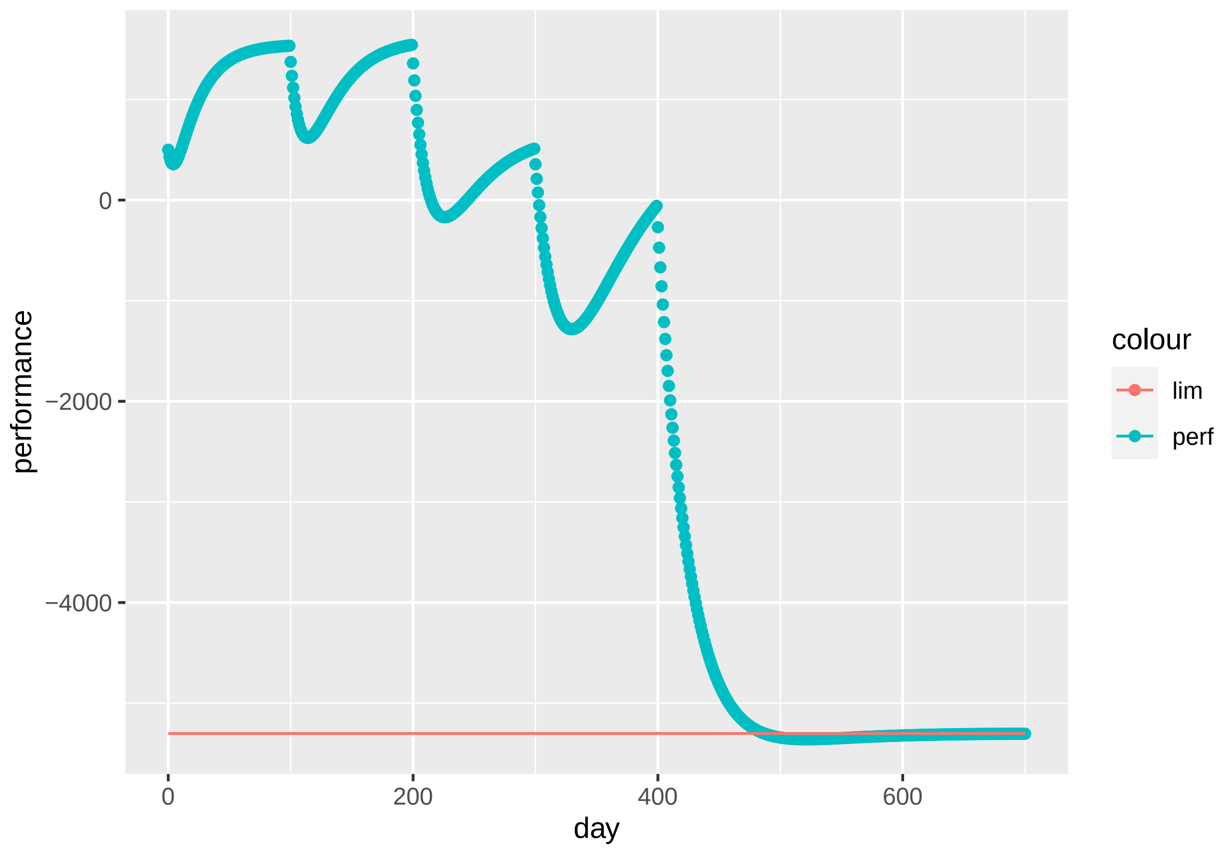


we will get
This method plot the performance in a discontinuous and unnatural way.

Improved model

```
mat_to_perf <- 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 + params_mat[i, "k_1"]*training_load[[i]])
    T_2 <- exp(-1/params_mat[i, "tau_2"])*(T_2 + params_mat[i, "k_2"]*training_load[[i]])
    perf_out[[i]] <- p_0 + T_1 - T_2
  }
  return(perf_out)
}
```

```
perf_tv(p_0 = 500,
          k_1 = c(1,2,3, 5, 6),
          tau_1 = c(20, 25, 30, 40, 40),
          k_2 = c(2, 4, 6, 8, 10),
          tau_2 = c(5,10, 15, 25, 30),
          change_days = c(100, 200, 300, 400),
          days = 700,
          training_stim = list("constant", 100))$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

$$p(t) = p_0 + k_1 \sum_{i=0}^{t-1} e^{\frac{t-i}{\tau_1}} w(i) + k_2 \sum_{i=0}^{t-1} e^{\frac{t-i}{\tau_2}} w(i)$$

Assume that $w(i) = C$ for all i. This takes the convention that p_0 happens on day 0. Note that

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

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,

$$\sum_{i=0}^{t-1} e^{\frac{t-i}{\tau_1}} = -1 + \frac{1}{1 - e^{-1/\tau_1}} = \frac{e^{-1/\tau_1}}{1 - e^{-1/\tau_1}}$$

Therefore

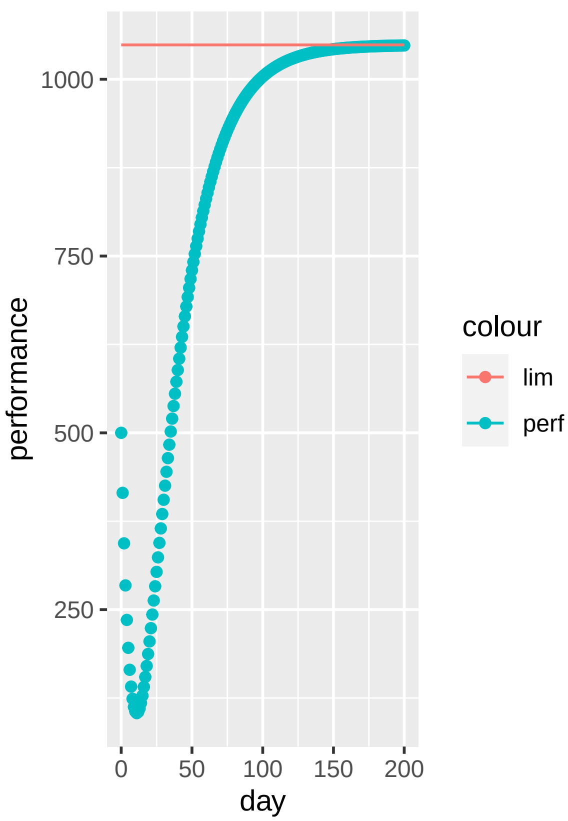
$$\lim_{t \to \infty} p(t) = p_0 + C \left(\frac{k_1 e^{-1/\tau_1}}{1 - e^{-1/\tau_1}} - \frac{k_2 e^{-1/\tau_1}}{1 - e^{-1/\tau_2}} \right)$$

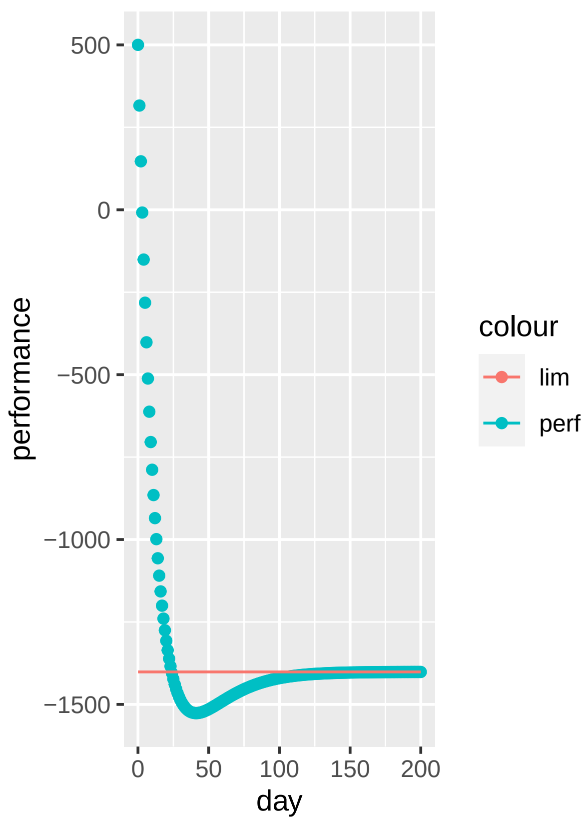
This is how we compute the red line in `perf_plot`.

Exploring the time, varying model

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

```
perf_tv(p_0 = 500,
        k_1 = 1,
        tau_1 = 25,
        k_2 = 2,
        tau_2 = 10,
        days = 200,
        training_stim = list("constant", 100))$plot +
perf_tv(p_0 = 500,
        k_1 = 2,
        tau_1 = 20,
        k_2 = 4,
        tau_2 = 15,
        days = 200,
        training_stim = list("constant", 100))$plot
```

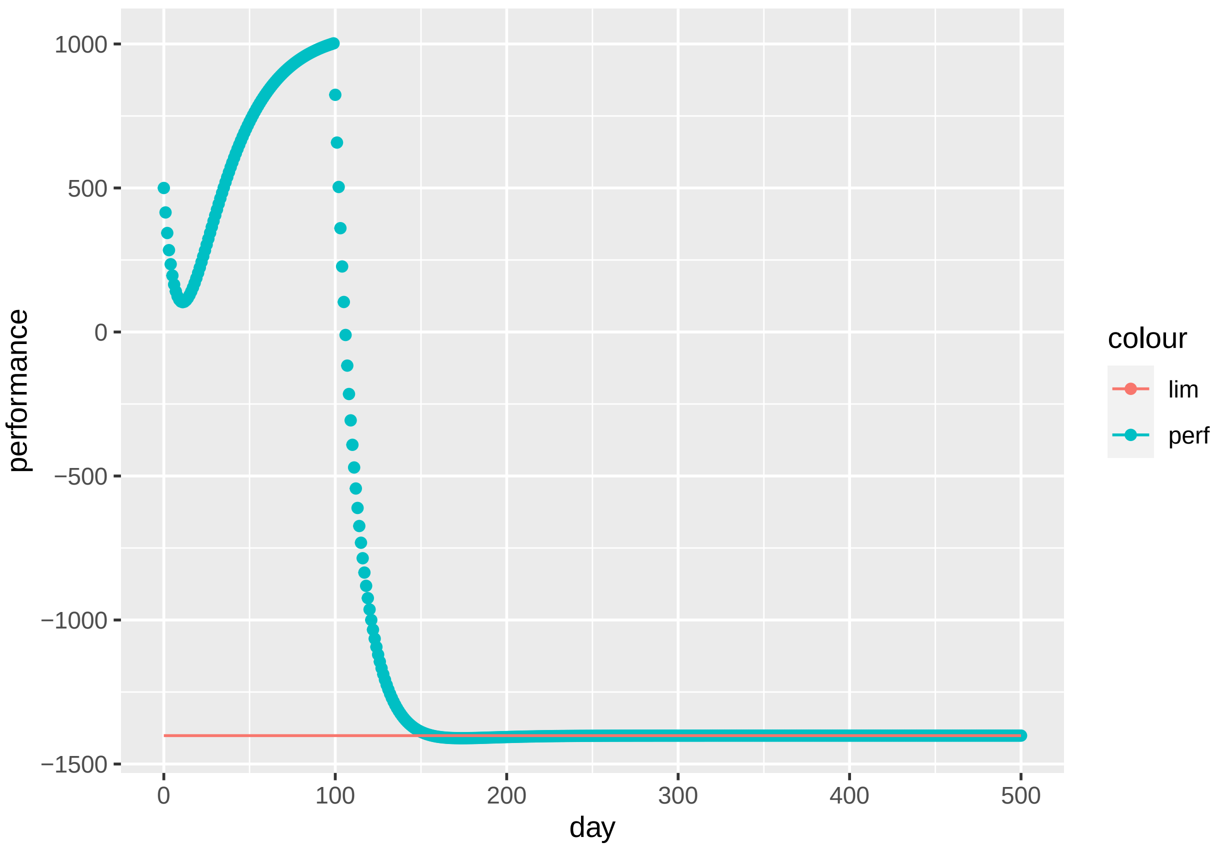




Addinng one change date

```
perf_tv(p_0 = 500,
        k_1 = c(1, 2),
        tau_1 = c(25, 20),
        k_2 = c(2, 4),
        tau_2 = c(10, 15) ,
```

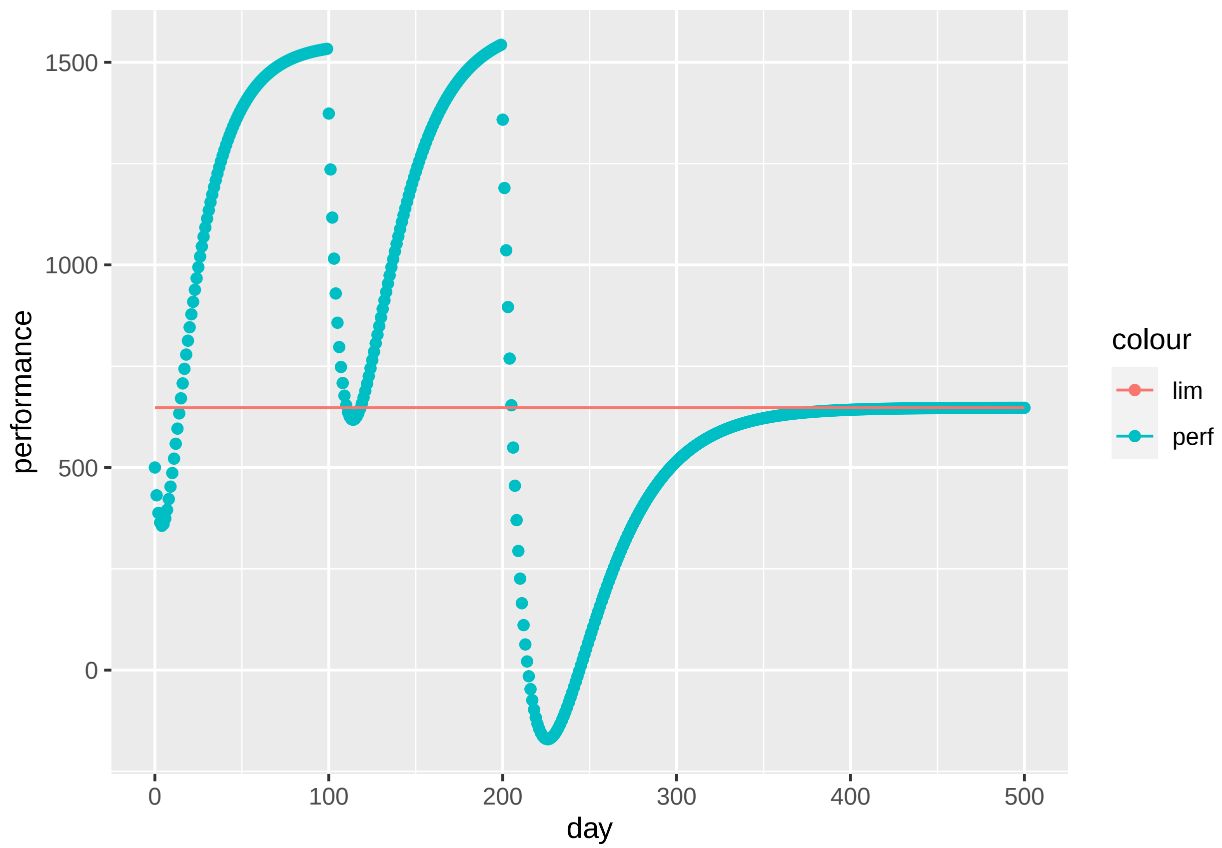
```
          change_days = c(100),
          days = 500,
          training_stim = list("constant", 100))$plot
```



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_tv(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))$plot
```



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

We can write

$$\text{dist}_{\text{perf}}(z, z') = |f(z) - f(z')|$$

where $f : (\mathbb{R}^2 \times \mathbb{R}_{>0})^2 \to \mathbb{R}$ is given by

$$z = (k_1, \tau_1, k_2, \tau_2) \mapsto \frac{k_1}{1 - e^{-1/\tau_1}} - \frac{k_2}{1 - e^{-1/\tau_2}}$$

$\text{dist}_{\text{perf}}$ is a metric, from the properties of the absolute value, except for the fact that the distance between two points has to be positive, since f is not injective.

Since

$$e^{n/x} \approx 1 - \frac{n}{x}$$

we have that

$$1 - e^{-1/x} \approx 1/x$$

Therefore,

$$f(z) = \frac{k_1}{1 - e^{-1/\tau_1}} - \frac{k_2}{1 - e^{-1/\tau_2}} \approx k_1\tau_1 - k_2\tau_2$$

I don't believe that this approximation is very good, but it provides a good intuition about this function.

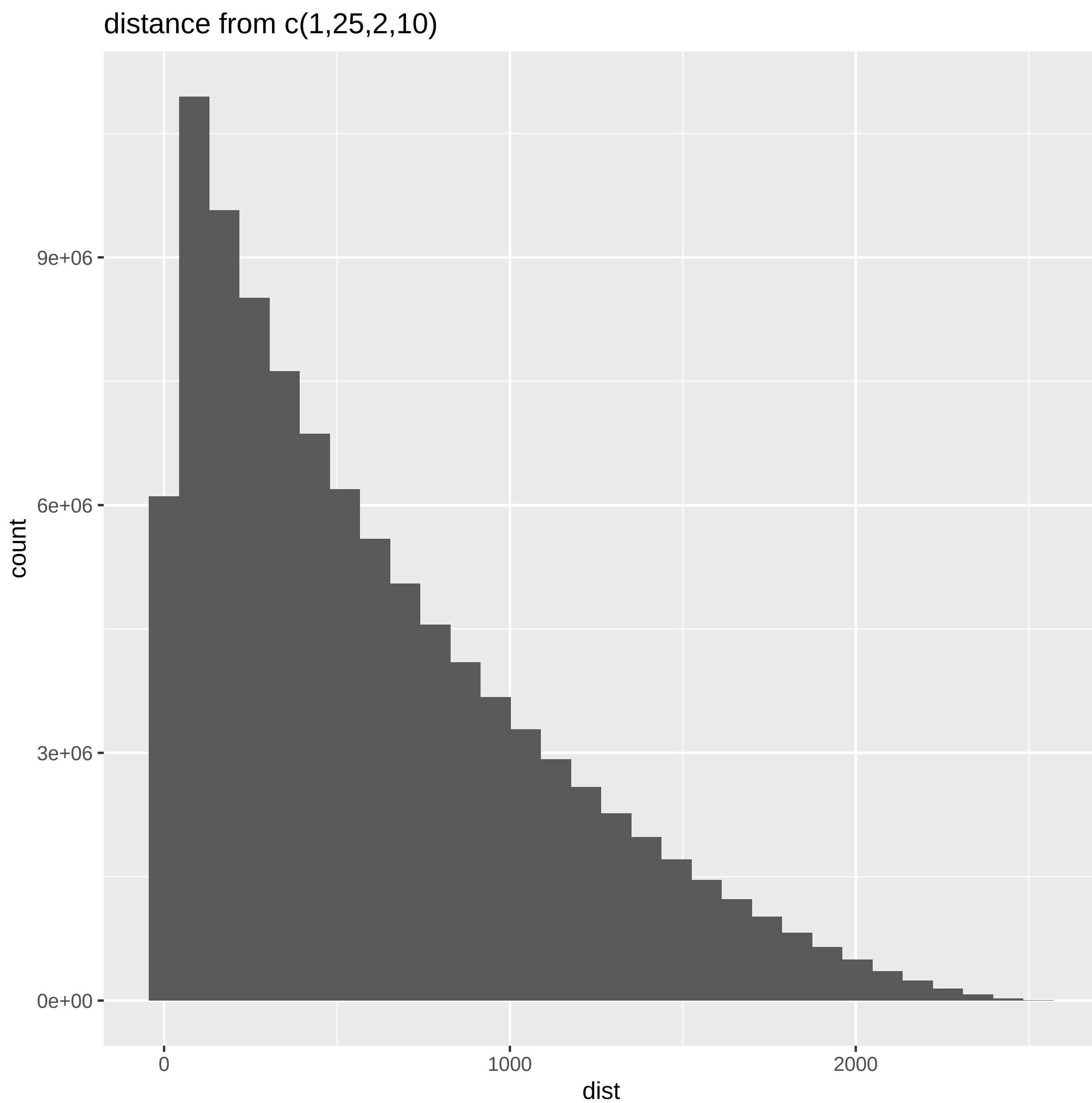
```r
lim_func <- function(k_1, tau_1, k_2, tau_2) {
  k_1/(1-exp(-1/tau_1))-k_2/(1-exp(-1/tau_2))
}

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]])
      )
}
```

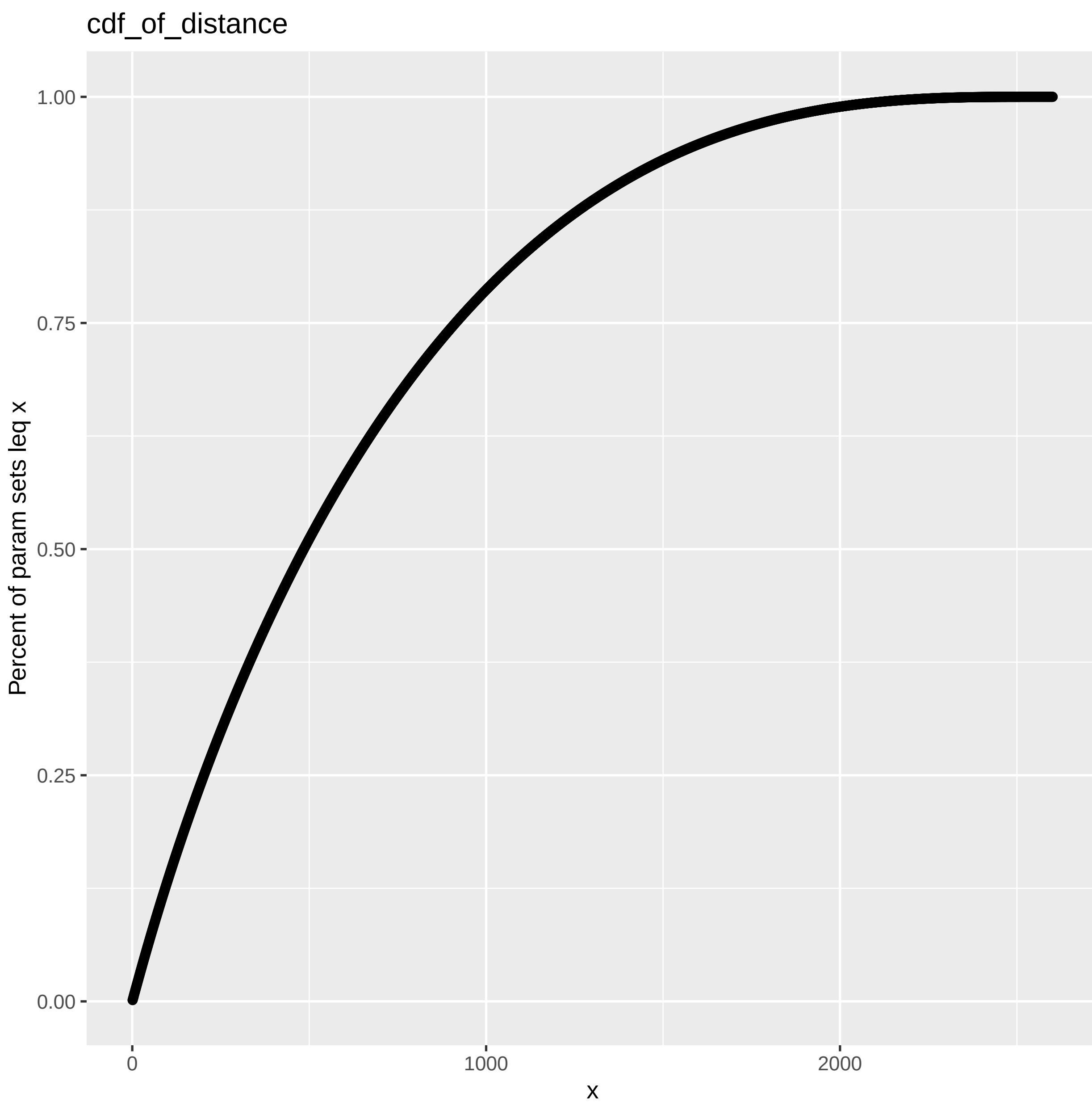
Applying the ''norm" to figure out ''distance" from a point

Don't run this chunk locally

```r
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)
            )

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.RData"))
# 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,
```

```r
       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 ")
cdf_plot

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


distance from c(1,25,2,10)
count
9e+06
6e+06
3e+06
0e+00
0
1000
2000
dist



Generating many performance curves with the same limit and plotting

```
set.seed(443)
C_1 <- round(lim_func(1,25,2,10), digits = 1)
n <- 100
vec_1<- runif(n, 5, 50)+C_1
k_1_same <- mapply(function(i) {runif(1, 1, vec_1[[i]])}, c(1:n))
tau_1_same <- -1/log(1-k_1_same/vec_1)

vec_2 <- vec_1-C_1
```

```r
k_2_same <- mapply(function(i) {runif(1, 1, vec_2[[i]])}, c(1:n))
tau_2_same <- -1/log(1-k_2_same/vec_2)


day <- 150
p_0 <- 500
training_load <- c(rep(100, day))
same_test_tib <- tibble(
  "day"  = c(0:day)
)

for (i in 1:n) {
  same_test_tib[, stringr::str_c("p", i)] <- c(p_0,
                                               perf_tv(
                                                 p_0 = p_0,
                                                   k_1 = k_1_same[[i]],
                                                   tau_1 = tau_1_same[[i]],
                                                   k_2 = k_2_same[[i]],
                                                   tau_2 = tau_2_same[[i]],
                                                   days = day,
                                                 training_stim = list("constant", 100)
                                               ))
}


pal_color <- scales::hue_pal()(n)
names(pal_color) <- names(same_test_tib[2:n+1])

cols <- as.list(names(same_test_tib[2:n+1]))
min_or_max_vec <- c(rep(0,n))
for (i in 1:n) {
  if (round(max(same_test_tib[, i+1]), digits = 1)<=950){
    min_or_max_vec[[i]] <- min(same_test_tib[, i+1])
  }
  else {
    min_or_max_vec[[i]] <- max(same_test_tib[, i+1])
  }
}

# reordering the columns so the plot looks nice
ord_min_or_max_vec <- sort(min_or_max_vec)
ord_min_or_max_vec
permutation_index <- c(rep(NA, n))
for (i in 1:n) {
  value <- min_or_max_vec[[i]]
  output_index <- which(ord_min_or_max_vec == value)
  while(is.na(permutation_index[[i]])) {
    if (output_index[[1]] %in% permutation_index){
      output_index <- output_index[-1]
    }
    else {
      permutation_index[[i]] <- output_index[[1]]
    }
```

```
  }
}


same_test_tib_new <- same_test_tib
for (i in 1:n) {
  same_test_tib_new[, permutation_index[[i]] + 1] <- same_test_tib[,i  + 1]
}


plot_same <- ggplot(data = same_test_tib_new, aes(x = day)) +
    lapply(cols, function(x) {
      geom_line(aes(y = .data[[x]], color = x))
    }) +
    scale_color_manual(values = pal_color) +
  labs(x = "Day",
       y = "Simulated Performance",
       title = "Simulated Performacne of Parameter Sets with Same Limit") +
  theme(legend.position = "none")

plot_same
```

The colors were sorted by the maximum value. It seems that the maximum value does not determine a convergence rate; there are some curves that peak late, but this generally doesn't happen.

Even though our metric is not truly a metric, it will probably become one when also restrict our curves with minimizing the sum of squared error.

```
p_0=500
k_1 <- seq(1, 50, length.out = 50)
k_2 <- seq(1, 50, length.out = 50)
tau_1 <- seq(1, 50, length.out = 50)
tau_2 <- seq(1, 50, length.out = 50)
days <- 100
training_load_1 <- c(rep(100, days))
params_grid_1 <- as_tibble(expand.grid(k_1, tau_1, k_2, tau_2))
output_matrix <- params_grid_1
output_matrix$performance
```

```
## Warning: Unknown or uninitialised column: `performance`.

## NULL
```

```
for (i in nrow(params_grid_1)){
  invariant_perf(params = c(p_0, params_grid_1[i, ]),
                 training_load = training_load_1)
}
```