Simulating Time-Varying Performance

Andrew Glover

2023-10-24

setup

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

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

```
## i Loading impulseResponse
```

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
#'
#' @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

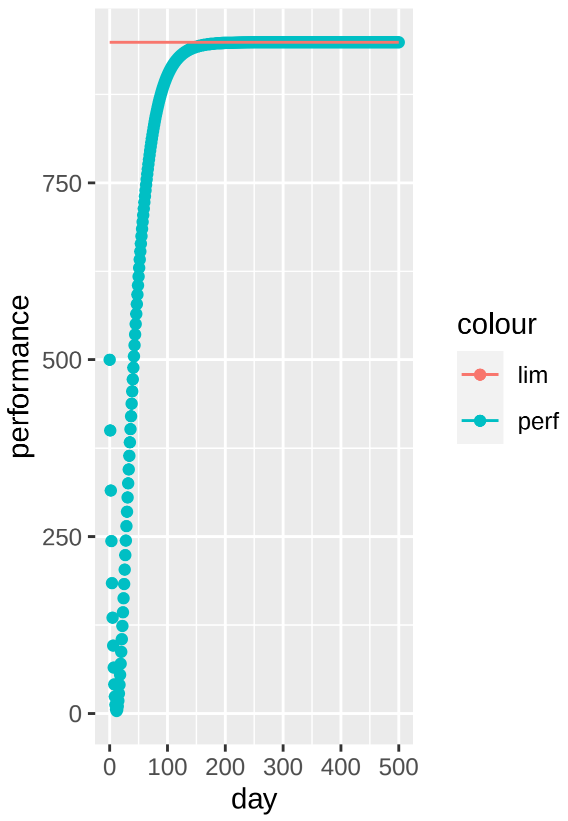
$$\lim_{t \to \infty} p(t) = p_0 + C \left(\frac{k_1}{1 - e^{-1/\tau_1}} - \frac{k_2}{1 - e^{-1/\tau_2}} \right)$$

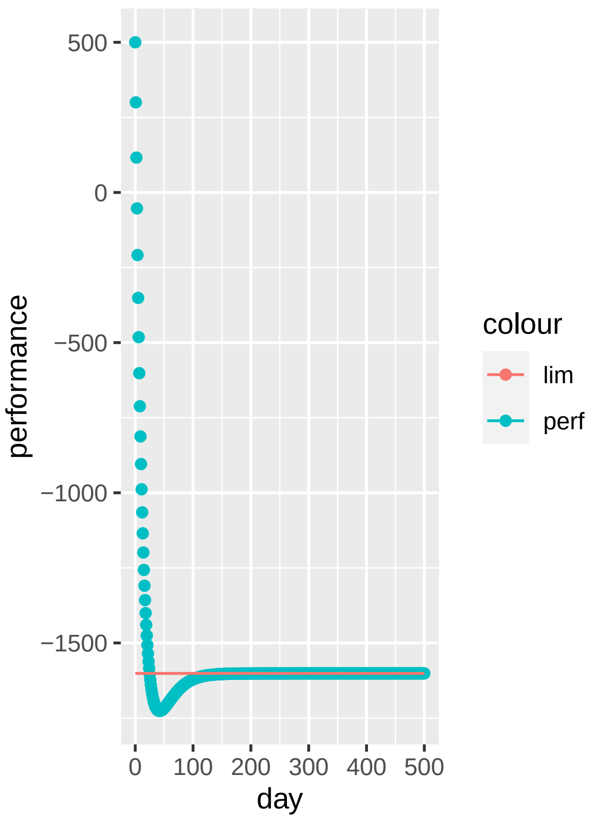
This is how we compute the red line in `perf_plot`.

Exploring the time, varying model

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

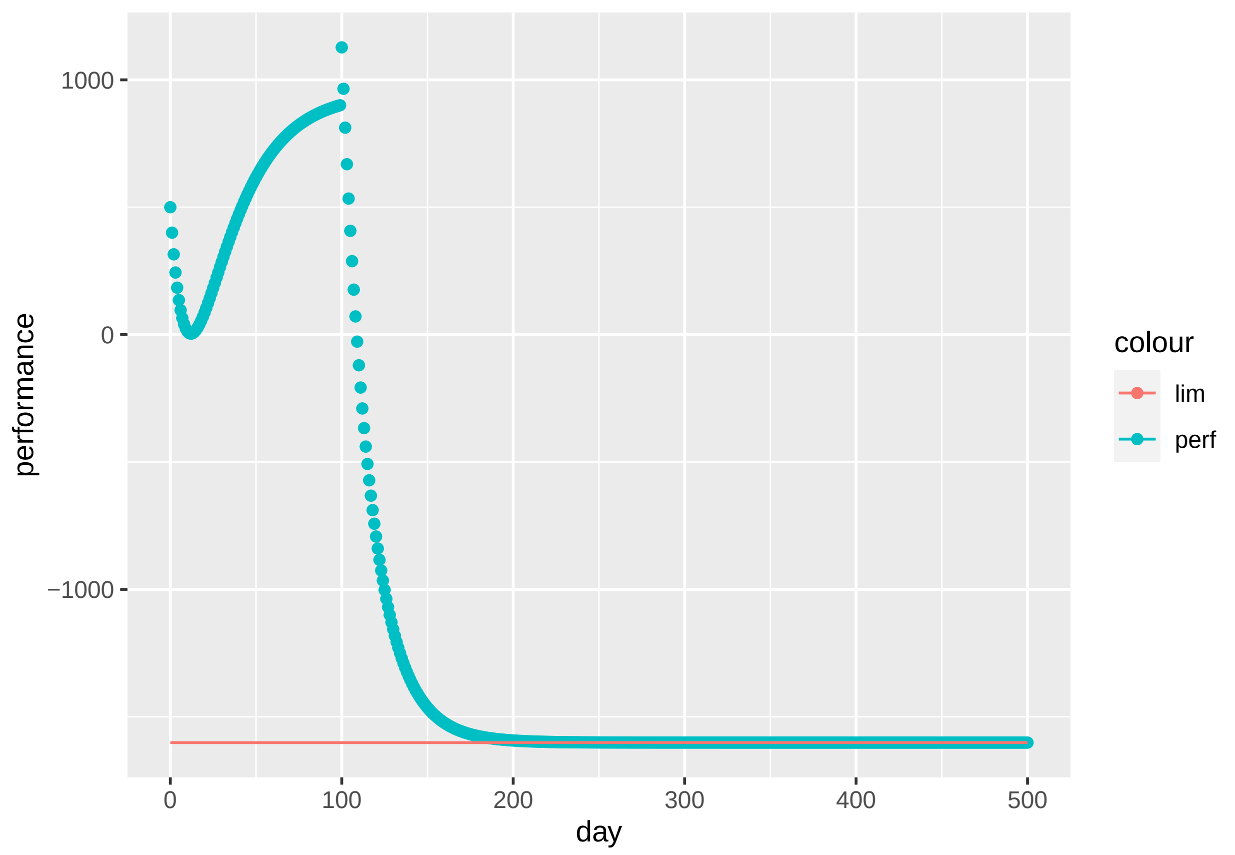
```
perf_plot(p_0 = 500,
          k_1 = 1,
          tau_1 = 25,
          k_2 = 2,
          tau_2 = 10,
          days = 500,
          training_stim = list("constant", 100)) +
perf_plot(p_0 = 500,
          k_1 = 2,
          tau_1 = 20,
          k_2 = 4,
          tau_2 = 15,
          days = 500,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





Addinng one change date

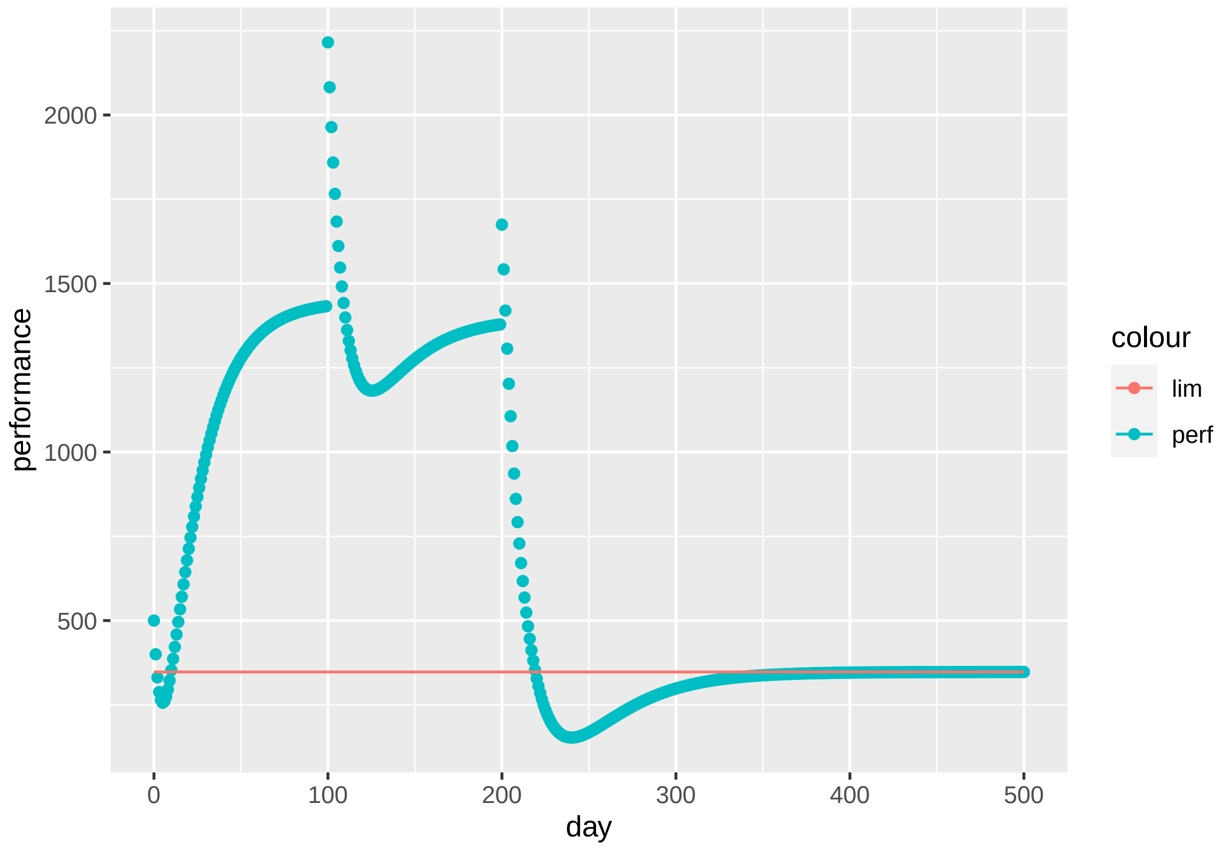
```
perf_plot(p_0 = 500,
          k_1 = c(1, 2),
          tau_1 = c(25, 20),
          k_2 = c(2, 4),
          tau_2 = c(10, 15) ,
          change_days = c(100),
          days = 500,
          training_stim = list("constant", 100))
```



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

```r
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

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,

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

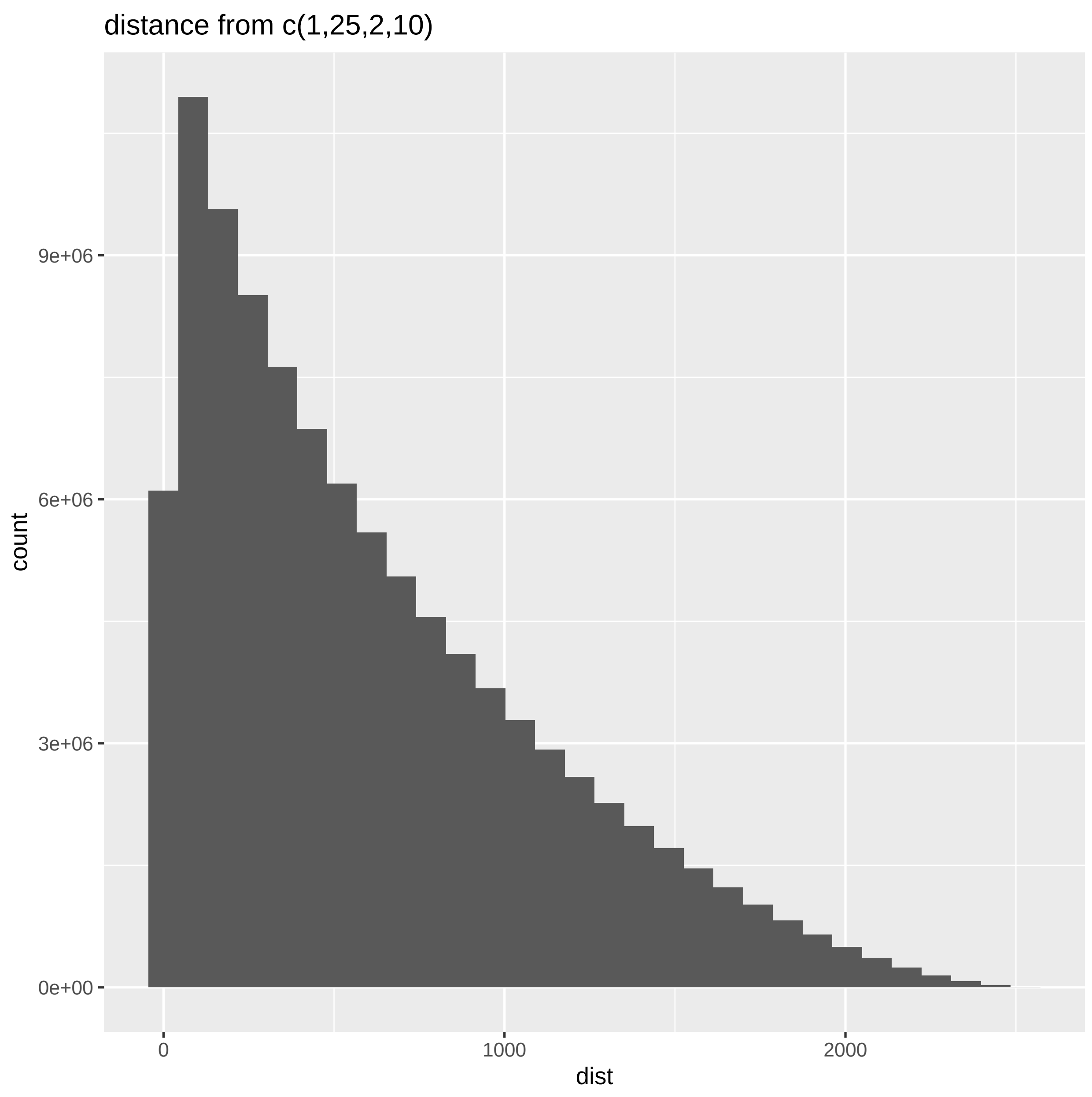
I don't believe that this approximation is very good, but it provides a good intuition about this function.

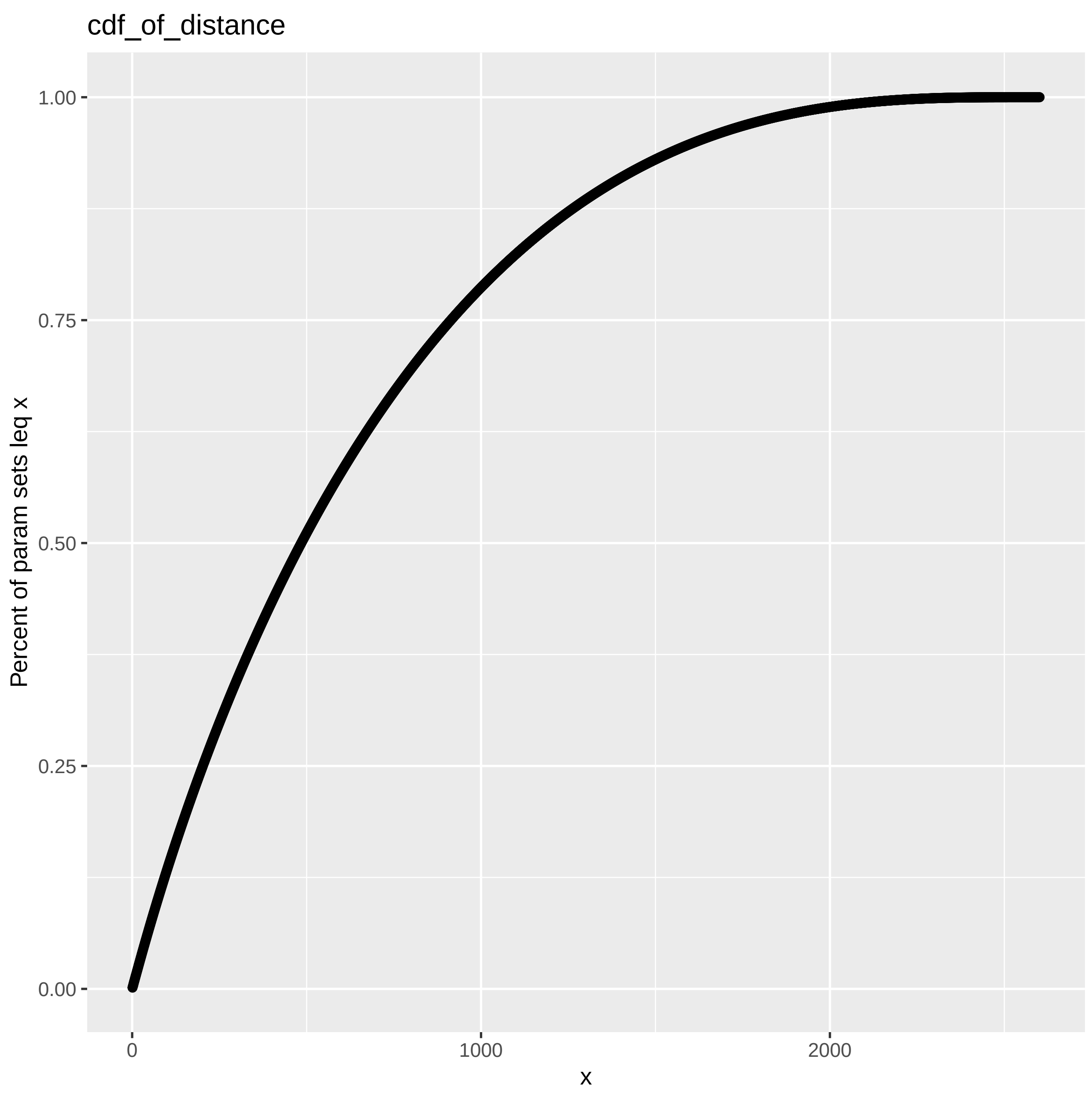
```r
lim_func <- function(k_1, tau_1, k_2, tau_2) {
  k_1/(1-exp(-1/tau_1))-k_2/(1-exp(-1/tau_2))
}

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,
       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



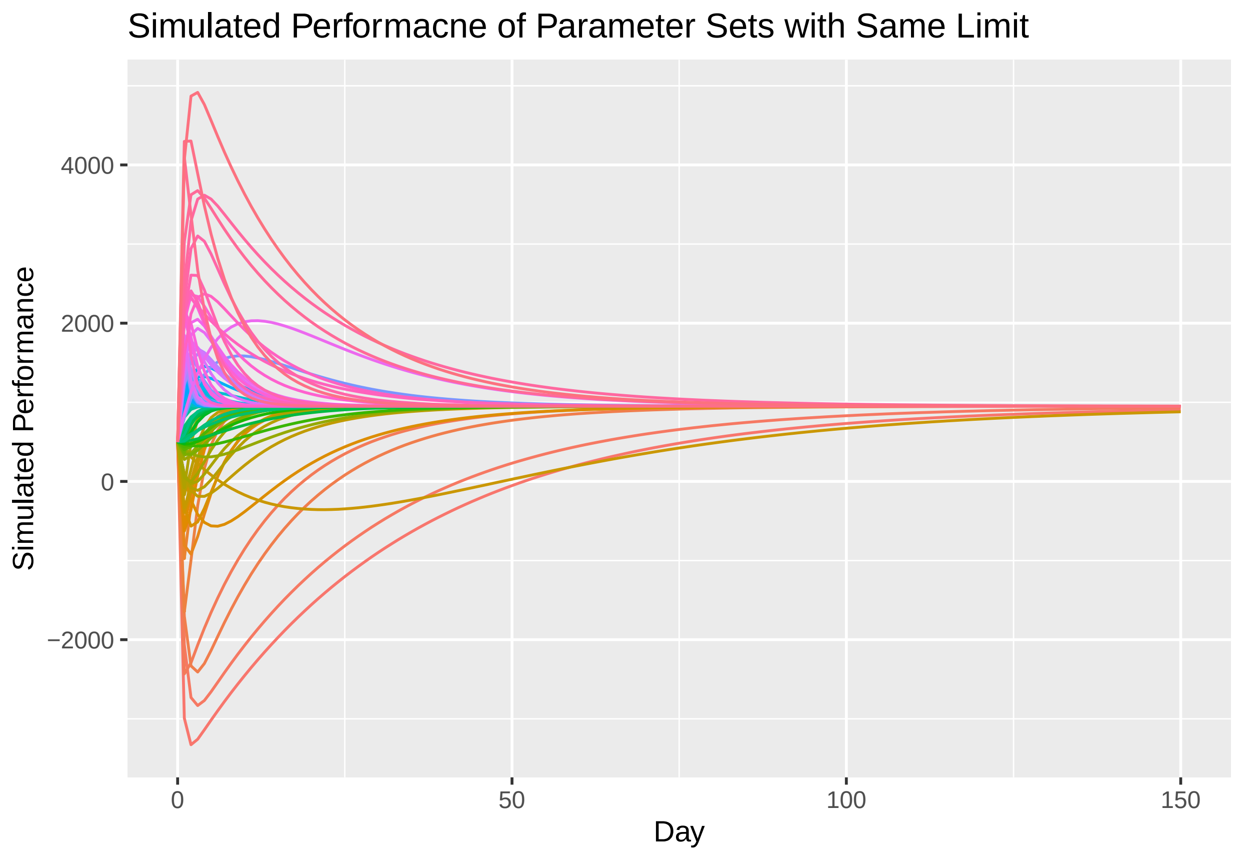
Generating many performance curves with the same limit and plotting

```
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,
                                                 params_mat = params_mat(
                                                   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_load = training_load
                                               ))
}


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
```

```
##   [1] -3602.189461 -3327.611469 -2832.581326 -2432.440204 -2408.966002
##   [6] -1648.104660  -975.945776  -918.660903  -626.968981  -572.658665
##  [11]  -565.992065  -565.048350  -392.771165  -389.042347  -356.617522
##  [16]  -335.648616  -188.585342  -170.469789  -113.114856   -81.874936
##  [21]   -28.535047    -3.657609   277.377738   309.045056   332.437935
##  [26]   332.500656   340.168229   365.358637   398.443765   403.399607
##  [31]   434.195087   446.938957   471.477564   480.863142   500.000000
##  [36]   500.000000   500.000000   500.000000   500.000000   500.000000
## [41]    500.000000   500.000000   500.000000   950.139202   950.904572
## [46]    954.043203   979.344754  1010.548716  1011.162789  1065.364592
## [51]   1091.073545  1104.515909  1126.229670  1129.689630  1160.708664
## [56]   1174.448884  1175.673531  1257.279924  1257.542711  1265.364346
## [61]   1301.476567  1326.798975  1339.303557  1368.451567  1371.699938
## [66]   1454.878860  1501.289621  1528.227053  1542.571439  1586.709490
## [71]   1587.609994  1593.589464  1609.753998  1652.418754  1661.571624
## [76]   1677.399624  1682.020849  1684.326918  1759.961127  1781.524675
## [81]   1848.557754  1934.673381  2031.722969  2049.825880  2076.976777
## [86]   2136.624471  2146.203990  2328.957673  2358.956663  2362.264050
## [91]   2371.730776  2396.775787  2408.549590  2607.199206  3102.314686
## [96]   3617.734852  3675.639829  4100.450575  4302.409166  4916.749802
```

```r
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)
}
```