A report generated from a pure R script

2025-03-20

First we generate the parameters for simulation:

```
# Load libraries
library("tictoc", lib = "/mnt/atgc-d3/sur/modules/pkgs/tidyverse_mrc")
library(tidyverse, lib.loc = "/mnt/atgc-d3/sur/modules/pkgs/tidyverse_mrc")
library(tidyr)
# library(tidyverse)

tictoc::tic("Section 0: Total running time")
tictoc::tic("Section 1: Time for Parameter Generation:")

# ==== Generate parameters ====
# Generate experiment name
generate_id <- function() {
  day <- format(Sys.Date(), "%d")
  month <- format(Sys.Date(), "%m")

  char_string <- paste0(sample(c(LETTERS, 0:9), 4, replace = TRUE), collapse = "")
  paste0("Exp_", char_string, "-D", day, "M", month)
}

wd <- "/mnt/atgc-d3/sur/users/mrivera/glv-research"
exp_id <- generate_id()
params_path <- file.path(wd, "Data", paste0(exp_id, ".tsv"))
```

Generate grid of parameters:

```
params_to_sim <- expand_grid(n_species = rep(c(20, 40, 100), times = 100), p_neg = 1, p_noint = seq(0,
  mutate(id = ids::random_id(n = length(n_species), bytes = 4)) %>%
  mutate(x0_seed = as.vector(sample(1:1e6, length(n_species), replace = FALSE))) %>%
  mutate(mu_seed = as.vector(sample(1:1e6, length(n_species),replace = FALSE))) %>%
  mutate(A_seed = as.vector(sample(1:1e6, length(n_species), replace = FALSE)))

# Verify if ids are unique and in case they are, save the parameters.
if (nrow(params_to_sim) == length(unique(params_to_sim$id))) {
  # Save Parameters as TSV
  data.table::fwrite(params_to_sim, exp_id, sep = "\t")
  message("\nParameteres generated and saved...\n")
} else {
  message("\nAn error ocurred. IDs are not unique\n")
}
tictoc::toc()
```

We split the data into chunks of `n_cores`:

```r
# ==== Split data into chunks====
tictoc::tic("Section 2: Divide data into chunks")
num_cores <- parallel::detectCores() - 1  # Use one less than the total number of cores
cat("The number of cores that will be used are: ", num_cores, "\n")

split_table <- function(df, n_chunks) {
  split(df, cut(seq_len(nrow(df)), breaks = n_chunks, labels = FALSE))
}

chunks <- split_table(params_to_sim, num_cores)
message("\nData split completed...\n")
tictoc::toc() # For section 2
```

We create a separate directory for each core to prevent race conditions (when two cores access the same directory simultaneously).

```r
# ==== Generate directories for each core ====
tictoc::tic("Section 3: Generate directories for each core")

# Generate workers directories
ode_dir <- file.path(wd, "Results", exp_id, "Parallel")

# Function to create main and worker directories
create_dirs <- function(main_dir, num_cores) {
  if (!dir.exists(main_dir)) dir.create(main_dir, recursive = TRUE)

  # Create worker directories
  worker_dirs <- file.path(main_dir, paste0("worker_", seq_len(num_cores)))
  invisible(lapply(worker_dirs, dir.create, showWarnings = FALSE))

  return(worker_dirs)
}
# Create directories
workers_ODE <- create_dirs(ode_dir, num_cores)
message("\nWorking directories created at path:\n", ode_dir,"\n")
tictoc::toc() # For section 3
```

We source the function to generate the gLV parameters using the initial parameters and for solve it:

```r
# Source function to regenerate parameters
source("/mnt/atgc-d3/sur/users/mrivera/glv-research/GIT-gLV/D20M03-tmpdir/gLV-Params.r")
print(regenerate)
```

```
## function (index)
## {
##     n_species <- as.numeric(index[["n_species"]])
##     set.seed(as.numeric(index[["x0_seed"]]))
##     x0 <- stats::runif(n_species, min = 0.1, max = 1)
##     set.seed(as.numeric(index[["mu_seed"]]))
##     mu <- stats::runif(n_species, min = 0.001, max = 1)
##     M <- matrix(NA, nrow = n_species, ncol = n_species)
##     diag(M) <- -0.5
##     p_noint <- as.numeric(index[["p_noint"]])
```

```
##     p_neg <- as.numeric(index[["p_neg"]])
##     num_off_diag <- n_species * (n_species - 1)
##     num_noint <- floor(p_noint * num_off_diag)
##     num_negs <- floor(p_neg * (num_off_diag - num_noint))
##     num_pos <- num_off_diag - (num_noint + num_negs)
##     set.seed(as.numeric(index[["A_seed"]]))
##     interaction_values <- c(rep(0, num_noint), -runif(num_negs,
##         min = 0, max = 1), runif(num_pos, min = 0, max = 1))
##     interaction_values <- sample(interaction_values)
##     M[upper.tri(M, diag = FALSE) | lower.tri(M, diag = FALSE)] <- interaction_values
##     M <- round(M, digits = 5)
##     id <- index[["id"]]
##     params <- list(x0 = x0, M = M, mu = mu, id = id, n = n_species)
##     return(params)
## }
```

```
# Source function to solve gLV equations
source("/mnt/atgc-d3/sur/users/mrivera/glv-research/GIT-gLV/D20M03-tmpdir/solve-gLV.r")
print(solve_gLV)
```

```
## function (times, params)
## {
##     glv_model <- function(t, x0, params) {
##         r <- params$mu
##         A <- params$M
##         dx <- x0 * (r + A %*% x0)
##         list(dx)
##     }
##     time_seq <- seq(1, times, by = 1)
##     results <- tryCatch(R.utils::withTimeout(deSolve::ode(y = params$x0,
##         times = time_seq, func = glv_model, parms = params, method = "ode45",
##         rtol = 1e-06, atol = 1e-06), timeout = 600), error = function(e) {
##         message(">> Simulation failed... skipping")
##         return(NULL)
##     })
##     if (!is.null(results) && ncol(results) > 1) {
##         return(t(results[, -1]))
##     }
##     else {
##         return(matrix(NA, nrow = nrow(params$M), ncol = times))
##     }
## }
```

We wrap the code for parallelizing the simulations.

```
# ==== Wrapper for running all required steps ====
parallel.sims <- function(index, path_ODE) {

  # Generate parameters
  params <- regenerate(index)

  # Run simulation
  output_ode <- solve_gLV(times = 700, params)
```

```
  # Define paths
  save_ode <- file.path(path_ODE, paste0("O_", params$id, ".tsv"))

  # Calculate NAs
  NA_count <- sum(is.na(output_ode))

  # Save simulation
  utils::write.table(output_ode, file = save_ode, sep = "\t", row.names = FALSE, col.names = TRUE)

  return(c(id = id, NA_count = NA_count))
}
```

We Parallelize the code and get the `NA counting`.

```
# ==== Parallelize it ====
tictoc::tic("Section 4: Run simulations using the parallel package")

NAs_vecs <- parallel::mclapply(1:num_cores, function(core_id) {

  message("Starting worker ", core_id, "....\n")

  core_chunk <- chunks[[core_id]]  # rows assigned to this core

  na.vec <- lapply(1:nrow(core_chunk), function(i) {
    parallel.sims(core_chunk[i, ],
    path_ODE = workers_ODE[core_id])
  })

  message("Ending worker ", core_id, "....\n")

  return(na.vec)

}, mc.cores = num_cores)
tictoc::toc() # For section 4


# ==== Get NAs number on simulations====
tictoc::tic("Section 5: Count total number of NAs")
NAs_counts <- unlist(NAs_vecs)
counts_df <- as.data.frame(matrix(NAs_counts, ncol = 2, byrow = TRUE))
colnames(counts_df) <- unique(names(NAs_counts))

# Save Parameters as TSV
save_path <- file.path(wd, "Nas-counting.tsv")
data.table::fwrite(x = counts_df, file = save_path, sep = "\t")
tictoc::toc() # For section 5
```

We create symbolic links of the simulation...

```
# ==== Create symbolic links====
tictoc::tic("Section 6: Generate symbolic links")

source("/mnt/atgc-d3/sur/users/mrivera/glv-research/GIT-gLV/Forge_symlinks.R")
```

```r
# Define source and target directories
source_path <- file.path(wd, "Simulate_ODE")
target_path <- file.path(wd, "Unified")
generate_symlinks(source_path = source_path, target_path = target_path)

tictoc::toc() # For section 6
tictoc::toc() # For Total running time
```