LPJmL Runner

Jannes Breier

2023-01-10

LPJmL Runner is the collective term for a set of functions within `lpjmlkit` that have the goal to simplify the execution of simulations with LPJmL and furthermore to execute complex, nested and multiple simulation sequences fast and error free without having a big (bash) script overhead.

Setup

Please make sure to have set the working environment for LPJmL correctly if you are not working on the PIK cluster (with Slurm Workload Manager). On the PIK cluster please load the lpjml module (below) or add it to your `".profile"`.

```
module load lpjml
```

Overview

1. First of all you have to think about what LPJmL parameters/settings (here all referred to as parameters) you want to change. You are free to change them directly in the corresponding `"js"` file or in the `tibble` (see example). For a single simulation this is a matter of personal routine, but when it comes to multiple runs where these parameters differ from each other, you have to specify them in a `tibble`.
   `?write_config` *for more information.*

```
my_params <- tibble(
  sim_name = c("scenario1", "scenario2"),
  random_seed = c(42, 404),
  param.k_temp = c(NA, 0.03),
  new_phenology = c(TRUE, FALSE)
)
```

2. Now the central function here is `write_config`, it creates and writes the LPJmL Configuration (Config) file(s) `"config_*.json"` from a table (`tibble`) with the parameters of a base `"lpjml.js"` file to be changed.
   `?write_config` *for more information.*

```
config_details <- write_config(my_params, model_path, output_path)
```

3. To check whether your Config(s) are valid for LPJmL you can pass the the returned `tibble` to `check_lpjml`. It won't raise an error (dependencies might not be satisfied yet) but will print/return the information of `lpjcheck`.

```
lpjml_check(config_details, model_path, output_path)
```

4. Based on `write_config` you are now able to either execute the LPJmL simulation(s) for the Config file(s) via `submit_lpjml` or run it interactively via `run_lpjml`. `run_lpjml` can also be utilized within slurm jobs to execute multiple single cell runs.
`?submit_lpjml` *or* `?run_lpjml` *for more information.*

```
# submit to Slurm
run_details <- submit_lpjml(config_details, model_path, output_path)

# OR run interactively
run_details <- run_lpjml(config_details, model_path, output_path)
```

5. Around the Config file there are more helpful utility functions:
    1. `read_config` to read a `"config_*.json"` file as a nested R list object
    2. use the R internal `View` function for a tree visualization of a `"config_*.json"` file

6. Besides `check_lpjml` there is also a `make_lpjml` function for compiling LPJmL.

Usage

```
# https://gitlab.pik-potsdam.de/lpjml/lpjmlkit
library(lpjmlkit)
# why tibble? -> https://r4ds.had.co.nz/tibbles.html
library(tibble)
#>
#> Attaching package: 'tibble'
#> The following object is masked from 'package:lpjmlkit':
#>
#>     as_tibble

model_path <- "./LPJmL_internal"
output_path <- "./my_runs"
```

Single cell run sequences

Potential natural vegetation and land-use run:

```
# create parameter tibble
params_1 <- tibble(
  sim_name = c("spinup", "lu", "pnv"),
  landuse = c("no", "yes", "no"),
  # only for demonstration
  nspinup = c(1000, NA, NA),
  reservoir = c(FALSE, TRUE, FALSE),
  startgrid = c(27410, 27410, 27410),
  river_routing = c(FALSE, FALSE, FALSE),
  wateruse = c("no", "yes", "no"),
  const_deposition = c(FALSE, FALSE, TRUE),
  # run parameter: order defines the execution order
  #   (spinup is always 1 -> no -DFROM_RESTART),
  #   could be followed by a historic (order=2) and a future run (order=3)
  #   can be extended as far as required (order=n)
  order = c(1, 2, 2),
  # run parameter: dependency sets the restart paths to the corresponding
  #   restart_filename
  dependency = c(
    NA, "spinup", "spinup"
  )
)

# write config files
config_details_1 <- write_config(
  params = params_1, # pass the defined parameter tibble
  model_path = model_path,
  output_path = output_path,
  js_filename = "lpjml.js" # (default) the base js file
)

# read and view config
config_lu <- read_config(
  filename = paste0(output_path,"/configurations/config_lu.json")
)
View(config_lu)

# check config & LPJmL
check_config(
  x = config_details_1, # can be filename (vector) or tibble
  model_path = model_path,
  output_path = output_path
)

# execute runs sequentially
run_details_1 <- run_lpjml(
  config_details_1,
  model_path = model_path,
  output_path = output_path)
```

Old vs. new phenology and old land-use vs. input toolbox:

```
# create parameter tibble
params_2 <- tibble(
  sim_name = c("spinup_oldphen",
               "spinup_newphen",
               "oldphen",
               "old_lu",
               "lu_toolbox"),
  # object oriented like syntax to access nested json elements
  input.landuse.name = c(
    NA,
    NA,
    NA,
    NA,
    "input_toolbox_30arcmin/cftfrac_1500-2017_64bands_f2o.clm"
  ),
  nspinup = c(1000, 1000, NA, NA, NA),
  new_phenology = c(FALSE, TRUE, FALSE, TRUE, TRUE),
  startgrid = c(27410, 27410, 27410, 27410, 27410),
  river_routing = c(FALSE, FALSE, FALSE, FALSE, FALSE),
  order = c(1, 1, 2, 2, 2),
  dependency = c(NA, NA, "spinup_oldphen", "spinup_newphen", "spinup_newphen")
)

# write config files
config_details_2 <- write_config(params_2, model_path, output_path)

# check config & LPJmL
check_config(config_details_2, model_path, output_path)

# execute runs sequentially
run_details_2 <- run_lpjml(config_details_2, model_path, output_path)
```

Submission of global run sequences to the PIK cluster

Compare old vs new land use (lpjml input toolbox):

```
# create parameter tibble
params_3 <- tibble(
  sim_name = c("spinup",
               "old_lu",
               "lu_toolbox"),
  input.landuse.name = c(
    NA,
    NA,
    "input_toolbox_30arcmin/cftfrac_1500-2017_64bands_f2o.clm"
  ),
  order = c(1, 2, 2),
  dependency = c(NA, "spinup", "spinup"),
  # slurm option wtime: analogous to sbatch -wtime defines slurm option
  #   individually per config, overwrites submit_lpjml argument
  #   (same for sclass, ntasks, blocking)
  wtime = c("15:00:00", "3:00:00", "3:00:00")
)

# write config files
config_details_3 <- write_config(
  params = params_3,
  model_path = model_path,
  output_path = output_path,
  output_list = c("vegc", "soilc", "cftfrac", "pft_harvestc", "irrig"),
  output_list_timestep = c("annual","annual", "annual", "annual", "monthly"),
  # output_list_timestep = "annual",
  output_format = "clm"
)

# check config & LPJmL
check_config(config_details_3, model_path, output_path)

# submit runs to slurm
run_details_3 <- submit_lpjml(
  x = config_details_3,
  model_path = model_path,
  output_path = output_path,
  group = "open",
  wtime = "15:00:00",
  # only for demonstration
  no_submit = TRUE)
```