LPJmL Runner

Jannes Breier

2022-02-18

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). Else it is recommended to add something like this to the working environment or your `".profile"`.

```
export LPJROOT=<PATH_TO_LPJML_REPOSITORY>

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`.

```r
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) locally 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.*

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

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

5. Around the Config file there are two helpful utility functions:
   1. `read_config` to read a `"config_*.json"` file as a nested R list object
   2. `view_config` to use the `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

```r
library(lpjmlKit)
library(tibble)

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

Interactive single cell runs

Compare the old and the new phenology:

```r
# create parameter tibble
my_params <- tibble(
  sim_name = c("spinup_oldphen",
               "spinup_newphen",
               "transient_oldphen",
               "transient_newphen"),
  new_phenology = c(FALSE, TRUE, FALSE, TRUE),
  startgrid = c(27410, 27410, 27410, 27410),
  river_routing = c(FALSE, FALSE, FALSE, FALSE),
  order = c(1, 1, 2, 2),
  dependency = c(NA, NA, "spinup_oldphen", "spinup_newphen")
)

# write config files
config_details <- write_config(my_params, model_path, output_path)

# check config & LPjMl
check_config(config_details, model_path, output_path)
```

```r
# execute runs sequently
run_details <- run_lpjml(config_details, model_path, output_path)
```

Nested simulation sequences on the PIK cluster TBD