Introduction to the 'Rlpj' package (version 0.1)

Ramiro Silveyra Gonzalez, Maurizio Bagnara, Florian Hartig

May 23, 2016

1 Introduction

This vignette describes the R package '`Rlpj`'. LPJ is something

The `LPJ` package is thought to ease the use of LPJ in general and has emphasis on the parallelization of the LPJ-GUESS in High Computer Performance Environments (HPC). The package is a collection of tools that will allow you to create a parallel setup and run in parallel not only the LPJ model, but also the processing of the model outputs. The package can be also used for parallel run in personal computers.

In this vignette you will learn how to run LPJ in R using RLPJ. You will also prepare a parallel setup for running RLP in a cluster.

2 Running LPJ in R

The Rlpj package will help you to run LPJ-GUESS in parallel, but first you will have to prepare your data, so that you can make use the of the package utilities. Specifically, Rlpj requires that *i)* you have already compiled the LPJ model in your computer or home folder in the HPC, and that *ii)*the existence of a directory where the outputs of the model will be saved, lets name it the Main Directory.

In the Main Directory, the following files must be available:

- link to the model executable — guess / guesscmd.exe
- list of gridcells — gridlist.txt
- model template (optional) — global.ins
- model template (optional) — global_cru.ins or global_cf.ins

The model templates are included within the package as part of the system data. They have been edited with some reference values, but you might want to change then. The function *getTemplate* allows you to obtain the templates from package. To learn more about how to writing parameters in the template, check out the **writeParameters** function and the **getParameterList**.

3 Running LPJ in parallel

Running the LPJ parallel involves two steps. First, to create a parallel setup (**setupParallel**) function, and second, to actually run in parallel the model (runLPJ). The parallelization requires the packages `snow` and if you aim at using a MPI cluster, also the `Rmpi` package.

The **setupParallel** function will help you at creating a directory structure for storing your model outputs and arranging the model templates for each run. Calling the **setupParallel** returns a list object that contains all information needed to run the LPJ in parallel.

The **runLPJ** function reads the setup parallel object and creates a cluster to which submits the model wrapper function with its respective parameters.

In MPI clusters you have to exit the cluster.

4 An example

Before calling the *setupParallel* function, we will prepare the input parameters for the function. In this case, I want to test 20 different values for the *emax* parameter. Concerning the outputs, I am only interested in *aaet* and *lai* outputs.

Lets choose a mainDir and have a look at it. Remember that the mainDir is the folder in the directory structure for the parallel runs will be created. It should contain a link to your guess executable and file with list of gridcells.

```
> # clearing local environment
> rm(list=ls())
> # Loading the library
> library(Rlpj)
> # The main dir in this case is within the package
> mainDir <- system.file("extdata", package = "Rlpj")
> list.files(mainDir)

[1] "exampleOutputs" "gridlist.txt"   "runDirectory1"
[4] "runDirectory2"  "runDirectory3"
```

We want to use the global templates. This is not a problem because they are part of the package. Since we are running this on a personal laptop, lets use a low number cores and create a SOCK cluster. Now that we have prepared the variables, we can call the *setupParallel* function and build the directory structure.

```
> # Now, I specify which outputs I want process.
> typeList <- c("aaet", "nuptake")
> # Choosing the parameters
> # Only modifying 1 parameter: emax.
> # The complete list of parameters that can be modified can be found in the "data_raw/creat
```

```
> # Chekc parameterList
> parameterDefault <- getParameterList("europe")
> parameterDefault[1:10]

$run_lamda_max
[1] 0.8

$run_emax
[1] 5

$run_reprfrac
[1] 0.1

$run_wscal_min
[1] 0.35

$run_drought_tolerance
[1] 1e-04

$run_turnover_harv_prod
[1] 1

$tree_crownarea_max
[1] 40

$tree_turnover_root
[1] 0.7

$tree_ltor_max
[1] 1

$tree_k_allom2
[1] 40

> parameterDefault <- list (run_emax = NULL)
> # I want to test 20 different values for emax. I want therefore to run 20 time the LPJ
> par <- seq(1,5, len = 20)
> print (par)

 [1] 1.000000 1.210526 1.421053 1.631579 1.842105 2.052632
 [7] 2.263158 2.473684 2.684211 2.894737 3.105263 3.315789
[13] 3.526316 3.736842 3.947368 4.157895 4.368421 4.578947
[19] 4.789474 5.000000

> # I create the list object with the parameters
> parameterList <- vector("list", length(par))
> for (i in 1:length(par)) {
```

```
+   parameterDefault$run_emax <- par[i]
+   parameterList[[i]] <- parameterDefault
+ }
> print(parameterList[c(2:3)])

[[1]]
[[1]]$run_emax
[1] 1.210526


[[2]]
[[2]]$run_emax
[1] 1.421053

> # We will be using only the cru data and run the model at 2 European locations
> # as specified in the gridlist.txt file.
> # Lets call the setupParallel
> options(error=traceback)
> setupObject <- setupLPJParallel(numCores= 3, clusterType = "SOCK", mainDir=mainDir)
> # if we check object, we see that contains the information for creating the cluster
> names(setupObject)

NULL
```

5 References