

LPJmL Data

Jannes Breier

2023-01-20

LPJmL Data is a `lpjmlkit` module that groups around the data class `LPJmLData` and aims to facilitate the reading and processing of LPJmL inputs and outputs by combining the raw data with available meta data (meta files, header files or manually) to avoid a large overhead. It further enables easy subsetting, transformations and basic statistics of the data and allows export to common data formats. All in all, with only a few lines of code.

Overview

LPJmL Data first requires the reading of LPJmL input or output data by applying the `read_io` function (1). The returned object is of class `LPJmLData` (2) for which base stats can be calculated (3), the inner data can be modified (4) or common data formats can be exported (5).

1. Data reading function `read_io`

The generic function to read LPJmL input and output, currently supported are three different file formats, “meta”, “clm” and “raw”:

- “meta” - *strongly recommended to use.*

For outputs from simulations in which `"output_metafile" : true` was configured or inputs using the format.

```
# read monthly runoff with meta file
runoff <- read_i("./output/mrunoff.bin.json")
```

- “clm” - *use if “meta” is not available or in combination.*

For all common LPJmL inputs or for outputs from simulations in which `"fmt" : "raw"` was configured. Some meta information (e.g. `band_names`) have to be specified manually.

```
# read monthly runoff data with header
runoff <- read_i("./output/mrunoff.clm",
  # set nbands and nstep manually otherwise month
  #   dimension is not recognized
  nbands = 1,
  nstep = 12,
  # not mandatory but useful information
  variable = "runoff",
  descr = "monthly runoff",
  unit = "mm/month")
```

- “raw” - *not recommended to use (for LPJmL).*

For old raw data. Every meta data has to be specified manually.

```
# read monthly runoff raw data
runoff <- read_i("./output/mrunoff.bin",
```

```
# every information has to be specified manually
...)
```

2. Data class LPJmLData

read_io returns an object of a R6 class LPJmLData with two main attributes, \$data and \$meta:

- **\$data** class base::array - returns the data array with default dimensions cell, time and band

```
runoff$data
#      , , band = 1
#
#      time
# cell    1901-01-31    1901-02-28    1901-03-31    1901-04-30
#  0      2.427786e+02    1.265680e+02    2.279087e+02    2.027685e+02
#  1      4.189225e-14    1.032507e-16    0.000000e+00    0.000000e+00
#  2      3.860512e-14    0.000000e+00    0.000000e+00    0.000000e+00
#  3      0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
#  4      0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
#  5      0.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
```

- **\$meta** Meta data of class LPJmLMetaData - returns the corresponding meta data (e.g. runoff\$meta\$unit)

```
runoff$meta
# $sim_name "lu_cf"
# $source "LPJmL C Version 5.3.001"
# $history "./LPJmL_internal/bin/lpjml ./configurations/config_lu_cf.json"
# $variable "runoff"
# $descr "monthly runoff"
# $unit "mm/month"
# $nbands 1
# $nyear 111
# $firstyear 1901
# $lastyear 2011
# $nstep 12
# $timestep 1
# $ncell 67420
# $firstcell 0
# $cellsize_lon 0.5
# $cellsize_lat 0.5
# $datatype "float"
# $scalar 1
# $order "cellseq"
# $bigendian FALSE
# $format "raw"
# $filename "runoff.bin"
```

3. Base stats of LPJmLData objects

To get an overview of the data, LPJmLData offers support for various base functions: length(), dim(), dimension(), summary() and plot().

More methods can be added upon request.

```

# self print, also via print(runoff)
runoff
# $meta %>%
#   .$sim_name "lu_cf"
#   .$variable "runoff"
#   .$descr "monthly runoff"
#   .$unit "mm/month"
#   .$nbands 1
#   .$nyear 111
#   .$nstep 12
#   .$timestep 1
#   .$ncell 67420
#   .$cellsize_lon 0.5
#   .$cellsize_lat 0.5
# Note: not printing all meta data, use $meta to get all.
# $data %>%
#   dimnames() %>%
#   .$cell "0" "1" "2" "3" ... "67419"
#   .$time "'1901-01-31" "1901-02-28" "1901-03-31" "1901-04-30" ... "2011-12-31"
#   .$band "1"
# $summary()
#   1
#   Min.   : 0.0000
#   1st Qu.: 0.0619
#   Median : 4.4320
#   Mean   : 28.7658
#   3rd Qu.: 27.5627
#   Max.   :2840.9602
# Note: summary is not weighted by grid area.

# dimension length of $data array, dimnames function also available
dim(runoff)
# cell time band
# 67420 1332 1

# plot as maps or time series, depending on the dimensions
plot(runoff)

```

4. Modify LPJmLData objects

LPJmLData objects come with a bundle of methods to modify its state: `add_grid()`, `transform()` and `subset()`.

- **** add_grid() **** Add a **\$grid** attribute (as LPJmLData object) to the object, providing the spatial reference (longitude and latitude) for every cell.

```

# object oriented (R6 class) notation (assigning grid directly to runoff)
runoff$add_grid()

# common R notation (overwriting the original object)
runoff <- add_grid(runoff)

```

```
# use read_io arguments if file_type != meta
runoff <- add_grid(runoff, "./output/grid.clm")
```

- **** transform() **** the \$data dimensions.
Either the cell dimension into two “lon” (longitude) and “lat” (latitude) dimensions or the time into “year”, “month” and “day” dimension (if available) - combinations and back transformations are also possible. For format “lon_lat” attribute \$grid is required.

```
# transform into lon and lat dimensions, if add_grid is not executed
# before, it is called implicitly, if file_format == "meta"
runoff <- transform(runoff, to = "lon_lat")
runoff
# [...]
# $data %>%
#   dimnames() %>%
#     .$lat   "-55.75" "-55.25" "-54.75" "-54.25" ... "83.75"
#     .$lon   "-179.75" "-179.25" "-178.75" "-178.25" ... "179.75"
#     .$time   "1901-01-31" "1901-02-28" "1901-03-31" "1901-04-30" ... "2011-12-31"
#     .$band   "1"
# [...]

# transform into year and month dimension (day not available for montly
# runoff)
runoff <- transform(runoff, to = "year_month_day")
runoff
# [...]
# $data %>%
#   dimnames() %>%
#     .$lat   "-55.75" "-55.25" "-54.75" "-54.25" ... "83.75"
#     .$lon   "-179.75" "-179.25" "-178.75" "-178.25" ... "179.75"
#     .$month  "1" "2" "3" "4" ... "12"
#     .$year   "1901" "1902" "1903" "1904" ... "2011"
#     .$band   "1"
# [...]

# transform back to original dimensions
runoff <- transform(runoff, to = c("cell", "time"))
runoff
# [...]
# $data %>%
#   dimnames() %>%
#     .$cell   "0" "1" "2" "3" ... "67419"
#     .$time   "1901-01-31" "1901-02-28" "1901-03-31" "1901-04-30" ... "2100-12-31"
#     .$band   "1"
# [...]
```

- **** subset() **** the \$data.
Use \$data dimensions as key and dimension names or indices as value to subset \$data. \$meta is adjusted according to the subset.

```
# subset by dimnames (character string)
runoff <- subset(runoff, cell = "1991-05-31")
runoff
# $meta %>%
#   .$year 1
```

```

#   .$ncell 67420
#   .$subset TRUE
# [...]
# Note: not printing all meta data, use $meta to get all.
# $data %>%
#   dimnames() %>%
#   .$cell "0" "1" "2" "3" ... "67419"
#   .$time "1991-05-31"
#   .$band "1"
# [...]

# subset by indices
runoff <- subset(runoff, cell = 28697:28700)
runoff
# $meta %>%
#   .$year 1
#   .$ncell 4
#   .$subset TRUE
# [...]
# Note: not printing all meta data, use $meta to get all.
# $data %>%
#   dimnames() %>%
#   .$cell "28696" "28697" "28698" "28699"
#   .$time "1991-05-31"
#   .$band "1"
# [...]

```

5. Export LPJmLData objects

Finally LPJmLData objects can be exported into common R data formats: `array`, `tibble`, `raster` and `terra`.

More export methods can be added upon request.

- `as_array()` Export as `array`. Besides directly accessing `$data` `as_array` further provides functionality to subset and aggregate `$data`.

```

# export as array with subset of first 4 time steps and aggregation of
#   dimension cell (mean)
as_array(runoff,
         subset = list(time = 1:6),
         aggregate = list(cell = mean))
#           band
# time      1
# 1901-01-31 19.49611
# 1901-02-28 20.28368
# 1901-03-31 27.93595
# 1901-04-30 36.90505
# 1901-05-31 39.38885
# 1901-06-30 32.80252

```

- `as_tibble()` Export `$data` as a `tibble` object, providing the same further functionality as `as_array`. `as_array` further provides functionality to subset and aggregate `$data`.

```

# export as array with subset of first 4 time steps and aggregation of
# dimension cell (mean)
as_tibble(runoff, subset = list(time = 1:6))
## A tibble: 404,520 × 4
#   cell time      band value
#   <fct> <fct>    <fct> <dbl>
# 1 0      1901-01-31 1      184.
# 2 1      1901-01-31 1          0
# 3 2      1901-01-31 1          0
# 4 3      1901-01-31 1          0
# 5 4      1901-01-31 1          0
# 6 5      1901-01-31 1          0
# 7 6      1901-01-31 1          0
# 8 7      1901-01-31 1          0
# 9 8      1901-01-31 1          0
# 10 9     1901-01-31 1          0
## ... with 404,510 more rows

```

- **** as_raster()** / as_terra()** Export \$data as a raster or a terra object (successor of raster), providing the same further functionality as as_array.

```

# export first time step as raster
as_raster(runoff, subset = list(time = 1))
# class      : RasterLayer
# dimensions : 280, 720, 201600 (nrow, ncol, ncell)
# resolution : 0.5, 0.5 (x, y)
# extent      : -180, 180, -56, 84 (xmin, xmax, ymin, ymax)
# crs         : +proj=longlat +datum=WGS84 +no_defs
# source      : memory
# names       : runoff
# values      : -1.682581e-13, 671.8747 (min, max)

# export first time step as terra SpatRaster
as_terra(runoff, subset = list(time = 1))
# class      : SpatRaster
# dimensions : 280, 720, 1 (nrow, ncol, nlyr)
# resolution : 0.5, 0.5 (x, y)
# extent      : -180, 180, -56, 84 (xmin, xmax, ymin, ymax)
# coord. ref. : lon/lat WGS 84 (EPSG:4326)
# source      : memory
# name        : runoff
# min value   : -1.682581e-13
# max value   : 6.718747e+02
# unit        : mm/month

# export first 4 time step as raster brick
as_raster(runoff, subset = list(time = 1:4))
# class      : RasterBrick
# dimensions : 280, 720, 201600, 4 (nrow, ncol, ncell, nlayers)
# resolution : 0.5, 0.5 (x, y)
# extent      : -180, 180, -56, 84 (xmin, xmax, ymin, ymax)
# crs         : +proj=longlat +datum=WGS84 +no_defs
# source      : memory
# names       : X1901.01.31, X1901.02.28, X1901.03.31, X1901.04.30

```

```

# min values : -1.682581e-13, -1.750495e-13, -2.918900e-13, -1.516298e-13
# max values :      671.8747,      785.2363,      828.2853,      987.4359

# export first 4 time step as terra SpatRaster
as_terra(runoff, subset = list(time = 1:4))
# class      : SpatRaster
# dimensions  : 280, 720, 4  (nrow, ncol, nlyr)
# resolution  : 0.5, 0.5  (x, y)
# extent      : -180, 180, -56, 84  (xmin, xmax, ymin, ymax)
# coord. ref. : lon/lat WGS 84 (EPSG:4326)
# source      : memory
# names       : 1901-01-31, 1901-02-28, 1901-03-31, 1901-04-30
# min values  : -1.682581e-13, -1.750495e-13, -2.918900e-13, -1.516298e-13
# max values  :  6.718747e+02,  7.852363e+02,  8.282853e+02,  9.874359e+02
# unit        :      mm/month,      mm/month,      mm/month,      mm/month
# time (days) : 1901-01-31 to 1901-04-30

```

miscellaneous

More helpful functions that come with LPJmL Data are:

- `read_meta` to read meta meta information of meta and header files as `LPJmLMetaData` objects
- `LPJmLMetaData` objects can be exported as `as_list` and `as_header` to use for creating header objects or write header files.

Usage

```
library(lpjmlkit)
```

1. Example *Global Trend in npp over the years*

```

npp <- read_io(filename = "./output/npp.bin.json",
               subset = list(year = 1970:2011))

# transform time in year and month dimension
npp$transform(to = "year_month_day")

# plot timeseries with aggregated cell and month dimension
plot(npp,
     aggregate = list(cell = sum, month = sum))

# also available as data array
global_npp_trend <- as_array(npp,
                             aggregate = list(cell = sum, month = sum))

```

2. Example *Runoff in northern hemisphere's summertime*

```

# alternative function notation via %>% operator (in magrittr package)
library("magrittr")
runoff <- read_io(filename = "./output/runoff.bin.json",

```

```

subset = list(year = 2002:2011))

runoff %>%
  # transform time and space dimensions
  transform(to = c("year_month_day", "lon_lat")) %>%
  # ... to subset summer month as well as northern hemisphere (positive)
  # latitudes
  subset(month = 6:9,
         lat = seq(0.25, 83.75, by = 0.5)) %>%
  # for plotting sum up summer month and take the average over the years
  plot(aggregate = list(year = mean, month = sum))

```

3. Example GPP per latitude

```

# alternative function notation via %>% operator (in magrittr package)
gpp <- read_io(filename = "./output/gpp.bin.json",
              subset = list(year = 2002:2011))

# transform into lon_lat format
gpp$transform(to = "lon_lat")

# plot gpp per latitude
plot(gpp, aggregate = list(time = mean, lon = mean))

```

4. Example CFT fractions for area around Potsdam

```

# coordinates for cells around Potsdam
coordinates <- tibble::tibble(lat = c(52.25, 52.400922, 53.25),
                             lon = c(12.75, 13.03638, 12.75))

# complete pipe notation, from reading to plotting data
read_io(
  filename = glue("{output_path}/cftfrac.bin.json"),
  subset = list(year = 2000:2018)
) %>%
  transform(to = "lon_lat") %>%
  # special case for subsetting of lat and lon pairs
  subset(coords = coordinates) %>%
  # mean spatial dimensions
  plot(aggregate = list(lon = mean, lat = mean))

```

Notes & tips

1. LPJmLData and LPJmLMetaData objects are closed environments, each of an R6 class that function as a data container.

Do not replicate R6 objects like

```

my_data <- lpjml_data
# instead use
my_data <- lpjml_data$clone(deep = TRUE)

```

Otherwise if any methods are performed on my_data the same method is performed on lpjml_data since they point to the same environment.

2. Do not try to overwrite any `LPJmLData` objects. It is either not possible or can mess up the integrity of the object. Therefore do not use method named like `$._<method>_` or attributes named like `$._<attribute>_`
3. When performance is important, choose R6 method notation `runoff$transform(to = "lon_lat")` over common R notation `transform(runoff, to = "lon_lat")`.
4. When comparing old (< LPJmL version 5.3) data with LPJmL 5.3 data it can be useful to combine meta (`"output_metafile" : true`) with the header file format (`"fmt": "clm"`) for simplification of process pipelines.