startR tutorial

Núria Pérez Zanón and An-Chi Ho

nuria.perez@bsc.es an.ho@bsc.es

User training
02/09/2020

Outline

LECTURE (1hr)

- The workflow of startR (+ demo)
- How to create a self-defined function for startR/multiApply

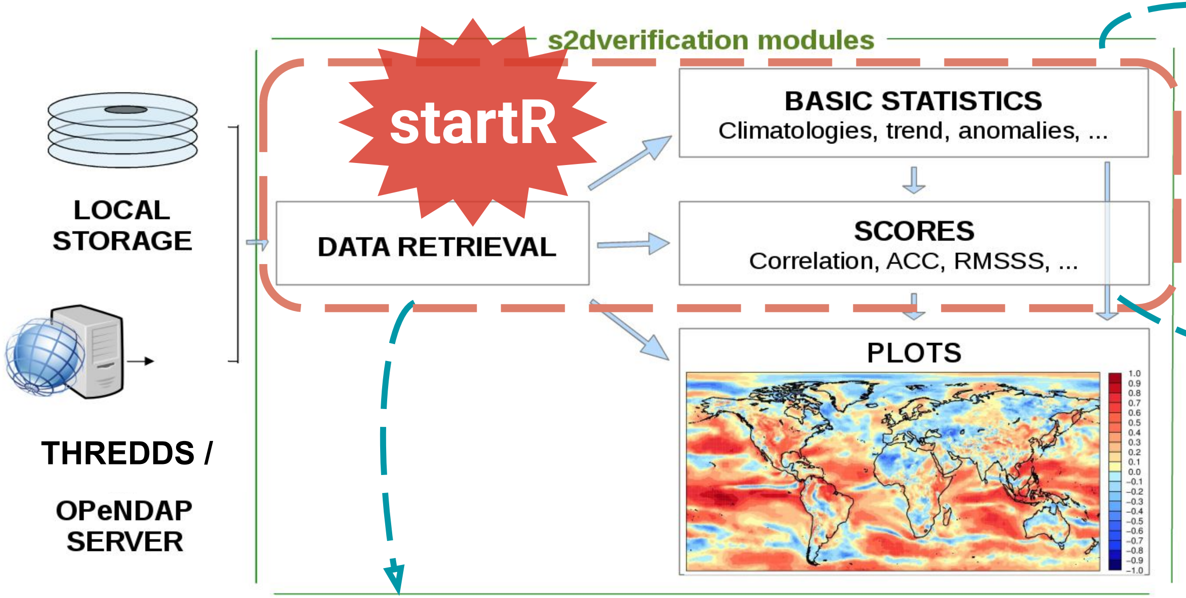
HANDS-ON (1hr)

- Use Start() parameters to get the desired data array structure
- Define the workflow and run the execution locally

LECTURE

The workflow of startR (+ demo)

How is startR born?



Complex analysis
- bootstrapping
- machine learning model
- etc.

Big data
- Higher resolution in all dimensions
- Various kinds of data required

Issues
- Longer data loading and processing time
- Limited memory space in local machine

startR feature

- ★ An R package tailored for big multi-dimensional data retrieval and processing
- ★ Automatic chunking of data set and parallel distributed data-processing on HPCs
- ★ Highly flexible according to the data structure and users' needs
- ★ Pre-processing: user-defined data transformation or reordering/merging/splitting dimension before performing analysis
- ★ Easy to reuse scripts due to the clear workflow
- ★ Use ecFlow workflow manager for job distribution and monitoring on HPC

startR function

`Start()`	Declare the data to be processed
`Step()` `AddStep()`	Specify the operation to be applied to the data
`Compute()`	Do chunking, specify the machine and its configuration for job employment, and trigger the execution
`Collect()`	Collect the results of background execution

And other helper functions.

startR workflow

Data declaration → Operation defining → Workflow defining → Job execution → Result collection

Following the startR framework, users can create an analysis in a concise script with all the information needed, including:

1. Data declaration: Declare the data sources and the required file/inner dimensions
2. Operation defining: Define the data processing operation to be applied
3. Workflow defining: Combine the elements from the previous steps to define the workflow
4. Job execution: Trigger the job execution and set up the configuration for the machine used for data processing
5. Result collection: Collect and combine the chunks when the execution is done

startR workflow

Data declaration → Operation defining → Workflow defining → Job execution → Result collection

```
repos <-
'/esarchive/exp/ecmwf/system5_m1/monthly_mean$var$_f6h/$var$_$sdate$.nc'

data <- Start(dat = repos,
              var = 'tas',
              sdate =  c('20170101', '20170201'),
              ensemble = indices(1:50),
              time = 'all',
              latitude = values(list(lat.min, lat.max)),
              longitude = values(list(lon.min, lon.max)),
              …,
              retrieve = FALSE)
```

data source

file dimension

inner dimension

Parameters for pre-processing, metadata, and definition etc.

Create a pointer to data repository

startR workflow

Data declaration → Operation defining → Workflow defining → Job execution → Result collection

Start() parameters

[define dimension]
pattern_dims
metadata_dims
path_glob_permissive
return_vars
synonims

[reshape]
merge_across_dims
merge_across_dims_narm
split_multiselected_dims

[interpolate]
transform
transform_params
transform_vars
transform_extra_cells
apply_indices_after_transform

[helper function] (no need to change in theory)
file_opener
file_var_reader
file_dim_reader
file_data_reader
file_closer
selector_checker

[operation]
num_procs
silent
debug

startR workflow

Data declaration → **Operation defining** → Workflow defining → Job execution → Result collection

Define the operation in the **R function form**.

The operation is only for **essential dimension** not the whole data, which is the same concept as multiApply.

The output size should be small enough to fit in the workstation.

It can be as simple as one function:

```
fun <- function(x) {
    a <- apply(x, 2, mean)
    dim(a) <- c(time = length(a))
    return(a)
}
```

Or a complicated user-defined operation:

```
stratify_atomic <-  function(field, MJO, season = c("JFM", "OND"), lag = 0, ampl = 2, relative = TRUE,
signif = 0.05) {
      # Arrange wind in form (days) to match MJO
      nmonths <- dim(field)[3]
      field <- aperm(field, c(1, 2, 4, 3))
      dim(field) <- c(31 * nmonths)
      if(season == "JFM") {
          daysok <- rep(c(rep(TRUE, 31), rep(TRUE, 28),
                          rep(FALSE, 3), rep(TRUE, 31)), nmonths / 3)
      } else if (season == "OND") {
          daysok <- rep(c(rep(TRUE, 31), rep(TRUE, 30),
                          rep(FALSE, 1), rep(TRUE, 31)), nmonths / 3)
      }
      field <- field[daysok]
      dim(field) <- c(days = length(field))

      if(dim(field)[1] != dim(MJO)[1]) {
          stop("MJO indices and wind data have different number of days")
      }

      idx <- function(MJO, phase, ampl, lag){
          if(lag == 0) {
              return(MJO$phase == phase & MJO$amplitude > ampl)
          }
          if(lag > 0) {
              return(dplyr::lag(MJO$phase == phase & MJO$amplitude > ampl,
                                lag, default = FALSE))
          }
          if(lag < 0) {
              return(dplyr::lead(MJO$phase == phase & MJO$amplitude > ampl,
                                 - 1 * lag, default = FALSE))
          }
      }
      strat <- plyr::laply(1:8, function(i) {
                           idx2 <- idx(MJO, i, ampl, lag)
                           if (relative) {
                                 return(mean(field[idx2]) / mean(field) - 1)
                           } else {
                                 return(mean(field[idx2]) - mean(field))
                           }})
      strat.t.test <- plyr::laply(1:8, function(i) {
                          idx2 <- idx(MJO, i, ampl, lag)
                          return(t.test(field[idx2], field)$p.value)})
      return(list(strat = strat, t.test = strat.t.test))
```

(Created by Llorenç)

startR workflow

Data declaration → Operation defining → **Workflow defining** → Job execution → Result collection

```
step <- Step(fun = fun,
             target_dims = c('ensemble'),
             output_dims = NULL)

wf <- AddStep(data, step, ...)
```

Additional parameters for the previous defined function

Which dimensions the operation performs on?

Which dimensions of output are expected?

Data dimension

dat	var	sdate	ensemble	time	latitude	longitude
1	1	1	51	7	256	512

```
fun <- function(x) {
  mean(x)
}
```

startR workflow

Data declaration | Operation defining | Workflow defining | **Job execution** | Result collection

```
res <- Compute(wf,
               chunks = list(latitude = 2,
                             longitude = 2),
               threads_load = 2,
               threads_compute = 4,
               cluster = list(queue_host = 'nord3',
                              queue_type = 'lsf',
                              …),
               ecflow_suite_dir = '/home/Earth/user_id/startR_local/',
               wait = TRUE
               )
```

Define chunking. Ensure each chunk size fits in the RAM memory module of HPC

Only needed for remote execution

startR workflow

Data declaration → Operation defining → Workflow defining → Job execution → **Result collection**

Use Collect() to collect and combine the results in the workstation if Compute() is on HPCs and its parameter ‘wait = FALSE’.

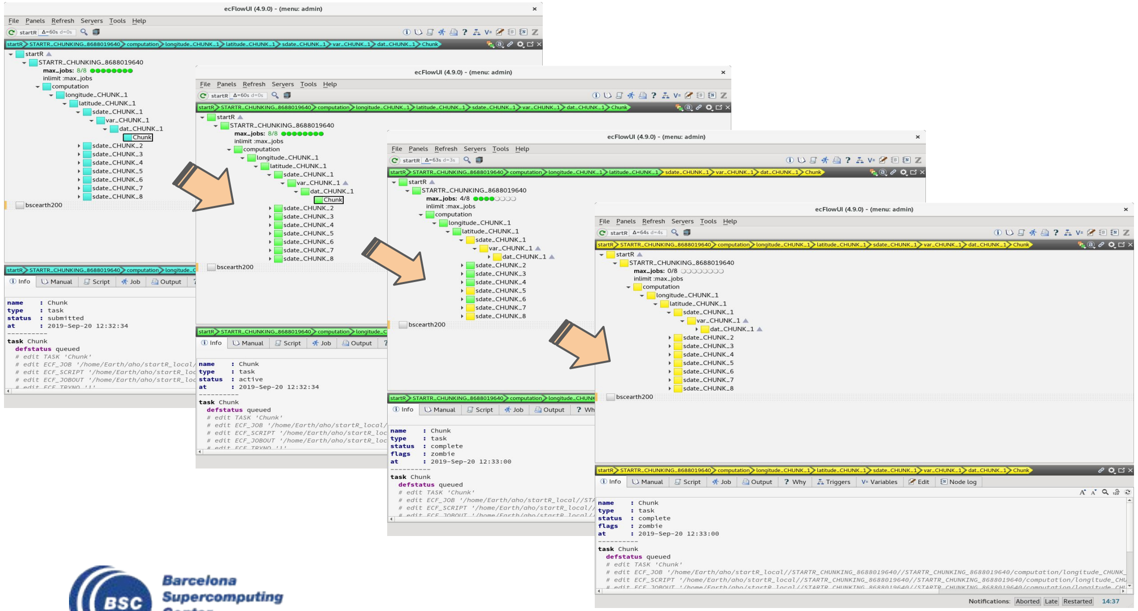
```
res <- Compute(wf,
               chunks = list(latitude = 2,
                             longitude = 2),
               cluster = list(queue_host = 'nord3',
                              queue_type = 'lsf',
                              …),
               ecflow_suite_dir = '/home/Earth/user_id/startR_local/',
               wait = FALSE
               )
saveRDS(res, file = 'test_collect.Rds')
collect_info <- readRDS('test_collect.Rds')
result <- Collect(collect_info, wait = TRUE)
```

store the descriptor of the execution

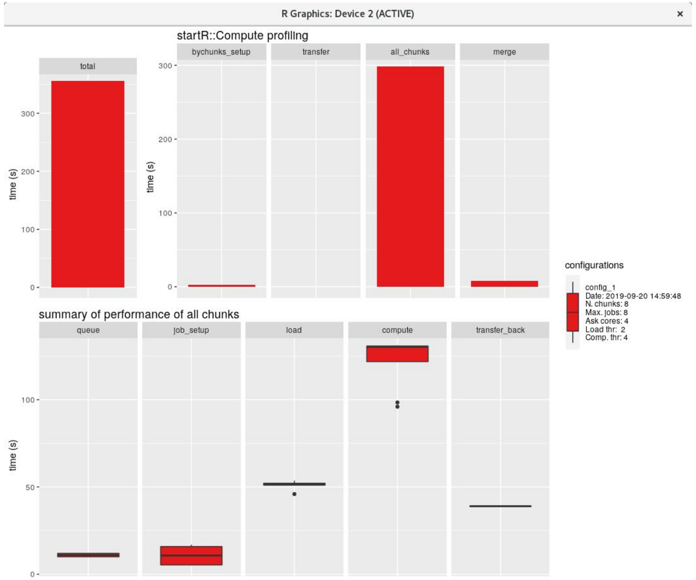
startR workflow (bonus)

Data declaration → Operation defining → Workflow defining → Job execution → Result collection

Monitoring the execution on ecFlow UI



Profiling the execution



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

startR workflow (demo)

https://earth.bsc.es/gitlab/es/startR/-/blob/develop-tutorial/inst/doc/tutorial/nord3_demo.R

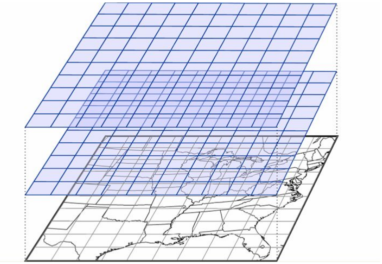
LECTURE

How to create a self-defined function for startR/multiApply

Apply from MultiApply

Basic example: **compute the mean to visualize the spatial distribution on the first lead time for each model**

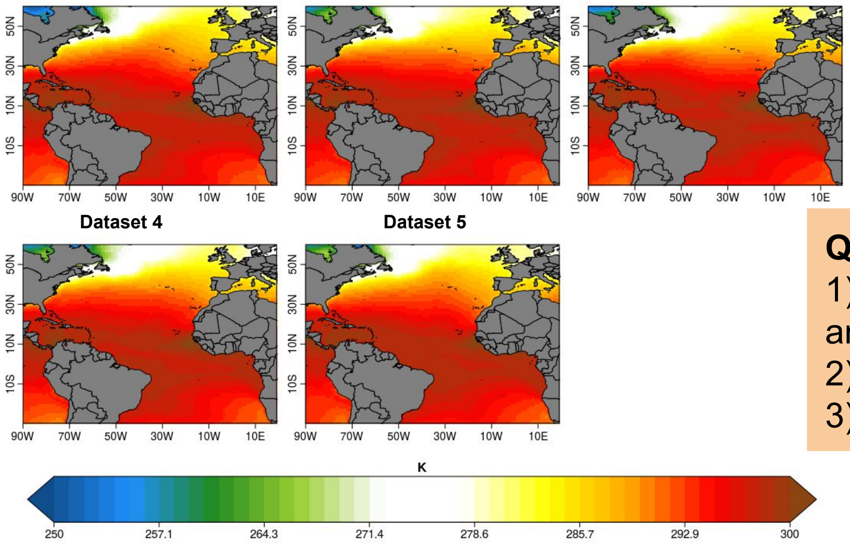
Input data:



Data dimension (~56GB)

dat	var	sdate	ensemble	time	latitude	longitude
5	1	30	51	7	256	512

Expected result:



Questions:
1) What data is involved in my analysis?
2) What analysis I want to perform?
3) What output data I will expect?

Apply from MultiApply

Basic example: **compute the mean to visualize the spatial distribution on the first lead time for each model**

<u>Data dimension</u> (~56GB)

dat	var	sdate	ensemble	time	latitude	longitude
5	1	30	51	7	256	512

Classical solution using loops:

```
map_lt1<- array(numeric(), dim = c(dat = 5, latitude = 256, longitude = 512))     # Initialization
for (d in 1:5) {                                                                   # Loop on models
  for (y in 1:256) {                                                               # Loop on latitude
    for (x in 1:512) {                                                             # Loop on longitude
      map_lt1[d, y, x] <- mean(data[d , , , , 1, y, x])                            # Fix leadtime 1
    }
  }
}
```

- *<u>dat</u>*, *<u>latitude</u>* and *<u>longitude</u>* are the output dims.
- *<u>var</u>* and *<u>time</u>* are fix dims (this dimensions are not contributing in this analysis).
- *<u>sdate</u>* and *<u>ensemble</u>* are the ***target_dimensions*** (in each step of the loop the mean is computed over 30 different start dates and its 51 members each one, so, 1530 values)
- the function in this case is <u>mean()</u>.

Apply from MultiApply

Basic example: **compute the mean to visualize the spatial distribution on the first lead time for each model**

Data dimension (~56GB)

dat	var	sdate	ensemble	time	latitude	longitude
5	1	30	51	7	256	512

Solution Using Apply:

```
library(multiApply)
result <- Apply(data,
                target_dims = c('sdate', 'ensemble'),
                fun = mean,
                 ncores = 4)$output1
```

 # Apply takes care of the loops and computes the mean in each piece of data

```
map_lt1 <- s2dverification::Subset(result, along = 'time', indices = 1)
```

Initially

Questions:
1) What data is involved in my analysis?
2) What analysis I want to perform?
3) What output data I will expect?

→

Finally

Questions:
1) Which are the dimensions of my data?
2') What function I want to apply?
2'') Over which dimensions?
3) Which are the output dimensions?

Apply from MultiApply

1) It extends the ***apply*** function to applications in which a function needs to be *applied simultaneously over multiple input arrays.*

2) Decreasing the length of the code,
    - *we get rid of error-prone* and
    - *we avoid memory-inefficient code*

3) Parameter **ncores** allows to parallelize the computation.
4) *Multiple outputs* can be returned in as a list of multidimensional arrays (by creating a function that returns several elements).

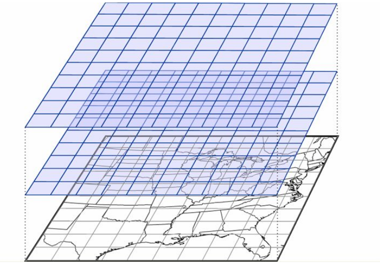
Apply from MultiApply

Next example: **Multi-Model Anomaly Agreement**

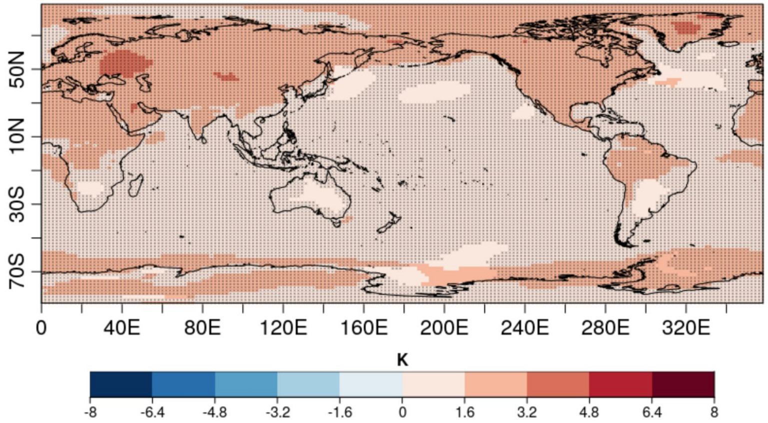
Input data:

Data dimension

dat	var	sdate	ensemble	ftime	latitude	longitude
5	1	30	25	31	89	180



Expected result:



Apply from MultiApply

Data dimension (~1583.8 Gb)						
dat	var	sdate	ensemble	ftime	latitude	longitude
5	1	30	25	31	89	180

Next example: **Multi-Model Anomaly Agreement**

Classical solution using loops:

```
library(s2dverification)
data <- array(runif(5 * 25 * 30 * 31 * 89 * 180, 5, 10),  dim = c(dat = 5, var = 1, sdate = 30, ensemble = 25, ftime = 31, latitude = 89, longitude = 180))
map_anom <- array(numeric(), dim = c(latitude = 89, longitude = 180))
map_agree <- array(numeric(), dim = c(latitude = 89, longitude = 180))

for (y in 1:89) {
   for (x in 1:180) {
      sub_dat <- Subset(Subset(data,  along = 'latitude', indices = y), along = 'longitude', indices = x)
      clim <- s2dv::MeanDims(sub_dat, dims = c('sdate', 'ensemble'))
      anom <- sub_dat - s2dv::InsertDim(s2dv::InsertDim(clim, posdim = 3, lendim = 25, name = 'ensemble'), posdim = 3, lendim = 30, name = 'sdate')
      ano_mean <- s2dv::MeanDims(anom, c('ensemble', 'ftime', 'sdate'))
      MM_mean <- s2dv::MeanDims(ano_mean, c('dat'))
           if (MM_mean > 0) {
              ano_agree <- 100 * sum(ano_mean[!is.na(ano_mean)] > 0) /
                               length(ano_mean[!is.na(ano_mean)])
           } else if (MM_mean < 0) {
               ano_agree <- 100 * sum(ano_mean[!is.na(ano_mean)] < 0)  /
                               length(ano_mean[!is.na(ano_mean)])
           } else if (MMc_mean == 0) {warning("Anomaly mean is equal to 0.")}
   map_anom[y,x] <- MM_mean
   map_agree[y,x] <- ano_agree
   }
}
```

Initialization

Anomaly

Agreement

output_dims (two arrays): lat, lon
target_dims: ensemble, dat, sdate

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Apply from MultiApply

Example: **Multi-Model Anomaly Agreement**

<u>Data dimension</u> (~56GB)

dat	var	sdate	ensemble	time	latitude	longitude]
5	1	1	51	**31**	256	512]

Create a function

```
data <- array(runif(5 * 25 * 30 * 31, 5, 10),  dim = c(dat = 5, var = 1, sdate = 30, ensemble = 25, ftime = 31))
Agreement <- function(data) {
     sub_dat <- Subset(Subset(data,  along = 'latitude', indices = y), along = 'longitude', indices = x)
    clim <- s2dv::MeanDims(sub_dat, dims = c('sdate', 'ensemble'))
     anom <- sub_dat - s2dv::InsertDim(s2dv::InsertDim(clim, posdim = 3, lendim = 25, name = 'ensemble'), posdim = 3, lendim = 30, name = 'sdate')
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Apply from MultiApply

Example: **Multi-Model Anomaly Agreement**

Data dimension (~1583.8 Gb)						
dat	var	**sdate**	**ensemble**	**ftime**	[latitude	longitude]
5	1	30	25	31	89	180

Solution Using Apply:

```
#data <- array(1:(5*30*25*31), dim = c(dat=5, sdate = 30, ensemble = 25, ftime = 31))

Agreement <- function(data) {
    clim <- s2dv::MeanDims(data, dims = c('sdate', 'ensemble'))
    anom <- Apply(data, c('dat', 'ftime'), function(x){x - clim}) $output1
     ano_mean <- s2dv::MeanDims(anom, c('sdate', 'ftime','ensemble'))
     MM_mean <- s2dv::MeanDims(ano_mean, c('dat'))
         if (MM_mean > 0) {
           ano_agree <- 100 * sum(ano_mean[!is.na(ano_mean)] > 0) /
                       length(ano_mean[!is.na(ano_mean)])
         } else if (MM_mean < 0) {
            ano_agree <- 100 * sum(ano_mean[!is.na(ano_mean)] < 0)  /
                       length(ano_mean[!is.na(ano_mean)])
         } else if (MMc_mean == 0) {warning("Anomaly mean is equal to 0.")}

   return(list(MMM = MM_mean, agree = ano_agree))
}
```

 # Apply takes care of the loops and computes the function in each piece of data

```
library(multiApply)
data <- array(runif(5 * 25 * 30 * 31 * 89 * 180, 5, 10),  dim =
c(dat = 5, var = 1, sdate = 30, ensemble = 25, ftime = 31,
latitude = 89, longitude = 180))
result <- Apply(data,
               target_dims = c('dat', 'sdate', 'ensemble', 'ftime'),
                fun = Agreement,
                ncores = 4)
str(result)
List of 2
 $ MMM  : num [1, 1:89, 1:180] 2.82e-18 -2.94e-17 2.87e-18 ...
 $ agree: num [1, 1:89, 1:180] 40 40 40 80 60 80 60 40 60 60 ...
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

The compatilibity between startR and other R packages (s2dv, CSTools, and ClimProjDiags)

startR workflow requires functions that works using dimensions names.

All functions on this packages use dimension names.

```
CST_YourFun(data1, ...) {
  YourFun(data1$data, …) {
    Apply(
          MyRequiredDimensions
          .yourfun)
  }
```

See **AnoAgree**() example from **ClimProjDiags**:
https://earth.bsc.es/gitlab/es/ClimProjDiags/-/blob/master/R/AnoAgree.R

LECTURE

The compatilibity between startR and other R packages (s2dv, CSTools, and ClimProjDiags)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

HANDS-ON

Preparation

Preparation

1. Log in your workstation or Nord3 interactive session
2. module load R
3. If you want to distribute jobs to Nord3… (not mandatory in this hands-on)
   a. the local R version has to be >= 3.5.0
   b. module load ecFlow

HANDS-ON

1. Use Start() parameters to get the desired data array structure

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Use Start() parameters to get the desired data array structure

https://earth.bsc.es/gitlab/es/startR/-/blob/develop-tutorial/inst/doc/tutorial/hands-on_part1.md

Answer:
https://earth.bsc.es/gitlab/es/startR/-/blob/develop-tutorial/inst/doc/tutorial/hands-on_part1_ans.md

HANDS-ON

2. Define the workflow and run the execution locally

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Define the workflow and run the execution locally

https://earth.bsc.es/gitlab/es/startR/-/blob/develop-tutorial/inst/doc/tutorial/hands-on_part2.md

Answer:
https://earth.bsc.es/gitlab/es/startR/-/blob/develop-tutorial/inst/doc/tutorial/hands-on_part2_ans.md

Development status

Merged requests: 93

Issues closed: 54

Open issues: 16

Future plans: improve the efficiency; enhance the features mentioned previously

Resources

startR GitLab
https://earth.bsc.es/gitlab/es/startR
startR CRAN documentation
https://cran.r-project.org/web/packages/startR/startR.pdf

Users' feedback is welcomed!