# daRt

The daRt package provides a very quick and flexible way to import data that is produced by the Discrete Anisotropic Radiative Transfer (DART) model. The data in daRt are formatted in a way that facilitates rapid data analysis.

## Installation

You can install the development version from [GitHub](https://github.com/) with:

# install.packages("devtools")  
devtools::install\_github("willmorrison1/daRt")

Load the package

library(daRt)

## Overview

This section demonstrates the most basic use of daRt to load the “directions” product data for the default “cesbio” simulation provided in this respository. Determine the type of files you want to load

sF <- simulationFilter(product = "directions")

Define a simulation directory

simulationDir <- "man/data/cesbio"

Load data for the given simulation using the predetermined file types

simData <- daRt::getData(x = simulationDir, sF = sF)

Use the data in the given “long” format

DF <- as.data.frame(simData)  
#> Warning: The `printer` argument is deprecated as of rlang 0.3.0.  
#> This warning is displayed once per session.  
head(DF, n = 3)  
#> # A tibble: 3 x 8  
#> # Groups: band, iter, typeNum, simName [1]  
#> zenith azimuth value band variable iter typeNum simName  
#> <dbl> <dbl> <dbl> <chr> <chr> <chr> <chr> <chr>   
#> 1 0 0 0.646 BAND0 BRF ITER1 "" cesbio   
#> 2 22.4 30 0.612 BAND0 BRF ITER1 "" cesbio   
#> 3 22.4 90 0.598 BAND0 BRF ITER1 "" cesbio

### SimulationFilter

The “SimulationFilter” object describes what data you want to extract from a DART output directory structure. Show the current configuration of the SimulationFilter

sF  
#> 'SimulationFilter' object for DART product: directions   
#>   
#> bands: BAND0   
#> variables: BRF   
#> iterations: ITER1, ITER2   
#> variablesRB3D: Intercepted, Scattered, Emitted, Absorbed, +ZFaceExit, +ZFaceEntry   
#> typeNums:   
#> imageType: ima, camera   
#> imageNo:

List the ‘setter’ and ‘accessor’ methods available

methods(class = "SimulationFilter")  
#> [1] bands bands<- getData getFiles   
#> [5] imageFiles imageNo imageNo<- imageType   
#> [9] imageType<- iters iters<- product   
#> [13] product<- show simdir typeNums   
#> [17] typeNums<- variables variables<- variablesRB3D   
#> [21] variablesRB3D<-  
#> see '?methods' for accessing help and source code

Use these methods to edit the SimulationFilter object e.g. the bands or iters (iterations) that you want to load

bands(sF) <- c("BAND0", "BAND1")  
iters(sF) <- "ITER1"

### SimulationFiles

The “SimulationFiles” object contains all information on the files that will be loaded, based on the provided SimulationFilter. It is used to explore the DART output directory structure. First define the simulation directory. For this example, simulationDir is a relative directory (based on the github data provided) and consists of one simulation.

#define the simulation directory  
simulationDir <- "man/data/cesbio"

If you install the package using devtools::install\_github then the “cesbio” simulation files will not be available automatically. To use these files, get them from github manually or use your own ‘cesbio’ simulation which is shipped with the DART model by default.

The simulation directory should be the base directory of the simulation. E.g. within simulationDir there should be the simulation ‘input’ and ‘output’ directories.

list.files(simulationDir)  
#> [1] "input" "output"

Now we have the simulation directory clarified, explore the files in the simulation that correspond to this filter

simFiles <- daRt::getFiles(x = simulationDir, sF = sF)

Explore the output of this to check we happy to continue and load the data. getFiles() is essentially a ‘dry-run’ of the data extraction

files(simFiles)  
#> band variable iter typeNum fileName  
#> 1 BAND0 BRF ITER1 man/data/cesbio/output//BAND0/BRF/ITER1/brf  
#> 2 BAND1 BRF ITER1 man/data/cesbio/output//BAND1/BRF/ITER1/brf  
#> simName  
#> 1 cesbio  
#> 2 cesbio

### SimulationData

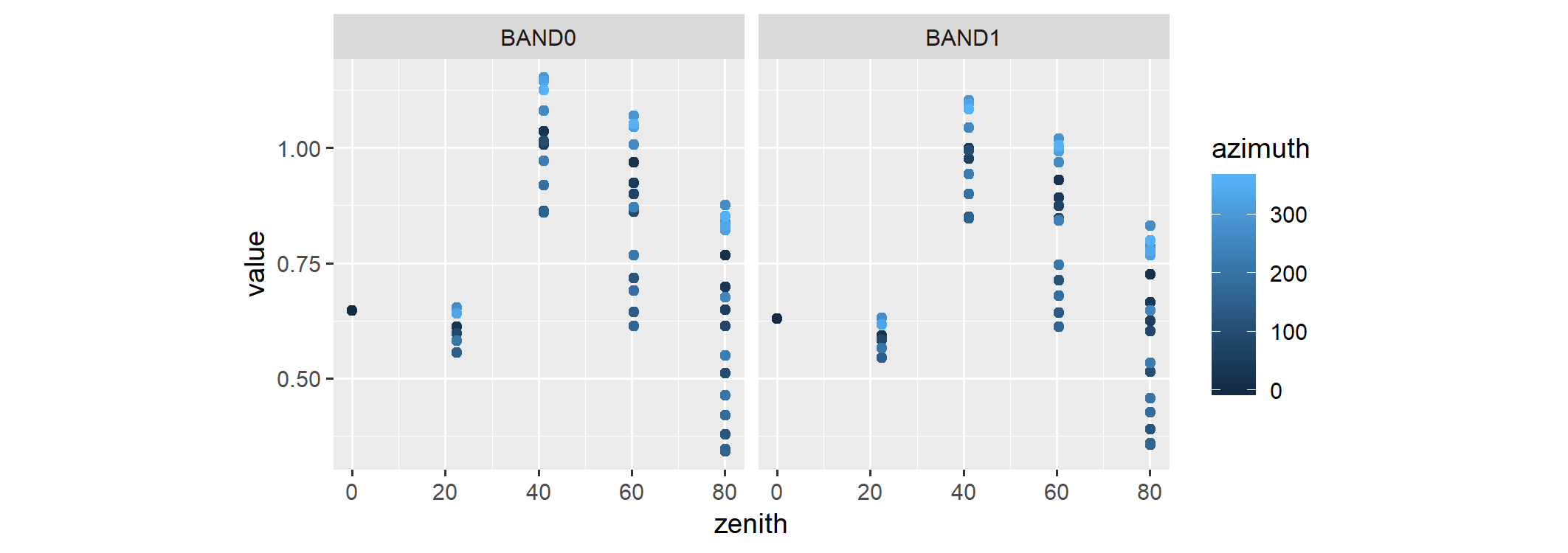
The SimulationData object contains all data for the given SimulationFilter. Do the following to extract DART output data using the getData() method

simData <- daRt::getData(x = simulationDir, sF = sF)  
#also can do this using simFiles object  
simData\_fromFiles <- daRt::getData(x = simFiles)  
identical(simData\_fromFiles, simData)  
#> [1] TRUE

### Simple plotting

By having data in a “long” format, it is easy to perform analysis on the data.

#plot using ggplot2  
library(ggplot2)  
plotOut <- ggplot(as.data.frame(simData)) +  
 geom\_point(aes(x = zenith, y = value, colour = azimuth)) +  
 facet\_wrap(~ band) +  
 theme(aspect.ratio = 1)  
plot(plotOut)



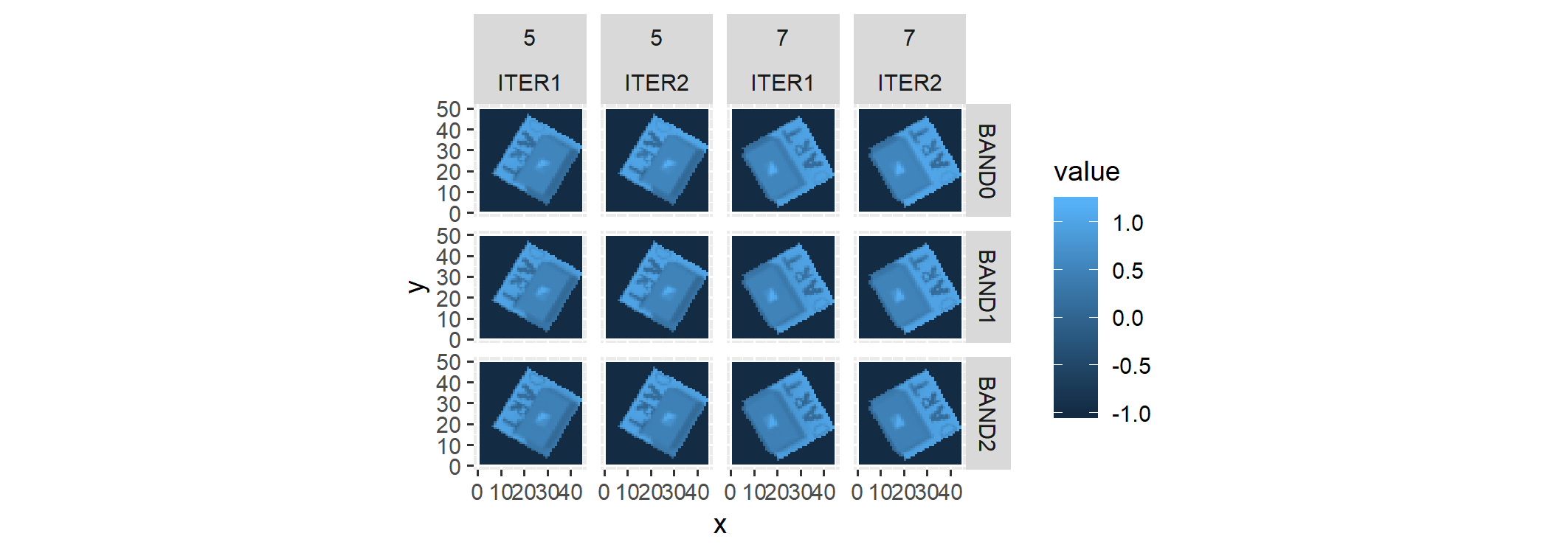
## Further examples

This section provides further misc examples and guidance for reference.

### SimulationFilter editing

To look at images for bands 0, 1 and 2; iters (iterations) 1 and 2, and imageNo (image numbers) 5 and 7, create the relevant SimulationFilter then load the data

#create SimulationFilter  
sF <- simulationFilter(product = "images",   
 bands = c("BAND0", "BAND1", "BAND2"),  
 iters = c("ITER1", "ITER2"),  
 variables = "BRF",  
 imageNo = c(5, 7),  
 imageType = "ima")  
#load data - 'nCores' allows parallel processing of files.  
#It is useful for access to drives that have optimised paralell I/O.  
#here load data using 2 cores.  
simData <- daRt::getData(x = simulationDir, sF = sF, nCores = 2)  
#simple plot of data  
ggplot(as.data.frame(simData)) +   
 geom\_raster(aes(x = x, y = y, fill = value)) +  
 facet\_grid(band ~ imageNo + iter) +  
 theme(aspect.ratio = 1)



### Radiative budget

Alter the SimulationFilter again to now look at files for the radiative budget product.

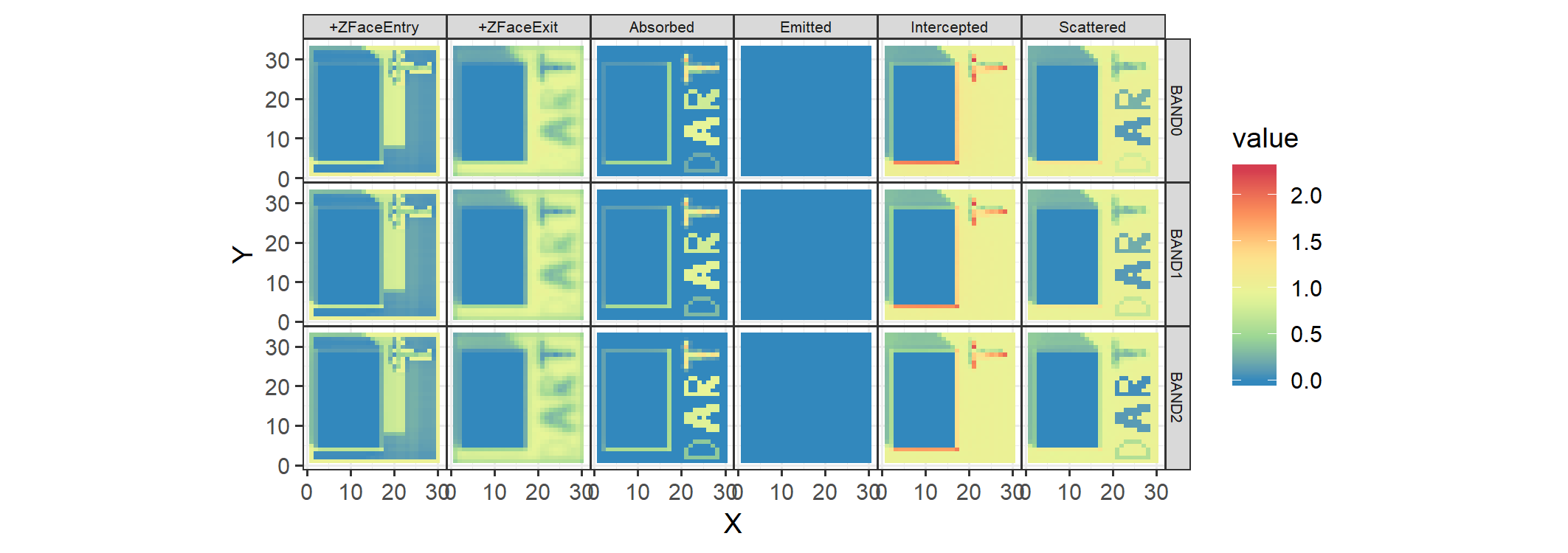
product(sF) <- "rb3D"  
simData <- daRt::getData(x = simulationDir, sF = sF)  
#> Warning in filesFun(x = x[i], sF = sF): Forcing 'RADIATIVE\_BUDGET' variable  
#> in 'simulationFilter' variables.

The 3D radiative budget data are stored with the X, Y and Z location of each cell (conforming to DART coordinate system i.e. "the part of the scene that horizontally is ‘top left’ and vertically is at the bottom is: X = 1, Y = 1, Z = 1), stored in 3 columns.

head(as.data.frame(simData), n = 3)  
#> # A tibble: 3 x 9  
#> # Groups: band, iter, typeNum, simName [1]  
#> X Y Z value variablesRB3D band iter typeNum simName  
#> <int> <int> <int> <dbl> <chr> <chr> <chr> <chr> <chr>   
#> 1 1 1 1 1.01 Intercepted BAND0 ITER1 "" cesbio   
#> 2 2 1 1 1.02 Intercepted BAND0 ITER1 "" cesbio   
#> 3 3 1 1 1.01 Intercepted BAND0 ITER1 "" cesbio

The below example uses “dplyr” to work with this data. Here we look at the lowest horizontal layer of each 3D radiative budget array (i.e. Z = 1).

library(dplyr)  
  
#filter lowest horizontal cross section of the radiative budget  
simData\_filtered <- as.data.frame(simData) %>%  
 dplyr::filter(Z == 1)  
  
ggplot(simData\_filtered) +   
 geom\_raster(aes(x = X, y = Y, fill = value)) +  
 facet\_grid(band ~ variablesRB3D) +  
 theme\_bw() +  
 theme(panel.spacing = unit(0, "cm"),   
 strip.text = element\_text(size = 6,   
 margin = margin(0.05, 0.05, 0.05, 0.05, unit = "cm"))) +  
 scale\_fill\_distiller(palette = "Spectral") +  
 theme(aspect.ratio = 1)



### Memory management

When performing analysis on a relatively large set of files, memory management is important. getData() loads all data to memory which is problematic when loading many large files (e.g. Radiative Budget). It is assumed that the user will perform some analysis on subsets of the raw data in a way that reduces the overall size of the data in memory. To demonstrate meory management, files in this section are loaded in two different scenarios: scenario 1 uses the default getData() to load and then analyse all data at once. Scenario 2 loads and analyses the data in pieces, which has a much smaller memory footprint (but may be slower). Both scenarios give the same result with different memory usage.

#### Scenario 1: Load data all at once

Load all radiative budget products at once into memory and take the mean of each horizontal layer.

sF <- simulationFilter(product = "rb3D",   
 bands = c("BAND0", "BAND1", "BAND2"),   
 iters = c("ITER1", "ITER2", "ILLUDIFF", "ILLUDIR"),  
 typeNums = "",  
 variables = "RADIATIVE\_BUDGET")  
simFiles <- daRt::getFiles(simulationDir, sF = sF)

There are twelve files each with 6 variables and each as a 3D array - i.e. quite a lot of data. Load in the data all at once. It is relatively memory intensive

simData <- daRt::getData(x = simFiles, nCores = 2)

and gives a relatively large array of data

DFdata <- as.data.frame(simData)  
head(DFdata, n = 3)  
#> # A tibble: 3 x 9  
#> # Groups: band, iter, typeNum, simName [1]  
#> X Y Z value variablesRB3D band iter typeNum simName  
#> <int> <int> <int> <dbl> <chr> <chr> <chr> <chr> <chr>   
#> 1 1 1 1 1.01 Intercepted BAND0 ITER1 "" cesbio   
#> 2 2 1 1 1.02 Intercepted BAND0 ITER1 "" cesbio   
#> 3 3 1 1 1.01 Intercepted BAND0 ITER1 "" cesbio  
dim(DFdata)  
#> [1] 784080 9

Do some analysis on the data. Get the mean of non-zero values across each vertical layer of each variablesRB3D, bands, iters (already grouped) according to the above column names

statVals <- DFdata %>%  
 dplyr::group\_by(X, Y, variablesRB3D, add = TRUE) %>%  
 dplyr::summarise(meanVal = mean(value[value != 0], na.rm = TRUE))

#### Scenario 2: Load data in sections and process each section

Do ‘scenario 1’ analysis but with data processed for each band separately to save on memory usage.

sF <- simulationFilter(product = "rb3D",   
 bands = c("BAND0", "BAND1", "BAND2"),   
 iters = c("ITER1", "ITER2", "ILLUDIFF", "ILLUDIR"),  
 typeNums = "",  
 variables = "RADIATIVE\_BUDGET")  
allBands <- bands(simData)  
allBands  
#> [1] "BAND0" "BAND1" "BAND2"  
simDataList <- vector(mode = "list", length = length(allBands))  
for (i in 1:length(allBands)) {  
 bands(sF) <- allBands[i]  
 simDataPiece <- daRt::getData(x = simulationDir, sF = sF)  
 simDataList[[i]] <- as.data.frame(simDataPiece) %>%  
 dplyr::group\_by(X, Y, variablesRB3D, add = TRUE) %>%  
 dplyr::summarise(meanVal = mean(value[value != 0], na.rm = TRUE))  
}

Now put together the list of data. As each list element is a summary of the raw data, it has a much smaller memory footprint. As the summary was performed on one band at a time, the amount of data loaded at once is less than if getFiles() was executed for all bands at once (scenario 1). By loading one band at a time as opposed to all three at once, the memory footprint is around 1/3 of scenario 1.

simDataDF <- dplyr::bind\_rows(simDataList)  
  
statVals1 <- simDataDF

Both scenarios give the same results

all.equal(statVals, statVals1)  
#> [1] TRUE

but by processing in parts, the latter (scenario 2) - produced by ‘statVals1’ - has a smaller memory footprint as the stats are calculated for each band separately. When inter-band stats are required, the example can be adapted to iterate over e.g. ‘iters’ or ‘variablesRB3D’.

### Compression of large binary files

DART radiative budget files are raw binary and can get very large. rb3DtoNc converts .bin to NetCDF (.nc) format, which gives smaller files sizes and can be compressed.

Get some DART radiative budget bindary data (the default data)

simulationDir <- "man/data/cesbio"  
sF <- simulationFilter(product = "rb3D",  
 bands = "BAND1",   
 iters = "ITER1",  
 typeNums = "",  
 variables = "RADIATIVE\_BUDGET")  
simFiles\_bin <- daRt::getFiles(simulationDir, sF = sF)  
simData\_bin <- as.data.frame(daRt::getData(simFiles\_bin))  
#get the file size - for later comparison  
fileSize\_bin <- file.size(files(simFiles\_bin)$fileName)

Convert the .bin data to .nc. The .bin file will be deleted by rb3DtoNc.

simFiles\_nc <- daRt::rb3DtoNc(simFiles\_bin)  
simData\_nc <- as.data.frame(daRt::getData(simFiles\_nc))

There are some very minor differences in the two products - likely due to the ncdf compression algorithm and/or rounding.

max(abs(simData\_nc$value - simData\_bin$value))  
#> [1] 9.187445e-08

The new .nc file is much smaller:

file.size(files(simFiles\_nc)$fileName) / fileSize\_bin  
#> [1] 0.1244318

and is much faster to read. It can also be read by third party NetCDF browsers e.g. ncview.