Embedding external functions in GGIR

Vincent van Hees

April 29 2020

Contents

1 Introduction

If you like GGIR but want to use algorithms for raw data not included in GGIR then the external function embedding feature can be the solution. For example, you may want to pilot a new machine learned classifiction algorithm but you do not want to write all the data cleaning and aggregation steps needed for analysis of real life 'out of the lab' acceleormeter data.

How it works:

Internally GGIR loads the raw accelerometer data in memory blocks of about 24 hours. When the data is in memory, corrected for calibration error, and resampled to the sample rate required by your function, GGIR applies its own default algorithms as well as the external function provided by you (Python or R). The external function is expected to take as input: A three-column matrix with the acceleration data corresponding to the three acceleration axes, and an optional parameters argument which can be of any R format (character, list, vector, data.frame, etc). As output your external function is expected to produce a matrix or data.frame with one or multiple columns corresponding to the output of your external function.

2 Example with external R function

In this example we will apply the function counts() from R package activityCounts to the raw data, which produces an estimate of Actigraph counts per second.

2.1 Write external function

Create file **calculateCounts.R** and insert the following code:

```
calculateCounts = function(data=c(), parameters=c()) {
  # data: 3 column matrix with acc data
  # parameters: the sample rate of data
  library("activityCounts")
  if (ncol(data) == 4) data= data[,2:4]
  mycounts = counts(data=data, hertz=parameters,
                    x_axis=1, y_axis=2, z_axis=3,
                    start_time = Sys.time())
  mycounts = mycounts[,2:4] #Note: do not provide timestamps to GGIR
  return(mycounts)
}
```

2.2 Provide external function to GGIR

Create a new .R file for running the GGIR analysis, e.g. named `myscript.R`, and insert the following code. Do not forget to update the filepath on the first line to point to your `calculateCounts.R` file.

```
source("~/calculateCounts.R")
myfun =  list(FUN=calculateCounts,
              parameters= 30,
              expected_sample_rate= 30,
              expected_unit="g",
              colnames = c("countsX","countsY","countsZ"),
              outputres = 1,
              minlength = 1,
              outputtype="numeric",
              aggfunction = sum,
              timestamp=F,
              reporttype="scalar")
```

The above code creates object `myfun` of type `list` which is expected to come with the following elements:

- `FUN` A character string specifying the location of the external function you want to apply.
- `parameters` The parameters used by the function, which can be stored in any format (vector, matrix, list, data.frame). The user should make sure that the external function can handle this object.
- `expected_sample_rate` Expected sample rate, if the inputdata has a difference sample rate, then the data will be resampled.
- `expected_unit` Expected unit of the acceleration by external function: “mg”, “g” or “ms2”. If input data is different it will be converted.
- `colnames` Character vector with the names of the columns produced by the external function.
- `outputres` The resolution (seconds) of the output produced by the external function. Note, that this needs to be equal to or a multitude of the short epoch size of the g.part1 output (5 seconds) or the short epoch size should be a multitude of this resolution. In this way GGIR can aggregate or repeat the external function output to be used inside GGIR.
- `minlength` The minimum length (seconds) of input data needed, typically the window per which output is provided.
- `outputtype` Character to indicate the type of external function output. Set to “numeric” if data is stored in numbers (any numeric format), or “character” if it is a character string.
- `aggfunction` If the data needs to be aggregated to match the short epoch size of the g.part1 output (5 seconds) then this element specifies what function should be used for the aggregation, e.g. mean, sum, median.

- `timestamp` Boolean to indicated whether timestamps (seconds since 1-1-1970) should be passed on to the external function as first columm of the data matrix..
- `reporttype` Character to indicate the type of reporting by GGIR: "scalar" if it should be averaged per day, "event" if it should be summed per day, or "type" if it is categorical variable that can only be aggregated per day by tabulating it.

Next, add a call to GGIR function g.shell.GGIR with `myfun` provided as one of its arguments:

```
library(GGIR)
g.shell.GGIR(datadir="~/myaccelerometerdata",
             outputdir="~/myresults",
             mode=1:2,
             epochvalues2csv = TRUE,
             do.report=2,
             myfun=myfun) #<= this is where object myfun is provided to g.shell.GGIR
```

Please see the general GGIR vignette for more information about function g.shell.GGIR.

3 Example with external Python function

In this example we will use an external Python function to estimate the dominant signal frequency per acceleration axis. Note this can also be done in R, but it shows that even Python functions can be provided.

3.1 Write external function

Create **dominant_frequency.py** and insert the code shown below:

```
import numpy

def dominant_frequency(x, sf):
  # x: vector with data values
  # sf: sample frequency
  fourier = numpy.fft.fft(x)
  frequencies = numpy.fft.fftfreq(len(x), 1/sf)
  magnitudes = abs(fourier[numpy.where(frequencies > 0)])
  peak_frequency = frequencies[numpy.argmax(magnitudes)]
  return peak_frequency
```

Create **dominant_frequency.R** that calls the python function and insert the following code:

```
dominant_frequency = function(data=c(), parameters=c()) {
  # data: 3 column matrix with acc data
  # parameters: the sample rate of data
  source_python("dominant_frequency.py")
  sf=parameters
  N = nrow(data)
  ws = 5 # windowsize
  if (ncol(data) == 4) data= data[,2:4]
  data = data.frame(t= floor(seq(0,(N-1)/sf,by=1/sf)/ws),
                    x=data[,1], y=data[,2], z=data[,3])
  df = aggregate(data, by = list(data$t),
                 FUN=function(x) {return(dominant_frequency(x,sf))})
  df = df[,-c(1:2)]
  return(df)
```

```
}
}
```

3.2 Provide external function to GGIR

Create a new .R file for running the GGIR analysis, e.g. named myscript.R, and insert the following blocks of code.

Specification of Python environment to use, this can also be a conda environment or docker container (see documentation R package reticulate for further details). Make sure that that this Python environment has all the required dependencies for the external function, here we will only need `numpy`.

```
  library("reticulate")
  use_virtualenv("~/myvenv", required = TRUE) # Local Python environment
  py_install("numpy", pip = TRUE)
```

Specify a `myfun` object as explained in the R example. Do not forget to update the filepath to the `"~/dominant_frequency.R"` file.

```
source("~/dominant_frequency.R")
myfun =  list(FUN=dominant_frequency,
              parameters= 30,
              expected_sample_rate= 30,
              expected_unit="g",
              colnames = c("domfreqX", "domfreqY", "domfreqZ"),
              minlength = 5,
              outputres = 5,
              outputtype="numeric",
              aggfunction = median
              timestamp=F,
              reporttype="scalar")
```

Add a call to GGIR function g.shell.GGIR where `myfun` is provided as argument. Note that, do.parallel is set to FALSE. Unfortunately Python embedding with R package reticulate and multi-threading with R package foreach as used in GGIR do not combine well.

```
library(GGIR)
g.shell.GGIR(datadir="~/myaccelerometerdata",
             outputdir="~/myresults",
             mode=1:2,
             epochvalues2csv = TRUE,
             do.report=2,
             myfun=myfun,
             do.parallel = FALSE)
```

4 Integration in GGIR output

4.1 Part 1

The external function output is included in the time series produced by function GGIR function g.part1 and stored in an RData-file in `/output_nameofstudy/meta/basic`. The resolution of these output in GGIR is set by g.shell.GGIR argument `windowsizes`, which is `c(5,900,3600)` by default. Here, the first element `5` specifies the short epoch size in seconds. If the output of the external function is less then this resolution it

will be aggregated with the function as specificied by aggfunction in the `myfun` object. In the count example we used the sum for this and for the dominant frequency example we used the median.

4.2 Part 2

Next, in part2 GGIR aims to detect non-wear periods and imputes those. The impute time series can be found in the part 2 milestone data in folder: `/output_nameofstudy/meta/ms2.out`. If you want these to be directly stored in a csv file then set argument `epochvalues2csv = TRUE`.

5 External functions released by GGIR collaborators:

- Wrist-based step detection algorithm: https://github.com/ShimmerEngineering/Verisense-Toolbox/tree/master/Verisense_step_algorithm