An Accelerometer-Based Intake-Balance Method for Assessing Energy Intake in Time Restricted Eating Trials

This vignette will walk you through sample code to execute our accelerometer-based intake balance method. It's interspersed with commentary that will hopefully make it easier to digest. You can view a web-based analog of this vignette where there are further instructions for getting help etc.

Prerequisites

Before moving forward with any other code, you need to (once only) check the following boxes to get set up:

1. Install R (required) and RStudio (optional)
2. Make sure you have the necessary R packages installed. To do that, open up R, paste the following code into the console, and hit `enter` to execute:

```r
dependencies <- c(
  "magrittr", "dplyr", "remotes", "Rcpp", "R.utils",
  "tools", "lubridate", "PAutilities", "PhysicalActivity",
  "data.table", "gsignal", "zoo"
)

sapply(
  dependencies,
  function(x) if (!x %in% installed.packages()) install.packages(x)
)

if (!"read.gt3x" %in% installed.packages()) remotes::install_github(
  "THLfi/read.gt3x", dependencies = FALSE
)

if (!"agcounts" %in% installed.packages()) remotes::install_github(
  "paulhibbing/agcounts", dependencies = FALSE
)
```

Now that you're set up

The rest of this vignette provides step-by-step code to implement the method. It is designed to let you follow along with the example of processing one file. In practice, you would likely want to set this up for batch processing multiple files. That's beyond our scope here, but not by much. The main thing you would have to do in that case is move this code into a *function* or a *for loop*, depending on your preference. For some helpful starting points, check out some information on list.files, for loops, and saveRDS.

Step 1: Reading in your data

The first step of the method requires you to read in some data, in both "raw acceleration" and "activity count" format. Below, we'll use a built in sample file that comes with the **read.gt3x** R package to show how this can be done.

```r
suppressPackageStartupMessages(library(magrittr))

# This will retrieve the existing sample file
sample_file <- system.file(
  "extdata/TAS1H30182785_2019-09-17.gt3x",
  package = "read.gt3x"
)

# Your own file might look like this:
# my_file <- "C:/users/myusername/Desktop/myfile.gt3x"

# Read the raw acceleration data (30+ Hz) and make sure timestamps
# are in UTC timezone -- Store this in an object called `AG`
accel <-
  read.gt3x::read.gt3x(sample_file, FALSE, TRUE, TRUE) %>%
  dplyr::mutate(time = lubridate::force_tz(time, "UTC"))

# Convert to activity counts (60-s epochs) and make sure timestamps
# are in UTC timezone; store this in a separate object called `counts`
# (The call to `slice` is needed because `calculate_counts` adds zeroes to the
# end of the file based on when the monitor was downloaed, whereas `read.gt3x`
# does not)
AG <-
  agcounts::calculate_counts(accel, 60, tz = "UTC") %>%
  dplyr::slice(which(time <= dplyr::last(accel$time)))
```

Step 2: Calculate minute-by-minute energy expenditure

This step breaks down into the following:

1. Calculate Euclidian Norm Minus One (ENMO) for each raw acceleration sample, rounding values < 0 up to 0
2. Calculate mean ENMO each second
3. Apply the Hildebrand non-linear oxygen consumption (VO2) equation presented by Ellingson et al. (2017)
4. Calculate the mean VO2 each minute
5. Convert the VO2 estimates (ml/kg/min) to caloric expenditure (kcal/kg/min)

Here's how we can accomplish all that:

```r
accel %<>%
  # Calculate ENMO, in milli-G
    dplyr::mutate(
      ENMO = {sqrt(X^2 + Y^2 + Z^2) - 1} %>% pmax(0) %>% {. * 1000}
    ) %>%
  # Average each second
```

```
    dplyr::group_by(time = lubridate::floor_date(time, "1 second")) %>%
    dplyr::summarise(dplyr::across(.fns = mean), .groups = "drop") %>%
  # Calculate VO2
    dplyr::mutate(VO2 = {ENMO ^ .534} %>% {0.901 * .} %>% pmax(3, .)) %>%
  # Average each minute
    dplyr::group_by(time = lubridate::floor_date(time, "1 minute")) %>%
    dplyr::summarise(dplyr::across(.fns = mean), .groups = "drop") %>%
  # Convert to kcal/kg/min
    dplyr::mutate(
      kcal_kg_min =
        (VO2 / 1000) *
        PAutilities::get_kcal_vo2_conversion(RER = 0.85, kcal_table = "Lusk")
    ) %>%
  # Remove unnecessary variables
    dplyr::select(time, kcal_kg_min)
```

Step 3: Calculate minute-by-minute non-wear

This is where the previously-calculated activity count data come into play. Here's how we calculate non-wear using the **PhysicalActivity** package (Choi et al. (2011) non-wear method).

```
AG %<>%
  PhysicalActivity::wearingMarking(
    TS = "time", cts = "Axis1", perMinuteCts = 1, tz = "UTC"
  ) %>%
  dplyr::mutate(is_nonwear = !wearing %in% "w") %>%
  dplyr::select(-c(wearing, weekday, days))
```

Step 4: Combine the energy expenditure and non-wear data

This step can be accomplished with a pretty straightforward merge.

```
AG %<>% merge(accel)
```

Step 5: Calculate daily totals

Now it is time to calculate total energy expenditure (and acceleration metrics) for each day of data. It's crucial here to exclude data from non-wear periods. To do that, we can replace the values with 0 during non-wear, so that the sum comes out correctly. We will also summarize the total number of nonwear minutes, both for screening purposes (e.g., throwing out days with 14+ hours of weartime) and for imputation of basal metabolic rate for all the nonwear minutes.

```
AG %<>%
  dplyr::group_by(time = as.Date(time)) %>%
  dplyr::mutate(dplyr::across(
    !dplyr::all_of("is_nonwear"),
    .fns = ~ ifelse(is_nonwear | is.na(.x), 0, .x)
  )) %>%
  dplyr::summarise(dplyr::across(.fns = sum), n_mins = dplyr::n()) %>%
  dplyr::rename(nonwear_mins = is_nonwear, date = time) %>%
  dplyr::relocate(date, n_mins, nonwear_mins)
```