Package: fpod (via r-universe)

November 22, 2026

Title Read and Process 'FPOD' and 'CPOD' Data

Version 1.1.0.9000

Date 2026-11-07

Description Read 'FPOD' and 'CPOD' data into 'R' directly from the 'FPOD' data files (i.e. .CP1, .CP3, .FP1 and .FP3 files). The 'FPOD' data files contain binary data, so they can't trivially be read into 'R' using the usual approach, e.g. fread() or read.csv(). This package decodes the binary data and imports all the data in one go (i.e. header/metadata, clicks, 'KERNO' classifications, environmental data and pseudo-WAV data). It is then trivial to aggregate data as you please, e.g. detection-positive-minutes per time block. The advantage of handling data processing in 'R' is a long topic, but suffice it to say that it 1) simplifies things (many fewer steps, as different vars have to be exported in multiple goes in the official 'FPOD' app), and more importantly, 2) makes data processing transparent and reproducible. References: Pirotta et al. 2014 <doi:10.1111/1365-2435.12146>.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

LinkingTo Rcpp

Imports Rcpp (>= 1.1.0), data.table

Suggests knitr, mixtools, rmarkdown, testthat (>= 3.0.0)

Depends R (>= 3.5)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/supermoan/fpod

BugReports https://github.com/supermoan/fpod/issues

Repository https://supermoan.r-universe.dev

Date/Publication 2026-11-22 08:25:17 UTC

RemoteUrl https://github.com/supermoan/fpod

RemoteRef HEAD

RemoteSha 8a5cc524e4edc5ff75be06be987ff9418931f02e

Contents

fp_example 2
fp_find_buzzes 3
fp_plot 5
fp_read 6
fp_summarize 8

Index **11**

`fp_example` *Get path to fpod example*

Description

Get path to fpod example

Usage

```
fp_example(path = NULL)
```

Arguments

`path`	Name of the file. If NULL, all example files are listed.

Value

A character vector, containing the path(s) that matched, or the empty string, "", if none matched.

Examples

```
# get a list all example files
fp_example()

# get the path for this particular file
fp_example("gullars_period1.FP3")
```

`fp_find_buzzes` *Finds harbor porpoise feeding buzzes*

Description

This function uses one of two methods to classify NBHF clicks into one of two classes: feeding buzz or non feeding buzz.

Usage

```
fp_find_buzzes(x, method = "clicks", ...)
```

Arguments

`x`	a data.table where each row is a click, as the "clicks" element in the list object returned by `fp_read()`. Each row must minimally have a POSIXct column `time`, with nanosecond precision.
`method`	the method to use to find feeding buzzes. See details.
`...`	Additional arguments passed to specific methods, when relevant. For method="threshold", you can pass `threshold=10` to define the maximum ICI for a buzz (with values interpreted in units of milliseconds)

Details

Note that the so-called "feeding buzzes" are usually considered to represent a combination of feeding buzzes and social calls. Even so, the classifications resulting from all of these methods are commonly used as a proxy of foraging activity.

The available methods are:

- `clicks` or `pirotta1` method: any inter-click-interval (ICI) less than 20 milliseconds is considered a NBHF feeding buzz.
- `trains` or `pirotta2` method: buzzes are identified using a mixture Gaussian model, with the number of components k=3. All clicks associated with the first component are considered a NBHF feeding buzz. This method requires the package `mixtools` to work. Note that different implementations of this method sometimes use different units for the ICIs (e.g. decimal minutes or milliseconds). The choice of units doesn't necessarily change the estimates, but it may affect the numerical stability of the computation and lead to false/early/non-optimal convergence. Therefore, the `fpod` package uses units of milliseconds.
- `sbfi` or `sbfi1` method: the Search Buzz Foraging Identification method described in Aofui et al 2026. Requires 3 or more consecutive ICIs between 1.25 - 6.65 ms, preceded in the same minute by a search/approach click (< 200 clicks/s). Minute groups are pre-defined based on the FPOD starting time.
- `sbfi2` method: as `sbfi`, but with a small refinement. Rather than restrict the identification of search-buzzes to pre-defined minute groups, this method uses an adaptive minute window. It first identifies clusters of buzzes, and then identifies the subset of those clusters that are preceded by a search click within 60s (disregarding minute chunks). This method is more

resilient to edge-cases, where you have a search-buzz sequence on either side of a minute (that would otherwise be missed with method="sbfi"). This affects absolute buzz numbers more than derived estimates such as buzz-positive-minutes. The latter isn't necessarily very different between these two variations of the SBFI. Note that buzzes for the edge cases always get assigned to the later minute.

- `threshold` method: any inter-click-interval (ICI) less than the defined threshold is considered a NBHF feeding buzz. The threshold must be specified in milliseconds.

Value

An integer vector of the same length as `nrow(x)`, where the values indicate that that click (or a sequence of clicks) can be considered a feeding buzz (value = 1) or not (value = 0).

References

Aoufi, S., Rumes, B., Vanaverbeke, J., Degraer, S., & Debusschere, E. (2026). A Novel Method to Infer Harbor Porpoise Foraging Behavior From Acoustic Data: A Comparative Assessment Against Established Techniques. Marine Mammal Science, 42(4), e70292. https://doi.org/10.1111/mms.70292

Pirotta, E., Thompson, P.M., Miller, P.I., Brookes, K.L., Cheney, B., Barton, T.R., Graham, I.M. and Lusseau, D. (2014), Scale-dependent foraging ecology of a marine top predator modelled using passive acoustic data. Funct Ecol, 28: 206-217. https://doi.org/10.1111/1365-2435.12146

Examples

```
# first read some FPOD data
fn <- fp_example("gullars_period1.FP3")
dat <- fp_read(fn)

# extract porpoise clicks of quality Hi and Mod
nbhf <- dat$clicks[species == "NBHF" & quality_level >= 2]

# then add a 'feeding buzz' column to the clicks data.table
nbhf$pirotta1 <- fp_find_buzzes(nbhf, method = "clicks")

# using method="clicks" is the equivalent of this:
nbhf$threshold20ms <- fp_find_buzzes(nbhf, method = "threshold", threshold = 20)

# using the gaussian mixture model approach is just as easy:
nbhf$pirotta2 <- fp_find_buzzes(nbhf, method = "pirotta2")

# or we can use one of two variations of the SBFI
nbhf$sbfi <- fp_find_buzzes(nbhf, method = "sbfi")
nbhf$sbfi2 <- fp_find_buzzes(nbhf, method = "sbfi2")

# we can easily compare all of these methods in terms of absolute buzz
# numbers and positive buzz minutes (BPMs)

# first extract only the cols that we need to do this
buzz <- nbhf[, .(minute, pirotta1, threshold20ms, pirotta2, sbfi, sbfi2)]
# define BPMs
bpm <- buzz[, lapply(.SD, function(x) any(x>0)), by = minute][,-1]
```

```
# finally, print comparative table
data.frame(
    method = names(bpm),
    tot_clicks = nrow(nbhf),
    tot_buzzes = as.integer(buzz[,lapply(.SD,sum)][,-1]),
    bpm = as.integer(bpm[,lapply(.SD,sum)])
)
```

`fp_plot` *Plot waveform and spectrum for a click*

Description

This function constructs and plots the waveform and spectrum for a click for which extra information has been recorded (IPI and SPL values for up to 21 cycles). By default, FPODs collect extra data for about 1% of detected clicks.

Usage

```
fp_plot(x, click_no = NULL, legend.pos = "topleft")
```

Arguments

`x`	A list object, as returned from `fp_read()`.
`click_no`	integer. The click number. If NULL, then all clicks are plotted, in sequential order, one at a time.
`legend.pos`	character/logical. If character; a single keyword specifying the legend position. This parameter is passed as `x` when calling `legend()`. See `legend()` for details. If legend.pos is logical and FALSE, then the legend is suppressed.

Details

According to the FPOD software guide, extra data are stored for a small percentage of clicks (about 1%). The clicks to record are selected by a "real-time train detection algorithm" running on the FPOD. The information stored is the amplitude and timing (at 250 nanosecond resolution) of the peak of the soundwave. This allows for the reconstruction of the waveform by fitting sine waves to the points stored.

Note that there's an option in the FPOD app to configure the FPOD to record `all` clicks rather than just a small sample..

Value

Invisibly returns a numeric vector with the frequency values used for plotting

See Also

fp_read()

Examples

```
# read a FP3 file
fn <- fp_example("gullars_period1.FP3")
dat <- fp_read(fn)

# get the click number of the first five clicks for which we have WAV data.
first5 <- head(unique(dat$wav$click_no))

# plot one of them. Here, we plot number three
fp_plot(dat, first5[3])

# If the legend covers the curve, we can move it somewhere else.
fp_plot(dat, first5[3], legend.pos = "topright")

# If we only want to plot clicks that meet certain criteria, we have to do that
# manually. For example, we can plot 9 random NBHF clicks:
nbhf_clicks <- dat$clicks[species == "NBHF" & has_wav == TRUE, click_no]
old.mfrow = par()$mfrow
par(mfrow = c(3,3)) # we'll do 3 by 3 panels
for (i in sample(nbhf_clicks, size = 9)) {
    fp_plot(dat, click_no = i, legend.pos = FALSE)
}
par(mfrow = old.mfrow) # reset graphics device to whatever it was before
```

---

fp_read *Read FPOD data*

Description

This function reads an FPOD or CPOD data file (FP1, FP3, CP1, CP3) into R.

Usage

```
fp_read(file, tz = "UTC", simplify = TRUE, amp = "extended", time_offset = 0)
```

Arguments

file	a character string. The path to the FPOD (or CPOD) data file.
tz	a character string. The time zone used to display times, e.g. "Europe/Oslo". Times are always calculated in UTC; tz only affects how they are displayed. Default "UTC".
simplify	logical. If TRUE, simplifies the clicks data.table by stripping away some columns, such as clk_ipi_range, ipi_pre_max, amp_reversals, duration, and has_wav.

`amp` a character string. With `amp="extended"`, higher values are extrapolated from the duration of clipping and the IPI. For any other values of amp, the compressed SPL values recorded by the FPOD are used directly.

`time_offset` numeric (seconds) or difftime. Added to all times in the file; negative values are subtracted. Use it to convert the POD clock to UTC and to correct a clock known to be wrong. For example, a clock set to Central European summer time (UTC+2) is 2 hours ahead of UTC, so use `as.difftime(-2, units = "hours")`; a clock that was 2 days slow needs `as.difftime(2, units = "days")`. Default 0 (the POD clock is assumed to run on UTC).

Details

The clicks data.frame contains the following columns:

- pod: the ID number of the pod
- time: The time and date of the click, at microsecond resolution. Note that R might only display dates and times to a second precision, but any date or time calculations will use the full precision.
- minute: minutes elapsed, since starting the FPOD
- microsec: microseconds elapsed, since the start of the minute.
- click_no: an ID number that uniquely identifies the click.
- train_id: an ID number from the KERNO classifier, reset for each minute.
- species: the species classification from the KERNO classifier
- quality_level: the quality level of the classification, 0 (?/echo), 1 (Lo), 2 (Mod) or 3 (Hi).
- echo: TRUE if the KERNO classifier thinks this click might be an echo of another click that has already been classified.
- ncyc: number of cycles in click. This is a proxy for the click duration.
- pkat: the number of the cycle with the highest amplitude
- clk_ipi_range: range of inter-peak-intervals (IPIs) among cycles in click
- ipi_pre_max: the IPI before pkat, in 250 nanosecond units
- ipi_at_max: the IPI at pkat, in 250 nanosecond units
- khz: The frequency (in kHz) of the peak cycle.
- amp_at_max: the peak amplitude (SPL) of the loudest cycle
- amp_reversals: the number of amplitude reversals
- duration: click duration
- has_wav: TRUE if there is a pseudo-WAV recorded for this click.

POD times are stored as minutes since 1899-12-30 (the Delphi/Excel convention), as read from the POD clock. `time_offset` is added to convert them to UTC, and they are then displayed in `tz`. The corrected first and last logged times are also returned as `header$first_logged_time` and `header$last_logged_time`.

A POD clock keeps the time it was set to, with no daylight saving changes, so a clock set to summer time stays on summer time through the winter. If a clock is known to have been wrong for a whole file, include the error in `time_offset` too, e.g. 2 days for a clock that was 2 days slow. See examples.

Value

A list, with one or more of the following data.frames (or data.tables, if available):

- header: a list with pod name, coordinates, starting time, stopping time, user notes, etc.
- clicks: A data.frame (or data.table) with data about each click. See details.
- wav (only FPx files): pseudo-wav data - inter-peak-intervals (in 250 ns units) and raw amplitudes for a subset of clicks
- env: misc data, angle from vertical (in degrees), ambient temperature (in deg C), battery voltage per stack (in units of volts), which battery column is in use, and the pod on/off state.

See Also

fp_find_buzzes(), fp_summarize()

Examples

```
# read a FP3 file
fn <- fp_example("gullars_period1.FP3")
dat <- fp_read(fn)

# show misc. information (pod number, deployment date, etc.)
dat$header

# a POD clock set to Norwegian summer time (UTC+2) and known to be 2 days
# slow, with times displayed in Norwegian local time
dat <- fp_read(fn, tz = "Europe/Oslo",
               time_offset = as.difftime(2, units = "days") -   # clock 2 days slow
                             as.difftime(2, units = "hours"))   # clock on UTC+2

# show battery levels and recorded temperatures for each minute
dat$env

# tally up the number of clicks in each species category
table(dat$clicks$species)
```

---

fp_summarize *Calculates minute-resolution summaries of clicks*

Description

Calculates minute-resolution summaries of clicks

Usage

```
fp_summarize(x, fpod_compat = TRUE)
```

Arguments

x	data.table where each row is a click, as the "clicks" element in the list object returned by `fp_read()`. Each row must minimally have a POSIXct column `time`. The return value of `fp_read()` is also accepted with a warning, provided that the clicks data.table is present .
fpod_compat	logical. If TRUE (the default), on-time follows the CPOD/FPOD official app: for CP3 files, every minute from the first click in x onwards counts as on, whatever the pod's tilt. Because x is usually filtered, this depends on the filter, as it does in the app. If FALSE, only minutes where the pod was probably logging (env$pod_on from `fp_read()`) count as on. That excludes minutes where the tilt exceeded the pod's switch angle, which the app counts as on once it has seen a click. For FP3 files, both settings give the same result.

Value

A data.table with three or four columns:

- time: POSIXct timestamp of the start of the 1-minute time chunk, in YYYY-mm-dd HH:MM format
- dpm: detection-positive-minutes, 1 if at least one click is registered during the time chunk; 0 otherwise.
- bpm: buzz-positive-minutes, 1 if at least one feeding buzz is registered during the time chunk, 0 otherwise. Note that bpm is only available if there is a 'buzz' column in the clicks data.table, e.g. from first calling `fp_find_buzzes()`

See Also

`fp_find_buzzes()`, `lubridate::floor_date()`

Examples

```
# first read some FPOD data
fn <- fp_example("gullars_period1.FP3")
dat <- fp_read(fn)

# extract porpoise clicks of quality Hi and Mod
nbhf <- dat$clicks[species == "NBHF" & quality_level >= 2]

# get a simple summary, with timestamp, degC and dpm.
dpm <- fp_summarize(nbhf)

# If we also wanted buzz-positive-minutes, we could do this first.
nbhf$buzz <- fp_find_buzzes(nbhf, method = "clicks")

# This time, when we call fp_summarize, it should detect that there's a
# buzz column, and automatically calculate feeding-positive-minutes as well.
dpm <- fp_summarize(nbhf)

# Once we have DPMs, we can easily aggregate these into coarser time chunks
dpm_per_hour <- dpm[, .(dpm = sum(dpm)),
```

```
    .(date = as.POSIXct(trunc(time, unit = "hours")))] # per hour
dpm_per_day <- dpm[, .(dpm = sum(dpm)),
    .(date = as.Date(time))] # per day
dpm_per_month <- dpm[, .(dpm = sum(dpm)),
    .(date = as.POSIXct(trunc(time, unit = "months")))] # per month
```

Index

difftime, *7*

fp_example, 2
fp_find_buzzes, 3
fp_find_buzzes(), *8*, *9*
fp_plot, 5
fp_read, 6
fp_read(), *3*, *5*, *6*, *9*
fp_summarize, 8
fp_summarize(), *8*

legend(), *5*
lubridate::floor_date(), *9*