gasExchangeR

The goal of gasExchangeR is to help exercise physiologist more easily analyze gas exchange data and to provide more directly control when doing so. The overview of the process of analyzing gas exchange data involves importing the data, identifying and removing outliers, interpolating if desired, and finally averaging the data. Afterwards then one can determine ventilatory thresholds, VO2max, and other important values.

Early acknowledgemets

This package borrows heavily from the work by Felipe Mattioni Maturana (https://orcid.org/0000-0002-4221-6104). Specifically, his work on the whippr and lacater packages. gasExchangeR focuses more on graded exercise testing than VO2 kinetics; it also emphasizes ventilatory breakpoint methods.

Installation

You can install the development version of gasExchangeR from GitHub with:

```
# install.packages("devtools")
devtools::install_github("ahesse2567/gasExchangeR")
```

Example

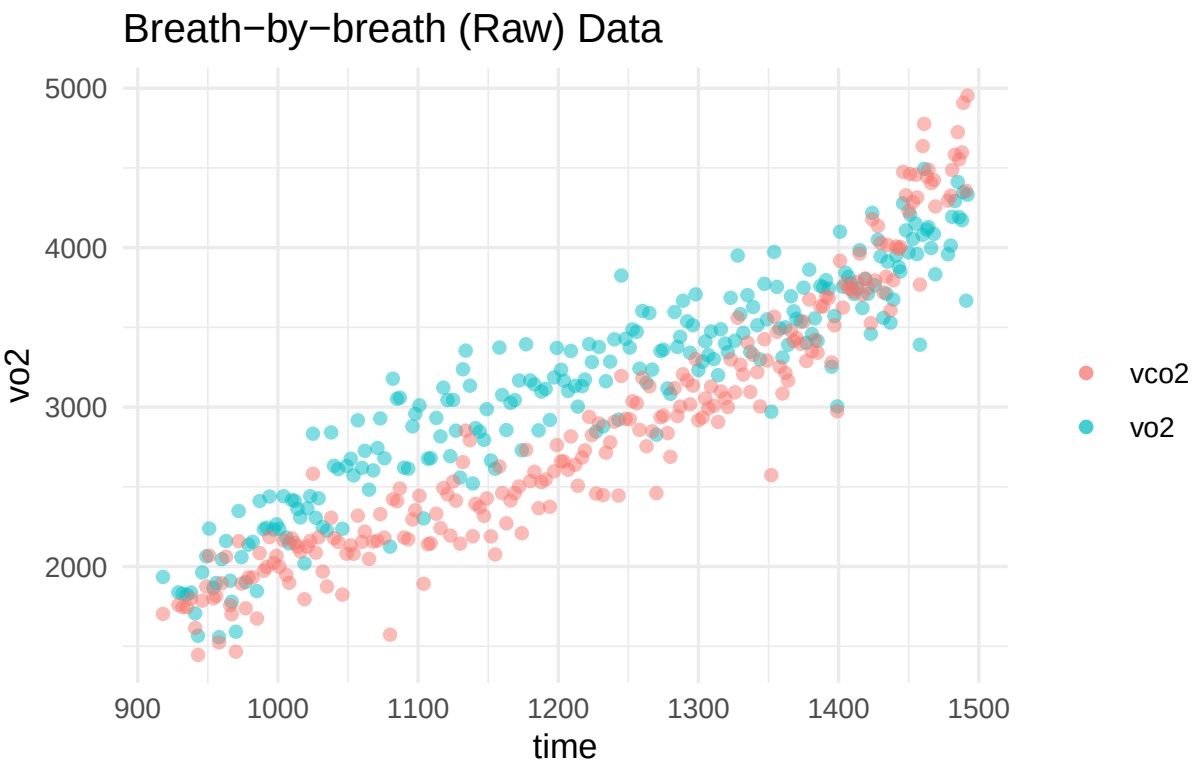
Many exercise studies require finding the first and second ventilatory thresholds (VT1 & VT2). However, breath-by-breath data is highly variable and requires some cleaning prior to finding these thresholds.

```
# Load libraries
library(gasExchangeR)
library(tidyverse)
library(janitor)
```

Data Processing

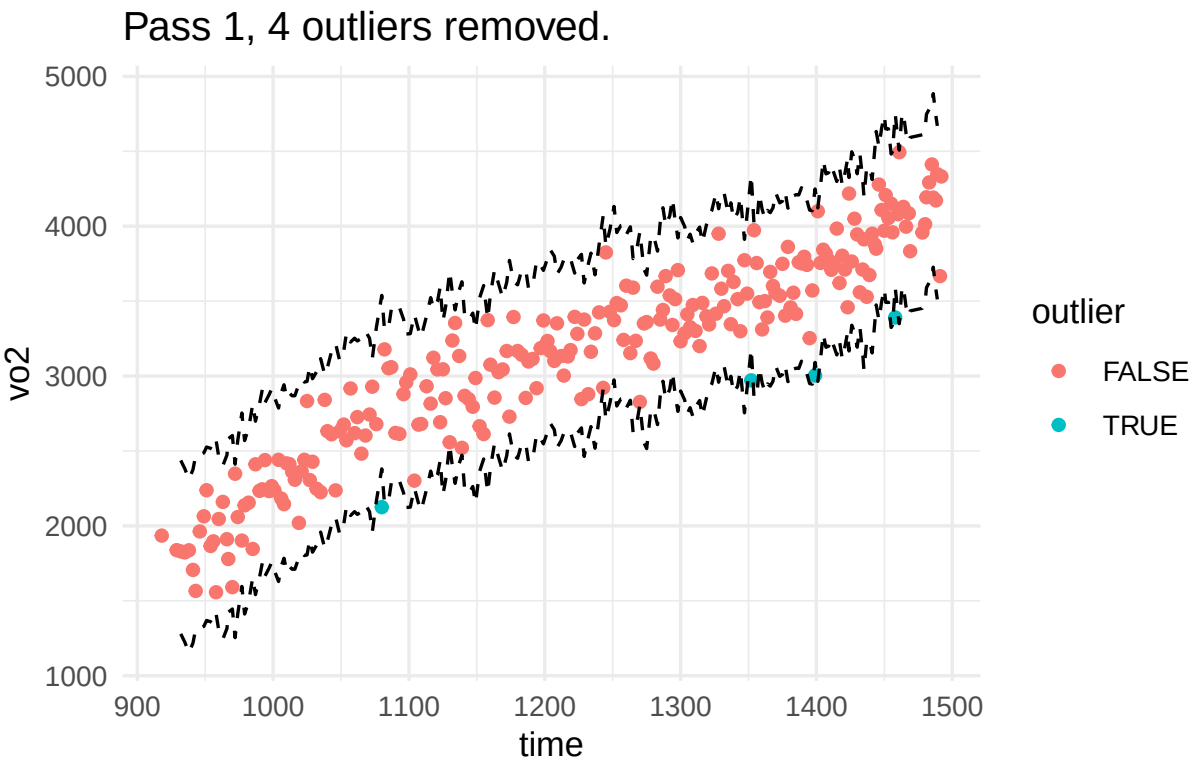
```
# read in raw data
file_lines <- readLines("inst/extdata/Anton_vo2max.txt")
df_raw <- read.table(textConnection(file_lines[-2]), header = TRUE, sep="\t")

# initial data tidying
df_unavg <- df_raw %>%
    as_tibble() %>%
    clean_names() %>%
    separate(`time`, into = c("m1", "s1"), sep = ":") %>%
    separate(ex_time, into = c("m2", "s2"), sep = ":") %>%
    separate(time_clock,
             into = c("h3", "m3", "s3"),
             sep = ":") %>%
    mutate(across(where(is.character), as.numeric)) %>%
    mutate(time = (m1*60 + s1), .keep = "unused") %>%
    mutate(ex_time = (m2*60 + s2 ), .keep = "unused") %>%
    mutate(clock_time = hms::hms(s3, m3, h3), .keep = "unused") %>%
    relocate(contains("time")) %>%
    filter(!is.na(ex_time)) %>%
    filter(speed >= 4.5 & ex_time >= 750) %>%
    select(-time) %>%
    rename(time = ex_time,
           vo2_kg = vo2,
           vo2 = vo2_1,
           ve = ve_btps) %>%
    # calculate common variables
    mutate(ve_vo2 = ve / vo2 * 1000,
           ve_vco2 = ve/vco2*1000,
           excess_co2 = vco2^2 / vo2 - vco2)
```

```
ggplot(data = df_unavg, aes(x = time)) +
  geom_point(aes(y = vo2, color = "vo2"), alpha = 0.5) +
  geom_point(aes(y = vco2, color = "vco2"), alpha = 0.5) +
  scale_color_manual(values = c("vo2" = "red", "vco2" = "blue")) +
  theme_minimal() +
  ggtitle("Breath-by-breath (Raw) Data") +
  scale_color_discrete(name = "", labels = c("vo2" = "vo2", "vco2" = "vco2"))
#> Scale for colour is already present.
#> Adding another scale for colour, which will replace the existing scale.
```



The raw data is obviously noisy. We will first use the absolute VO2 values to remove outliers.

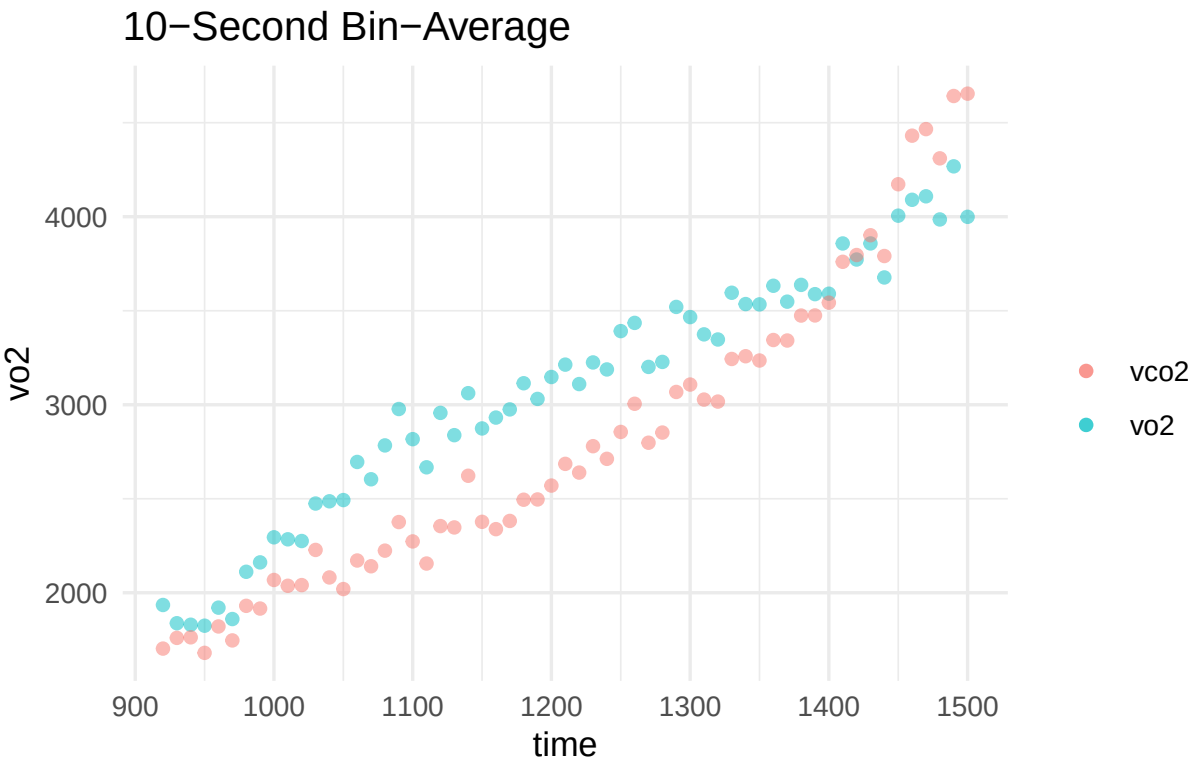
```
df_unavg_no_outliers <- df_unavg %>%
    ventilatory_outliers(outlier_cols = "vo2", max_passes = 1,
                         plot_outliers = TRUE)
```




```
#> 4 outliers removed at indicies 62, 170, 193, 225
```

Removing outliers helps, but some averaging is required,

```
df_avg <- df_unavg_no_outliers %>%
    avg_exercise_test(type = "time", subtype = "bin", bin_w = 10)

ggplot(data = df_avg, aes(x = time)) +
  geom_point(aes(y = vo2, color = "vo2"), alpha = 0.5) +
  geom_point(aes(y = vco2, color = "vco2"), alpha = 0.5) +
  scale_color_manual(values = c("vo2" = "red", "vco2" = "blue")) +
  theme_minimal() +
  ggtitle("10-Second Bin-Average") +
  scale_color_discrete(name = "", labels = c("vo2" = "vo2", "vco2" = "vco2"))
#> Scale for colour is already present.
#> Adding another scale for colour, which will replace the existing scale.
```

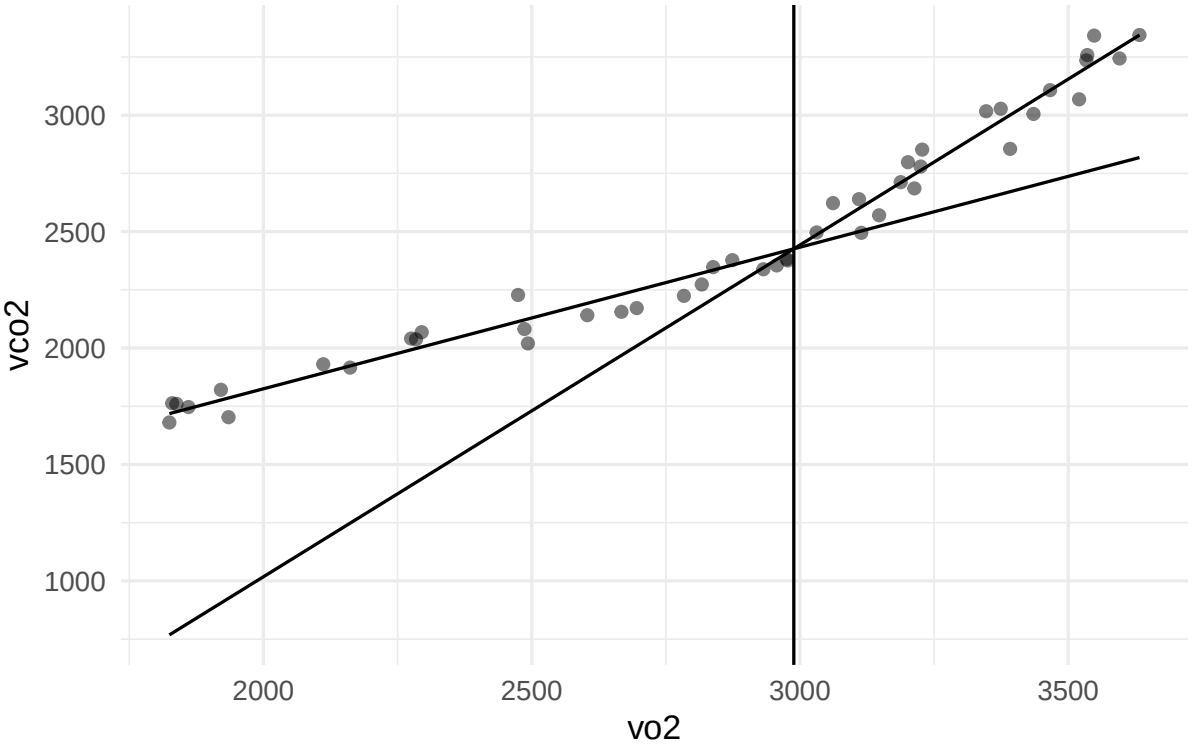


Finding Ventilatory Thresholds

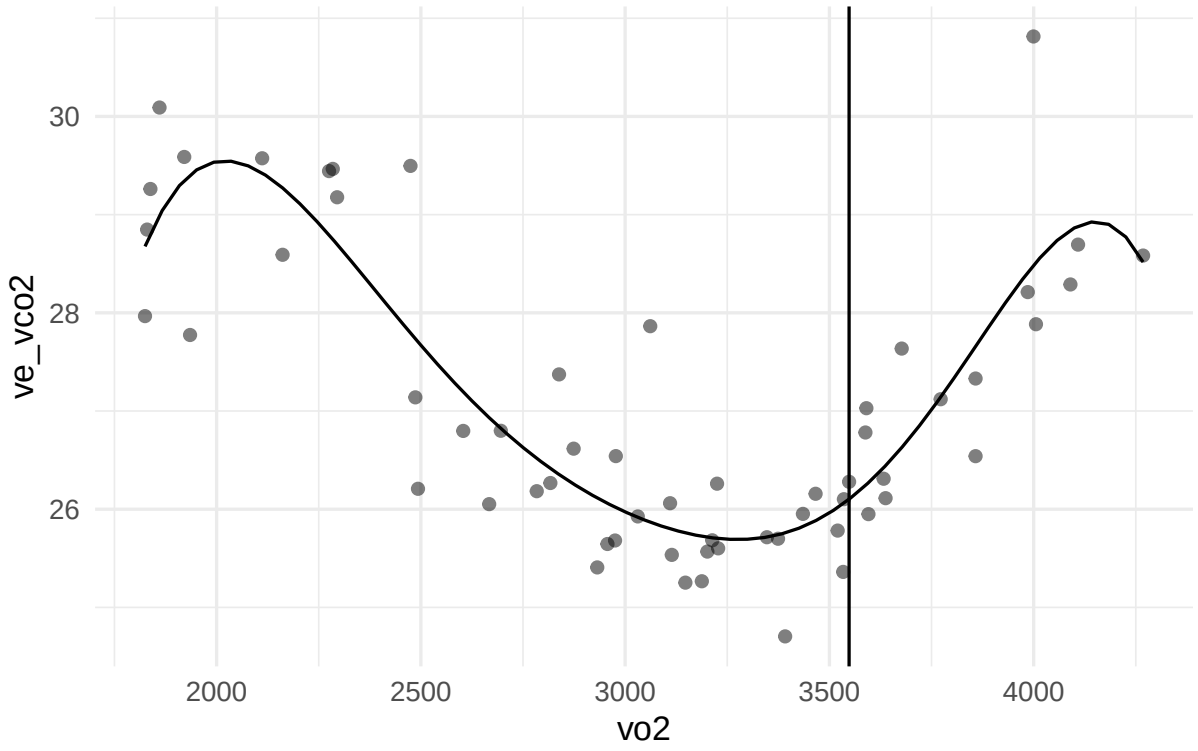
```
bp_dat <- breakpoint(.data = df_avg, method = "v-slope",
                     algorithm_vt2 = "d2_reg_spline_maxima",
                     x_vt2 = "vo2", y_vt2 = "ve_vco2",
                     vo2 = "vo2", vco2 = "vco2", ve = "ve", time = "time",
                     bp = "both", truncate = TRUE, front_trim_vt1 = 60,
                     pos_change = TRUE)
print(bp_dat$bp_dat, width = Inf)
#> # A tibble: 2 x 27
#>   bp    algorithm            x_var y_var   determinant_bp pct_slope_change
#>   <chr> <chr>                <chr> <chr>   <lgl>                      <dbl>
#> 1 vt1   v-slope              vo2   vco2    TRUE                        134.
#> 2 vt2   d2_reg_spline_maxima vo2   ve_vco2 TRUE                         NA
#>   f_stat  p_val_f  time clock_time speed grade vo2_kg   vo2  vco2   rer    rr
#>    <dbl>    <dbl> <dbl>      <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1   38.7  1.09e-9 1143.     42610.  6.72   1.1   38.1 2988. 2426. 0.804  21.6
#> 2   NA   NA       1342.     42809.  8.82   1.1   45.4 3548. 3271. 0.923  26.9
#>   vt_btps    ve    br peto2 petco2    hr   hrr ve_vo2 ve_vco2 excess_co2
```

```
#>     <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>  <dbl>   <dbl>      <dbl>
#> 1   2873.  61.9  65.1  92.8   40.9  156. 18.8    20.8    25.9      -471.
#> 2   3174.  85.4  51.9  97.7   40.8  178.  7.12   24.1    26.1      -252.
```

```
bp_dat$vt1_dat$bp_plot
```

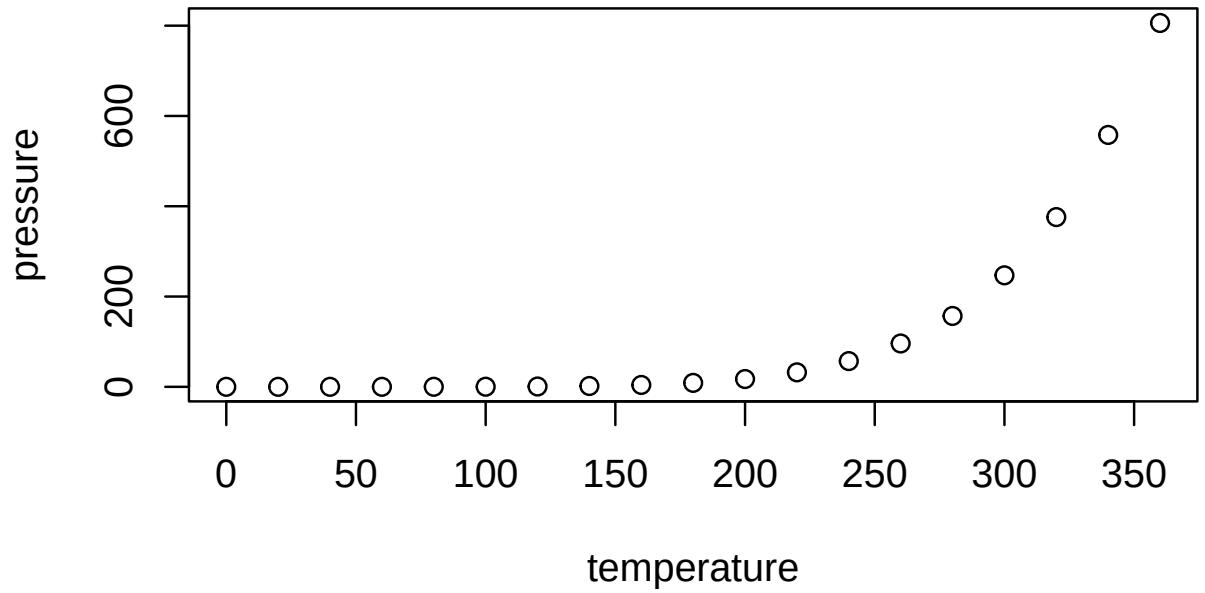



```
bp_dat$vt2_dat$bp_plot
```



You'll still need to render `README.Rmd` regularly, to keep `README.md` up-to-date. `devtools::build_readme()` is handy for this. You could also use GitHub Actions to re-render `README.Rmd` every time you push. An example workflow can be found here: https://github.com/r-lib/actions/tree/v1/examples.

You can also embed plots, for example:



In that case, don't forget to commit and push the resulting figure files, so they display on GitHub and CRAN.