Intro to stamina

Aaron Pearson

2022-02-15

stamina is a lightweight package that provides Sports Scientists the opportunity to get more out of an athlete’s positional tracking data. With this package, you can model:
* player load
* metabolic power
* V(dot)O2 max
* critical speed
* D’ balance

This package compliments {fvp} (github.com/aaronzpearson/fvp) by returning different aspects of a player’s aerobic and anaerobic abilities. Like {fvp}, the package returns modelled observation that can be plotted.

Installing the Package

To install the package, copy-and-paste the following code into your R console. The package is very small and should download quickly.

```
devtools::install_github("aaronzpearson/stamina")
library(stamina)
```

The examples rely on three other packages for efficient data cleaning and aesthetically pleasing plots. If you don’t have these packages installed on your computer, you can download them copy-and-pasting the following into your R console. You do not need to install these packages for the package to work.

```
install.packages("ggplot2")
install.packages("dplyr")
install.packages("magrittr")

library(ggplot2)
library(dplyr)
library(magrittr)
```

Package Functionality

This package was built to return as much information as possible with minimal effort for the user. As such, the end-user only needs to run the `stamina.player.profile()` and `stamina.results.model()` functions.

Note The critical speed, D’, and VO2 max models are estimates that have **not** been validated. Please use caution when tracking an athlete’s fitness levels using these models.

Note Player speed and acceleration must be in metric. It is suggested that you have speed in m/s and km/h and acceleration in m/s/s.

Function Families

To provide practitioners the ability to produce multiple analyses, functions are grouped by *family*. As such, each family of functions begins with the same prefix. Expanding on the models outlined above, the prefixes are:

- `stamina`: All-encompassing model
- `vo2`: V(dot)O2 and VO2 max
- `met`: Metabolic energy and metabolic power
- `cs`: Critical speed
- `dp`: D prime balance
- `pl`: Player load

Function Naming Conventions

For consistency, the function names (after the prefix) follow the following naming convention:

- `.player.profile`: Models a player's abilities and returns a summarized data frame
- `.results.model`: Data set containing modelled observations.

Or they are indicative of their output for the current instance. For example, `vo2.jog` returns the approximate VO2 utilization while a player moves at a jogging pace.

If you are familiar with {fvp}, you'll notice that the function naming is similar.

Sample Code

This vignette provides a brief overview of each *family* of functions in the package using sample code. As was stated above, the `stamina` functions are over-arching and can be used without worrying about the minutia.

The analyses below are completed using the data sets `jog.1` and `jog.2` which are cleaned and anonymized positional tracking data from a state-ranked cross-country skier's dryland running sessions. The data is pulled from a smart-watch that recorded data at 1Hz (1 observation per second).

```
data(jog.1)
head(jog.1)
#> # A tibble: 6 x 6
#>   altitude distance heartrate vel.kph   vel accel
#>      <dbl>    <dbl>     <dbl>   <dbl> <dbl> <dbl>
#> 1     251.        0       129       0     0     0
#> 2     251.        0       129       0     0     0
#> 3     251.        0       129       0     0     0
#> 4     252.        0       129       0     0     0
#> 5     252.        0       130       0     0     0
#> 6     252.        0       130       0     0     0
```

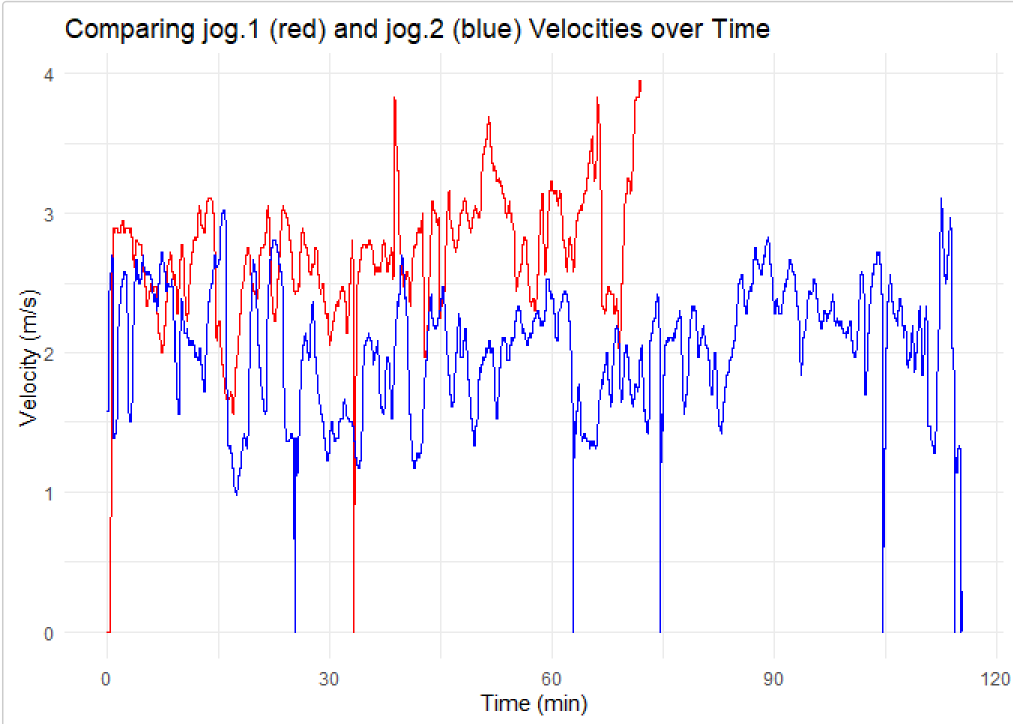
```
data(jog.2)
head(jog.2)
#> # A tibble: 6 x 5
#>   altitude heartrate vel.kph   vel accel
#>      <dbl>     <dbl>   <dbl> <dbl> <dbl>
#> 1     143.        98     5.7  1.58 0
#> 2     143.        98     5.7  1.58 0
#> 3     143.        98     5.9  1.64 0.200
#> 4     143.        99     6.3  1.75 0.400
#> 5     143.       101     6.9  1.92 0.600
#> 6     143.       102     7.5  2.08 0.600
```

To visualize the athlete's session intensity, we'll overlay the athlete's speed over time. You'll notice that `jog.1` is shorter in duration and at a greater intensity than `jog.2`. This becomes important when modelling critical speed, max aerobic speed, and VO2max.

Since the data does not come with time (these are snippets of training sessions), we'll add time before plotting. We'll also use `ggplot2` for advanced graphics.

```
jog.1$duration <- 1:nrow(jog.1)
jog.2$duration <- 1:nrow(jog.2)

ggplot() +
  geom_line(data = jog.1, aes(x = duration/60, y = vel), colour = "red") +
  geom_line(data = jog.2, aes(x = duration/60, y = vel), colour = "blue") +
  ggtitle("Comparing jog.1 (red) and jog.2 (blue) Velocities over Time") +
  xlab("Time (min)") +
  ylab("Velocity (m/s)")
```



`stamina` The complete model

The `stamina.player.profile` function is built to model the athlete's critical speed (m/s), d' (m), max aerobic speed (m/s), and VO2max (ml/kg/min).

We'll use both data sets to compare model outputs for each training session.

Note The `stamina.player.profile` returns values that have not been validated.

Note The `stamina` functions can take up to 1 minute to run the multiple models on the back-end.

```
jog.1.stamina.profile <- stamina.player.profile(speed = jog.1$vel,
                                                dur = 600, # the duration in the speed-duration model
                                                sample.rate = 1,
                                                algo = "slow") # algo is covered in the `cs` section
```

```
jog.1.stamina.profile
#>   crit.speed d.prime max.aerobic.speed vo2.max
#> 1       3.07      50              3.17   39.89
```

```
jog.2.stamina.profile <- stamina.player.profile(speed = jog.2$vel,
                                                dur = 600, # the duration in the speed-duration model
                                                sample.rate = 1,
                                                algo = "slow")
```

```
jog.2.stamina.profile
#>   crit.speed d.prime max.aerobic.speed vo2.max
#> 1       2.48      50              2.48   31.27
```

You'll notice that the outputs are different. This is because the models depend on the athlete's intensity level during the given session. If they ran at maximal efforts, we should expect to see more accurate values returned.

That said, the athlete's critical speed modelled from `jog.1` is close to reality. VO2max estimates are much lower than the athlete's actual VO2max of 69 ml/kg/min.

If you want to plot the athlete's speed, d' balance, metabolic power, V(dot)O2, etc, use the `stamina.results.model` function. The `vo2`, `crit.speed`, and `d.prime` arguments are set to `auto` which will estimate the athlete's abilities. If these values are known, input them where appropriate for better model outputs. In this example, `vo2` and `crit.speed` are set to `auto` and override `d.prime` with the athlete's known d' value.

If your data does not provide acceleration values, set `ax = NA` and `player.load = FALSE`.

```r
jog.1.stamina.model <- stamina.results.model(speed = jog.1$vel,
                                             ax = jog.1$accel,
                                             vo2 = "auto",
                                             crit.speed = "auto",
                                             d.prime = 180,
                                             player.load = TRUE,
                                             sample.rate = 1)

head(jog.1.stamina.model)
#>   duration player.speed ax ay az player.load player.load.sum met.energy
#> 1        0            0  0 NA NA           0               0      4.644
#> 2        1            0  0 NA NA           0               0      4.644
#> 3        2            0  0 NA NA           0               0      4.644
#> 4        3            0  0 NA NA           0               0      4.644
#> 5        4            0  0 NA NA           0               0      4.644
#> 6        5            0  0 NA NA           0               0      4.644
#>   met.power   vo2 vo2.kcal vo2.kcal.sum d.prime.bal
#> 1         0 4.501        0            0         180
#> 2         0 4.501        0            0         180
#> 3         0 4.501        0            0         180
#> 4         0 4.501        0            0         180
#> 5         0 4.501        0            0         180
#> 6         0 4.501        0            0         180


tail(jog.1.stamina.model)
#>      duration player.speed   ax ay az player.load player.load.sum met.energy
#> 4308     4307     3.944444  0.0 NA NA         0.0           235.0   4.644000
#> 4309     4308     3.944444  0.0 NA NA         0.0           235.0   4.644000
#> 4310     4309     3.944444  0.0 NA NA         0.0           235.0   4.644000
#> 4311     4310     3.944444  0.0 NA NA         0.0           235.0   4.644000
#> 4312     4311     3.888889 -0.2 NA NA         0.2           235.2   4.156447
#> 4313     4312     3.861111 -0.1 NA NA         0.1           235.3   4.393843
#>      met.power   vo2    vo2.kcal vo2.kcal.sum d.prime.bal
#> 4308  18.31800 49.70 0.004141667     12.12740    94.03524
#> 4309  18.31800 49.70 0.004141667     12.13154    93.16080
#> 4310  18.31800 49.70 0.004141667     12.13569    92.28635
#> 4311  18.31800 49.70 0.004141667     12.13983    91.41191
#> 4312  16.16396 49.00 0.004083333     12.14391    90.53747
#> 4313  16.96512 48.65 0.004054167     12.14796    89.71858
```

`vo2` V(dot)O2 and VO2 max

The following examples use the `jog.1` data set. I encourage you to also complete the same analyses using `jog.2`.

Estimate VO2 Utilization

The majority of `vo2` functions are built to estimate an athlete's instantaneous and cumulative VO2 utilization or the kcal equivalence. These functions take the athlete's speed (in kph). Therefore, we'll use the athlete's `vel.kph` observations.

The example below will extend the `jog.1` data set by adding instantaneous VO2 estimates. The `vo2.auto()` function selects the most-appropriate V(dot)O2 formula. You can override this by calling other `vo2` functions. Type `help(vo2.auto)` in the console to view the other functions that are available.

```r
jog.vo2 <- jog.1 # build a new data set to make sure we don't overwrite anything

jog.vo2$vo2 <- vo2.auto(speed.kph = jog.vo2$vel.kph, # returns values in ml/kg/min
                        vo2.max = 69) # VO2max is known

tail(jog.vo2)
#> # A tibble: 6 x 8
#>   altitude distance heartrate vel.kph   vel  accel duration   vo2
#>      <dbl>    <dbl>     <dbl>   <dbl> <dbl>  <dbl>    <int> <dbl>
#> 1     257.   11538.       195    14.2  3.94  0         4308  49.7
#> 2     257.   11542.       196    14.2  3.94  0         4309  49.7
#> 3     257.   11546.       195    14.2  3.94  0         4310  49.7
```

```
#> 4     257.   11550.       196    14.2  3.94  0          4311  49.7
#> 5     257.   11553.       196    14    3.89 -0.200      4312  49
#> 6     257.   11555.       196    13.9  3.86 -0.100      4313  48.6
```

kcal Equivalent

For this example, `jog.vo2$vo2` will be used to return estimated instantaneous and cummulative caloric expenditure in kcal/kg.

```
jog.vo2$kcal.vo2 <- vo2.kcal.dist(speed = jog.vo2$vel.kph,
                                  sample.rate = 1)
jog.vo2$kcal.vo2.sum <- cumsum(jog.vo2$kcal.vo2) # cumulative summation
```

```
tail(jog.vo2)
#> # A tibble: 6 x 10
#>   altitude distance heartrate vel.kph   vel  accel duration   vo2 kcal.vo2
#>      <dbl>    <dbl>     <dbl>   <dbl> <dbl>  <dbl>    <int> <dbl>    <dbl>
#> 1     257.   11538.       195    14.2  3.94  0          4308  49.7  0.00414
#> 2     257.   11542.       196    14.2  3.94  0          4309  49.7  0.00414
#> 3     257.   11546.       195    14.2  3.94  0          4310  49.7  0.00414
#> 4     257.   11550.       196    14.2  3.94  0          4311  49.7  0.00414
#> 5     257.   11553.       196    14    3.89 -0.200      4312  49    0.00408
#> 6     257.   11555.       196    13.9  3.86 -0.100      4313  48.6  0.00405
#> # ... with 1 more variable: kcal.vo2.sum <dbl>
```

The cumulative VO2 kcal of 12.15 is not the kcalories that the athlete expeneded. The value is returned as kcal/kg. Therefore, with this athlete weighting approximately 55 kg, their total expenditure is 55*12.15 = 668.25 kcal. To add context, this session is 70 minutes which resulted in ~670kcal expended (which makes sense).

Plotting

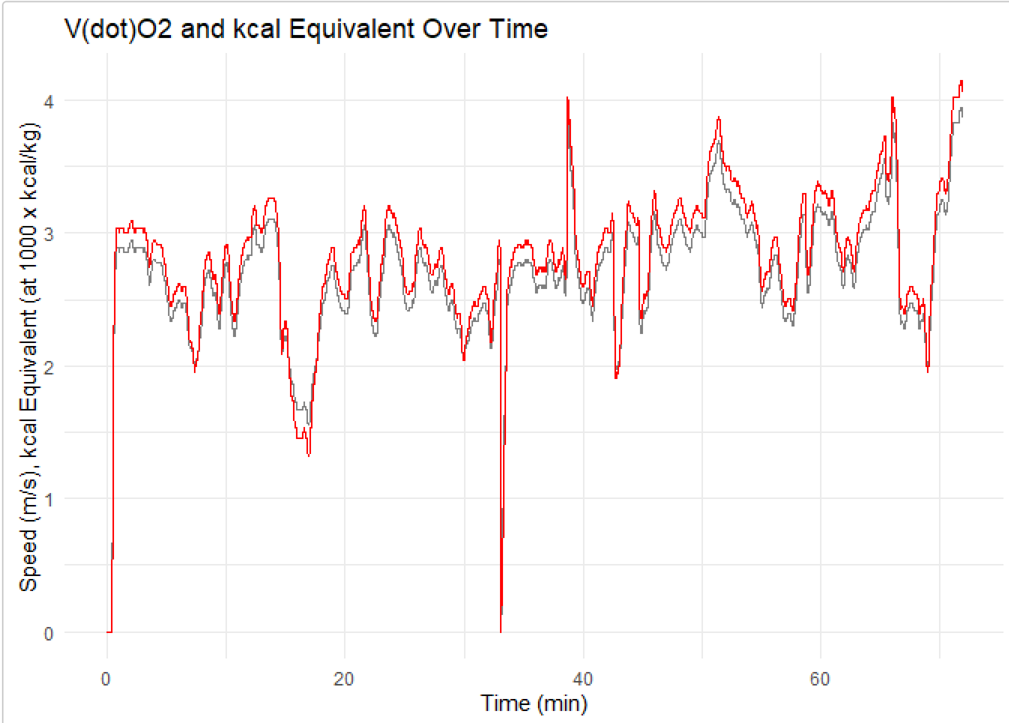
Below, we overlay V(dot)O2 kcal and the athlete's speed in m/s. As would be expected, V(dot)O2 kcal mirrors changes in the athlete's intensity level.

Note The athlete's V(dot)O2 kcal is multiplied by 1000 for improved visualizations.

```
p.vo2 <- ggplot(data = jog.vo2, aes(duration/60)) +
  # geom_point(aes(y = vo2), colour = "grey", alpha = 0.5) +
  geom_line(aes(y = vel), colour = "black", alpha = 0.5) +
  geom_line(aes(y = kcal.vo2 * 1000), colour = "red") +

  xlab("Time (min)") +
  ylab("Speed (m/s), kcal Equivalent (at 1000 x kcal/kg)") +
  ggtitle("V(dot)O2 and kcal Equivalent Over Time")

p.vo2
```



`met` Metabolic power and energy

Metabolic power and energy functions are similar to those in the `vo2` family by taking on a player's instantaneous rate acceleration.

```
jog.met <- jog.1 # so we don't overwrite jog.1

jog.met$met.energy = met.energy(accel = jog.met$accel)
jog.met$met.power = met.power(speed = jog.met$vel,
                              accel = jog.met$accel)


head(jog.met)
#> # A tibble: 6 x 9
#>   altitude distance heartrate vel.kph   vel accel duration met.energy met.power
#>      <dbl>    <dbl>     <dbl>   <dbl> <dbl> <dbl>    <int>      <dbl>     <dbl>
#> 1     251.        0       129       0     0     0        1       4.64         0
#> 2     251.        0       129       0     0     0        2       4.64         0
#> 3     251.        0       129       0     0     0        3       4.64         0
#> 4     252.        0       129       0     0     0        4       4.64         0
#> 5     252.        0       130       0     0     0        5       4.64         0
#> 6     252.        0       130       0     0     0        6       4.64         0
```

The data set returns metabolic energy < 0, which is not possible. Therefore, we must remove these observations.

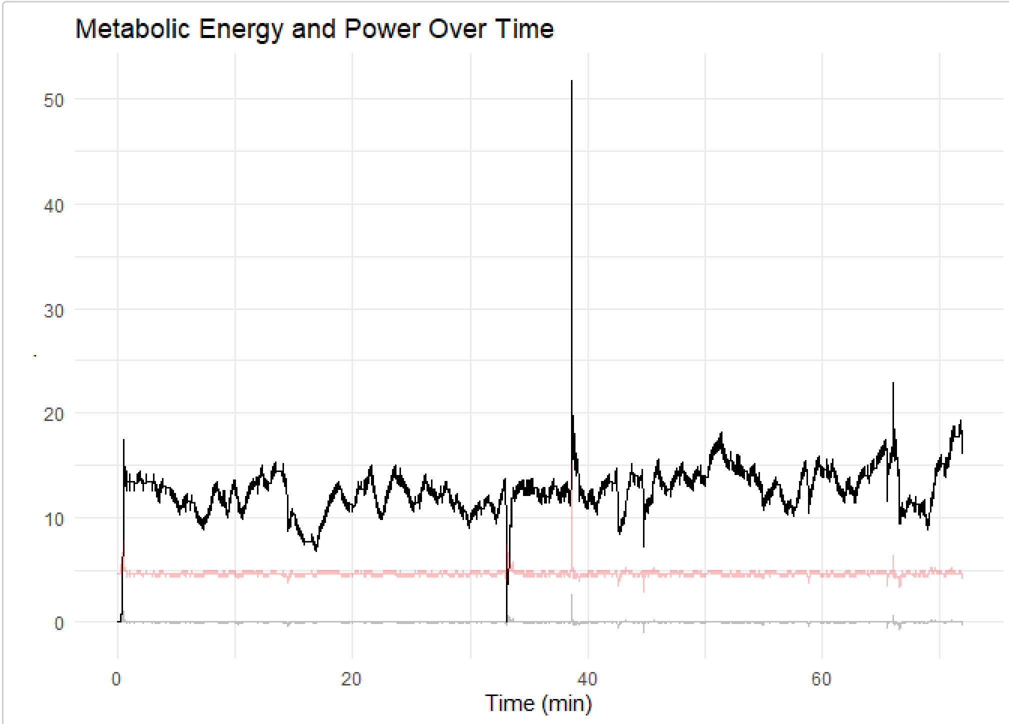
```
jog.met <- jog.met %>%
  filter(met.energy > 0)
```

Plotting

```
p.met <- ggplot(data = jog.met, aes(x = duration/60)) +
  geom_line(aes(y = accel), colour = "grey") + # very little deviations in speed results in minimal
         acceleration changes
  geom_line(aes(y = met.power), colour = "black") +
  geom_line(aes(y = met.energy), colour = "red", alpha = 0.25) +

  xlab("Time (min)") +
  ylab(".") +
  ggtitle("Metabolic Energy and Power Over Time")
```

```
p.met
```



`cs` Critical Speed

The `cs` family can be used solo or in conjunction with the `dp` family. `cs.results.model` returns the athlete's speed-duration curve for the session. The algorithm (`algo`) argument can be set to `fast` or `slow`. If your data set is large, setting it to `fast` will significantly speed up the process but return a very rough model output.

We'll compare the `fast` and `slow` data sets below.

```
jog.cs <- jog.1

jog.cs.model.slow <- cs.results.model(speed = jog.1$vel, # this took ~10 s on my local machine
                                      algo = "slow",
                                      sample.rate = 1,
                                      dur = 600) # model over 600 s

jog.cs.model.fast <- cs.results.model(speed = jog.1$vel,
                                      algo = "fast",
                                      sample.rate = 1,
                                      dur = 600)
```

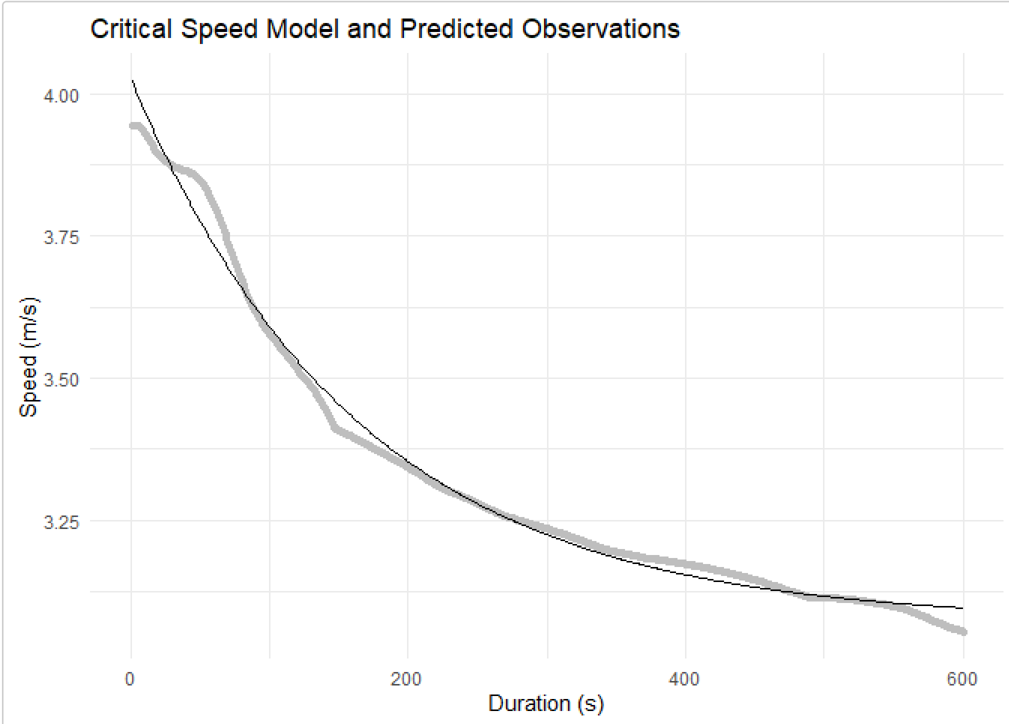
Plotting

Critical speed observations versus modelled data:

```
p.cs <- ggplot(jog.cs.model.slow, aes(x = dur)) +
  geom_point(aes(y = speed), colour = "grey") +
  geom_line(aes(y = pred), colour = "black") +

  ggtitle("Critical Speed Model and Predicted Observations") +
  ylab("Speed (m/s)") +
  xlab("Duration (s)")

p.cs
```



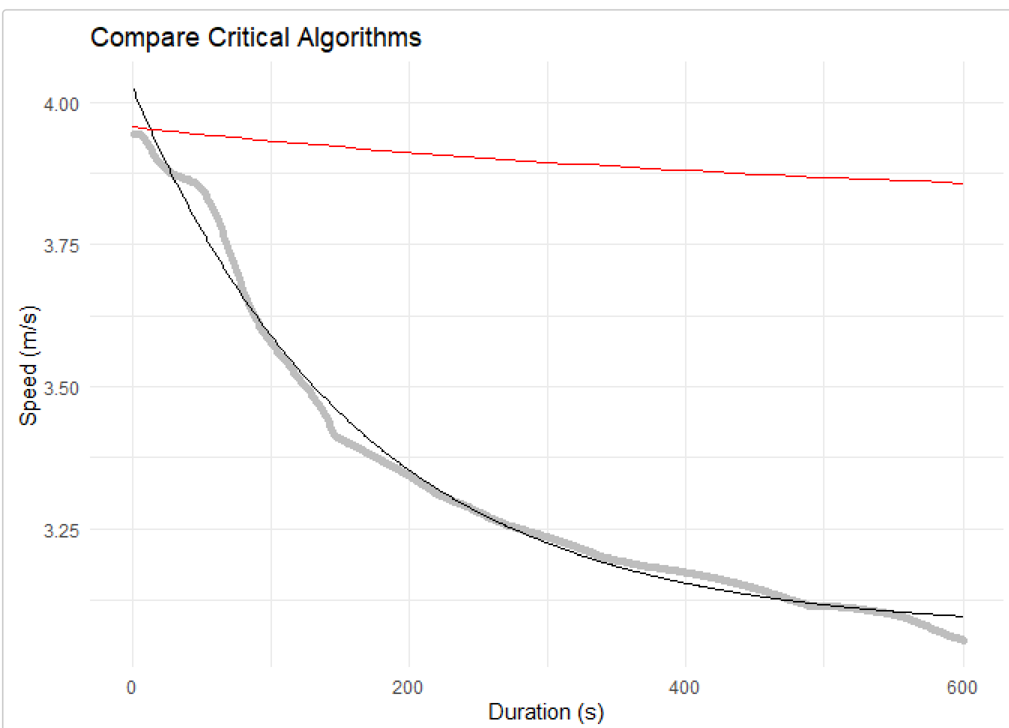
Comparing algorithms:

As you can see, there is a stark contrast between algorithms!

```
p.cs.compare <- ggplot(jog.cs.model.slow, aes(x = dur)) +
  geom_point(data = jog.cs.model.slow, aes(y = speed), colour = "grey") +
  geom_line(data = jog.cs.model.slow, aes(y = pred), colour = "black") +
  geom_line(data = jog.cs.model.fast, aes(y = pred), colour = "red") +

  ggtitle("Compare Critical Algorithms") +
  ylab("Speed (m/s)") +
  xlab("Duration (s)")

p.cs.compare
```



`dp` D Prime Balance

The `dp` family of functions are limited and are mainly used as helper functions and rely on the `dp.bal()` function. Since the model is nowhere near perfect, D' balance can drop below 0. A lower limit of -50 m is set and can be adjusted. If your values are regularly returned below 0, the critical speed and d' estimates require attention.

In the example below, I set the athlete's critical speed and d' below their actual abilities to for improved visualization.

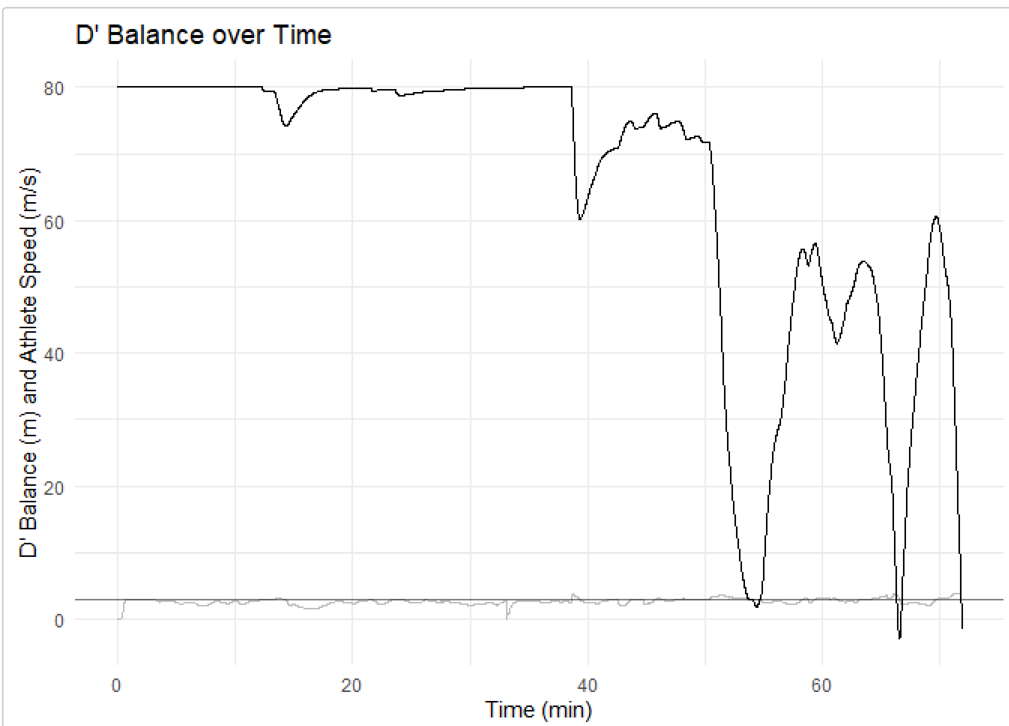
```
jog.dp <- jog.1

jog.dp$d.bal <- dp.bal(speed = jog.1$vel,
                       cs = 3,
                       d0 = 80,
                       t.diff = 1,
                       d.bal.min = -50)
```

Plotting

```
p.dp <- ggplot(data = jog.dp, aes(x = duration/60)) +
  geom_line(aes(y = vel), colour = "grey") +
  geom_hline(aes(yintercept = 3), alpha = 0.5) +
  geom_line(aes(y = d.bal)) +
  ggtitle("D' Balance over Time") +
  xlab("Time (min)") +
  ylab("D' Balance (m) and Athlete Speed (m/s)")

p.dp
```



`pl` Player Load

The `pl` functions are built to emulate the Player Load metric introduced by Catapult Sports(R). This function deviates by providing the option of including acceleration in 1, 2, or 3 planes of motion. This was done to provide practitioners with limited acceleration observations in their player tracking data the ability to model Player Load.

Player load can be returned as an instantaneous or cumulative value.

```r
jog.pl <- jog.1

jog.pl$player.load <- pl.player.load(ax = jog.pl$accel, # this data set doesn't have ay or az
                                     cumulative = FALSE, # ay and az are optional, so you can leave
        them out
                                     sample.rate = 1)

jog.pl$player.load.sum <- pl.player.load(ax = jog.pl$accel,
                                         cumulative = TRUE,
                                         sample.rate = 1)
```

Plotting

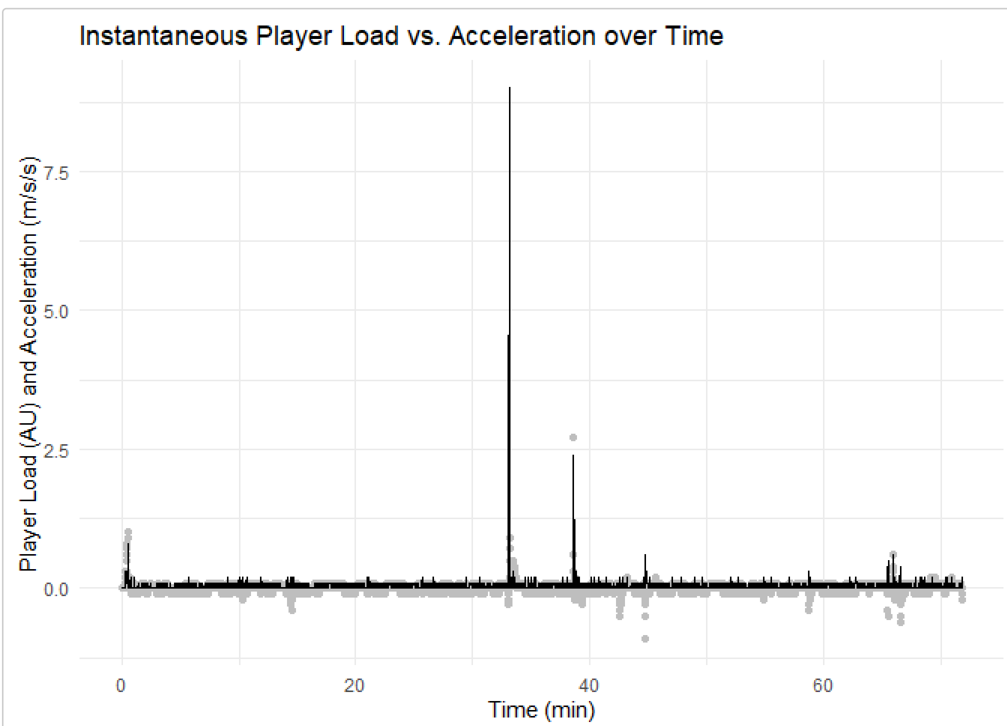
For this example, we'll remove deceleration below -1 m/s/s to improve visualization.

```r
jog.pl <- jog.pl %>%
  filter(accel >= -1)

p.pl <- ggplot(jog.pl, aes(x = duration/60)) +
  geom_point(aes(y = accel), colour = "grey") +
  geom_line(aes(y = player.load)) +

  ggtitle("Instantaneous Player Load vs. Acceleration over Time") +
  xlab("Time (min)") +
  ylab("Player Load (AU) and Acceleration (m/s/s)")

p.pl
```

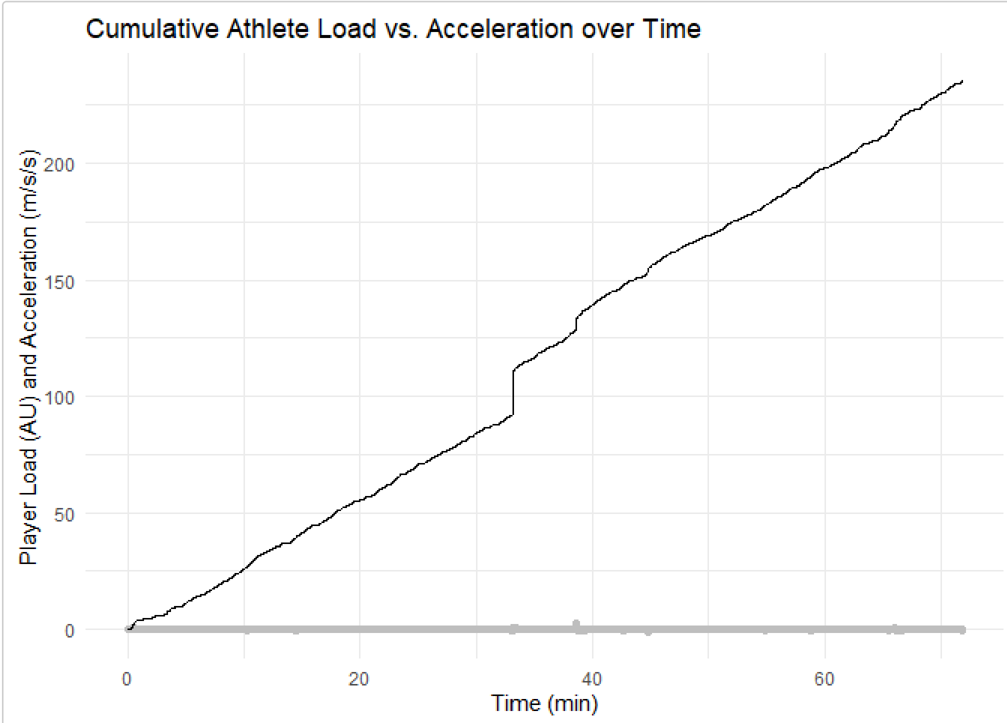


Cumulative player load over time:

```r
p.pl.sum <- ggplot(jog.pl, aes(x = duration/60)) +
  geom_point(aes(y = accel), colour = "grey") +
  geom_line(aes(y = player.load.sum)) +

  ggtitle("Cumulative Athlete Load vs. Acceleration over Time") +
  xlab("Time (min)") +
  ylab("Player Load (AU) and Acceleration (m/s/s)")

p.pl.sum
```


Cumulative Athlete Load vs. Acceleration over Time
Player Load (AU) and Acceleration (m/s/s)
0
50
100
150
200
0
20
40
60
Time (min)