# Introduction to fvp

Aaron Pearson

2021-12-23

The fvp package was written to extend the functionality of its predecessory, *midsprint*. Users can use this package to model and athlete's sprint- and jump-based force-velocity profile. These models include:

- Sprint abilities modelled over time and distance
- In-game speed-acceleration model
- Sprint test models using distance and time splits
- Jump-based force-velocity model

Once a player's abilities are modelled, the user can opt to return a data set that encompasses their modelled force-velocity-power profile.

Since this package is built to provide practitioners modelled observations, the package does not support plotting functions like those in *midsprint*. As such, *midsprint* will be updated to include these reporting functions with *fvp* providing the back-end analyses.

## Topics Covered

1. Installing the Package
2. Package Functionality
3. Sample Code
4. Extending the Package
5. Citing fvp

## 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.

```r
devtools::install_github("aaronzpearson/fvp")
library(fvp)
```

The plotting examples rely on two other packages to return 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.

```r
install.packages("ggplot2")
install.packages("patchwork")

library(ggplot2)
library(patchwork)
```

# Package Functionality

**Note** The models have been validated using values in metric (m/s, m/s/s, etc.). To convert your current values to metric, use the `convert.to.metric()` function. This function can be applied to multiple variables effectively using a function like `apply()` from base R or `mutate()` from the *dplyr* package.

## 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:

- `gps`: Sprint abilities modelled over time and distance
- `sa`: In-game speed-acceleration model
- `scout`: Sprint test models using distance and time splits
- `fv`: Jump-based force-velocity model
- `fvp`: Modelled force-velocity-power profile from sprint models

## Function Naming Conventions

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

- `.data`: Cleaned and formatted speed and acceleration observations
- `.data.player`: A player's anthropomorphic data and weather conditions
- `.data.testing`: Supplemental testing information like load and athlete testing results
- `.player.profile`: Models a player's abilities and returns a summarized data frame
- `.player.profile.game`: Unique to the *gps* family, returns a player's observed sprint abilities
- `.player.splits`: Speed, acceleration, and time at distinct distances
- `.results.model`: Data set containing modelled observations
- `.results.observed`: Data set containing observations that are used to model a player's abilities
- `.results.game`: Unique to the *gps* family, returns a player's modelled in-game abilities
- `.results.fitted`: Unique to the *sa* family, returns the data that were maintained to fit the linear model

There are supplementary functions that are also included that do not belong to a particular family. These generic functions can be applied to any family of functions.

# Sample Code

This vignette provides sample code for all five modelling families. Although some steps seem redundant, it is important to note that similar functions that are in different families often work differently on the back-end. It is highly recommended to use the functions within the same family when working on a given model.

Every family starts with the `.data` function to clean and process the data. Unlike *midsprint*, the modelled results do not need practitioners to input a player's profile. It was decided that removing the need to build player profiles for subsequent analyses minimize the time needed to model a player's abilities.

The analyses below are completed using sample data. player.one is tracking data, player.forty is a player's 40 yard dash sprint test, and player.jump is a player's jump-test results.

```
head(player.one)
#> # A tibble: 6 x 2
#>   speed accel
#>   <dbl> <dbl>
#> 1  0.02  0.03
#> 2  0.03  0.03
#> 3  0.02  0.03
#> 4  0.02  0.02
#> 5  0.02  0.02
#> 6  0.01  0.02
```

```
player.forty <- data.frame(distance = c(9.14, 18.3, 36.6),
                           split.time = c(1.66, 2.67, 4.72))
player.forty
#>   distance split.time
#> 1     9.14       1.66
#> 2    18.30       2.67
#> 3    36.60       4.72
```

The arguments `trial.one`, `trial.two`, and `trial.three` are for every condition. That is, `trial.one` should contain the results for all of the player's first trial results for all conditions. To avoid confusion, the `trial.` arguments can take-on a vector that contains all of the first trial results. Therefore, the user is encouraged to put all results into a table and set the `trial.` arguments as something like `player.one$jump.trial.one`.

```
player.jump <- data.frame(percent.bw = c(0, 25, 50, 75),
                          add.mass = c(0, 20.6, 41.2, 61.8),
                          trial.one = c(0.34, 0.29, 0.21, 0.18),
                          trial.two = c(0.35, 0.28, 0.19, 0.18),
                          trial.three = c(0.34, 0.29, 0.20, 0.17))
player.jump
#>   percent.bw add.mass trial.one trial.two trial.three
#> 1          0      0.0      0.34      0.35        0.34
#> 2         25     20.6      0.29      0.28        0.29
#> 3         50     41.2      0.21      0.19        0.20
#> 4         75     61.8      0.18      0.18        0.17
```

## `gps.` Player Tracking

### Clean and Process

```
player.one.gps <- gps.data(player.one$speed, player.one$accel)
head(player.one.gps)
#>   game.speed game.accel
#> 1       0.02       0.03
#> 2       0.03       0.03
#> 3       0.02       0.03
#> 4       0.02       0.02
#> 5       0.02       0.02
#> 6       0.01       0.02
```

### Player Profile

`gps.player.profile()` returns the player’s name and their potential max speed, max acceleration, and acceleration constant.

```
player.one.gps.profile <- gps.player.profile(player.name = "Player 1 GPS",
                                             game.data = player.one.gps)
player.one.gps.profile
#>    player.name max.speed max.accel player.tau
#> 1 Player 1 GPS     10.39      15.6  0.6660256
```

`gps.player.profile.game()` return’s the player’s name and their observed max speed, max acceleration, and acceleration constant.

```
player.one.game.profile <- gps.player.profile.game(player.name = "Player 1 GAME",
                                                   game.data = player.one.gps)
player.one.game.profile
#>     player.name max.speed max.accel player.tau
#> 1 Player 1 GAME  9.534202  7.155227   1.228297
```

### Modelling

`gps.results.model()` returns the player modelled potential sprint ability

```
player.one.gps.model <- gps.results.model(game.data = player.one.gps,
                                          sample.rate = 10)
head(player.one.gps.model)
#>   splits model.speed model.acceleration model.distance
#> 1    0.0    0.000000          15.600000     0.00000000
#> 2    0.1    1.448535          13.425106     0.07423849
#> 3    0.2    2.695121          11.553428     0.28298043
#> 4    0.3    3.767912           9.942692     0.60747385
#> 5    0.4    4.691139           8.556519     1.03158113
#> 6    0.5    5.485653           7.363601     1.54141445
```

`gps.results.observed()` returns the observations that make-up the player’s best sprint. The function includes the arguments `min.speed` and `max.speed.threshold`. `min.speed` is the speed that the function should consider the start of the sprint. Since players are constantly moving, a velocity of 0 m/s is not realistic. Therefore, a minimum speed of 0.3 m/s is pre-set as the starting speed of a player’s sprint. `max.speed.threshold` is the percent of max speed in the data set that should be considered the end of the sprint. This is to provide some flexibility when trying to isolate a player’s on-field maximal effort. Since players reach max speed once or twice a game, it is unrealistic to have the `max.speed.threshold` set to 100%.

This function returns the observations that make up the players fastest time from their `min.speed` to their `max.speed.threshold`.

```
player.one.gps.observed <- gps.results.observed(game.data = player.one.gps,
                                                min.speed = 0.3,
                                                max.speed.threshold = 95,
                                                sample.rate = 10)
head(player.one.gps.observed)
#>   split.time observed.speed
#> 1        0.0           0.00
#> 2        0.1           0.22
#> 3        0.2           0.59
#> 4        0.3           0.96
#> 5        0.4           1.50
#> 6        0.5           2.01
```

`gps.results.game()` returns the player’s modelled actual sprint ability.

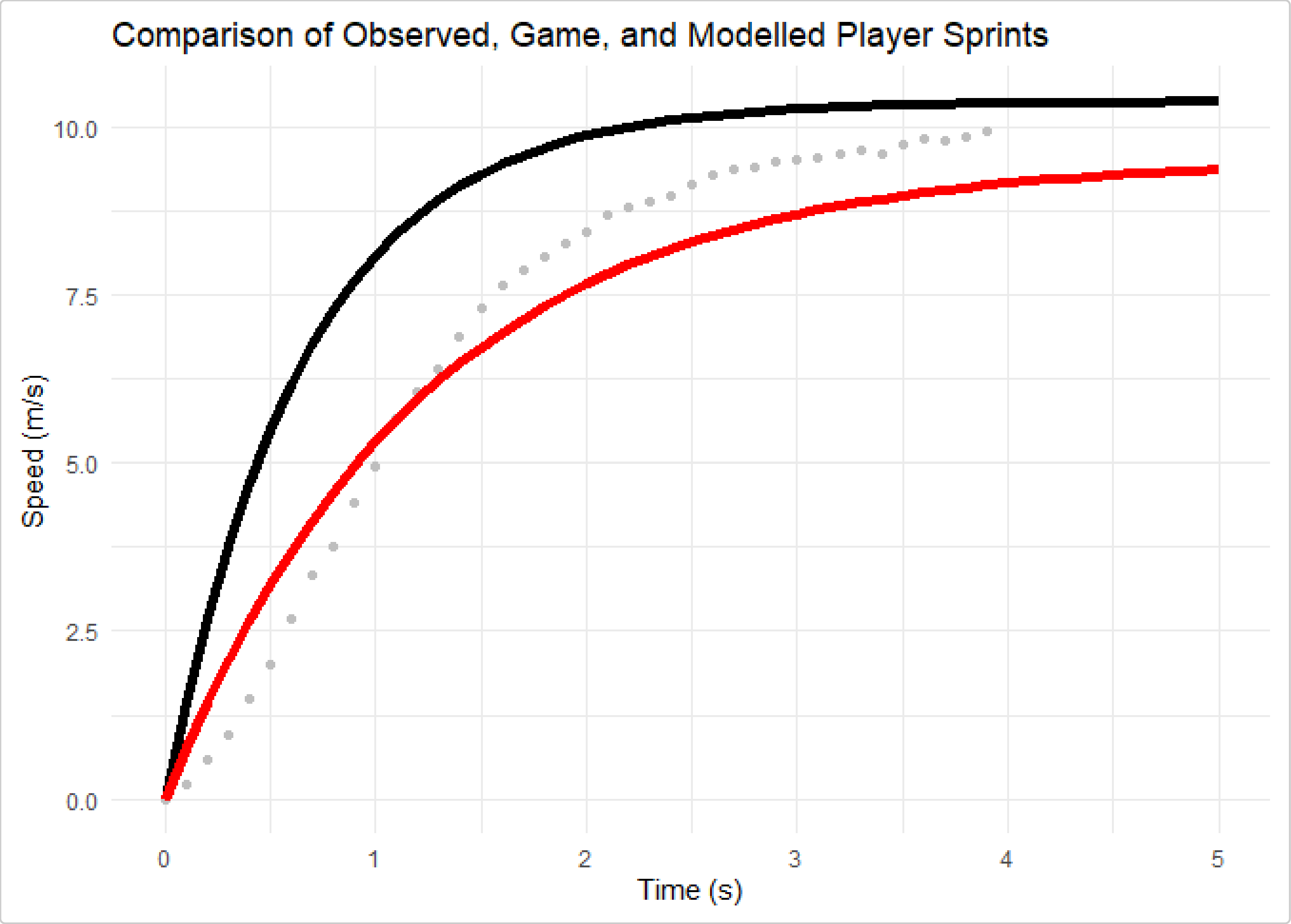
```
player.one.gps.game <- gps.results.game(game.data = player.one.gps,
                                        sample.rate = 10)
head(player.one.gps.game)
#>   splits model.speed model.acceleration model.distance
#> 1    0.0   0.0000000           7.762129      0.0000000
#> 2    0.1   0.7454561           7.155227      0.0377785
#> 3    0.2   1.4326267           6.595777      0.1471488
#> 4    0.3   2.0660692           6.080069      0.3225133
#> 5    0.4   2.6499843           5.604683      0.5587121
#> 6    0.5   3.1882446           5.166467      0.8509887
```

### Plotting

Below is an example of the plots users can create from the modelled data. This plot compares the player’s potential, observed, and modelled best-sprint abilities.

```
p1 <- ggplot() +

  geom_point(data = player.one.gps.observed, # observed best-sprint
             aes(x = split.time, y = observed.speed),
             colour = "grey") +

  geom_line(data = player.one.gps.model, # player potential
            aes(x = splits, y = model.speed),
            size = 2,
            colour ="black") +

  geom_line(data = player.one.gps.game, # modelled best sprint
            aes(x = splits, y = model.speed),
            size = 2,
            colour = "red") +

  xlab("Time (s)") +
  ylab("Speed (m/s)") +
  ggtitle("Comparison of Observed, Game, and Modelled Player Sprints")
```

```
p1
```



## `sa.` Speed-Acceleration

### Clean and Process

```
player.one.sa <- sa.data(player.one$speed, player.one$accel)
head(player.one.sa)
#>   game.speed game.accel
#> 1       0.02       0.03
#> 2       0.03       0.03
#> 3       0.02       0.03
#> 4       0.02       0.02
#> 5       0.02       0.02
#> 6       0.01       0.02
```

### Player Profile

`sa.player.profile()` returns the player's name and their max speed, max acceleration, acceleration constant, number of observations maintained to build the model, and the r^2 values of the linear model. The r^2 can be adjusted by the user.

```
player.one.sa.profile <- sa.player.profile(player.name = "Player 1 SA",
                                           game.data = player.one.sa,
                                           r2 = 0.95)
player.one.sa.profile
#>   player.name max.speed max.accel player.tau  r.square n.obervation
#> 1 Player 1 SA  12.39248  10.51855   1.178155 0.9535664           45
```

## Modelling

`sa.results.model()` returns the player modelled potential sprint ability

```
player.one.sa.model <- sa.results.model(game.data = player.one.sa,
                                        r2 = 0.95)
head(player.one.sa.model)
#>   game.speed game.accel
#> 1       0.00   10.51855
#> 2       0.01   10.51006
#> 3       0.02   10.50157
#> 4       0.03   10.49308
#> 5       0.04   10.48459
#> 6       0.05   10.47611
```

`sa.results.fitted()` returns the observations onto which the speed-acceleration model is built.

```
player.one.sa.fitted <- sa.results.fitted(game.data = player.one.sa,
                                          r2 = 0.95)
head(player.one.sa.fitted)
#>   game.speed game.accel speed.bins fit.predict  residual
#> 1       3.07       7.96    (3,3.2]    7.920347 0.0396530
#> 2       3.43       7.09  (3.4,3.6]    7.615954 0.5259542
#> 3       3.48       6.70  (3.4,3.6]    7.573677 0.8736774
#> 4       3.75       6.66  (3.6,3.8]    7.345383 0.6853827
#> 5       3.64       6.59  (3.6,3.8]    7.438392 0.8483917
#> 6       3.92       7.42    (3.8,4]    7.201642 0.2183583
```

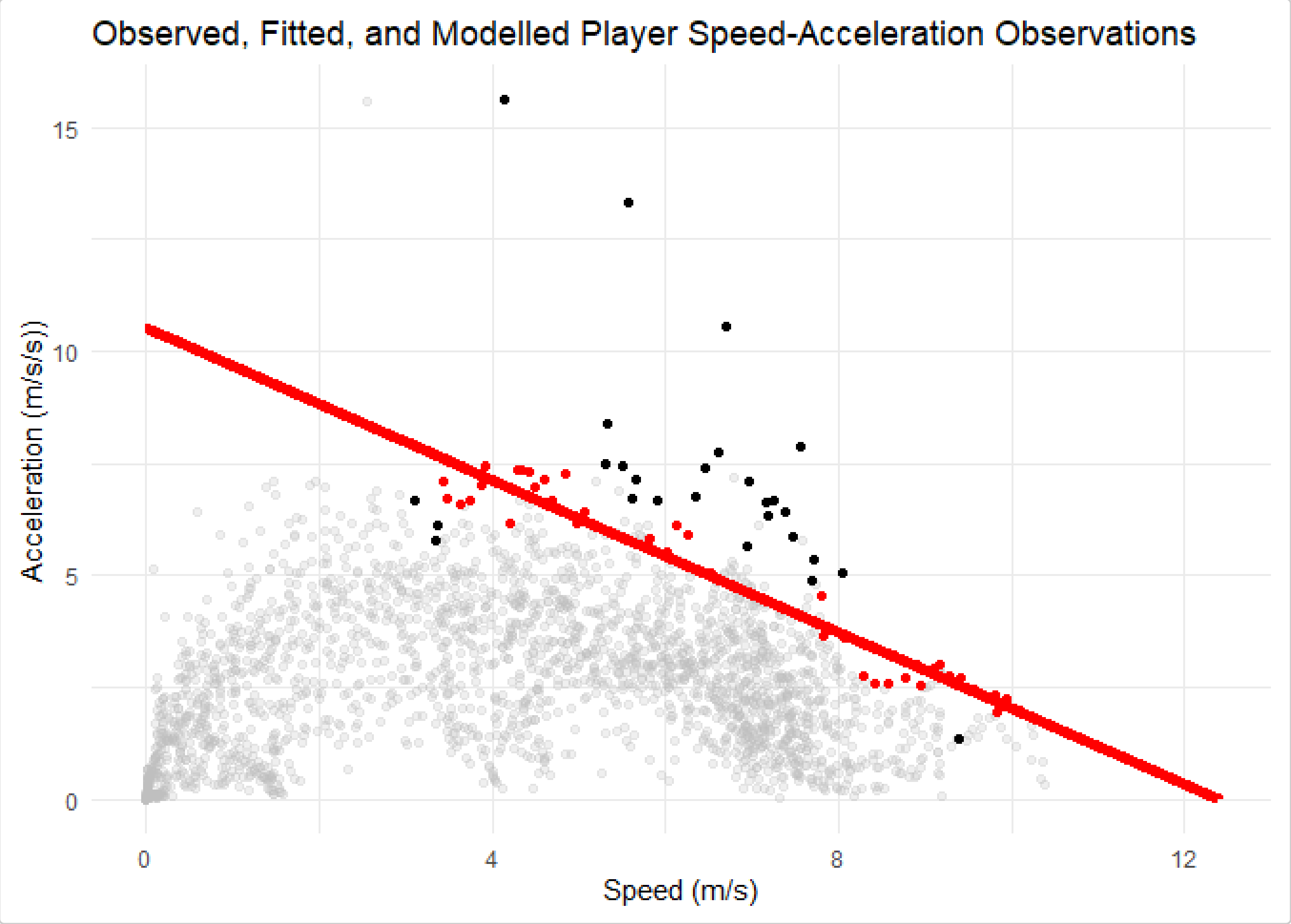
`sa.results.observed()` returns the observations that met inclusion criteria for the model, before fitting the linear model.

```
player.one.sa.observed <- sa.results.observed(game.data = player.one.sa)
head(player.one.sa.observed)
#>   game.speed game.accel speed.bins
#> 1       3.11       6.67    (3,3.2]
#> 2       3.07       7.96    (3,3.2]
#> 3       3.35       5.78  (3.2,3.4]
#> 4       3.38       6.10  (3.2,3.4]
#> 5       3.48       6.70  (3.4,3.6]
#> 6       3.43       7.09  (3.4,3.6]
```

## Plotting

Below is a sample plot that includes all observations from the original data set in light grey, the observations that met inclusion criteria in dark grey, the fitted observations in red, and the modelled speed-acceleration observations as the linear model.

```
p1 <- ggplot(data = player.one.sa,
             aes(x = game.speed, y = game.accel)) +

  geom_point(colour = "grey", # all player observations
             alpha = 0.25) +

  geom_point(data = player.one.sa.observed, # observations that met inclusion criteria
             colour = "black") +

  geom_point(data = player.one.sa.fitted, # fitted observations
             colour = "red") +

  geom_line(data = player.one.sa.model,
            colour = "red",
            size = 2) +

  xlab("Speed (m/s)") +
  ylab("Acceleration (m/s/s))") +
  ggtitle("Observed, Fitted, and Modelled Player Speed-Acceleration Observations")

p1
```



## `scout.` Sprint Test

### Clean and Process

```
player.one.scout <- scout.data(distance = player.forty$distance,
                               split.time = player.forty$split.time)
head(player.one.scout)
#>   distance split.time
#> 1     0.00       0.00
#> 2     9.14       1.66
#> 3    18.30       2.67
#> 4    36.60       4.72
```

### Player Profile

`scout.player.profile()` returns the player's name and their potential max speed, max acceleration, and acceleration constant.

```
player.one.scout.profile <- scout.player.profile(player.name = "Player 1 SCOUT",
                                                 sprint.data = player.one.scout)
player.one.scout.profile
#>      player.name max.speed max.accel player.tau
#> 1 Player 1 SCOUT  9.121661  12.98373  0.7025456
```

### Modelling

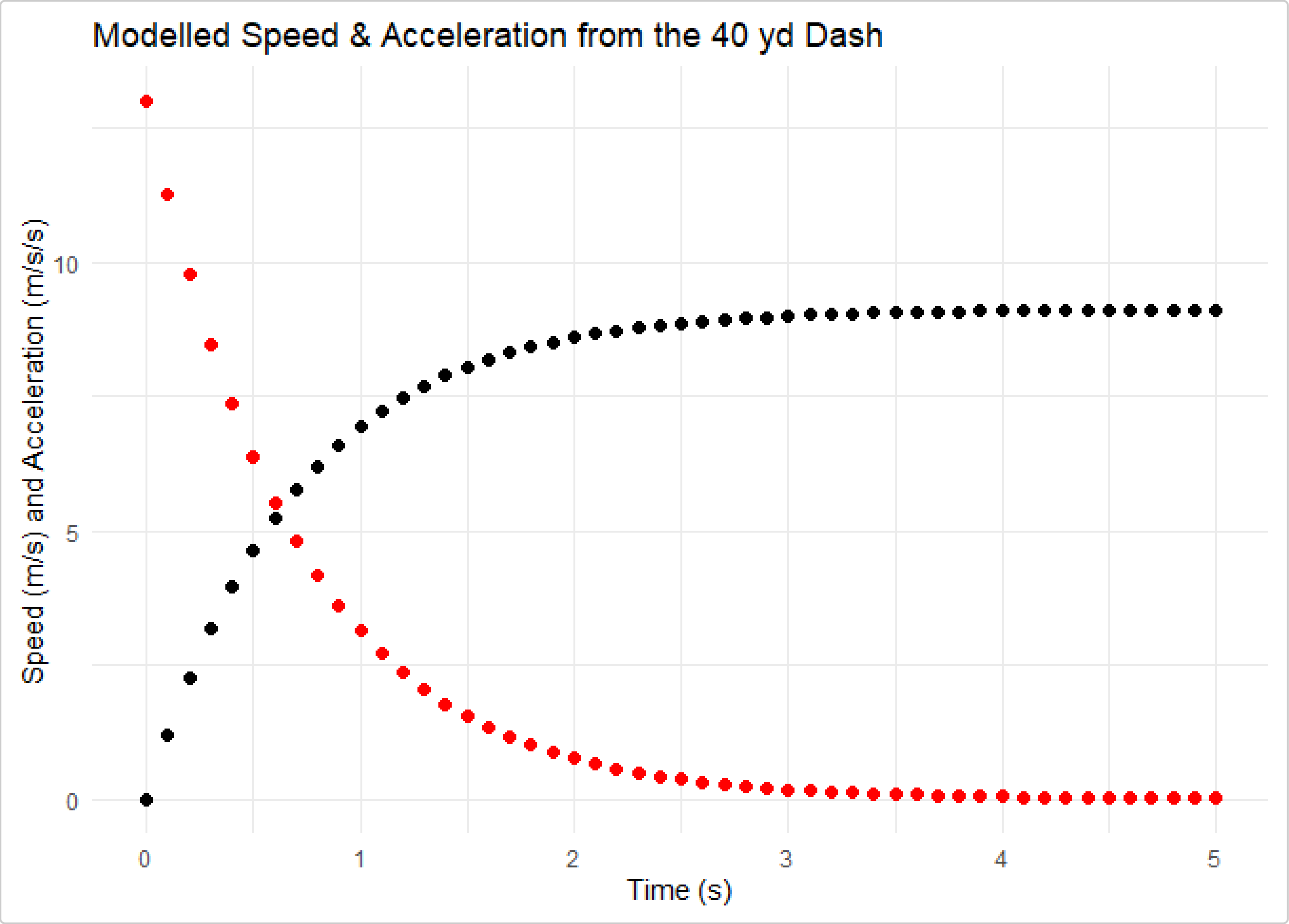
`scout.results.model()` returns the player modelled potential sprint ability over time and distance.

```
player.one.scout.model <- scout.results.model(sprint.data = player.one.scout)
head(player.one.scout.model)
#>   splits model.speed acceleration   distance
#> 1    0.0    0.000000    12.983727 0.00000000
#> 2    0.1    1.210200    11.261134 0.06194503
#> 3    0.2    2.259840     9.767082 0.23669166
#> 4    0.3    3.170220     8.471252 0.50927414
#> 5    0.4    3.959817     7.347343 0.86671225
#> 6    0.5    4.644656     6.372547 1.29774793
```

### Plotting

Below is a sample plot that fits modelled speed and acceleration over time.

```
p1 <- ggplot(data = player.one.scout.model, aes(x = splits)) +

  geom_point(aes(y = acceleration),
             colour = "red",
             size = 2) +

  geom_point(aes(y = model.speed),
             colour = "black",
             size = 2) +

  xlab("Time (s)") +
  ylab("Speed (m/s) and Acceleration (m/s/s)") +
  ggtitle("Modelled Speed & Acceleration from the 40 yd Dash")

p1
```



## `fv.` Force-Velocity Jump-Test

The `fv.` family of functions is more complex than those working with player tracking data. This is because the testing data is jump-tests which require information on the player and their testing conditions. The code below uses the `player.jump` data that was generated earlier.

### Clean and Process

`fv.data.player()` returns a player's anthropomorphic data.

```
player.one.fv <- fv.data.player(body.mass = 82.4,
                                lower.limb.length = 1.07,
                                initial.height = 0.74,
                                push.off.angle = 90)
player.one.fv
#>   body.mass lower.limb.length initial.height push.off.distance push.off.angle
#> 1      82.4              1.07           0.74              0.33             90
```

`fv.data.testing()` returns the player's testing results with some additional analyses. `player.one.jump` in the code chunk below is the same as the one generated above.

```
player.one.jump <- fv.data.testing(player.data = player.one.fv,
                                   percent.bw = c(0, 25, 50, 75),
                                   add.mass = c(0, 20.6, 41.2, 61.8),
                                   trial.one = c(0.34, 0.29, 0.21, 0.18),
                                   trial.two = c(0.35, 0.28, 0.19, 0.18),
                                   trial.three = c(0.34, 0.29, 0.20, 0.17))
player.one.jump
#>   condition additional.mass trial.one trial.two trial.three hmax total.mass
#> 1         0             0.0      0.34      0.35        0.34 0.35       82.4
#> 2        25            20.6      0.29      0.28        0.29 0.29      103.0
#> 3        50            41.2      0.21      0.19        0.20 0.21      123.6
#> 4        75            61.8      0.18      0.18        0.17 0.18      144.2
#>      force       vel    power
#> 1 1665.679 1.3102481 2182.452
#> 2 1898.384 1.1926651 2264.136
#> 3 1984.117 1.0149138 2013.708
#> 4 2186.203 0.9396276 2054.217
```

### Player Profile

`fv.player.profile()` returns a lot more than the `.player.profile()` functions previously seen. The data set returned contains 24 variables in total.

```
player.one.fv.profile <- fv.player.profile(player.name = "Player 1 FV",
                                           player.data = player.one.fv,
                                           testing.data = player.one.jump)
player.one.fv.profile
#>   player.name body.mass lower.limb.length initial.height push.off.distance
#> 1 Player 1 FV      82.4              1.07           0.74              0.33
#>   push.off.angle       f0 f0.normalized f0.optimal.30 f0.optimal.90
#> 1             90 3303.534      40.09143      36.38203      40.77494
#>   perc.optimal.30 perc.optimal.90     pmax pmax.normalized         r        r2
#> 1        121.4309        96.67548 2219.338        26.93372 0.9595935 0.9208196
#>         sfv sfv.normalized sfv.optimal training.focus.30 training.focus.90
#> 1 -1229.346      -14.91924   -15.43229          velocity          velocity
#>         v0 v0.optimal.30 v0.optimal.90
#> 1 2.687229      2.961211      2.642183
```

### Modelling

`fv.results.observed()` returns the fitted values from the linear model built from the testing data.

```
player.one.fv.observed <- fv.results.observed(player.data = player.one.fv,
                                              testing.data = player.one.jump)
head(player.one.fv.observed)
#>   velocity    force
#> 1     0.00 40.09143
#> 2     0.01 39.94224
#> 3     0.02 39.79304
#> 4     0.03 39.64385
#> 5     0.04 39.49466
#> 6     0.05 39.34547
```

`fv.results.model()` returns the modelled force-velocity observations for 30 and 90 degree push-off angles.

*30 degrees*

```
player.one.fv.modeled.30 <- fv.results.model(player.data = player.one.fv,
                                             testing.data = player.one.jump,
                                             push.off.angle = 30)
head(player.one.fv.modeled.30)
#>   velocity    force
#> 1     0.00 36.38203
#> 2     0.01 36.25917
#> 3     0.02 36.13631
#> 4     0.03 36.01344
#> 5     0.04 35.89058
#> 6     0.05 35.76772
```

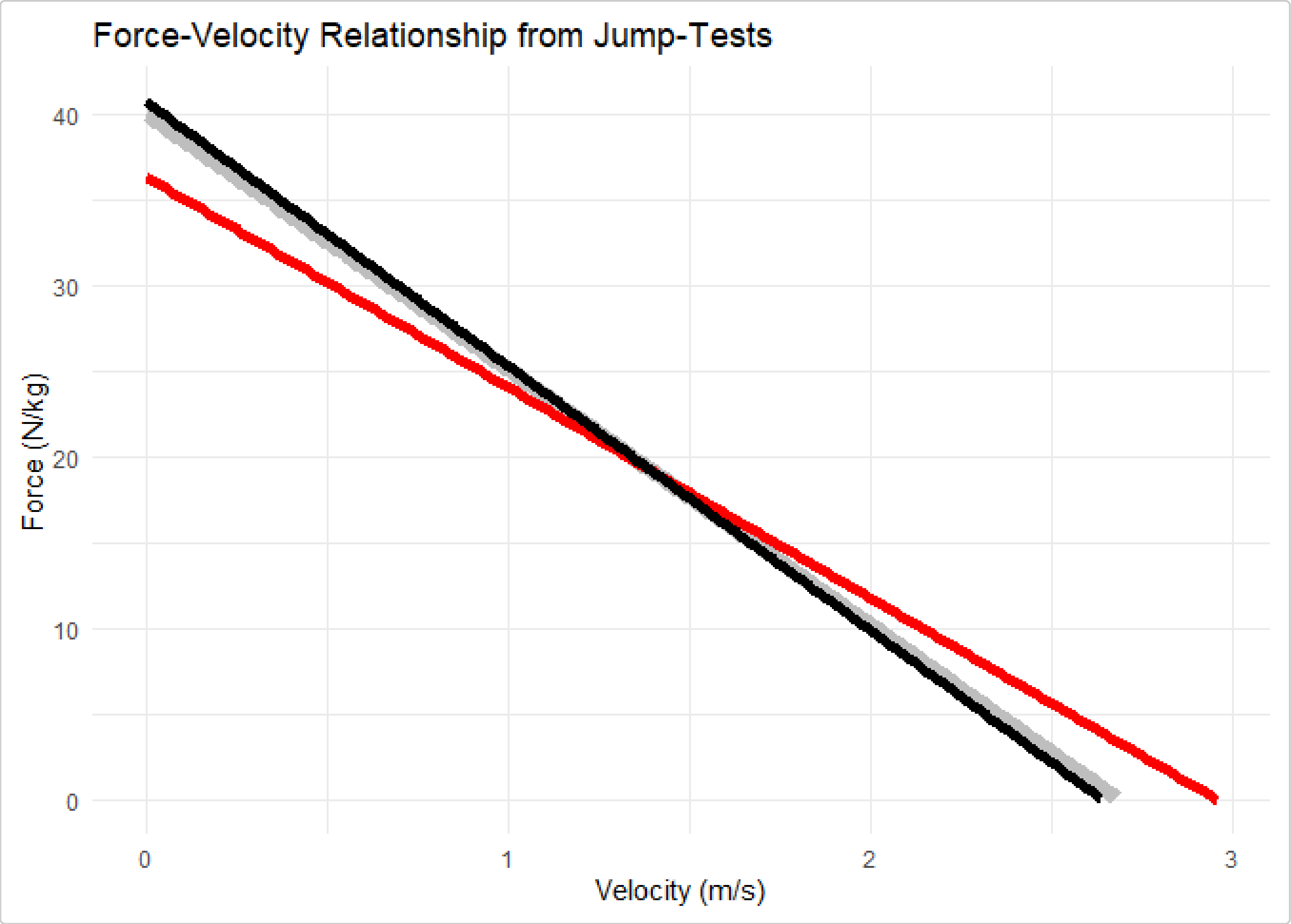
*90 degrees*

```
player.one.fv.modeled.90 <- fv.results.model(player.data = player.one.fv,
                                             testing.data = player.one.jump,
                                             push.off.angle = 90)
head(player.one.fv.modeled.90)
#>   velocity    force
#> 1     0.00 40.77494
#> 2     0.01 40.62062
#> 3     0.02 40.46630
#> 4     0.03 40.31197
#> 5     0.04 40.15765
#> 6     0.05 40.00333
```

### Plotting

Below is a sample plot that fits the observed results and is overlayed by the optimal results for push-off angles of 30 and 90 degrees.

```
p1 <- ggplot() +

  geom_line(data = player.one.fv.observed, # actual results
            aes(x = velocity, y = force),
            colour = "grey",
            size = 3) +

  geom_line(data = player.one.fv.modeled.30, # optimal 30 degree push-off angle
            aes(x = velocity, y = force),
            colour = "red",
            size = 2) +

  geom_line(data = player.one.fv.modeled.90, # optimal 90 degree pus-off angle
            aes(x = velocity, y = force),
            colour = "black",
               size = 2) +

    xlab("Velocity (m/s)") +
    ylab("Force (N/kg)") +
    ggtitle("Force-Velocity Relationship from Jump-Tests")

p1
```



## `fvp.` Force-Velocity-Power

The `fvp.` family of functions returns extensive force-velocity-power analyses. To analyse a player's *fvp* abilities, the user needs to have previously built player profiles from the `gps.`, `sa.`, or `scout.` families.

For the examples below, the `player.profile` argument will use `player.one.gps.profile` that was created above.

### Clean and Process

```
player.one.fvp <- fvp.data.player(player.height = 1.74,
                                  body.mass = 115,
                                  ambient.temp = 25,
                                  air.pressure = 760)
player.one.fvp
```

```
#>   player.height body.mass ambient.temp air.pressure
#> 1          1.74       115           25          760
```

### Player Profile

`fvp.player.profile()` returns an extensive amount of information on the player for a total of 16 variables.

```
player.one.fvp.profile <- fvp.player.profile(player.name = "Player 1 FVP",
                                             player.data = player.one.fvp,
                                             player.profile = player.one.gps.profile)
player.one.fvp.profile
#>    player.name player.height body.mass ambient.temp air.pressure max.speed
#> 1 Player 1 FVP          1.74       115           25          760     10.39
#>   max.accel player.tau       f0 f0.normalized  fv.slope     pmax
#> 1      15.6  0.6660256 1783.478       15.5085 -1.501444 4605.411
#>   pmax.normalized      rf.d   rf.max       v0
#> 1        40.04705 -9.694457 84.65317 10.32906
```

### Modelling

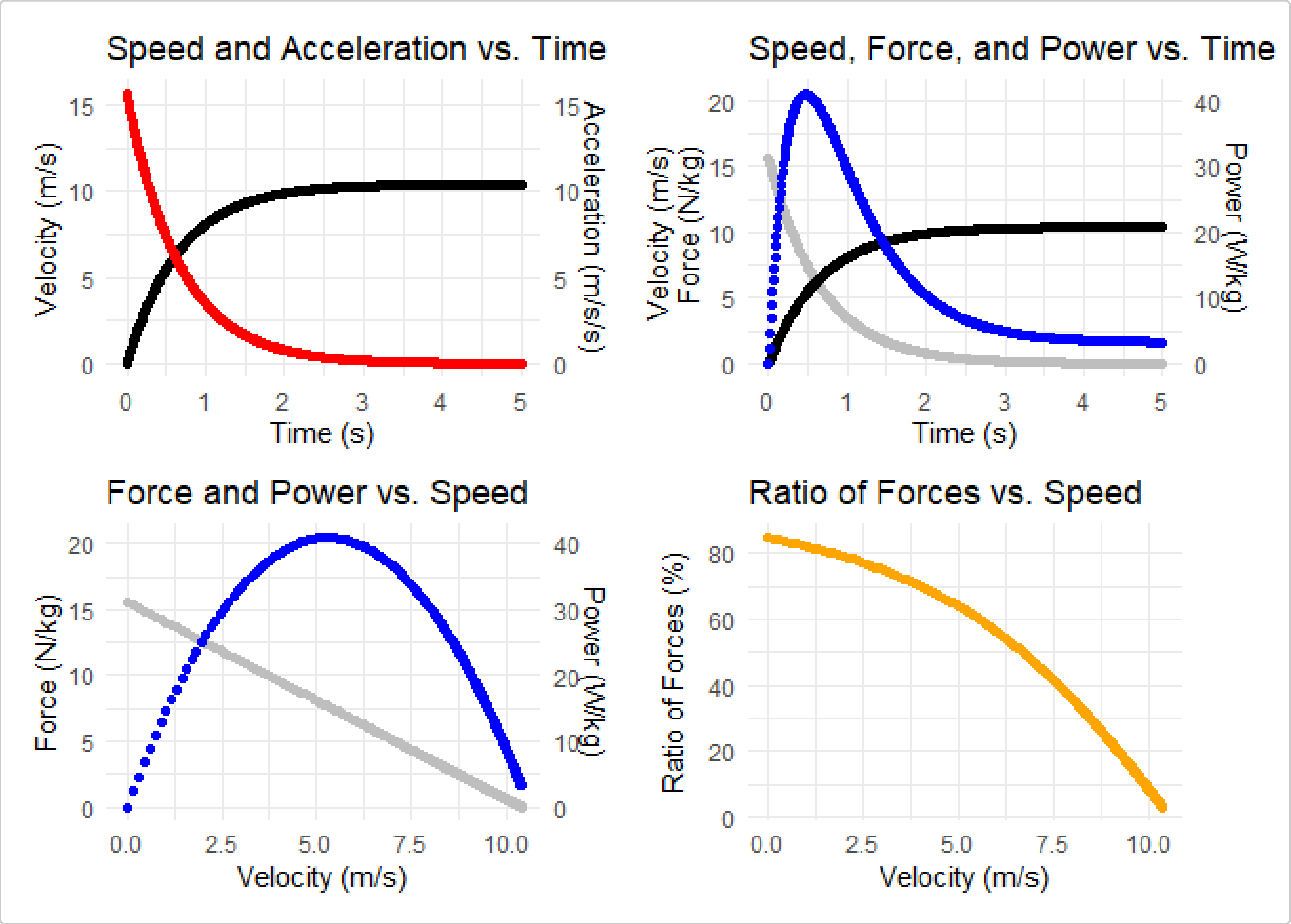
`fvp.results.model()` returns the player extensively modelled sprint abilities.

```
player.one.fvp.model <- fvp.results.model(player.data = player.one.fvp,
                                          player.profile = player.one.gps.profile,
                                          sprint.duration = 5)
head(player.one.fvp.model)
#>   split.time     speed acceleration     distance horizontal.force   air.force
#> 1       0.00 0.0000000     15.60000 0.0000000000         1794.000 0.000000000
#> 2       0.01 0.1548347     15.36752 0.0007761109         1767.265 0.007834464
#> 3       0.02 0.3073620     15.13851 0.0030890030         1740.929 0.030872589
#> 4       0.03 0.4576164     14.91291 0.0069157749         1714.985 0.068434591
#> 5       0.04 0.6056315     14.69068 0.0122338664         1689.428 0.119864222
#> 6       0.05 0.7514410     14.47175 0.0190210532         1664.252 0.184528024
#>   horizontal.net.force horizontal.net.force.normalized horizontal.power
#> 1             1794.000                        15.60000           0.0000
#> 2             1767.273                        15.36752         273.6352
#> 3             1740.960                        15.13851         535.1050
#> 4             1715.054                        14.91291         784.8366
#> 5             1689.548                        14.69068        1023.2435
#> 6             1664.436                        14.47175        1250.7255
#>   horizontal.power.normalized vertical.force net.force angle.of.forces
#> 1                    0.000000        1128.15  2119.235        32.16358
#> 2                    2.379437        1128.15  2096.658        32.55238
#> 3                    4.653087        1128.15  2074.527        32.94342
#> 4                    6.824666        1128.15  2052.835        33.33663
#> 5                    8.897769        1128.15  2031.574        33.73195
#> 6                   10.875874        1128.15  2010.739        34.12930
#>   ratio.of.forces
#> 1        84.65317
#> 2        84.28999
#> 3        83.92080
#> 4        83.54562
#> 5         83.16446
#> 6         82.77735
```

### Plotting

Below is a plot that returns a player’s force-velocity-power profile on 4 graphs.

```
p1 <- ggplot(player.one.fvp.model, aes(x = split.time)) +
        geom_point(aes(y = speed), colour = "black") +
        geom_point(aes(y = acceleration), colour = "red") +
        xlab("Time (s)") +
        scale_y_continuous("Velocity (m/s)",
                           sec.axis = sec_axis(~., name = "Acceleration (m/s/s)")) +
        ggtitle("Speed and Acceleration vs. Time")

p2 <- ggplot(player.one.fvp.model, aes(x = split.time)) +
        geom_point(aes(y = horizontal.net.force.normalized), colour = "grey") +
        geom_point(aes(y = speed), colour = "black") +
        geom_point(aes(y = horizontal.power.normalized/ 2), colour = "blue") +
        xlab("Time (s)") +
        scale_y_continuous("Velocity (m/s) \n Force (N/kg)",
                         sec.axis = sec_axis(~.*2, name = "Power (W/kg)")) +
        ggtitle("Speed, Force, and Power vs. Time")

p3 <- ggplot(player.one.fvp.model, aes(x = speed)) +
        geom_point(aes(y = horizontal.net.force.normalized), colour = "grey") +
        geom_point(aes(y = horizontal.power.normalized/ 2), colour = "blue") +
        xlab("Velocity (m/s)") +
        scale_y_continuous("Force (N/kg)",
                       sec.axis = sec_axis(~.*2, name = "Power (W/kg)")) +
        ggtitle("Force and Power vs. Speed")

p4 <- ggplot(player.one.fvp.model, aes(x = speed)) +
        geom_point(aes(y = ratio.of.forces), colour = "orange") +
        xlab("Velocity (m/s)") +
        ylab("Ratio of Forces (%)") +
        xlim(c(0, NA)) +
        ggtitle("Ratio of Forces vs. Speed")


p1 + p2 + p3 + p4
```



# Extending the Package

## Compare Player Abilities

Although the package is built to return modelled outcomes, there are secondary functions that can be useful in comparing athletes. For example, the `compare.player()` functions provide users a simple way of comparing athletes' profiles or split times.

In the code below, the `build.player.profile()` function is used twice to build new player profiles. These profiles are then used to compare the players' abilities.

### Build Profiles

```
new.profile.one <- build.player.profile(player.name = "New Profile 1",
                                        max.speed = 10,
                                        max.accel = 14)
new.profile.two <- build.player.profile(player.name = "New Profile 2",
                                        max.speed = 12,
                                        max.accel = 9)

new.profile.one; new.profile.two
#>      player.name max.speed max.accel player.tau
#> 1 New Profile 1        10        14  0.7142857
#>     player.name max.speed max.accel player.tau
#> 1 New Profile 2        12         9   1.333333
```

### Compare Player Profiles

```
compare.player.profiles(player.profiles = list(new.profile.one,
                                               new.profile.two))
#>     player.name max.speed max.accel player.tau
#> 1 New Profile 1        10        14  0.7142857
#> 2 New Profile 2        12         9  1.3333333
```

### Compare Player Splits

```
compare.player.splits(player.profiles = list(new.profile.one,
                                             new.profile.two),
                      distance = 9.14) # 10 yard split
#>     player.name distance split.time split.speed split.accel
#> 1 New Profile 1     9.14   1.546308    8.852306    1.606771
#> 2 New Profile 2     9.14   1.730983    8.723849    2.457113
```

## Save Data Sets

To save data sets like player profiles, player data, modelled data, and player comparisons, you can used the `save.as.()` functions. You have the option of saving the data as a Microsoft Excel sheet (.csv) or Rdata file (.rds). The function will return a message after saving the file to your R console.

```
save.as.excel(data.set = player.one.fvp.model,
              file.name = "player one fvp excel")

save.as.csv(data.set = player.one.fvp.model,
            file.name = "player one fvp csv")

save.as.rds(data.set = player.one.fvp.model,
            file.name = "player one fvp rds")

#> [1] "player one fvp excel.csv" was save to (your working directory here)"
```

## Mechanical Sprint Abilities

The package also provides the user the ability to model specific sprint abilities. By using a player's profile, the user can return the relationships between speed, acceleration, distance, and time. With the introduction of *midsprint*, many of these functions no longer rely on a player starting with zero velocity. The functions that are accessible include:

```
speed.time(): a player's speed after a given amount of time
accel.time(): a player's rate of acceleration after a given amount of time
distance.time(): the distance reached at max effort after a given amount of time
distance.speed(): the distance it takes for the player to reach a given speed
```

```
time.speed(): the time it takes to reach a given speed
time.distance(): the time it takes to reach a given distance
time.to.position(): the time it takes to reach a given distance while a player is already in motion
```

You can also call `midsprint()` for `time.to.position()`

## Citing fvp

When using the fvp package to present and publish, please reference the package by calling the `cite.fvp()` function.