



shorts: An R Package for Modeling Short Sprints

Mladen Jovanović
University of Belgrade

Jason D. Vescovi
University of Toronto

Abstract

The abstract of the article.

Keywords: sprints, sports, modeling.

1. Introduction

Short sprint performance is one of the most distinguishable and admired physical trait in sports. Short sprints, commonly performed in most team sports (e.g., soccer, field hockey, handball, football, etc.), are defined as maximal running from a stand still position over a distance that doesn't result in deceleration at the end. Peak anaerobic power is achieved within the first few seconds (<5 sec) of maximal efforts (Mangine, Hoffman, Gonzalez, Wells, Townsend, Jajtner, McCormack, Robinson, Fragala, Fukuda, and Stout 2014), whereas the ability to achieve maximal sprint speed varies based on the type of sport. For example, track and field sprinters are trained to achieve maximal speed later in a race (i.e., 50-60 m) (Ward-Smith 2001), but team sport athletes have sport-specific attributes and reach it much sooner (i.e., 30-40 m) (Brown, Vescovi, and Vanheest 2004).

Describing individual differences in short sprint performance, quantifying training intervention effects, but more importantly, understanding short sprint *bottle necks* (i.e. limiting factors) has been one of the prominent tasks of sports science profession since its establishment. The **shorts** package (Jovanovic 2020), written in the R language (R Core Team 2020), represents an open-source tool that aims to help sports scientist in achieving these goals through mathematical modeling.

In the current paper, we will provide explanation of one commonly used mathematical equation to model short sprints, modeling applications using the **shorts** package, issues that can

Table 1: Four athletes with different MSS and MAC parameters.

Athlete	MSS	MAC	TAU
Athlete A	12	10	1.20
Athlete B	12	6	2.00
Athlete C	8	10	0.80
Athlete D	8	6	1.33

arise during measurement and estimation, and potential solutions to those problems.

2. Mathematical model

Short sprints have been modeled using the mono-exponential equation (1) originally proposed by Furusawa, Hill, and Parkinson (1927), and more recently popularized by Clark, Rieger, Bruno, and Stearne (2017), and Samozino, Rabita, Dorel, Slawinski, Peyrot, Saez de Villarreal, and Morin (2016). Equation (1) represents function for instantaneous horizontal velocity v given the time t and two model parameters:

$$v(t) = MSS \times (1 - e^{-\frac{t}{TAU}}) \quad (1)$$

The parameters of the equation (1) are *maximum sprinting speed* (MSS; expressed in ms^{-1}) and *relative acceleration* (TAU). Mathematically, TAU represents the ratio of MSS to initial acceleration (MAC; *maximal acceleration*, expressed in ms^{-2}) (2).

$$MAC = \frac{MSS}{TAU} \quad (2)$$

Although TAU is used in the equations, and later estimated, it is preferred to use MAC instead since it is easier to grasp, particularly for less math inclined coaches.

By derivating equation (1), we can get equation for horizontal acceleration (3).

$$a(t) = \frac{MSS}{TAU} \times e^{-\frac{t}{TAU}} \quad (3)$$

By integrating equation (1), we can get equation for distance covered (4).

$$d(t) = MSS \times (t + TAU \times e^{-\frac{t}{TAU}}) - MSS \times TAU \quad (4)$$

Let's consider four athletes with different levels of MSS (high versus low maximal sprinting speed) and MAC (high versus low maximal acceleration; as mentioned previously, using MAC is preferred over using TAU) (Table 1).

The Figure 1 depicts distance, velocity, and acceleration over time (from 0 to 6s).

If we plot acceleration against velocity (Figure 2), we will get Acceleration-Velocity profile, which is linear, according to the mathematical model.

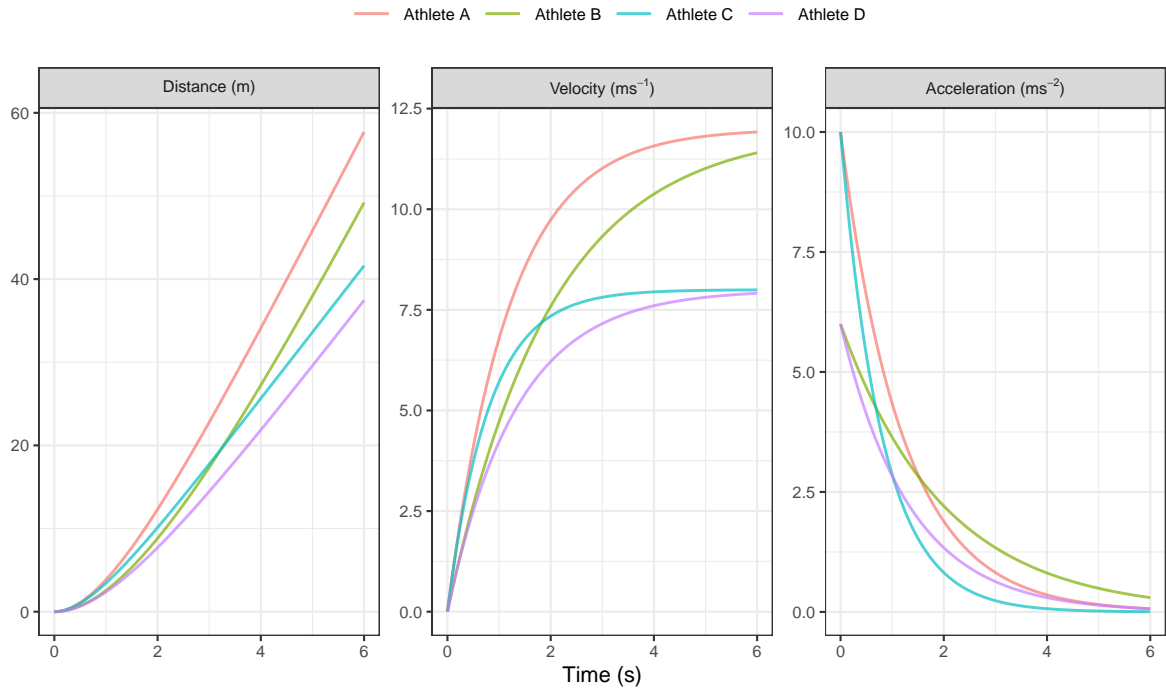


Figure 1: Kinematic characteristic of four athletes with different MSS and MAC parameters over a period of 0 to 6seconds.

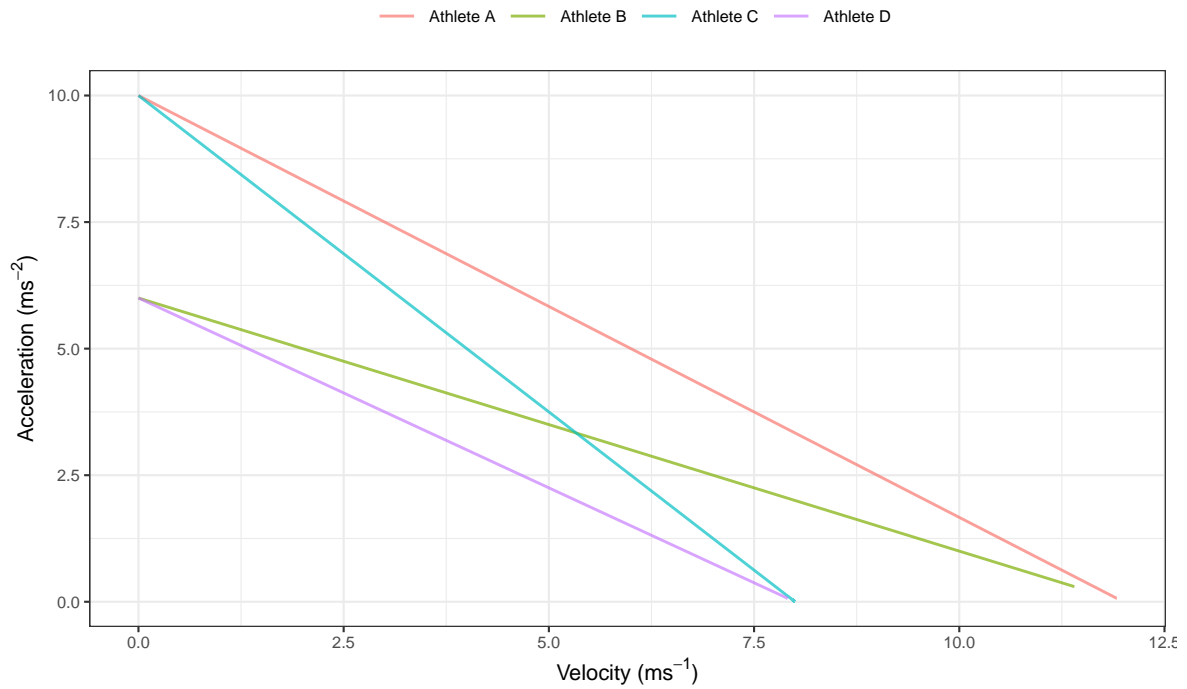


Figure 2: Acceleration-Velocity profile of four athletes with different MSS and MAC parameters.

3. Estimation using shorts package

Short sprints profiling is usually performed by: (1) measuring split times using timing gates (i.e., positioned at various distances, e.g., 5, 10, 20, 30, 40m), and (2) getting a velocity trace using a radar gun. Estimation of MSS and TAU parameters from equation (1) is performed in **shorts** package using non-linear least squares regression implemented in the **nls** function in the **base R** (R Core Team 2020) and **nlme** function in the **nlme** package (Pinheiro, Bates, and R-core 2020) for the mixed-effect models.

3.1. Estimating short sprint parameters using split times

Let's consider an example of an athlete with MSS equal to 9 ms^{-1} , TAU equal to 1.3, and MAC equal to 6.92 ms^{-2} performing 40m sprint with timing gates positioned at each 10m split. When it comes to split times, distance is a predictor, and time is outcome variable, thus the equation (1) takes another form:

$$t(d) = TAU \times W(-e^{\frac{-d}{MSS \times TAU}} - 1) + \frac{d}{MSS} + TAU \quad (5)$$

W in equation (1) represents Lambert's W function (Goerg 2020). MSS and TAU parameters are estimated using **shorts::model_using_splits** function:

```
R> require(shorts)
R>
R> split_distance <- c(10, 20, 30, 40)
R>
R> split_time <- c(2.17, 3.43, 4.60, 5.73)
R>
R> m1 <- shorts::model_using_splits(
+   distance = split_distance,
+   time = split_time
+ )
R>
R> m1
```

```
#> Estimated model parameters
#> -----
#>           MSS           TAU           MAC
#>          9.01          1.31          6.89
#>       PMAX   time_correction distance_correction
#>        15.52             0.00             0.00
#>
#> Model fit estimators
#> -----
#>      RSE R_squared   minErr   maxErr maxAbsErr   RMSE
#>  0.00249  1.00000 -0.00178  0.00265  0.00265  0.00176
#>      MAE      MAPE
#>  0.00157  0.04742
```

Maximal relative power (PMAX) from the output is estimated using $\frac{MSS \times MAC}{4}$, which disregards the air resistance. `time_correction` and `distance_corection` parameters will be covered later in the article.

Besides providing *residual standard error* (RSE), **shorts** functions provide additional model fit estimators. Additional information can be gained by exploring the returned object, particularly object returned from the `nls` function:

```
R> summary(m1$model)

#>
#> Formula: corrected_time ~ TAU * I(LambertW::W(-exp(1)^(-distance/(MSS *
#>      TAU) - 1))) + distance/MSS + TAU
#>
#> Parameters:
#>      Estimate Std. Error t value Pr(>|t|)
#> MSS    9.0121      0.0155    581 3.0e-06 ***
#> TAU    1.3083      0.0066    198 2.5e-05 ***
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.00249 on 2 degrees of freedom
#>
#> Number of iterations to convergence: 4
#> Achieved convergence tolerance: 3.11e-06
```

Once we have estimated MSS and TAU, we can use `shorts::predict_` family of functions to predict various relationships (i.e. time at distance, acceleration at distance, velocity at time):

```
R> # Predict time at distance
R> shorts::predict_time_at_distance(
+   distance = split_distance,
+   MSS = m1$parameters$MSS,
+   TAU = m1$parameters$TAU
+ )

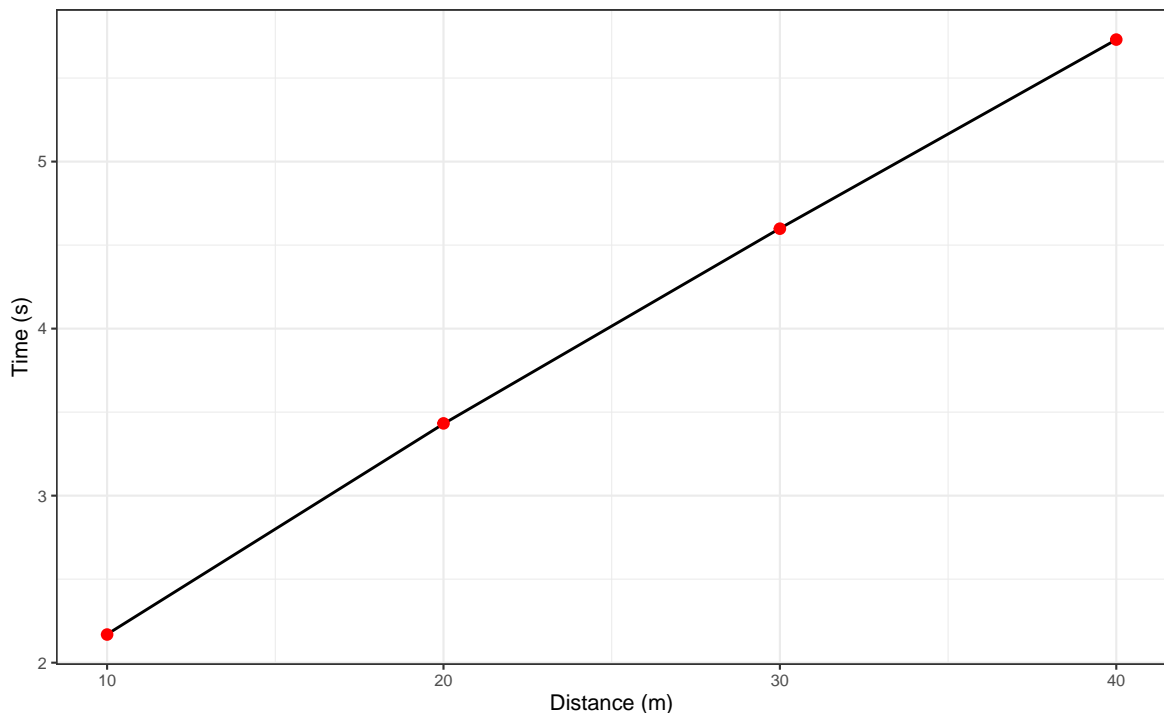
#> [1] 2.17 3.43 4.60 5.73

R>
R> # Predict acceleration at time
R> shorts::predict_acceleration_at_time(
+   time = c(0, 1, 2, 3, 4, 5, 6),
+   MSS = m1$parameters$MSS,
+   TAU = m1$parameters$TAU
+ )

#> [1] 6.8884 3.2075 1.4935 0.6954 0.3238 0.1508 0.0702
```

Here is a plot of observed and model predicted split times, with the help of the **ggplot2** package (Wickham, Chang, Henry, Pedersen, Takahashi, Wilke, Woo, Yutani, and Dunnington 2020a):

```
R> require(ggplot2)
R>
R> df <- data.frame(
+   distance = split_distance,
+   time = split_time,
+   pred_time = shorts::predict_time_at_distance(
+     distance = split_distance,
+     MSS = m1$parameters$MSS,
+     TAU = m1$parameters$TAU
+   )
+ )
R>
R> ggplot(df, aes(x = distance)) +
+   theme_bw(8) +
+   geom_line(aes(y = time)) +
+   geom_point(aes(y = pred_time), color = "red") +
+   xlab("Distance (m)") +
+   ylab("Time (s)")
```



Air resistance and the calculation of force and mechanical power

To estimate force production at distance or time (using `shorts::predict_force_at_distance` and `shorts::predict_force_at_time` functions), as well as power production (using `shorts::predict_power_at_distance` and `shorts::predict_power_at_time` functions),

and `shorts::predict_power_at_time` functions), one needs to take into account the air resistance. Air resistance (N) is estimated using `shorts::get_air_resistance` function, which takes velocity, body mass (kg), body height (m), barometric pressure (Torr), air temperature (C°), and wind velocity (ms^{-1}) as parameters (please refer to [Arsac and Locatelli \(2002\)](#), [Samozino *et al.* \(2016\)](#), and [van Ingen Schenau, Jacobs, and de Koning \(1991\)](#) for more information):

```
R> shorts::get_air_resistance(
+   velocity = 5,
+   bodymass = 80,
+   bodyheight = 1.85,
+   barometric_pressure = 780,
+   air_temperature = 20,
+   wind_velocity = 0.5
+ )
```

```
#> [1] 6.1
```

When estimating force and power, the air resistance parameters can be set using "...", which are forwarded to the `shorts::get_air_resistance`:

```
R> # To calculate horizontal force produced
R> shorts::predict_force_at_distance(
+   distance = split_distance,
+   MSS = m1$parameters$MSS,
+   TAU = m1$parameters$TAU,
+   # Additional parameters forwarded to shorts::get_air_resistance
+   # Otherwise, defaults are used
+   bodymass = 80,
+   bodyheight = 1.85,
+   barometric_pressure = 780,
+   air_temperature = 20,
+   wind_velocity = 0.5
+ )
```

```
#> [1] 119.0  58.6  36.9  28.2
```

```
R>
R> # To calculate power produced
R> shorts::predict_power_at_distance(
+   distance = split_distance,
+   MSS = m1$parameters$MSS,
+   TAU = m1$parameters$TAU,
+   # Additional parameters forwarded to shorts::get_air_resistance
+   # Otherwise, defaults are used
+   bodymass = 80,
```

```
+   bodyheight = 1.85,
+   barometric_pressure = 780,
+   air_temperature = 20,
+   wind_velocity = 0.5
+ )
```

```
#> [1] 868 490 323 251
```

The easiest way to get all kinematics for short sprints is to use `shorts::predict_kinematics` functions:

```
R> df <- shorts::predict_kinematics(
+   m1,
+   max_time = 6,
+   frequency = 100,
+   # Additional parameters forwarded to shorts::get_air_resistance
+   # Otherwise, defaults are used
+   bodymass = 80,
+   bodyheight = 1.85,
+   barometric_pressure = 780,
+   air_temperature = 20,
+   wind_velocity = 0.5
+ )
R>
R> head(df)
```

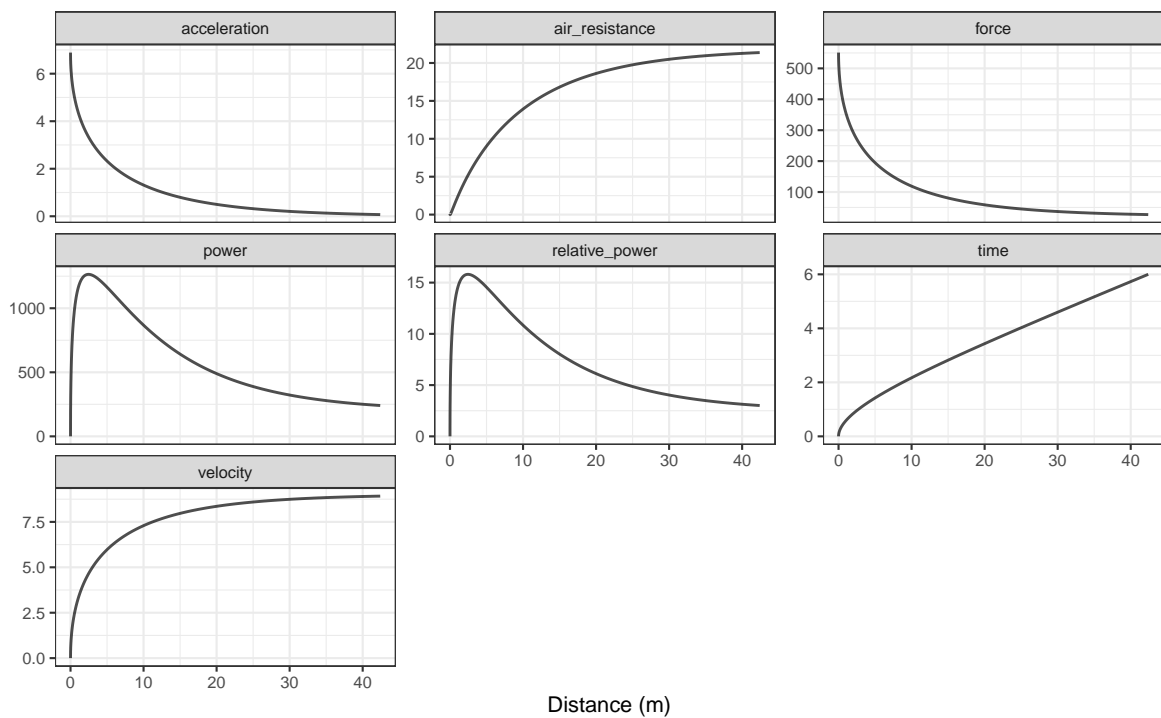
```
#>   time distance velocity acceleration air_resistance force
#> 1 0.00 0.000000  0.0000         6.89         0.07536  551
#> 2 0.01 0.000344  0.0686         6.84         0.05609  547
#> 3 0.02 0.001371  0.1367         6.78         0.03978  543
#> 4 0.03 0.003076  0.2043         6.73         0.02636  539
#> 5 0.04 0.005455  0.2714         6.68         0.01576  534
#> 6 0.05 0.008502  0.3379         6.63         0.00792  530
#>   power relative_power
#> 1   0.0           0.000
#> 2  37.5           0.469
#> 3  74.2           0.928
#> 4 110.0           1.375
#> 5 145.0           1.813
#> 6 179.2           2.241
```

Plotting the model predictions can be done once we convert data from wide to long with the help of **dplyr** (Wickham, François, Henry, and Müller 2020b), **tidyr** (Wickham 2020), and **tidyverse** (Wickham 2019) packages:

```
R> require(tidyverse)
R>
```



```
R> df <- pivot_longer(data = df, cols = -2)
R>
R> ggplot(df, aes(x = distance, y = value)) +
+   theme_bw(8) +
+   facet_wrap(~name, scales = "free_y") +
+   geom_line(alpha = 0.7) +
+   ylab(NULL) +
+   xlab("Distance (m)")
```



Utility functions

Another valuable addition for sport scientists and coaches is the ability to determine the distances and times where 90% of maximum sprinting speed is reached, or where peak power is within 90% range. To identify these values, **shorts** package comes with **shorts::find_** family of functions:

```
R> # Finds distance where 90% of maximum sprinting speed is reached
R> shorts::find_velocity_critical_distance(
+   MSS = m1$parameters$MSS,
+   TAU = m1$parameters$TAU,
+   percent = 0.9
+ )
```

```
#> [1] 16.5
```

```

R>
R> # Finds maximal power and distance (this time using air resistance)
R> shorts::find_max_power_distance(
+   MSS = m1$parameters$MSS,
+   TAU = m1$parameters$TAU,
+   # Additional parameters forwarded to shorts::get_air_resistance
+   # Otherwise, defaults are used
+   bodymass = 80,
+   bodyheight = 1.85,
+   barometric_pressure = 780,
+   air_temperature = 20,
+   wind_velocity = 0.5
+ )

#> $max_power
#> [1] 1264
#>
#> $distance
#> [1] 2.46

R>
R>
R> # Finds distance over 90% power range
R> shorts::find_power_critical_distance(
+   MSS = m1$parameters$MSS,
+   TAU = m1$parameters$TAU,
+   # Additional parameters forwarded to shorts::get_air_resistance
+   # Otherwise, defaults are used
+   bodymass = 80,
+   bodyheight = 1.85,
+   barometric_pressure = 780,
+   air_temperature = 20,
+   wind_velocity = 0.5
+ )

#> $lower
#> [1] 0.959
#>
#> $upper
#> [1] 5.44

```

Mixed-effects model

Sprint performance is often evaluated with a group of athletes (e.g., soccer club) representing a single strata of interest. Sports scientists can estimate individual profiles, or utilize mixed-effects models. To perform mixed-effects models in **shorts** for split times, one can use

`shorts::mixed_model_using_splits` function. To demonstrate this functionality, we load the `split_times` dataset provided in the **shorts** package:

```
R> data(split_times)
R>
R> head(split_times)
```

```
#> # A tibble: 6 x 4
```

	athlete	bodyweight	distance	time
#>	<chr>	<dbl>	<dbl>	<l<dbl>>
#> 1	John	75	5	1.20
#> 2	John	75	10	1.97
#> 3	John	75	15	2.66
#> 4	John	75	20	3.31
#> 5	John	75	30	4.59
#> 6	John	75	40	5.85

```
R>
R> # Mixed model
R> m2 <- shorts::mixed_model_using_splits(
+   data = split_times,
+   distance = "distance",
+   time = "time",
+   athlete = "athlete",
+
+   # Select random effects
+   # Default is MSS and TAU
+   random = MSS + TAU ~ 1
+ )
R>
R> m2
```

```
#> Estimated fixed model parameters
#> -----
```

	MSS	TAU	MAC
#>	8.065	0.655	12.309
#>	PMAX	time_correction	distance_correction
#>	24.818	0.000	0.000

```
#>
#> Estimated random model parameters
#> -----
```

	athlete	MSS	TAU	MAC	PMAX	time_correction
#> 1	James	9.69	0.847	11.4	27.7	0
#> 2	Jim	7.83	0.505	15.5	30.4	0
#> 3	John	7.78	0.727	10.7	20.8	0
#> 4	Kimberley	8.57	0.802	10.7	22.9	0
#> 5	Samantha	6.45	0.395	16.3	26.4	0

```

#> distance_correction
#> 1 0
#> 2 0
#> 3 0
#> 4 0
#> 5 0
#>
#> Model fit estimators
#> -----
#>      RSE R_squared minErr maxErr maxAbsErr RMSE
#> 0.0260 0.9998 -0.0293 0.0496 0.0496 0.0214
#>      MAE MAPE
#> 0.0172 0.9019

```

Additional information about mixed-effects model performed using the `nlme` package ([Pinheiro *et al.* 2020](#)) can be obtained using `summary`:

```
R> summary(m2)
```

```

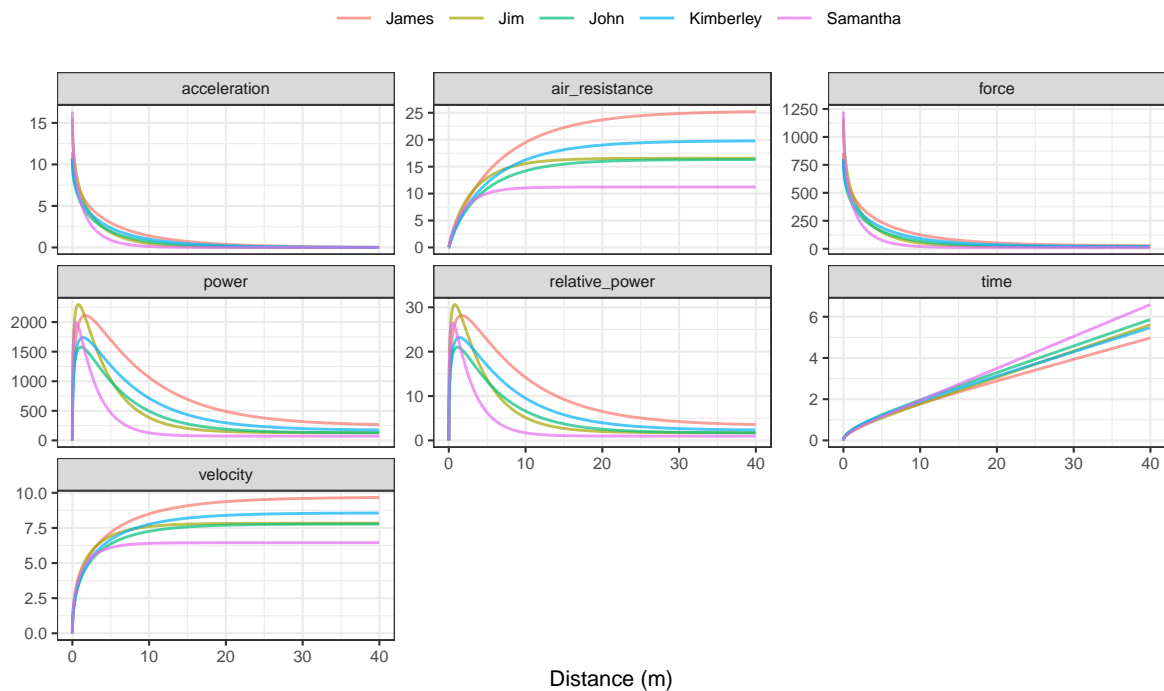
#> Nonlinear mixed-effects model fit by maximum likelihood
#> Model: corrected_time ~ TAU * I(LambertW::W(-exp(1)^(-distance/(MSS * TAU) - 1)))
#> Data: train
#>      AIC   BIC logLik
#> -75.1 -66.7  43.5
#>
#> Random effects:
#> Formula: list(MSS ~ 1, TAU ~ 1)
#> Level: athlete
#> Structure: General positive-definite, Log-Cholesky parametrization
#>      StdDev Corr
#> MSS      1.066 MSS
#> TAU      0.178 0.877
#> Residual 0.026
#>
#> Fixed effects: MSS + TAU ~ 1
#>      Value Std.Error DF t-value p-value
#> MSS  8.06      0.495 24  16.30      0
#> TAU  0.66      0.084 24   7.82      0
#> Correlation:
#>      MSS
#> TAU 0.874
#>
#> Standardized Within-Group Residuals:
#>      Min      Q1      Med      Q3      Max
#> -1.909 -0.605  0.154  0.523  1.129
#>

```

```
#> Number of Observations: 30
#> Number of Groups: 5
```

The following plot contains kinematics for all athletes in `split_times` dataset. Please note that power calculation takes default parameters for each individual:

```
R> df <- predict_kinematics(m2, max_time = 10)
R>
R> df <- pivot_longer(df, cols = c(-1, -3))
R>
R> ggplot(
+   filter(df, distance < 40),
+   aes(x = distance, y = value, group = athlete, color = athlete)
+ ) +
+   theme_bw(8) +
+   facet_wrap(~name, scales = "free_y") +
+   geom_line(alpha = 0.7) +
+   ylab(NULL) +
+   xlab("Distance (m)") +
+   theme(legend.position = "top", legend.title = element_blank())
```



3.2. Estimating short sprint parameters using radar gun

Estimation of the short sprint profile using radar gun data takes time as predictor and velocity as an target or outcome variable. Thus equation (1) is used to estimate MSS and TAU.

Let's consider the same example of an athlete with MSS equal to $9ms^{-1}$, TAU equal to 1.3, and MAC equal to $6.92ms^{-2}$ performing 40m sprint with velocity estimated using radar run (in this case with 1Hz sampling rate).

```
R> sprint_time <- seq(0, 6, 1)
R>
R> sprint_velocity <- c(0.00, 4.83, 7.07, 8.10, 8.59, 8.81, 8.91)
R>
R> m3 <- shorts::model_using_radar(
+   velocity = sprint_velocity,
+   time = sprint_time
+ )
R>
R> m3
```

```
#> Estimated model parameters
#> -----
#>           MSS           TAU           MAC
#>           9.00           1.30           6.92
#>           PMAX   time_correction distance_correction
#>           15.58           0.00           0.00
#>
#> Model fit estimators
#> -----
#>           RSE R_squared   minErr   maxErr maxAbsErr   RMSE
#>    0.00327   1.00000  -0.00406   0.00532   0.00532   0.00276
#>           MAE           MAPE
#>    0.00207           NaN
```

Both split and radar gun models allow the use of *weighted* non-linear regression. For example, we can give more weight to shorter distance or faster velocities. Weighted non-linear regression is performed by setting `weights` parameter:

```
R> m3_weighted <- shorts::model_using_radar(
+   velocity = sprint_velocity,
+   time = sprint_time,
+   weights = 1 / (sprint_velocity + 1)
+ )
R>
R> m3_weighted

#> Estimated model parameters
#> -----
#>           MSS           TAU           MAC
#>           9.00           1.30           6.92
#>           PMAX   time_correction distance_correction
```

```
#>          15.58          0.00          0.00
#>
#> Model fit estimators
#> -----
#>      RSE R_squared   minErr   maxErr maxAbsErr   RMSE
#> 0.00108 1.00000 -0.00406 0.00534 0.00534 0.00276
#>      MAE      MAPE
#> 0.00206      NaN
```

Mixed-effects model

Mixed-effects model using radar data is done using `shorts::mixed_model_using_radar` function. To perform mixed model, let's load data that comes with **shorts** package.

```
R> data("radar_gun_data")
R>
R> head(radar_gun_data)

#> # A tibble: 6 x 4
#>   athlete bodyweight   time velocity
#>   <chr>      <dbl> <dbl>    <dbl>
#> 1 John         75  0      0
#> 2 John         75 0.01    0.075
#> 3 John         75 0.02    0.149
#> 4 John         75 0.03    0.222
#> 5 John         75 0.04    0.291
#> 6 John         75 0.05    0.367

R>
R> m4 <- shorts::mixed_model_using_radar(
+   radar_gun_data,
+   time = "time",
+   velocity = "velocity",
+   athlete = "athlete"
+ )
R>
R> m4
```

```
#> Estimated fixed model parameters
#> -----
#>      MSS          TAU          MAC
#>      8.30          1.01          8.24
#>      PMAX   time_correction distance_correction
#>      17.09          0.00          0.00
#>
#> Estimated random model parameters
```

```
#> -----
#>   athlete   MSS   TAU   MAC PMAX time_correction
#> 1     James 10.00 1.111 9.00 22.5              0
#> 2       Jim  8.00 0.889 9.00 18.0              0
#> 3      John  8.00 1.069 7.48 15.0              0
#> 4 Kimberley 9.01 1.286 7.01 15.8              0
#> 5  Samantha 6.50 0.685 9.50 15.4              0
#> distance_correction
#> 1              0
#> 2              0
#> 3              0
#> 4              0
#> 5              0
#>
#> Model fit estimators
#> -----
#>      RSE R_squared   minErr   maxErr maxAbsErr      RMSE
#> 0.0516 0.9994 -0.2191 0.1983 0.2191 0.0516
#>      MAE      MAPE
#> 0.0395      NaN
```

4. Problems with estimation

There is a challenge when collecting sprint data that could have a substantial impact on modeled outcomes. To ensure accurate parameter outcomes, the initial force production must be synced with start time. Below we describe this challenge when using radar guns or timing gates and suggest potential solutions within the **shorts** package.

4.1. Problems with time sync with radar gun

One source of error in the modeled estimation using radar gun is the time synchronization. In theory, synchronization is ideal when sprint is initiated at $t = 0$ (i.e., $v(t = 0) = 0$). In practice, this is often not the case. Let's use our athlete and add and deduct 0.5second to simulate an error in synchronization and its effect on estimated MSS and TAU.

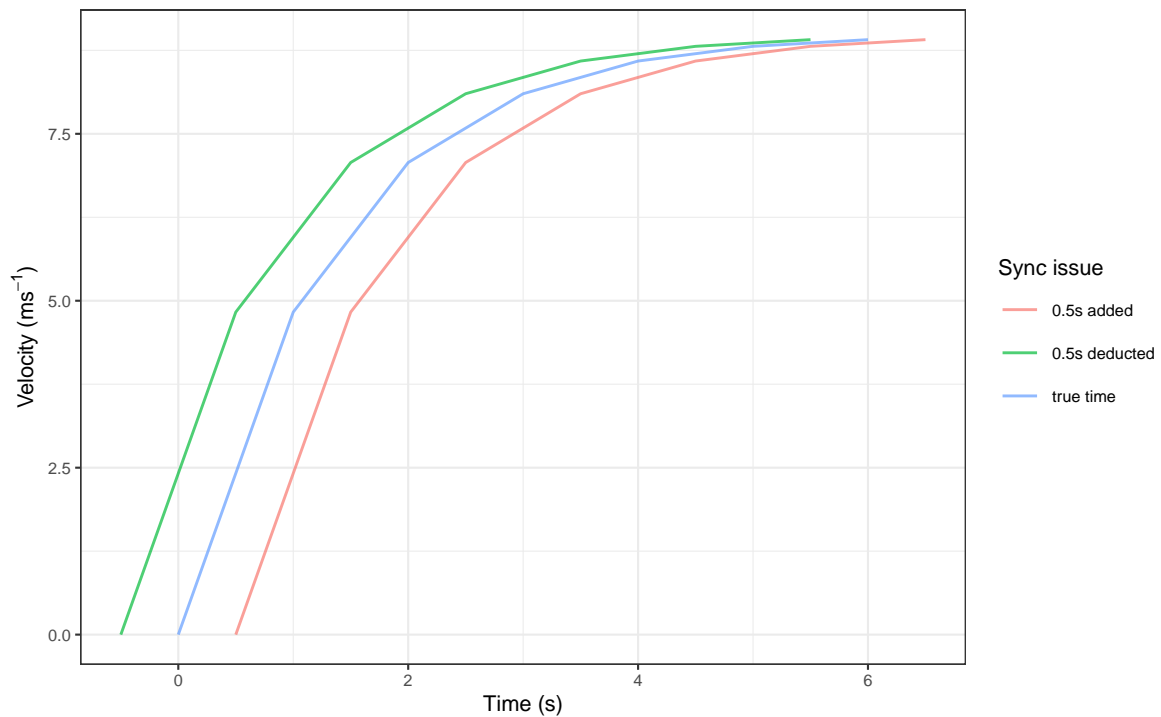
```
R> df <- tibble(
+   `true time` = sprint_time,
+   velocity = sprint_velocity,
+   `0.5s added` = `true time` + 0.5,
+   `0.5s deducted` = `true time` - 0.5
+ )
R>
R> head(df)

#> # A tibble: 6 x 4
#>   `true time` velocity `0.5s added` `0.5s deducted`
```



```
#>      <dbl>    <dbl>      <dbl>      <dbl>
#> 1         0         0         0.5        -0.5
#> 2         1     4.83         1.5         0.5
#> 3         2     7.07         2.5         1.5
#> 4         3     8.1          3.5         2.5
#> 5         4     8.59         4.5         3.5
#> 6         5     8.81         5.5         4.5
```

```
R> plot_df <- pivot_longer(df, cols = -2, names_to = "Sync issue")
R>
R> ggplot(plot_df, aes(x = value, y = velocity, color = `Sync issue`)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   xlab("Time (s)") +
+   ylab(expression("Velocity (" * ms^-1 * ")"))
```



The following three models estimate MSS and TAU from the three datasets:

```
R> # Without synchronization issues
R> m5 <- shorts::model_using_radar(
+   velocity = df$velocity,
+   time = df$`true time`
+ )
R>
R> # With time added
```

```

R> m6 <- shorts::model_using_radar(
+   velocity = df$velocity,
+   time = df$`0.5s added`
+ )
R>
R> # With time deducted
R> m7 <- shorts::model_using_radar(
+   velocity = df$velocity,
+   time = df$`0.5s deducted`
+ )
R>
R> rbind(
+   data.frame(
+     model = "True time",
+     t(coef(m5))
+   ),
+   data.frame(
+     model = "Added 0.5s time",
+     t(coef(m6))
+   ),
+   data.frame(
+     model = "Deducted 0.5s time",
+     t(coef(m7))
+   )
+ )

#>           model    MSS  TAU  MAC PMAX time_correction
#> 1      True time   9.00 1.30 6.92 15.6              0
#> 2  Added 0.5s time   9.91 2.34 4.23 10.5              0
#> 3 Deducted 0.5s time 10.08 1.86 5.43 13.7              0
#> distance_correction
#> 1              0
#> 2              0
#> 3              0

```

As can be seen from the example, TAU is affected by an error in synchronization of time with velocity. The potential solution incorporated into the **shorts** package involves estimation of the *time correction* parameter using the following equation:

$$v(t) = MSS \times (1 - e^{-\frac{t + \text{time correction}}{TAU}}) \quad (6)$$

This model is incorporated in the `shorts::model_using_radar_with_time_correction` function:

```

R> # With time added
R> m8 <- shorts::model_using_radar_with_time_correction(
+   velocity = df$velocity,

```

```

+   time = df$`0.5s added`
+ )
R> coef(m8)

#>              MSS              TAU              MAC
#>              9.00              1.30              6.92
#>          PMAX   time_correction distance_correction
#>          15.58             -0.50              0.00

R>
R> # With time deducted
R> m9 <- shorts::model_using_radar_with_time_correction(
+   velocity = df$velocity,
+   time = df$`0.5s deducted`
+ )
R> coef(m9)

#>              MSS              TAU              MAC
#>              9.00              1.30              6.92
#>          PMAX   time_correction distance_correction
#>          15.58              0.50              0.00

```

When using `shorts::predict_` family of functions, one can provide estimated time correction to get predictions at original time scale.

```

R> # Using the true time
R> predict_velocity_at_time(
+   time = df$`true time`,
+   MSS = m5$parameters$MSS,
+   TAU = m5$parameters$TAU
+ )

#> [1] 0.00 4.83 7.07 8.11 8.59 8.81 8.91

R>
R> # Using time with sync issues
R> predict_velocity_at_time(
+   time = df$`0.5s added`,
+   MSS = m8$parameters$MSS,
+   TAU = m8$parameters$TAU,
+   time_correction = m8$parameters$time_correction
+ )

#> [1] 7.82e-05 4.83e+00 7.07e+00 8.11e+00 8.59e+00 8.81e+00
#> [7] 8.91e+00

```

Mixed-model approach

When it comes to mixed-model approach, time correction can be modeled as a fixed effect or random effect using the `shorts::mixed_model_using_radar_with_time_correction` function.

```
R> # Adding 0.5s to radar_gun_data
R> radar_gun_data$time <- radar_gun_data$time + 0.5
R>
R> # Mixed model with time correction being fixed effect
R> m10 <- shorts::mixed_model_using_radar_with_time_correction(
+   radar_gun_data,
+   time = "time",
+   velocity = "velocity",
+   athlete = "athlete",
+   random = MSS + TAU ~ 1
+ )
R>
R> m10
```

```
#> Estimated fixed model parameters
#> -----
#>               MSS               TAU               MAC
#>             8.30             1.01             8.24
#>             PMAX   time_correction distance_correction
#>             17.10             -0.50             0.00
#>
```

```
#> Estimated random model parameters
#> -----
#>   athlete  MSS  TAU  MAC PMAX time_correction
#> 1   James 10.00 1.111 9.00 22.5          -0.5
#> 2    Jim  8.00 0.889 9.00 18.0          -0.5
#> 3   John  8.00 1.069 7.48 15.0          -0.5
#> 4 Kimberley 9.01 1.285 7.01 15.8          -0.5
#> 5 Samantha 6.50 0.685 9.50 15.4          -0.5
#> distance_correction
#> 1              0
#> 2              0
#> 3              0
#> 4              0
#> 5              0
#>
```

```
#> Model fit estimators
#> -----
#>      RSE R_squared   minErr   maxErr maxAbsErr   RMSE
#>    0.0516    0.9994   -0.2190    0.1983    0.2190    0.0516
#>      MAE      MAPE
#>    0.0395      Inf
```

```

R>
R>
R> # Mixed model with time correction being random effect
R> m11 <- shorts::mixed_model_using_radar_with_time_correction(
+   radar_gun_data,
+   time = "time",
+   velocity = "velocity",
+   athlete = "athlete",
+   random = MSS + TAU + time_correction ~ 1
+ )
R>
R> m11

#> Estimated fixed model parameters
#> -----
#>               MSS               TAU               MAC
#>               8.30               1.01               8.24
#>               PMAX      time_correction distance_correction
#>               17.10               -0.50               0.00
#>
#> Estimated random model parameters
#> -----
#>   athlete  MSS  TAU  MAC PMAX time_correction
#> 1   James 10.00 1.110 9.00 22.5          -0.5
#> 2    Jim  8.00 0.889 9.00 18.0          -0.5
#> 3   John  8.00 1.069 7.48 15.0          -0.5
#> 4 Kimberley 9.01 1.285 7.01 15.8          -0.5
#> 5 Samantha 6.50 0.685 9.50 15.4          -0.5
#> distance_correction
#> 1              0
#> 2              0
#> 3              0
#> 4              0
#> 5              0
#>
#> Model fit estimators
#> -----
#>      RSE R_squared   minErr   maxErr maxAbsErr   RMSE
#> 0.0516  0.9994   -0.2188    0.1982    0.2188    0.0516
#>      MAE      MAPE
#> 0.0395      Inf

```

4.2. Problems at the start when using split times

Let's imagine we have two twin brothers with same short sprint characteristics: MSS equal to $9ms^{-1}$, TAU equal to 1.3, and MAC equal to $6.92ms^{-2}$. Let's call them John and Jack.

They are both performing 40m sprint using timing gates set at 5, 10, 20, 30, and 40m. The initial timing gate at the start (i.e., $d = 0m$) serves to activate the timing system (i.e., when they cross the beam).

John represents the *theoretical model*, in which we assume that the initial force production and the timing initiation are perfectly synchronized. Jack, on the other hand, represents a *practical model*, and decides to move slightly behind the initial timing gate (i.e. for 0.5m) and use body rocking to initiate the sprint start. In other words, Jack is using a *flying start*, a common scenario when testing field sports athletes. Let's see how their sprints differ.

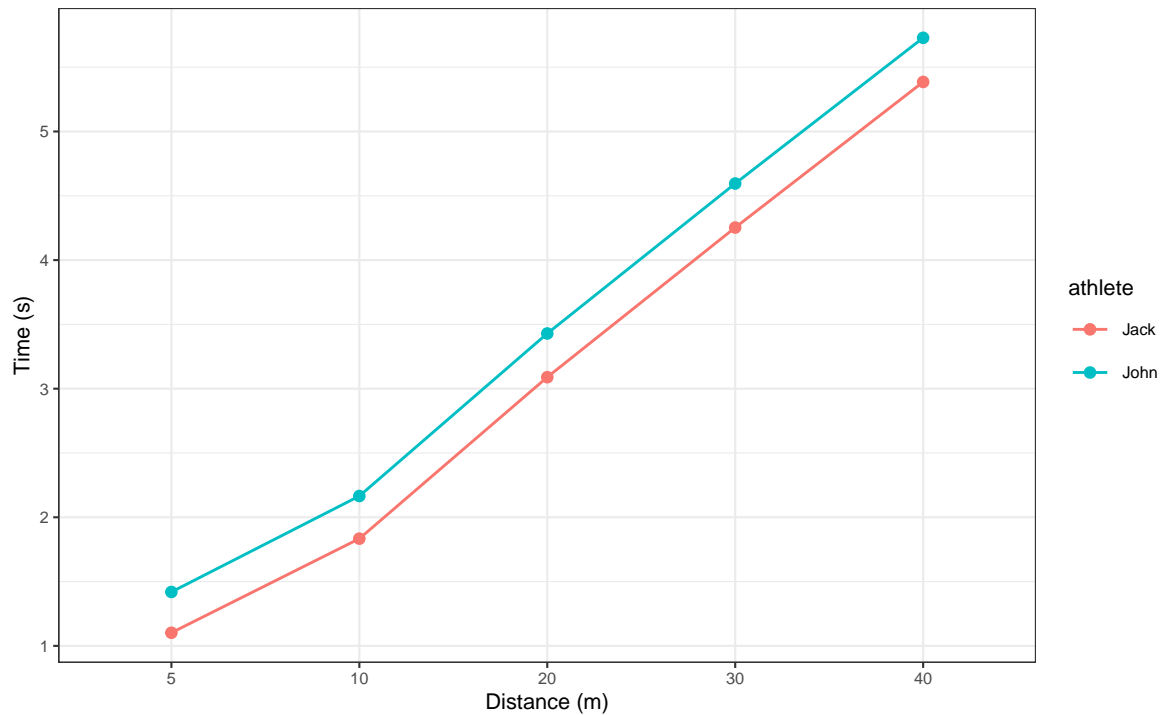
```
R> MSS <- 9
R> TAU <- 1.3
R> MAC <- MSS / TAU
R>
R> split_times <- tibble(
+   distance = c(5, 10, 20, 30, 40),
+   john_time = shorts::predict_time_at_distance(distance, MSS, TAU),
+   # Jack's performance
+   jack_distance = distance + 0.5,
+   jack_true_time = shorts::predict_time_at_distance(jack_distance, MSS, TAU),
+   time_05m = shorts::predict_time_at_distance(0.5, MSS, TAU),
+   jack_time = jack_true_time - time_05m
+ )
R>
R> split_times
```

```
#> # A tibble: 5 x 6
#>   distance john_time jack_distance jack_true_time time_05m
#>   <dbl>   <I<dbl>>      <dbl>         <I<dbl>> <I<dbl>>
#> 1      5      1.42        5.5           1.50    0.400
#> 2     10      2.17       10.5           2.23    0.400
#> 3     20      3.43       20.5           3.49    0.400
#> 4     30      4.60       30.5           4.65    0.400
#> 5     40      5.73       40.5           5.78    0.400
#> # ... with 1 more variable: jack_time <I<dbl>>
```

And here is a graphical representation of the sprint splits:

```
R> plot_df <- split_times %>%
+   select(distance, john_time, jack_time) %>%
+   rename(John = john_time, Jack = jack_time) %>%
+   pivot_longer(cols = -1, names_to = "athlete", values_to = "time") %>%
+   mutate(distance = factor(distance))
R>
R> ggplot(plot_df, aes(x = distance, y = time, color = athlete, group = athlete)) +
+   theme_bw(8) +
```

```
+ geom_point() +
+ geom_line() +
+ xlab("Distance (m)") +
+ ylab("Time (s)")
```



Using the following code, we can see how different are the estimated MSS and TAU parameters:

```
R> # Since this is a perfect simulation and stats::nls will complain
R> # we need to add very small noise, or measurement error to the times
R> set.seed(1667)
R> rand_noise <- rnorm(nrow(split_times), 0, 10^-5)
R> split_times$john_time <- split_times$john_time + rand_noise
R> split_times$jack_time <- split_times$jack_time + rand_noise
R>
R> john_profile <- shorts::model_using_splits(
+   distance = split_times$distance,
+   time = split_times$john_time
+ )
R>
R> jack_profile <- shorts::model_using_splits(
+   distance = split_times$distance,
+   time = split_times$jack_time
+ )
R>
```

```
R> sprint_parameters <- rbind(
+   unlist(john_profile$parameters),
+   unlist(jack_profile$parameters)
+ )
R>
R> rownames(sprint_parameters) <- c("John", "Jack")
R>
R> sprint_parameters
```

```
#>      MSS   TAU   MAC PMAX time_correction
#> John 9.00 1.300 6.92 15.6              0
#> Jack 8.49 0.704 12.06 25.6              0
#>      distance_correction
#> John              0
#> Jack              0
```

As can be seen from the results, flying starts yields biased estimates, particularly for the TAU, MAC and PMAX.

Below is a simulation sprint with 5, 10, 20, 30, 40, and 50m splits, with MSS and MAC varying from 6 to 9 (ms^{-1} and ms^{-2} respectively), and flying start distance varying from 0 to 1m.

```
R> sim_df <- expand.grid(
+   MSS = c(6, 7, 8, 9),
+   MAC = c(6, 7, 8, 9),
+   flying_start_distance = c(
+     seq(0, 0.001, length.out = 20),
+     seq(0.001, 0.01, length.out = 20),
+     seq(0.01, 0.1, length.out = 20),
+     seq(0.1, 1, length.out = 20)
+   ),
+   distance = c(5, 10, 20, 30, 40, 50)
+ )
R>
R> sim_df <- sim_df %>%
+   mutate(
+     TAU = MSS / MAC,
+     PMAX = MSS * MAC / 4,
+     true_distance = distance + flying_start_distance,
+     true_time = shorts::predict_time_at_distance(true_distance, MSS, TAU),
+     stolen_time = shorts::predict_time_at_distance(flying_start_distance, MSS, TAU),
+     time = true_time - stolen_time
+   )
R>
R> # Add small noise to allow model fit
R> set.seed(1667)
```



```

R> rand_noise <- rnorm(nrow(sim_df), 0, 10^-4)
R> sim_df$time <- sim_df$time + rand_noise
R>
R> head(sim_df)

#>   MSS MAC flying_start_distance distance   TAU PMAX
#> 1    6    6                      0       5 1.000  9.0
#> 2    7    6                      0       5 1.167 10.5
#> 3    8    6                      0       5 1.333 12.0
#> 4    9    6                      0       5 1.500 13.5
#> 5    6    7                      0       5 0.857 10.5
#> 6    7    7                      0       5 1.000 12.2
#>   true_distance   true_time stolen_time      time
#> 1              5 1.639197....          0 1.639356....
#> 2              5 1.579728....          0 1.579675....
#> 3              5 1.537453....          0 1.537537....
#> 4              5 1.505904....          0 1.505900....
#> 5              5 1.549960....          0 1.549860....
#> 6              5 1.488596....          0 1.488548....

```

Now when we have a simulation dataset, we can check the model estimates and predictions, given the flying start distance:

```

R> # Prediction wrapper
R> pred_wrapper <- function(data) {
+   model <- shorts::model_using_splits(
+     distance = data$distance,
+     time = data$time
+   )
+
+   params <- data.frame(t(unlist(model$parameters)))
+
+   predicted_time <- shorts::predict_time_at_distance(
+     distance = data$distance,
+     MSS = model$parameters$MSS,
+     TAU = model$parameters$TAU
+   )
+
+   colnames(params) <- c(
+     "est_MSS", "est_TAU", "est_MAC", "est_PMAX",
+     "est_time_correction", "est_distance_correction"
+   )
+
+   cbind(data, params, data.frame(predicted_time = as.numeric(predicted_time)))
+ }
R>
R> # estimated parameters and predited time

```

```

R> model_df <- sim_df %>%
+   group_by(MSS, TAU, flying_start_distance) %>%
+   do(pred_wrapper(.)) %>%
+   ungroup()
R>
R> # Prediction residuals
R> model_df$residuals <- model_df$predicted_time - model_df$time

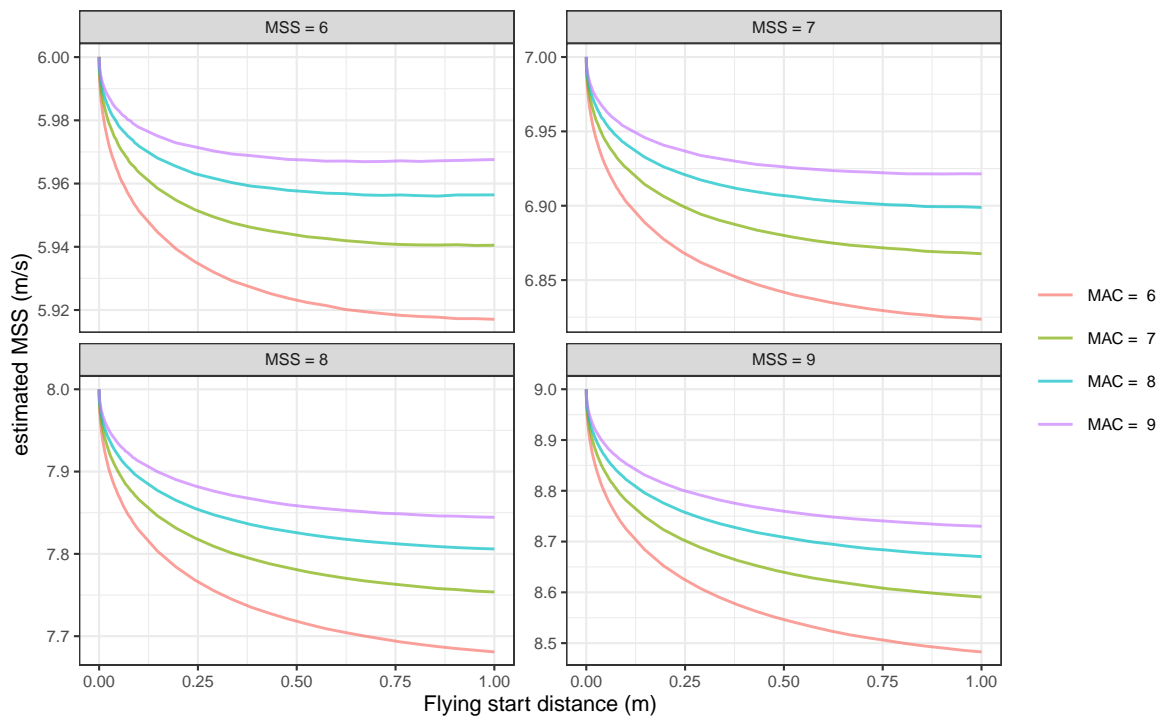
```

The following image demonstrates the effect of flying start distance on estimated MSS:

```

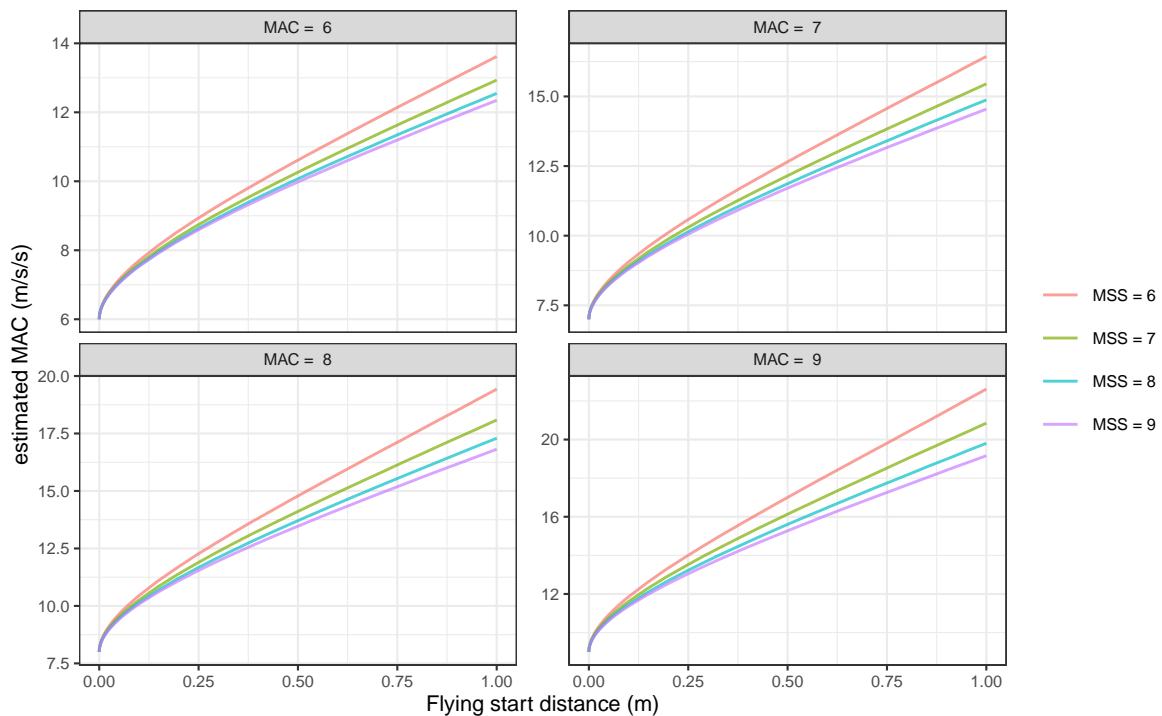
R> # Estimates plot
R> df <- model_df %>%
+   group_by(MSS, TAU, flying_start_distance) %>%
+   slice(1) %>%
+   mutate(
+     MSS_string = paste("MSS =", MSS),
+     TAU_string = paste("TAU =", TAU),
+     MAC_string = paste("MAC = ", round(MAC, 2)),
+     PMAX_string = paste("PMAX = ", round(PMAX, 2))
+   )
R>
R> # MSS
R> ggplot(df, aes(x = flying_start_distance, y = est_MSS, color = MAC_string)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_wrap(~MSS_string, scales = "free_y") +
+   xlab("Flying start distance (m)") +
+   ylab("estimated MSS (m/s)") +
+   theme(legend.title = element_blank())

```



As can be seen from the image, MSS is underestimated as flying start distance increases. The following image demonstrates the effect of flying start distance on estimated MAC:

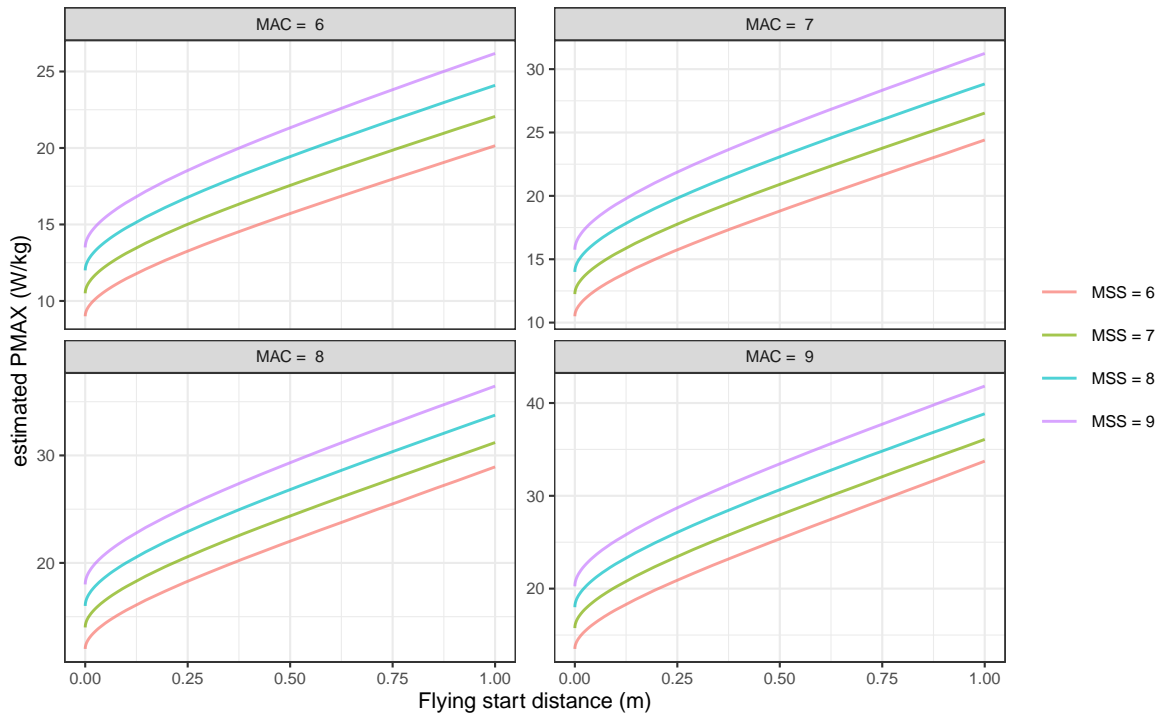
```
R> # MAC
R> ggplot(df, aes(x = flying_start_distance, y = est_MAC, color = MSS_string)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_wrap(~MAC_string, scales = "free_y") +
+   xlab("Flying start distance (m)") +
+   ylab("estimated MAC (m/s/s)") +
+   theme(legend.title = element_blank())
```



MAC (and also TAU) are highly affected by the flying start distance, and from the figure we can notice that MAC is overestimated as cheat distance increases.

And finally, the following image demonstrates the effect of flying start distance on estimated PMAX:

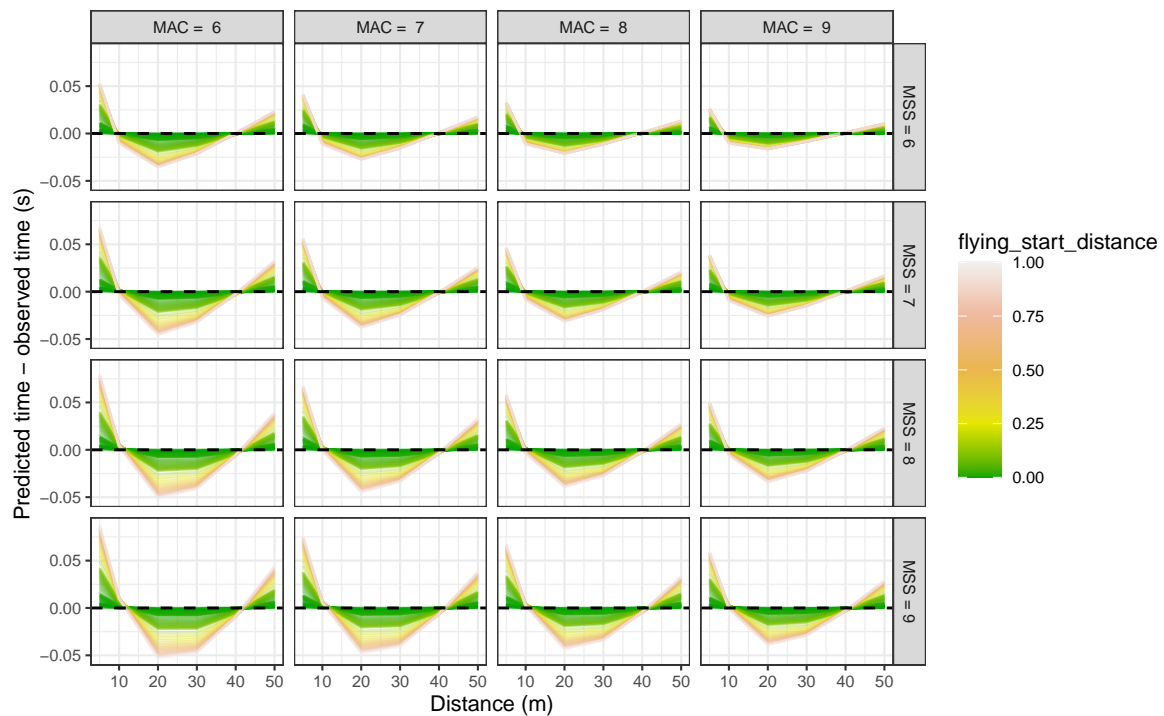
```
R> # PMAX
R> ggplot(df, aes(x = flying_start_distance, y = est_PMAX, color = MSS_string)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_wrap(~MAC_string, scales = "free_y") +
+   xlab("Flying start distance (m)") +
+   ylab("estimated PMAX (W/kg)") +
+   theme(legend.title = element_blank())
```



Estimated PMAX is also overestimated as cheat distance increases.

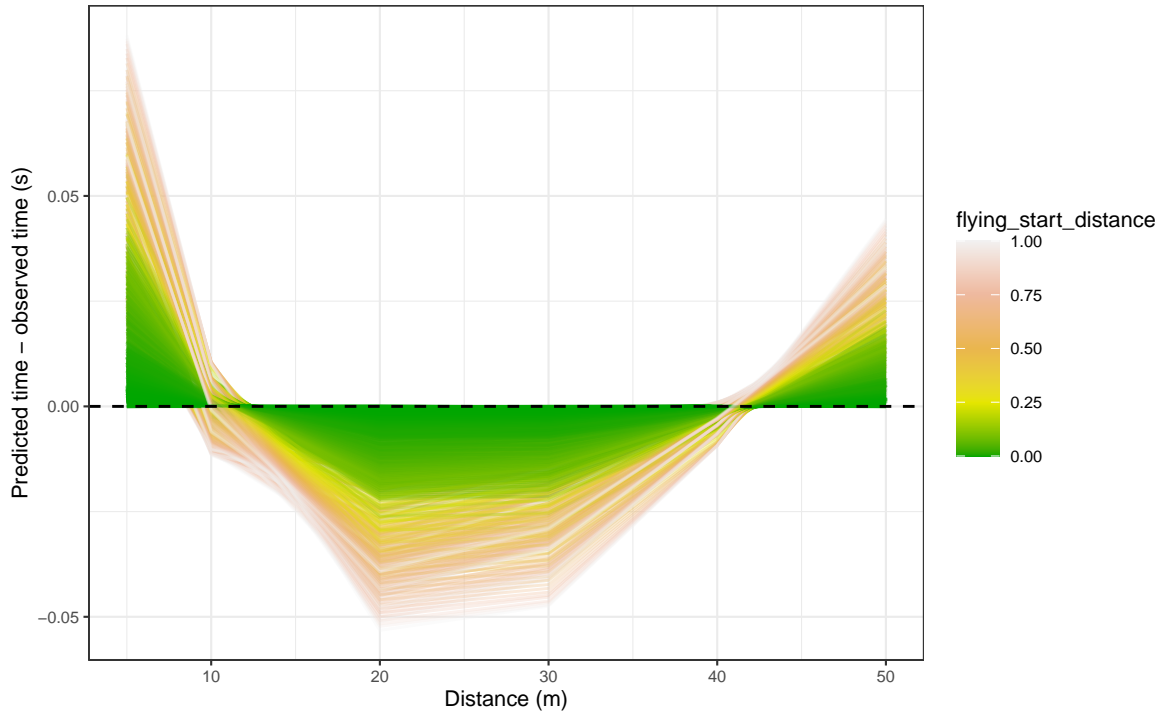
Model residuals are also affected by flying start distance. The shape of residuals distribution depends on number and splits utilized (e.g., 10, 20, 30, 40m versus 5, 15, 30m), but here we can see the effect of the flying start distance on the model residuals per split distance utilized in our simulation:

```
R> # Residuals
R> model_df <- model_df %>%
+   mutate(
+     MSS_string = paste("MSS =", MSS),
+     TAU_string = paste("TAU =", TAU),
+     MAC_string = paste("MAC = ", round(MAC, 2)),
+     PMAX_string = paste("PMAX = ", round(PMAX, 2)),
+     group = paste(MSS, MAC, flying_start_distance)
+   )
R>
R> ggplot(model_df, aes(y = residuals, x = distance, color = flying_start_distance, group
+   theme_bw(8) +
+   geom_line(alpha = 0.3) +
+   facet_grid(MSS_string ~ MAC_string) +
+   geom_hline(yintercept = 0, linetype = "dashed") +
+   scale_color_gradientn(colours = terrain.colors(5, rev = FALSE)) +
+   xlab("Distance (m)") +
+   ylab("Predicted time - observed time (s)"))
```



If we merge individual facets (i.e., combinations of MSS and MAC), we can get simpler images conveying issues with residuals when there is a flying start:

```
R> ggplot(model_df, aes(y = residuals, x = distance, color = flying_start_distance, group
+   theme_bw(8) +
+   geom_line(alpha = 0.3) +
+   geom_hline(yintercept = 0, linetype = "dashed") +
+   scale_color_gradientn(colours = terrain.colors(5, rev = FALSE)) +
+   xlab("Distance (m)") +
+   ylab("Predicted time - observed time (s)"))
```



Clearly, any type of flying start where there is a difference between initial force production and start time can result in biased parameters and predictions. Since maximal sprint speed is difficult to improve, the effects of start inconsistencies can mask effects of the training intervention. It is thus crucial to standardize the start when testing and implementing the following techniques when using the **shorts** package.

How to overcome missing the initial force production when using timing gates?

A potential solution is to use a correction factor - the recommendation in the literature is +0.5sec (Haugen, Breitschädel, and Seiler 2020b, 2019). Interestingly, the average difference between using timing gates and a block start for 40 m sprint time was 0.27 sec (Haugen, Tønnessen, and Seiler 2012). So, while a timing correction factor is warranted to avoid subsequent errors in estimates of kinetic variables (e.g., overestimate power), a correction factor that is too large will have the opposite effect (e.g., underestimate power).

When implementing *time correction*, equation (5) gets another form:

$$t(d) = TAU \times W(-e^{\frac{-d}{MSS \times TAU}} - 1) + \frac{d}{MSS} + TAU - \text{time correction} \quad (7)$$

Rather than providing *apriori* time correction from the literature, **shorts** package provides estimation of this parameter from the data provide, together with MSS and TAU. To estimate time correction, we use `shorts::model_using_splits_with_time_correction` function. Here is how we can estimate Jack parameters using either provided time correction (e.g., +0.3 and +0.5s) or estimated time correction:

```
R> jack_profile_fixed_time_short <- shorts::model_using_splits(
+   distance = split_times$distance,
```

```

+   time = split_times$jack_time,
+   time_correction = 0.3
+ )
R>
R> jack_profile_fixed_time_long <- shorts::model_using_splits(
+   distance = split_times$distance,
+   time = split_times$jack_time,
+   time_correction = 0.5
+ )
R>
R> jack_profile_time_estimated <- shorts::model_using_splits_with_time_correction(
+   distance = split_times$distance,
+   time = split_times$jack_time
+ )
R>
R> jack_parameters <- rbind(
+   unlist(john_profile$parameters),
+   unlist(jack_profile$parameters),
+   unlist(jack_profile_fixed_time_short$parameters),
+   unlist(jack_profile_fixed_time_long$parameters),
+   unlist(jack_profile_time_estimated$parameters)
+ )
R>
R> rownames(jack_parameters) <- c(
+   "John",
+   "Jack - No corrections",
+   "Jack - Fixed time correction (+0.3s)",
+   "Jack - Fixed time correction (+0.5s)",
+   "Jack - Estimated time correction"
+ )
R>
R> jack_parameters

#>           MSS    TAU    MAC PMAX
#> John          9.00 1.300  6.92 15.6
#> Jack - No corrections      8.49 0.704 12.06 25.6
#> Jack - Fixed time correction (+0.3s) 9.00 1.251  7.19 16.2
#> Jack - Fixed time correction (+0.5s) 9.62 1.770  5.43 13.1
#> Jack - Estimated time correction      8.96 1.216  7.37 16.5
#>           time_correction
#> John                      0.000
#> Jack - No corrections      0.000
#> Jack - Fixed time correction (+0.3s) 0.300
#> Jack - Fixed time correction (+0.5s) 0.500
#> Jack - Estimated time correction      0.284
#>           distance_correction
#> John                      0

```



```
#> Jack - No corrections 0
#> Jack - Fixed time correction (+0.3s) 0
#> Jack - Fixed time correction (+0.5s) 0
#> Jack - Estimated time correction 0
```

In Jack's case, both +0.3s fixed time correction and time correction estimation yield parameters closer to John's (i.e. true parameters).

Another model definition, which is novel approach implemented in the **shorts** packages, is to utilize *distance correction*, besides time correction. Thus, equation (5) gets another form:

$$t(d) = TAU \times W(-e^{\frac{-d + \text{distance correction}}{MSS \times TAU}} - 1) + \frac{d + \text{distance correction}}{MSS} + TAU - \text{time correction} \quad (8)$$

This model is implemented in `shorts::model_using_splits_with_corrections` function. Let's check this model estimates:

```
R> jack_profile_distance_correction <- shorts::model_using_splits_with_corrections(
+   distance = split_times$distance,
+   time = split_times$jack_time
+ )
R>
R> jack_parameters <- rbind(
+   unlist(john_profile$parameters),
+   unlist(jack_profile$parameters),
+   unlist(jack_profile_fixed_time_short$parameters),
+   unlist(jack_profile_fixed_time_long$parameters),
+   unlist(jack_profile_time_estimated$parameters),
+   unlist(jack_profile_distance_correction$parameters)
+ )
R>
R> rownames(jack_parameters) <- c(
+   "John",
+   "Jack - No corrections",
+   "Jack - Fixed time correction (+0.3s)",
+   "Jack - Fixed time correction (+0.5s)",
+   "Jack - Estimated time correction",
+   "Jack - Estimated distance correction"
+ )
R>
R> jack_parameters
```

#>	MSS	TAU	MAC	PMAX
#> John	9.00	1.300	6.92	15.6
#> Jack - No corrections	8.49	0.704	12.06	25.6
#> Jack - Fixed time correction (+0.3s)	9.00	1.251	7.19	16.2
#> Jack - Fixed time correction (+0.5s)	9.62	1.770	5.43	13.1

```

#> Jack - Estimated time correction      8.96 1.216  7.37 16.5
#> Jack - Estimated distance correction  9.00 1.301  6.92 15.6
#>                                     time_correction
#> John                                     0.000
#> Jack - No corrections                   0.000
#> Jack - Fixed time correction (+0.3s)    0.300
#> Jack - Fixed time correction (+0.5s)    0.500
#> Jack - Estimated time correction        0.284
#> Jack - Estimated distance correction    0.400
#>                                     distance_correction
#> John                                     0.000
#> Jack - No corrections                   0.000
#> Jack - Fixed time correction (+0.3s)    0.000
#> Jack - Fixed time correction (+0.5s)    0.000
#> Jack - Estimated time correction        0.000
#> Jack - Estimated distance correction    0.503

```

As can be seen from the results, adding distance correction results in correctly estimating Jack's sprint parameters. There are few issues with this model definition. Besides being novel and still not validated with practical data, distance correction model has four parameters to estimate, which implies that one needs at least five sprint splits. This imposes practical limitations, since acquiring six timing gate (one for the start and five for splits) might be practically troublesome.

We will get back to these issues later, but let's see how these models perform using simulated data with varying flying start distance. The following code contains the wrapper that performs all four models (no correction, fixed time correction, estimated time correction, and estimated time and distance correction):

```

R> pred_wrapper <- function(data) {
+   no_correction <- shorts::model_using_splits(
+     distance = data$distance,
+     time = data$time
+   )
+
+   fixed_correction_short <- shorts::model_using_splits(
+     distance = data$distance,
+     time = data$time,
+     time_correction = 0.3
+   )
+
+   fixed_correction_long <- shorts::model_using_splits(
+     distance = data$distance,
+     time = data$time,
+     time_correction = 0.5
+   )
+
+   time_correction <- shorts::model_using_splits_with_time_correction(

```

```

+   distance = data$distance,
+   time = data$time,
+   control = nls.control(tol = 1)
+ )
+
+ time_dist_correction <- shorts::model_using_splits_with_corrections(
+   distance = data$distance,
+   time = data$time,
+   control = nls.control(tol = 1)
+ )
+
+ params <- rbind(
+   data.frame(
+     model = "No correction",
+     t(unlist(no_correction$parameters))
+   ),
+   data.frame(
+     model = "Fixed correction +0.3s",
+     t(unlist(fixed_correction_short$parameters))
+   ),
+   data.frame(
+     model = "Fixed correction +0.5s",
+     t(unlist(fixed_correction_long$parameters))
+   ),
+   data.frame(
+     model = "Time correction",
+     t(unlist(time_correction$parameters))
+   ),
+   data.frame(
+     model = "Time and distance correction",
+     t(unlist(time_dist_correction$parameters))
+   )
+ )
+
+ colnames(params) <- c(
+   "model", "est_MSS", "est_TAU", "est_MAC", "est_PMAX",
+   "est_time_correction", "est_distance_correction"
+ )
+
+ df <- expand_grid(
+   data,
+   params
+ )
+
+ df$predicted_time <- shorts::predict_time_at_distance(
+   distance = df$distance,

```

```

+   MSS = df$est_MSS,
+   TAU = df$est_TAU,
+   time_correction = df$est_time_correction,
+   distance_correction = df$est_distance_correction
+ )
+
+ df$residuals <- df$predicted_time - df$time
+ return(df)
+ }
R>
R> # estimated parameters and predicted time
R> model_df <- sim_df %>%
+   group_by(MSS, TAU, flying_start_distance) %>%
+   do(pred_wrapper(.)) %>%
+   ungroup()

```

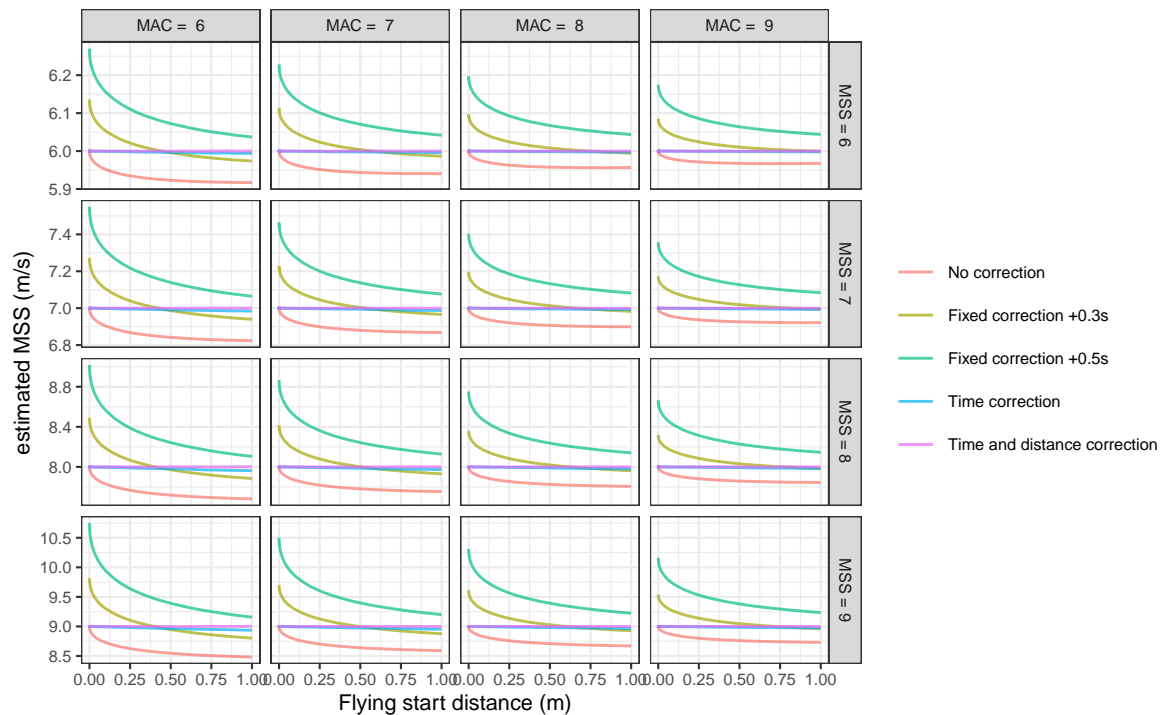
As can be seen from the next figure, estimated time correction model estimates MSS almost perfectly, while estimated time and distance correction model estimates MSS perfectly.

```

R> model_df$model <- factor(
+   model_df$model,
+   levels = c(
+     "No correction",
+     "Fixed correction +0.3s",
+     "Fixed correction +0.5s",
+     "Time correction",
+     "Time and distance correction"
+   )
+ )
R> # Estimates plot
R> df <- model_df %>%
+   group_by(MSS, TAU, flying_start_distance, model) %>%
+   slice(1) %>%
+   mutate(
+     MSS_string = paste("MSS =", MSS),
+     TAU_string = paste("TAU =", TAU),
+     MAC_string = paste("MAC = ", round(MAC, 2)),
+     PMAX_string = paste("PMAX = ", round(PMAX, 2))
+   )
R>
R> # MSS
R> ggplot(df, aes(x = flying_start_distance, y = est_MSS, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_grid(MSS_string ~ MAC_string, scales = "free_y") +
+   xlab("Flying start distance (m)") +

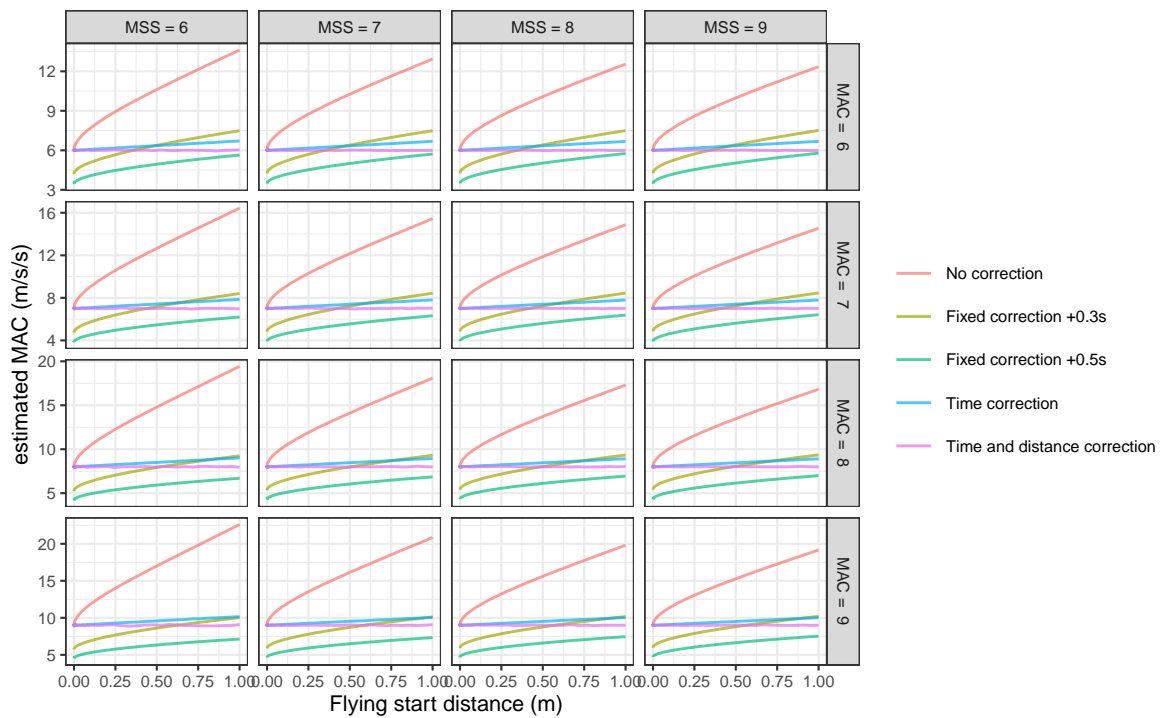
```

```
+ ylab("estimated MSS (m/s)") +
+ theme(legend.title = element_blank())
```



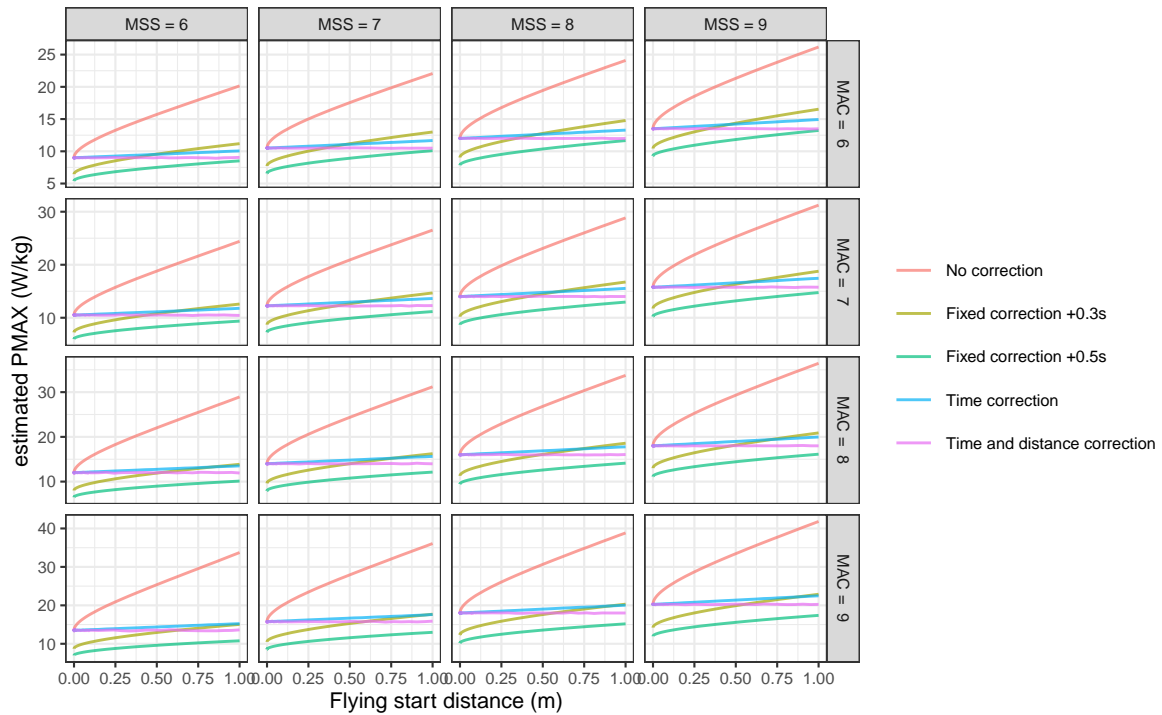
Same conclusions can be said for the MAC parameter. Time and distance corrections model performs perfectly, while time correction model performs almost as good (although worse than for the MSS parameter).

```
R> # MAC
R> ggplot(df, aes(x = flying_start_distance, y = est_MAC, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
+   xlab("Flying start distance (m)") +
+   ylab("estimated MAC (m/s/s)") +
+   theme(legend.title = element_blank())
```



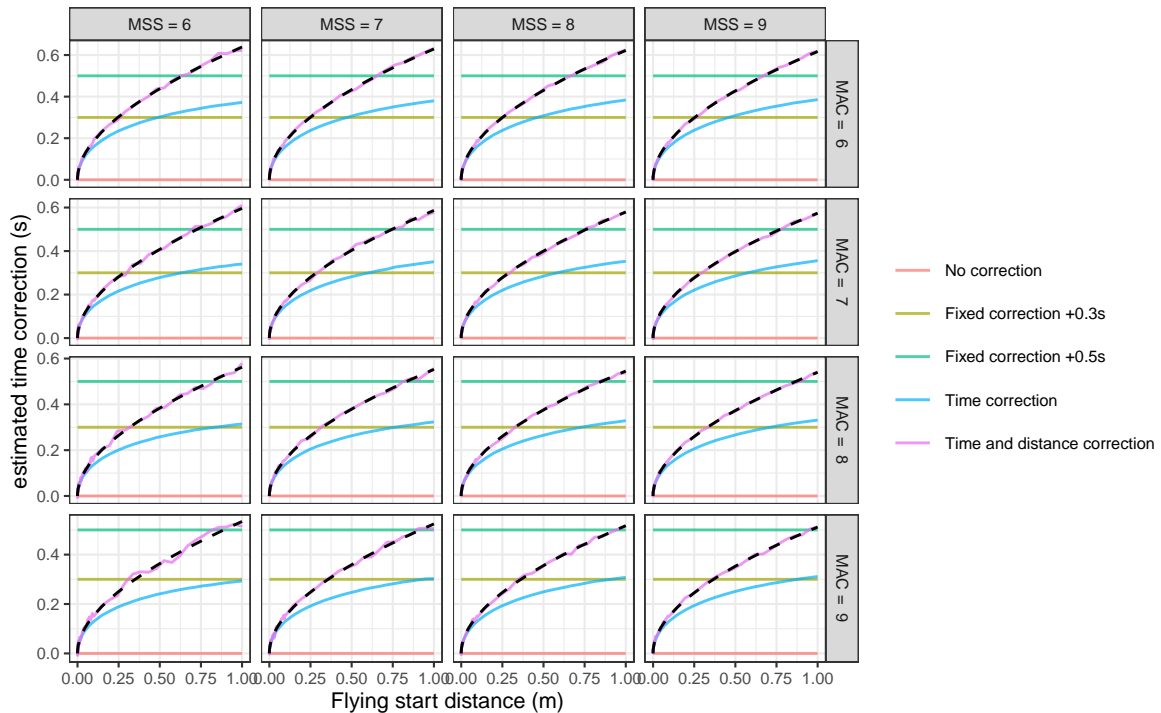
When it comes to P_{MAX}, same conclusion can be reached.

```
R> # PMAX
R> ggplot(df, aes(x = flying_start_distance, y = est_PMAX, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
+   xlab("Flying start distance (m)") +
+   ylab("estimated PMAX (W/kg)") +
+   theme(legend.title = element_blank())
```



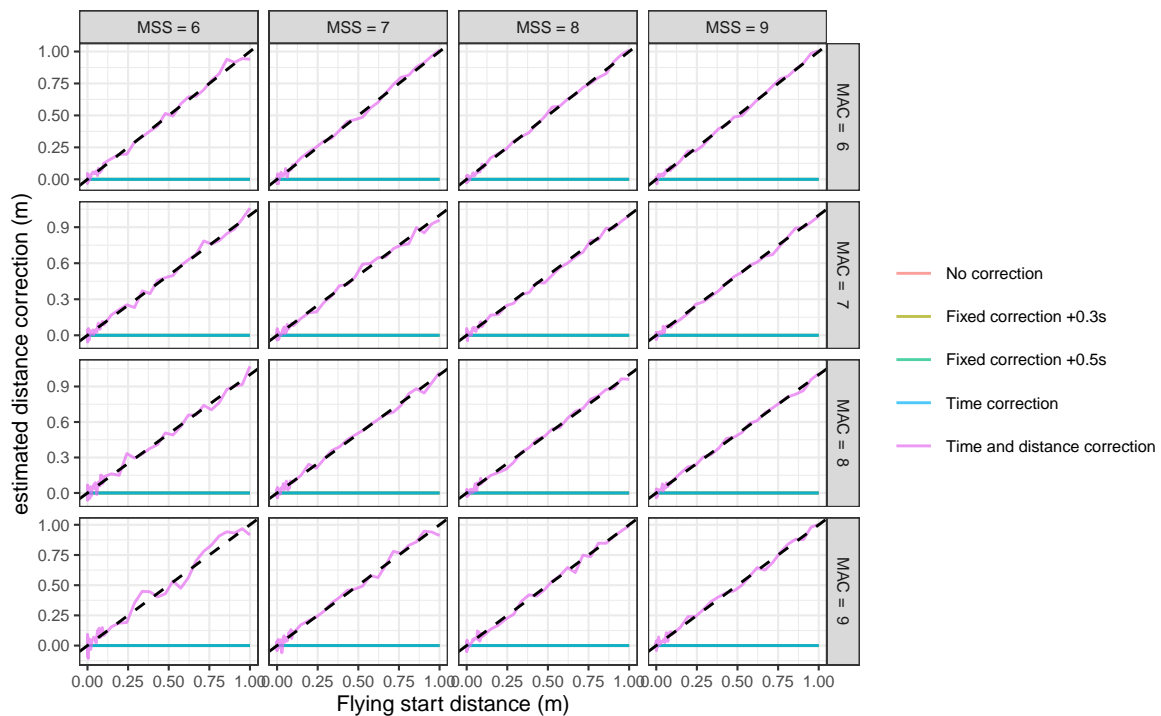
The following figure depicts estimated time correction, and as can be seen, only the time and distance correction model estimated the time correction correctly (i.e., the *stolen time*; indicated by the dashed line on the figure).

```
R> # time_correction
R> ggplot(df, aes(x = flying_start_distance, y = est_time_correction, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   geom_line(aes(y = stolen_time), color = "black", linetype = "dashed") +
+   facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
+   xlab("Flying start distance (m)") +
+   ylab("estimated time correction (s)") +
+   theme(legend.title = element_blank())
```



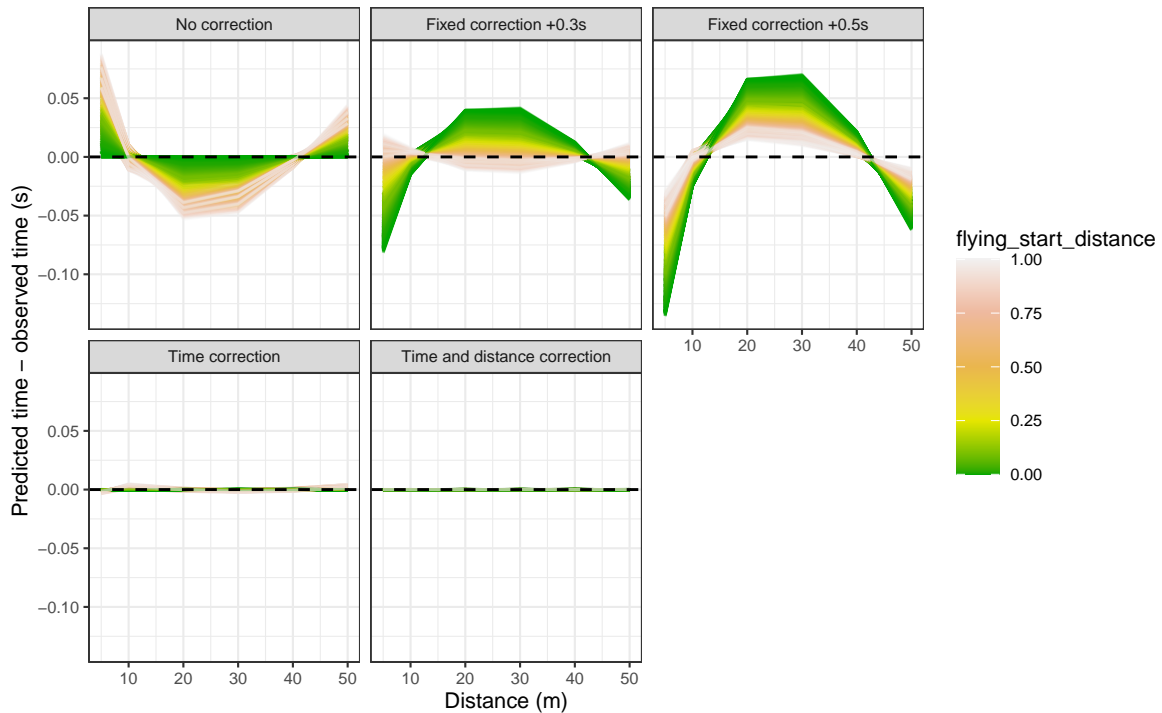
The following figure depicts estimated distance correction, and same as with the time correction, only the time and distance correction model estimated the distance correction correctly (i.e., flying start distance; indicated by the dashed line on the figure, which represents *identity line* since flying start distance is already on the x-axis).

```
R> # distance_correction
R> ggplot(df, aes(x = flying_start_distance, y = est_distance_correction, color = model))
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   geom_abline(slope = 1, color = "black", linetype = "dashed") +
+   facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
+   xlab("Flying start distance (m)") +
+   ylab("estimated distance correction (m)") +
+   theme(legend.title = element_blank())
```

The following figure depicts model residuals against the distance, and as can be seen, time correction and time and distance correction models performs much better than no correction and fixed correction models:

```
R> # Residuals
R> model_df <- model_df %>%
+   mutate(
+     MSS_string = paste("MSS =", MSS),
+     TAU_string = paste("TAU =", TAU),
+     MAC_string = paste("MAC = ", round(MAC, 2)),
+     PMAX_string = paste("PMAX = ", round(PMAX, 2)),
+     group = paste(MSS, MAC, flying_start_distance)
+   )
R>
R> ggplot(model_df, aes(y = residuals, x = distance, color = flying_start_distance, group
+   theme_bw(8) +
+   geom_line(alpha = 0.3) +
+   facet_wrap(~model) +
+   geom_hline(yintercept = 0, linetype = "dashed") +
+   scale_color_gradientn(colours = terrain.colors(5, rev = FALSE)) +
+   xlab("Distance (m)") +
+   ylab("Predicted time - observed time (s)"))
```



Given the simulation data, time correction and time and distance correction models represent sound improvements in parameter estimation and model fit compared to no corrections model and fixed correction model in the face of flying start issues. Since time correction model is simpler and demand three parameters to be estimated, it might be practically more useful than time and distance correction model, which demand four parameters estimation and thus more than five timing gates and sprint splits.

Time correction and time and distance corrections are also implemented in the mixed-models using `shorts::mixed_model_using_splits_with_time_correction` and `shorts::mixed_model_using_splits_with_time_and_distance_correction`. We will showcase their use at the end of this article.

Simulation of additional starting issues

Starting behind the initial timing gate represent only one issue (i.e. flying start). In this section, we simulate a few other issues to check how sensitive the presented models are to perturbations that can happen in practical settings during field testing.

Triggering the gate before the start One issue that might happen in certain timing gates setups is an individual triggering timing system before the sprint is initiated (e.g., by cutting the beam with an arm swing prematurely). This is very similar to the situation when timing starts on a signal (i.e., gun during 100m sprint race) and there is *reaction time* (RT) involved. Both of these scenarios represent *time lag* that is added to the split times. Let's simulate the effect of this time lag on model estimates and predictions.

```
R> sim_df <- expand.grid(
+   MSS = c(6, 7, 8, 9),
```

```

+   MAC = c(6, 7, 8, 9),
+   time_lag = seq(0, 0.5, length.out = 50),
+   distance = c(5, 10, 20, 30, 40, 50)
+ )
R>
R> sim_df <- sim_df %>%
+   mutate(
+     TAU = MSS / MAC,
+     PMAX = MSS * MAC / 4,
+     true_time = shorts::predict_time_at_distance(distance, MSS, TAU),
+     time = true_time + time_lag
+   )
R>
R> # Add small noise to allow model fit
R> set.seed(1667)
R> rand_noise <- rnorm(nrow(sim_df), 0, 10^-4)
R> sim_df$time <- sim_df$time + rand_noise
R>
R> head(sim_df)

#>   MSS MAC time_lag distance   TAU PMAX   true_time
#> 1    6  6         0         5 1.000  9.0 1.639197....
#> 2    7  6         0         5 1.167 10.5 1.579728....
#> 3    8  6         0         5 1.333 12.0 1.537453....
#> 4    9  6         0         5 1.500 13.5 1.505904....
#> 5    6  7         0         5 0.857 10.5 1.549960....
#> 6    7  7         0         5 1.000 12.2 1.488596....
#>
#>           time
#> 1 1.639356....
#> 2 1.579675....
#> 3 1.537537....
#> 4 1.505900....
#> 5 1.549860....
#> 6 1.488548....

R> # estimated parameters and predicted time
R> model_df <- sim_df %>%
+   group_by(MSS, TAU, time_lag) %>%
+   do(pred_wrapper(.)) %>%
+   ungroup()

```

From the figure below it can be seen that time lag affects estimated MSS for the the model without correction and fixed correction model. Time correction and time and distance corrections model correctly estimated MSS.

```

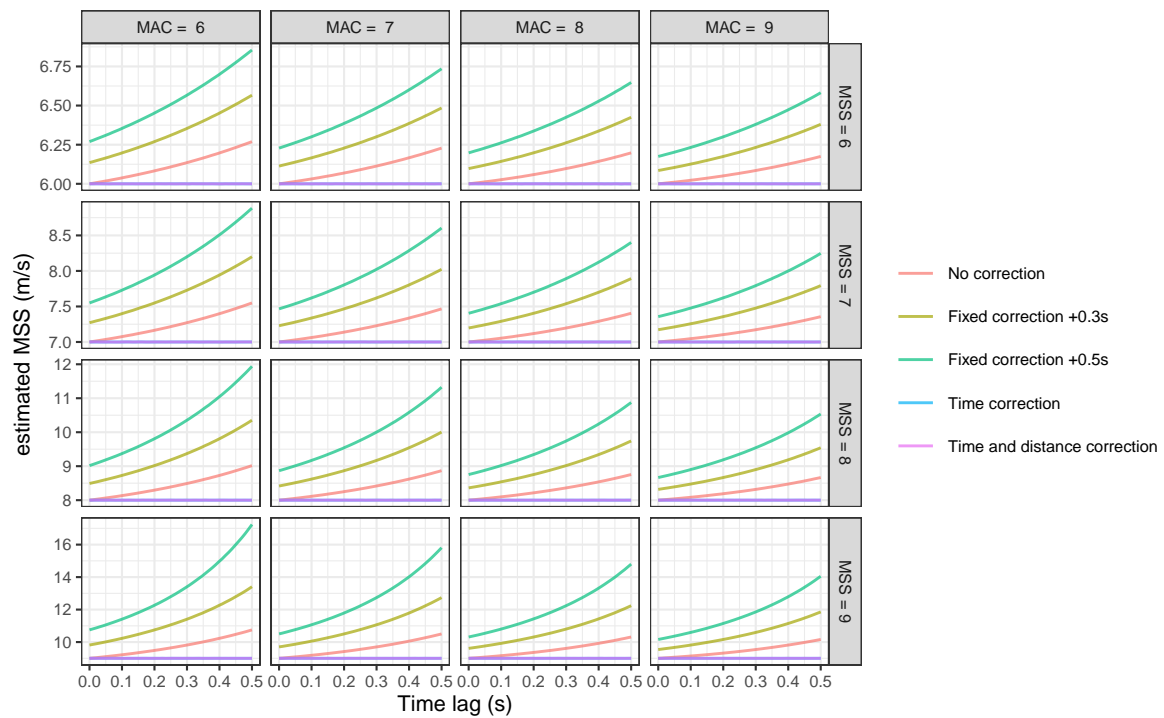
R> model_df$model <- factor(
+   model_df$model,

```

```

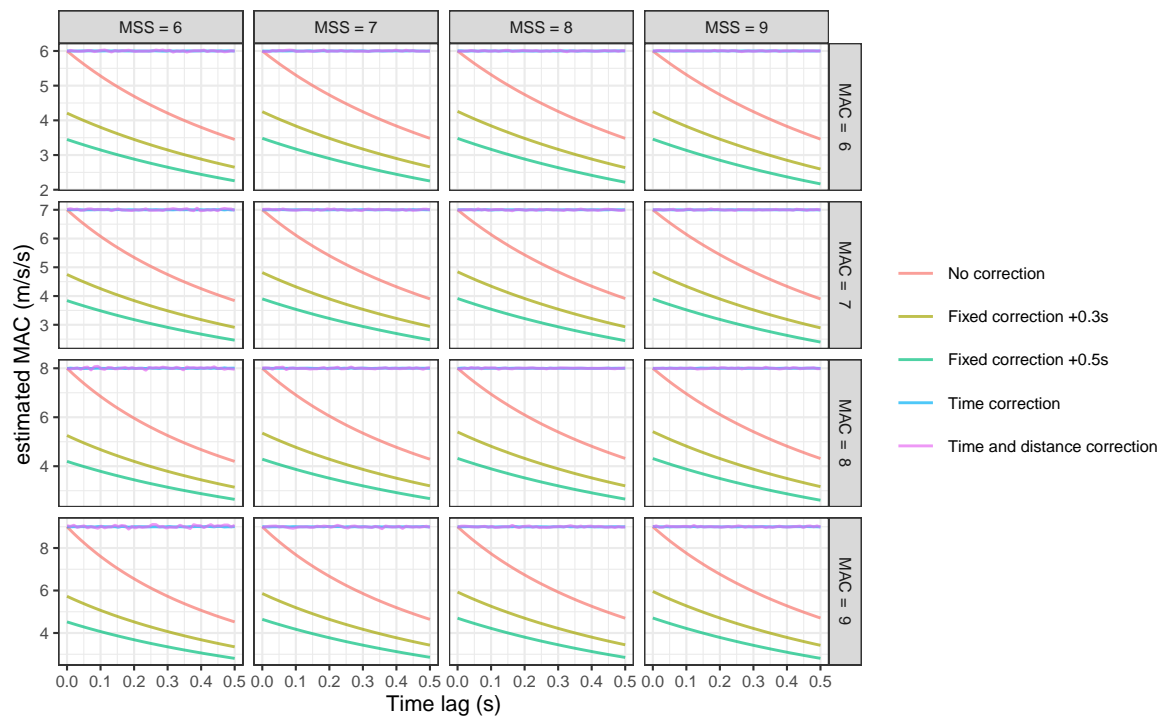
+   levels = c(
+     "No correction",
+     "Fixed correction +0.3s",
+     "Fixed correction +0.5s",
+     "Time correction",
+     "Time and distance correction"
+   )
+ )
R> # Estimates plot
R> df <- model_df %>%
+   group_by(MSS, TAU, time_lag, model) %>%
+   slice(1) %>%
+   mutate(
+     MSS_string = paste("MSS =", MSS),
+     TAU_string = paste("TAU =", TAU),
+     MAC_string = paste("MAC = ", round(MAC, 2)),
+     PMAX_string = paste("PMAX = ", round(PMAX, 2))
+   )
R>
R> # MSS
R> ggplot(df, aes(x = time_lag, y = est_MSS, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_grid(MSS_string ~ MAC_string, scales = "free_y") +
+   xlab("Time lag (s)") +
+   ylab("estimated MSS (m/s)") +
+   theme(legend.title = element_blank())

```



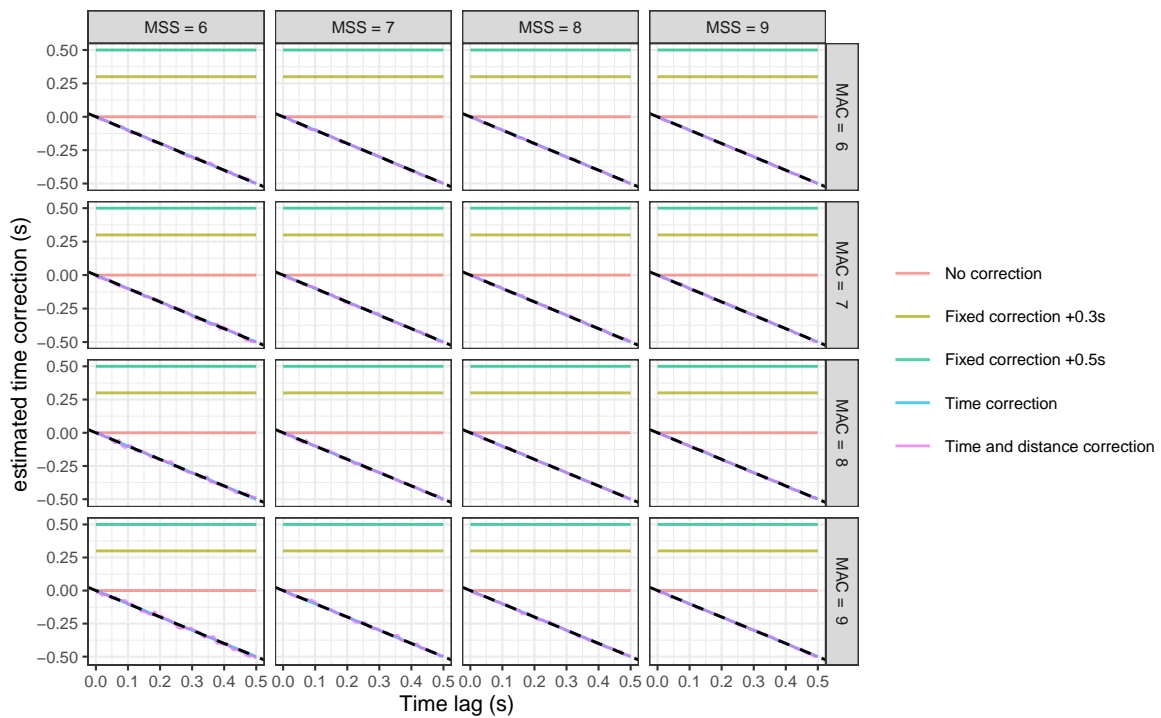
From the figure below it can be seen that time lag affects estimated MAC for the the model without correction and fixed correction models. Time correction and time and distance corrections model correctly estimated MAC.

```
R> # MAC
R> ggplot(df, aes(x = time_lag, y = est_MAC, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
+   xlab("Time lag (s)") +
+   ylab("estimated MAC (m/s/s)") +
+   theme(legend.title = element_blank())
```



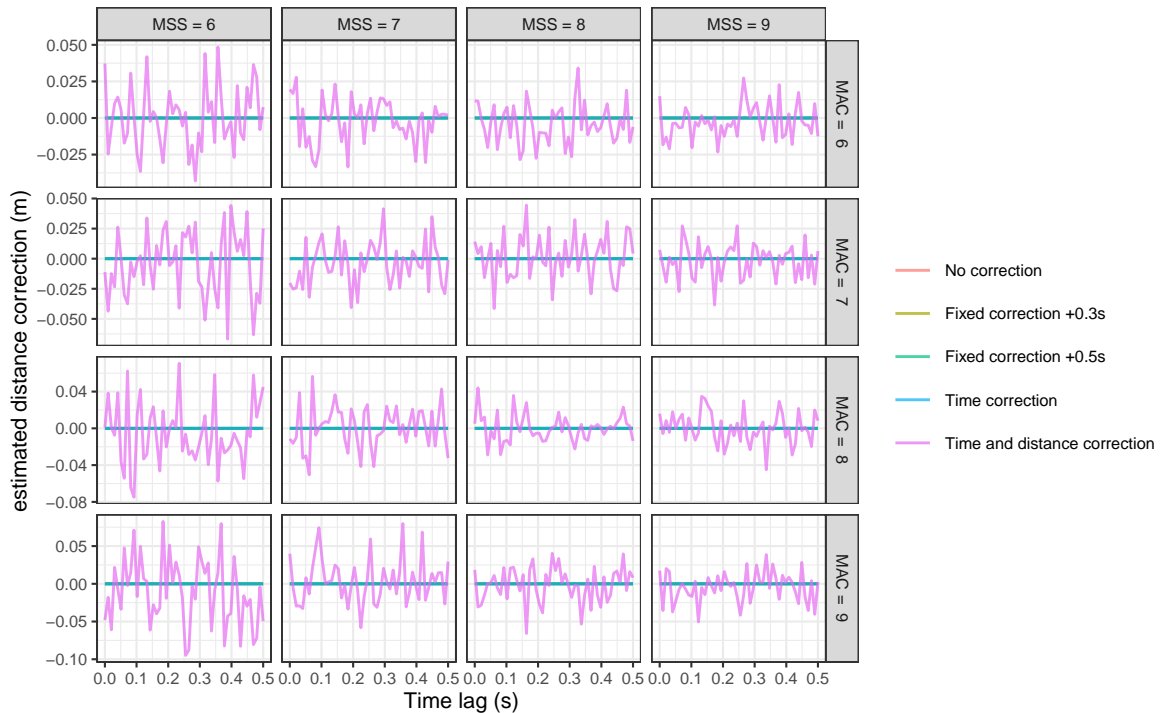
Two figure below depicts correctly identified time lag (i.e. using time correction parameter) using time correction and time and distance corrections models.

```
R> # time_correction
R> ggplot(df, aes(x = time_lag, y = est_time_correction, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   geom_abline(slope = -1, color = "black", linetype = "dashed") +
+   facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
+   xlab("Time lag (s)") +
+   ylab("estimated time correction (s)") +
+   theme(legend.title = element_blank())
```



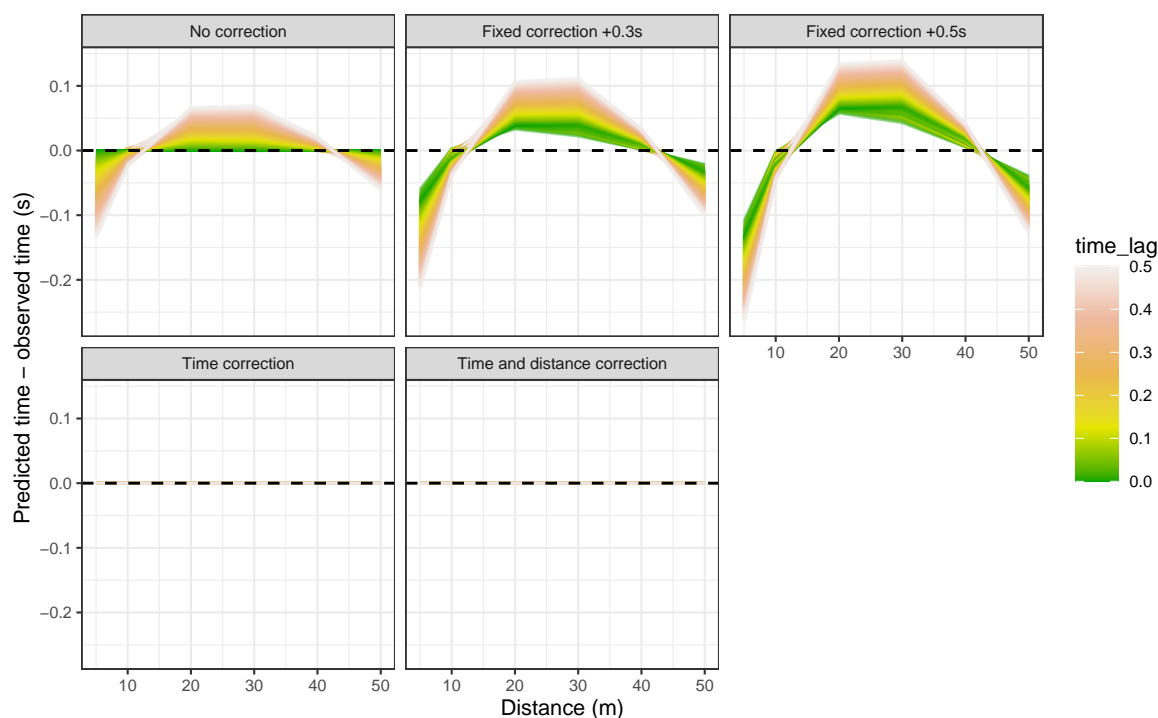
The next figure depicts estimated distance correction for the time and distance correction model. The estimated distance correction parameters looks jumpy due to random noise that we have to added to allow model fit.

```
R> # distance_correction
R> ggplot(df, aes(x = time_lag, y = est_distance_correction, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
+   xlab("Time lag (s)") +
+   ylab("estimated distance correction (m)") +
+   theme(legend.title = element_blank())
```



The following figure depicts residuals (i.e., predicted time minus observed time).

```
R> # Residuals
R> model_df <- model_df %>%
+   mutate(
+     MSS_string = paste("MSS =", MSS),
+     TAU_string = paste("TAU =", TAU),
+     MAC_string = paste("MAC = ", round(MAC, 2)),
+     PMAX_string = paste("PMAX = ", round(PMAX, 2)),
+     group = paste(MSS, MAC, time_lag)
+   )
R>
R> ggplot(model_df, aes(y = residuals, x = distance, color = time_lag, group = group)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.3) +
+   facet_wrap(~model) +
+   geom_hline(yintercept = 0, linetype = "dashed") +
+   scale_color_gradientn(colours = terrain.colors(5, rev = FALSE)) +
+   xlab("Distance (m)") +
+   ylab("Predicted time - observed time (s)")
```

Additional scenario can create time lag: if the initial timing gate has a time delay (i.e., once triggered, there is a time delay before the timing starts). In this case, time lag is a negative number since it reduces the split times. This scenario will not be simulated in the current paper.

Bad position of the timing gates Another common issue with timing gates in the practical field settings is the bad measurement of the distance and thus bad positions of the timing gates. In this example, we will simulate the scenario where the initial timing gate is not positioned at $d = 0$, but rather with some error (i.e., -0.5 to 0.5 m). Although this scenario represents poor procedure implementation, we have included this simulation to depict effects on estimated parameters.

```
R> sim_df <- expand.grid(
+   MSS = c(6, 7, 8, 9),
+   MAC = c(6, 7, 8, 9),
+   displacement = seq(-0.5, 0.5, length.out = 51),
+   distance = c(5, 10, 20, 30, 40, 50)
+ )
R>
R> sim_df <- sim_df %>%
+   mutate(
+     TAU = MSS / MAC,
+     PMAX = MSS * MAC / 4,
+     true_distance = distance + displacement,
+     time = shorts::predict_time_at_distance(true_distance, MSS, TAU)
```

```

+   )
R>
R> # Add small noise to allow model fit
R> set.seed(1667)
R> rand_noise <- rnorm(nrow(sim_df), 0, 10^-5)
R> sim_df$time <- sim_df$time + rand_noise
R>
R> head(sim_df, 2)

#>   MSS MAC displacement distance  TAU PMAX true_distance
#> 1    6    6          -0.5        5 1.00   9.0           4.5
#> 2    7    6          -0.5        5 1.17  10.5           4.5
#>           time
#> 1 1.534434....
#> 2 1.481967....

R> # estimated parameters and predicted time
R> model_df <- sim_df %>%
+   group_by(MSS, TAU, displacement) %>%
+   do(pred_wrapper(.)) %>%
+   ungroup()

```

Figure below depicts effect of timing gate displacement on the MSS parameter. Time correction and time and distance correction models are almost identical in their correct estimation of the MSS.

```

R> model_df$model <- factor(
+   model_df$model,
+   levels = c(
+     "No correction",
+     "Fixed correction +0.3s",
+     "Fixed correction +0.5s",
+     "Time correction",
+     "Time and distance correction"
+   )
+ )
R>
R> # Estimates plot
R> df <- model_df %>%
+   group_by(MSS, TAU, displacement, model) %>%
+   slice(1) %>%
+   mutate(
+     MSS_string = paste("MSS =", MSS),
+     TAU_string = paste("TAU =", TAU),
+     MAC_string = paste("MAC = ", round(MAC, 2)),
+     PMAX_string = paste("PMAX = ", round(PMAX, 2))
+   )

```

```
R>
R> # MSS
R> ggplot(df, aes(x = displacement, y = est_MSS, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_grid(MSS_string ~ MAC_string, scales = "free_y") +
+   xlab("Displacement (m)") +
+   ylab("estimated MSS (m/s)") +
+   theme(legend.title = element_blank())
```

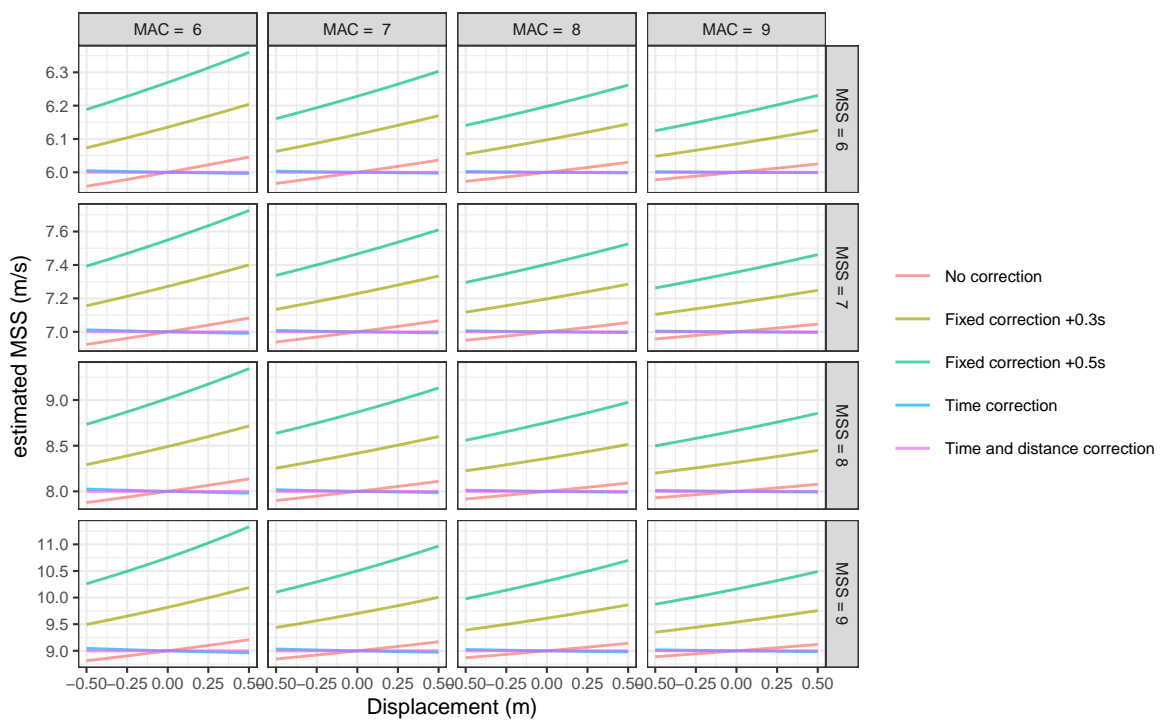
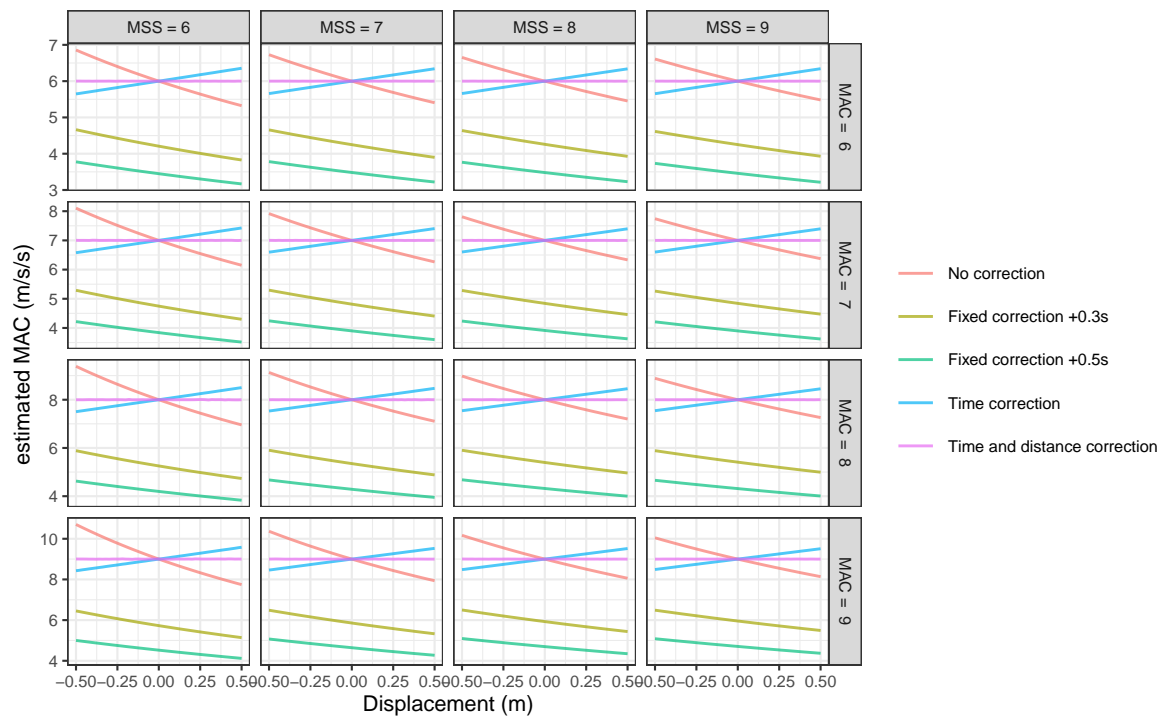


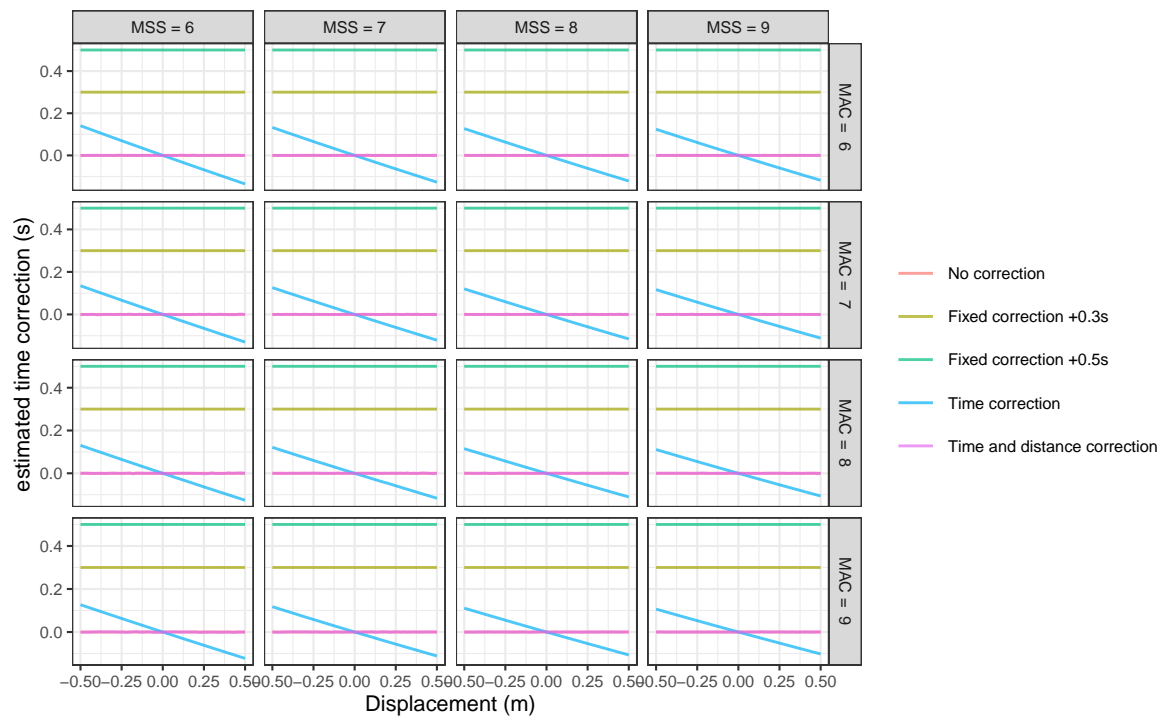
Figure below depicts effect of timing gate displacement on the MSS parameter. Time and distance correction model is the only model that estimated MAC correctly.

```
R> # MAC
R> ggplot(df, aes(x = displacement, y = est_MAC, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
+   xlab("Displacement (m)") +
+   ylab("estimated MAC (m/s/s)") +
+   theme(legend.title = element_blank())
```

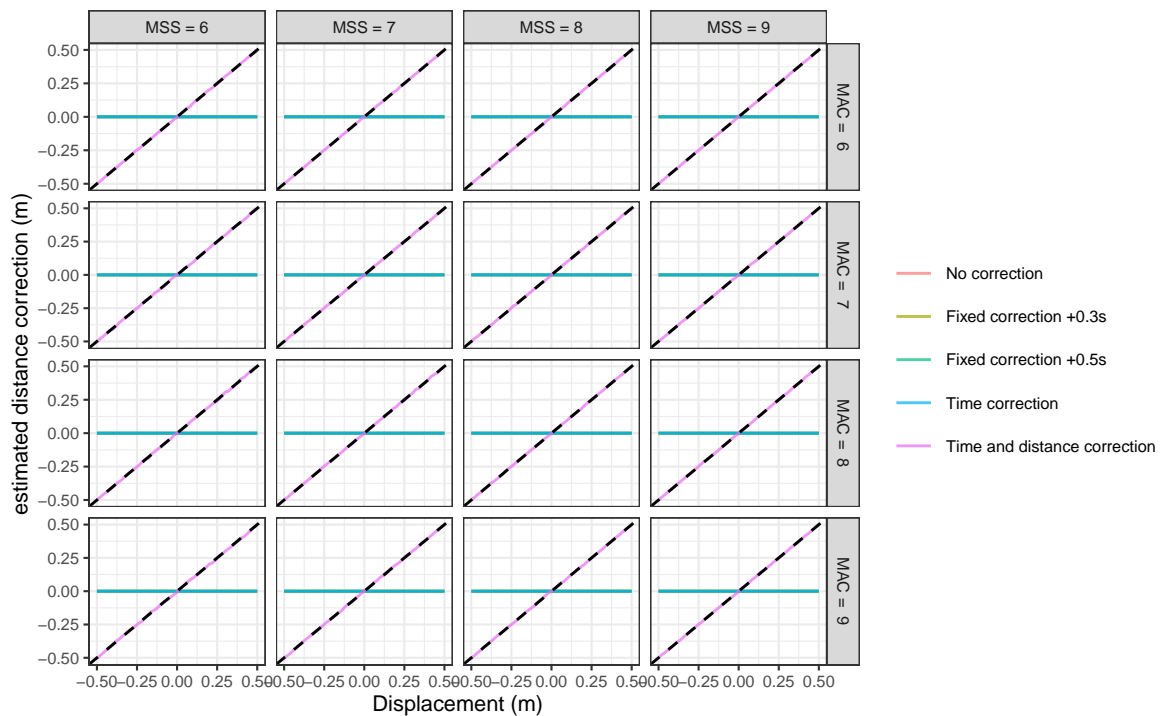


Figures below depicts estimated time and distance corrections parameters.

```
R> # time_correction
R> ggplot(df, aes(x = displacement, y = est_time_correction, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
+   xlab("Displacement (m)") +
+   ylab("estimated time correction (s)") +
+   theme(legend.title = element_blank())
```

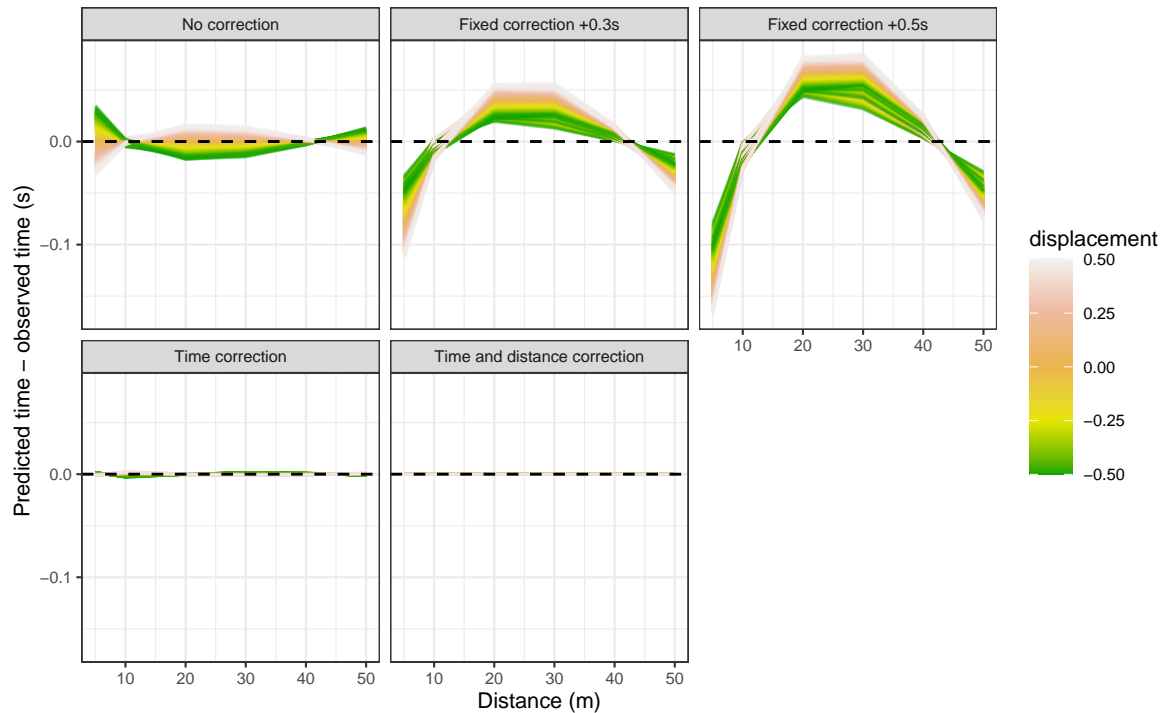


```
R> # distance_correction
R> ggplot(df, aes(x = displacement, y = est_distance_correction, color = model)) +
+   theme_bw(8) +
+   geom_line(alpha = 0.7) +
+   geom_abline(slope = 1, color = "black", linetype = "dashed") +
+   facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
+   xlab("Displacement (m)") +
+   ylab("estimated distance correction (m)") +
+   theme(legend.title = element_blank())
```



The following figure depicts residuals (i.e., predicted time minus observed time).

```
R> # Residuals
R> model_df <- model_df %>%
+   mutate(
+     MSS_string = paste("MSS =", MSS),
+     TAU_string = paste("TAU =", TAU),
+     MAC_string = paste("MAC = ", round(MAC, 2)),
+     PMAX_string = paste("PMAX = ", round(PMAX, 2)),
+     group = paste(MSS, MAC, displacement)
+   )
R>
R> ggplot(model_df, aes(y = residuals, x = distance, color = displacement, group = group))
+   theme_bw(8) +
+   geom_line(alpha = 0.3) +
+   facet_wrap(~model) +
+   geom_hline(yintercept = 0, linetype = "dashed") +
+   scale_color_gradientn(colours = terrain.colors(5, rev = FALSE)) +
+   xlab("Distance (m)") +
+   ylab("Predicted time - observed time (s)")
```



Additional sources of issues with the timing gates involve combination of the previous scenarios, as well as effect of number and distances of the timing gates (Haugen, Breitschädel, and Samozino 2020a; also see Haugen *et al.* 2012). For example, one might have a bad position of the initial gate, athlete might be moved back but also manage to trigger the gate before the start commence. More elaborate simulation is beyond the scope of the current paper.

5. Leave-one-out Cross-Validation

To estimate parameter stability, model over-fitting, and performance on the unseen data, **shorts** model function comes with implemented *leave-one-out cross validation* (LOOCV) (James, Witten, Hastie, and Tibshirani 2017; Jovanović 2020; Kuhn and Johnson 2018). LOOCV involves a simple, yet powerful procedure, of removing each observation, rebuilding the model, and making predictions for that removed observation. This process is repeated for each observations in the model dataset. LOOCV allows one to check estimated parameters stability, and model performance on the unseen data.

Let's perform LOOCV using Jack's data and time correction model:

```
R> jack_LOOCV <- shorts::model_using_splits_with_time_correction(
+   distance = split_times$distance,
+   time = split_times$jack_time,
+   LOOCV = TRUE
+ )
R>
R> jack_LOOCV
```

```

#> Estimated model parameters
#> -----
#>           MSS           TAU           MAC
#>           8.958           1.216           7.367
#>           PMAX    time_correction distance_correction
#>           16.499           0.284           0.000
#>
#> Model fit estimators
#> -----
#>      RSE R_squared    minErr    maxErr maxAbsErr    RMSE
#> 0.00181  1.00000 -0.00109  0.00189  0.00189  0.00114
#>      MAE      MAPE
#> 0.00104  0.04981
#>
#>
#> Leave-One-Out Cross-Validation
#> -----
#> Parameters:
#>      MSS  TAU  MAC PMAX time_correction distance_correction
#> 1 8.98 1.25 7.21 16.2          0.300          0
#> 2 8.97 1.22 7.35 16.5          0.284          0
#> 3 8.95 1.21 7.40 16.5          0.282          0
#> 4 8.96 1.21 7.39 16.5          0.282          0
#> 5 8.93 1.20 7.45 16.6          0.278          0
#>
#> Model fit:
#>      RSE R_squared    minErr    maxErr maxAbsErr    RMSE
#>      NA  1.00000 -0.00639  0.00510  0.00639  0.00401
#>      MAE      MAPE
#> 0.00349  0.18387

```

The model print output provides training dataset estimates and model performance, as well as LOOCV estimates and model performance.

Let's plot estimated parameters across LOOCV folds:

```

R> df <- jack_LOOCV$LOOCV$parameters
R>
R> df <- pivot_longer(df, cols = 1:6, names_to = "parameter")
R>
R> df$parameter <- factor(
+   df$parameter,
+   levels = c(
+     "MSS",
+     "TAU",
+     "MAC",
+     "PMAX",
+     "time_correction",

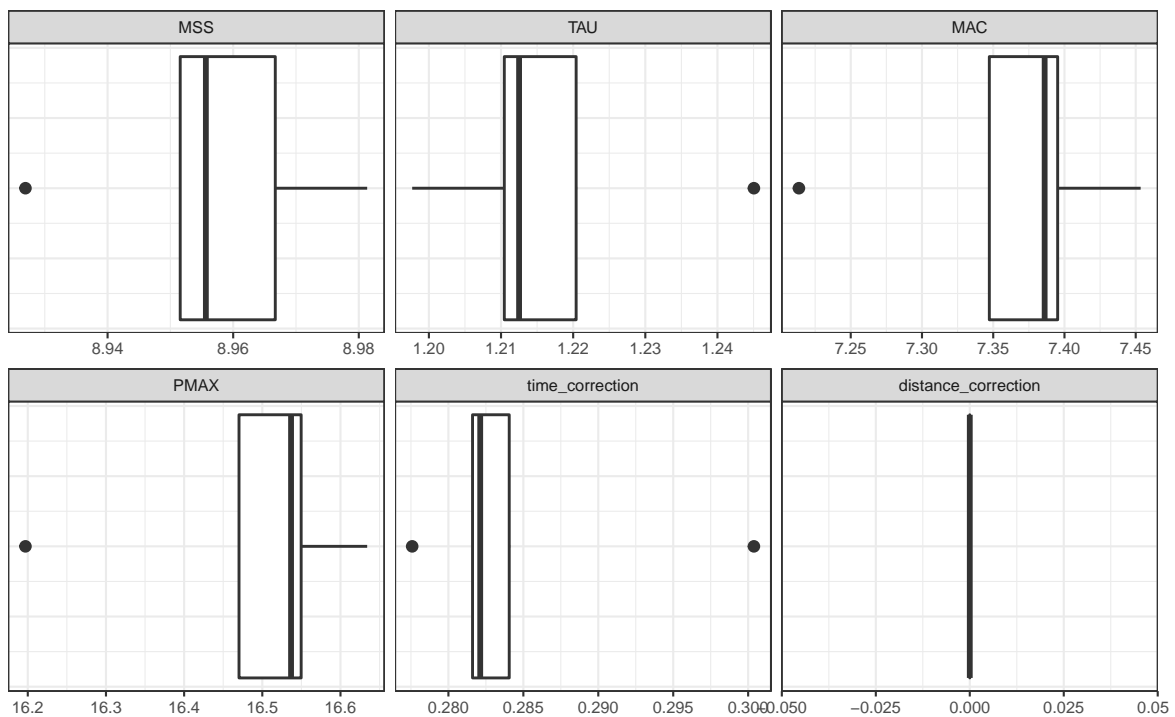
```



```

+   "distance_correction"
+ )
+ )
R>
R> ggplot(df, aes(x = value)) +
+   theme_bw(8) +
+   geom_boxplot() +
+   facet_wrap(~parameter, scales = "free_x") +
+   xlab(NULL) +
+   ylab(NULL) +
+   theme(
+     axis.ticks.y = element_blank(),
+     axis.text.y = element_blank()
+   )

```



Here is the plot of the training and LOOCV residuals:

```

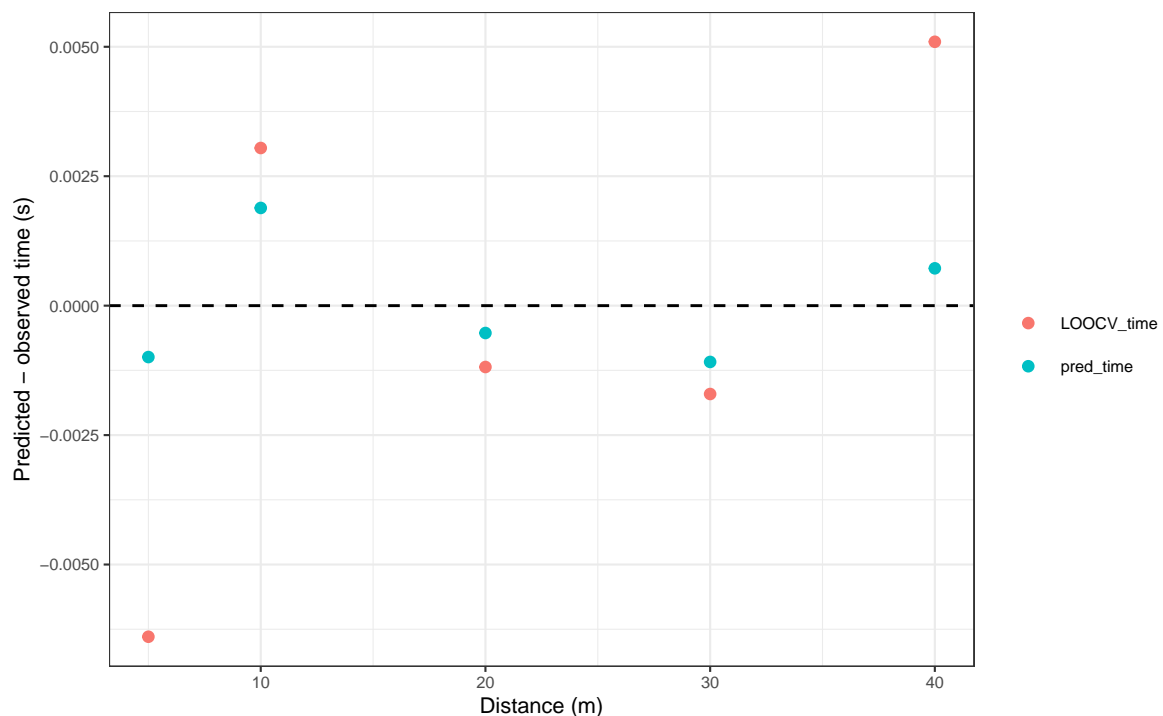
R> df <- data.frame(
+   distance = jack_LOOCV$data$distance,
+   time = jack_LOOCV$data$time,
+   pred_time = jack_LOOCV$data$pred_time,
+   LOOCV_time = jack_LOOCV$LOOCV$data$pred_time
+ )
R>
R> df <- df %>%
+   pivot_longer(cols = c("pred_time", "LOOCV_time"))

```

```

R>
R> df$resid <- df$value - df$time
R>
R> ggplot(df, aes(x = distance, y = resid, color = name)) +
+   theme_bw(8) +
+   geom_hline(yintercept = 0, linetype = "dashed") +
+   geom_point() +
+   theme(legend.title = element_blank()) +
+   xlab("Distance (m)") +
+   ylab("Predicted - observed time (s)")

```



As expected, model has more issues predicting unseen short or long sprints. Please note, that since LOOCV removes one observation, if model estimates three parameters, one needs to have at least five observations, since we need to make sure that model can be estimated once a single observation is removed. LOOCV can also be implemented with the mixed-effects models in the **shorts** package.

6. Example analysis

Let's utilize demonstrated functionalities of the **shorts** package using real-world data. The first dataset comes from Usain Bolt run from IAAF World Championship held in London, 2017, and the second dataset involve Jason Vescovi's sample data set for 52 female soccer and field hockey athletes which comes with the **shorts** package.

6.1. Usain Bolt's run from London 2017

The following dataset represents Usain Bolt's race in the finals at the IAAF World Championship held in London, 2017. Since reaction time enters the splits, we want to see how will that affect the model estimates, and particularly, will estimated time correction model pick reaction time up.

For the sake of this analysis, only 10m splits over 60m are used.

```
R> bolt_reaction_time <- 0.183
R>
R> bolt_distance <- c(10, 20, 30, 40, 50, 60)
R> bolt_time <- c(1.963, 2.983, 3.883, 4.763, 5.643, 6.493)
R>
R> # No corrections model
R> bolt_m1 <- shorts::model_using_splits(
+   distance = bolt_distance,
+   time = bolt_time
+ )
R>
R> # Model with reaction time as fixed time correction
R> bolt_m2 <- shorts::model_using_splits(
+   distance = bolt_distance,
+   time = bolt_time,
+   time_correction = -bolt_reaction_time
+ )
R>
R> # Model with estimated time correction
R> bolt_m3 <- shorts::model_using_splits_with_time_correction(
+   distance = bolt_distance,
+   time = bolt_time
+ )
R>
R> # Model with estimated time correction, but deducted reaction time
R> bolt_m4 <- shorts::model_using_splits_with_time_correction(
+   distance = bolt_distance,
+   time = bolt_time - bolt_reaction_time
+ )
R>
R> # Model with estimated time and distance corrections
R> bolt_m5 <- shorts::model_using_splits_with_corrections(
+   distance = bolt_distance,
+   time = bolt_time
+ )
R>
R> # Model with estimated time and distance corrections and deducted reaction time
R> bolt_m6 <- shorts::model_using_splits_with_corrections(
+   distance = bolt_distance,
+   time = bolt_time - bolt_reaction_time
+ )
```

```

R>
R> bolt_model <- rbind(
+   data.frame(
+     model = "No correction",
+     t(coef(bolt_m1))
+   ),
+   data.frame(
+     model = "No correction - RT",
+     t(coef(bolt_m2))
+   ),
+   data.frame(
+     model = "Time correction",
+     t(coef(bolt_m3))
+   ),
+   data.frame(
+     model = "Time correction - RT",
+     t(coef(bolt_m4))
+   ),
+   data.frame(
+     model = "Distance correction",
+     t(coef(bolt_m5))
+   ),
+   data.frame(
+     model = "Distance correction - RT",
+     t(coef(bolt_m6))
+   )
+ )
R>
R> bolt_model

#>               model  MSS   TAU   MAC PMAX
#> 1      No correction 12.1 1.564  7.77 23.6
#> 2  No correction - RT 11.7 1.205  9.74 28.6
#> 3      Time correction 11.7 1.202  9.76 28.6
#> 4  Time correction - RT 11.7 1.202  9.76 28.6
#> 5      Distance correction 11.6 0.855 13.56 39.3
#> 6 Distance correction - RT 11.6 0.855 13.56 39.3
#>   time_correction distance_correction
#> 1          0.00000             0.00
#> 2         -0.18300             0.00
#> 3         -0.18483             0.00
#> 4         -0.00183             0.00
#> 5         -0.81151            -3.98
#> 6         -0.62851            -3.98

```

6.2. Vescovi data

Vescovi's data set represents a sub-set of data from a total of 220 high-level female athletes (151 soccer players and 69 field hockey players). Using a random number generator, a total of 52 players (35 soccer and 17 field hockey) were selected for the sample dataset. Soccer players were older (24.6 ± 3.6 vs. 18.9 ± 2.7 yr, $p < 0.0001$), however there were no differences for height (167.3 ± 5.9 vs. 167.0 ± 5.7 cm, $p = 0.886$), body mass (62.5 ± 5.9 vs. 64.0 ± 9.4 kg, $p = 0.500$) or any sprint interval time ($p > 0.650$).

The protocol for assessing linear sprint speed has been described previously (Vescovi 2014, 2016, 2012) and was identical for each cohort. Briefly, all athletes performed a standardized warm-up that included general exercises such as jogging, shuffling, multi-directional movements, and dynamic stretching exercises. Infrared timing gates (Brower Timing, Utah) were positioned at the start line and at 5, 10, 20, 30, and 35 meters at a height of approximately 1.0 meter. Participants stood with their lead foot positioned approximately 5 cm behind the initial infrared beam (i.e., start line). Only forward movement was permitted (no leaning or rocking backwards) and timing started when the laser of the starting gate was triggered. The best 35 m time, and all associated split times were kept for analysis.

Below is the mixed-effects models analysis of the dataset.

```
R> data("vescovi")
R>
R> # Convert data to long
R> df <- vescovi %>%
+   select(1:13) %>%
+   # slice(1:10) %>%
+   pivot_longer(cols = 9:13, names_to = "distance", values_to = "time") %>%
+   mutate(
+     distance = as.numeric(str_extract(distance, "[0-9]+"))
+   )
R>
R> head(df)

#> # A tibble: 6 x 10
#>   Team Surface Athlete   Age Height Bodyweight   BMI   BSA
#>   <chr> <chr>   <fct>   <dbl> <dbl>      <dbl> <dbl> <dbl>
#> 1 W So~ Natura~ FMSS-1~    21  163.        54.8  20.5  1.6
#> 2 W So~ Natura~ FMSS-1~    21  163.        54.8  20.5  1.6
#> 3 W So~ Natura~ FMSS-1~    21  163.        54.8  20.5  1.6
#> 4 W So~ Natura~ FMSS-1~    21  163.        54.8  20.5  1.6
#> 5 W So~ Natura~ FMSS-1~    21  163.        54.8  20.5  1.6
#> 6 FH Sr Hard C~ FMSS-2~    21  172.        63.6  21.5  1.7
#> # ... with 2 more variables: distance <dbl>, time <dbl>
```

Here we will utilize the following models: no corrections model, fixed time correction model (using 0.3s heuristic rule of thumb), estimated time correction as a fixed effect model, estimated time correction as a random effect model, estimated distance correction as fixed effect model (and time correction as random effect), and estimated distance correction as random effect model.

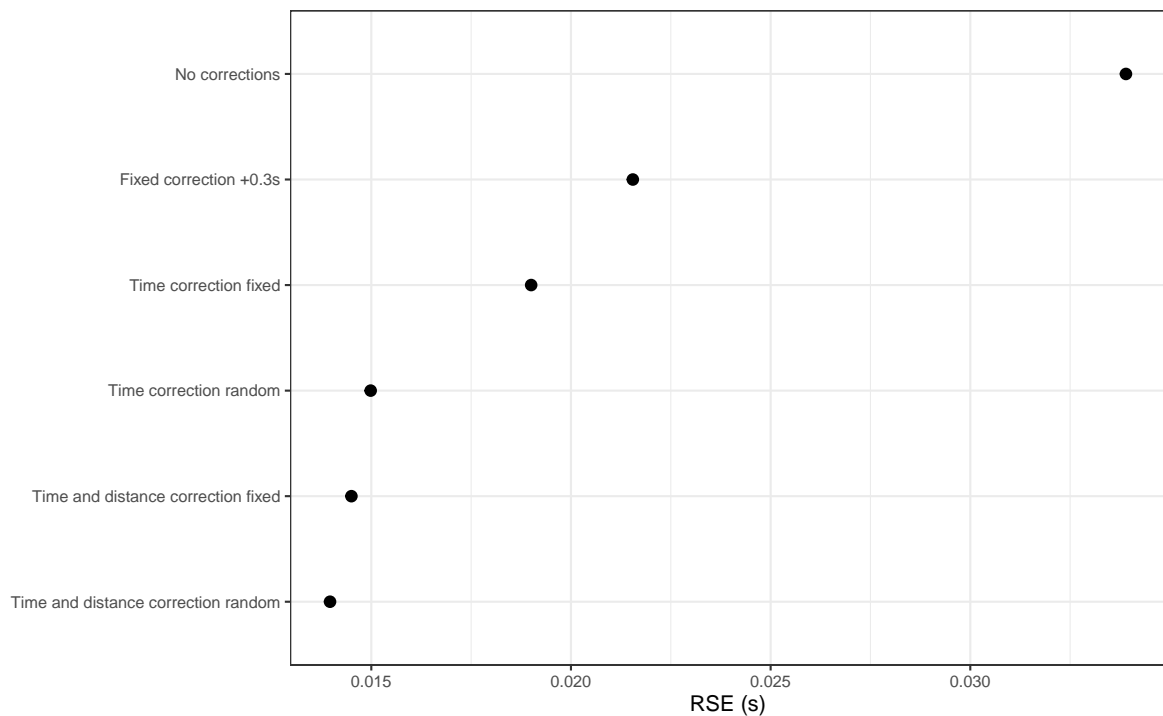
```

R> no_corrections <- shorts::mixed_model_using_splits(
+   df,
+   distance = "distance",
+   time = "time",
+   athlete = "Athlete"
+ )
R>
R> fixed_correction <- shorts::mixed_model_using_splits(
+   df,
+   distance = "distance",
+   time = "time",
+   athlete = "Athlete",
+   time_correction = 0.3
+ )
R>
R> time_correction_fixed <- shorts::mixed_model_using_splits_with_time_correction(
+   df,
+   distance = "distance",
+   time = "time",
+   athlete = "Athlete",
+   random = MSS + TAU ~ 1
+ )
R>
R> time_correction_random <- shorts::mixed_model_using_splits_with_time_correction(
+   df,
+   distance = "distance",
+   time = "time",
+   athlete = "Athlete",
+   random = MSS + TAU + time_correction ~ 1
+ )
R>
R> time_distance_correction_fixed <- shorts::mixed_model_using_splits_with_corrections(
+   df,
+   distance = "distance",
+   time = "time",
+   athlete = "Athlete",
+   random = MSS + TAU + time_correction ~ 1
+ )
R>
R> time_distance_correction_random <- shorts::mixed_model_using_splits_with_corrections(
+   df,
+   distance = "distance",
+   time = "time",
+   athlete = "Athlete",
+   random = MSS + TAU + time_correction + distance_correction ~ 1
+ )

```

The following image represents model fit estimator RSE for each model. As can be seen, RSE is reduced the more flexible the model.

```
R> model_fit <- rbind(
+   data.frame(
+     model = "No corrections",
+     t(unlist(no_corrections$model_fit))
+   ),
+   data.frame(
+     model = "Fixed correction +0.3s",
+     t(unlist(fixed_correction$model_fit))
+   ),
+   data.frame(
+     model = "Time correction fixed",
+     t(unlist(time_correction_fixed$model_fit))
+   ),
+   data.frame(
+     model = "Time correction random",
+     t(unlist(time_correction_random$model_fit))
+   ),
+   data.frame(
+     model = "Time and distance correction fixed",
+     t(unlist(time_distance_correction_fixed$model_fit))
+   ),
+   data.frame(
+     model = "Time and distance correction random",
+     t(unlist(time_distance_correction_random$model_fit))
+   )
+ )
R>
R> model_fit$model <- factor(
+   model_fit$model,
+   levels = rev(c(
+     "No corrections",
+     "Fixed correction +0.3s",
+     "Time correction fixed",
+     "Time correction random",
+     "Time and distance correction fixed",
+     "Time and distance correction random"
+   ))
+ )
R>
R> ggplot(model_fit, aes(x = RSE, y = model)) +
+   theme_bw(8) +
+   geom_point() +
+   xlab("RSE (s)") +
+   ylab(NULL)
```

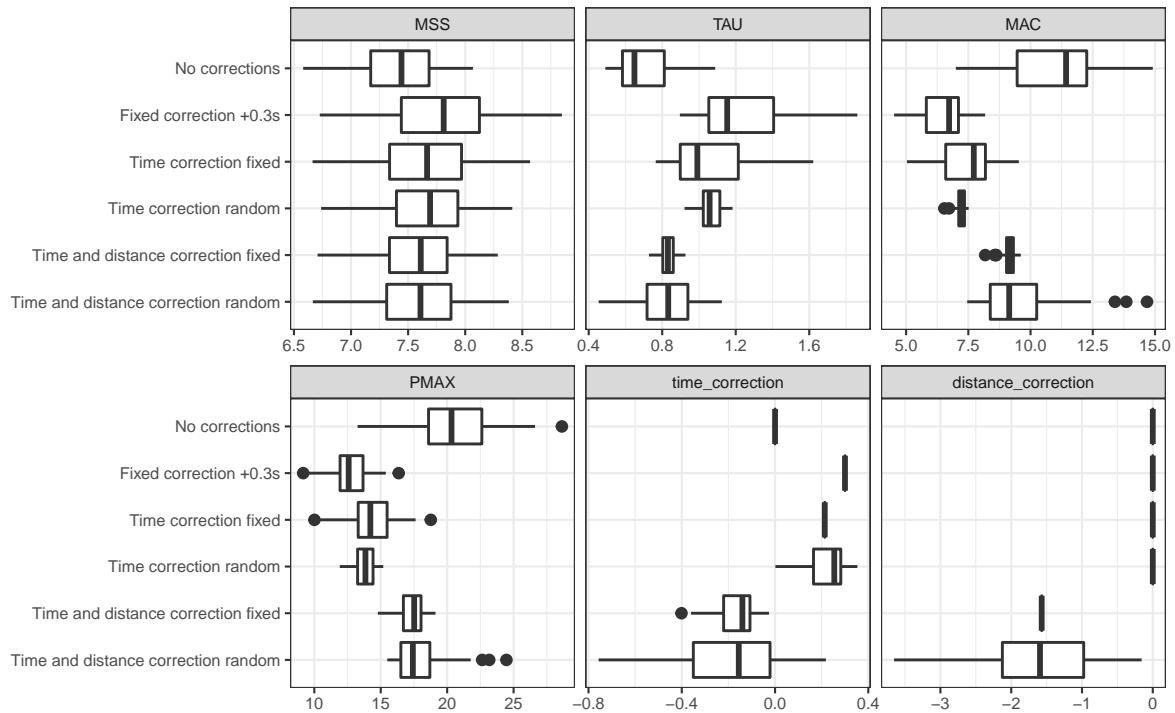


The following image depicts estimated parameters for each model:

```
R> est_params <- rbind(
+   data.frame(
+     model = "No corrections",
+     no_corrections$parameters$random
+   ),
+   data.frame(
+     model = "Fixed correction +0.3s",
+     fixed_correction$parameters$random
+   ),
+   data.frame(
+     model = "Time correction fixed",
+     time_correction_fixed$parameters$random
+   ),
+   data.frame(
+     model = "Time correction random",
+     time_correction_random$parameters$random
+   ),
+   data.frame(
+     model = "Time and distance correction fixed",
+     time_distance_correction_fixed$parameters$random
+   ),
+   data.frame(
+     model = "Time and distance correction random",
+     time_distance_correction_random$parameters$random
+   )
+ )
```



```
+ )
+ )
R>
R> est_params$model <- factor(
+   est_params$model,
+   levels = rev(c(
+     "No corrections",
+     "Fixed correction +0.3s",
+     "Time correction fixed",
+     "Time correction random",
+     "Time and distance correction fixed",
+     "Time and distance correction random"
+   ))
+ )
R>
R> est_params <- est_params %>%
+   pivot_longer(cols = -(1:2), names_to = "parameter")
R>
R> est_params$parameter <- factor(
+   est_params$parameter,
+   levels = c(
+     "MSS",
+     "TAU",
+     "MAC",
+     "PMAX",
+     "time_correction",
+     "distance_correction"
+   )
+ )
R>
R> ggplot(est_params, aes(y = model, x = value)) +
+   theme_bw(8) +
+   geom_boxplot() +
+   facet_wrap(~parameter, scales = "free_x") +
+   xlab(NULL) +
+   ylab(NULL)
```



The following image depicts model residuals across distance splits. To provide practical magnitude of the residuals, we have used between subject observed time SD multiplied with 0.2 and -0.2. This provides practical anchor for the residual magnitude, often referred to as *smallest worthwhile change* (SWC) or *smallest effect size of interest* (SESOI) (Jovanović 2020). If the residuals are within this magnitude band, they have no practical significance, and thus the model is good in making practically useful predictions.

Error bars represent residual bias and ± 1 SD.

```
R> model_resid <- rbind(
+   data.frame(
+     model = "No corrections",
+     no_corrections$data
+   ),
+   data.frame(
+     model = "Fixed correction +0.3s",
+     fixed_correction$data
+   ),
+   data.frame(
+     model = "Time correction fixed",
+     time_correction_fixed$data
+   ),
+   data.frame(
+     model = "Time correction random",
+     time_correction_random$data
+   ),
+ )
```

```

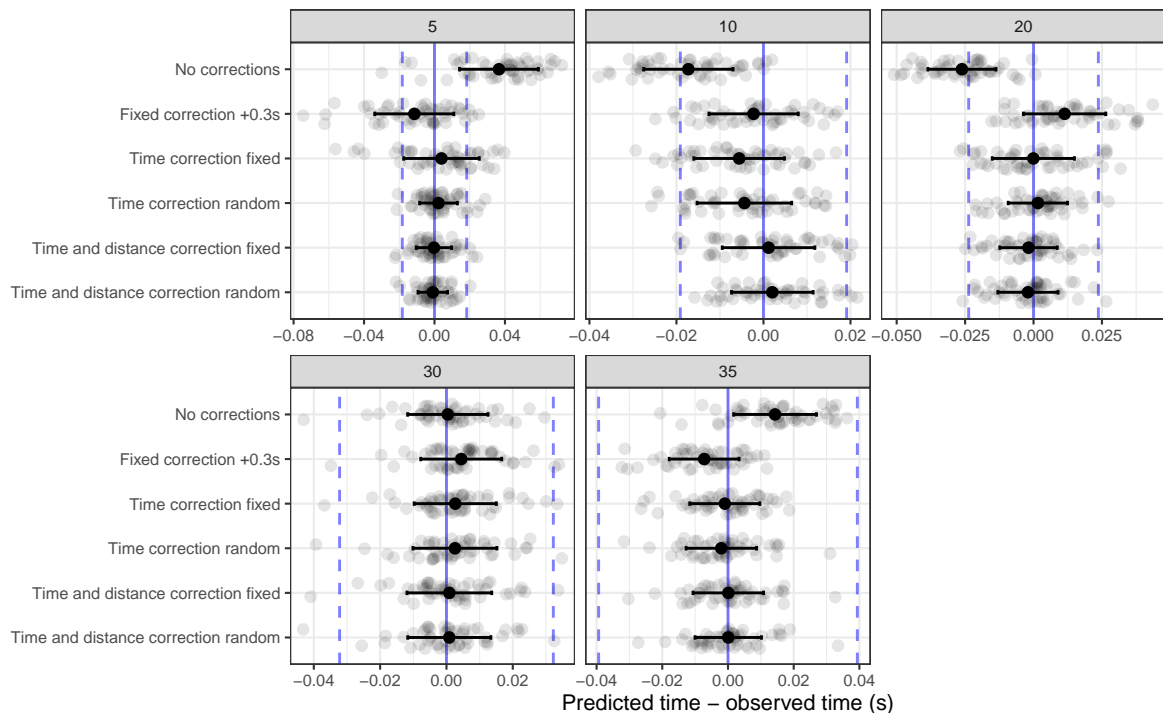
+   data.frame(
+     model = "Time and distance correction fixed",
+     time_distance_correction_fixed$data
+   ),
+   data.frame(
+     model = "Time and distance correction random",
+     time_distance_correction_random$data
+   )
+ )
R>
R>
R> model_resid$model <- factor(
+   model_resid$model,
+   levels = rev(c(
+     "No corrections",
+     "Fixed correction +0.3s",
+     "Time correction fixed",
+     "Time correction random",
+     "Time and distance correction fixed",
+     "Time and distance correction random"
+   ))
+ )
R>
R> model_resid$resid <- model_resid$pred_time - model_resid$time
R>
R> # Create SWC / SESOI band
R> model_SESOI <- model_resid %>%
+   group_by(model, distance) %>%
+   summarise(
+     bias = mean(resid),
+     variance = sd(resid),
+     upper = bias + variance,
+     lower = bias - variance,
+     MAD = mean(abs(resid)),
+     SESOI_upper = sd(time) * 0.2,
+     SESOI_lower = -sd(time) * 0.2
+   )
R>
R> # Plot
R> ggplot(model_resid, aes(y = model)) +
+   theme_bw(8) +
+   geom_vline(
+     data = model_SESOI,
+     aes(xintercept = SESOI_lower), color = "blue", alpha = 0.5, linetype = "dashed"
+   ) +
+   geom_vline(
+     data = model_SESOI,

```

```

+   aes(xintercept = SESOI_upper), color = "blue", alpha = 0.5, linetype = "dashed"
+ ) +
+   geom_vline(xintercept = 0, color = "blue", alpha = 0.5) +
+   geom_jitter(aes(x = resid), alpha = 0.1, height = 0.25) +
+   geom_errorbarh(data = model_SESOI, aes(xmin = lower, xmax = upper), height = 0.1, color = "black") +
+   geom_point(data = model_SESOI, aes(x = bias), color = "black") +
+   facet_wrap(~distance, scales = "free_x") +
+   xlab("Predicted time - observed time (s)") +
+   ylab(NULL)

```



The following figure depicts model residuals estimators (bias, or mean residual; variance, or SD of the residuals, and MAD, or mean absolute difference).

```

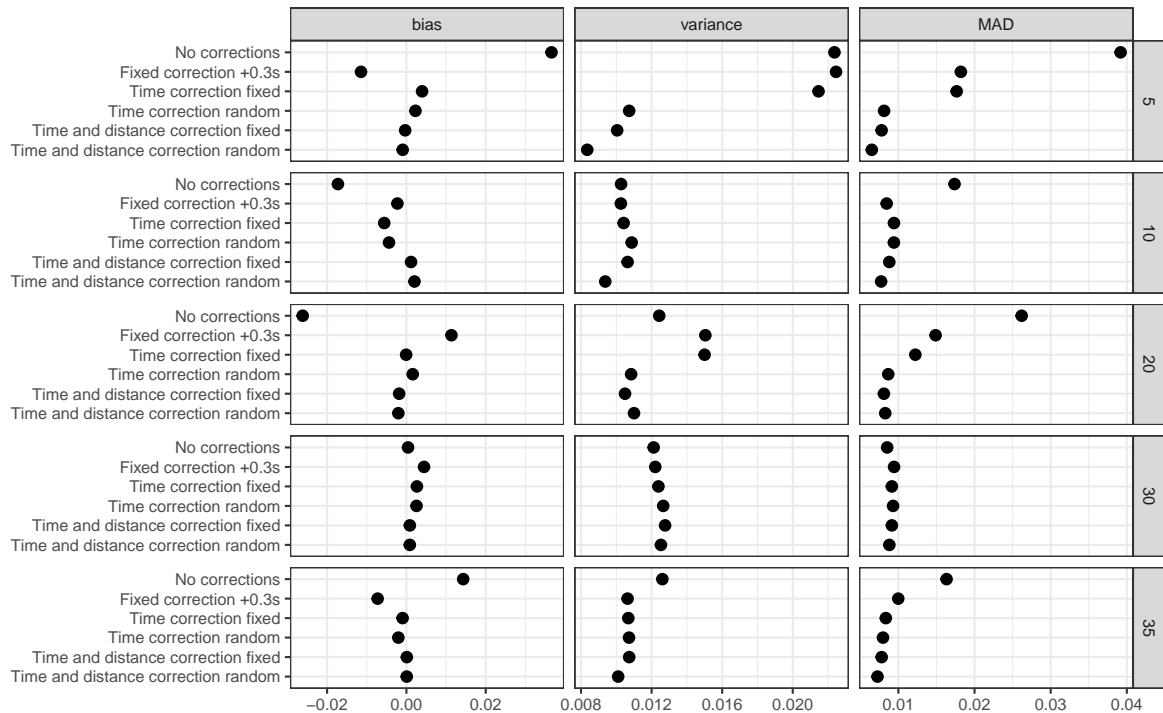
R> df <- model_SESOI %>%
+   pivot_longer(cols = -(1:2), names_to = "estimator") %>%
+   filter(estimator %in% c("bias", "variance", "MAD"))
R>
R> df$model <- factor(
+   df$model,
+   levels = rev(c(
+     "No corrections",
+     "Fixed correction +0.3s",
+     "Time correction fixed",
+     "Time correction random",
+     "Time and distance correction fixed",

```

```

+   "Time and distance correction random"
+ ))
+ )
R>
R> df$estimator <- factor(
+   df$estimator,
+   levels = c("bias", "variance", "MAD")
+ )
R>
R> ggplot(df, aes(x = value, y = model)) +
+   theme_bw(8) +
+   geom_point() +
+   facet_grid(distance ~ estimator, scales = "free_x") +
+   xlab(NULL) +
+   ylab(NULL)

```



Which model should be used? Although providing a better fit (using RSE as an estimator of model fit), time and distance correction models often estimate these parameters that are harder to interpret (e.g., negative distance correction). Although providing novel theoretical models in this paper, we acknowledge the need for validating them in practice, against gold-standard methods, assessing their agreement, as well as their power in detecting and adjusting for timing inconsistencies.

We are hoping that the **shorts** package will help fellow sports scientists and coaches in exploring short sprint profiles and help in driving research, particularly in devising measuring protocols that are sensitive enough to capture training intervention changes, but also robust enough to take into account potential sprint initiation and timing inconsistencies.

References

- Arsac LM, Locatelli E (2002). “Modeling the Energetics of 100-m Running by Using Speed Curves of World Champions.” *Journal of Applied Physiology*, **92**(5), 1781–1788. ISSN 8750-7587, 1522-1601. doi:[10.1152/japplphysiol.00754.2001](https://doi.org/10.1152/japplphysiol.00754.2001).
- Brown TD, Vescovi JD, Vanheest JL (2004). “Assessment of Linear Sprinting Performance: A Theoretical Paradigm.” *Journal of Sports Science & Medicine*, **3**(4), 203–210. ISSN 1303-2968.
- Clark KP, Rieger RH, Bruno RF, Stearne DJ (2017). “The NFL Combine 40-Yard Dash: How Important Is Maximum Velocity?” *Journal of Strength and Conditioning Research*, p. 1. ISSN 1064-8011. doi:[10.1519/JSC.0000000000002081](https://doi.org/10.1519/JSC.0000000000002081).
- Furusawa K, Hill AV, Parkinson JL (1927). “The Dynamics of ”Sprint” Running.” *Proceedings of the Royal Society of London. Series B, Containing Papers of a Biological Character*, **102**(713), 29–42. doi:[10.1098/rspb.1927.0035](https://doi.org/10.1098/rspb.1927.0035). <https://royalsocietypublishing.org/doi/pdf/10.1098/rspb.1927.0035>.
- Goerg GM (2020). *LambertW: Probabilistic Models to Analyze and Gaussianize Heavy-Tailed, Skewed Data*. R package version 0.6.5, URL <https://CRAN.R-project.org/package=LambertW>.
- Haugen TA, Breitschädel F, Samozino P (2020a). “Power-Force-Velocity Profiling of Sprinting Athletes: Methodological and Practical Considerations When Using Timing Gates.” *Journal of Strength and Conditioning Research*, **34**(6), 1769–1773. ISSN 1064-8011. doi:[10.1519/JSC.0000000000002890](https://doi.org/10.1519/JSC.0000000000002890).
- Haugen TA, Breitschädel F, Seiler S (2019). “Sprint Mechanical Variables in Elite Athletes: Are Force-Velocity Profiles Sport Specific or Individual?” *PLOS ONE*, **14**(7), e0215551. ISSN 1932-6203. doi:[10.1371/journal.pone.0215551](https://doi.org/10.1371/journal.pone.0215551).
- Haugen TA, Breitschädel F, Seiler S (2020b). “Sprint Mechanical Properties in Soccer Players According to Playing Standard, Position, Age and Sex.” *Journal of Sports Sciences*, **38**(9), 1070–1076. ISSN 0264-0414, 1466-447X. doi:[10.1080/02640414.2020.1741955](https://doi.org/10.1080/02640414.2020.1741955).
- Haugen TA, Tønnessen E, Seiler SK (2012). “The Difference Is in the Start: Impact of Timing and Start Procedure on Sprint Running Performance.” *Journal of Strength and Conditioning Research*, **26**(2), 473–479. ISSN 1064-8011. doi:[10.1519/JSC.0b013e318226030b](https://doi.org/10.1519/JSC.0b013e318226030b).
- James G, Witten D, Hastie T, Tibshirani R (2017). *An Introduction to Statistical Learning: With Applications in R*. 1st ed. 2013, corr. 7th printing 2017 edition edition. Springer, New York. ISBN 978-1-4614-7137-0.
- Jovanović M (2020). *Bmbstats: Bootstrap Magnitude-Based Statistics for Sports Scientists*. Mladen Jovanović. ISBN 978-86-900803-5-9.
- Jovanovic M (2020). *shorts: Short Sprints*. R package version 1.1.2, URL <https://mladenjovanovic.github.io/shorts/>.

- Kuhn M, Johnson K (2018). *Applied Predictive Modeling*. 1st ed. 2013, corr. 2nd printing 2016 edition edition. Springer, New York. ISBN 978-1-4614-6848-6.
- Mangine GT, Hoffman JR, Gonzalez AM, Wells AJ, Townsend JR, Jajtner AR, McCormack WP, Robinson EH, Fragala MS, Fukuda DH, Stout JR (2014). “Speed, Force, and Power Values Produced From Nonmotorized Treadmill Test Are Related to Sprinting Performance:.” *Journal of Strength and Conditioning Research*, **28**(7), 1812–1819. ISSN 1064-8011. doi:10.1519/JSC.0000000000000316.
- Pinheiro J, Bates D, R-core (2020). *nlme: Linear and Nonlinear Mixed Effects Models*. R package version 3.1-149, URL <https://svn.r-project.org/R-packages/trunk/nlme/>.
- R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Samozino P, Rabita G, Dorel S, Slawinski J, Peyrot N, Saez de Villarreal E, Morin JB (2016). “A Simple Method for Measuring Power, Force, Velocity Properties, and Mechanical Effectiveness in Sprint Running: Simple Method to Compute Sprint Mechanics.” *Scandinavian Journal of Medicine & Science in Sports*, **26**(6), 648–658. ISSN 09057188. doi:10.1111/sms.12490.
- van Ingen Schenau GJ, Jacobs R, de Koning JJ (1991). “Can Cycle Power Predict Sprint Running Performance?” *European Journal of Applied Physiology and Occupational Physiology*, **63**(3-4), 255–260. ISSN 0301-5548, 1439-6327. doi:10.1007/BF00233857.
- Vescovi JD (2012). “Sprint Speed Characteristics of High-Level American Female Soccer Players: Female Athletes in Motion (FAiM) Study.” *Journal of Science and Medicine in Sport*, **15**(5), 474–478. ISSN 14402440. doi:10.1016/j.jsams.2012.03.006.
- Vescovi JD (2014). “Impact of Maximum Speed on Sprint Performance During High-Level Youth Female Field Hockey Matches: Female Athletes in Motion (FAiM) Study.” *International Journal of Sports Physiology and Performance*, **9**(4), 621–626. ISSN 1555-0265, 1555-0273. doi:10.1123/ijssp.2013-0263.
- Vescovi JD (2016). “Locomotor, Heart-Rate, and Metabolic Power Characteristics of Youth Women’s Field Hockey: Female Athletes in Motion (FAiM) Study.” *Research Quarterly for Exercise and Sport*, **87**(1), 68–77. ISSN 0270-1367, 2168-3824. doi:10.1080/02701367.2015.1124972.
- Ward-Smith A (2001). “Energy Conversion Strategies during 100 m Sprinting.” *Journal of Sports Sciences*, **19**(9), 701–710. ISSN 0264-0414, 1466-447X. doi:10.1080/02640410152475838.
- Wickham H (2019). *tidyverse: Easily Install and Load the Tidyverse*. R package version 1.3.0, URL <https://CRAN.R-project.org/package=tidyverse>.
- Wickham H (2020). *tidyr: Tidy Messy Data*. R package version 1.1.2, URL <https://CRAN.R-project.org/package=tidyr>.

Wickham H, Chang W, Henry L, Pedersen TL, Takahashi K, Wilke C, Woo K, Yutani H, Dunnington D (2020a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.3.2, URL <https://CRAN.R-project.org/package=ggplot2>.

Wickham H, François R, Henry L, Müller K (2020b). *dplyr: A Grammar of Data Manipulation*. R package version 1.0.2, URL <https://CRAN.R-project.org/package=dplyr>.

Affiliation:

Mladen Jovanović
University of Belgrade
Faculty of Sport and Physical Education
Belgrade, Serbia
E-mail: coach.mladen.jovanovic@gmail.com

Jason D. Vescovi
University of Toronto
Faculty of Kinesiology and Physical Education
Graduate School of Exercise Science
Toronto, ON Canada
E-mail: vescovij@gmail.com