Contents

1 Introduction

~~Short~~ sprint speed is one of the most distinguishable and admired physical traits in sports, ~~and one that have puzzled coaches and sports scientists from the time when two humanoids raced against each other (if they were available back then)~~. Short sprints are defined as maximal running from a stand still position over a distance that doesn't result in deceleration at the end. ~~This is~~ approximately less than 6 seconds in duration (why? ATP/CP? — REF), but the distance can vary based on the level of the athlete (i.e. lower level athletes can reach this point earlier, around 30-40m, while elite sprinters much later, around 50-60m — REF).

Short sprints have been modeled using the mono-exponential equation (1) originally proposed by Furusawa, Hill, and Parkinson (1927), and more recently popularized by Greene (1986), Clark et al. (2017), Chelly and Denis (2001), Morin et al. (2019), Morin and Samozino (2016), ~~Thomas A.~~ Haugen, Breitschädel, and Samozino (2020), Haugen, Breitschädel, and Seiler (2019), ~~Thomas A.~~ Haugen, Breitschädel, and Seiler (2020), Furusawa, Hill, and Parkinson (1927), Haugen, Tønnessen, and Seiler (2012), and Samozino et al. (2016):

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

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).

$$MAC = \frac{MSS}{TAU} \tag{2}$$

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.

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

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

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

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

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

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).

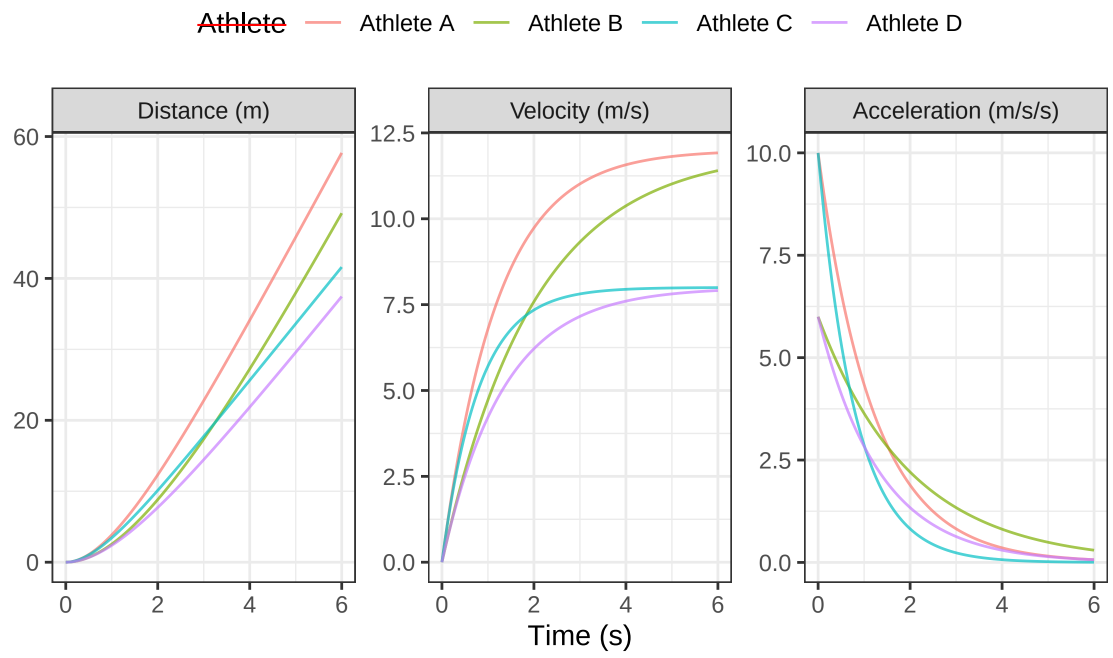


Figure 1: Kinematic characteristic of four athletes with different MSS and MAC parameters over a period of 0 to 6seconds.

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

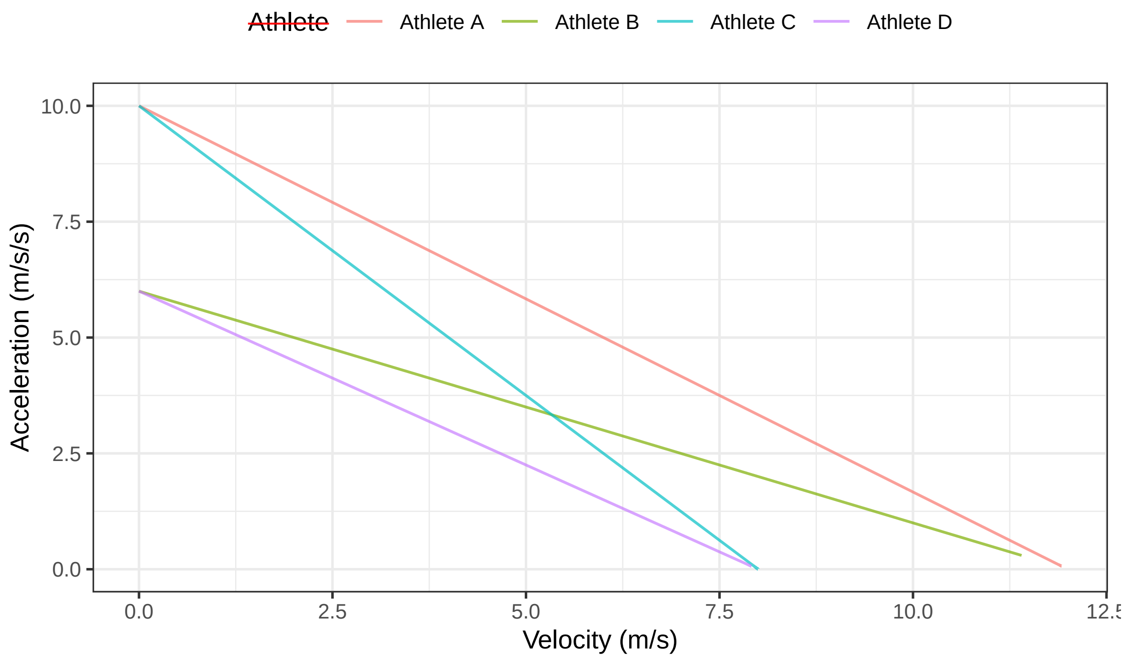


Figure 2: Acceleration-Velocity profile of four athletes with different MSS and MAC parameters.

2 Estimation using `shorts` package

Short sprints profiling is usually performed by: (1) measuring split times using timing gates (i.e., positioned at 0m, 10m, 20m, 30m, 40m), and (2) getting a velocity trace using radar gun ~~(usually at around 100Hz)~~. Estimation of ~~the~~ MSS and TAU parameters from ~~the~~ 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.

2.1 Estimating short sprint parameters using split times

Let's consider an example of an athlete with MSS equal to $9ms^-1$, TAU equal to 1.3, and MAC equal to $6.92m/s^2$ 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:

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

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

```r
require(shorts)

split_distance <- c(10, 20, 30, 40)

# Generate times by using:
# round(predict_time_at_distance(distance = split_distance, MSS = 9, TAU = 1.3), 2)
```

```
split_time <- c(2.17, 3.43, 4.60, 5.73)

m1 <- shorts::model_using_splits(
  distance = split_distance,
  time = split_time
)

m1
#> Estimated model parameters
#> --------------------------
#>                  MSS                 TAU                 MAC                PMAX
#>             9.012088            1.308294            6.888426           15.519775
#>      time_correction distance_correction
#>             0.000000            0.000000
#>
#> Model fit estimators
#> --------------------
#>          RSE    R_squared       minErr       maxErr    maxAbsErr         RMSE
#>  0.002492369  0.999998245 -0.001775176  0.002645299  0.002645299  0.001762371
#>          MAE         MAPE
#>  0.001567075  0.047417028
```

Maximal relative power (PMAX) from the output is estimated using $\frac{MSS \times MAC}{4}$, which disregards the air resistance (which we will deal with soon). `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` functions:

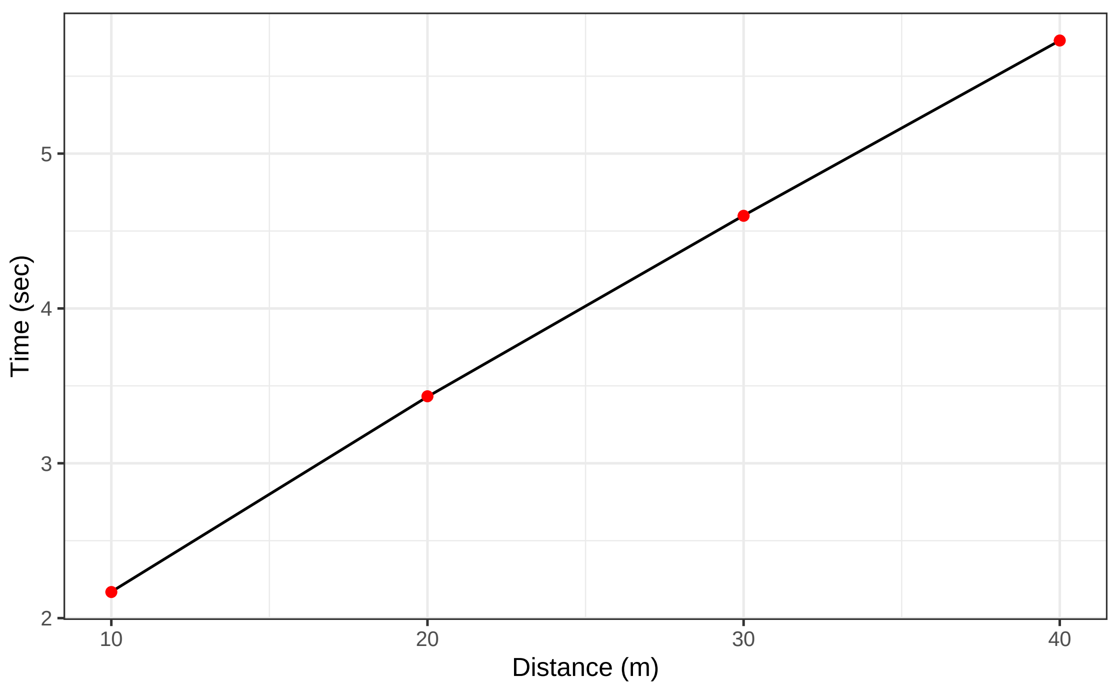
```
summary(m1$model)
#>
#> Formula: corrected_time ~ TAU * I(LambertW::W(-exp(1)^(-distance/(MSS *
#>     TAU) - 1))) + distance/MSS + TAU
#>
#> Parameters:
#>     Estimate Std. Error t value Pr(>|t|)
#> MSS 9.012088   0.015507   581.2 2.96e-06 ***
#> TAU 1.308294   0.006599   198.3 2.54e-05 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.002492 on 2 degrees of freedom
#>
#> Number of iterations to convergence: 4
#> Achieved convergence tolerance: 3.112e-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):

```
# Predict time at distance
shorts::predict_time_at_distance(
  distance = split_distance,
  MSS = m1$parameters$MSS,
  TAU = m1$parameters$TAU
)
#> [1] 2.168543 3.432645 4.598225 5.730391

# Predict acceleration at time
shorts::predict_acceleration_at_time(
  time = c(0, 1, 2, 3, 4, 5, 6),
  MSS = m1$parameters$MSS,
  TAU = m1$parameters$TAU
)
#> [1] 6.88842562 3.20748963 1.49351830 0.69543387 0.32381810 0.15078093 0.07020882
```

Here is a plot of observed and model predicted split times, with the help of the `ggplot2` package (Wickham, Chang, et al. 2020):

```
require(ggplot2)

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
  )
)

ggplot(df, aes(x = distance)) +
  theme_bw(10) +
  geom_line(aes(y = time)) +
  geom_point(aes(y = pred_time), color = "red") +
  xlab("Distance (m)") +
  ylab("Time (sec)")
```



2.1.1 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), and later power production (using `shorts::predict_power_at_distance` and `shorts::predict_power_at_time` functions), ~~one needs to take into account the~~ air resistance. Air resistance (in Newtons) is estimated using `shorts::get_air_resistance` function, which takes velocity, body mass (~~in~~ kg), body height (~~in meters~~), barometric pressure (~~in~~ Torrs), air temperature (~~in~~ Celzius), and wind velocity (~~in~~ 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] 6.104239
```

When estimating force and power, ~~one can set~~ the air resistance parameters using ...:

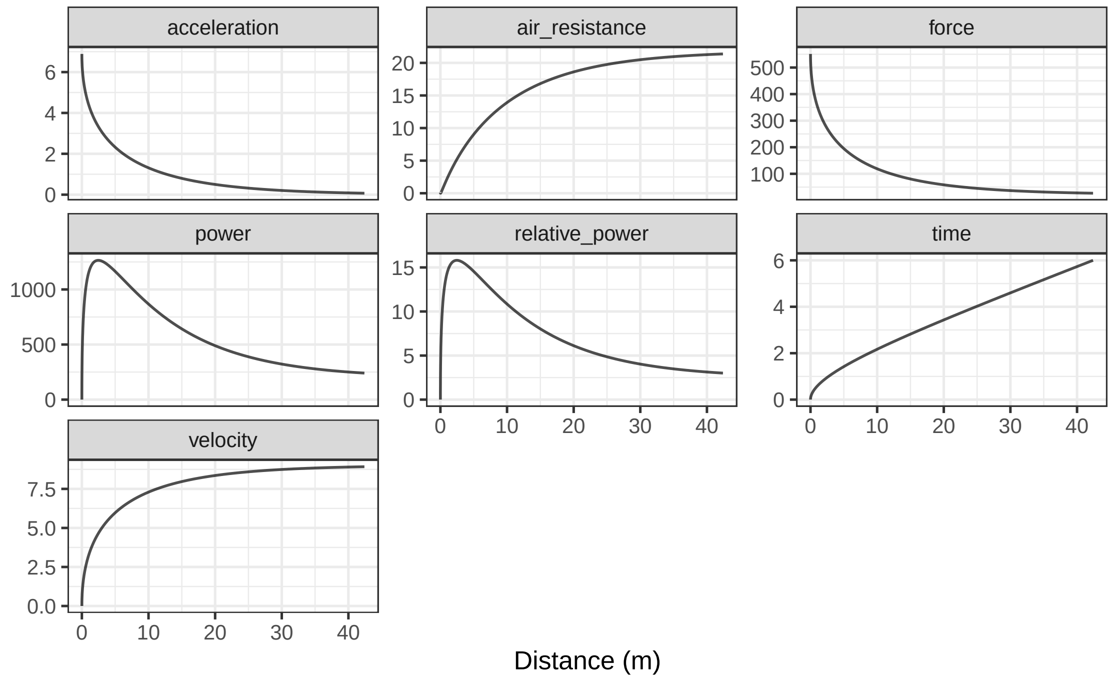
```r
# To calculate horizontal force produced
shorts::predict_force_at_distance(
  distance = split_distance,
  MSS = m1$parameters$MSS,
  TAU = m1$parameters$TAU,
  # Additional parameters forwarded to shorts::get_air_resistance
  # Otherwise, defaults are used
  bodymass = 80,
  bodyheight = 1.85,
  barometric_pressure = 780,
  air_temperature = 20,
  wind_velocity = 0.5
)
#> [1] 118.95474  58.58506  36.88538  28.16798

# To calculate power produced
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] 867.6927 489.6799 322.5224 250.6728
```

The easiest way to get all kinematics for ~~0-6sec~~ short sprints is to use `shorts::predict_kinematics` functions:

```
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
)

head(df)
#>   time     distance   velocity acceleration air_resistance    force     power
#> 1 0.00 0.0000000000 0.00000000     6.888426    0.075360973 551.1494   0.00000
#> 2 0.01 0.0003435454 0.06862167     6.835974    0.056094870 546.9340  37.53153
#> 3 0.02 0.0013706916 0.13672082     6.783923    0.039782082 542.7536  74.20571
#> 4 0.03 0.0030762334 0.20430144     6.732267    0.026357542 538.6077 110.03833
#> 5 0.04 0.0054550051 0.27136747     6.681005    0.015757326 534.4961 145.04487
#> 6 0.05 0.0085018805 0.33792284     6.630133    0.007918632 530.4186 179.24054
#>   relative_power
#> 1      0.0000000
#> 2      0.4691441
#> 3      0.9275714
#> 4      1.3754791
#> 5      1.8130608
#> 6      2.2405068
```

Plotting the model predictions can be done once we convert data from wide to long with the help of `dplyr`(Wickham, François, et al. 2020), `tidyr` (Wickham 2020), and `tidyverse` (Wickham 2019) packages:

```r
require(tidyverse)

df <- pivot_longer(data = df, cols = -2)

ggplot(df, aes(x = distance, y = value)) +
  theme_bw(10) +
  facet_wrap(~name, scales = "free_y") +
  geom_line(alpha = 0.7) +
  ylab(NULL) +
  xlab("Distance (m)")
```



2.1.2 Utility functions

Sports scientists and coaches might be interested in finding distances and times where 90% of maximum sprinting speed is reached, or where peak power is within 90% range. To ~~help finding~~ these ~~threshold~~ `shorts` package comes with `shorts::find_` family of functions:

```r
# Finds distance where 90% of maximum sprinting speed is reached
shorts::find_velocity_critical_distance(
  MSS = m1$parameters$MSS,
  TAU = m1$parameters$TAU,
  percent = 0.9
)
#> [1] 16.53714

# Finds maximal power and distance (this time using air resistance)
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] 1264.459
#>
#> $distance
#> [1] 2.462303


# Finds distance over 90% power range
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] 0.9585353
#>
#> $upper
#> [1] 5.436432
```

2.1.3 Mixed-effects model

~~Short sprints~~ are often times performed with a group of athlete (e.g., soccer club) representing a single strata of interest. Sports scientists can estimate individual profiles ~~(i.e., for each individual)~~, 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 ~~will~~ load the `split_times` dataset provided in the `shorts` package:

```
data(split_times)

head(split_times)
#> # A tibble: 6 x 4
#>   athlete bodyweight distance     time
#>   <chr>        <dbl>    <dbl> <I<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

# Mixed model
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
)

m2
#> Estimated fixed model parameters
#> --------------------------------
#>                  MSS                 TAU                 MAC                PMAX
#>            8.0649112           0.6551988          12.3091052          24.8179600
#>      time_correction distance_correction
#>            0.0000000           0.0000000
#> 
#> Estimated random model parameters
#> ----------------------------------
#>     athlete      MSS       TAU      MAC     PMAX time_correction
#> 1     James 9.691736 0.8469741 11.44278 27.72510               0
#> 2       Jim 7.833622 0.5048535 15.51663 30.38785               0
#> 3      John 7.780395 0.7274302 10.69573 20.80424               0
#> 4 Kimberley 8.569518 0.8022235 10.68221 22.88535               0
#> 5  Samantha 6.449284 0.3945129 16.34746 26.35735               0
#>   distance_correction
#> 1                   0
#> 2                   0
#> 3                   0
#> 4                   0
#> 5                   0
#> 
#> Model fit estimators
#> --------------------
#>         RSE   R_squared      minErr      maxErr   maxAbsErr        RMSE
#>  0.02600213  0.99982036 -0.02934519  0.04964582  0.04964582  0.02139178
#>         MAE        MAPE
#>  0.01722581  0.90185579
```

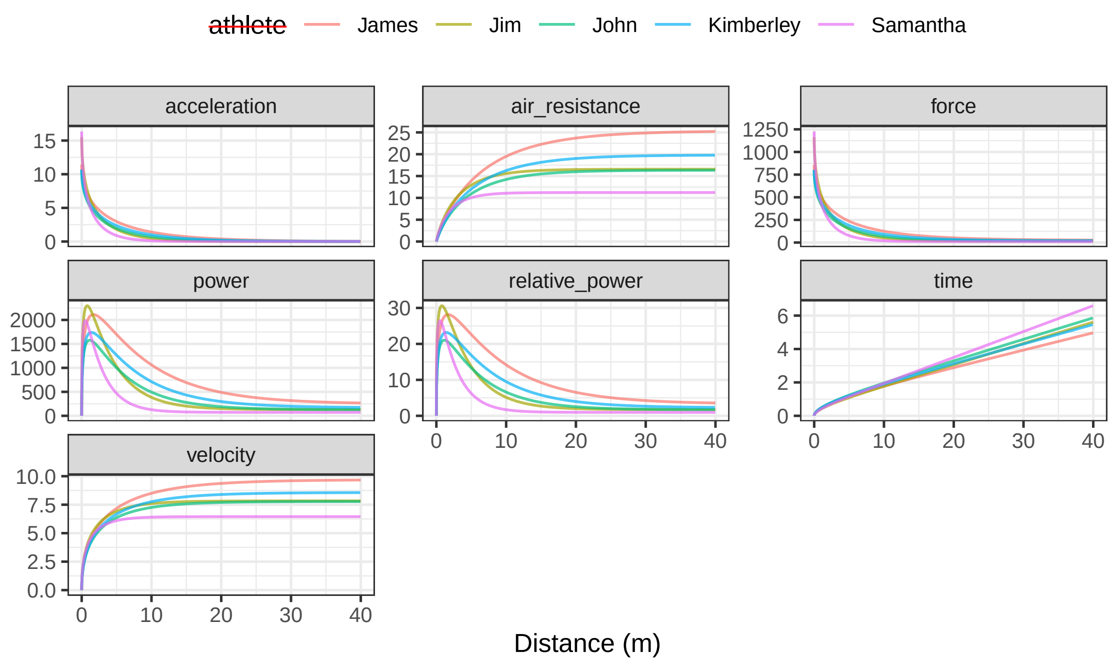
Additional information about mixed-effects model performed using the `nlme` package (Pinheiro, Bates, and R-core 2020) can be obtained using `summary`:

```
summary(m2)
#> Nonlinear mixed-effects model fit by maximum likelihood
#>   Model: corrected_time ~ TAU * I(LambertW::W(-exp(1)^(-distance/(MSS *      TAU) - 1))) + distance/
#>  Data: train
#>         AIC       BIC  logLik
#>   -75.06719 -66.66001 43.5336
```

```
#>
#> Random effects:
#>  Formula: list(MSS ~ 1, TAU ~ 1)
#>  Level: athlete
#>  Structure: General positive-definite, Log-Cholesky parametrization
#>          StdDev     Corr
#> MSS      1.06581655 MSS
#> TAU      0.17821114 0.877
#> Residual 0.02600213
#>
#> Fixed effects: MSS + TAU ~ 1
#>        Value Std.Error DF   t-value p-value
#> MSS 8.064911 0.4949104 24 16.295699       0
#> TAU 0.655199 0.0837593 24  7.822404       0
#>  Correlation:
#>     MSS
#> TAU 0.874
#>
#> Standardized Within-Group Residuals:
#>        Min         Q1        Med         Q3        Max
#> -1.9092981 -0.6050683  0.1536529  0.5226467  1.1285687
#>
#> Number of Observations: 30
#> Number of Groups: 5
```

~~To~~ 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)

df <- pivot_longer(df, cols = c(-1, -3))

ggplot(
  filter(df, distance < 40),
  aes(x = distance, y = value, group = athlete, color = athlete)
) +
  theme_bw(10) +
  facet_wrap(~name, scales = "free_y") +
  geom_line(alpha = 0.7) +
  ylab(NULL) +
  xlab("Distance (m)") +
  theme(legend.position = "top")
```



2.2 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 ~~the~~ 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 6.92m/s^2 performing 40m sprint with velocity estimated using radar run (in this case with 1Hz sampling rate).

```
sprint_time <- seq(0, 6, 1)

# Generate velocity by using:
# round(predict_velocity_at_time(time = sprint_time, MSS = 9, TAU = 1.3), 2)

sprint_velocity <- c(0.00, 4.83, 7.07, 8.10, 8.59, 8.81, 8.91)

m3 <- shorts::model_using_radar(
  velocity = sprint_velocity,
  time = sprint_time
)

m3
#> Estimated model parameters
#> --------------------------
#>                  MSS                 TAU                 MAC                PMAX
#>             9.000997            1.300100            6.923312           15.579177
#>      time_correction distance_correction
#>             0.000000            0.000000
#>
#> Model fit estimators
#> --------------------
#>         RSE    R_squared       minErr       maxErr    maxAbsErr         RMSE
#>  0.003270473  0.999999156 -0.004055148  0.005323673  0.005323673  0.002764054
#>         MAE         MAPE
#>  0.002066849          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:

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

m3_weighted
#> Estimated model parameters
#> --------------------------
#>                 MSS                 TAU                 MAC                PMAX
#>            9.000951            1.300063            6.923473           15.579460
#>     time_correction distance_correction
#>            0.000000            0.000000
#>
#> Model fit estimators
#> --------------------
#>         RSE    R_squared       minErr       maxErr    maxAbsErr         RMSE
#>  0.001075694  0.999999156 -0.004062660  0.005341016  0.005341016  0.002764282
#>         MAE         MAPE
#>  0.002061301          NaN
```

2.2.1 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.

```
data("radar_gun_data")

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

m4 <- shorts::mixed_model_using_radar(
  radar_gun_data,
  time = "time",
  velocity = "velocity",
  athlete = "athlete"
)

m4
#> Estimated fixed model parameters
#> --------------------------------
#>                 MSS                 TAU                 MAC                PMAX 
#>            8.301178            1.007782            8.237080           17.094367 
#>     time_correction distance_correction 
#>            0.000000            0.000000 
#> 
#> Estimated random model parameters
#> ----------------------------------
#>     athlete      MSS       TAU      MAC     PMAX time_correction
#> 1     James 9.998556 1.1108457 9.000851 22.49888               0
#> 2       Jim 7.997945 0.8886712 8.999892 17.99516               0
#> 3      John 8.000051 1.0690357 7.483427 14.96695               0
#> 4 Kimberley 9.005500 1.2855706 7.005061 15.77102               0
#> 5  Samantha 6.503839 0.6847851 9.497635 15.44277               0
#>   distance_correction
#> 1                   0
#> 2                   0
#> 3                   0
#> 4                   0
#> 5                   0
#> 
#> Model fit estimators
#> --------------------
#>         RSE   R_squared      minErr      maxErr   maxAbsErr        RMSE 
#>  0.05164818  0.99942171 -0.21912952  0.19832897  0.21912952  0.05156203 
#>         MAE        MAPE 
#>  0.03949473         NaN
```

3 Problems with estimation

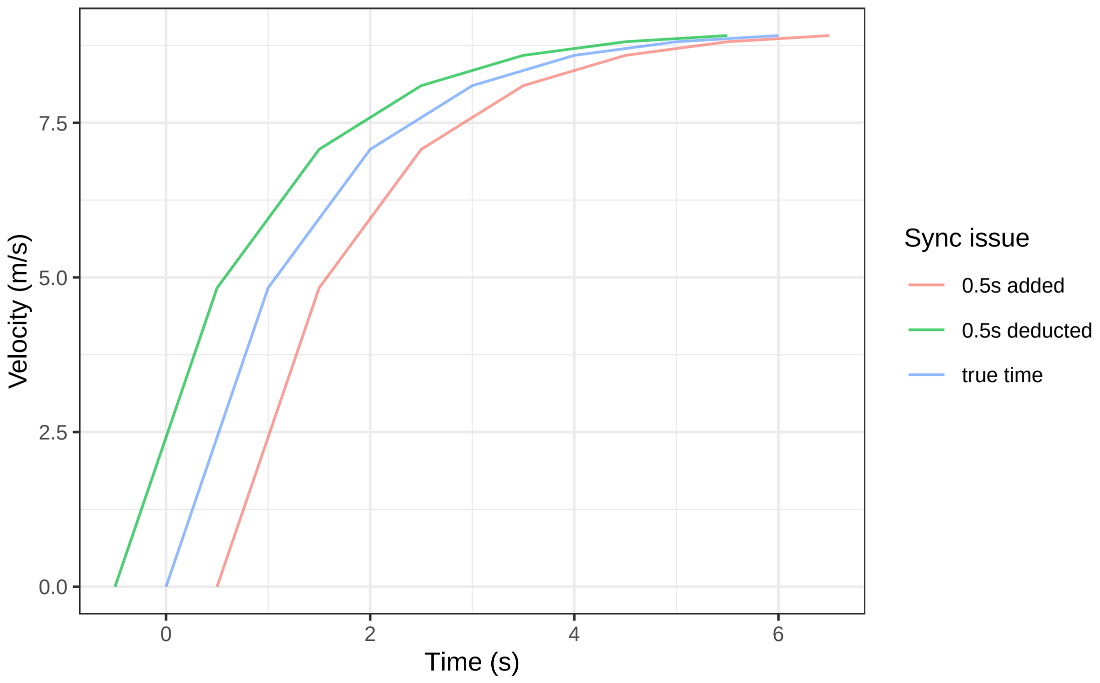
With the previous examples, estimation can look simple. Unfortunately, in real life things are a bit more *messy*, and this messiness can bias the estimated MSS and TAU parameters. In the next section, we will demonstrate how real life issues can introduce biases, but we will also present potential solutions within the `shorts` package.

3.1 Problems with time sync with radar gun

One source of ~~messiness~~ in the estimation using radar gun, ~~beside measurement error,~~ is the time synchronization, where zero velocity ~~happens~~ at $t = 0$. Let's use our athlete and add and deduct 0.5second to simulate ~~bad~~ synchronization and its effect on estimated MSS and TAU.

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

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
```

```
plot_df <- pivot_longer(df, cols = -2, names_to = "Sync issue")

ggplot(plot_df, aes(x = value, y = velocity, color = `Sync issue`)) +
  theme_bw(10) +
  geom_line(alpha = 0.7) +
  xlab("Time (s)") +
  ylab("Velocity (m/s)")
```



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

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

# With time added
```

```
m6 <- shorts::model_using_radar(
  velocity = df$velocity,
  time = df$`0.5s added`
)

# With time deducted
m7 <- shorts::model_using_radar(
  velocity = df$velocity,
  time = df$`0.5s deducted`
)

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  9.000997 1.300100 6.923312 15.57918               0
#> 2    Added 0.5s time  9.912729 2.342198 4.232233 10.48824               0
#> 3 Deducted 0.5s time 10.075994 1.856318 5.427945 13.67299               0
#>   distance_correction
#> 1                   0
#> 2                   0
#> 3                   0
```

As can be seen from the example, ~~it's the~~ TAU ~~estimate that is mostly~~ affected by a ~~bad~~ synchronization of time with velocity. ~~Solution implemented~~ in `shorts` package involves estimation of the *time correction* parameters using the following equation:

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

This model ~~is~~ utilizes ~~using~~ the `shorts::model_using_radar_with_time_correction` function:

```
# With time added
m8 <- shorts::model_using_radar_with_time_correction(
  velocity = df$velocity,
  time = df$`0.5s added`
)
coef(m8)
#>                 MSS                 TAU                 MAC                PMAX
#>           9.0010048           1.3001127           6.9232495          15.5790506
#>     time_correction distance_correction
#>          -0.4999887           0.0000000
```

```r
# With time deducted
m9 <- shorts::model_using_radar_with_time_correction(
  velocity = df$velocity,
  time = df$`0.5s deducted`
)
coef(m9)
#>                 MSS                 TAU                 MAC                PMAX 
#>           9.0010048           1.3001127           6.9232495          15.5790506 
#>     time_correction distance_correction 
#>           0.5000113           0.0000000
```

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
round(
  predict_velocity_at_time(
    time = df$`true time`,
    MSS = m5$parameters$MSS,
    TAU = m5$parameters$TAU
  ),
  2
)
#> [1] 0.00 4.83 7.07 8.11 8.59 8.81 8.91

# Using time with sync issues
round(
  predict_velocity_at_time(
    time = df$`0.5s added`,
    MSS = m8$parameters$MSS,
    TAU = m8$parameters$TAU,
    time_correction = m8$parameters$time_correction
  ),
  2
)
#> [1] 0.00 4.83 7.07 8.11 8.59 8.81 8.91
```

3.1.1 Mixed-model approach

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

```r
# Adding 0.5s to radar_gun_data
radar_gun_data$time <- radar_gun_data$time + 0.5

# Mixed model with time correction being fixed effect
m10 <- shorts::mixed_model_using_radar_with_time_correction(
  radar_gun_data,
  time = "time",
  velocity = "velocity",
  athlete = "athlete",
  random = MSS + TAU ~ 1
)

m10
#> Estimated fixed model parameters
#> --------------------------------
#>                MSS                TAU                MAC               PMAX 
#>          8.3010696          1.0076178          8.2383117         17.0966996 
#>    time_correction distance_correction 
#>         -0.5001251          0.0000000 
#> 
#> Estimated random model parameters
#> ----------------------------------
#>     athlete      MSS       TAU      MAC     PMAX time_correction
#> 1     James 9.998418 1.1106775 9.002089 22.50166      -0.5001251
#> 2       Jim 7.997855 0.8885163 9.001361 17.99789      -0.5001251
#> 3      John 7.999943 1.0688684 7.484498 14.96889      -0.5001251
#> 4 Kimberley 9.005352 1.2853859 7.005952 15.77277      -0.5001251
#> 5  Samantha 6.503780 0.6846410 9.499548 15.44574      -0.5001251
#>   distance_correction
#> 1                   0
#> 2                   0
#> 3                   0
#> 4                   0
#> 5                   0
#> 
#> Model fit estimators
#> --------------------
#>         RSE   R_squared      minErr      maxErr   maxAbsErr        RMSE 
#>  0.05164778  0.99942172 -0.21898003  0.19828533  0.21898003  0.05156164 
#>         MAE        MAPE 
#>  0.03950528         Inf


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

m11
#> Estimated fixed model parameters
#> --------------------------------
#>                MSS                TAU                MAC               PMAX 
#>          8.3010618          1.0076291          8.2382113         17.0964754 
#>    time_correction distance_correction 
#>         -0.5001121          0.0000000 
#> 
#> Estimated random model parameters
#> ----------------------------------
#>     athlete      MSS       TAU      MAC     PMAX time_correction
#> 1     James 9.998234 1.1104471 9.003792 22.50550      -0.5002956
#> 2       Jim 7.997868 0.8885384 9.001151 17.99751      -0.5001077
#> 3      John 8.000002 1.0689597 7.483913 14.96783      -0.5000570
#> 4 Kimberley 9.005356 1.2853906 7.005929 15.77272      -0.5001216
#> 5  Samantha 6.503849 0.6848099 9.497306 15.44226      -0.4999783
#>   distance_correction
#> 1                   0
#> 2                   0
#> 3                   0
#> 4                   0
#> 5                   0
#>
#> Model fit estimators
#> --------------------
#>         RSE   R_squared      minErr      maxErr   maxAbsErr        RMSE
#>  0.05164744  0.99942173 -0.21876869  0.19823176  0.21876869  0.05156130
#>         MAE        MAPE
#>  0.03950374         Inf
```

3.2 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 6.92m/s^2. Let's call them John and Jack. The are both performing 40m sprint using timing gates set at 5, 10, 20, 30, and 40m. The initial timing gate at the start serves ~~the purpose of activating~~ the timing system (i.e., when they cross the beam).

John is our *theoretical model*, while Jack is *street smart guy*, most likely playing soccer, and decides to move slightly back (i.e. 0.5m) and use body rocking to initiate the start. Let's see how their sprints differ.

```
MSS <- 9
TAU <- 1.3
MAC <- MSS / TAU

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
)

split_times
#> # A tibble: 5 x 6
#>   distance john_time jack_distance jack_true_time time_05m jack_time
#>      <dbl>  <I<dbl>>         <dbl>       <I<dbl>> <I<dbl>>  <I<dbl>>
#> 1        5      1.42           5.5           1.50    0.400      1.10
#> 2       10      2.17          10.5           2.23    0.400      1.83
#> 3       20      3.43          20.5           3.49    0.400      3.09
#> 4       30      4.60          30.5           4.65    0.400      4.25
#> 5       40      5.73          40.5           5.78    0.400      5.39
```

~~As expected, Jack performs better than John, even if they have same short sprint capabilities. Let's see how this affects the MSS and TAU estimates:~~

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

john_profile <- shorts::model_using_splits(
  distance = split_times$distance,
  time = split_times$john_time
)

jack_profile <- shorts::model_using_splits(
  distance = split_times$distance,
  time = split_times$jack_time
)

sprint_parameters <- rbind(
  unlist(john_profile$parameters),
  unlist(jack_profile$parameters)
)

rownames(sprint_parameters) <- c("John", "Jack")

round(sprint_parameters, 2)
#>       MSS TAU   MAC  PMAX time_correction distance_correction
#> John 9.00 1.3  6.92 15.58               0                   0
#> Jack 8.49 0.7 12.06 25.62               0                   0
```

As can be seen from the results, ~~*cheating* (or not having consistent start) at the begin of the sprint~~ yields biased estimates, particularly for the TAU, MAC and PMAX. ~~Let's create a small simulation to estimate effects of cheating distance over combinations of MSS and MAC on their estimates as well as residuals.~~

~~Simulated splits are~~ 5, 10, 20, 30, 40, and 50m, with MSS and MAC varying from 6 to 9, and ~~cheating~~ distance varying from 0 to 1m.

```
sim_df <- expand.grid(
  MSS = c(6, 7, 8, 9),
  MAC = c(6, 7, 8, 9),
  cheat_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)
)

sim_df <- sim_df %>%
  mutate(
    TAU = MSS / MAC,
    PMAX = MSS * MAC / 4,
    true_distance = distance + cheat_distance,
    true_time = shorts::predict_time_at_distance(true_distance, MSS, TAU),
    stolen_time = shorts::predict_time_at_distance(cheat_distance, MSS, TAU),
    time = true_time - stolen_time
  )

# Add small noise to allow model fit
set.seed(1667)
rand_noise <- rnorm(nrow(sim_df), 0, 10^-4)
sim_df$time <- sim_df$time + rand_noise

head(round(sim_df, 2))
#>   MSS MAC cheat_distance distance  TAU  PMAX true_distance true_time
#> 1   6   6              0        5 1.00  9.00             5      1.64
#> 2   7   6              0        5 1.17 10.50             5      1.58
#> 3   8   6              0        5 1.33 12.00             5      1.54
#> 4   9   6              0        5 1.50 13.50             5      1.51
#> 5   6   7              0        5 0.86 10.50             5      1.55
#> 6   7   7              0        5 1.00 12.25             5      1.49
#>   stolen_time time
#> 1           0 1.64
#> 2           0 1.58
#> 3           0 1.54
#> 4           0 1.51
#> 5           0 1.55
#> 6           0 1.49
```

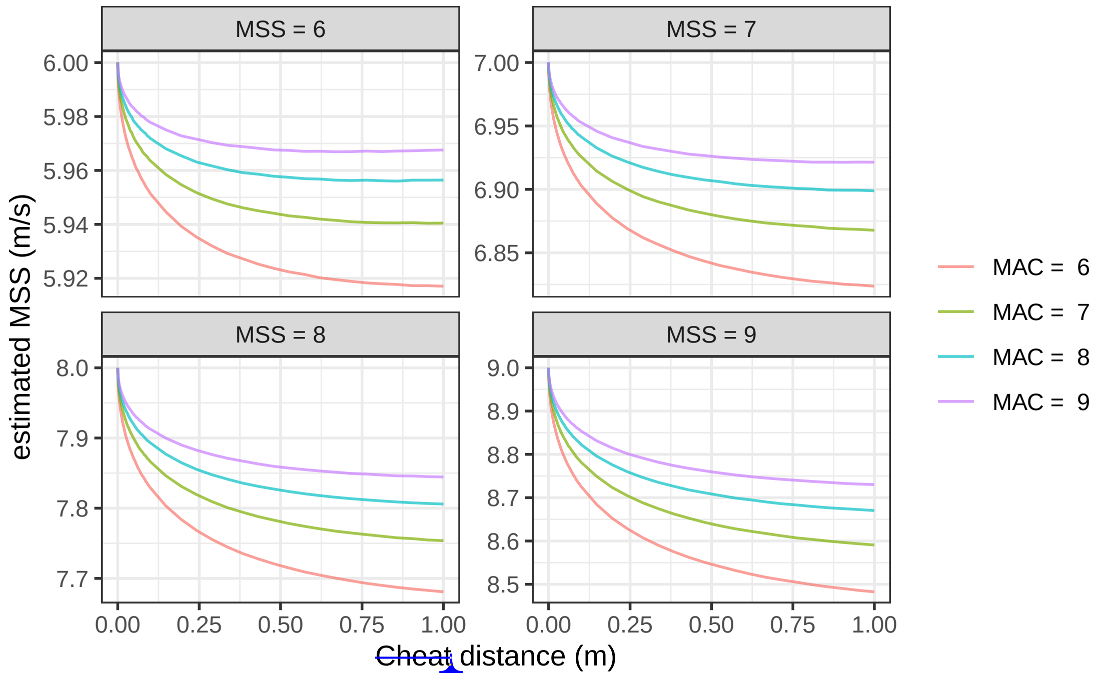
Now when we have a simulation dataset, we can check the model estimates and predictions, given the ~~cheating~~ distance:

```
# Prediction wrapper
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
# estimated parameters and predited time
model_df <- sim_df %>%
  group_by(MSS, TAU, cheat_distance) %>%
  do(pred_wrapper(.)) %>%
  ungroup()

# Prediction residuals
model_df$residuals <- model_df$predicted_time - model_df$time
```

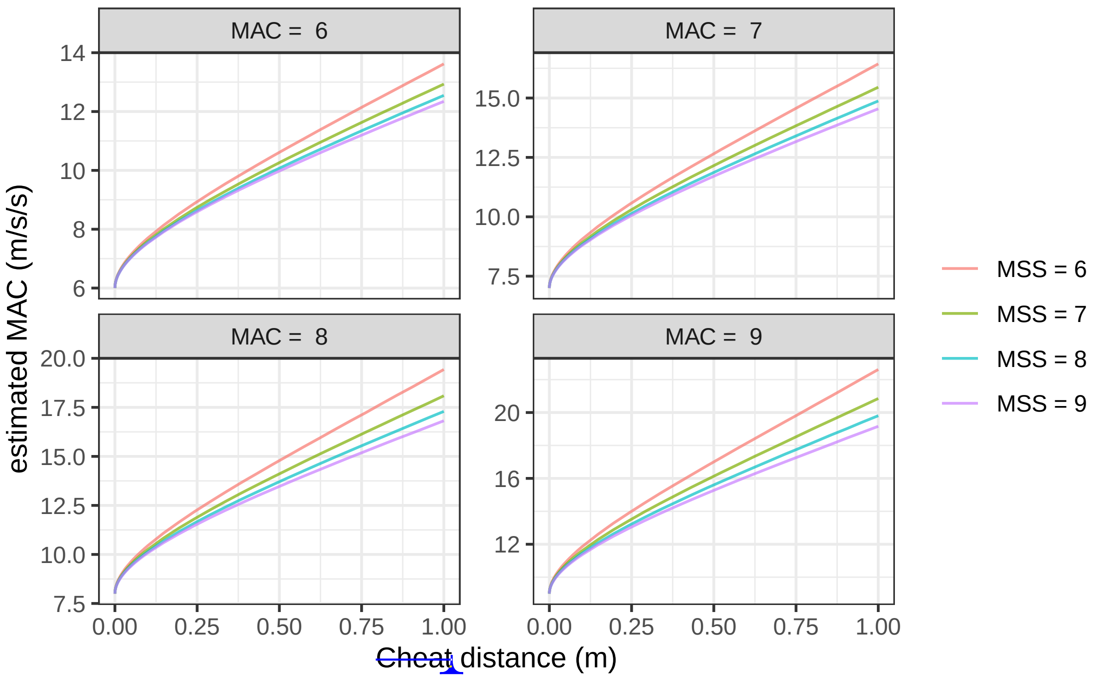
The following image demonstrates the effect of ~~cheating~~ distance on estimated MSS:

```r
# Estimates plot
df <- model_df %>%
  group_by(MSS, TAU, cheat_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))
  )

# MSS
ggplot(df, aes(x = cheat_distance, y = est_MSS, color = MAC_string)) +
  theme_bw() +
  geom_line(alpha = 0.7) +
  facet_wrap(~MSS_string, scales = "free_y") +
  xlab("Cheat distance (m)") +
  ylab("estimated MSS (m/s)") +
  theme(legend.title = element_blank())
```



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

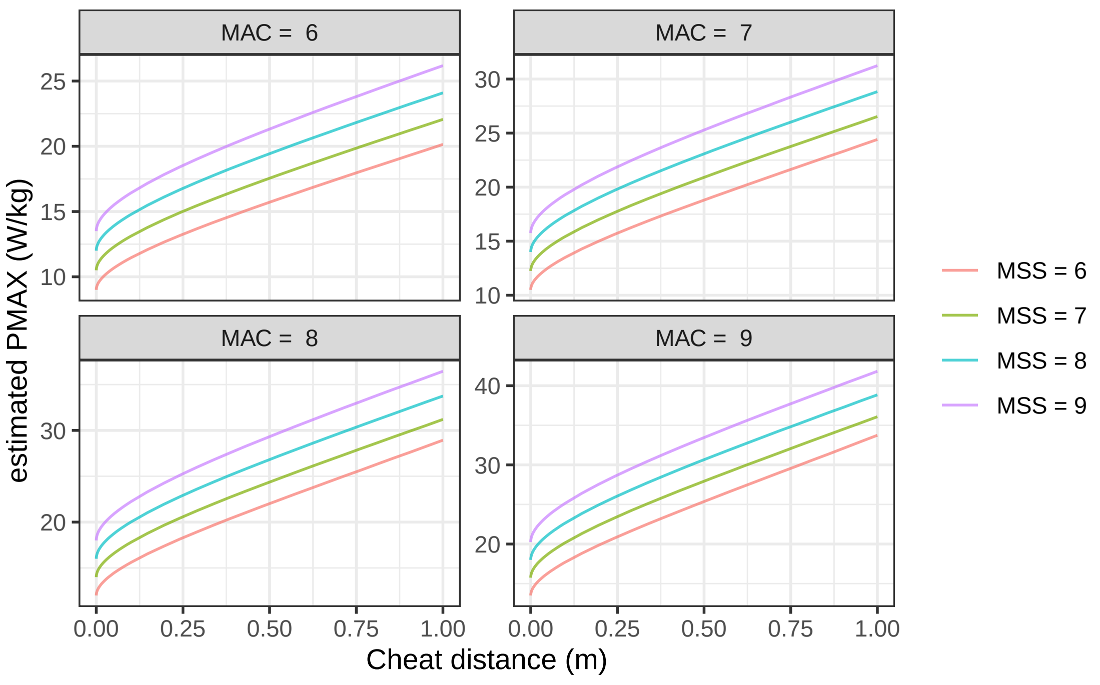
```
# MAC
ggplot(df, aes(x = cheat_distance, y = est_MAC, color = MSS_string)) +
  theme_bw() +
  geom_line(alpha = 0.7) +
  facet_wrap(~MAC_string, scales = "free_y") +
  xlab("Cheat distance (m)") +
  ylab("estimated MAC (m/s/s)") +
  theme(legend.title = element_blank())
```



MAC (and also TAU) are highly affected by ~~the cheating~~ 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 ~~cheating~~ distance on estimated PMAX:

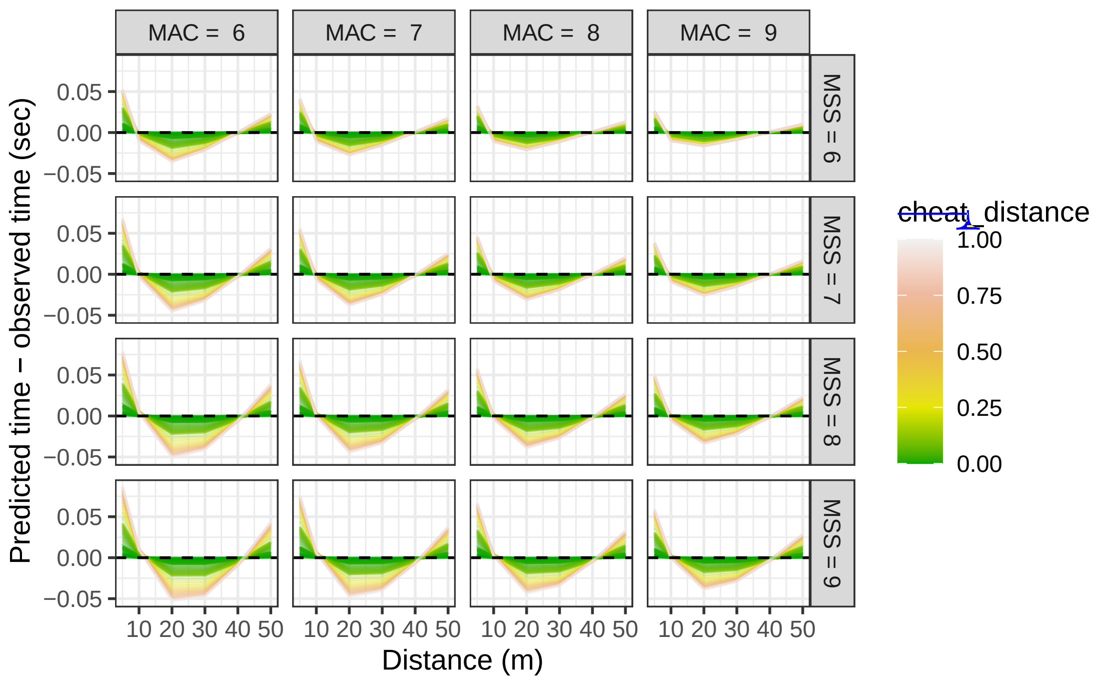
```
# PMAX
ggplot(df, aes(x = cheat_distance, y = est_PMAX, color = MSS_string)) +
  theme_bw() +
  geom_line(alpha = 0.7) +
  facet_wrap(~MAC_string, scales = "free_y") +
  xlab("Cheat distance (m)") +
  ylab("estimated PMAX (W/kg)") +
  theme(legend.title = element_blank())
```



Estimated PMAX is also overestimated as ~~cheat~~ distance increases.

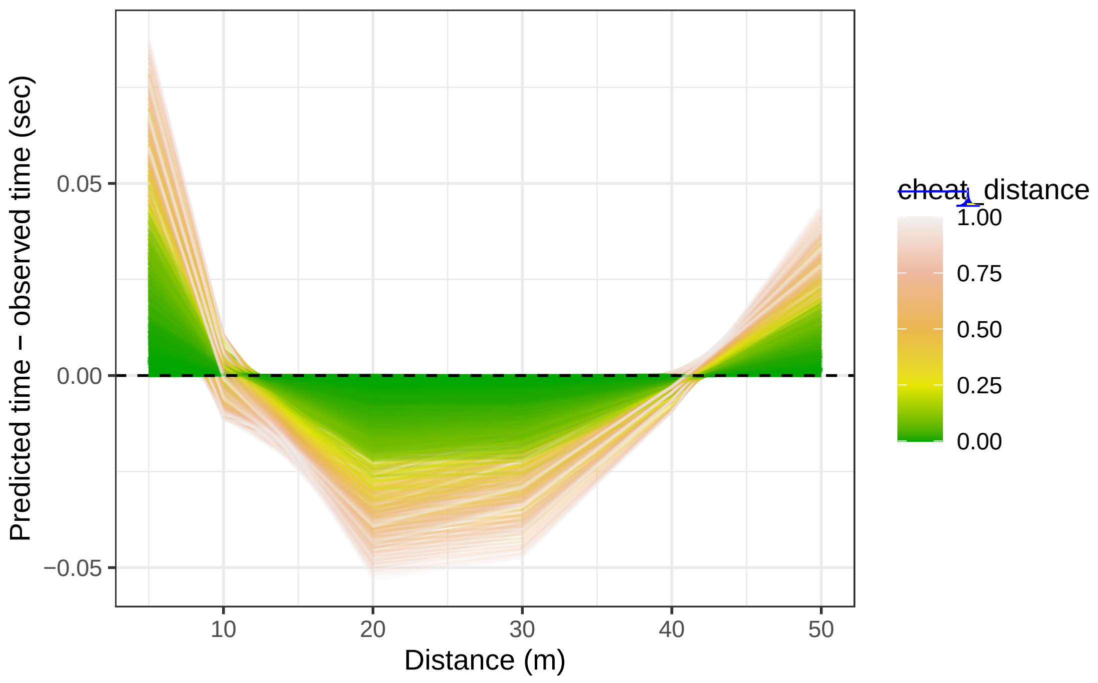
Model residuals are also affected by ~~cheating~~ distance. The shape of residuals depends on splits utilized, but here we can see the effect of the ~~cheating~~ distance on the model residuals per split distance:

```r
# Residuals
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, cheat_distance)
  )

ggplot(model_df, aes(y = residuals, x = distance, color = cheat_distance, group = group)) +
  theme_bw() +
  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 (sec)")
```



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

```r
ggplot(model_df, aes(y = residuals, x = distance, color = cheat_distance, group = group)) +
  theme_bw() +
  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 (sec)")
```



Clearly, ~~cheating and sprint start timing issues and inconsistencies~~ can result in biased parameters and predictions. ~~This is particularly important in testing and comparing test results implementing different starting techniques and timing initiations (**REFERENCE**).~~ Since speed ~~trait~~ is one of the hardest to improve, the effects of start inconsistencies can mask effects of the training intervention. It is thus ~~of~~ crucial ~~importance~~ to ~~take these into account, either by standardizing~~ the start ~~(which is always a must), or~~ by implementing the following techniques.

3.2.1 How to ~~deal with the cheating or sprint start timing issues~~?

~~Besides standardizing sprint starts, particularly for longitudinal monitoring of the training effects, one can implement additional strategies to make model estimates more robust.~~

~~The simplest strategy is to used validity study results (i.e. comparing timing technique of interest to a gold standard measurement) and to use correction factors (**REFERENCE**). One way to implement correction is to use *time correction* in a same manner used with the radar gun synchronization issues.~~ Thus, equation ?? gets another form:

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

~~If using validity studies, this time correction can be provided (i.e., 0.3-0.5seconds; *REFERENCE*), which is oftentimes the case, or it can me estimated as MSS and TAU is estimated.~~ To estimate time correction, we use `shorts::model_using_splits_with_time_correction` function. Here is how we can estimate Jack parameters using these two methods:

```
jack_profile_fixed_time <- shorts::model_using_splits(
  distance = split_times$distance,
  time = split_times$jack_time,
  time_correction = 0.3
)
```

```
jack_profile_time_estimated <- shorts::model_using_splits_with_time_correction(
  distance = split_times$distance,
  time = split_times$jack_time
)

jack_parameters <- rbind(
  unlist(john_profile$parameters),
  unlist(jack_profile$parameters),
  unlist(jack_profile_fixed_time$parameters),
  unlist(jack_profile_time_estimated$parameters)
)

rownames(jack_parameters) <- c(
  "John",
  "Jack - No corrections",
  "Jack - Fixed time correction",
  "Jack - Estimated time correction"
)

jack_parameters
#>                                       MSS       TAU       MAC     PMAX
#> John                             9.000075 1.3000302  6.922974 15.57682
#> Jack - No corrections            8.494666 0.7042268 12.062400 25.61651
#> Jack - Fixed time correction     8.996166 1.2513983  7.188891 16.16811
#> Jack - Estimated time correction 8.958067 1.2159504  7.367132 16.49882
#>                                  time_correction distance_correction
#> John                                   0.0000000                   0
#> Jack - No corrections                  0.0000000                   0
#> Jack - Fixed time correction           0.3000000                   0
#> Jack - Estimated time correction       0.2837092                   0
```

In Jack's case, both 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 ulitize *distance correction*, besides time correction. Thus, equation ?? gets another form:

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

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

```
jack_profile_distance_correction <- shorts::model_using_splits_with_corrections(
  distance = split_times$distance,
  time = split_times$jack_time
)

jack_parameters <- rbind(
  unlist(john_profile$parameters),
  unlist(jack_profile$parameters),
  unlist(jack_profile_fixed_time$parameters),
  unlist(jack_profile_time_estimated$parameters),
  unlist(jack_profile_distance_correction$parameters)
)

rownames(jack_parameters) <- c(
  "John",
  "Jack - No corrections",
  "Jack - Fixed time correction",
  "Jack - Estimated time correction",
  "Jack - Estimated distance correction"
)

jack_parameters
#>                                           MSS       TAU       MAC     PMAX
#> John                                 9.000075 1.3000302  6.922974 15.57682
#> Jack - No corrections                8.494666 0.7042268 12.062400 25.61651
#> Jack - Fixed time correction         8.996166 1.2513983  7.188891 16.16811
#> Jack - Estimated time correction     8.958067 1.2159504  7.367132 16.49882
#> Jack - Estimated distance correction 9.000291 1.3005017  6.920630 15.57192
#>                                      time_correction distance_correction
#> John                                       0.0000000           0.0000000
#> Jack - No corrections                      0.0000000           0.0000000
#> Jack - Fixed time correction               0.3000000           0.0000000
#> Jack - Estimated time correction           0.2837092           0.0000000
#> Jack - Estimated distance correction       0.4001907           0.5029882
```

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 estimated, 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 to splits) might be practically troublesome.

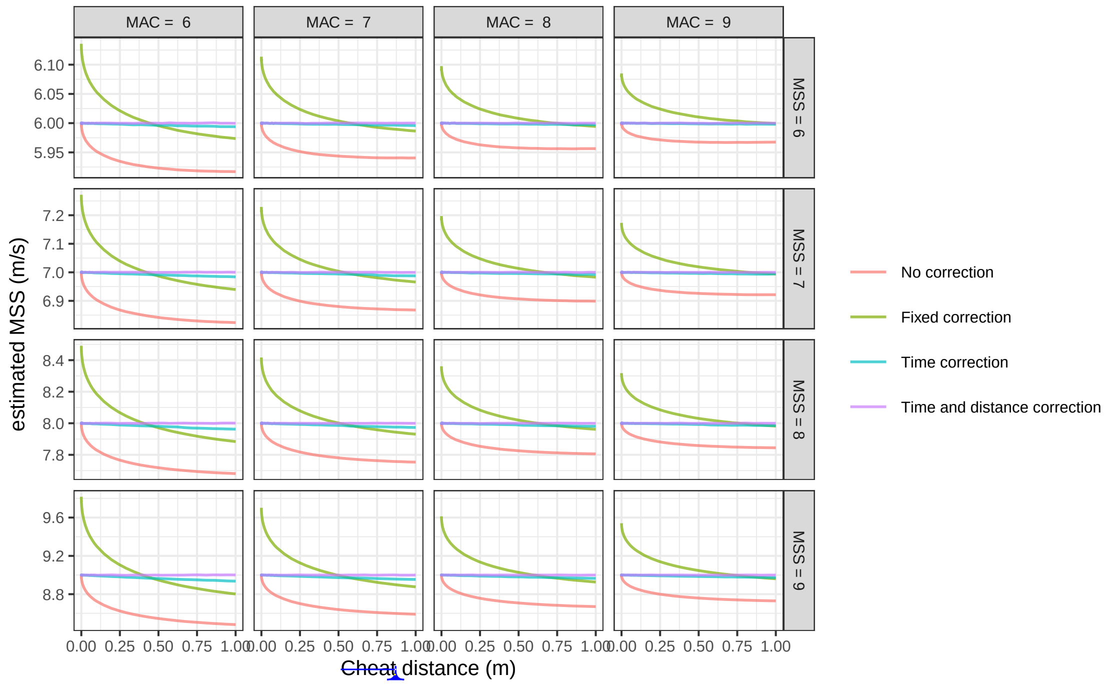
We will get back to these issues later, but let's see how these models perform using simulated data with varying ~~cheating~~ distance. The following code contains the wrapper that performs all four models (no correction, fixed time correction, estimated time correction, and estimated time and distance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

```
pred_wrapper <- function(data) {
  no_correction <- shorts::model_using_splits(
    distance = data$distance,
    time = data$time
  )

  fixed_correction <- shorts::model_using_splits(
    distance = data$distance,
    time = data$time,
    time_correction = 0.3
  )

  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(
```

```r
    distance = data$distance,
    time = data$time,
    control = nls.control(tol = 1)
  )


  params <- rbind(
    data.frame(
      model = "No correction",
      t(unlist(no_correction$parameters))
    ),
    data.frame(
      model = "Fixed correction",
      t(unlist(fixed_correction$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)
}

# estimated parameters and predited time
model_df <- sim_df %>%
  group_by(MSS, TAU, cheat_distance) %>%
  do(pred_wrapper(.)) %>%
  ungroup()
```

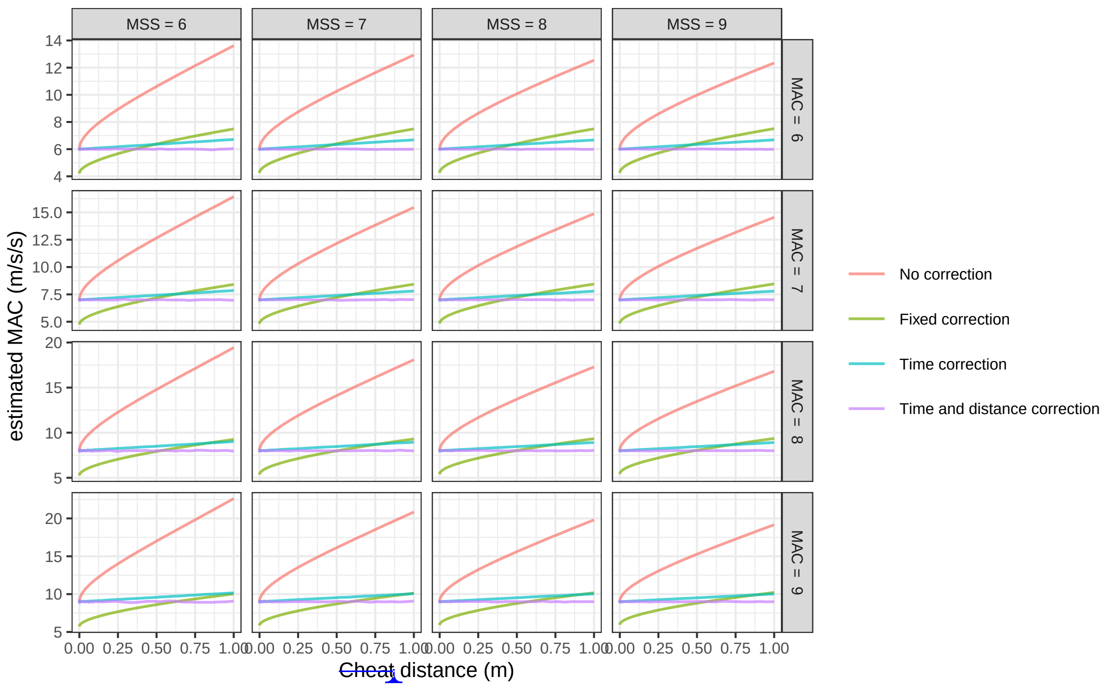
As can be seen from the next figure, estimated time correction model estimates MSS almost perfectly, while estimated time and distance correction model estimates MSS perfectly.

```r
model_df$model <- factor(
  model_df$model,
  levels = c(
    "No correction",
    "Fixed correction",
    "Time correction",
    "Time and distance correction"
  )
)
# Estimates plot
df <- model_df %>%
  group_by(MSS, TAU, cheat_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))
  )

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



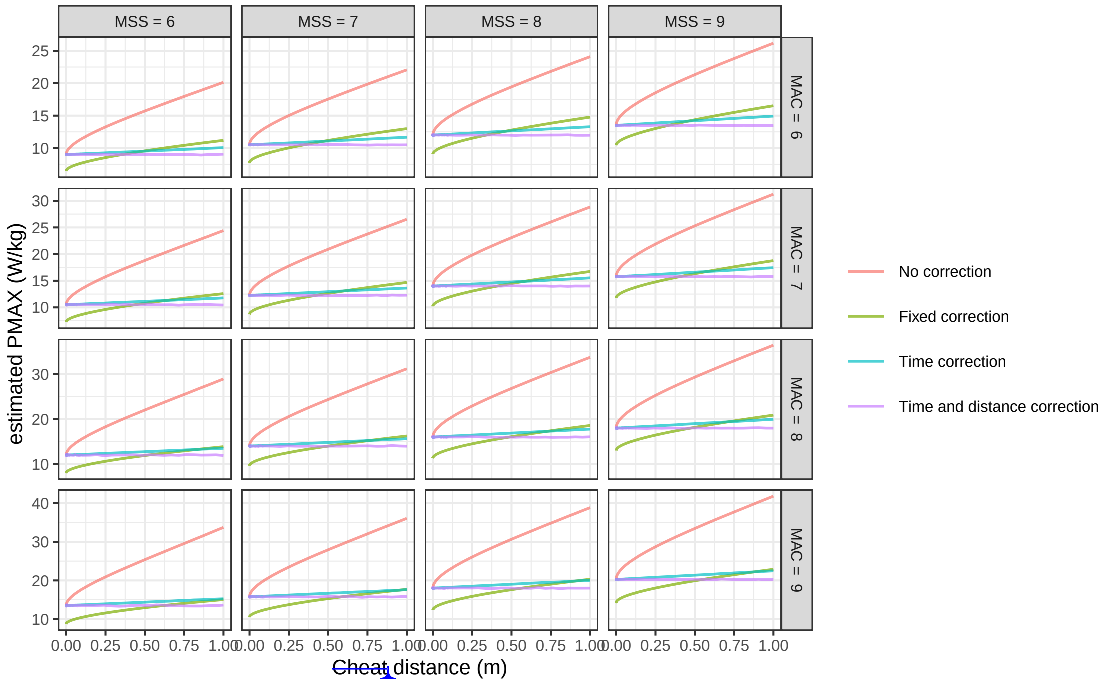
Same conclusions can be said for the MAC parameter. Time and distance corrections model performs perfectly, while time correction model performs almost as good (although worse than for MSS parameter).

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



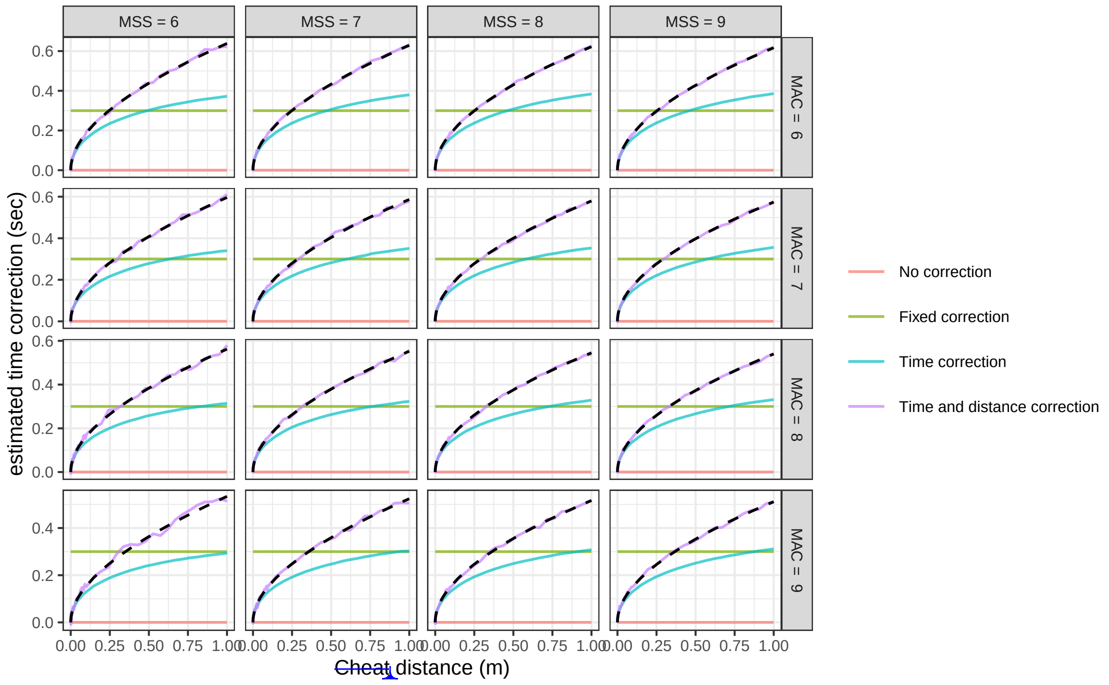
~~When it comes to PMAX,~~ same conclusion can be reached ~~as~~ for MAC.

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



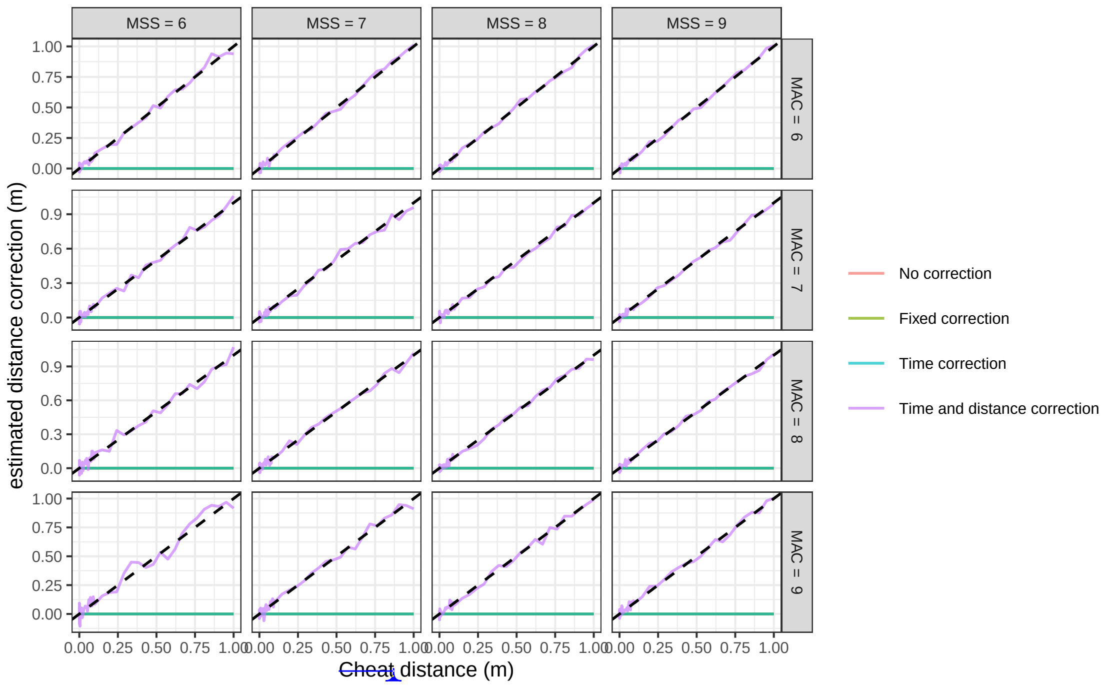
The following figure depicts estimated time correction, and as can be seen, only the time and distance correction model estimated the time correction correctly (i.e., the stolen time; indicated by the dashed line on the figure).

```r
# time_correction
ggplot(df, aes(x = cheat_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("Cheat distance (m)") +
  ylab("estimated time correction (sec)") +
  theme(legend.title = element_blank())
```



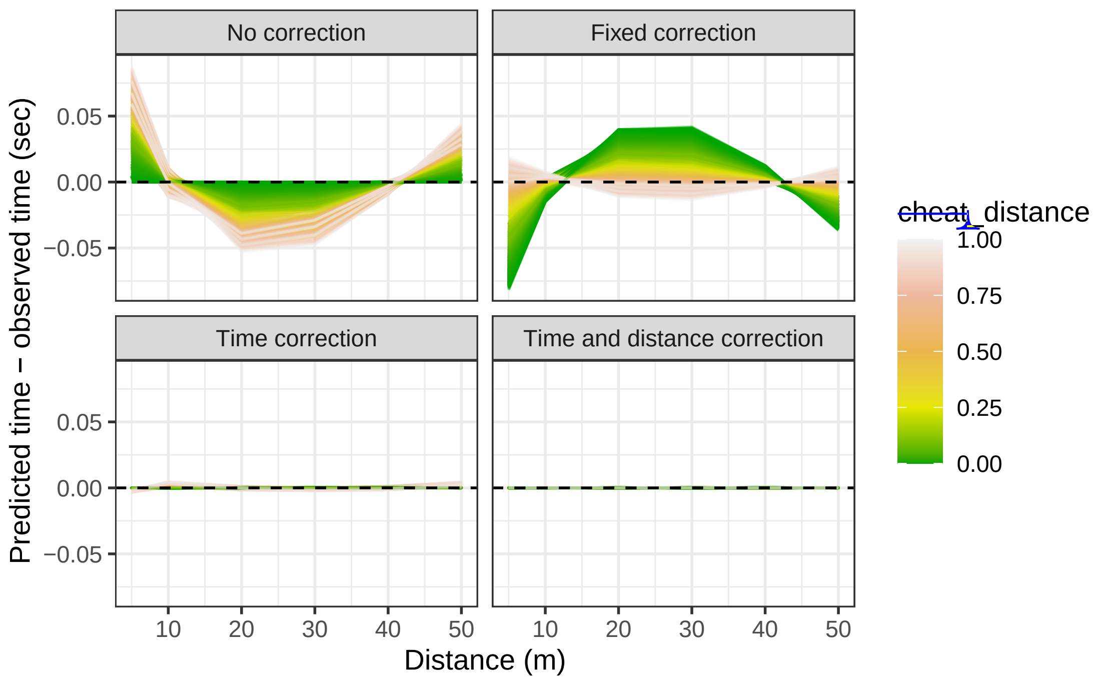
The following figure depicts estimated distance correction, and same as with the time correction, only the time and distance correction model estimated the distance correction correctly (i.e., ~~cheat~~ distance; indicated by the dashed line on the figure, which represents *identity line* since ~~cheat~~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

```
# distance_correction
ggplot(df, aes(x = cheat_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("Cheat 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, particularly without issues at specific distance zones.

```
# Residuals
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, cheat_distance)
  )

ggplot(model_df, aes(y = residuals, x = distance, color = cheat_distance, group = group)) +
  theme_bw() +
  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 (sec)")
```



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 start cheating. 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_correctio`
We will showcase their use at the end of this article.

3.2.2 Simulation of additional starting issues

Starting behind the initial timing gate represent only one issue (i.e. ~~cheating).~~ In this section, we ~~will~~ simulate few other issues to check how sensitive ~~are~~ the presented models to perturbations that can happen in ~~the~~ practice.

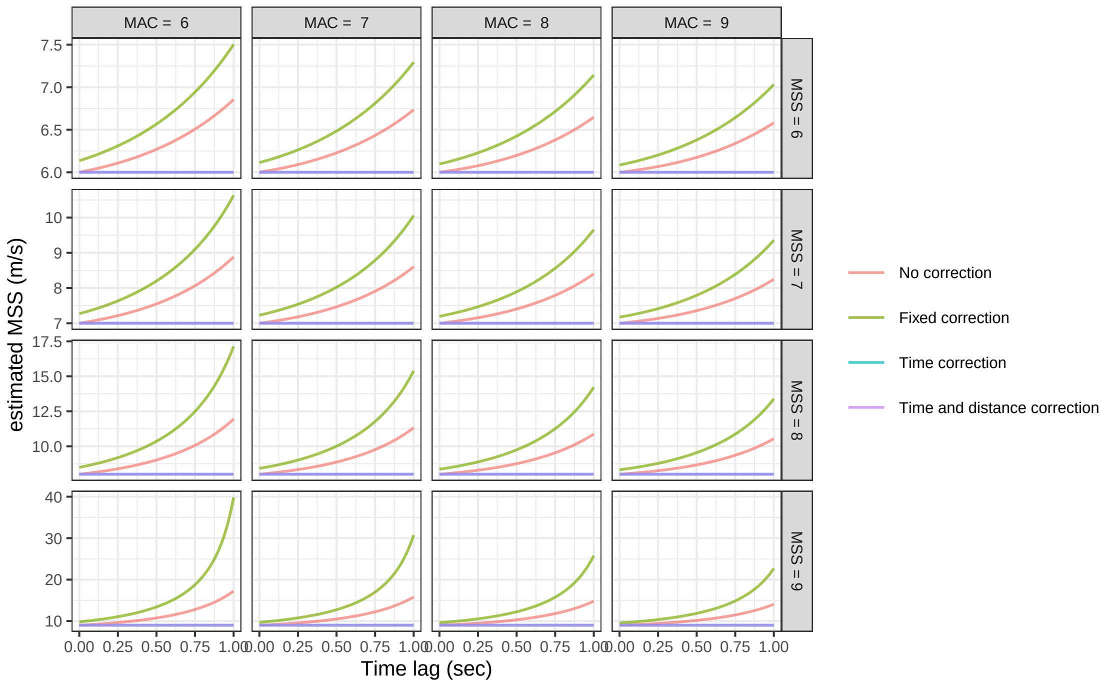
3.2.2.1 Triggering the gate before the start One issue that might happen in certain timing gates setups is an individual triggering timing system before the sprint is initiated. This is very similar to the situation when timing starts on a signal (i.e., gun during 100m sprint race) and there is *reaction 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.

```
sim_df <- expand.grid(
  MSS = c(6, 7, 8, 9),
  MAC = c(6, 7, 8, 9),
  time_lag = seq(0, 1, length.out = 50),
  distance = c(5, 10, 20, 30, 40, 50)
)

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
  )

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

head(round(sim_df, 2))
#>   MSS MAC time_lag distance  TAU  PMAX true_time time
#> 1   6   6        0        5 1.00  9.00      1.64 1.64
#> 2   7   6        0        5 1.17 10.50      1.58 1.58
#> 3   8   6        0        5 1.33 12.00      1.54 1.54
#> 4   9   6        0        5 1.50 13.50      1.51 1.51
#> 5   6   7        0        5 0.86 10.50      1.55 1.55
#> 6   7   7        0        5 1.00 12.25      1.49 1.49
```

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

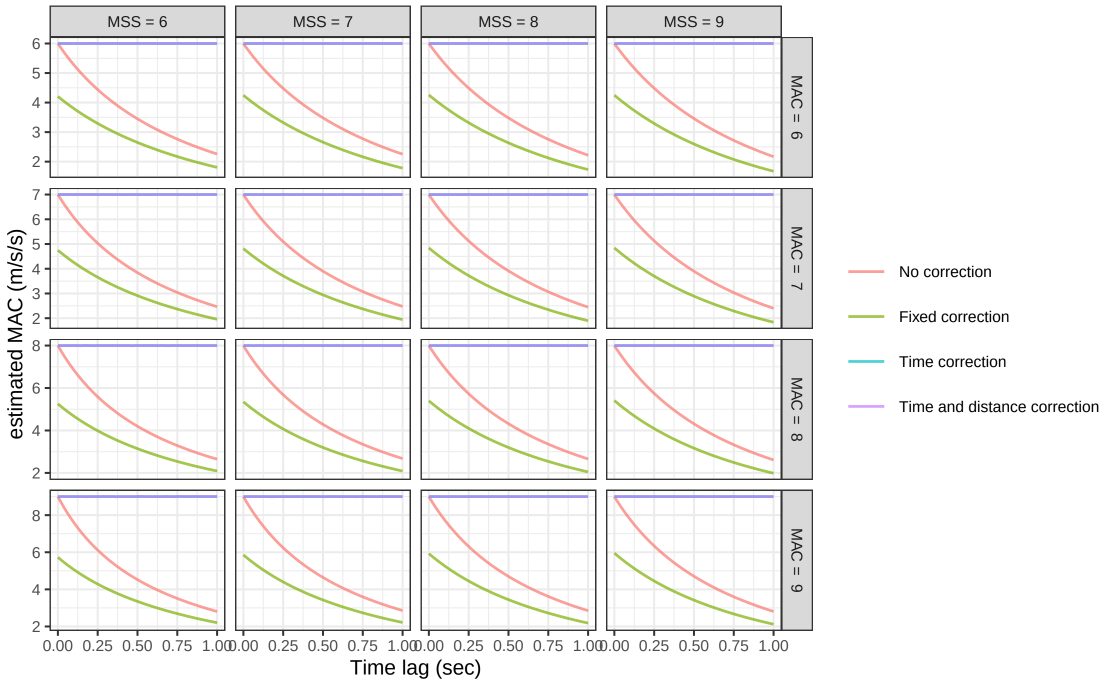
From the figure below it can be seen that time lag affects estimated MSS for the the model without correction and fixed correction model. Time correction and time and distance corrections model correctly estimated MSS.

```
model_df$model <- factor(
  model_df$model,
  levels = c(
    "No correction",
    "Fixed correction",
    "Time correction",
    "Time and distance correction"
  )
)
# Estimates plot
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))
  )

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



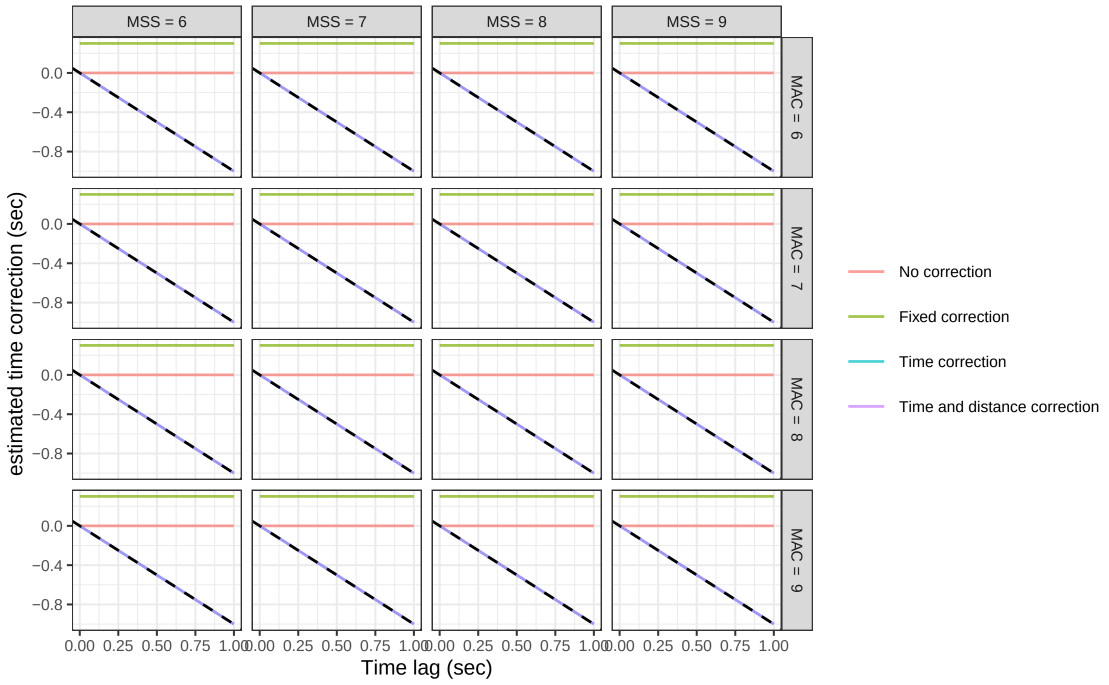
From the figure below it can be seen that time lag affects estimated MAC for the the model without correction and fixed correction model. Time correction and time and distance corrections model correctly estimated MAC.

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



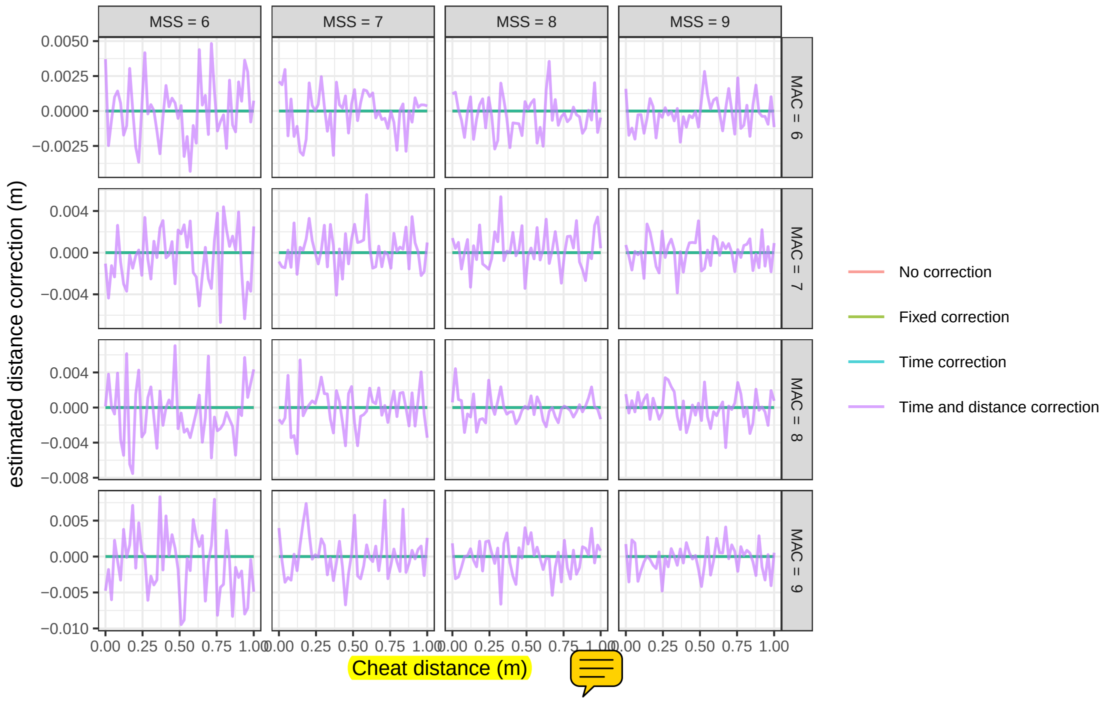
Two figure below depicts correctly identified time lag (i.e. using time correction parameter) using time correction and time and distance corrections models.

```r
# time_correction
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 (sec)") +
  ylab("estimated time correction (sec)") +
  theme(legend.title = element_blank())
```



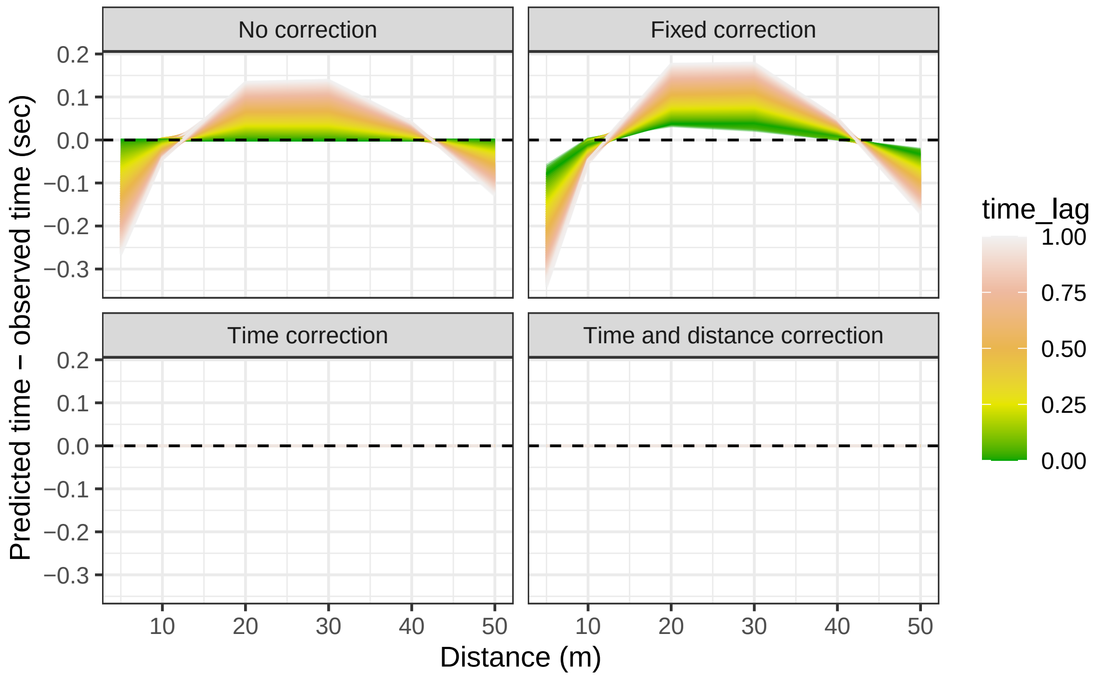
The next figure depicts estimated distance correction for the time and distance correction model. The estimated distance correction parameters looks jumpy due random noise that we have to inject to allow model fit.

```r
# distance_correction
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("Cheat distance (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
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)
  )

ggplot(model_df, aes(y = residuals, x = distance, color = time_lag, group = group)) +
  theme_bw() +
  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 (sec)")
```



3.2.2.2 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 0.5m).

```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)
)

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)
  )

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

head(round(sim_df, 2))
#>   MSS MAC displacement distance  TAU  PMAX true_distance time
#> 1   6   6         -0.5        5 1.00  9.00           4.5 1.53
#> 2   7   6         -0.5        5 1.17 10.50           4.5 1.48
#> 3   8   6         -0.5        5 1.33 12.00           4.5 1.44
#> 4   9   6          -0.5         5 1.50 13.50            4.5 1.42
#> 5   6   7          -0.5         5 0.86 10.50            4.5 1.45
#> 6   7   7          -0.5         5 1.00 12.25            4.5  1.4
```

```
# estimated parameters and predicted time
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.

```
model_df$model <- factor(
  model_df$model,
  levels = c(
    "No correction",
    "Fixed correction",
    "Time correction",
    "Time and distance correction"
  )
)

# Estimates plot
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))
  )

# MSS
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())
```

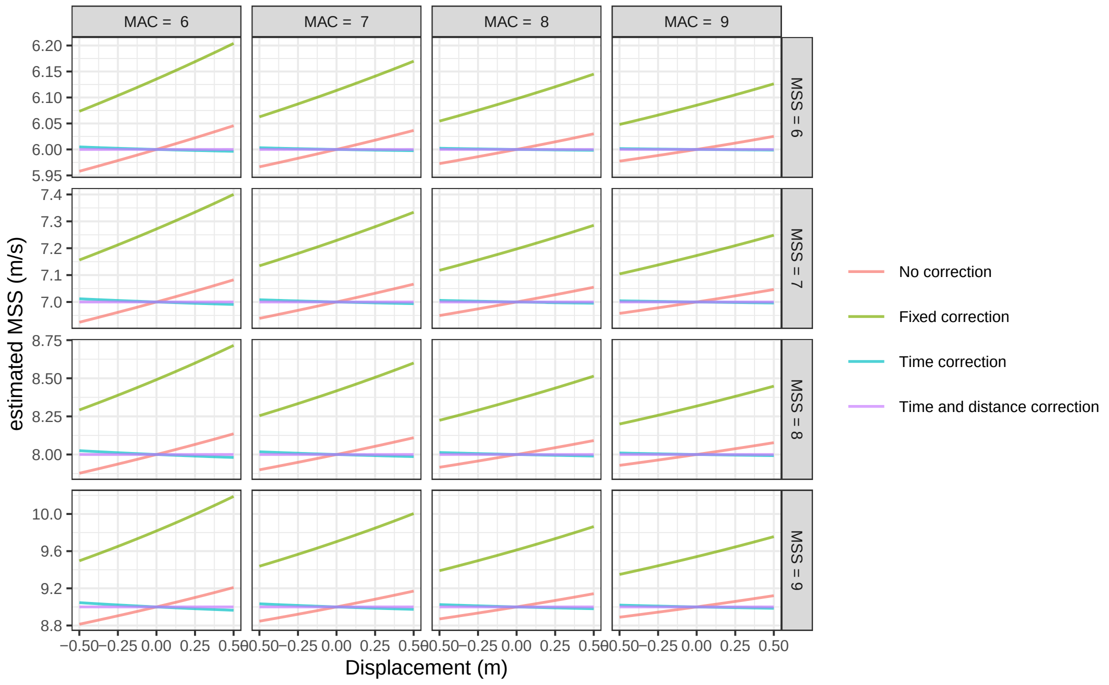
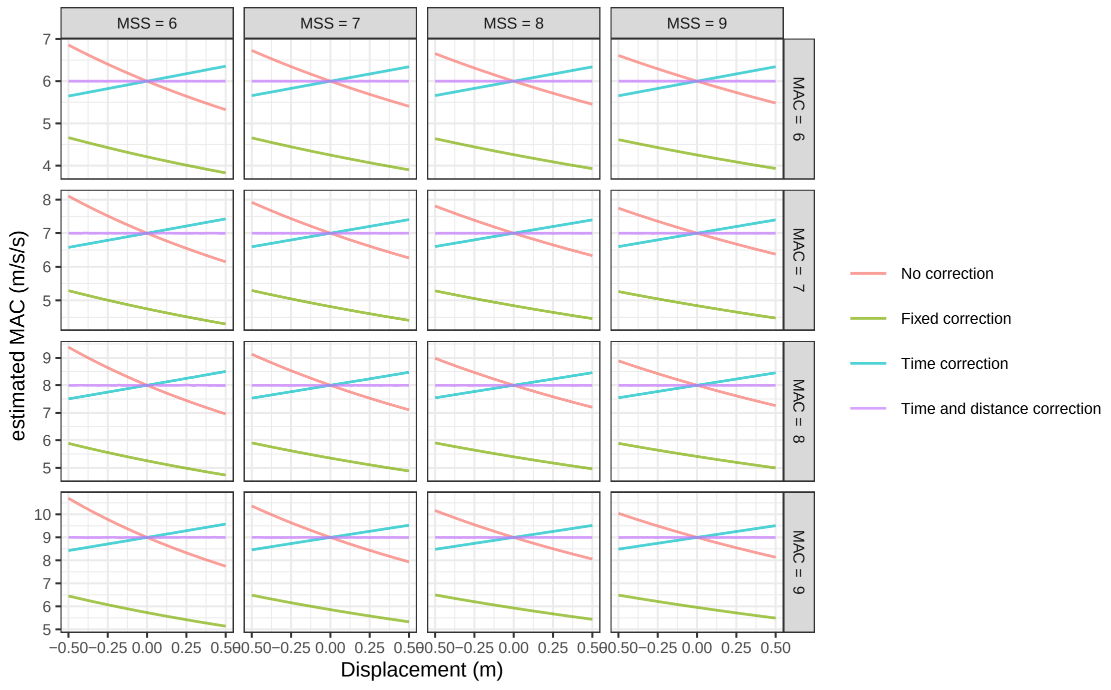


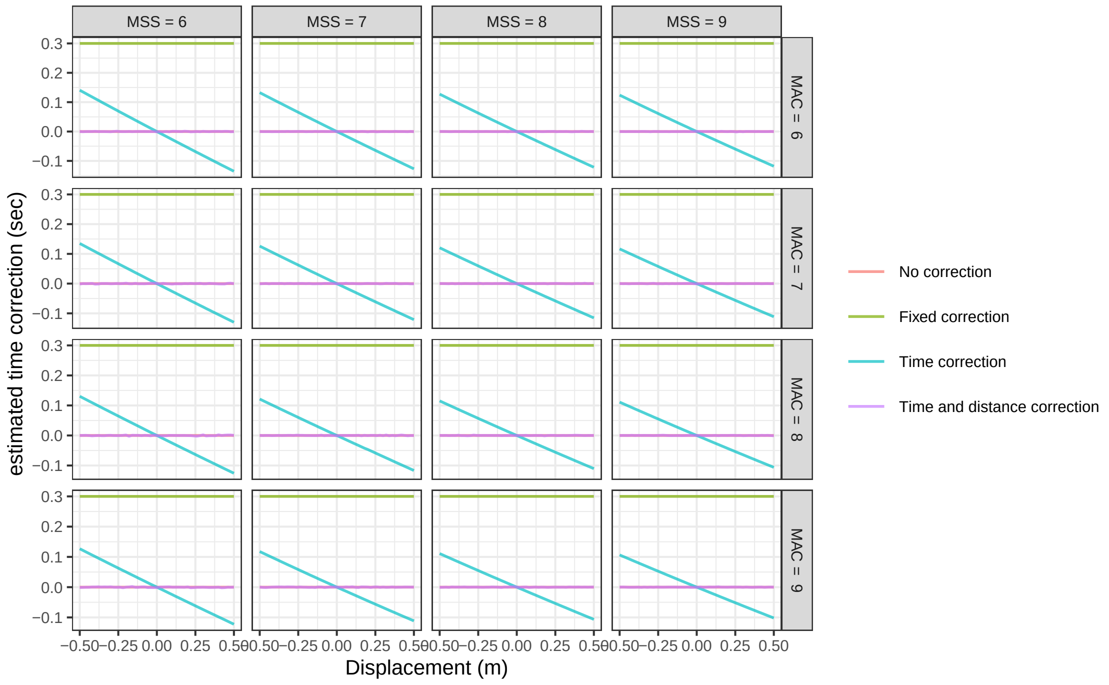
Figure below depicts effect of timing gate displacement on the MSS parameter. Time and distance correction model is the only model that estimated MAC correctly.

```r
# MAC
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())
```

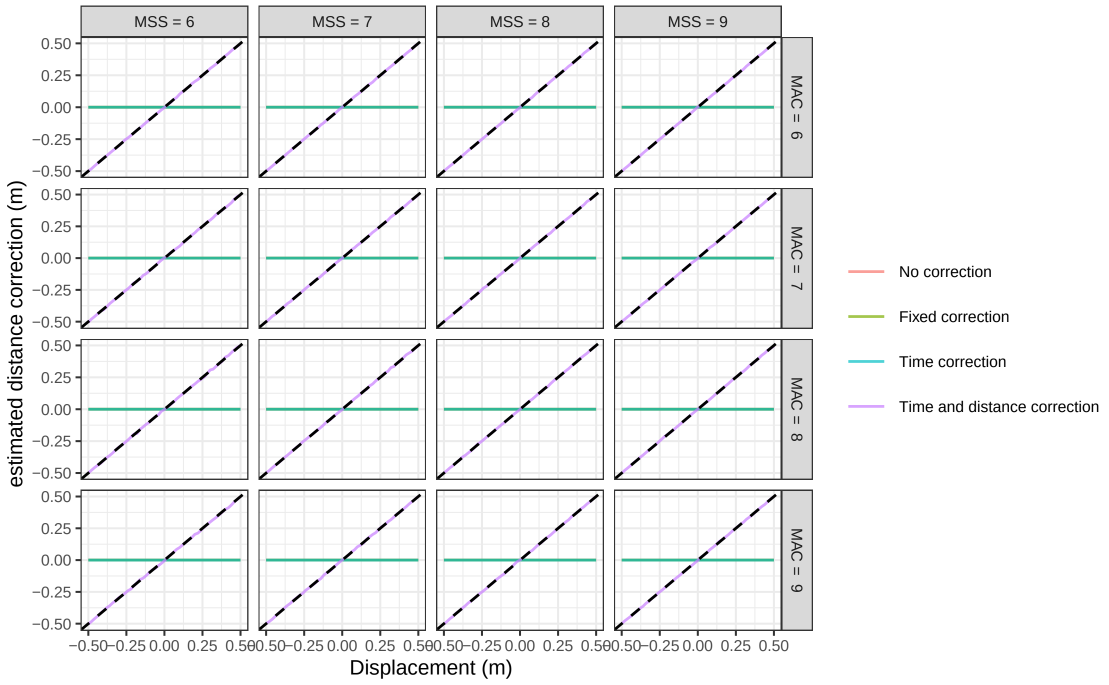


Figures below depicts estimated time and distance corrections parameters.

```
# time_correction
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 (sec)") +
  theme(legend.title = element_blank())
```

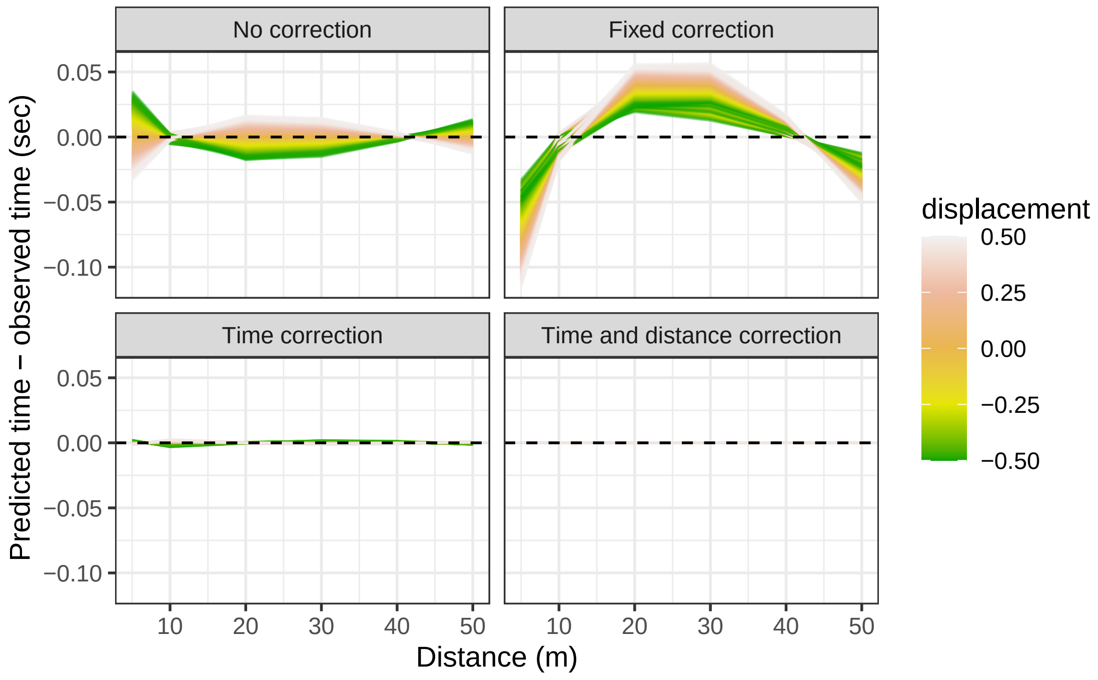


```r
# distance_correction
ggplot(df, aes(x = displacement, y = est_distance_correction, color = model)) +
  theme_bw(8) +
  geom_line(alpha = 0.7) +
  geom_abline(slope = 1, color = "black", linetype = "dashed") +
  facet_grid(MAC_string ~ MSS_string, scales = "free_y") +
  xlab("Displacement (m)")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



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

```r
# Residuals
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)
  )

ggplot(model_df, aes(y = residuals, x = distance, color = displacement, group = group)) +
  theme_bw() +
  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 (sec)")
```



Additional sources of issues with the timing gate involve combination of the previous three. 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.

4 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 et al. 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.

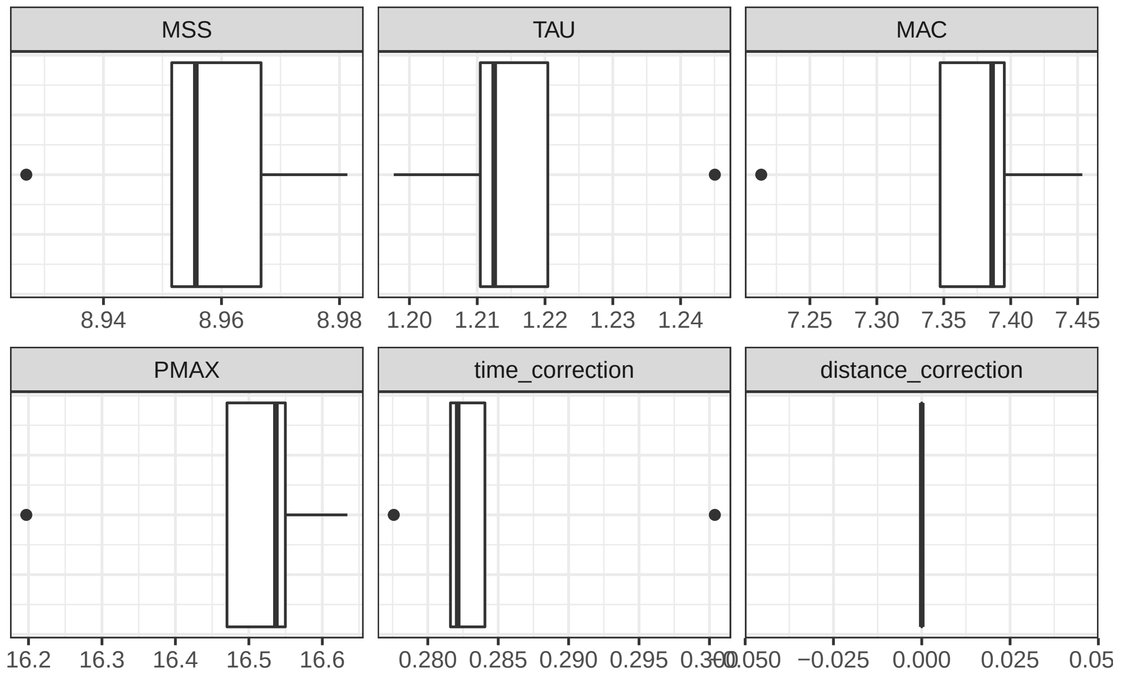
Let's perform LOOCV using Jack's data and time correction model:

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

jack_LOOCV
#> Estimated model parameters
#> --------------------------
#>               MSS              TAU              MAC             PMAX
#>         8.9580670        1.2159504        7.3671322       16.4988160
#>     time_correction distance_correction
#>         0.2837092        0.0000000
#>
#> Model fit estimators
#> --------------------
#>          RSE    R_squared       minErr       maxErr    maxAbsErr         RMSE
#>  0.001806726  0.999999462 -0.001087146  0.001887354  0.001887354  0.001142674
#>          MAE         MAPE
#>  0.001043315  0.049810042
#> 
#> 
#> Leave-One-Out Cross-Validation
#> ------------------------------
#> Parameters:
#>        MSS      TAU      MAC     PMAX time_correction distance_correction
#> 1 8.981334 1.245051 7.213627 16.19700       0.3003841                   0
#> 2 8.966705 1.220412 7.347275 16.47021       0.2840574                   0
#> 3 8.951566 1.210453 7.395217 16.54969       0.2816093                   0
#> 4 8.955649 1.212504 7.386081 16.53679       0.2821228                   0
#> 5 8.926920 1.197687 7.453469 16.63413       0.2775823                   0
#> 
#> Model fit:
#>          RSE    R_squared       minErr       maxErr    maxAbsErr         RMSE
#>           NA  0.999995768 -0.006394368  0.005096562  0.006394368  0.004011220
#>          MAE         MAPE
#>  0.003485425  0.183873171
```

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

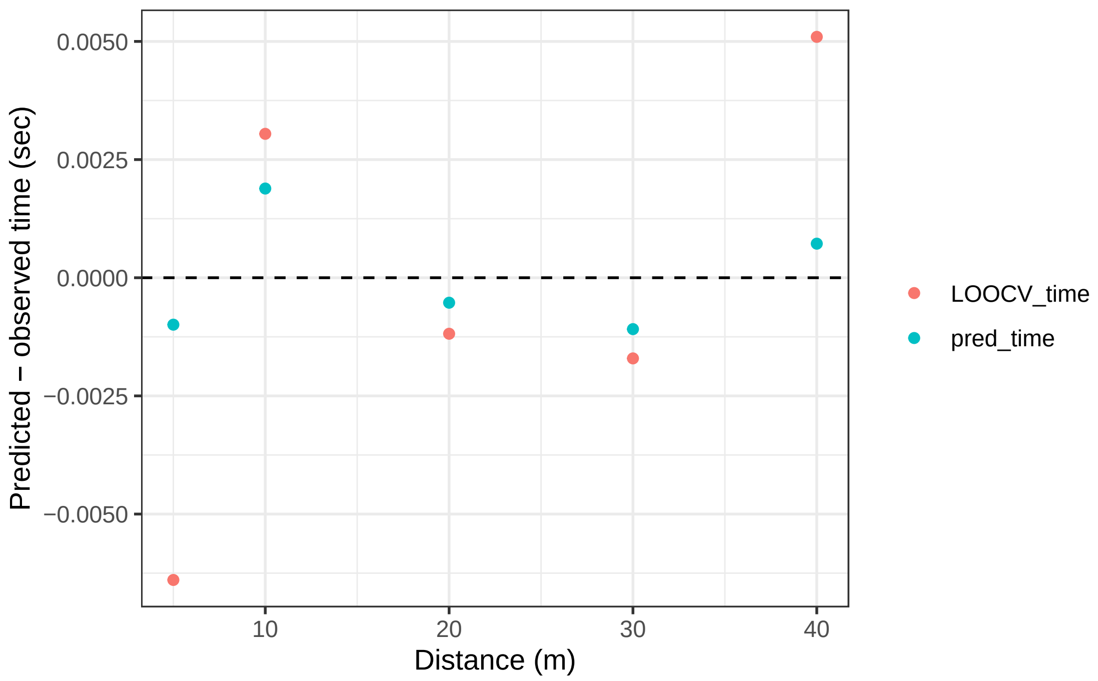
Let's plot estimated parameters across LOOCV folds:

```r
df <- jack_LOOCV$LOOCV$parameters

df <- pivot_longer(df, cols = 1:6, names_to = "parameter")

df$parameter <- factor(
  df$parameter,
  levels = c(
    "MSS",
    "TAU",
    "MAC",
    "PMAX",
    "time_correction",
    "distance_correction"
  )
)

ggplot(df, aes(x = value)) +
  theme_bw() +
  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
)

df <- df %>%
  pivot_longer(cols = c("pred_time", "LOOCV_time"))

df$resid <- df$value - df$time

ggplot(df, aes(x = distance, y = resid, color = name)) +
  theme_bw() +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_point() +
  theme(legend.title = element_blank()) +
  xlab("Distance (m)") +
  ylab("Predicted - observed time (sec)")
```



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.

5 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 athletes which comes with the `shorts` package.

5.1 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
# Taken from London 2017
# https://barrunning.com/resources/SiteUploads/20180715/London%202017%20REPORT%20FOR%20THE%20100%20m%20
bolt_reaction_time <- 0.183

bolt_distance <- c(10, 20, 30, 40, 50, 60)
bolt_time <- c(1.963, 2.983, 3.883, 4.763, 5.643, 6.493)

# No corrections model
bolt_m1 <- shorts::model_using_splits(
  distance = bolt_distance,
  time = bolt_time
)

# Model with reaction time as fixed time correction
bolt_m2 <- shorts::model_using_splits(
  distance = bolt_distance,
  time = bolt_time,
  time_correction = -bolt_reaction_time
)

# Model with estimated time correction
bolt_m3 <- shorts::model_using_splits_with_time_correction(
  distance = bolt_distance,
  time = bolt_time
)

# Model with estimated time correction, but deducted reaction time
bolt_m4 <- shorts::model_using_splits_with_time_correction(
  distance = bolt_distance,
  time = bolt_time - bolt_reaction_time
)

# Model with estimated time and distance corrections
bolt_m5 <- shorts::model_using_splits_with_corrections(
  distance = bolt_distance,
  time = bolt_time
)

# Model with estimated time and distance corrections and deducted reaction time
bolt_m6 <- shorts::model_using_splits_with_corrections(
  distance = bolt_distance,
  time = bolt_time - bolt_reaction_time
)

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))
  )
)

rownames(bolt_model) <- bolt_model$model

round(bolt_model[-1], 3)
#>                              MSS   TAU    MAC   PMAX time_correction
#> No correction             12.145 1.564  7.766 23.579           0.000
#> No correction - RT        11.736 1.205  9.737 28.569          -0.183
#> Time correction           11.732 1.202  9.761 28.629          -0.185
#> Time correction - RT      11.732 1.202  9.761 28.629          -0.002
#> Distance correction       11.591 0.855 13.560 39.293          -0.812
#> Distance correction - RT 11.591 0.855 13.560 39.293          -0.629
#>                          distance_correction
#> No correction                          0.000
#> No correction - RT                     0.000
#> Time correction                        0.000
#> Time correction - RT                   0.000
#> Distance correction                   -3.983
#> Distance correction - RT              -3.983
```

5.2 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 < 0.000), 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, 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. ~~The assessment of linear sprints using infrared timing gates does not require familiarization (Moir et al. 2004).~~

~~To analyze this dataset, we will utilize~~ mixed-effects models~~, although individual models can be utilized as well.~~

```
data("vescovi")

# Convert data to long
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]+"))
  )

head(df)
#> # A tibble: 6 x 10
#>   Team     Surface     Athlete   Age Height Bodyweight   BMI   BSA distance  time
#>   <chr>    <chr>       <fct>   <dbl>  <dbl>      <dbl> <dbl> <dbl>    <dbl> <dbl>
#> 1 W Socc~ Natural Gr~ FMSS-1~    21   163.       54.8  20.5   1.6        5  1.04
#> 2 W Socc~ Natural Gr~ FMSS-1~    21   163.       54.8  20.5   1.6       10  1.82
#> 3 W Socc~ Natural Gr~ FMSS-1~    21   163.       54.8  20.5   1.6       20  3.16
#> 4 W Socc~ Natural Gr~ FMSS-1~    21   163.       54.8  20.5   1.6       30  4.42
#> 5 W Socc~ Natural Gr~ FMSS-1~    21   163.       54.8  20.5   1.6       35  5.04
#> 6 FH Sr   Hard Court  FMSS-2~    21   172.       63.6  21.5   1.7        5  1.14
```

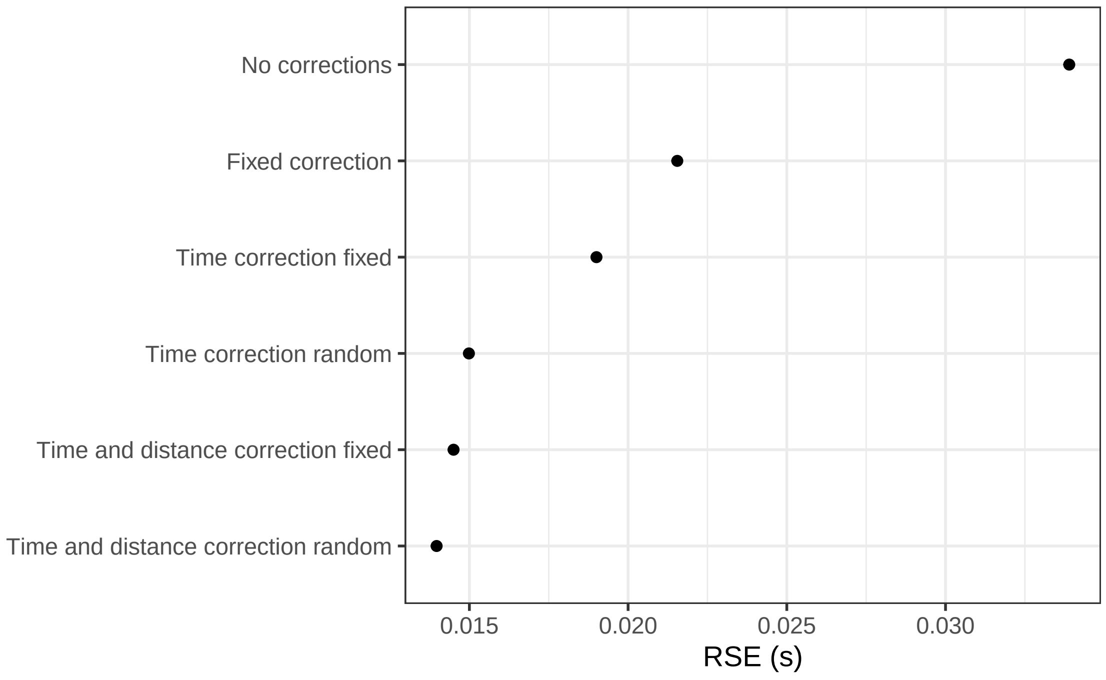
Here we will utilize the following models: no corrections model, fixed time correction model (using 0.3s heuristic rule of thumb), estimated time correction as a fixed effect model, estimated time correction as a random effect model, estimated distance correction as fixed effect model (and time correction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

```
no_corrections <- shorts::mixed_model_using_splits(
  df,
  distance = "distance",
  time = "time",
  athlete = "Athlete"
)

fixed_correction <- shorts::mixed_model_using_splits(
  df,
  distance = "distance",
  time = "time",
  athlete = "Athlete",
  time_correction = 0.3
)

time_correction_fixed <- shorts::mixed_model_using_splits_with_time_correction(
  df,
  distance = "distance",
  time = "time",
  athlete = "Athlete",
  random = MSS + TAU ~ 1
)

time_correction_random <- shorts::mixed_model_using_splits_with_time_correction(
  df,
  distance = "distance",
  time = "time",
  athlete = "Athlete",
  random = MSS + TAU + time_correction ~ 1
)

time_distance_correction_fixed <- shorts::mixed_model_using_splits_with_corrections(
```

```r
  df,
  distance = "distance",
  time = "time",
  athlete = "Athlete",
  random = MSS + TAU + time_correction ~ 1
)

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 ~~drops~~ the more flexible the model.

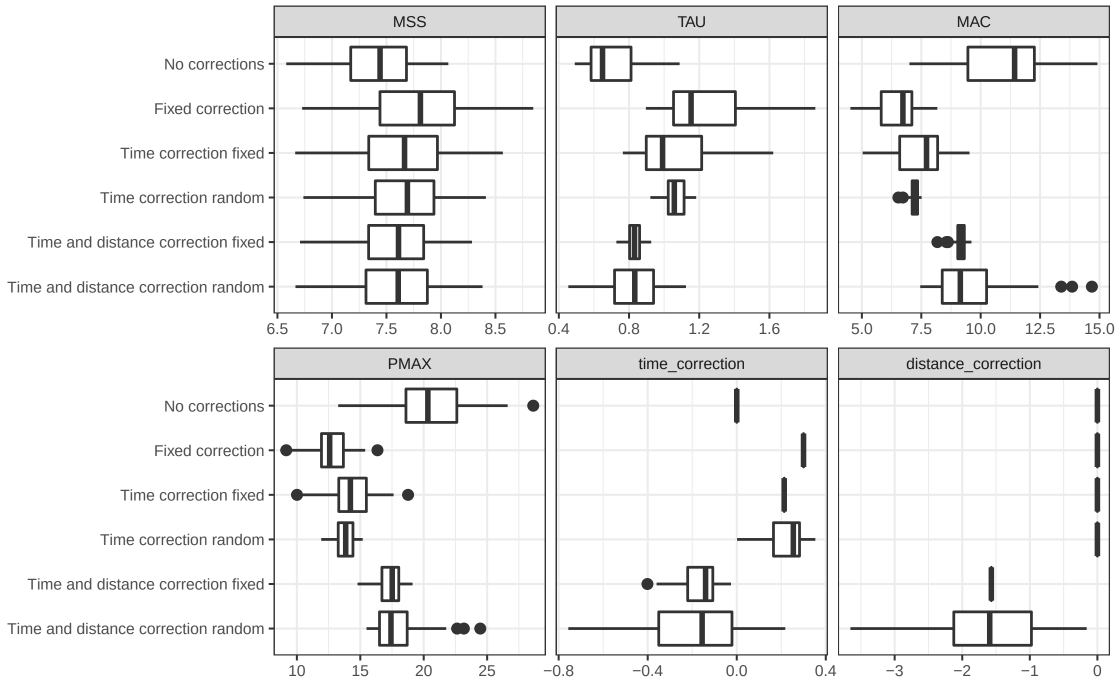
```r
model_fit <- rbind(
  data.frame(
    model = "No corrections",
    t(unlist(no_corrections$model_fit))
  ),
  data.frame(
    model = "Fixed correction",
    t(unlist(fixed_correction$model_fit))
  ),
  data.frame(
    model = "Time correction fixed",
    t(unlist(time_correction_fixed$model_fit))
  ),
  data.frame(
    model = "Time correction random",
    t(unlist(time_correction_random$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))
  )
)

model_fit$model <- factor(
  model_fit$model,
  levels = rev(c(
    "No corrections",
    "Fixed correction",
    "Time correction fixed",
    "Time correction random",
    "Time and distance correction fixed",
    "Time and distance correction random"
  ))
)

ggplot(model_fit, aes(x = RSE, y = model)) +
  theme_bw() +
  geom_point() +
  xlab("RSE (s)") +
  ylab(NULL)
```



The following image depicts estimated parameters for each model:

```
est_params <- rbind(
  data.frame(
    model = "No corrections",
    no_corrections$parameters$random
  ),
  data.frame(
    model = "Fixed correction",
    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(
```

```r
    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
  )
)

est_params$model <- factor(
  est_params$model,
  levels = rev(c(
    "No corrections",
    "Fixed correction",
    "Time correction fixed",
    "Time correction random",
    "Time and distance correction fixed",
    "Time and distance correction random"
  ))
)

est_params <- est_params %>%
  pivot_longer(cols = -(1:2), names_to = "parameter")

est_params$parameter <- factor(
  est_params$parameter,
  levels = c(
    "MSS",
    "TAU",
    "MAC",
    "PMAX",
    "time_correction",
    "distance_correction"
  )
)

ggplot(est_params, aes(y = model, x = value)) +
  theme_bw(8) +
  geom_boxplot() +
  facet_wrap(~parameter, scales = "free_x") +
  xlab(NULL) +
  ylab(NULL)
```

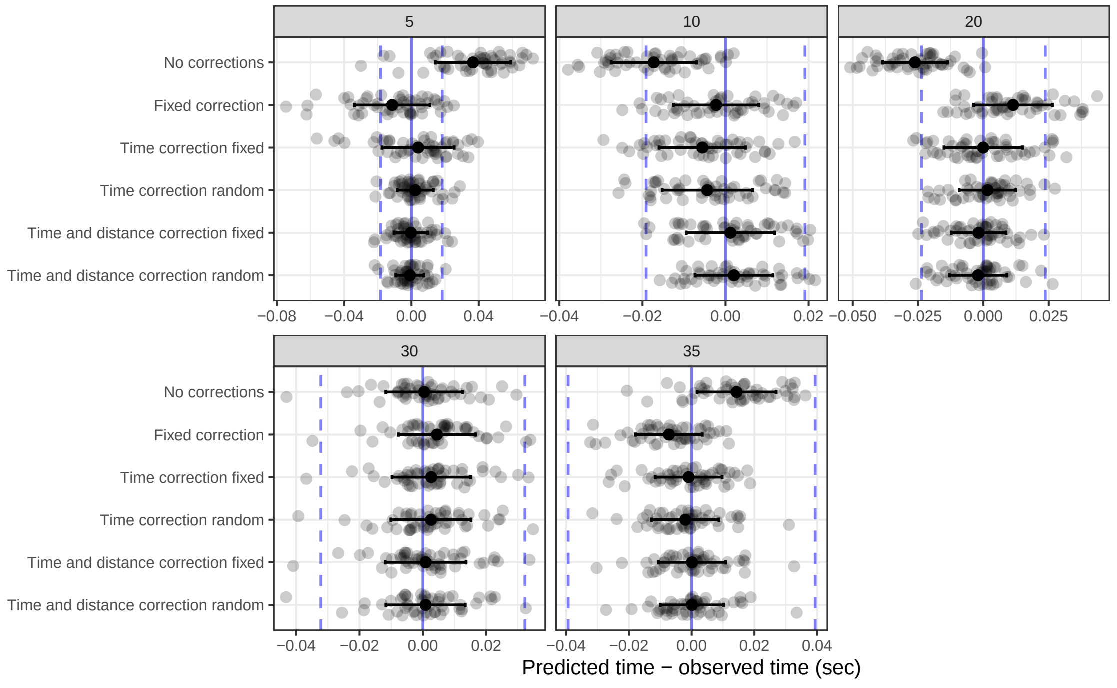


The following image depicts model residuals across distance splits. To provide practical magnitude of the residuals, we have used between subject observed time SD multiplied with 0.2 and -0.2. This provides practical anchor for the residual magnitude, often referred to as *smallest worthwhile change*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

Error bars represent residual bias and ± 1SD.

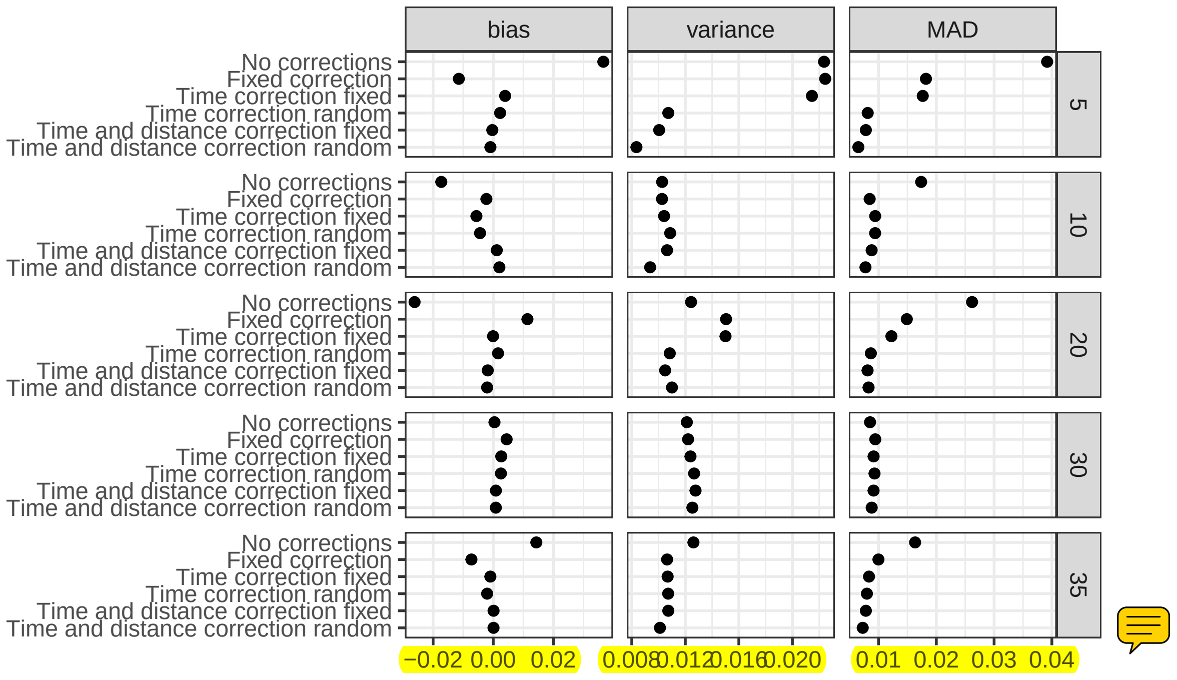
```
model_resid <- rbind(
  data.frame(
    model = "No corrections",
    no_corrections$data
  ),
  data.frame(
    model = "Fixed correction",
    fixed_correction$data
  ),
  data.frame(
    model = "Time correction fixed",
    time_correction_fixed$data
  ),
  data.frame(
    model = "Time correction random",
    time_correction_random$data
  ),
  data.frame(
    model = "Time and distance correction fixed",
    time_distance_correction_fixed$data
  ),
  data.frame(
    model = "Time and distance correction random",
```

```r
    time_distance_correction_random$data
  )
)


model_resid$model <- factor(
  model_resid$model,
  levels = rev(c(
    "No corrections",
    "Fixed correction",
    "Time correction fixed",
    "Time correction random",
    "Time and distance correction fixed",
    "Time and distance correction random"
  ))
)

model_resid$resid <- model_resid$pred_time - model_resid$time

# Create SWC / SESOI band
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
  )

# Plot
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 = 0.2, height = 0.25) +
  geom_errorbarh(data = model_SESOI, aes(xmin = lower, xmax = upper), height = 0.1) +
  geom_point(data = model_SESOI, aes(x = bias)) +
  facet_wrap(~distance, scales = "free_x") +
  xlab("Predicted time - observed time (sec)") +
  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"))

df$model <- factor(
  df$model,
  levels = rev(c(
    "No corrections",
    "Fixed correction",
    "Time correction fixed",
    "Time correction random",
    "Time and distance correction fixed",
    "Time and distance correction random"
  ))
)

df$estimator <- factor(
  df$estimator,
  levels = c("bias", "variance", "MAD")
)

ggplot(df, aes(x = value, y = model)) +
  theme_bw() +
  geom_point() +
  facet_grid(distance ~ estimator, scales = "free_x") +
  xlab(NULL) +
  ylab(NULL)
```



Which model should 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 hard to interpret. 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 inconsistencies and/or ~~cheating at the sprint starts.~~

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 ~~a~~ measuring protocols that are sensitive enough to capture training intervention changes, but also robust enough to take into account potential sprint initiation inconsistencies.

References

Arsac, Laurent M., and Elio Locatelli. 2002. "Modeling the Energetics of 100-M Running by Using Speed Curves of World Champions." *Journal of Applied Physiology* 92 (5): 1781–8. https://doi.org/10.1152/japplphysiol.00754.2001.

Chelly, Souhaiel M., and Christian Denis. 2001. "Leg Power and Hopping Stiffness: Relationship with Sprint Running Performance:" *Medicine and Science in Sports and Exercise*, February, 326–33. https://doi.org/10.1097/00005768-200102000-00024.

Clark, Kenneth P., Randall H. Rieger, Richard F. Bruno, and David J. Stearne. 2017. "The NFL Combine 40-Yard Dash: How Important Is Maximum Velocity?" *Journal of Strength and Conditioning Research*, June, 1. https://doi.org/10.1519/JSC.0000000000002081.

Furusawa, K., Archibald Vivian Hill, and J. L. Parkinson. 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. https://doi.org/10.1098/rspb.1927.0035.

Goerg, Georg M. 2020. *LambertW: Probabilistic Models to Analyze and Gaussianize Heavy-Tailed, Skewed Data*. https://CRAN.R-project.org/package=LambertW.

Greene, Peter R. 1986. "Predicting Sprint Dynamics from Maximum-Velocity Measurements." *Mathematical Biosciences* 80 (1): 1–18. https://doi.org/10.1016/0025-5564(86)90063-5.

Haugen, Thomas A., Felix Breitschädel, and Pierre Samozino. 2020. "Power-Force-Velocity Profiling of Sprinting Athletes: Methodological and Practical Considerations When Using Timing Gates." *Journal of Strength and Conditioning Research* 34 (6): 1769–73. https://doi.org/10.1519/JSC.0000000000002890.

Haugen, Thomas A., Felix Breitschädel, and Stephen Seiler. 2019. "Sprint Mechanical Variables in Elite Athletes: Are Force-Velocity Profiles Sport Specific or Individual?" Edited by Leonardo A. Peyré-Tartaruga. *PLOS ONE* 14 (7): e0215551. https://doi.org/10.1371/journal.pone.0215551.

———. 2020. "Sprint Mechanical Properties in Soccer Players According to Playing Standard, Position, Age and Sex." *Journal of Sports Sciences* 38 (9): 1070–6. https://doi.org/10.1080/02640414.2020.1741955.

Haugen, Thomas A, Espen Tønnessen, and Stephen K Seiler. 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): 473–79. https://doi.org/10.1519/JSC.0b013e318226030b.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2017. *An Introduction to Statistical Learning: With Applications in R*. 1st ed. 2013, Corr. 7th printing 2017 edition. New York: Springer.

Jovanović, Mladen. 2020. *Bmbstats: Bootstrap Magnitude-Based Statistics for Sports Scientists*. Mladen Jovanović.

Kuhn, Max, and Kjell Johnson. 2018. *Applied Predictive Modeling*. 1st ed. 2013, Corr. 2nd printing 2016 edition. New York: Springer.

Moir, Gavin, Chris Button, Mark Glaister, and Michael H. Stone. 2004. "Influence of Familiarization on the Reliability of Vertical Jump and Acceleration Sprinting Performance in Physically Active Men." *The Journal of Strength and Conditioning Research* 18 (2): 276. https://doi.org/10.1519/R-13093.1.

Morin, Jean-Benoit, Pierre Samozino, Munenori Murata, Matt R Cross, and Ryu Nagahara. 2019. "A Simple Method for Computing Sprint Acceleration Kinetics from Running Velocity Data: Replication Study with Improved Design." *Journal of Biomechanics* 94 (September): 82–87. https://doi.org/10.1016/j.jbiomech.2019.07.020.

Morin, Jean-Benoît, and Pierre Samozino. 2016. "Interpreting Power-Force-Velocity Profiles for Individualized and Specific Training." *International Journal of Sports Physiology and Performance* 11 (2): 267–72. https://doi.org/10.1123/ijspp.2015-0638.

Pinheiro, José, Douglas Bates, and R-core. 2020. *Nlme: Linear and Nonlinear Mixed Effects Models*. https://svn.r-project.org/R-packages/trunk/nlme/.

R Core Team. 2020. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Samozino, P., G. Rabita, S. Dorel, J. Slawinski, N. Peyrot, E. Saez de Villarreal, and J.-B. Morin. 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): 648–58. https://doi.org/10.1111/sms.12490.

van Ingen Schenau, Gerrit Jan, Ron Jacobs, and Jos J. de Koning. 1991. "Can Cycle Power Predict Sprint Running Performance?" *European Journal of Applied Physiology and Occupational Physiology* 63 (3-4): 255–60. https://doi.org/10.1007/BF00233857.

Vescovi, Jason D. 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): 474–78. https://doi.org/10.1016/j.jsams.2012.03.006.

———. 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): 621–26. https://doi.org/10.1123/ijspp.2013-0263.

———. 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. https://doi.org/10.1080/02701367.2015.1124972.

Wickham, Hadley. 2019. *Tidyverse: Easily Install and Load the Tidyverse.* https://CRAN.R-project.org/package=tidyverse.

———. 2020. *Tidyr: Tidy Messy Data.* https://CRAN.R-project.org/package=tidyr.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2020. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics.* https://CRAN.R-project.org/package=ggplot2.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2020. *Dplyr: A Grammar of Data Manipulation.* https://CRAN.R-project.org/package=dplyr.