Package ‘shorts’

March 13, 2023

Type Package
Title Short Sprints
Version 2.4.0
Description Create short sprint (<6sec) profiles using the split times or the radar gun data.
Mono-exponential equation is used to estimate maximal sprinting speed (MSS), relative acceleration (TAU),
and other parameters such us maximal acceleration (MAC) and maximal relative power (PMAX). These parameters
can be used to predict kinematic and kinetics variables and to compare individuals. The modeling method utilized
in this package is based on the works of Chelly SM, Denis C. (2001) <doi:10.1097/00005768-200102000-00024>,
Clark KP, Rieger RH, Bruno RF, Stearne DJ. (2017) <doi:10.1519/JSC.0000000000002081>,
Furusawa K, Hill AV, Parkinson JL (1927) <doi:10.1098/rspb.1927.0035>,

URL https://mladenjovanovic.github.io/shorts/

BugReports https://github.com/mladenjovanovic/shorts/issues
License MIT + file LICENSE
Encoding UTF-8
LazyData true
RoxygenNote 7.2.2
Depends R (>= 2.10)
Imports stats, LambertW, tidyr, ggplot2, minpack.lm, purrr
NeedsCompilation no
Author Mladen Jovanović [aut, cre],
Jason D. Vescovi [dtc]
Maintainer Mladen Jovanović <coach.mladen.jovanovic@gmail.com>
Repository CRAN
Date/Publication 2023-03-13 10:20:02 UTC
R topics documented:

- coef.shorts_model
- create_timing_gates_splits
- find_functions
- find_optimal_distance
- fitted.shorts_model
- format_splits
- get_air_resistance
- jb_morin
- make_FV_profile
- model_radar_gun
- model_tether
- model_timing_gates
- optimal_functions
- plot.shorts_fv_profile
- plot.shorts_model
- predict.shorts_model
- predict_kinematics
- print.shorts_fv_profile
- print.shorts_model
- probe_functions
- radar_gun_data
- residuals.shorts_model
- split_times
- summary.shorts_model
- vescovi

Index

- coef.shorts_model

---

**Description**

S3 method for extracting model parameters from `shorts_model` object

**Usage**

```r
## S3 method for class 'shorts_model'
coef(object, ...)
```

**Arguments**

- `object` shorts_model object
- `...` Extra arguments. Not used
create_timing_gates_splits

Create Timing Gates Splits

Description

This function is used to generate timing gates splits with predetermined parameters

Usage

create_timing_gates_splits(
  MSS,
  MAC,
  gates = c(5, 10, 20, 30, 40),
  FD = 0,
  TC = 0,
  noise = 0
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSS, MAC</td>
<td>Numeric vectors. Model parameters</td>
</tr>
<tr>
<td>gates</td>
<td>Numeric vectors. Distances of the timing gates</td>
</tr>
<tr>
<td>FD</td>
<td>Numeric vector. Flying start distance. Default is 0</td>
</tr>
<tr>
<td>TC</td>
<td>Numeric vector. Time-correction added to split times (e.g., reaction time). Default is 0</td>
</tr>
<tr>
<td>noise</td>
<td>Numeric vector. SD of Gaussian noise added to the split times. Default is 0</td>
</tr>
</tbody>
</table>

Examples

```r
split_distances <- c(10, 20, 30, 40, 50)
split_times <- create_timing_gates_splits(
  gates = split_distances,
  MSS = 10,
  MAC = 9,
  FD = 0.25,
  TC = 0
)

# Simple model
simple_model <- model_timing_gates(split_distances, split_times)
coef(simple_model)
```
find_functions

Examples

```r
create_timing_gates_splits(
  gates = c(10, 20, 30, 40, 50),
  MSS = 10,
  MAC = 9,
  FD = 0.5,
  TC = 0
)
```

Description

Family of functions that serve a purpose of finding maximal value and critical distances and times at which power, acceleration or velocity drops below certain threshold.

- `find_max_power_distance` finds maximum power and distance at which max power occurs
- `find_max_power_time` finds maximum power and time at which max power occurs
- `find_velocity_critical_distance` finds critical distance at which percent of MSS is achieved
- `find_velocity_critical_time` finds critical time at which percent of MSS is achieved
- `find_acceleration_critical_distance` finds critical distance at which percent of MAC is reached
- `find_acceleration_critical_time` finds critical time at which percent of MAC is reached
- `find_power_critical_distance` finds critical distances at which maximal power over percent is achieved
- `find_power_critical_time` finds critical times at which maximal power over percent is achieved

Usage

```r
find_max_power_distance(MSS, MAC, ...)
find_max_power_time(MSS, MAC, ...)
find_velocity_critical_distance(MSS, MAC, percent = 0.9)
find_velocity_critical_time(MSS, MAC, percent = 0.9)
find_acceleration_critical_distance(MSS, MAC, percent = 0.9)
find_acceleration_critical_time(MSS, MAC, percent = 0.9)
find_power_critical_distance(MSS, MAC, percent = 0.9, ...)
find_power_critical_time(MSS, MAC, percent = 0.9, ...)
```
find_functions

Arguments

MSS, MAC  Numeric vectors. Model parameters
...
percent  Numeric vector. Used to calculate critical distance. Default is 0.9

Value

find_max_power_distance returns list with two elements: max_power and distance at which max power occurs
find_max_power_time returns list with two elements: max_power and time at which max power occurs

References


Examples

dist <- seq(0, 40, length.out = 1000)
velocity <- predict_velocity_at_distance(
  distance = dist,
  MSS = 10,
  MAC = 9
)
acceleration <- predict_acceleration_at_distance(
  distance = dist,
  MSS = 10,
  MAC = 9
)
# Use ... to forward parameters to the shorts::get_air_resistance
pwr <- predict_relative_power_at_distance(
  distance = dist,
  MSS = 10,
  MAC = 9
  # bodyweight = 100,
  # bodyheight = 1.9,
  # barometric_pressure = 760,
  # air_temperature = 25,
  # wind_velocity = 0
find_optimal_distance

Function that finds the distance at which the sprint, probe, or FV profile is optimal (i.e., equal to 100 perc)

Description

Function that finds the distance at which the sprint, probe, or FV profile is optimal (i.e., equal to 100 perc)

Usage

find_optimal_distance(..., optimal_func = optimal_FV, min = 1, max = 60)

Arguments

... Forwarded to selected optimal_func
optimal_func Selected profile optimization function. Default is optimal_FV
min, max Distance over which to find optimal profile distance
fitted.shorts_model

Value

Distance

Examples

MSS <- 10
MAC <- 8
bodymass <- 75

fv <- make_FV_profile(MSS, MAC, bodymass)

find_optimal_distance(
  F0 = fv$F0_poly,
  V0 = fv$V0_poly,
  bodymass = fv$bodymass,
  optimal_func = optimal_FV,
  method = "max"
)

find_optimal_distance(
  MSS = MSS,
  MAC = MAC,
  optimal_func = optimal_MSS_MAC
)

find_optimal_distance(
  MSS = MSS,
  MAC = MAC,
  optimal_func = probe_MSS_MAC
)

fitted.shorts_model  S3 method for returning predictions of shorts_model

Description

S3 method for returning predictions of shorts_model

Usage

## S3 method for class 'shorts_model'
fitted(object, ...)

Arguments

  object          shorts_model object
  ...             Extra arguments. Not used
Examples

```r
split_distances <- c(10, 20, 30, 40, 50)
split_times <- create_timing_gates_splits(
  gates = split_distances,
  MSS = 10,
  MAC = 9,
  FD = 0.25,
  TC = 0
)

# Simple model
simple_model <- model_timing_gates(split_distances, split_times)
fitted(simple_model)
```

---

### format_splits

**Format Split Data**

Function formats split data and calculates split distances, split times and average split velocity

**Usage**

```r
format_splits(distance, time)
```

**Arguments**

- `distance` Numeric vector
- `time` Numeric vector

**Value**

Data frame with the following columns:

- `split` Split number
- `split_distance_start` Distance at which split starts
- `split_distance_stop` Distance at which split ends
- `split_distance` Split distance
- `split_time_start` Time at which distance starts
- `split_time_stop` Time at which distance ends
- `split_time` Split time
- `split_mean_velocity` Mean velocity over split distance
- `split_mean_acceleration` Mean acceleration over split distance
get_air_resistance

Examples

data("split_times")

john_data <- split_times[split_times$athlete == "John", ]

format_splits(john_data$distance, john_data$time)

get_air_resistance

Get Air Resistance

Description

get_air_resistance estimates air resistance in Newtons

Usage

get_air_resistance(
  velocity,
  bodymass = 75,
  bodyheight = 1.75,
  barometric_pressure = 760,
  air_temperature = 25,
  wind_velocity = 0
)

Arguments

velocity Instantaneous running velocity in meters per second (m/s)
bodymass In kilograms (kg)
bodyheight In meters (m)
barometric_pressure In Torrs
air_temperature In Celzius (C)
wind_velocity In meters per second (m/s). Use negative number as head wind, and positive number as back wind

Value

Air resistance in Newtons (N)
References


Examples

g Moffin(  
    velocity = 5,  
    bodymass = 80,  
    bodyheight = 1.90,  
    barometric_pressure = 760,  
    air_temperature = 16,  
    wind_velocity = -0.5  
)

jb_morin

---

**jb_morin**  
**JB Morin Sample Dataset**

Description

Sample radar gun data provided by Jean-Benoît Morin on his website. See https://jbmorin.net/2017/12/13/a-spreadsheet-for-sprint-acceleration-force-velocity-power-profiling/ for more details.

Usage

data(jb_morin)

Format

Data frame with 2 variables and 232 observations:

- **time**  Time in seconds
- **velocity**  Velocity in m/s
make_FV_profile

Details

This dataset represents a sample data provided by Jean-Benoît Morin on a single individual running approximately 35m from a stand still position that is measured with the radar gun. Individual’s body mass is 75kg, height is 1.72m. Conditions of the run are the following: air temperature 25C, barometric pressure 760mmHg, wind velocity 0m/s.

The purpose of including this dataset in the package is to check the agreement of the model estimates with Jean-Benoît Morin Microsoft Excel spreadsheet.

Author(s)

Jean-Benoît Morin
Inter-university Laboratory of Human Movement Biology
Saint-Étienne, France https://jbmorin.net/

References


make_FV_profile

Get Force-Velocity Profile

Description


Usage

```r
make_FV_profile(
  MSS,
  MAC,
  bodymass = 75,
  max_time = 6,
  frequency = 100,
  RFmax_cutoff = 0.3,
  ...
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSS, MAC</td>
<td>Numeric vectors. Model parameters</td>
</tr>
<tr>
<td>bodymass</td>
<td>Body mass in kg. Used to calculate relative power and forwarded to <code>get_air_resistance</code></td>
</tr>
<tr>
<td>max_time</td>
<td>Predict from 0 to max_time. Default is 6seconds</td>
</tr>
<tr>
<td>frequency</td>
<td>Number of samples within one second. Default is 100Hz</td>
</tr>
</tbody>
</table>
RFmax_cutoff  Time cut-off used to estimate RFmax and Drf. Default is 0.3s

...  Forwarded to get_air_resistance for the purpose of calculation of air resistance and power

Value

List containing the following elements:

- **bodymass**  Returned bodymass used in FV profiling
- **F0**  Horizontal force when velocity=0
- **F0_rel**  $F_0$ divided by bodymass
- **V0**  Velocity when horizontal force=0
- **Pmax**  Maximal horizontal power
- **Pmax_rel**  $P_{max}$ divided by bodymass
- **FV_slope**  Slope of the FV profile. See References for more info
- **RFmax**  Maximal force ratio after 0.3sec. See References for more info
- **RFmax_cutoff**  Time cut-off used to estimate RFmax
- **Drf**  Slope of Force Ratio (RF) and velocity. See References for more info
- **RSE_FV**  Residual standard error of the FV profile.
- **RSE_Drf**  Residual standard error of the RF-velocity profile
- **F0_poly**  Horizontal force when velocity=0, estimated using the analytics/polynomial method
- **F0_poly_rel**  $F_0_{poly}$ divided by bodymass
- **V0_poly**  Velocity when horizontal force=0, estimated using the analytics/polynomial method
- **Pmax_poly**  Maximal horizontal power, estimated using the analytics/polynomial method
- **Pmax_poly_rel**  $P_{max}_{poly}$ divided by bodymass
- **FV_slope_poly**  Slope of the FV profile, estimated using the analytics/polynomial method. See References for more info

**data**  Data frame containing simulated data used to estimate parameters

References


Examples

```r
data("jb_morin")

m1 <- model_radar_gun(time = jb_morin$time, velocity = jb_morin$velocity)

ev_profile <- make_FV_profile(
  MSS = m1$parameters$MSS,
  MAC = m1$parameters$MAC,
  bodyheight = 1.72,
  bodymass = 120
)

print(ev_profile)
plot(ev_profile)
plot(ev_profile, "time")
```

---

**model_radar_gun**  
*Model Using Instantaneous Velocity or Radar Gun*

**Description**

This function models the sprint instantaneous velocity using mono-exponential equation that estimates maximum sprinting speed (MSS) and relative acceleration (TAU). velocity is used as target or outcome variable, and time as predictor.

**Usage**

```r
model_radar_gun(
  time,  
  velocity,  
  weights = 1,  
  CV = NULL,  
  use_observed_MSS = FALSE,  
  control = minpack.lm::nls.lm.control(maxiter = 1000),  
  na.rm = FALSE,  
  ...  
)
```

**Arguments**

- **time**: Numeric vector
- **velocity**: Numeric vector
- **weights**: Numeric vector. Default is 1
- **CV**: Should cross-validation be used to estimate model fit? Default is NULL. Otherwise use integer indicating number of folds. See Example for more information
- **use_observed_MSS**: Should MSS be estimated from the observed velocity? Default is FALSE
model_tether

Control object forwarded to \texttt{nlsLM}. Default is \texttt{minpack.lm::nls.lm.control(maxiter = 1000)}

\texttt{na.rm} Logical. Default is \texttt{FALSE}

... Forwarded to \texttt{nlsLM} function

\textbf{Value}

List object with the following elements:

\begin{itemize}
  \item \textbf{parameters} List with the following estimated parameters: MSS, TAU, MAC, PMAX, and TC
  \item \textbf{model_fit} List with the following components: RSE, R\textsubscript{\textit{\textunderscore squared}}, minErr, maxErr, and RMSE
  \item \textbf{model} Model returned by the \texttt{nlsLM} function
  \item \textbf{data} Data frame used to estimate the sprint parameters, consisting of \textit{time}, \textit{velocity}, \textit{weights}, and \textit{pred\_velocity} columns
\end{itemize}

\textbf{References}


\textbf{Examples}

\begin{verbatim}
instant_velocity <- data.frame(
  time = c(0, 1, 2, 3, 4, 5, 6),
  velocity = c(0.00, 4.99, 6.43, 6.84, 6.95, 6.99, 7.00)
)

sprint_model <- with(
  instant_velocity,
  model_radar_gun(time, velocity)
)

print(sprint_model)
coef(sprint_model)
plot(sprint_model)
\end{verbatim}

\textbf{Description}

This function models the sprint instantaneous velocity using mono-exponential equation that estimates maximum sprinting speed (MSS) and relative acceleration (TAU). \textit{Velocity} is used as target or outcome variable, and \textit{distance} as predictor.
**model_tether**

**Usage**

```r
model_tether(
  distance,
  velocity,
  weights = 1,
  CV = NULL,
  use_observed_MSS = FALSE,
  control = minpack.lm::nls.lm.control(maxiter = 1000),
  na.rm = FALSE,
  ...
)
```

**Arguments**

- `distance`  Numeric vector
- `velocity`  Numeric vector
- `weights`  Numeric vector. Default is 1
- `CV`  Should cross-validation be used to estimate model fit? Default is NULL. Otherwise use integer indicating number of folds. See Example for more information
- `use_observed_MSS`  Should MSS be estimated from the observed velocity? Default is FALSE
- `control`  Control object forwarded to `nlsLM`. Default is `minpack.lm::nls.lm.control(maxiter = 1000)`
- `na.rm`  Logical. Default is FALSE
- `...`  Forwarded to `nlsLM` function

**Value**

List object with the following elements:

- **parameters**  List with the following estimated parameters: MSS, TAU, MAC, and PMAX
- **model_fit**  List with the following components: RSE, R_squared, minErr, maxErr, and RMSE
- **model**  Model returned by the `nlsLM` function
- **data**  Data frame used to estimate the sprint parameters, consisting of distance, velocity, weights, and pred_velocity columns

**Examples**

```r
distance <- c(5, 10, 20, 30, 40)
velocity <- predict_velocity_at_distance(distance, MSS = 10, MAC = 8)
m1 <- model_tether(distance = distance, velocity = velocity)
m1
plot(m1)
```
**model_timing_gates**  
*Models Using Timing Gates Split Times*

**Description**

These functions model the sprint split times using mono-exponential equation, where time is used as target or outcome variable, and distance as predictor.

- **model_timing_gates** Provides the simplest model with estimated MSS and MAC parameters
- **model_timing_gates_TC** Besides estimating MSS and MAC parameters, this function estimates additional parameter TC or time correction
- **model_timing_gates_FD** In addition to estimating MSS and MAC parameters, this function estimates FD or flying distance
- **model_timing_gates_FD_TC** Combines the approach of the model_timing_gates_FD with that one of model_timing_gates_TC. In other words, it add extra parameter TC to be estimated in the model_timing_gates_FD model

**Usage**

```r
model_timing_gates(  
  distance,  
  time,  
  weights = 1,  
  LOOCV = FALSE,  
  control = minpack.lm::nls.lm.control(maxiter = 1000),  
  na.rm = FALSE,  
  ...
)
```

```r
model_timing_gates_TC(  
  distance,  
  time,  
  weights = 1,  
  LOOCV = FALSE,  
  control = minpack.lm::nls.lm.control(maxiter = 1000),  
  na.rm = FALSE,  
  ...
)
```

```r
model_timing_gates_FD(  
  distance,  
  time,  
  weights = 1,  
  FD = NULL,  
  LOOCV = FALSE,  
  control = minpack.lm::nls.lm.control(maxiter = 1000),  
  ...
)
```
model_timing_gates

```r
na.rm = FALSE,
...
)
```

```r
model_timing_gates_FD_TC(
  distance,
  time,
  weights = 1,
  FD = NULL,
  LOOCV = FALSE,
  control = minpack.lm::nls.lm.control(maxiter = 1000),
  na.rm = FALSE,
  ...
)
```

### Arguments

- **distance, time**: Numeric vector. Indicates the position of the timing gates and time measured
- **weights**: Numeric vector. Default is vector of 1. This is used to give more weight to particular observations. For example, use `1/distance` to give more weight to observations from shorter distances.
- **LOOCV**: Should Leave-one-out cross-validation be used to estimate model fit? Default is FALSE
- **control**: Control object forwarded to `nlsLM`. Default is `minpack.lm::nls.lm.control(maxiter = 1000)`
- **na.rm**: Logical. Default is FALSE
- **...**: Extra parameters forwarded to `nlsLM` function
- **FD**: Use this parameter if you do not want the `FD` parameter to be estimated, but rather take the provided value

### Value

List object with the following elements:

- **data**: Data frame used to estimate the sprint parameters, consisting of `distance`, `time`, `weights`, and `pred_time` columns
- **model**: Model returned by the `nlsLM` function
- **parameters**: List with the estimated parameters, of which the following are always returned: MSS, TAU, MAC, and PMAX
- **model_fit**: List with the following components: RSE, R_squared, minErr, maxErr, and RMSE

### References


optimal_functions

Examples

```r
split_distances <- c(10, 20, 30, 40, 50)
split_times <- create_timing_gates_splits(
  gates = split_distances,
  MSS = 10,
  MAC = 9,
  FD = 0.25,
  TC = 0
)

# Simple model
simple_model <- model_timing_gates(split_distances, split_times)

print(simple_model)
coef(simple_model)
plot(simple_model)

# Model with correction of 0.3s
model_with_correction <- model_timing_gates(split_distances, split_times + 0.3)

print(model_with_correction)
plot(model_with_correction)

# Model with time_correction estimation
model_with_TC <- model_timing_gates_TC(split_distances, split_times)

print(model_with_TC)
plot(model_with_TC)

# Model with flying distance estimations
model_with_FD <- model_timing_gates_FD(split_distances, split_times)

print(model_with_FD)
plot(model_with_FD)

# Model with flying distance estimations and time correction
model_with_FD_TC <- model_timing_gates_FD_TC(split_distances, split_times)

print(model_with_FD_TC)
plot(model_with_FD_TC)
```

optimal_functions

Optimal profile functions

Description

Family of functions that serve a purpose of finding optimal sprint or force-velocity profile

optimal_FV finds "optimal" F0 and V0 where time at distance is minimized, while keeping the power the same
optimal_MSS_MAC finds "optimal" MSS and MAS where time at distance is minimized, while keeping the Pmax the same

Usage

optimal_FV(distance, F0, V0, bodymass = 75, method = "max", ...)

optimal_MSS_MAC(distance, MSS, MAC)

Arguments

distance Numeric vector
F0, V0 Numeric vectors. FV profile parameters
bodymass Body mass in kg
method Method to be utilized. Options are "peak" and "max" (default)
... Forwarded to predict_power_at_distance for the purpose of calculation of air resistance
MSS, MAC Numeric vectors. Model parameters

Value

optimal_FV returns a data frame with the following columns

F0 Original F0
V0 Original V0
bodymass Bodymass
Pmax Maximal power estimated using F0 * V0 / 4
Pmax_rel Relative maximal power
slope FV profile slope
distance Distance
time Time to cover distance
Ppeak Peak power estimated quantitatively
Ppeak_rel Relative peak power
Ppeak_dist Distance at which peak power is manifested
Ppeak_time Time at which peak power is manifested
F0_optim Optimal F0
F0_coef Ratio between F0_optim an F0
V0_optim Optimal V0
V0_coef Ratio between V0_optim an V0
Pmax_optim Optimal maximal power estimated F0_optim * V0_optim / 4
Pmax_rel_optim Optimal relative maximal power
slope_optim Optimal FV profile slope
profile_imb  Percent ratio between slope and optimal slope

time_optim  Time to cover distance when profile is optimal

time_gain  Difference in time to cover distance between time_optimal and time

Ppeak_optim  Optimal peak power estimated quantitatively

Ppeak_rel_optim  Optimal relative peak power

Ppeak_dist_optim  Distance at which optimal peak power is manifested

Ppeak_time_optim  Time at which optimal peak power is manifested

columns

MSS  Original MSS
MAC  Original MAC
Pmax_rel  Relative maximal power estimated using MSS * MAC / 4
slope  Sprint profile slope
distance  Distance
time  Time to cover distance
MSS_optim  Optimal MSS
MSS_coef  Ratio between MSS_optim an MSS
MAC_optim  Optimal MAC
MAC_coef  Ratio between MAC_optim an MAC
Pmax_rel_optim  Optimal relative maximal power estimated using MSS_optim * MAC_optim / 4
slope_optim  Optimal sprint profile slope
profile_imb  Percent ratio between slope and optimal slope
time_optim  Time to cover distance when profile is optimal
time_gain  Difference in time to cover distance between time_optimal and time

References


Examples

MSS <- 10
MAC <- 8
bodymass <- 75

fv <- make_FV_profile(MSS, MAC, bodymass)

dist <- seq(5, 40, by = 5)

opt_MSS_MAC_profile <- optimal_MSS_MAC(
  distance = dist,
plot.shorts_fv_profile

S3 method for plotting shorts_fv_profile object

Description

S3 method for plotting shorts_fv_profile object

Usage

## S3 method for class 'shorts_fv_profile'
plot(x, type = "velocity", ...)

Arguments

x            shorts_fv_profile object

type         Type of plot. Options are "velocity" (default) and "time"

...          Not used

Value

ggplot object
Examples

```r
data("jb_morin")

m1 <- model_radar_gun(time = jb_morin$time, velocity = jb_morin$velocity)

fv_profile <- make_FV_profile(
  MSS = m1$parameters$MSS,
  MAC = m1$parameters$MAC,
  bodyheight = 1.72,
  bodymass = 120
)

plot(fv_profile)
plot(fv_profile, "time")
```

Description

S3 method for plotting `shorts_model` object

Usage

```r
# S3 method for class 'shorts_model'
plot(x, type = NULL, …)
```

Arguments

- `x` : `shorts_model` object
- `type` : Not used
- `…` : Not used

Value

`ggplot` object

Examples

```r
split_times <- data.frame(
  distance = c(5, 10, 20, 30, 35),
  time = c(1.20, 1.96, 3.36, 4.71, 5.35)
)

# Simple model with time splits
simple_model <- with(
  split_times, 
  model_timing_gates(distance, time)
)```
predict.shorts_model

)

coef(simple_model)
plot(simple_model)

# Simple model with radar gun data
instant_velocity <- data.frame(  
time = c(0, 1, 2, 3, 4, 5, 6),
velocity = c(0.00, 4.99, 6.43, 6.84, 6.95, 6.99, 7.00)
)

radar_model <- with(  
instant_velocity,
model_radar_gun(time, velocity)
)

# sprint_model$parameters
coef(radar_model)
plot(radar_model)

predict.shorts_model  S3 method for making predictions using shorts_model

Description

S3 method for making predictions using shorts_model

Usage

## S3 method for class 'shorts_model'
predict(object, ...)

Arguments

object           shorts_model object
...
Forwarded to generic predict() function

Examples

split_distances <- c(10, 20, 30, 40, 50)
split_times <- create_timing_gates_splits(  
gates = split_distances,
MSS = 10,
MAC = 9,
FD = 0.25,
TC = 0
)

# Simple model
simple_model <- model_timing_gates(split_distances, split_times)
predict(simple_model)
predict_kinematics  

Kinematics prediction functions

Description

Predicts kinematics from known MSS and MAC parameters

Usage

```
predict_velocity_at_time(time, MSS, MAC)
predict_distance_at_time(time, MSS, MAC)
predict_acceleration_at_time(time, MSS, MAC)
predict_time_at_distance(distance, MSS, MAC)
predict_time_at_distance_FV(distance, F0, V0, bodymass = 75, ...)
predict_velocity_at_distance(distance, MSS, MAC)
predict_acceleration_at_distance(distance, MSS, MAC)
predict_acceleration_at_velocity(velocity, MSS, MAC)
predict_air_resistance_at_time(time, MSS, MAC, ...)
predict_air_resistance_at_distance(distance, MSS, MAC, ...)
predict_force_at_time(time, MSS, MAC, bodymass = 75, ...)
predict_force_at_distance(distance, MSS, MAC, bodymass = 75, ...)
predict_power_at_distance(distance, MSS, MAC, bodymass = 75, ...)
predict_power_at_time(time, MSS, MAC, bodymass = 75, ...)
predict_relative_power_at_distance(distance, MSS, MAC, bodymass = 75, ...)
predict_relative_power_at_time(time, MSS, MAC, bodymass = 75, ...)
predict_kinematics(object, max_time = 6, frequency = 100, bodymass = 75, ...)
```

Arguments

- `time, distance, velocity`  
  Numeric vectors
predict_kinematics

MSS, MAC  Numeric vectors. Model parameters
F0, V0     Numeric vectors. FV profile parameters
bodymass   Body mass in kg. Used to calculate relative power and forwarded to get_air_resistance

...       Forwarded to get_air_resistance for the purpose of calculation of air resistance and power
object     shorts_model object
max_time   Predict from 0 to max_time. Default is 6seconds
frequency  Number of samples within one second. Default is 100Hz

Value

Numeric vector

Data frame with kinetic and kinematic variables

References


Examples

MSS <- 8
MAC <- 9

time_seq <- seq(0, 6, length.out = 10)

df <- data.frame(
  time = time_seq,
  distance_at_time = predict_distance_at_time(time_seq, MSS, MAC),
  velocity_at_time = predict_velocity_at_time(time_seq, MSS, MAC),
  acceleration_at_time = predict_acceleration_at_time(time_seq, MSS, MAC)
)

df$time_at_distance <- predict_time_at_distance(df$distance_at_time, MSS, MAC)
df$velocity_at_distance <- predict_velocity_at_distance(df$distance_at_time, MSS, MAC)
df$acceleration_at_distance <- predict_acceleration_at_distance(df$distance_at_time, MSS, MAC)
df$acceleration_at_velocity <- predict_acceleration_at_velocity(df$velocity_at_time, MSS, MAC)

# Power calculation uses shorts::get_air_resistance function and its defaults
# values to calculate power. Use the ... to setup your own parameters for power calculations
df$power_at_time <- predict_power_at_time(
  time = df$time, MSS = MSS, MAC = MAC,
  # Check shorts::get_air_resistance for available params
  bodymass = 100, bodyheight = 1.85
)

df

# Example for predict_kinematics
split_times <- data.frame(
  distance = c(5, 10, 20, 30, 35),
  time = c(1.20, 1.96, 3.36, 4.71, 5.35)
)

# Simple model
simple_model <- with(
  split_times,
  model_timing_gates(distance, time)
)

predict_kinematics(simple_model)

print.shorts_fv_profile
S3 method for printing shorts_fv_profile object

Description
S3 method for printing shorts_fv_profile object

Usage
## S3 method for class 'shorts_fv_profile'
print(x, ...)

Arguments
x shorts_fv_profile object
...
Not used

Examples
data("jb_morin")
m1 <- model_radar_gun(time = jb_morin$time, velocity = jb_morin$velocity)
fv_profile <- make_FV_profile(
  MSS = m1$parameters$MSS,
  MAC = m1$parameters$MAC,
print.shorts_model

```r
bodyheight = 1.72,
bodymass = 120
)

print(fv_profile)
```

print.shorts_model  

---

print.shorts_model  

### S3 method for printing shorts_model object

**Description**

S3 method for printing shorts_model object

**Usage**

```r
## S3 method for class 'shorts_model'
print(x, ...)
```

**Arguments**

- `x`  
  shorts_model object  

- `...`  
  Not used

**Examples**

```r
split_distances <- c(10, 20, 30, 40, 50)
split_times <- create_timing_gates_splits(
  gates = split_distances,
  MSS = 10,
  MAC = 9,
  FD = 0.25,
  TC = 0
)

# Simple model
simple_model <- model_timing_gates(split_distances, split_times)
simple_model
```
probe_functions  

**Probe profile functions**

**Description**

Family of functions that serve a purpose of probing sprint or force-velocity profile. This is done by increasing individual sprint parameter for a percentage and calculating which parameter improvement yield biggest deduction in sprint time.

`probe_FV` "probes" F0 and V0 and calculates which one improves sprint time for a defined distance.

`probe_MSS_MAC` "probes" MSS and MAC and calculates which one improves sprint time for a defined distance.

**Usage**

`probe_FV(distance, F0, V0, bodymass = 75, perc = 2.5, ...)`

`probe_MSS_MAC(distance, MSS, MAC, perc = 2.5)`

**Arguments**

- **distance**: Numeric vector
- **F0, V0**: Numeric vectors. FV profile parameters
- **bodymass**: Body mass in kg
- **perc**: Numeric vector. Probing percentage. Default is 2.5 percent
- **...**: Forwarded to `predict_power_at_distance` for the purpose of calculation of air resistance
- **MSS, MAC**: Numeric vectors. Model parameters

**Value**

`probe_FV` returns a data frame with the following columns:

- **F0**: Original F0
- **V0**: Original F0
- **bodymass**: Bodymass
- **Pmax**: Maximal power estimated using F0 * V0 / 4
- **Pmax_rel**: Relative maximal power
- **slope**: FV profile slope
- **distance**: Distance
- **time**: Time to cover distance
- **probe_perc**: Probe percentage
- **F0_probe**: Probing F0
**probe_functions**

**F0_probe_time** Predicted time for distance when F0 is probed

**F0_probe_time_gain** Difference in time to cover distance between time_optimal and time

**V0_probe** Probing V0

**V0_probe_time** Predicted time for distance when V0 is probed

**V0_probe_time_gain** Difference in time to cover distance between time_optimal and time

**profile_imb** Percent ratio between V0_probe_time_gain and F0_probe_time_gain

`probe_MSS_MAC` returns a data frame with the following columns:

- **MSS** Original MSS
- **MAC** Original MAC
- **Pmax_rel** Relative maximal power estimated using MSS * MAC / 4
- **slope** Sprint profile slope
- **distance** Distance
- **time** Time to cover distance
- **probe_perc** Probe percentage
- **MSS_probe** Probing MSS

**MSS_probe_time** Predicted time for distance when MSS is probed

**MSS_probe_time_gain** Difference in time to cover distance between probe time and time

**MAC_probe** Probing MAC

**MAC_probe_time** Predicted time for distance when MAC is probed

**MAC_probe_time_gain** Difference in time to cover distance between probing time and time

**profile_imb** Percent ratio between MSS_probe_time_gain and MAC_probe_time_gain

### Examples

```r
MSS <- 10
MAC <- 8
bodymass <- 75

fv <- make_FV_profile(MSS, MAC, bodymass)

dist <- seq(5, 40, by = 5)

probe_MSS_MAC_profile <- probe_MSS_MAC(
    distance = dist,
    MSS, MAC)

probe_FV_profile <- probe_FV(
    distance = dist,
    fv$F0_poly, fv$V0_poly, fv$bodymass
)```
residuals.shorts_model

S3 method for providing residuals for the shorts_model object

Description

S3 method for providing residuals for the shorts_model object

Usage

## S3 method for class 'shorts_model'
residuals(object, ...)

Arguments

- object: shorts_model object
- ...: Not used
Examples

```r
split_distances <- c(10, 20, 30, 40, 50)
split_times <- create_timing_gates_splits(
  gates = split_distances,
  MSS = 10,
  MAC = 9,
  FD = 0.25,
  TC = 0
)

# Simple model
simple_model <- model_timing_gates(split_distances, split_times)
residuals(simple_model)
```

<table>
<thead>
<tr>
<th>split_times</th>
<th>Split Testing Data</th>
</tr>
</thead>
</table>

Description

Data generated from known MSS and TAU and measurement error for N=5 athletes using 6 timing gates: 5m, 10m, 15m, 20m, 30m, 40m

Usage

```r
data(split_times)
```

Format

Data frame with 4 variables and 30 observations:

- **athlete**: Character string
- **bodyweight**: Bodyweight in kilograms
- **distance**: Distance of the timing gates from the sprint start in meters
- **time**: Time reported by the timing gate

<table>
<thead>
<tr>
<th>summary.shorts_model</th>
<th>S3 method for providing summary for the shorts_model object</th>
</tr>
</thead>
</table>

Description

S3 method for providing summary for the shorts_model object

Usage

```r
## S3 method for class 'shorts_model'
summary(object, ...)
```
Arguments

object shorts_model object
... Not used

Examples

```r
split_distances <- c(10, 20, 30, 40, 50)
split_times <- create_timing_gates_splits(
  gates = split_distances,
  MSS = 10,
  MAC = 9,
  FD = 0.25,
  TC = 0
)

# Simple model
simple_model <- model_timing_gates(split_distances, split_times)
summary(simple_model)
```

---

**vescovi**  

**Vescovi Timing Gates Sprint Times**

**Description**

Timing gates sprint times involving 52 female athletes. Timing gates were located at 5m, 10m, 20m, 30m, and 35m. See **Details** for more information.

**Usage**

data(vescovi)

**Format**

Data frame with 17 variables and 52 observations:

**Team** Team or sport. Contains the following levels: 'W Soccer' (Women Soccer), 'FH Sr' (Field Hockey Seniors), 'FH U21' (Field Hockey Under 21), and 'FH U17' (Field Hockey Under 17)

**Surface** Type of testing surface. Contains the following levels: 'Hard Cours' and 'Natural Grass'

**Athlete** Athlete ID

**Age** Athlete age in years

**Height** Body height in cm

**Bodyweight** Body weight in kg

**BMI** Body Mass Index

**BSA** Body Surface Area. Calculated using Mosteller equation sqrt((height/weight)/3600)
5m Time in seconds at 5m gate
10m Time in seconds at 10m gate
20m Time in seconds at 20m gate
30m Time in seconds at 30m gate
35m Time in seconds at 35m gate
10m-5m split Split time in seconds between 10m and 5m gate
20m-10m split Split time in seconds between 20m and 10m gate
30m-20m split Split time in seconds between 30m and 20m gate
35m-30m split Split time in seconds between 35m and 30m gate

Details

This data-set represents 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 this data-set. Soccer players were older (24.6±3.6 vs. 18.9±2.7 yr, p < 0.001), 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, Button, Glaister, and Stone 2004).

Author(s)

Jason D. Vescovi
University of Toronto
Faculty of Kinesiology and Physical Education
Graduate School of Exercise Science
Toronto, ON Canada
<vescovij@gmail.com>

References


Index

* datasets
  - jb_morin, 10
  - radar_gun_data, 30
  - split_times, 31
  - vescovi, 32

coef.shorts_model, 2
create_timing_gates_splits, 3
find_acceleration_critical_distance (find_functions), 4
find_acceleration_critical_time (find_functions), 4
find_functions, 4
find_max_power_distance (find_functions), 4
find_max_power_time (find_functions), 4
find_optimal_distance, 6
find_power_critical_distance (find_functions), 4
find_power_critical_time (find_functions), 4
find_velocity_critical_distance (find_functions), 4
find_velocity_critical_time (find_functions), 4
fitted.shorts_model, 7
format_splits, 8
get_air_resistance, 9, 11, 12, 25
ggplot, 21, 22
jb_morin, 10
make_FV_profile, 11
model_radar_gun, 13
model_tether, 14
model_timing_gates, 16, 16
model_timing_gates_FD, 16
model_timing_gates_FD (model_timing_gates), 16

model_timing_gates_FD_TC, 16
model_timing_gates_FD_TC (model_timing_gates), 16
model_timing_gates_TC, 16
model_timing_gates_TC (model_timing_gates), 16

nlsLM, 14, 15, 17

optimal_functions, 18
optimal_FV, 6
optimal_FV (optimal_functions), 18
optimal_MSS_MAC (optimal_functions), 18

plot.shorts_fv_profile, 21
plot.shorts_model, 22
predict.shorts_model, 23
predict_acceleration_at_distance (predict_kinematics), 24
predict_acceleration_at_time (predict_kinematics), 24
predict_acceleration_at_velocity (predict_kinematics), 24
predict_air_resistance_at_distance (predict_kinematics), 24
predict_air_resistance_at_time (predict_kinematics), 24
predict_distance_at_time (predict_kinematics), 24
predict_force_at_distance (predict_kinematics), 24
predict_force_at_time (predict_kinematics), 24
predict_kinematics, 24
predict_power_at_distance, 5, 19, 28
predict_power_at_distance (predict_kinematics), 24
predict_power_at_time (predict_kinematics), 24