




JumpMetrics: A Python package computing countermovement and squat jump events and metrics

Steven Mark Hirsch^{1,2*} and Samuel Howarth^{2*}

¹ Tonal Data Science and AI, San Francisco, United States of America ² Canadian Memorial
Chiropractic College, Toronto, Canada * These authors contributed equally.

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: 

Submitted: 10 August 2025

Published: unpublished

License

Authors of papers retain copyright
and release the work under a
Creative Commons Attribution 4.0
International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

Researchers and practitioners (e.g., sports team scientists, analysts, or coaches) commonly assess countermovement jump and squat jump performance on force plates. A countermovement jump involves a vertical jump where the jumper first dips downwards before immediately jumping upwards as high as possible. In contrast, a squat jump is a vertical jump whereby the jumper pauses briefly in the bottom “squat” position after dipping downwards, and then jumps upwards to minimize contributions from the stretch-shortening cycle. A scientist, analyst, or coach can use data from these vertical jump variations for various reasons, such as to evaluate people’s neuromuscular capacity, injury risk, or readiness/preparedness for high-intensity training. For evaluating capacity, some researchers have examined variables such as maximum jump height, peak force, rate of force development, and impulse [mcmahon:2017]. When assessing injury risk, researchers have examined landing forces (e.g., [pedley2020]) or have leveraged statistical techniques on various countermovement jump variables simultaneously [bird:2022]. For training readiness, researchers have measured changes in vertical jump height between consecutive training sessions (e.g., [watkins:2017]). Examining the difference in performance between the two jump variations may also provide insights into the strengths and weaknesses of the athlete to direct future training [vanHooren2017]. For example, some researchers may compute an “eccentric utilization ratio” by comparing metrics from the countermovement jump relative to the squat jump and use that to inform whether someone should focus their training on improving their ability to leverage the stretch shortening cycle to maximize jumping performance [vanHooren2017]. Although it is common to collect this kinetic data from a force plate for these jump variations for several applications, there are currently no free, open-source resources to detect events and compute metrics for reproducible and accessible data processing. JumpMetrics fills this gap in both applied practice and in the sports science literature.

Statement of need

JumpMetrics is a free, open-source Python package for computing countermovement jump and squat jump events and metrics. Currently, there are no free alternatives for processing force plate data for vertical jumps relative to the numerous proprietary (i.e., closed-source) algorithms sold by commercial force plates companies (e.g., Vald, Hawkin Dynamics). This makes both the analyses, and ensuring reproducibility of results from these analyses, more challenging for the various professionals conducting jump analyses from force plate data. The API for JumpMetrics was designed to be modular and easy to use for those familiar with Python. Researchers, data scientists, analysts, and even coaches can examine each jump’s takeoff and landing phases individually or together, depending on their needs and research questions. Furthermore, JumpMetrics provides various helper functions to prepare the data for

42 detecting events and computing metrics. These helper functions involve cropping longer force
43 traces and digitally low-pass filtering time series data (if these processing steps are required for
44 a researcher or practitioner's particular analysis). In addition to its applications in sports science
45 research and in professional practice, JumpMetrics can also be leveraged by undergraduate or
46 graduate students to assist with sport or exercise science-related projects. Finally, the code,
47 docstrings, and documentation provided in this package may help sports science students learn
48 Python programming if they aren't already familiar.

49 How the Library Works

50 Event Detections and Metrics

51 JumpMetrics computes various events and metrics for the countermovement and squat jump
52 leveraging the vertical axis data from a force plate. The events (and thus metrics) are slightly
53 different between the jump variations, given the differences in their movement executions.
54 There are classes for processing various phases of the countermovement jump, a helper function
55 to process the entire jump and landing, as well as individual functions for even more granularity
56 for analyses. Furthermore, JumpMetrics computes the vertical axis acceleration, velocity, and
57 displacement of the estimated center of mass trajectory for each frame of data, irrespective
58 of whether one is using a triaxial or uniaxial force plate. These data are computed by first
59 dividing the force trace by the individual's computed bodymass to obtain the acceleration data.
60 Then, the acceleration signal is integrated to compute the instantaneous velocity. Finally, the
61 signal is integrated one more time to compute the instantaneous displacement. Some examples
62 of these computed waveforms are shown in Figures 1, 2, and 3.

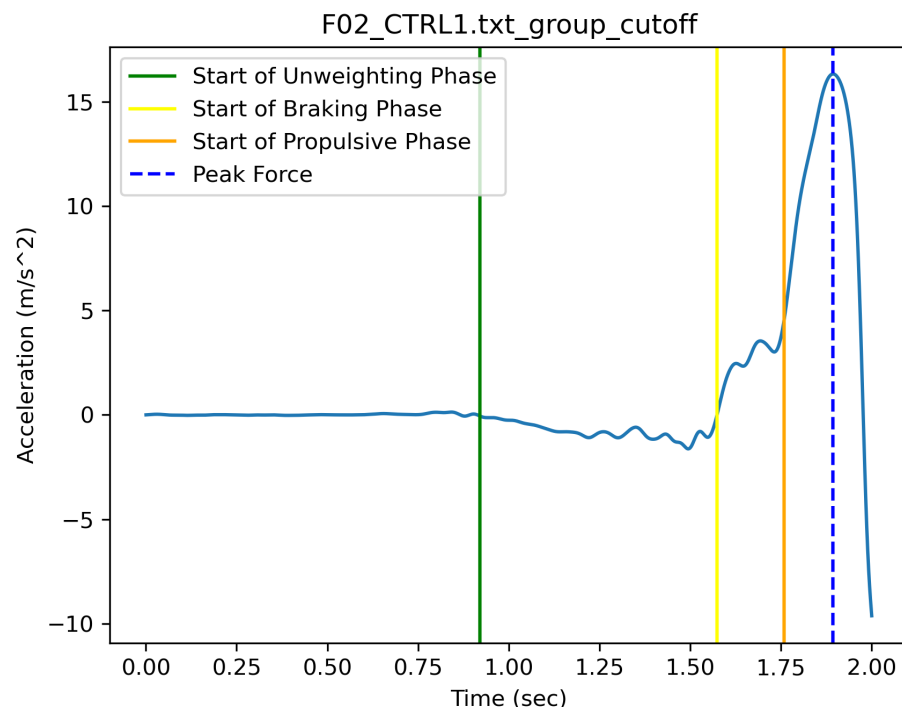


Figure 1: Example acceleration waveform (i.e., the raw force trace divided by the body mass, which was estimated from bodyweight).

63 Figure 1. Example countermovement jump acceleration trace with events detected during

64 the takeoff phase. Positive accelerations represent the center of mass accelerating upwards,
65 whereas negative accelerations represent the center of mass accelerating downwards towards
66 the floor/force plate.

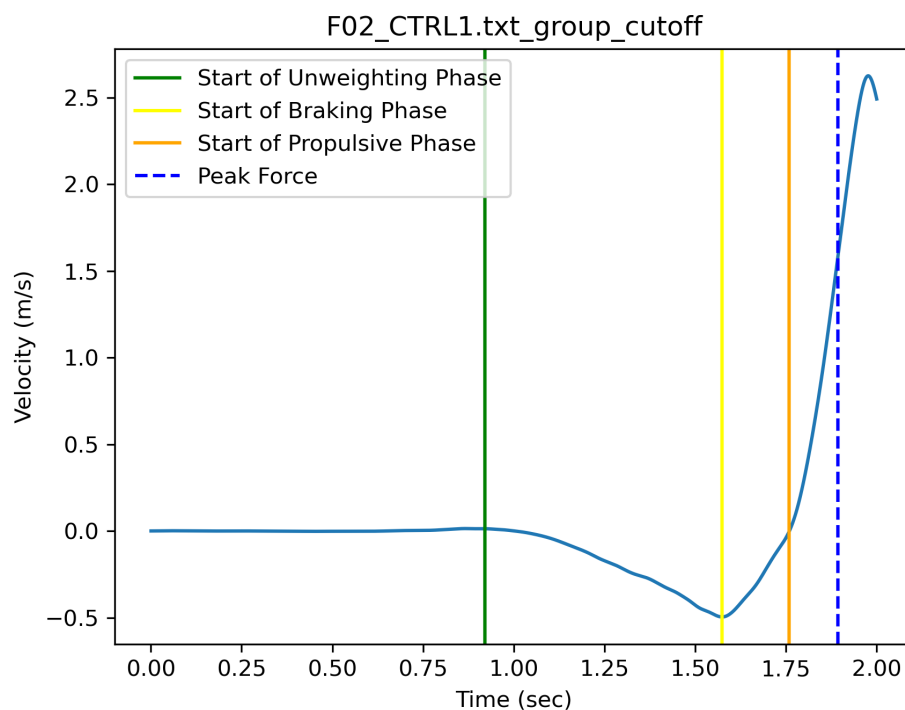


Figure 2: Example velocity waveform (i.e., the integrated acceleration waveform, assuming 0 velocity during quiet standing).

67 Figure 2. Example countermovement jump velocity trace with events detected during the
68 takeoff phase. Positive velocities represent the center of mass is moving upwards, whereas
69 negative velocities represent the center of mass moving downwards.

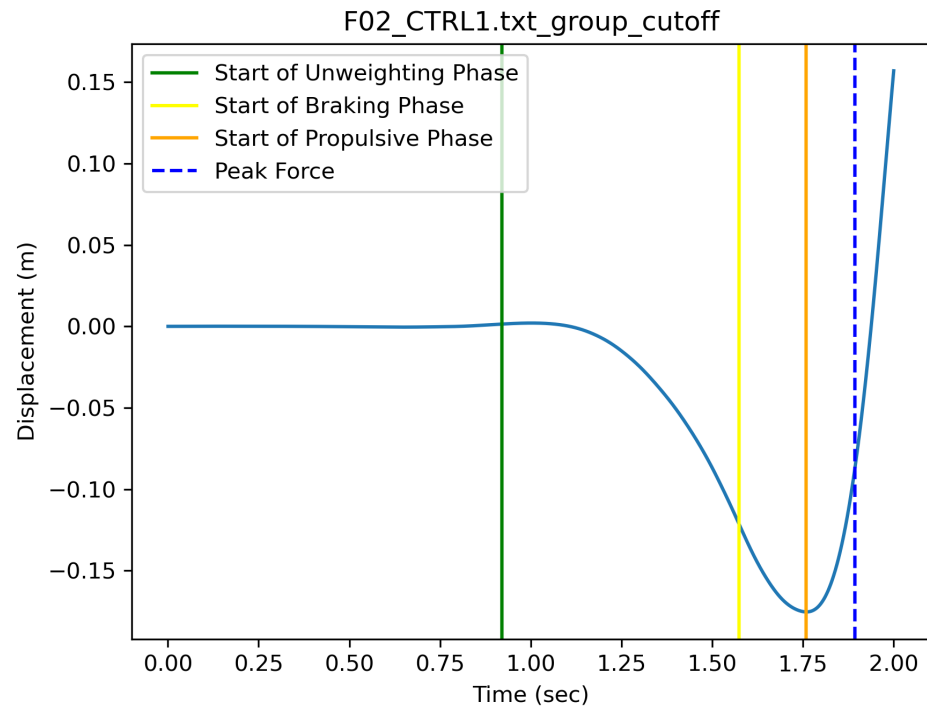


Figure 3: Example displacement waveform (i.e., the integrated velocity waveform, assuming 0 displacement during quiet standing).

70 Figure 3. Example countermovement jump displacement trace with events detected during the
71 takeoff phase. Positive displacements represent the center of mass being higher relative to
72 quiet standing, whereas negative displacements represent the center of mass being closer to
73 the floor relative to quiet standing.

74 To compute the relevant events and metrics, there are specific methods that a user should
75 adhere to that are outlined in @mcmahon:2018. These methods underpin the assumptions
76 required to collect and process data with the code provided in this package. First, the jumper
77 must stand still (i.e., minimizing swaying or any other body movements) at the start of the data
78 collection and for at least 1 second before starting the initiation of the jump. This quiet standing
79 is used to calculate one's bodyweight, and bodyweight is used for subsequent acceleration,
80 velocity, and displacement calculations used for event detections (as well as for computing net
81 vertical impulse). The default setting in this package is currently to use the first 0.4 seconds
82 of the trial to compute bodyweight (as this was found to work well for previous analyses),
83 but users can tune this parameter themselves for their own data collections depending on the
84 length of the quiet standing at the start of the data collection. For a countermovement jump,
85 the functions in this package also require the person to perform one continuous downwards
86 and upwards motion during the jump; any pausing may negatively impact the event detection
87 algorithms provided. In contrast, for the squat jump the default parameter for identifying the
88 start of the propulsive phase expects at least a 1 second pause. In practice, previous research
89 has outlined a pause should be approximately 3 seconds [@vanHooren2017]. The functions
90 provided in JumpMetrics permit the user to select a different minimum pause to assume if the
91 default of 1 second is not appropriate for their research.

92 Phases Until Takeoff

93 Countermovement Jumps

94 There are two main phases preceding the moment of takeoff during countermovement jumps.
95 The two phases are the “lowering” (sometimes referred to as the eccentric) and “ascending”
96 (sometimes referred to as the concentric) phases. The specific events within these two phases
97 detected in this package are based on @mcmahon:2018. Each event corresponds the start of
98 each subphase within the jump. These involve: 1) the start of the unweighting phase, 2) the
99 start of the braking phase, 3) the start of the propulsive phase (the event which separates the
100 lowering and ascending phases), 4) the frame corresponding to the peak force, and 5) the
101 frame corresponding to takeoff.

102 JumpMetrics computes the start of the unweighting phase in the same manner as outlined in
103 @owen:2014 whereby the first frame of force data that exceeds five times the standard deviation
104 of the force data (default value; this is a tuneable parameter depending on the data collection
105 parameters) during quiet standing (sometimes referred to as the weighing phase) defines the
106 start of the unweighting phase. The braking phase starts at the frame corresponding to the
107 maximum downward movement velocity (of the individual’s estimated center of mass; see
108 Figure 2). The propulsive phase starts at the frame corresponding to the minimum downward
109 displacement (of the individual’s estimated center of mass; see Figure 3). This event is used
110 because it is reliably detected and avoids any potential awkwardness of using a minimum
111 positive velocity to define the start of this event whereby the person is moving upwards, but
112 is still considered to be in the “braking” phase (e.g., if the minimum threshold to determine
113 the start of the propulsive phase is 0.10m/s, but the person is currently moving 0.05m/s
114 upwards they would still be in the “braking” phase). The peak force event is captured using
115 find_peaks from the scipy package and looks for a “peak” in the force series. The takeoff
116 event is detected by looking for the first frame of data whereby the force series is below a
117 certain threshold (default is 10 Newtons) for a specific period of time (default is 0.25 seconds).
118 JumpMetrics makes these various parameters for detecting events tunable in cases where the
119 defaults may not accurately detect events due to unexpected noise in the data or if there are
120 any changes in the methodology proposed with future research.

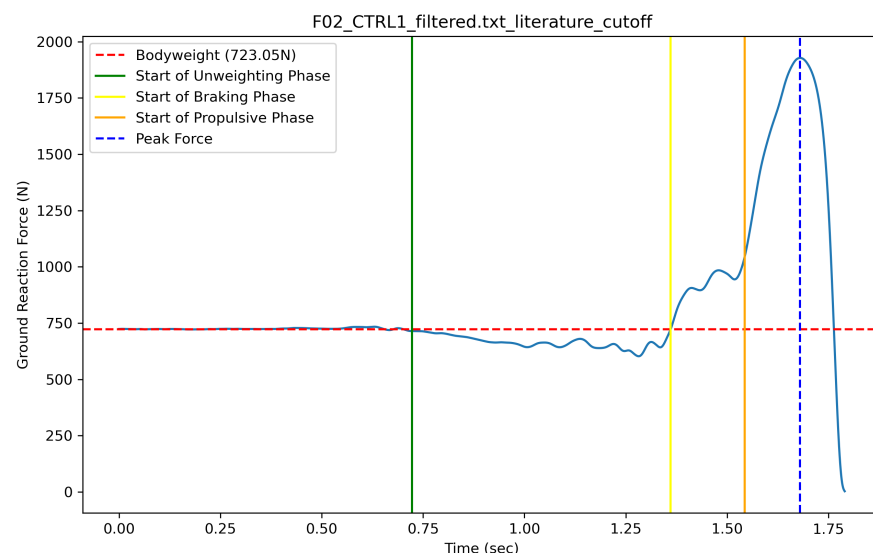


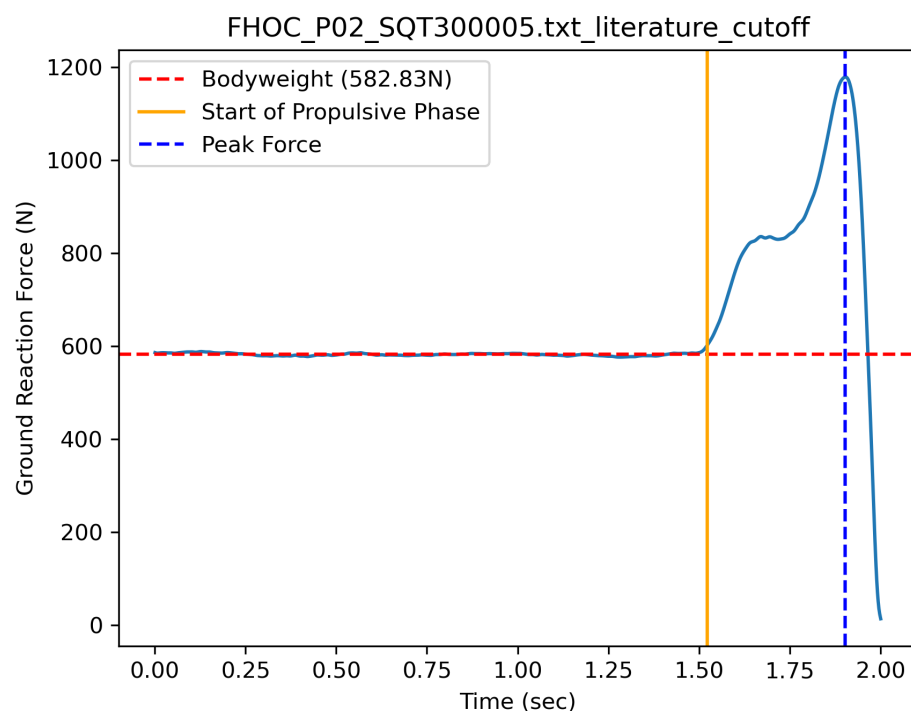
Figure 4: Example countermovement jump force-time trace with events detected during the takeoff phase.

121 Figure 4. Example countermovement jump force-time trace with events detected during the
122 takeoff phase.

JumpMetrics computes the rate of force development, net vertical impulse, and average force between all events detected during the countermovement jump. Additionally, metrics such as the jump height (based on the net vertical impulse using the impulse-momentum relationship as well as the center of mass velocity at the final frame of data before takeoff), takeoff velocity, movement time, unweighting time, braking time, propulsive time, and lowering displacement are also all computed. Table 1 contains a complete list of the metrics JumpMetrics exports for countermovement jumps. Note that in order for the events and metrics to be computed accurately, a brief weighing phase (default is the first 0.25 seconds of data, but this can be modified depending on how long the individual was standing) must be present at the start of the waveform in order to determine the individual's bodyweight and to detect subsequent jump events. Table 2 contains a small sample of how, more generally, some of the metrics computed in JumpMetrics can be leveraged for various applications (please note that this table only covers potential applications and the final decision of when to use a particular metric is dependent on how the data is collected and the particular research question the analyst wishes to answer). Critically, the helper functions in jump metrics, such as `compute_rfd` (i.e., compute rate of force development) allow the user to easily compute any necessary metric beyond the defaults computed in the current version of this package.

Squat Jumps

Given that the squat jump is intentionally performed with a pause to minimize the influence of the stretch shortening cycle of the lower body muscles from a continuous countermovement (i.e., there is no lowering phase), the only events JumpMetrics detects are the start of the propulsive phase, the peak force event, and the takeoff event. The start of the propulsive phase is the first frame of data that exceeds five times the standard deviation of the force data (default value; this is a tuneable parameter depending on the data collection parameters) during the squat phase. The peak force event is computed similarly to the countermovement jumps whereby `find_peaks` from the `scipy` package and looks for a "peak" in the force series. The takeoff event is detected by looking for the first frame of data whereby the force series is below a certain threshold (default is 10 Newtons) for a specific period of time (default is 0.25 seconds).



151

Figure 5. Example squat jump force-time trace during the takeoff phase.

Although there is not supposed to be any countermovement/lowering phase during a squat jump, depending on the instructions and guidance provided to the participant, as well as their general movement behaviours, there may be a minor countermovement that would negate the trial from being a true squat jump. JumpMetrics detects and flags this motion (with a warning and an estimated frame in the metrics output) to make the user aware of this potential flaw in the squat jump trial.

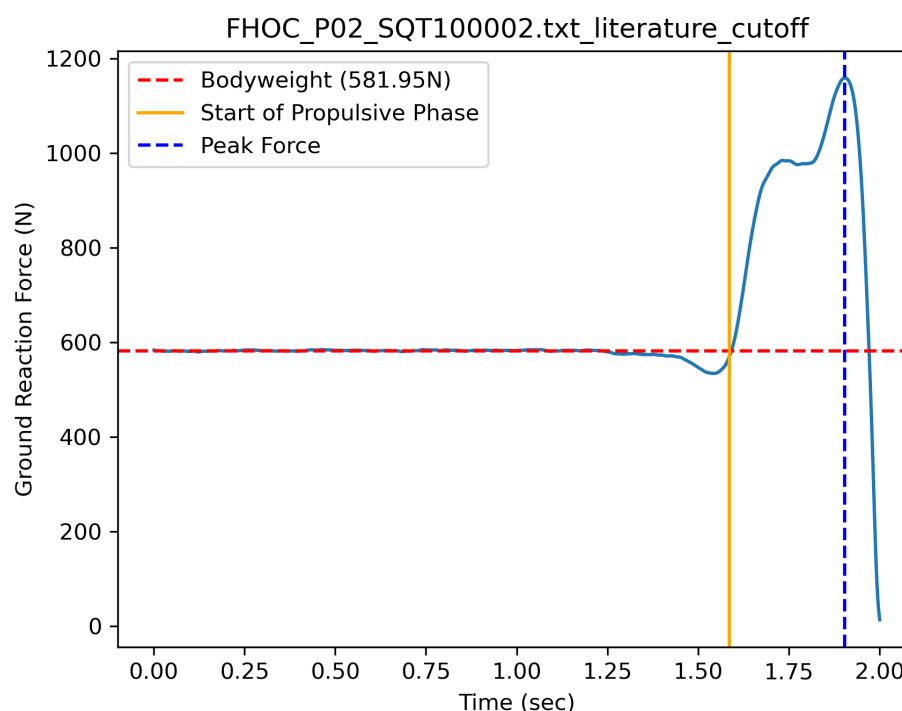


Figure 6. Example squat jump force-time trace with an inappropriate countermovement detected during the takeoff phase.

JumpMetrics computes the rate of force development, net vertical impulse, and average force between all events detected during the squat jump. Additionally, metrics such as the jump height (based on the net vertical impulse and using the impulse-momentum relationship and the velocity at the final frame of data before takeoff), takeoff velocity, movement time, and propulsive time are also all computed. Table 1 contains a complete list of the metrics JumpMetrics exports for squat jumps. Note that in order for the events and metrics to be computed accurately, a brief weighing phase (default is the first 0.25 seconds of data, but this can be modified depending on how long the individual was standing) must be present in the waveform in order to determine the individual's bodyweight.

Landing Phase

The landing phase is defined by the methodology outlined in @mcmahon:2018. The initial landing event is detected by looking for the first frame of data whereby the force series is above a certain threshold (default is 20 Newtons) for a specific period of time (default is 0.015 seconds). The landing phase's end is the point where the estimated center of mass velocity becomes greater than, or equal to, 0 meters per second (given that a negative velocity represents a downward movement).

During the jump's landing phase, JumpMetrics computes the maximum landing force, average landing force, landing time, landing displacement, and various landing rate of force development

180 metrics.

181 Wrapper Function

182 There is also a wrapper function named `process_jump_trial()` that combines the classes for
 183 both the phases up to takeoff and following the landing of the jump to compute all relevant
 184 events and metrics. Additionally, because this function uses the entire force trace as its input,
 185 this function also computes two additional metrics. These metrics are the jump height based
 186 on the flight time (i.e., the between takeoff and landing) as well as the flight time itself.
 187 Although jump height based on flight time is less robust relative to jump height based on
 188 the impulse-momentum theorem [xu:2023], the flight time jump height is still included for
 189 researchers or practitioners who may have historical reference data using this method.

190 Tables

191 Table 1. Specific metrics computed and exported by JumpMetrics at Takeoff for Coun-
 192 termovement Jumps (CMJ) and Squat Jumps (SQJ). | Metric | CMJ | SQJ | Descrip-
 193 tion | |-----|-----|-----| | propul-
 194 sive_peakforce_rfd_slope_between_events | yes | yes | Rate of force development computed
 195 as the slope between the start of the propulsive phase to the frame of peak force (N/s) |
 196 propulsive_peakforce_rfd_instantaneous_average_between_events | yes | yes | Rate of force
 197 development computed as the average instantaneous value between each frame the start of
 198 the propulsive phase to the frame of peak force (N/s) | propulsive_peakforce_rfd_instanta-
 199 neous_peak_between_events | yes | yes | Rate of force development computed as the peak
 200 instantaneous value between each frame the start of the propulsive phase to the frame of peak
 201 force (N/s) | braking_peakforce_rfd_slope_between_events | yes | | Rate of force development
 202 computed as the slope between the start of the braking phase to the frame of peak force
 203 (N/s) | braking_peakforce_rfd_instantaneous_average_between_events | yes | | Rate of force
 204 development computed as the average instantaneous value between each frame during the
 205 start of the braking phase to the frame of peak force (N/s) | braking_peakforce_rfd_in-
 206 stantaneous_peak_between_events | yes | | Rate of force development computed as the peak
 207 instantaneous value between each frame during the start of the braking phase to the frame
 208 of peak force (N/s) | braking_propulsive_rfd_slope_between_events | yes | | Rate of force
 209 development slope from the braking phase to the start of the propulsive phase (N/s) | brak-
 210 ing_propulsive_rfd_instantaneous_average_between_events | yes | | Rate of force development
 211 computed as the average instantaneous value from the braking to the propulsive phase (N/s) |
 212 | braking_propulsive_rfd_instantaneous_peak_between_events | yes | | Rate of force develop-
 213 ment computed as the peak instantaneous value from braking to the propulsive phase (N/s) |
 214 braking_net_vertical_impulse | yes | | Net vertical impulse during the braking phase (N s) |
 215 propulsive_net_vertical_impulse | yes | | Net vertical impulse during the propulsive phase (N s) |
 216 | braking_to_propulsive_net_vertical_impulse | yes | | Net vertical impulse between the start of
 217 the braking phase to the propulsive phase (N s) | total_net_vertical_impulse | yes | yes | Net
 218 vertical impulse during the entire jump (N s) | peak_force | yes | yes | Peak force defined using
 219 `find_peaks()` in `scipy` (N) | maximum_force | yes | yes | Global maximum value of force recorded
 220 during the jump (N) | average_force_of_braking_phase | yes | | Average force applied during
 221 the braking phase (N) | average_force_of_propulsive_phase | yes | yes | Average force applied
 222 during the propulsive phase (N) | takeoff_velocity | yes | yes | Estimated takeoff velocity of the
 223 center of mass (m/s) | jump_height_takeoff_velocity | yes | yes | Jump height calculated using
 224 the estimated takeoff velocity of the center of mass (m) | jump_height_net_vertical_impulse
 225 | yes | yes | Jump height calculated using the impulse-momentum theorem. Usually identical to
 226 the jump height based on takeoff velocity except for rounding errors due to slightly different
 227 computations (m) | jump_height_flight_time** | yes | yes | Jump height calculated using
 228 flight time (m) | flight_time** | yes | yes | Flight time of the jump (s) | movement_time | yes
 229 | yes | Time between initiation of jump and takeoff (s) | unweighting_time | yes | | Time during

the unweighting phase of the jump (s) | braking_time | yes | Time during the braking phase of the jump (s) | propulsive_time | yes | yes | Time during the propulsive phase of the jump (s) | lowering_displacement | yes | Estimated displacement of the center of mass during the lowering phase (m) | frame_start_of_unweighting_phase | yes | Frame corresponding to the start of the unweighting phase | frame_start_of_breaking_phase | yes | Frame corresponding to the start of the braking phase | frame_start_of_propulsive_phase | yes | yes | Frame corresponding to the start of the propulsive phase | frame_peak_force | yes | yes | Frame corresponding to the peak force during the jump | frame_of_potential_unweighting_start | yes | yes | Frame corresponding to the potential start of the unweighting phase

Table 2. Table of general metrics along with potential typical applications. Note that the appropriateness of a potential metric for the listed purpose (or beyond the listed purpose) is dependent on how the data was collected and the particular question the analyst is attempting to answer. This table is a guide only based on a limited selection of papers.

General Metric	Capacity	Injury Risk	Readiness	Example Paper(s)
RFD (Rate of Force Development)	yes	yes		[@mcmahon:2017], [@bird:2022]
Net Vertical Impulse	yes	yes		[@mcmahon:2017], [@bird:2022]
Peak Force	yes	yes (during landing)		[@mcmahon:2017], [@pedley2020]
Maximum Force	yes	yes (during landing)		[@mcmahon:2017], [@pedley2020]
Jump Height	yes		yes	[@watkins:2017], [@mcmahon:2017]
Braking Time		yes		[@bird:2022]
Propulsive Time		yes		[@bird:2022]

Note that jump_height_flight_time and flight_time are only available when using the process_jump_trial() function as both takeoff and landing must be detected to determine the flight time.

Acknowledgements

I acknowledge contributions from Malinda Hapuarachchi for providing the data required to develop, test, and verify the functions in this package. I also wish to acknowledge Jacob Rauch for his feedback on the paper.

References