Package: pressR (via r-universe)

November 3, 2026

Title Pressure Distribution Analysis for Capacitive Sensor Systems

Version 0.1.0

Description Parse, analyze, and visualize pressure distribution data from capacitive sensor systems. Provides predefined sensor coordinate layouts for in-shoe pressure measurement, saddle pressure mapping (equine and bicycle), seating assessment, and barefoot pedography. Includes functions for peak pressure, center of pressure, pressure-time integrals, regional analysis with anatomical masks, symmetry indices, and gait cycle detection. Ships with an interactive Shiny application for data exploration and report generation.

License MIT + file LICENSE

URL https://github.com/cttir/pressR, https://cttir.github.io/pressR/

BugReports https://github.com/cttir/pressR/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Depends R (>= 4.1.0)

Imports tibble, dplyr, tidyr, readr, rlang, cli, ggplot2, plotly, DT, shiny, bslib, patchwork, stats, utils, tools

Suggests gganimate, gifski, withr, testthat (>= 3.0.0), knitr, rmarkdown, pkgdown, covr, shinytest2, vdiffr

VignetteBuilder knitr

Config/Needs/website CTTIR/themakR

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev libx11-dev zlib1g-dev

Repository https://cttir.r-universe.dev

Date/Publication 2026-10-29 07:36:05 UTC

RemoteUrl https://github.com/CTTIR/pressR

RemoteRef HEAD

RemoteSha d8fbd2cac9b97027029f5bc5f65e447c9e34462b

Contents

plot.pr_layout . . . 4
plot.pr_trial . . . 5
pr_assert_cohort . . . 5
pr_batch_frame_summary . . . 6
pr_batch_summary . . . 8
pr_calc_contact_area . . . 9
pr_calc_contact_time . . . 9
pr_calc_cop . . . 10
pr_calc_cop_excursion . . . 10
pr_calc_cop_grid . . . 11
pr_calc_cop_masked . . . 12
pr_calc_cop_path . . . 13
pr_calc_dominant_freq . . . 14
pr_calc_force . . . 15
pr_calc_gait_cycles . . . 15
pr_calc_gradient . . . 16
pr_calc_impulse . . . 18
pr_calc_loaded_count . . . 18
pr_calc_loaded_rate . . . 19
pr_calc_mean_pressure . . . 20
pr_calc_mean_pressure_grid . . . 20
pr_calc_pci . . . 21
pr_calc_peak_pressure . . . 22
pr_calc_phase_map . . . 23
pr_calc_pti . . . 25
pr_calc_regional . . . 25
pr_calc_regional_map . . . 26
pr_calc_rollover . . . 28
pr_calc_saddle_bridge . . . 28
pr_calc_saddle_slip . . . 29
pr_calc_seat_hotspot . . . 30
pr_calc_spectrum . . . 30
pr_calc_stride_cycles . . . 32
pr_calc_symmetry_index . . . 34
pr_calc_symmetry_map . . . 35
pr_calc_total_pressure . . . 37
pr_channel_order . . . 38
pr_cop . . . 39
pr_cop_shape . . . 39
pr_dataset . . . 41
pr_dataset_filter . . . 42
pr_design_gaps . . . 43
pr_design_table . . . 44
pr_downsample . . . 45
pr_example_files . . . 46
pr_example_trial . . . 46

pr_export_csv . . . 47
pr_export_report . . . 48
pr_factor_contract . . . 49
pr_filter_time . . . 50
pr_frame_metrics . . . 51
pr_interpolate . . . 52
pr_layout . . . 53
pr_layout_from_index_map . . . 54
pr_layout_glove . . . 56
pr_layout_insole . . . 56
pr_layout_list . . . 57
pr_layout_mat . . . 57
pr_layout_platform . . . 58
pr_layout_saddle . . . 58
pr_layout_saddle_novel . . . 59
pr_layout_seat . . . 60
pr_lock_table . . . 61
pr_mask . . . 62
pr_mask_apply . . . 63
pr_mask_default . . . 64
pr_mask_foot_auto . . . 64
pr_mask_mirror_balance . . . 65
pr_mask_rowbands . . . 66
pr_mask_saddle_6 . . . 67
pr_mask_symmetry . . . 68
pr_merge_trials . . . 69
pr_meta_from_filename . . . 69
pr_plot_3d . . . 70
pr_plot_comparison . . . 71
pr_plot_contact_area . . . 72
pr_plot_cop . . . 72
pr_plot_cop_butterfly . . . 73
pr_plot_foot_report . . . 73
pr_plot_force_time . . . 74
pr_plot_heatmap . . . 74
pr_plot_heatmap_masked . . . 75
pr_plot_pressure_time . . . 76
pr_plot_regional_bar . . . 77
pr_plot_saddle_report . . . 77
pr_plot_symmetry . . . 78
pr_profile_matrix . . . 79
pr_read_ascii . . . 80
pr_read_auto . . . 81
pr_read_csv . . . 81
pr_read_dir . . . 82
pr_read_forcesensor . . . 84
pr_read_mask . . . 85
pr_read_pliance . . . 86

pr_ref_diabetic_foot . . . 88
pr_ref_pci . . . 88
pr_ref_saddle . . . 89
pr_ref_wheelchair . . . 90
pr_run_app . . . 90
pr_sensor_map . . . 91
pr_sensor_quality . . . 92
pr_summary . . . 95
pr_symmetry_sensitivity . . . 96
pr_trial . . . 97
pr_validate_dataset . . . 98
pr_validate_layout . . . 99
pr_validate_summary . . . 100
pr_verify_lock . . . 102

Index **104**

`plot.pr_layout` *Plot a Sensor Layout*

Description

Visualizes a pr_layout showing active sensor positions and colored region masks (if any).

Usage

```
## S3 method for class 'pr_layout'
plot(x, show_regions = TRUE, show_ids = FALSE, ...)
```

Arguments

`x`	A pr_layout object.
`show_regions`	Logical. If `TRUE` (default), draws region masks in colors. If `FALSE`, shows only sensor positions.
`show_ids`	Logical. If `TRUE`, labels each sensor with its id. Default `FALSE`.
`...`	Not used.

Value

A `ggplot2` object.

Examples

```
plot(pr_layout_saddle("horse"))
```

plot.pr_trial *Plot a Pressure Trial*

Description

Default plot method for `pr_trial` — shows the maximum pressure picture (MPP) as a heatmap.

Usage

```
## S3 method for class 'pr_trial'
plot(x, ...)
```

Arguments

`x`	A pr_trial object.
`...`	Passed to `pr_plot_heatmap()`.

Value

A `ggplot2` object.

pr_assert_cohort *Assert Cohort Size*

Description

Checks that a cohort holds exactly the expected number of units (trials or rows) and, optionally, of distinct subjects. Aborts with a message naming the expectation, the actual count, and the shortfall or surplus.

Usage

```
pr_assert_cohort(
  x,
  n_units = NULL,
  n_subjects = NULL,
  unit_col = NULL,
  subject_col = NULL,
  label = NULL
)
```

Arguments

x	A pr_dataset object, a list of pr_trial objects, or a data frame (for example the output of pr_design_table()).
n_units	Integer. Expected number of units, or NULL to skip. A unit is a trial (dataset) or a row (data frame) unless unit_col names a field/column, in which case units are its distinct values.
n_subjects	Integer. Expected number of distinct subjects, or NULL to skip.
unit_col	Character. Field/column whose distinct values are the units. NULL (default) counts trials or rows.
subject_col	Character. Field/column identifying the subject. NULL (default) uses the first of "subject_id" or "ID" that exists, and aborts if neither does.
label	Character. Name of the cohort, used in messages. NULL (default) uses the dataset's filter_label or name, else "cohort".

Details

Use it as a tripwire around every step that can silently drop data — reading a directory, filtering, joining. A guard that has never fired proves nothing, so give it the real numbers from your design.

Value

Invisibly returns x, so the assertion can sit in a pipeline. Aborts with condition class pr_cohort_assert_failed on mismatch.

See Also

Other cohort functions: pr_dataset_filter(), pr_design_table(), pr_validate_dataset()

Examples

```
design <- data.frame(
  ID = c("ID001", "ID001", "ID002"),
  Mode = c("MS", "MH", "MS")
)
pr_assert_cohort(design, n_units = 3, n_subjects = 2, subject_col = "ID")

# A wrong expectation fails loudly:
try(pr_assert_cohort(design, n_units = 431, label = "all recordings"))
```

---

pr_batch_frame_summary

Per-Recording Frame Summary for a Whole Cohort

Description

Runs pr_frame_metrics() over every trial in a dataset and reduces each frame table to a single row: the nine summary statistics a cohort analysis reports per recording, plus the trial's metadata.

Usage

```
pr_batch_frame_summary(
  dataset,
  threshold = 0,
  meta_fields = NULL,
  .progress = FALSE
)
```

Arguments

dataset	A pr_dataset object, or a list of pr_trial objects.
threshold	Numeric. Cells at or below this value count as unloaded; passed to pr_frame_metrics(), where it affects loaded and the COP weighting only. Default 0.
meta_fields	Character vector of metadata field names to prepend, in the order given. NULL (default) auto-detects: every field that holds a single value in every trial, dropping those that are NA throughout. Pass character(0) for the nine statistics alone.
.progress	Logical. Show a progress bar over trials. Default FALSE.

Details

Columns, in order: the requested metadata fields, then

- mean_kPa_avg — mean over frames of the whole-grid frame mean.
- mean_kPa_sd — standard deviation over frames of that same quantity, i.e. how much overall load varied during the recording (NA for a single-frame trial).
- peak_kPa_max — the largest single-sensor reading anywhere in the recording. On a saturating device this pins to the hardware ceiling.
- peak_kPa_avg — mean over frames of the frame maximum. Together with mean_kPa_avg this gives the concentration index peak_kPa_avg / mean_kPa_avg (see pr_calc_pci()).
- total_kPa_avg — mean over frames of the frame sum. Exactly mean_kPa_avg * n_sensors; a sum of pressures, never a force.
- loaded_avg — mean over frames of the loaded cell count.
- cop_row_mean, cop_col_mean — mean over frames of the centre of pressure, in grid index units.
- n_frames — frames in the recording, after any the reader dropped.

These names and definitions are the cohort contract, chosen so the result can be compared column-for-column against a study's own summary table. Note that mean_kPa_avg is the *whole-grid* mean including unloaded cells (see pr_calc_mean_pressure_grid()), which is a different quantity from pr_calc_mean_pressure().

Value

A tibble::tibble with one row per trial.

Memory

The frame tables are built and discarded one at a time, so peak memory is one recording's frames, not the cohort's. That matters: 431 recordings of 5.5 M frames total would need several gigabytes if every frame table were materialised first, to produce nine numbers each.

See Also

`pr_frame_metrics()` for the per-frame table this summarises, `pr_design_table()` for the metadata side alone.

Other sensor aggregation functions: `pr_profile_matrix()`, `pr_sensor_map()`

Examples

```
ds <- pr_dataset(list(
  pr_example_trial("saddle_horse", seed = 1),
  pr_example_trial("saddle_horse", seed = 2)
))
fs <- pr_batch_frame_summary(ds, meta_fields = character(0))
names(fs)
round(fs$mean_kPa_avg, 3)

# total_kPa_avg is mean_kPa_avg scaled by the sensor count:
n_sensors <- ds$trials[[1]]$layout$n_sensors
all.equal(fs$total_kPa_avg, fs$mean_kPa_avg * n_sensors)
```

---

pr_batch_summary *Batch Summary Across Multiple Trials*

Description

Runs `pr_summary()` on each trial in a dataset and row-binds the results, prepending trial metadata columns.

Usage

```
pr_batch_summary(dataset)
```

Arguments

`dataset`	A pr_dataset object or a list of pr_trial objects.

Value

A tibble::tibble with one row per trial.

Examples

```
ds <- pr_dataset(list(
  pr_example_trial("insole", seed = 1),
  pr_example_trial("insole", seed = 2)
))
pr_batch_summary(ds)
```

---

pr_calc_contact_area *Calculate Contact Area Per Frame*

Description

Number of sensors above `threshold` times the per-sensor area (cm^2).

Usage

```
pr_calc_contact_area(trial, threshold = 0)
```

Arguments

`trial`	A pr_trial object.
`threshold`	Numeric. Default `0`.

Value

Numeric vector of contact area (cm^2) per frame.

---

pr_calc_contact_time *Total Contact Time*

Description

Duration for which at least one sensor exceeds `threshold`.

Usage

```
pr_calc_contact_time(trial, threshold = 0)
```

Arguments

`trial`	A pr_trial object.
`threshold`	Numeric. Default `0`.

Value

Numeric scalar (s).

pr_calc_cop *Calculate Center of Pressure*

Description

Computes the pressure-weighted centroid of active sensors for each frame. Frames with no loaded sensors return NA for x and y.

Usage

```
pr_calc_cop(trial, threshold = 0)
```

Arguments

trial	A pr_trial object.
threshold	Numeric. Sensors at or below are treated as unloaded. Default 0.

Value

A pr_cop object containing x (mm), y (mm), time (s), and derived metrics.

Examples

```
cop <- pr_calc_cop(pr_example_trial("saddle_horse"))
print(cop)
```

pr_calc_cop_excursion *COP Excursion (Anterior-Posterior and Medial-Lateral Range)*

Description

COP Excursion (Anterior-Posterior and Medial-Lateral Range)

Usage

```
pr_calc_cop_excursion(trial, threshold = 0)
```

Arguments

trial	A pr_trial object.
threshold	Numeric. Default 0.

Value

Named numeric vector with ap (y-range, mm) and ml (x-range, mm).

pr_calc_cop_grid *Center of Pressure in Grid Index Units*

Description

Pressure-weighted centroid of each frame expressed in grid indices: a `cop_row` of 1 is the first sensor row, a `cop_col` of 16 the sixteenth sensor column. Row and col are read from the layout's coordinate table positionally, so the result stays correct under any device channel permutation applied by the reader.

Usage

```
pr_calc_cop_grid(trial, threshold = 0, eps = 1e-12)
```

Arguments

`trial`	A pr_trial object.
`threshold`	Numeric. Cells at or below this value count as unloaded. Default `0`.
`eps`	Numeric. Added to the weight sum to keep unloaded frames finite. Default `1e-12`.

Details

`pr_calc_cop()` returns the same centroid in millimetres and gives `NA` for frames with no load. This function is the index-unit counterpart used when a device's physical sensor pitch was never published, so millimetre coordinates would be fabricated. Instead of `NA`, an unloaded frame gets a centroid of approximately `0` for both axes: `eps` is added to the weight sum, so the quotient is `0 / eps` rather than `0 / 0`. That keeps the output a complete, non-`NA` time series, and `cop_row == 0` is unambiguous because a real centroid is always at least 1.

Note also that the whole-grid metrics here differ from `pr_calc_mean_pressure()`, which is loaded-cells-only by definition.

Value

A tibble::tibble with one row per frame and columns `frame` (integer), `time_s`, `cop_row`, `cop_col` (grid index units).

See Also

Other whole-grid frame metrics: `pr_calc_loaded_count()`, `pr_calc_mean_pressure_grid()`, `pr_calc_pci()`, `pr_calc_total_pressure()`, `pr_frame_metrics()`, `pr_ref_pci()`

Examples

```
trial <- pr_example_trial("saddle_horse")
cop <- pr_calc_cop_grid(trial)
range(cop$cop_row)
```

`pr_calc_cop_masked` *Centre of Pressure Per Region*

Description

Computes one centre-of-pressure trajectory for each region mask, weighted only by that region's sensors. Returns a list of pr_cop objects, so the existing trajectory vocabulary — path length, mean and maximum velocity, the x/y ranges `pr_calc_cop_excursion()` reports, and the 95% ellipse sway area — is available per region with no further work.

Usage

```
pr_calc_cop_masked(trial, masks = NULL, threshold = 0, units = c("grid", "mm"))
```

Arguments

`trial`	A pr_trial object.
`masks`	Named list of pr_mask objects or logical matrices. `NULL` (default) uses the layout's own regions.
`threshold`	Numeric. Cells at or below this value are given zero weight. Default `0`.
`units`	Character. `"grid"` (default) or `"mm"`; see *Units*.

Details

A whole-mat COP is a single point that can sit in an unloaded gap between two loaded zones, and it moves whenever load shifts *between* zones. A per-region COP cannot: it stays inside its own region and reports how load migrated within it, which is what separates a saddle that rocks fore-and-aft from one that slides sideways.

Frames in which a region carries no load at all get `NA` for both axes, exactly as `pr_calc_cop()` does for the whole mat. `pr_cop()` excludes those frames from the trajectory metrics, so a region that is loaded only intermittently still reports a meaningful path length — though one measured across the gaps, not through them.

Value

A named list of pr_cop objects, one per mask, in the order the masks were given.

Units

`units = "grid"` (the default) returns grid index units: `x` is the fractional sensor column, `y` the fractional sensor row, both 1-based, in the convention of `pr_calc_cop_grid()`. This is always available.

`units = "mm"` returns the layout's physical `x_mm` / `y_mm` coordinates, as `pr_calc_cop()` does. It is refused for a layout whose coordinates are grid indices because no sensor pitch was ever published (see the *Missing physical scale* section of `pr_layout_from_index_map()`) — millimetres would be fabricated, and a path length in fabricated millimetres is worse than none.

Note that a pr_cop object labels its fields mm regardless; under units = "grid" they are grid indices, and the derived path length and velocity are in index units and index units per second.

See Also

pr_calc_cop() and pr_calc_cop_grid() for the whole mat, pr_calc_cop_excursion() for the range summary.

Other regional symmetry functions: pr_calc_regional_map(), pr_calc_symmetry_map(), pr_mask_mirror_balance(), pr_mask_rowbands(), pr_symmetry_sensitivity()

Examples

```
trial <- pr_example_trial("saddle_horse")
cops <- pr_calc_cop_masked(trial)
names(cops)

# Each region's COP stays inside its own rows:
round(vapply(cops, function(c) mean(c$y, na.rm = TRUE), numeric(1)), 2)

# The full pr_cop vocabulary works unchanged:
round(vapply(cops, function(c) c$path_length, numeric(1)), 3)

# This layout has a real sensor pitch, so millimetres are honest:
round(pr_calc_cop_masked(trial, units = "mm")$cranial_left$range_x, 1)
```

---

pr_calc_cop_path *Total COP Path Length*

Description

Total COP Path Length

Usage

```
pr_calc_cop_path(trial, threshold = 0)
```

Arguments

trial	A pr_trial object.
threshold	Numeric. Default 0.

Value

Numeric scalar (mm).

pr_calc_dominant_freq *Dominant Oscillation Frequency in a Band*

Description

The strongest frequency of pr_calc_spectrum() inside a search band — on a saddle recording, the stride frequency.

Usage

```
pr_calc_dominant_freq(
  trial,
  band = c(0.3, 5),
  signal = "total",
  window = "hann"
)
```

Arguments

trial	A pr_trial object.
band	Numeric vector of length 2. Inclusive search band in Hz. Default c(0.3, 5).
signal	Character. Frame-level signal to transform. Default "total".
window	Character. Window to taper with. Default "hann".

Details

The default band of 0.3-5 Hz brackets equine stride rates from a slow walk to a fast canter while excluding both the baseline drift below it and sensor noise above it. Band edges are inclusive, and the search happens on the exact frequency axis documented in pr_calc_spectrum().

The result is the single largest bin, with no interpolation between bins, so its resolution is fs / n_frames — 0.058 Hz for a 869-frame recording at 50 Hz. A recording with no oscillation still returns its largest in-band bin; read peak_power before trusting the frequency, and compare against pr_calc_stride_cycles(), which measures the same rhythm in the time domain and disagrees when there is nothing periodic to find.

Value

A one-row tibble::tibble with columns stride_freq_Hz and peak_power. Both are NA (with a warning) when the band contains no frequency bin.

See Also

pr_calc_spectrum() for the full spectrum.

Other temporal structure functions: pr_calc_phase_map(), pr_calc_spectrum(), pr_calc_stride_cycles(), pr_cop_shape()

Examples

```
trial <- pr_example_trial("saddle_horse")
pr_calc_dominant_freq(trial)

# A stride frequency and a mean stride duration are reciprocals:
f <- pr_calc_dominant_freq(trial)$stride_freq_Hz
cycles <- pr_calc_stride_cycles(trial)
round(c(from_spectrum = 1 / f, from_cycles = mean(cycles$duration_s)), 2)
```

---

pr_calc_force *Calculate Total Force Per Frame*

Description

Force in Newtons equals sum(pressure * sensor_area). With pressure in kPa and area in cm^2, force[N] = sum(p_kPa * a_cm2) * 0.1.

Usage

```
pr_calc_force(trial)
```

Arguments

`trial`	A pr_trial object.

Value

Numeric vector of length `n_frames`.

---

pr_calc_gait_cycles *Detect Gait Cycles from Foot Pressure Data*

Description

Identifies heel-strike and toe-off events from the total force curve of in-shoe or platform pressure data, segmenting the trial into individual stance phases.

Usage

```
pr_calc_gait_cycles(
  trial,
  force_threshold = 20,
  min_stance_ms = 200,
  min_swing_ms = 100
)
```

Arguments

`trial`	A pr_trial object from an insole or platform recording.
`force_threshold`	Numeric. Force threshold (N) for foot contact detection. Default `20`.
`min_stance_ms`	Numeric. Minimum stance duration (ms). Shorter contacts are rejected. Default `200`.
`min_swing_ms`	Numeric. Minimum swing duration (ms). Default `100`.

Value

A tibble::tibble with one row per detected cycle.

Examples

```
cycles <- pr_calc_gait_cycles(pr_example_trial("insole"))
cycles
```

`pr_calc_gradient` *Spatial Pressure Gradient Magnitude*

Description

How sharply pressure changes from one sensor to its neighbours. The per-sensor map produced by `pr_sensor_map()` is arranged back onto the device grid, differenced in both directions, and returned as one gradient magnitude per sensor.

Usage

```
pr_calc_gradient(
  trial,
  statistic = "mean",
  method = "central",
  edges = c("na", "zero"),
  threshold = 0
)
```

Arguments

`trial`	A pr_trial object.
`statistic`	Character. The per-sensor map to differentiate; any statistic accepted by `pr_sensor_map()`. Default `"mean"`, which is the cohort definition.
`method`	Character. `"central"` (default) or `"forward"`; see *Details*.
`edges`	Character. `"na"` (default) or `"zero"`; see *The edge ring*.
`threshold`	Numeric. Cells at or below this value count as unloaded; passed to `pr_sensor_map()`. Default `0`.

Details

For a grid `M` of per-sensor values, `method = "central"` computes

$$g_{r,c} = \sqrt{\left(\frac{M_{r+1,c} - M_{r-1,c}}{2}\right)^2 + \left(\frac{M_{r,c+1} - M_{r,c-1}}{2}\right)^2}$$

and `method = "forward"` uses the one-sided differences `M[r+1, c] - M[r, c]` and `M[r, c+1] - M[r, c]` instead. Central differences are symmetric and less noisy; forward differences reach one cell further into the edge, which matters on a small grid.

The result is in the trial's pressure unit **per sensor spacing** — kPa per cell step, not kPa/mm. Converting to kPa/mm needs a published cell pitch, and the saddle mat has none (see the *Missing physical scale* section of `pr_layout_from_index_map()`), so no distance scaling is done anywhere in this function. Gradients are therefore comparable across recordings on the same device and not across devices with different pitches.

Value

A tibble::tibble with one row per sensor, in the same column-major grid order as `pr_sensor_map()`, and columns `sensor` (integer), `row`, `col` (integer grid indices) and `gradient` (numeric).

The edge ring

A central difference needs a neighbour on both sides, so it is undefined on the outermost row and column. The two `edges` settings differ only there, and the interior is bit-for-bit identical either way:

- `edges = "na"` (default) leaves the ring `NA`. This is the honest answer: no gradient was computed for those cells.
- `edges = "zero"` fills the ring with `0`. This reproduces the cohort analysis, which wrote central differences for rows and columns `2:(n-1)` into a zero-initialised matrix and never revisited the border. On a 16x16 mat that makes 60 of 256 cells — nearly a quarter — a **structural zero** rather than a measurement. Use it to reproduce published numbers, but never average, rank or threshold over the whole grid with it: the mean gradient is biased down by a factor of roughly `196/256`, and "the lowest-gradient cells" will be the border, every time. `edges = "na"` with `na.rm = TRUE` gives the interior mean the zero-filled version was probably meant to be.

Cells the layout marks inactive are `NA` in the grid, so a gradient that would have had to read one comes back `NA` under either setting.

See Also

`pr_sensor_map()` for the map being differentiated.

Other sensor quality functions: `pr_sensor_quality()`

Examples

```
trial <- pr_example_trial("saddle_horse")
g <- pr_calc_gradient(trial)
g[which.max(g$gradient), ]
```

```
# The default leaves the border -- and the layout's inactive gullet
# cells -- missing ...
sum(is.na(g$gradient))
# ... while the cohort's zero-filled convention hides the border:
gz <- pr_calc_gradient(trial, edges = "zero")
sum(gz$gradient == 0, na.rm = TRUE)

# Same interior, different whole-grid mean:
c(interior = mean(g$gradient, na.rm = TRUE),
  filled = mean(gz$gradient, na.rm = TRUE))
```

---

pr_calc_impulse *Force-Time Integral (Impulse)*

Description

Integrates total force over time. Result is in N*s.

Usage

```
pr_calc_impulse(trial)
```

Arguments

`trial`	A pr_trial object.

Value

Numeric scalar.

---

pr_calc_loaded_count *Loaded Cell Count Per Frame*

Description

Number of sensors reading strictly above `threshold` in each frame.

Usage

```
pr_calc_loaded_count(trial, threshold = 0)
```

Arguments

`trial`	A pr_trial object.
`threshold`	Numeric. Cells at or below this value count as unloaded. Default `0`.

Details

The count that pr_calc_mean_pressure() divides by, exposed on its own. pr_calc_loaded_rate() returns the same information as a fraction of the grid and pr_calc_contact_area() scales it by the cell area; this function keeps it as a plain cell count, which is the form the cohort summaries report because the device's cell area is unknown.

Value

Integer vector of length n_frames.

See Also

Other whole-grid frame metrics: pr_calc_cop_grid(), pr_calc_mean_pressure_grid(), pr_calc_pci(), pr_calc_total_pressure(), pr_frame_metrics(), pr_ref_pci()

Examples

```
trial <- pr_example_trial("saddle_horse")
loaded <- pr_calc_loaded_count(trial)
max(loaded) <= trial$layout$n_sensors
```

---

pr_calc_loaded_rate *Calculate Fraction of Loaded Sensors Per Frame*

Description

Calculate Fraction of Loaded Sensors Per Frame

Usage

```
pr_calc_loaded_rate(trial, threshold = 0)
```

Arguments

trial	A pr_trial object.
threshold	Numeric. Default 0.

Value

Numeric vector in [0, 1] of length n_frames.

pr_calc_mean_pressure *Calculate Mean Pressure Per Frame*

Description

Mean of loaded sensors per frame (sensors at or below `threshold` are excluded).

Usage

```
pr_calc_mean_pressure(trial, threshold = 0)
```

Arguments

`trial`	A pr_trial object.
`threshold`	Numeric. Sensors at or below this value are treated as unloaded and excluded. Default `0`.

Value

Numeric vector of length `n_frames`.

pr_calc_mean_pressure_grid *Whole-Grid Mean Pressure Per Frame*

Description

Mean pressure across the entire sensor grid for each frame: the frame sum divided by the number of sensors, with unloaded cells counted as zeros.

Usage

```
pr_calc_mean_pressure_grid(trial)
```

Arguments

`trial`	A pr_trial object.

Details

This is deliberately *not* the same quantity as `pr_calc_mean_pressure()`, which averages only the cells above `threshold` (loaded-cells-only, by definition) and is left untouched. On a saddle mat where a typical frame loads roughly 4% of the 256 cells, the loaded-cells-only mean is around 25 times the whole-grid mean, so the two are never interchangeable. Use this one when the mat itself is the region of interest and an unloaded cell is a real "no pressure here" measurement; use `pr_calc_mean_pressure()` when the sensor sheet is larger than the loaded object and only the contact patch is meaningful.

The denominator is `trial$layout$n_sensors`, i.e. the number of *active* sensors, which equals `ncol(trial$pressure)`. Inactive grid cells are not measurements and never enter the average.

Value

Numeric vector of length `n_frames`, in the trial's pressure unit.

See Also

Other whole-grid frame metrics: `pr_calc_cop_grid()`, `pr_calc_loaded_count()`, `pr_calc_pci()`, `pr_calc_total_pressure()`, `pr_frame_metrics()`, `pr_ref_pci()`

Examples

```
trial <- pr_example_trial("saddle_horse")
grid_mean <- pr_calc_mean_pressure_grid(trial)
loaded_mean <- pr_calc_mean_pressure(trial)
# The whole-grid mean is the smaller of the two by construction:
all(grid_mean <= loaded_mean + 1e-9)
```

---

pr_calc_pci *Peak/Mean Pressure Concentration Index*

Description

Ratio of a recording's average frame peak to its average frame mean. A PCI of 1 would mean a perfectly uniform mat; the larger the value, the more of the load sits under a single hotspot.

Usage

```
pr_calc_pci(trial, denominator = c("grid", "loaded"), eps = 1e-12)
```

Arguments

`trial`	A pr_trial object.
`denominator`	Character. `"grid"` (default) or `"loaded"`; see Details.
`eps`	Numeric. Added to the mean-pressure denominator. Default `1e-12`.

Details

`PCI = mean(peak_kPa) / mean(mean_kPa)` — a ratio of two per-recording averages, not the average of a per-frame ratio; the two differ whenever load varies over time, and the cohort summaries use the former.

`denominator` selects which mean goes underneath:

- `"grid"` (default) uses `pr_calc_mean_pressure_grid()`, the whole-grid mean including unloaded cells. This is the cohort definition and the one whose interpretation bands `pr_ref_pci()` documents.
- `"loaded"` uses the loaded-cells-only mean of `pr_calc_mean_pressure()`, which is a different quantity — that function is loaded-cells-only by definition and is untouched here. Values come out roughly an order of magnitude smaller on a sparsely loaded mat, and `pr_ref_pci()` does **not** apply to them.

`eps` guards the denominator so a recording with no load at all returns `0` instead of `NaN`.

Value

A single numeric value (dimensionless ratio).

See Also

Other whole-grid frame metrics: `pr_calc_cop_grid()`, `pr_calc_loaded_count()`, `pr_calc_mean_pressure_grid()`, `pr_calc_total_pressure()`, `pr_frame_metrics()`, `pr_ref_pci()`

Examples

```
trial <- pr_example_trial("saddle_horse")
pr_calc_pci(trial)
# The loaded-cells-only denominator is larger, so the index is smaller:
pr_calc_pci(trial, denominator = "loaded") < pr_calc_pci(trial)
```

---

pr_calc_peak_pressure *Calculate Peak Pressure Per Frame*

Description

Returns the maximum pressure across all active sensors for each frame.

Usage

```
pr_calc_peak_pressure(trial, threshold = 0)
```

Arguments

`trial`	A pr_trial object.
`threshold`	Numeric. Sensors at or below this value are treated as unloaded and excluded. Default `0`.

Value

Numeric vector of length `n_frames`.

Examples

```
trial <- pr_example_trial("insole")
pp <- pr_calc_peak_pressure(trial)
length(pp) == trial$n_frames
```

`pr_calc_phase_map` *Sensor Map Averaged Over the Stride Cycle*

Description

Resamples the whole sensor map — all 256 cells on a saddle mat — onto percent-of-cycle: for each phase bin, the mean pressure every sensor carried while the stride was in that part of its cycle. This is the ensemble average an animation or a phase-by-phase heatmap series is drawn from.

Usage

```
pr_calc_phase_map(
  trial,
  cycles = NULL,
  n_bins = 10L,
  trim = 0.1,
  min_frames = 10L
)
```

Arguments

`trial`	A pr_trial object.
`cycles`	A cycle table from `pr_calc_stride_cycles()`, or `NULL` (default) to detect cycles with that function's defaults. Any data frame with `start_idx` and `end_idx` columns is accepted.
`n_bins`	Integer. Number of phase bins spanning 0-100% of the cycle. Default `10L`.
`trim`	Numeric in `[0, 0.5)`. Fraction trimmed from each end when averaging across cycles. Default `0.1`.
`min_frames`	Integer. Cycles with fewer frames than this are skipped. Default `10L`.

Details

Each cycle from `pr_calc_stride_cycles()` is stretched onto 0-100% of itself — frame `i` of a cycle spanning `m` frames sits at `(i - 1) / (m - 1) * 100` percent — and its frames are dropped into `n_bins` equal-width bins, with 100% falling in the last bin. Every sensor is averaged within a bin, giving one map per bin per cycle, and those per-cycle maps are then averaged across cycles.

Averaging per cycle first, rather than pooling all frames, weights every stride equally. Pooling would let a long stride, which contributes more frames, dominate the ensemble; strides in a recording vary in length by a factor of three or more, so the difference is real.

`trim` is passed to `mean()` as it averages across cycles within a bin, so `trim = 0.1` discards the highest and lowest 10% of cycles in each bin and each cell before averaging. That protects the ensemble from a single stride where the horse stumbled or the rider shifted, at the cost of ignoring genuine extremes. `trim = 0` gives the plain mean.

Cycles shorter than `min_frames` frames are skipped: too few frames make the phase bins mostly empty, and an empty bin contributes nothing rather than a zero. A bin no cycle ever reached comes back `NA`.

Value

A tibble::tibble with `n_bins * n_sensors` rows, ordered by `phase_bin` then sensor, and columns `phase_bin` (integer, 1 to `n_bins`), `sensor`, `row`, `col` (integer, as in `pr_sensor_map()`) and `mean_kPa`. Zero rows, with a warning, when no cycle qualifies.

Agreement with the source study

The study's phase maps used the plain mean across cycles (`trim = 0`). On the reference recording `ID052_K_K_KG00_MS`, and on the same 14 cycles, this function at `trim = 0` reproduces all 2,560 of its values with a correlation of 0.99998, a mean absolute difference of 0.007 kPa and a maximum of 0.12 kPa, on values ranging up to 6.9 kPa. Close, but not exact: the residual difference has not been traced to any documented step, so treat the two as equivalent in aggregate rather than interchangeable cell by cell. At the default `trim = 0.1` the same comparison gives 0.011 kPa mean and 0.18 kPa maximum, the extra difference being the trimming itself.

See Also

`pr_sensor_map()` for the same map reduced over the whole recording instead of over the cycle.

Other temporal structure functions: `pr_calc_dominant_freq()`, `pr_calc_spectrum()`, `pr_calc_stride_cycles()`, `pr_cop_shape()`

Examples

```
trial <- pr_example_trial("saddle_horse")
pm <- pr_calc_phase_map(trial)
nrow(pm) == 10 * trial$n_sensors

# Where the load sits at the start of the cycle versus its middle:
first <- pm$mean_kPa[pm$phase_bin == 1]
mid <- pm$mean_kPa[pm$phase_bin == 5]
round(c(bin1 = max(first), bin5 = max(mid)), 2)

# Averaged over all bins, the phase map returns the recording's own
# per-sensor mean to within the trimming:
avg <- tapply(pm$mean_kPa, pm$sensor, mean)
round(stats::cor(as.numeric(avg), pr_sensor_map(trial, "mean")$value), 3)
```

pr_calc_pti *Pressure-Time Integral Per Sensor*

Description

Integrates pressure over time for each sensor using the trapezoidal rule. Result is in kPa*s.

Usage

```
pr_calc_pti(trial)
```

Arguments

trial	A pr_trial object.

Value

Numeric vector of length n_sensors.

pr_calc_regional *Compute Parameters by Region*

Description

Applies region masks to a trial and computes summary parameters independently for each region. Returns one row per region.

Usage

```
pr_calc_regional(
  trial,
  masks = NULL,
  parameters = c("mpp", "mvp", "max_force", "contact_area", "pti_mean"),
  threshold = 0
)
```

Arguments

trial	A pr_trial object.
masks	Named list of pr_mask objects or logical matrices. If NULL (default), uses the layout's default regions.
parameters	Character vector. Parameters to compute. Supported values: "mpp", "mvp", "max_force", "mean_force", "contact_area", "pti_max", "pti_mean". Default c("mpp", "mvp", "max_force", "contact_area", "pti_mean").
threshold	Numeric. Default 0.

Value

A tibble::tibble with one column `region` plus one column per requested parameter.

Examples

```
trial <- pr_example_trial("saddle_horse")
pr_calc_regional(trial)
```

`pr_calc_regional_map` *Zone Statistics of the Time-Averaged Sensor Map*

Description

Summarises each region mask on the **per-sensor map**: every sensor is first reduced over its own frames, and the zone statistics are then taken across the sensors of the zone.

Usage

```
pr_calc_regional_map(
  trial,
  masks = NULL,
  statistic = c("mean", "max", "loaded"),
  threshold = 0
)
```

Arguments

`trial`	A pr_trial object.
`masks`	Named list of pr_mask objects or logical matrices. `NULL` (default) uses the layout's own regions; a layout without regions is an error, since a zone table has to say which zones.
`statistic`	Character. The per-sensor reduction over time: `"mean"` (default) the time-average, `"max"` the maximum pressure picture, or `"loaded"` the fraction of frames above `threshold`.
`threshold`	Numeric. Cells at or below this value count as unloaded. Default `0`.

Value

A tibble::tibble with one row per mask and columns `zone`, `zone_mean_kPa`, `zone_peak_kPa`, `zone_loaded`.

Operator order

This is the opposite order to `pr_calc_regional()`, and the two give different numbers on purpose.

- `pr_calc_regional()` reduces **over sensors within a frame**, then over frames. Its `mpp` is the largest single reading the zone ever produced, in any frame.
- `pr_calc_regional_map()` reduces **over frames within a sensor**, then over sensors. Its `zone_peak_kPa` is the largest *time-averaged* cell, which is smaller — often several-fold — because no cell holds its maximum for a whole recording.

Neither is more correct; they answer different questions. Use `pr_calc_regional()` to report what the worst moment looked like, and this function to report which cells carry load over the ride, which is the quantity a cohort-level zone table compares between horses.

Columns

Writing `v` for the per-sensor map and `Z` for a zone's sensors:

- `zone_mean_kPa` — `mean(v[Z])`, over **all** the zone's sensors, with never-loaded ones contributing zeros. The mat is the region of interest, so an unloaded cell is a real measurement (compare `pr_calc_mean_pressure_grid()`).
- `zone_peak_kPa` — `max(v[Z])`.
- `zone_loaded` — `sum(v[Z] > 0)`, the number of the zone's sensors that ever carried load above `threshold`.

A zone with no sensors at all returns three zeros rather than `NaN`.

With `statistic = "loaded"` the map is a duty cycle in `[0, 1]`, not a pressure, and the first two columns carry fractions; the column names are kept fixed so a cohort table has one schema.

See Also

`pr_sensor_map()` for the map being reduced, `pr_calc_regional()` for the frame-first counterpart.

Other regional symmetry functions: `pr_calc_cop_masked()`, `pr_calc_symmetry_map()`, `pr_mask_mirror_balance()`, `pr_mask_rowbands()`, `pr_symmetry_sensitivity()`

Examples

```
trial <- pr_example_trial("saddle_horse")
pr_calc_regional_map(trial)

# The sensor-first peak is never above the frame-first peak:
zm <- pr_calc_regional_map(trial)
zf <- pr_calc_regional(trial, parameters = "mpp")
all(zm$zone_peak_kPa <= zf$mpp + 1e-9)

# An explicit band split, rather than the layout's own regions:
pr_calc_regional_map(trial, pr_mask_rowbands(trial$layout, sides = FALSE))
```

pr_calc_rollover *Analyze COP Rollover Pattern*

Description

Computes the center of pressure progression during the stance phase of each gait cycle, resampled to percent of stance (0-100).

Usage

```
pr_calc_rollover(trial, cycles = NULL, n_points = 51L)
```

Arguments

trial	A pr_trial object.
cycles	Output of pr_calc_gait_cycles(). If NULL, cycles are detected automatically.
n_points	Integer. Number of points to sample along each stance phase. Default 51.

Value

A tibble::tibble with columns cycle, percent_stance, cop_x_mm, cop_y_mm, cop_velocity.

pr_calc_saddle_bridge *Detect Saddle Bridge Formation*

Description

Identifies "bridging" — a saddle fit problem where the middle region is unloaded while cranial and caudal regions carry excessive pressure. Bridging is quantified as the ratio of middle-region pressure to the mean of cranial and caudal pressures.

Usage

```
pr_calc_saddle_bridge(trial, masks = NULL, bridge_threshold = 0.5)
```

Arguments

trial	A pr_trial from a horse saddle recording.
masks	Named list of 6 region masks (from pr_mask_saddle_6()). If NULL, generated from the trial's layout.
bridge_threshold	Numeric. Middle/ends ratio below this value indicates bridging. Default 0.5.

Value

A list with class `pr_saddle_bridge`:

- `bridge_ratio`: middle / mean(cranial, caudal) MVP ratio.
- `is_bridged`: logical.
- `regional_pressures`: tibble with MVP per region.
- `recommendation`: character.

Examples

```
pr_calc_saddle_bridge(pr_example_trial("saddle_horse"))
```

---

pr_calc_saddle_slip *Detect Saddle Slip / Asymmetric Loading*

Description

Analyzes left/right pressure asymmetry over time to detect saddle slip or systematic asymmetric loading.

Usage

```
pr_calc_saddle_slip(trial, masks = NULL, slip_threshold = 15)
```

Arguments

`trial`	A pr_trial object.
`masks`	Named list of region masks. If `NULL`, auto-generated via `pr_mask_symmetry()`.
`slip_threshold`	Numeric. SI% above which asymmetry is flagged. Default 15.

Value

A list with class `pr_saddle_slip`.

Examples

```
pr_calc_saddle_slip(pr_example_trial("saddle_horse"))
```

pr_calc_seat_hotspot *Identify Pressure Hotspots*

Description

Finds sensors exceeding a clinical pressure threshold — relevant for pressure ulcer risk assessment in wheelchair seating.

Usage

```
pr_calc_seat_hotspot(trial, threshold = 4.7, duration_s = 0)
```

Arguments

`trial`	A pr_trial from a seating recording.
`threshold`	Numeric. Pressure threshold (kPa) above which a sensor is considered a hotspot. Default `4.7` (equivalent to 32 mmHg, the capillary closing pressure).
`duration_s`	Numeric. Minimum sustained duration (s) above threshold for a hotspot to be flagged. Default `0`.

Value

A tibble::tibble with columns `sensor_id`, `row`, `col`, `x_mm`, `y_mm`, `max_pressure`, `mean_pressure`, `duration_above_s`.

Examples

```
pr_calc_seat_hotspot(pr_example_trial("wheelchair"))
```

pr_calc_spectrum *Power Spectrum of a Frame-Level Signal*

Description

Frequency content of a whole-mat time series: how much of the recording's variation sits at each frequency. On a saddle mat the stride shows up as a peak somewhere between 0.5 and 2 Hz.

Usage

```
pr_calc_spectrum(
  trial,
  signal = c("total", "mean", "peak"),
  window = c("hann", "none"),
  detrend = TRUE
)
```

Arguments

`trial`	A pr_trial object.
`signal`	Character. Which frame-level signal to transform: `"total"` (default), `"mean"` or `"peak"`. See *Details*.
`window`	Character. `"hann"` (default) or `"none"`.
`detrend`	Logical. Subtract the signal mean first. Default `TRUE`.

Details

The signal is taken from the pressure matrix directly:

- `"total"` — the frame sum, `pr_calc_total_pressure()`.
- `"mean"` — the whole-grid frame mean, `pr_calc_mean_pressure_grid()`. This is `"total"` divided by the constant `n_sensors`, so its spectrum is the `"total"` spectrum scaled by `1 / n_sensors^2` and the peak sits at exactly the same frequency.
- `"peak"` — the frame maximum, which is *not* a scaled copy of the other two and saturates flat on a device that clips at its ceiling.

`detrend = TRUE` subtracts the mean before transforming, which removes the DC term that would otherwise dominate a signal sitting on a large constant offset. It does not remove a linear ramp. `window = "hann"` then tapers the ends, trading a little frequency resolution for far less leakage from the strong low-frequency components a drifting baseline produces; `window = "none"` leaves the segment rectangular.

Power is `Mod(fft(x))^2 / n`, kept for the non-negative frequencies only (`n %/% 2 + 1` values, including DC and — for even `n` — Nyquist). It is a squared pressure quantity in the device's own unit, not a density: it is not divided by the frequency bin width, and it is not corrected for the window's power loss, so absolute values are comparable only between recordings of the same length analysed the same way.

Value

A tibble::tibble with `n %/% 2 + 1` rows and columns `freq_Hz` (ascending, starting at 0) and `power`.

Frequency axis

The axis is built as `(0:(n %/% 2)) * fs / n`, which is the exact centre frequency of FFT bin `k` for both odd and even `n`.

The study this function reproduces built it as `seq(0, fs / 2, length.out = length(power))` instead. For even `n` the two agree exactly. For odd `n` they do not: that sequence has spacing `fs / (n - 1)` rather than `fs / n`, and it places its last value at `fs / 2`, which for odd `n` is not an FFT bin at all — the highest real bin is at `((n - 1) / 2) * fs / n`, just below Nyquist. On the 869-frame reference recording the error is a uniform stretch of `n / (n - 1)`, about 0.12%: the study reports a stride at 0.7488 Hz where the exact axis puts it at 0.7480 Hz. `power` is unaffected — only the labels move. **This function implements the exact axis.**

See Also

pr_calc_dominant_freq() for the peak of this spectrum, pr_calc_stride_cycles() for the time-domain counterpart.

Other temporal structure functions: pr_calc_dominant_freq(), pr_calc_phase_map(), pr_calc_stride_cycles(), pr_cop_shape()

Examples

```
trial <- pr_example_trial("saddle_horse")
spec <- pr_calc_spectrum(trial)
nrow(spec) == trial$n_frames %/% 2 + 1

# The strongest non-DC component is the stride:
round(spec$freq_Hz[which.max(spec$power[-1]) + 1], 3)

# "mean" is "total" scaled by n_sensors, so power scales by its square:
m <- pr_calc_spectrum(trial, signal = "mean")
all.equal(m$power * trial$layout$n_sensors^2, spec$power)
```

---

pr_calc_stride_cycles *Segment a Recording into Stride Cycles*

Description

Peak-to-peak segmentation of a never-unloading recording. The frame-level signal is band-passed to isolate the stride oscillation from the baseline the rider's mass sets, and each interval between two successive peaks of that oscillation is one cycle.

Usage

```
pr_calc_stride_cycles(
  trial,
  signal = "total",
  lp_window = 11L,
  hp_window_s = 5,
  min_period_s = 0.3,
  height_quantile = 0.3,
  max_period_s = 3
)
```

Arguments

trial	A pr_trial object.
signal	Character. Frame-level signal to segment. Default "total".
lp_window	Integer. Width in samples of the centred moving average. Default 11L. An odd width keeps the average centred; 1L disables the low pass.

`hp_window_s` Numeric. Width in seconds of the running-median baseline. Default `5`.

`min_period_s` Numeric. Shortest acceptable cycle, in seconds, and the minimum peak separation. Default `0.3`.

`height_quantile` Numeric in `[0, 1)`. Peak height threshold, as a quantile of the positive part of the band-passed signal. Default `0.3`.

`max_period_s` Numeric. Longest acceptable cycle, in seconds. Default `3`; an interval longer than this is a detection gap rather than a stride.

Details

This is the replacement for `pr_calc_gait_cycles()` on mats that stay loaded. That function pairs a threshold crossing up with the next crossing down; a saddle mat never crosses back down, so it returns a single "cycle" spanning the whole recording no matter how `force_threshold` is set. The problem is the segmentation model, not its tuning, so nothing about `pr_calc_gait_cycles()` is changed here.

The pipeline, in order:

1. **Signal.** `signal` selects the frame-level series exactly as in `pr_calc_spectrum()`. `"total"` and `"mean"` differ by the constant `n_sensors`, so they give identical segmentation; only `peak_total` changes scale.
2. **Low pass.** An `lp_window`-sample centred moving average. Frames at the two ends, where the window overhangs the recording, keep their raw value rather than becoming `NA`, so the first and last strides are still detectable.
3. **High pass.** Subtract a running median over `hp_window_s` seconds. The median steps over the stride peaks instead of being pulled up by them, so the residual keeps its amplitude while the baseline drift — the rider settling, the horse changing gait — is removed.
4. **Peaks.** Strict local maxima of the residual at or above `stats::quantile(residual[residual > 0], height_quantile)`, thinned tallest-first so no two peaks sit closer than `round(min_period_s * fs)` samples.
5. **Cycles.** Every consecutive pair of peaks, keeping those whose duration lies in `[min_period_s, max_period_s]`.

`cycle` numbers the peak pairs **before** the duration filter, so a gap in the sequence marks an interval that was rejected as too short or too long — information a renumbered column would hide.

`peak_total` is the band-passed amplitude at the cycle's opening peak, in the device's pressure unit. It is an excursion above the local baseline, not a raw frame total: on the reference recording the frame totals run 383-434 kPa while `peak_total` runs 2.5-13.8 kPa.

Value

A tibble::tibble with one row per accepted cycle and columns `cycle` (integer, gapped — see *Details*), `start_idx`, `end_idx` (integer frame indices of the bounding peaks), `start_s`, `end_s`, `duration_s` (seconds), `peak_total` (band-passed amplitude at `start_idx`) and `stride_freq` (`1 / duration_s`, Hz). Zero rows when fewer than two peaks survive.

Agreement with the source study

On the 869-frame reference recording `ID052_K_K_KG00_MS`, the defaults reproduce the study's own segmentation exactly: 14 cycles, identical `cycle`, `start_idx`, `end_idx`, `duration_s`, `peak_total` and `stride_freq` to machine precision. Rerun over the study's whole cohort, all 323 of its 50 Hz recordings match cycle for cycle — 137,761 cycles, no differences.

The remaining 108 recordings were sampled at 10 Hz, and there the two disagree. The study hard-coded its running-median width at 251 samples and its minimum peak separation at 15 samples, which are 5 s and 0.3 s only at 50 Hz; the parameters here are in seconds and scale with the trial's own sampling rate. Passing `hp_window_s = 25.1` and `min_period_s = 1.5` reproduces the study's fixed sample counts on a 10 Hz recording.

That difference shows up in the published cohort means. All 34,095 of the study's slow-gait (`MG`) cycles come from 50 Hz recordings, and this function returns their published mean duration of 0.719 s exactly. The `MH` and `MS` means, 0.748 s and 0.773 s, mix in 10 Hz recordings; on the 50 Hz half alone both this function and the study give 0.650 s and 0.726 s.

See Also

`pr_calc_dominant_freq()` for the frequency-domain estimate of the same rhythm, `pr_calc_phase_map()` to average the sensor map over these cycles.

Other temporal structure functions: `pr_calc_dominant_freq()`, `pr_calc_phase_map()`, `pr_calc_spectrum()`, `pr_cop_shape()`

Examples

```
trial <- pr_example_trial("saddle_horse")
cycles <- pr_calc_stride_cycles(trial)
nrow(cycles)
round(range(cycles$duration_s), 2)

# stride_freq is the reciprocal of duration_s, cycle by cycle:
all.equal(cycles$stride_freq, 1 / cycles$duration_s)

# Cycles tile the recording end to end: each one starts where the
# previous accepted one stopped, unless an interval was rejected.
head(cbind(cycles$start_idx, cycles$end_idx), 3)
```

---

pr_calc_symmetry_index

Symmetry Index

Description

Computes the symmetry index (SI%) between left and right sides of a layout for a given parameter. `SI = (L - R) / (0.5 * (L + R)) * 100`.

Usage

```
pr_calc_symmetry_index(
  trial,
  parameter = c("peak_pressure", "mean_pressure", "force", "contact_area")
)
```

Arguments

`trial`	A pr_trial object.
`parameter`	Character. One of `"peak_pressure"`, `"mean_pressure"`, `"force"`, `"contact_area"`. Default `"peak_pressure"`.

Value

Numeric scalar. Positive values indicate left dominance.

Examples

```
pr_calc_symmetry_index(pr_example_trial("saddle_horse"))
```

---

pr_calc_symmetry_map *Left/Right Asymmetry of the Time-Averaged Sensor Map*

Description

The classic asymmetry index `(L - R) / (0.5 * (L + R)) * 100`, computed on the per-sensor map rather than frame by frame. Negative values are right-biased, positive values left-biased, and zero is symmetric.

Usage

```
pr_calc_symmetry_map(
  trial,
  masks = NULL,
  statistic = "mean",
  denominator = c("grid", "loaded")
)
```

Arguments

`trial`	A pr_trial object.
`masks`	What to compare. `NULL` (default) uses the whole active grid, split at `floor(grid_cols / 2)` exactly as `pr_mask_symmetry()` does. A single pr_mask or logical matrix restricts that comparison to a region of interest, split at the same midline. A list of exactly two masks gives the two sides explicitly, left first.
`statistic`	Character. `"mean"` (default), `"max"` or `"loaded"`; see `pr_calc_regional_map()`.
`denominator`	Character. `"grid"` (default) or `"loaded"`; see *Details*.

Details

`L` and `R` are the mean of the per-sensor map over the left and right sensors of the region of interest. Because the map is taken first, the index describes where load sat *over the whole recording*, not where the loudest frame happened to be; it is the counterpart of the frozen `pr_calc_symmetry_index()`, which averages within frames.

`denominator` chooses what goes into each side's mean:

- `"grid"` (default) averages over **every** sensor of the side, counting never-loaded ones as zeros. This is the cohort definition: on a mat that *is* the region of interest, an unloaded cell is a real measurement.
- `"loaded"` averages only the sensors whose map value is above zero. That answers a different question — how hard the loaded cells were pressed, ignoring how many there were — and is usually much closer to zero, because a side that loses contact area keeps its intensity.

The index is undefined when both sides are zero; that case returns `0`, as `pr_calc_symmetry_index()` does.

Nothing here forces the two sides to hold the same number of sensors. When they do not, part of the index is a property of the mask — see `pr_mask_mirror_balance()` and `pr_symmetry_sensitivity()`.

Value

A one-row tibble::tibble with columns `statistic`, `denominator`, `n_left`, `n_right`, `left_value`, `right_value` and `asymmetry_pct`.

See Also

`pr_calc_symmetry_index()` for the frame-first version, `pr_symmetry_sensitivity()` to vary the mask.

Other regional symmetry functions: `pr_calc_cop_masked()`, `pr_calc_regional_map()`, `pr_mask_mirror_balance()`, `pr_mask_rowbands()`, `pr_symmetry_sensitivity()`

Examples

```
trial <- pr_example_trial("saddle_horse")
pr_calc_symmetry_map(trial)

# Restricted to the cranial band only:
bands <- pr_mask_rowbands(trial$layout, sides = FALSE)
pr_calc_symmetry_map(trial, bands$cranial)

# Intensity-only comparison, ignoring how much area each side loaded:
pr_calc_symmetry_map(trial, denominator = "loaded")$asymmetry_pct
```

pr_calc_total_pressure *Total Pressure Per Frame*

Description

Sum of all sensor readings in each frame.

Usage

```
pr_calc_total_pressure(trial)
```

Arguments

trial	A pr_trial object.

Details

The result is a sum of pressures (kPa), not a force. Converting it to Newtons requires a per-cell area; pr_calc_force() does that using layout$sensor_area_cm2. For devices whose cell pitch was never published, sensor_area_cm2 is NA and no force value is defensible — hence this raw kPa sum, which is what the cohort analyses use. It is the unscaled counterpart of pr_calc_mean_pressure_grid() (total / n_sensors).

Value

Numeric vector of length n_frames, summed pressure per frame.

See Also

Other whole-grid frame metrics: pr_calc_cop_grid(), pr_calc_loaded_count(), pr_calc_mean_pressure_grid(), pr_calc_pci(), pr_frame_metrics(), pr_ref_pci()

Examples

```
trial <- pr_example_trial("saddle_horse")
tot <- pr_calc_total_pressure(trial)
round(max(tot), 1)
```

pr_channel_order *Raw Device Channel Order for a Layout*

Description

Returns the integer permutation that reorders the columns of a raw device export into the column-major grid order every other pressR function assumes. A reader applies it once: `pressure <- raw[, pr_channel_order(layout), drop = FALSE]`.

Usage

```
pr_channel_order(layout_or_map)
```

Arguments

layout_or_map	A pr_layout built by pr_layout_from_index_map() (or pr_layout_saddle_novel()), a channel index matrix as accepted by pr_layout_from_index_map(), or an integer vector that already is such a permutation.

Details

Element k of the result is the *device channel* that belongs in pressure column k, where column k corresponds to `which(layout$active)[k]` — the same column-major ordering used by pr_mask() and `.layout_build_coords()`.

Value

An integer vector of length `layout$n_sensors`.

See Also

Other channel map functions: pr_layout_from_index_map(), pr_layout_saddle_novel()

Examples

```
pr_channel_order(rbind(c(2L, 1L), c(4L, 3L)))

ord <- pr_channel_order(pr_layout_saddle_novel())
length(ord)
ord[1:8]
```

pr_cop *Center of Pressure Object*

Description

Construct a pr_cop object from x/y coordinates and time. Normally created by pr_calc_cop() rather than called directly.

Usage

```
pr_cop(x, y, time)
```

Arguments

x	Numeric vector of COP x-coordinates (mm).
y	Numeric vector of COP y-coordinates (mm).
time	Numeric vector of timestamps (s).

Value

A pr_cop S3 object with fields x, y, time, path_length, velocity_mean, velocity_max, range_x, range_y, sway_area.

pr_cop_shape *Loop Geometry of a COP Trajectory*

Description

Describes a centre-of-pressure trajectory as a loop: how far its end lands from its start, and whether the path crosses itself on the way.

Usage

```
pr_cop_shape(cop, close = TRUE)
```

Arguments

cop	A pr_cop object from pr_calc_cop(), or a data frame with x/y columns, or one with cop_row/cop_col columns as returned by pr_calc_cop_grid() and pr_frame_metrics() (cop_col is treated as x, cop_row as y).
close	Logical. Append the first point to the end before looking for crossings. Default TRUE.

Details

A stride that returns the rider to where they began traces a closed loop, and how that loop is shaped separates a clean, repeatable seat from a wandering one. Two numbers capture most of it:

- `closure_dist` — the straight-line distance from the last point back to the first, in whatever units the trajectory carries. Small relative to the path travelled means the trajectory came home.
- `n_crossings` — how many times the path cuts through itself. A simple oval crosses zero times; a figure-of-eight crosses once; a trajectory that scribbles crosses many times. `self_intersects` is `n_crossings > 0`.

With `close = TRUE` (default) the first point is appended to the end before the crossing sweep, so the closing chord counts as part of the loop — the right question for a cycle that should repeat. With `close = FALSE` only the path as travelled is examined. Either way `closure_dist` and `path_length` describe the open trajectory.

Crossings are counted with orientation tests on every pair of non-adjacent segments, an exact predicate with no tolerance to tune. Two segments that merely touch end to end, or that overlap along a line, are not counted; only a genuine transverse crossing is. The sweep is quadratic in the number of points, which is immaterial for one stride and worth remembering before feeding it a whole recording.

Units are the input's own and nothing is converted: a `pr_cop` from `pr_calc_cop()` is in millimetres, while `cop_row`/`cop_col` from `pr_calc_cop_grid()` or `pr_frame_metrics()` are in grid index units, so `path_length` and `closure_dist` follow suit. `closure_ratio` is dimensionless and comparable across both.

Missing coordinates are dropped pairwise before anything is measured — `pr_calc_cop()` returns `NA` for unloaded frames.

Value

A one-row tibble::tibble with columns `n_points` (integer, after dropping missing coordinates), `path_length`, `closure_dist`, `closure_ratio` (`closure_dist / path_length`, `NA` for a stationary trajectory), `self_intersects` (logical) and `n_crossings` (integer).

See Also

`pr_calc_cop_grid()` and `pr_frame_metrics()` for trajectories in grid units, `pr_calc_cop()` for millimetres.

Other temporal structure functions: `pr_calc_dominant_freq()`, `pr_calc_phase_map()`, `pr_calc_spectrum()`, `pr_calc_stride_cycles()`

Examples

```
# A clean rectangle: comes home, never crosses itself.
square <- data.frame(x = c(0, 1, 1, 0), y = c(0, 0, 1, 1))
pr_cop_shape(square)

# A bow tie crosses once.
bow <- data.frame(x = c(0, 1, 0, 1), y = c(0, 0, 1, 1))
```

```
pr_cop_shape(bow)$n_crossings

# One stride of a real trajectory, in grid index units:
trial <- pr_example_trial("saddle_horse")
cycles <- pr_calc_stride_cycles(trial)
cop <- pr_calc_cop_grid(trial)
pr_cop_shape(cop[cycles$start_idx[1]:cycles$end_idx[1], ])
```

`pr_dataset` *Create a Pressure Dataset*

Description

A `pr_dataset` is a collection of pr_trial objects with shared grouping metadata, intended for batch analysis.

Usage

```
pr_dataset(trials, group_var = "condition", name = "dataset")
```

Arguments

`trials`	A list of pr_trial objects.
`group_var`	Character. Name of the metadata field to use as the grouping variable. Default `"condition"`.
`name`	Character. Name of the dataset. Default `"dataset"`.

Value

A `pr_dataset` S3 object.

Examples

```
t1 <- pr_example_trial("insole", seed = 1)
t2 <- pr_example_trial("insole", seed = 2)
ds <- pr_dataset(list(t1, t2))
print(ds)
```

pr_dataset_filter *Filter a Dataset by Trial Metadata, With Count Assertions*

Description

Keeps the trials whose metadata satisfy a predicate, records the filter label on the returned dataset, and — when `n_units` or `n_subjects` are given — asserts that the surviving cohort has exactly the expected size. The assertion is the point: a cohort step that cannot fail cannot protect a result.

Usage

```
pr_dataset_filter(
  dataset,
  f,
  label = NULL,
  n_units = NULL,
  n_subjects = NULL,
  subject_field = "ID"
)
```

Arguments

`dataset`	A pr_dataset object, or a list of pr_trial objects.
`f`	A predicate applied to each trial's `metadata` list. Either a function of one argument, or a one-sided formula such as `~ .x$Mode == "MS"`. It must return a single `TRUE`/`FALSE` per trial; `NA` is treated as `FALSE` with a warning.
`label`	Character. Short name for this filter, stored on the result as `filter_label` and appended to `filter_history`, and used in assertion messages. Default `NULL` uses `"filter"`.
`n_units`	Integer. Expected number of trials after filtering, or `NULL` (default) for no assertion.
`n_subjects`	Integer. Expected number of distinct subjects after filtering, or `NULL` (default) for no assertion.
`subject_field`	Character. Metadata field identifying the subject. Default `"ID"`.

Value

A pr_dataset containing the surviving trials, with extra elements `filter_label` and `filter_history`.

See Also

Other cohort functions: `pr_assert_cohort()`, `pr_design_table()`, `pr_validate_dataset()`

Examples

```
layout <- pr_layout_mat("16")
mk <- function(id, mode) {
  pr_trial(matrix(1, 3, layout$n_sensors), time = c(0, 0.02, 0.04),
           layout = layout, metadata = list(ID = id, Mode = mode))
}
ds <- pr_dataset(list(mk("ID001", "MS"), mk("ID001", "MH"),
                      mk("ID002", "MS")))
walk <- pr_dataset_filter(ds, ~ .x$Mode == "MS", label = "walk",
                          n_units = 2, n_subjects = 2)
walk$filter_label
```

`pr_design_gaps` *Observed Design Cells Against the Full Crossing*

Description

Crosses the levels of the named factors, counts how many rows fall in each cell, and labels every cell of the crossing as missing, under-represented, balanced, or over-represented. This is what surfaces a nesting or aliasing problem before anything is modelled.

Usage

```
pr_design_gaps(x, factors, expected = NULL)
```

Arguments

`x`	A data frame.
`factors`	Character vector of column names to cross, in the order they should nest.
`expected`	Single non-negative whole number: the replicate count a complete cell should hold. `NULL` (default) uses the modal non-zero observed count.

Details

A design described as "all saddles crossed with all pads" can turn out to be nothing of the sort: one pad may only ever have been used with one saddle, in which case the two factors are aliased and their effects are not separately estimable — a fact no model summary will state out loud. Counting the cells of the full crossing shows it immediately: a factor nested inside another leaves whole blocks of the crossing at `n == 0`.

Levels come from `levels()` for a factor column and from the sorted unique non-`NA` values otherwise. Sorting uses `method = "radix"`, which forces C-locale collation, so the row order of the result does not depend on the machine's locale. The crossing is generated by `tidyr::expand_grid()`, so the first factor varies slowest and the row order is the natural nesting order.

`expected` sets the replicate count a balanced cell should hold. Left `NULL`, it is the most common non-zero cell count — the design's own idea of a full cell — with ties broken towards the larger count, so a tie errs towards reporting cells as under-represented rather than as balanced.

Rows with `NA` in any crossing factor cannot be placed in a cell. They are dropped and a warning names how many, because dropping them silently would understate the gaps.

Value

A tibble::tibble with one row per cell of the full crossing: the factor columns (as character), then `n` (rows observed), `expected`, `delta` (`n - expected`) and `status`, one of `"missing"` (`n == 0`), `"under"`, `"ok"` or `"over"`.

See Also

`pr_factor_contract()` for the level sets being crossed, `pr_design_table()` for building `x` from a pr_dataset.

Other provenance functions: `pr_factor_contract()`, `pr_lock_table()`, `pr_validate_summary()`, `pr_verify_lock()`

Examples

```
design <- data.frame(
  Saddle = c("K", "K", "K", "S", "S", "W"),
  Pad = c("F", "K", "K", "F", "F", "W")
)
gaps <- pr_design_gaps(design, c("Saddle", "Pad"))
gaps

# Pad is nested within Saddle, not crossed with it:
sum(gaps$status == "missing")

# With an explicit replicate expectation:
pr_design_gaps(design, c("Saddle", "Pad"), expected = 2)
```

`pr_design_table` *Cohort Design Table*

Description

Reduces a pr_dataset to one row per trial: the chosen metadata fields plus `n_frames` and `duration_s`. This is the study design as a table — the object you count, cross-tabulate, and assert against with `pr_assert_cohort()`.

Usage

```
pr_design_table(dataset, fields = NULL)
```

Arguments

`dataset`	A pr_dataset object, or a list of pr_trial objects.
`fields`	Character vector of metadata field names to include, in the order given. `NULL` (default) auto-detects: every metadata field that holds a single value in every trial, dropping fields that are missing (`NA`) throughout — which removes the empty `pr_trial` defaults such as `notes`. Fields named `n_frames` or `duration_s` are dropped in auto mode and rejected when named explicitly, because both columns are recomputed from the trial itself.

Details

`duration_s` is the trial's `duration` field, i.e. `diff(range(time))`, so a 23432-frame recording at 50 Hz gives 468.62 s (not 468.64 s).

Value

A tibble::tibble with one row per trial and columns `fields`, `n_frames` (integer) and `duration_s` (numeric).

See Also

Other cohort functions: `pr_assert_cohort()`, `pr_dataset_filter()`, `pr_validate_dataset()`

Examples

```
layout <- pr_layout_mat("16")
mk <- function(id, mode, n) {
  pr_trial(
    matrix(1, n, layout$n_sensors),
    time = seq(0, (n - 1) / 50, by = 1 / 50),
    layout = layout,
    metadata = list(ID = id, Mode = mode)
  )
}
ds <- pr_dataset(list(mk("ID001", "MS", 4), mk("ID002", "MH", 6)))
pr_design_table(ds, fields = c("ID", "Mode"))
```

`pr_downsample` *Downsample a Trial*

Description

Keeps every `factor`-th frame.

Usage

```
pr_downsample(trial, factor = 2L)
```

Arguments

`trial`	A pr_trial object.
`factor`	Integer. Downsample factor. Default `2`.

Value

A pr_trial object with reduced frame count.

Examples

```
tr <- pr_downsample(pr_example_trial("insole"), factor = 2)
tr$n_frames
```

`pr_example_files` *Write Sample Pressure Data Files*

Description

Creates sample ASCII files in a temporary directory for demonstrating the parser functions.

Usage

```
pr_example_files(type = c("insole", "saddle", "platform", "all"))
```

Arguments

`type`	Character. `"insole"` (default), `"saddle"`, `"platform"`, or `"all"`.

Value

Character. Path to the sample file (single type) or directory (`"all"`).

Examples

```
path <- pr_example_files("insole")
file.exists(path)
```

`pr_example_trial` *Generate a Synthetic Pressure Trial*

Description

Creates a pr_trial with realistic synthetic pressure data for demonstration, testing, and vignette purposes.

Usage

```
pr_example_trial(
  type = c("insole", "platform", "saddle_horse", "saddle_bicycle", "wheelchair",
    "custom"),
  duration_s = NULL,
  sampling_hz = 50,
  seed = 42
)
```

Arguments

type	Character. Type of trial to generate: "insole", "platform", "saddle_horse", "saddle_bicycle", "wheelchair", or "custom" (small uniform grid).
duration_s	Numeric. Trial duration in seconds. NULL (default) uses a type-specific default.
sampling_hz	Numeric. Sampling rate. Default 50.
seed	Integer. Random seed for reproducibility. Default 42.

Value

A pr_trial object.

Examples

```
trial <- pr_example_trial("insole")
print(trial)
```

---

pr_export_csv *Export Analysis Results to CSV*

Description

Export Analysis Results to CSV

Usage

```
pr_export_csv(
  trial,
  path,
  what = c("summary", "regional", "pressure", "cop"),
  masks = NULL
)
```

Arguments

trial	A pr_trial object.
path	Character. Output file path.
what	Character. "summary" (default), "regional", "pressure", or "cop".
masks	Named list of masks. Required for what = "regional".

Value

Invisibly returns the written file path.

Examples

```
tmp <- tempfile(fileext = ".csv")
pr_export_csv(pr_example_trial("insole"), tmp, what = "summary")
```

`pr_export_report` *Generate an Analysis Report*

Description

Renders a parameterized R Markdown template into an HTML or PDF report for a trial.

Usage

```
pr_export_report(
  trial,
  output_file,
  format = c("html", "pdf"),
  template = c("generic", "saddle", "foot"),
  masks = NULL,
  thresholds = NULL
)
```

Arguments

`trial`	A pr_trial object.
`output_file`	Character. Output file path.
`format`	Character. `"html"` (default) or `"pdf"`.
`template`	Character. `"generic"` (default), `"saddle"`, or `"foot"`.
`masks`	Named list of masks. `NULL` uses the layout defaults.
`thresholds`	Optional reference threshold tibble.

Value

Invisibly returns the generated file path.

Examples

```
## Not run:
tmp <- tempfile(fileext = ".html")
pr_export_report(pr_example_trial("insole"), tmp, template = "foot")

## End(Not run)
```

`pr_factor_contract` *Assert the Level Sets of Design Factors*

Description

Checks that named columns of a data frame carry exactly the levels they are contracted to carry — no unexpected value, no silently absent level, and for `factor` columns the declared order as well. Aborts naming the offending column, the offending values, and the rows they sit in.

Usage

```
pr_factor_contract(x, levels_list, id_cols = NULL)
```

Arguments

`x`	A data frame.
`levels_list`	Named list. Each name is a column of `x`; each element is the vector of levels that column must carry, in order. A `factor` may be supplied instead, in which case its `levels()` are used. Include `NA` to permit missing values in that column.
`id_cols`	Character vector of column names identifying a row, used to point at the offending rows in the error message (for example `"ID"`). `NULL` (default) reports values without row identifiers.

Details

This is the check that catches a typo'd condition code, a level lost to a filter, and a `factor()` call whose `levels =` argument drifted out of step with the data — all of which change a model's reference level or its contrast matrix without changing anything visible in a `head()`.

Both directions are enforced, because "exactly" is the point: a value in the data that the contract does not list is an `unexpected_level`, and a level in the contract that never appears in the data is a `missing_level`. The second half is the one that matters after a subset — a design that lost a whole condition still looks perfectly well formed.

Order is enforced only where order exists. A `factor` column stores its levels, so `levels(x[[col]])` must equal the contract element by element; a mismatch in order alone is reported as `wrong_order`. A `character` column has no stored order, so only its level *set* is contracted — convert with `factor(x, levels = ...)` first if the order has to be pinned.

`NA` values are reported as `missing_value` unless `NA` appears in the contract for that column, in which case they are accepted.

Value

Invisibly, `x` — so the contract can sit inside a pipeline. On violation, aborts with condition class `pr_factor_contract_failed`, carrying a `problems` tibble::tibble (columns `column`, `issue`, `value`, `n_rows`, `ids`) in the condition.

See Also

pr_design_gaps() for the combinations those levels do and do not form.

Other provenance functions: pr_design_gaps(), pr_lock_table(), pr_validate_summary(), pr_verify_lock()

Examples

```
design <- data.frame(
  ID = c("ID001", "ID001", "ID003", "ID003"),
  Mode = c("MG", "MH", "MS", "MH"),
  Saddle = c("K", "S", "W", "K")
)

# Passes and returns the data invisibly:
out <- pr_factor_contract(
  design,
  list(Mode = c("MG", "MH", "MS"), Saddle = c("K", "S", "W")),
  id_cols = "ID"
)
identical(out, design)

# A wrong contract fails loudly, naming the column and the values:
try(pr_factor_contract(design, list(Mode = c("MG", "MH"))))
```

---

pr_filter_time *Subset a Trial to a Time Window*

Description

Subset a Trial to a Time Window

Usage

```
pr_filter_time(trial, from = 0, to = Inf)
```

Arguments

trial	A pr_trial object.
from	Numeric. Start time (s). Default 0.
to	Numeric. End time (s). Default Inf.

Value

A pr_trial containing only frames with time in [from, to].

Examples

```
tr <- pr_filter_time(pr_example_trial("insole"), from = 1, to = 3)
tr$duration
```

pr_frame_metrics *Per-Frame Metric Table*

Description

Computes the whole-grid frame vocabulary — mean, peak, total, loaded cell count and grid-index centre of pressure — in a single vectorised pass over the pressure matrix.

Usage

```
pr_frame_metrics(trial, threshold = 0, eps = 1e-12)
```

Arguments

trial	A pr_trial object.
threshold	Numeric. Cells at or below this value count as unloaded. Default 0.
eps	Numeric. Added to the weight sum to keep unloaded frames finite. Default 1e-12.

Details

Every column is computed with whole-matrix operations — two rowSums(), one max.col(), and a single matrix product carrying both COP axes. There is no per-frame R loop, because cohort-scale inputs run to millions of frames. Measured on a 200,000 x 256 matrix (R 4.6.1, one core): about 145,000 frames per second, i.e. roughly 70 s for a 5.5 M-frame cohort, against 73-83 s for the equivalent vapply() over rows. The default threshold = 0 also skips masking the matrix entirely when no reading is negative, which saves a full copy of the pressure data.

Column definitions:

- mean_kPa — total_kPa / n_sensors, whole grid, zeros included. This is **not** pr_calc_mean_pressure(), which is loaded-cells-only by definition and is left unchanged; see pr_calc_mean_pressure_grid().
- peak_kPa — the plain row maximum, not filtered by threshold.
- total_kPa — the plain row sum, not filtered by threshold.
- loaded — count of cells strictly above threshold.
- cop_row, cop_col — grid index units, eps on the denominator so an unloaded frame gives approximately 0 rather than NA.

threshold therefore affects loaded and the COP weighting only; peak and total always describe the frame as recorded.

Value

A tibble::tibble with n_frames rows and columns frame, time_s, mean_kPa, peak_kPa, total_kPa, loaded, cop_row, cop_col. Pressure columns carry the trial's own unit; the kPa suffixes follow the cohort naming convention.

See Also

Other whole-grid frame metrics: pr_calc_cop_grid(), pr_calc_loaded_count(), pr_calc_mean_pressure_grid(), pr_calc_pci(), pr_calc_total_pressure(), pr_ref_pci()

Examples

```
trial <- pr_example_trial("saddle_horse")
fm <- pr_frame_metrics(trial)
names(fm)
all.equal(fm$mean_kPa, fm$total_kPa / trial$layout$n_sensors)
```

---

pr_interpolate *Spatial Interpolation of Pressure Data (for Display)*

Description

Performs bilinear interpolation on the maximum pressure picture between active sensor positions to produce a smoother image. The underlying trial data is not modified.

Usage

```
pr_interpolate(trial, factor = 2L)
```

Arguments

trial	A pr_trial object.
factor	Integer. Interpolation factor in each dimension. Default 2.

Value

A list with pressure_interp (interpolated matrix), x_mm, y_mm.

Examples

```
out <- pr_interpolate(pr_example_trial("insole"), factor = 2)
dim(out$pressure_interp)
```

`pr_layout` *Create a Pressure Sensor Layout*

Description

Defines the physical arrangement of sensors in a pressure measurement system, including grid dimensions, active sensor positions, physical coordinates, and named region masks.

Usage

```
pr_layout(
  grid_rows,
  grid_cols,
  active,
  coords_mm,
  regions = list(),
  sensor_area_cm2 = 1,
  pressure_range = c(0, 600),
  pressure_unit = "kPa",
  name = "custom",
  description = "",
  manufacturer = "",
  model = ""
)
```

Arguments

`grid_rows`	Integer. Number of rows in the sensor grid.
`grid_cols`	Integer. Number of columns in the sensor grid.
`active`	Logical matrix of dimensions `grid_rows x grid_cols`. `TRUE` for cells that contain active sensors.
`coords_mm`	Data frame with columns `sensor_id` (integer), `row` (integer), `col` (integer), `x_mm` (numeric), `y_mm` (numeric). Physical coordinates of each active sensor in millimeters.
`regions`	Named list of logical matrices (same dims as `active`). Each entry defines an anatomical or functional region.
`sensor_area_cm2`	Numeric. Area of a single sensor cell in cm².
`pressure_range`	Numeric vector of length 2. Min and max measurable pressure in kPa.
`pressure_unit`	Character. Unit of pressure measurement. Default `"kPa"`.
`name`	Character. Short identifier for this layout (e.g., `"insole_standard"`).
`description`	Character. Human-readable description.
`manufacturer`	Character. Sensor manufacturer. Default `""`.
`model`	Character. Sensor model/system name.

Value

A pr_layout S3 object (list with class attribute).

Examples

```
active <- matrix(TRUE, 4, 4)
coords <- data.frame(
  sensor_id = seq_len(16),
  row = rep(1:4, each = 4),
  col = rep(1:4, times = 4),
  x_mm = rep(seq(0, 30, 10), times = 4),
  y_mm = rep(seq(0, 30, 10), each = 4)
)
layout <- pr_layout(4, 4, active, coords, name = "example_4x4")
print(layout)
```

---

pr_layout_from_index_map

Build a Layout From an Explicit Channel Map

Description

Creates a pr_layout from a matrix that states, for every grid cell, which *device channel* is wired to it. This is the general answer to the problem that a sensor mat's channel numbering rarely matches the order in which the grid is read. The resulting layout carries the map with it, so pr_channel_order() can hand a reader the one permutation that puts raw device columns into the column-major grid order that pr_mask(), pr_calc_cop() and the plotting functions already assume.

Usage

```
pr_layout_from_index_map(
  index_matrix,
  name = "custom",
  sensor_area_cm2 = NA_real_,
  spacing_mm = NA_real_,
  pressure_range = NULL,
  ...
)
```

Arguments

index_matrix	Numeric matrix. index_matrix[r, c] is the device channel number sitting at grid cell (r, c). Channel numbers must be whole, positive and unique. NA or 0 marks a grid cell with no sensor behind it (that cell becomes inactive). If every cell is active the channel numbers must be exactly 1:length(index_matrix).
name	Character. Short layout identifier. Default "custom".

`sensor_area_cm2`
: Numeric. Area of one sensor cell in cm², or `NA_real_` (the default) when the manufacturer never published one. See *Missing physical scale* below.

`spacing_mm`
: Numeric. Centre-to-centre sensor pitch in mm, or `NA_real_` (the default) when it is unknown. See *Missing physical scale* below.

`pressure_range`
: Numeric vector of length 2, or `NULL` (default) for `c(0, NA_real_)` — a device whose ceiling is not known.

`...`
: Further arguments passed to `pr_layout()`, e.g. `regions`, `description`, `manufacturer`, `model`, `pressure_unit`.

Value

A pr_layout object with three extra fields: `channel_map` (the validated integer map, `NA` for empty cells), `channel_order` (the permutation returned by `pr_channel_order()`) and `coords_units`.

Missing physical scale

`pr_layout()` requires a positive `sensor_area_cm2` and physical `x_mm` / `y_mm` coordinates. Many real datasets have neither: the cell area and the sensor pitch were simply never recorded. Rather than invent a number, this function

- builds the object through `pr_layout()` with a placeholder area and then writes `NA_real_` into `sensor_area_cm2`, so the layout stays a valid `pr_layout` and `pr_validate_layout()` still passes, while anything derived from area (force in N, contact area in cm²) honestly returns `NA`; and
- when `spacing_mm` is `NA`, lays the coordinates out on a unit grid (`x_mm` = `col` − 1, `y_mm` = `row` − 1). Ordering, masks and centre of pressure are then all correct, but **the `x_mm` / `y_mm` columns are grid-index offsets, not millimetres** — a COP of `x_mm` = 8 means column 9, not 8 mm. The layout records this in `$coords_units` (`"grid_index"` or `"mm"`) and keeps the requested pitch in `$spacing_mm`.

Supply a real `sensor_area_cm2` / `spacing_mm` and the layout behaves as a fully physical one. Note that `plot.pr_layout()` sizes region tiles from `sensor_area_cm2`, so pass `regions` only alongside a real area.

See Also

Other channel map functions: `pr_channel_order()`, `pr_layout_saddle_novel()`

Examples

```
# A 2x2 mat whose channels are wired right-to-left within each row.
m <- rbind(c(2L, 1L), c(4L, 3L))
lay <- pr_layout_from_index_map(m, name = "toy_2x2")
lay$n_sensors
pr_channel_order(lay)

# Raw device frame: only channel 1 is loaded.
raw <- matrix(c(10, 0, 0, 0), nrow = 1)
grid_cols <- raw[, pr_channel_order(lay), drop = FALSE]
```

```
# Channel 1 is at grid (1, 2), i.e. the 3rd cell in column-major order.
which(grid_cols[1, ] > 0)
```

---

pr_layout_glove *Get Glove Sensor Layout*

Description

Returns a pr_layout for a glove-style hand pressure sensor (32 sensors across palm and fingertips).

Usage

```
pr_layout_glove()
```

Value

A pr_layout object.

---

pr_layout_insole *Get In-Shoe Pressure Insole Layout*

Description

Returns a pr_layout representing a typical in-shoe pressure measurement insole. The insole contains 99 capacitive sensors arranged in an anatomically shaped grid.

Usage

```
pr_layout_insole(size = c("standard", "wide"))
```

Arguments

size	Character. Insole size variant: "standard" (default) or "wide" (wider shoe insole with broader sensor spacing).

Value

A pr_layout object with predefined foot region masks.

Examples

```
layout <- pr_layout_insole()
print(layout)
```

pr_layout_list *List All Built-In Sensor Layouts*

Description

List All Built-In Sensor Layouts

Usage

```
pr_layout_list()
```

Value

A tibble::tibble with columns `name`, `system`, `grid_rows`, `grid_cols`, `n_sensors`, `n_regions`, `description`.

Examples

```
pr_layout_list()
```

pr_layout_mat *Get Generic Sensor Mat Layout*

Description

Returns a pr_layout for a generic capacitive pressure sensor mat. These general-purpose sensor arrays are used in research, seating, sports, and ergonomics applications.

Usage

```
pr_layout_mat(size = c("16", "32"))
```

Arguments

`size`	Character. `"16"` (default) for 16x16 = 256 sensors or `"32"` for 32x32 = 1024 sensors.

Value

A pr_layout with no predefined regions.

pr_layout_platform *Get Pressure Platform Layout*

Description

Returns a pr_layout for a barefoot pressure platform. Provides high-resolution plantar pressure distribution during static and dynamic conditions.

Usage

```
pr_layout_platform(model = c("st", "xl", "cl"))
```

Arguments

model	Character. Platform model: "st" (standard, default), "xl" (extended length), or "cl" (clinical/compact).

Value

A pr_layout object. Region masks are not predefined — use pr_mask_foot_auto() to auto-segment footprints from the data.

Examples

```
layout <- pr_layout_platform()
layout$grid_rows
layout$grid_cols
```

pr_layout_saddle *Get Saddle Pressure Mat Layout*

Description

Returns a pr_layout for saddle pressure measurement using a capacitive sensor mat. Includes predefined region masks based on published research protocols.

Usage

```
pr_layout_saddle(type = c("horse", "bicycle"))
```

Arguments

type	Character. "horse" (default) for equine saddle fitting with 6-region mask (cranial / middle / caudal x left / right per Werner et al. 2002), or "bicycle" for bicycle saddle with ischial/perineal regions.

Value

A pr_layout with application-specific region masks.

Examples

```
layout <- pr_layout_saddle("horse")
names(layout$regions)
```

`pr_layout_saddle_novel` *Novel/Pliance 16x16 Saddle Mat Layout*

Description

The ready-made layout for the two-panel 16 x 16 (256 sensor) capacitive saddle mat exported by Novel/Pliance systems. All 256 sensors are active.

Usage

```
pr_layout_saddle_novel(
  pressure_range = c(0, 63.75),
  sensor_area_cm2 = NA_real_,
  ...
)
```

Arguments

`pressure_range`	Numeric vector of length 2. Default `c(0, 63.75)` kPa, the hardware ceiling (255 x 0.25 kPa).
`sensor_area_cm2`	Numeric. Cell area in cm²; `NA_real_` by default because the manufacturer never published one.
`...`	Further arguments passed to `pr_layout_from_index_map()`, e.g. `name`, `spacing_mm`, `description`, `regions`.

Details

The mat is built from two 16 x 8 panels. Device channel 1 sits at grid (row 1, column 9) — the first cell of the *right* panel — and channel 9 at (row 1, column 8), with each grid row advancing the channel number by 16 and the left panel running right-to-left. Applying `pressure[, pr_channel_order(layout)]` to a raw export therefore places every channel at its true anatomical position, which is what makes `pr_mask_saddle_6()`, `pr_mask_symmetry()`, `pr_calc_cop()` and the heatmaps simultaneously correct. Without the permutation the aggregate statistics (mean, peak, total) are unchanged, so a wrong map is invisible in every summary and wrong in every map.

No sensor pitch or cell area is published for this mat, so both default to `NA` and the coordinates are grid indices — see the *Missing physical scale* section of `pr_layout_from_index_map()`.

Value

A pr_layout with 256 active sensors and a channel_map field.

See Also

Other channel map functions: pr_channel_order(), pr_layout_from_index_map()

Examples

```
layout <- pr_layout_saddle_novel()
layout$n_sensors
layout$channel_map[1, 1:10]

# Device channel 1 lands at grid (row 1, col 9):
pos <- match(1L, pr_channel_order(layout))
layout$coords_mm[pos, c("row", "col")]
```

---

pr_layout_seat *Get Seating Pressure Mat Layout*

Description

Returns a pr_layout for seating pressure assessment. Uses a 32x32 grid (1024 sensors) with application-specific regions.

Usage

```
pr_layout_seat(type = c("wheelchair", "car", "office"))
```

Arguments

type	Character. "wheelchair" (default), "car", or "office".

Value

A pr_layout object.

`pr_lock_table` *Lock a Table's Content and Shape*

Description

Writes a sidecar lock file recording a data frame's row count, column count, column names, column types and a content hash. Committed next to the analysis, it turns "the input has not changed" from an assumption into something `pr_verify_lock()` can check.

Usage

```
pr_lock_table(x, path, algo = "md5")
```

Arguments

`x`	A data frame (or tibble::tibble) to lock.
`path`	Path to the lock file to write. The parent directory must exist. A `.lock` extension is conventional but not required.
`algo`	Hash algorithm. Only `"md5"` is available, because `tools::md5sum()` is the only hash in base R and this package takes no extra dependencies. Anything else is an error.

Details

The hash is md5 — `tools::md5sum()` is the only hash function in base R — computed over a byte stream this function writes itself: a fixed magic string, then for each column an `NA` mask of one byte per row followed by the values as big-endian IEEE-754 doubles, 4-byte big-endian integers, or NUL-terminated UTF-8 strings.

That stream is used in preference to the two obvious alternatives because both are unstable in ways that would make a lock fail for the wrong reason:

- `utils::write.csv()` renders doubles as decimal text. R 4.3.0 changed how `as.character()` formats doubles, so identical numbers can produce different bytes under different R versions, and decimal text does not round-trip a double in any case.
- `serialize()` writes the R version that produced the stream into its own header, so the same object hashes differently under R 4.3 and R 4.6.

Pinning the byte order makes x86 and ARM agree; writing doubles as bits removes every formatting decision; carrying `NA` in a side mask means no real value can impersonate a missing one. Before hashing, each column is reduced to a bare vector — factors to their labels, `Date`/`POSIXct` to seconds, all other attributes dropped — so a tibble::tibble and a `data.frame` holding the same numbers hash alike, and a class change is reported through the `types` line rather than as a content change.

Column names and types are recorded but are deliberately *not* part of the content hash. That is what lets `pr_verify_lock()` distinguish a pure rename ("columns changed, content unchanged") from an edit to the data.

No timestamp is written. Re-locking an unchanged table leaves the file byte-identical, so a lock file under version control produces a diff only when the table really moved.

List columns cannot be hashed and are rejected with an error rather than skipped, because a silently unhashed column is worse than no lock at all.

Value

Invisibly, a one-row tibble::tibble with columns `path`, `algo`, `nrow`, `ncol`, `columns` (list of character), `types` (list of character) and `content` (the hash).

See Also

`pr_verify_lock()` to check a table against a lock file.

Other provenance functions: `pr_design_gaps()`, `pr_factor_contract()`, `pr_validate_summary()`, `pr_verify_lock()`

Examples

```
design <- data.frame(
  ID = c("ID001", "ID001", "ID003"),
  Mode = c("MH", "MS", "MG"),
  n_frames = c(23432L, 10662L, 8000L)
)
path <- tempfile(fileext = ".lock")
lock <- pr_lock_table(design, path)
lock$content

# Locking is deterministic: the same table gives the same file.
path2 <- tempfile(fileext = ".lock")
pr_lock_table(design, path2)
identical(readLines(path), readLines(path2))

unlink(c(path, path2))
```

---

pr_mask *Create a Region Mask*

Description

Wraps a logical matrix describing a named region over a sensor layout.

Usage

```
pr_mask(mask_matrix, name, layout)
```

Arguments

mask_matrix	Logical matrix matching the dimensions of layout$active.
name	Character. Region name.
layout	A pr_layout object this mask applies to.

Value

A pr_mask S3 object.

Examples

```
layout <- pr_layout_saddle("horse")
m <- layout$regions$cranial_left
pr_mask(m, "cranial_left", layout)
```

---

pr_mask_apply *Apply Masks to Extract Regional Pressure Data*

Description

Returns a named list of numeric matrices. Each entry has dimensions n_frames x n_sensors_in_region and contains the pressure values of that region's sensors for each frame.

Usage

```
pr_mask_apply(trial, masks = NULL)
```

Arguments

trial	A pr_trial object.
masks	Named list of pr_mask objects or logical matrices. If NULL (default), the trial's layout regions are used.

Value

Named list of numeric matrices.

Examples

```
trial <- pr_example_trial("saddle_horse")
reg <- pr_mask_apply(trial)
vapply(reg, ncol, integer(1))
```

pr_mask_default *Get Default Region Masks for a Layout*

Description

Returns a named list of pr_mask objects built from the `regions` field of a pr_layout.

Usage

```
pr_mask_default(layout)
```

Arguments

`layout`	A pr_layout object.

Value

Named list of pr_mask objects (possibly empty).

Examples

```
masks <- pr_mask_default(pr_layout_saddle("horse"))
names(masks)
```

pr_mask_foot_auto *Auto-Segment Foot Regions from Pressure Data*

Description

For platform recordings where foot regions must be detected from the raw pressure matrix. Uses the maximum pressure picture (MPP) to identify the foot outline, then segments into anatomical regions based on proportional geometry.

Usage

```
pr_mask_foot_auto(trial, n_regions = 7L, threshold = 1)
```

Arguments

`trial`	A pr_trial object from a pressure platform recording.
`n_regions`	Integer. `7` (default, full segmentation into heel, midfoot, met1, met2-3, met4-5, hallux, lesser toes) or `3` (heel/midfoot/forefoot).
`threshold`	Numeric. Minimum pressure (kPa) to consider a sensor loaded. Default `1.0`.

Value

Named list of pr_mask objects.

`pr_mask_mirror_balance` *Largest Mirror-Symmetric Subset of a Mask*

Description

Trims a region mask down to the cells whose mirror image across the mat's midline is also in the mask. The result is symmetric by construction, so a left/right comparison made on it cannot be an artefact of one side simply owning more sensors than the other.

Usage

```
pr_mask_mirror_balance(mask, layout, axis = c("vertical", "horizontal"))
```

Arguments

`mask`	A pr_mask object or a logical matrix matching the layout grid.
`layout`	A pr_layout object.
`axis`	Character. `"vertical"` (default) mirrors columns, i.e. left against right; `"horizontal"` mirrors rows, i.e. anterior against posterior.

Details

This is the largest mirror-symmetric subset of the input: a cell survives if and only if both it and its reflection were present, and no mirror-symmetric superset of that set exists inside the mask. The two halves of the result are exact reflections of each other, which is a stronger guarantee than equal counts.

The reason to want it is that an asymmetry index divides by the mean of the two sides. If the region of interest happens to contain more cells on one side — because a sensor died, or because a data-driven mask was built from a load pattern that is itself lopsided — then part of the resulting index measures the mask, not the horse. In the study cohort, dropping sensors that were flat for more than 95% of every recording left 106 live cells on the left and 117 on the right; balancing to 106/106 moved the cohort median asymmetry from -30.5% to -39.0%, the second of which agrees with the whole-grid figure and the first of which does not.

Cells outside `layout$active` are never kept. When the axis has an odd number of lines the centre line reflects onto itself and belongs to neither half, so it is dropped; on an even grid (such as the 16 x 16 saddle mat) nothing is lost to this.

Value

A pr_mask object carrying two extra fields: `axis`, and `halves`, a named integer vector of the two sides' sensor counts, which are equal by construction.

See Also

pr_symmetry_sensitivity(), which applies this across a family of masks.

Other regional symmetry functions: pr_calc_cop_masked(), pr_calc_regional_map(), pr_calc_symmetry_map(), pr_mask_rowbands(), pr_symmetry_sensitivity()

Examples

```
layout <- pr_layout_saddle("horse")

# A deliberately lopsided region: all of the left half, but only three
# columns of the right.
m <- matrix(FALSE, 16, 16)
m[, 1:8] <- TRUE
m[, 9:11] <- TRUE
bal <- pr_mask_mirror_balance(m, layout)
bal$halves

# The surviving columns are 6:11 -- the ones with a partner on both sides.
which(apply(bal$matrix, 2, any))
```

---

pr_mask_rowbands *Row-Band Region Masks With an Explicit Split*

Description

Builds cranial / middle / caudal row bands (optionally crossed with the left and right halves of the mat) from **stated** split rows rather than from a derived `floor(grid_rows / 3)`. The split is the argument, so a study's own band definition survives into the code instead of being approximated by a division.

Usage

```
pr_mask_rowbands(
  layout,
  breaks = c(5, 11),
  sides = TRUE,
  band_names = c("cranial", "middle", "caudal")
)
```

Arguments

layout	A pr_layout object.
breaks	Numeric vector of `length(band_names) - 1` strictly increasing whole numbers in `1:(grid_rows - 1)`. Each is the last row of its band. Default `c(5, 11)`.
sides	Logical. Cross each band with the left and right halves of the mat. Default `TRUE`.
band_names	Character vector of band names, cranial to caudal. Default `c("cranial", "middle", "caudal")`.

Details

breaks gives the **last row of each band except the last**. With the default c(5, 11) on a 16-row mat the bands are rows 1:5, 6:11 and 12:16 — the split used by pr_layout_saddle()'s regions, in which the middle band is one row deeper than the outer two.

pr_mask_saddle_6() uses third <- floor(16 / 3), which gives 1:5, 6:10, 11:16 instead: the *caudal* band is the deep one. Passing breaks = c(5, 10) reproduces those cells exactly, with the same names in the same order, so the two conventions can be compared rather than confused. Because that one-row difference moves eight sensors between two zones, it is worth stating which convention a number came from.

Left and right are split at floor(grid_cols / 2), matching pr_mask_symmetry() and pr_mask_saddle_6(). With sides = TRUE the masks come back in band-major order (cranial_left, cranial_right, middle_left, ...); with sides = FALSE there is one full-width mask per band. Every mask is intersected with layout$active, so inactive cells (a withers cutout, say) never enter a band.

Value

A named list of pr_mask objects: length(band_names) of them when sides = FALSE, twice that when sides = TRUE.

See Also

pr_mask_saddle_6() for the derived-thirds convention, pr_calc_regional_map() which consumes these masks.

Other regional symmetry functions: pr_calc_cop_masked(), pr_calc_regional_map(), pr_calc_symmetry_map(), pr_mask_mirror_balance(), pr_symmetry_sensitivity()

Examples

```
layout <- pr_layout_saddle("horse")
masks <- pr_mask_rowbands(layout)
names(masks)
vapply(masks, function(m) m$n_sensors, integer(1))

# breaks = c(5, 10) is the pr_mask_saddle_6() convention:
same <- pr_mask_rowbands(layout, breaks = c(5, 10))
identical(same$middle_left$matrix, pr_mask_saddle_6(layout)$middle_left$matrix)

# Full-width bands, no left/right split:
names(pr_mask_rowbands(layout, sides = FALSE))
```

---

pr_mask_saddle_6 *Standard 6-Region Saddle Mask*

Description

Creates the 6-region saddle pressure analysis mask dividing the sensor mat into cranial / middle / caudal thirds crossed with left / right halves, as established by Werner et al. (2002) and used in subsequent equine back-pressure studies.

Usage

```
pr_mask_saddle_6(layout)
```

Arguments

layout	A pr_layout object (typically one returned by pr_layout_saddle() with type = "horse").

Value

Named list of 6 pr_mask objects.

Examples

```
pr_mask_saddle_6(pr_layout_saddle("horse"))
```

---

pr_mask_symmetry *Split Layout into Left/Right Halves*

Description

Produces a two-element named list of pr_mask objects dividing the layout along the specified axis.

Usage

```
pr_mask_symmetry(layout, axis = c("vertical", "horizontal"))
```

Arguments

layout	A pr_layout object.
axis	Character. "vertical" (default, split columns into left/right) or "horizontal" (split rows into anterior/posterior).

Value

A named list of two pr_mask objects.

Examples

```
pr_mask_symmetry(pr_layout_saddle("horse"))
```

pr_merge_trials *Merge Trials into a Dataset*

Description

Merge Trials into a Dataset

Usage

```
pr_merge_trials(..., group_var = "condition")
```

Arguments

`...`	pr_trial objects or lists of them.
`group_var`	Character. Name of a metadata field to use as the grouping variable.

Value

A pr_dataset object.

Examples

```
pr_merge_trials(
  pr_example_trial("insole", seed = 1),
  pr_example_trial("insole", seed = 2)
)
```

pr_meta_from_filename *Split Metadata Fields Out of a File Name*

Description

Many measurement archives encode a recording's design cells in its file name, e.g. `ID052_K_K_KG00_MS.asc` for subject `ID052`, saddle `K`, pad `K`, weight `KG00`, mode `MS`. This splits such a base name into a one-row tibble of named fields.

Usage

```
pr_meta_from_filename(
  path,
  fields,
  sep = "_",
  on_fail = c("skip", "error", "na")
)
```

Arguments

path	Character scalar. A file path or bare name. The directory and the extension are removed before splitting.
fields	Character vector of output column names, in the order the parts appear in the name.
sep	Character scalar. Literal separator between parts. Default "_".
on_fail	What to do when the name does not split into exactly length(fields) non-empty parts: "skip" (default) returns NULL, "error" throws, "na" returns a one-row tibble of NA_character_.

Details

A name matches only when it splits into *exactly* length(fields) non-empty parts. That strictness is the point: it is what lets a directory read reject stray files (repeat takes such as ID003_K_K_KG60_MH_R01, notes, exports) instead of silently mislabelling them.

Value

A one-row tibble with one character column per entry of fields, or NULL (see on_fail).

See Also

Other Pliance ingest functions: pr_read_dir(), pr_read_pliance()

Examples

```
fields <- c("ID", "Saddle", "Pad", "Weight", "Mode")
pr_meta_from_filename("ID052_K_K_KG00_MS.asc", fields)

# A repeat take carries a sixth part, so it does not match:
pr_meta_from_filename("ID003_K_K_KG60_MH_R01.asc", fields)

pr_meta_from_filename("ID003_K_K_KG60_MH_R01.asc", fields, on_fail = "na")
```

---

pr_plot_3d *Interactive 3D Pressure Surface*

Description

Renders a pressure frame as a 3D surface using plotly.

Usage

```
pr_plot_3d(trial, frame = NULL, palette = "viridis")
```

Arguments

trial	A pr_trial object.
frame	Integer. Frame to display. If NULL, uses the MPP.
palette	Character. One of the heatmap palettes.

Value

A plotly object.

Examples

```
pr_plot_3d(pr_example_trial("saddle_horse"))
```

---

pr_plot_comparison *Compare Two Trials*

Description

Produces a side-by-side heatmap comparison, a difference map, or a parameter-table plot for two trials.

Usage

```
pr_plot_comparison(
  trial_a,
  trial_b,
  type = c("heatmap", "difference", "parameters"),
  labels = c("A", "B")
)
```

Arguments

trial_a	A pr_trial object. Reference.
trial_b	A pr_trial object. Comparison.
type	Character. "heatmap" (default), "difference", or "parameters".
labels	Character vector of length 2. Default c("A", "B").

Value

A ggplot2 (or patchwork) object.

Examples

```
a <- pr_example_trial("insole", seed = 1)
b <- pr_example_trial("insole", seed = 2)
pr_plot_comparison(a, b, type = "difference")
```

pr_plot_contact_area *Plot Contact Area vs Time*

Description

Plot Contact Area vs Time

Usage

```
pr_plot_contact_area(trial, show_cycles = FALSE)
```

Arguments

trial	A pr_trial object.
show_cycles	Logical. If TRUE, shade detected stance phases. Default FALSE.

Value

A ggplot2 object.

pr_plot_cop *Plot Center of Pressure Trajectory*

Description

Plot Center of Pressure Trajectory

Usage

```
pr_plot_cop(trial, show_layout = TRUE, color_by = c("time", "velocity"))
```

Arguments

trial	A pr_trial object.
show_layout	Logical. Draw sensor grid in background. Default TRUE.
color_by	Character. "time" (default) or "velocity".

Value

A ggplot2 object.

Examples

```
pr_plot_cop(pr_example_trial("saddle_horse"))
```

pr_plot_cop_butterfly *COP Butterfly Plot Across Gait Cycles*

Description

Overlays COP trajectories from each detected stance phase, centered on heel strike.

Usage

```
pr_plot_cop_butterfly(trial, cycles = NULL)
```

Arguments

trial	A pr_trial object.
cycles	Output of pr_calc_gait_cycles(). If NULL, auto-detected.

Value

A ggplot2 object.

Examples

```
pr_plot_cop_butterfly(pr_example_trial("insole"))
```

pr_plot_foot_report *Foot Pressure Report Panel*

Description

Composite figure for foot pressure analysis: MPP map, regional bar chart, force-time curve, COP trajectory.

Usage

```
pr_plot_foot_report(trial)
```

Arguments

trial	A pr_trial object from an insole or platform recording.

Value

A patchwork object.

Examples

```
pr_plot_foot_report(pr_example_trial("insole"))
```

pr_plot_force_time *Plot Force vs Time*

Description

Plot Force vs Time

Usage

```
pr_plot_force_time(trial, show_cycles = FALSE)
```

Arguments

trial	A pr_trial object.
show_cycles	Logical. If TRUE, shade detected stance phases. Default FALSE.

Value

A ggplot2 object.

Examples

```
pr_plot_force_time(pr_example_trial("insole"))
```

pr_plot_heatmap *Plot Pressure Heatmap*

Description

Displays a 2D heatmap for a single frame or a summary image (MPP, MVP, PTI, contact frequency).

Usage

```
pr_plot_heatmap(
  trial,
  frame = NULL,
  type = c("mpp", "mvp", "pti", "contact"),
  show_regions = FALSE,
  palette = "viridis",
  range = NULL,
  interpolate = FALSE,
  title = NULL
)
```

Arguments

`trial`	A pr_trial object.
`frame`	Integer. Frame number to display. If `NULL` (default), shows the summary picture specified by `type`.
`type`	Character. Summary type when `frame` is `NULL`: `"mpp"` (max), `"mvp"` (mean), `"pti"` (pressure-time integral), `"contact"` (contact frequency).
`show_regions`	Logical. Overlay region boundaries. Default `FALSE`.
`palette`	Character. One of `"viridis"` (default), `"inferno"`, `"plasma"`, `"magma"`, `"jet"`, `"classic"`.
`range`	Numeric vector of length 2. Colour range. `NULL` auto-scales.
`interpolate`	Logical. Reserved for spatial interpolation (currently draws the raw sensor grid). Default `FALSE`.
`title`	Character. Plot title. `NULL` auto-generates.

Value

A `ggplot2` object.

Examples

```
trial <- pr_example_trial("insole")
pr_plot_heatmap(trial)
```

---

pr_plot_heatmap_masked

Plot Pressure Heatmap with Region Overlay

Description

Convenience wrapper for `pr_plot_heatmap()` that always overlays region boundaries.

Usage

```
pr_plot_heatmap_masked(
  trial,
  masks = NULL,
  frame = NULL,
  type = c("mpp", "mvp", "pti", "contact"),
  palette = "viridis",
  range = NULL,
  title = NULL
)
```

Arguments

trial	A pr_trial object.
masks	Named list of pr_mask objects or logical matrices. If NULL, uses layout regions.
frame	Integer. Frame number to display. If NULL (default), shows the summary picture specified by type.
type	Character. Summary type when frame is NULL: "mpp" (max), "mvp" (mean), "pti" (pressure-time integral), "contact" (contact frequency).
palette	Character. One of "viridis" (default), "inferno", "plasma", "magma", "jet", "classic".
range	Numeric vector of length 2. Colour range. NULL auto-scales.
title	Character. Plot title. NULL auto-generates.

Value

A ggplot2 object.

Examples

```
pr_plot_heatmap_masked(pr_example_trial("saddle_horse"))
```

---

pr_plot_pressure_time *Plot Peak and Mean Pressure vs Time*

Description

Plot Peak and Mean Pressure vs Time

Usage

```
pr_plot_pressure_time(trial, show_cycles = FALSE)
```

Arguments

trial	A pr_trial object.
show_cycles	Logical. If TRUE, shade detected stance phases. Default FALSE.

Value

A ggplot2 object.

Examples

```
pr_plot_pressure_time(pr_example_trial("insole"))
```

pr_plot_regional_bar *Regional Parameter Comparison Bar Chart*

Description

Regional Parameter Comparison Bar Chart

Usage

```
pr_plot_regional_bar(regional_data, parameter = "mpp", thresholds = NULL)
```

Arguments

regional_data	Output of pr_calc_regional().
parameter	Character. Parameter column to plot. Default "mpp".
thresholds	A tibble::tibble from pr_ref_*() functions. If non-NULL, draws horizontal threshold lines where parameter matches.

Value

A ggplot2 object.

Examples

```
trial <- pr_example_trial("saddle_horse")
reg <- pr_calc_regional(trial)
pr_plot_regional_bar(reg, "mpp")
```

pr_plot_saddle_report *Saddle Fit Report Panel*

Description

Composite figure for equine saddle analysis: MPP heatmap, 6-region bar chart, COP trajectory, and left/right symmetry summary.

Usage

```
pr_plot_saddle_report(trial, thresholds = NULL)
```

Arguments

trial	A pr_trial from a saddle recording.
thresholds	A reference threshold tibble (typically from pr_ref_saddle()). If NULL, uses the von Peinen 2010 defaults.

Value

A patchwork object (4-panel composite).

Examples

```
pr_plot_saddle_report(pr_example_trial("saddle_horse"))
```

`pr_plot_symmetry` *Left/Right Symmetry Plot*

Description

Left/Right Symmetry Plot

Usage

```
pr_plot_symmetry(
  trial,
  parameter = c("peak_pressure", "mean_pressure", "force", "contact_area")
)
```

Arguments

`trial`	A pr_trial object.
`parameter`	Character. One of "peak_pressure", "mean_pressure", "force", "contact_area".

Value

A ggplot2 object with a mirrored bar chart.

Examples

```
pr_plot_symmetry(pr_example_trial("saddle_horse"))
```

pr_profile_matrix *Per-Sensor Feature Matrix for a Dataset*

Description

Stacks pr_sensor_map() across every trial in a dataset: a n_trials x n_sensors numeric matrix whose rows are recordings and whose columns are sensors, in layout (column-major grid) order. This is the input stats::prcomp(), stats::dist(), stats::cmdscale() or a classifier expects, with no further reshaping.

Usage

```
pr_profile_matrix(dataset, statistic = "mean", threshold = 0)
```

Arguments

dataset	A pr_dataset object, or a list of pr_trial objects.
statistic	Character. Any statistic accepted by pr_sensor_map(). Default "mean".
threshold	Numeric. Cells at or below this value count as unloaded. Default 0.

Details

Row names are the trials' identifying metadata (trial_id, else file_label, else subject_id, else trial_<i>), made unique with make.unique(). Column names are sensor_<id> taken from the first trial's layout.

Every trial must expose the same number of sensors, since column k has to mean the same grid cell in every row; a mismatch is an error rather than a recycled row. Differing *layout names* at the same sensor count are only a warning: two layouts may legitimately share a grid, but if they do not, the matrix silently compares different anatomy, so the warning is worth reading.

Value

A numeric matrix with length(dataset) rows and n_sensors columns, with row and column names as described above.

See Also

pr_sensor_map() for a single trial.

Other sensor aggregation functions: pr_batch_frame_summary(), pr_sensor_map()

Examples

```
ds <- pr_dataset(lapply(1:3, function(s) {
  pr_example_trial("saddle_horse", seed = s)
}))
X <- pr_profile_matrix(ds, "mean")
dim(X)
```

```
X[, 1:4]

# Straight into a PCA of pressure distribution shape:
pc <- stats::prcomp(X)
round(pc$sdev[1:2], 3)
```

`pr_read_ascii` *Read ASCII Pressure Data Export*

Description

Parses ASCII data files containing pressure sensor data. Automatically detects header format and extracts metadata, the pressure matrix, and timing.

Usage

```
pr_read_ascii(
  path,
  layout = NULL,
  sampling_hz = NULL,
  separator = "auto",
  skip = NULL,
  verbose = TRUE
)
```

Arguments

`path`	Character. Path to the ASCII file (`.asc`, `.txt`).
`layout`	A pr_layout object. If `NULL`, inferred from the header or sensor count.
`sampling_hz`	Numeric. Sampling rate (Hz). If `NULL`, taken from the header or defaulted to 50.
`separator`	Character. Column separator. `"auto"` (default) detects tab/space/comma/semicolon.
`skip`	Integer. Header lines to skip. If `NULL`, auto-detected.
`verbose`	Logical. Default `TRUE`.

Value

A pr_trial object.

Examples

```
path <- pr_example_files("insole")
trial <- pr_read_ascii(path, verbose = FALSE)
trial$n_frames
```

pr_read_auto *Auto-Detect and Read a Pressure Data File*

Description

Inspects the file extension and header, then dispatches to the correct parser.

Usage

```
pr_read_auto(path, layout = NULL, verbose = TRUE)
```

Arguments

path	Character. Path to any supported pressure data file.
layout	A pr_layout object. If NULL, inferred by the dispatched parser.
verbose	Logical. Default TRUE.

Value

A pr_trial object (for .asc / .txt / .csv), or a logical mask matrix / pr_mask object (for .msa/.msr/.msp).

Examples

```
path <- pr_example_files("insole")
tr <- pr_read_auto(path, verbose = FALSE)
inherits(tr, "pr_trial")
```

pr_read_csv *Read Pressure Data from CSV*

Description

Reads pressure data from a CSV file in either wide (one column per sensor) or long (tidy) format.

Usage

```
pr_read_csv(
  path,
  format = c("wide", "long"),
  layout = NULL,
  time_col = NULL,
  sampling_hz = 100,
  verbose = TRUE
)
```

Arguments

path	Character. Path to CSV file.
format	Character. "wide" (default, columns are sensors) or "long" (a tidy format with frame, sensor_id, pressure).
layout	A pr_layout object. Required for wide format when the sensor count cannot be inferred.
time_col	Character. Name of the time column. If NULL, timestamps are generated from sampling_hz.
sampling_hz	Numeric. Sampling rate. Default 100.
verbose	Logical. Default TRUE.

Value

A pr_trial object.

Examples

```
tmp <- tempfile(fileext = ".csv")
m <- matrix(runif(10 * 99, 0, 100), nrow = 10, ncol = 99)
utils::write.csv(m, tmp, row.names = FALSE)
trial <- pr_read_csv(tmp, format = "wide", layout = pr_layout_insole(),
                     verbose = FALSE)
trial$n_frames
```

---

pr_read_dir *Read a Directory of Pressure Exports*

Description

Walks a directory, reads every matching file with reader, and collects the results into a pr_dataset. Files that cannot be read - or whose names do not match meta_fields - are reported and skipped rather than aborting the whole batch.

Usage

```
pr_read_dir(
  dir,
  pattern = "\\.asc$",
  reader = pr_read_pliance,
  layout = NULL,
  meta_fields = NULL,
  recursive = TRUE,
  on_error = c("skip", "abort"),
  quiet = FALSE,
  ...,
  sep = "_",
  name = NULL
)
```

Arguments

dir	Character scalar. Directory to walk.
pattern	Regular expression matching file names to read. Default "\\.asc$".
reader	Function called as reader(path, layout = layout, ...). Default pr_read_pliance().
layout	A pr_layout passed to reader. Default NULL.
meta_fields	Character vector of file-name fields, or NULL (default) to read every matching file and parse no names.
recursive	Logical. Descend into sub-directories. Default TRUE.
on_error	"skip" (default) records a failure and continues; "abort" re-throws the first one.
quiet	Logical. Suppress the progress and summary messages and the readers' warnings. Default FALSE.
...	Further arguments passed to reader.
sep	Character scalar. Separator passed to pr_meta_from_filename(). Default "_".
name	Character scalar. Dataset name. NULL (default) uses the directory's base name.

Details

When meta_fields is supplied, each file name is split with pr_meta_from_filename() and the parts are stored in the trial's metadata under those names. The first part additionally fills subject_id and the remaining parts, rejoined with sep, fill condition - the field pr_dataset() groups by - unless the reader already set them. A name that does not split into exactly length(meta_fields) parts is skipped, which is the intended way to exclude repeat takes and stray exports from a cohort.

Value

A pr_dataset. Attribute "read_report" holds a tibble with one row per candidate file (file, path, status, reason).

See Also

Other Pliance ingest functions: pr_meta_from_filename(), pr_read_pliance()

Examples

```
dir <- tempfile()
dir.create(dir)
hdr <- c(
  "Dateiname:  demo.mat\tDatum/Zeit: 01.01.25 09.00",
  "Kalibrationsdatei:  demo_cal",
 "gesamte Messzeit [Sek.]: 0.040\tZeit pro Bild [Sek.]:  0.02000\tMessfrequenz [Hz]:  50",
  "Druckwerte in  kPa", "", "elektrisch:",
  "Zeit [Sek.]\t1\t2\t3\t4",
  "0.02000\t10\t0\t0\t0",
  "0.04000\t0\t20\t0\t0"
)
writeLines(hdr, file.path(dir, "ID001_K_KG00.asc"))
```

```
writeLines(hdr, file.path(dir, "ID002_K_KG40.asc"))
writeLines(hdr, file.path(dir, "ID002_K_KG40_R01.asc"))  # repeat take

lay <- pr_layout(
  2, 2, matrix(TRUE, 2, 2),
  data.frame(sensor_id = 1:4, row = c(1L, 2L, 1L, 2L),
             col = c(1L, 1L, 2L, 2L),
             x_mm = c(0, 0, 1, 1), y_mm = c(0, 1, 0, 1))
)
ds <- pr_read_dir(dir, layout = lay,
                  meta_fields = c("ID", "Saddle", "Weight"), quiet = TRUE)
length(ds)
attr(ds, "read_report")$status
unlink(dir, recursive = TRUE)
```

`pr_read_forcesensor` *Read Force Sensor Data*

Description

Parses a CSV file containing force data from a portable force-sensing insole or similar device. These systems typically record a small number of force zones (not a full pressure matrix); this parser returns a simplified pr_trial where each "sensor" represents a force zone. Pressure values are reported in Newtons.

Usage

```
pr_read_forcesensor(path, force_cols = NULL, time_col = "time", verbose = TRUE)
```

Arguments

`path`	Character. Path to the CSV file.
`force_cols`	Character vector. Names of force columns. If `NULL`, all numeric columns other than the first (time) column are used.
`time_col`	Character. Name of the time column. Default `"time"`.
`verbose`	Logical. Default `TRUE`.

Value

A pr_trial object with a minimal single-row layout (one sensor per force zone).

Examples

```
tmp <- tempfile(fileext = ".csv")
df <- data.frame(time = seq(0, 1, by = 0.01),
                 heel = runif(101, 0, 200),
                 mid  = runif(101, 0, 150),
                 fore = runif(101, 0, 300))
```

```
utils::write.csv(df, tmp, row.names = FALSE)
trial <- pr_read_forcesensor(tmp, verbose = FALSE)
trial$n_sensors
```

`pr_read_mask` *Read Mask File*

Description

Parses `.msa` / `.msr` / `.msp` mask files or any text file containing a binary region matrix. The expected format is a text matrix of 0/1 (or non-zero) values, optionally preceded by a short header.

Usage

```
pr_read_mask(path, layout = NULL, name = "imported", verbose = TRUE)
```

Arguments

`path`	Character. Path to the mask file.
`layout`	A pr_layout object. Required if the mask dimensions must be validated against a specific layout; otherwise, the mask is returned as-is.
`name`	Character. Name for the resulting pr_mask object. Default `"imported"`.
`verbose`	Logical. Default `TRUE`.

Value

A pr_mask object (if a layout is supplied) or a logical matrix.

Examples

```
tmp <- tempfile(fileext = ".msa")
writeLines(c(
  "1 1 0 0",
  "1 1 0 0",
  "0 0 1 1",
  "0 0 1 1"
), tmp)
m <- pr_read_mask(tmp, verbose = FALSE)
dim(m)
```

`pr_read_pliance` *Read a Novel/Pliance ASCII Export*

Description

Reads one Novel/Pliance `.asc` pressure export into a pr_trial. The recorded time column is used as-is rather than a synthesised sequence, so a trial whose frames were dropped keeps honest timestamps.

Usage

```
pr_read_pliance(
  path,
  layout = NULL,
  skip = 7L,
  sep = "\t",
  time_col = 1L,
  channel_map = NULL,
  na_strings = c("", "NA", "___"),
  encoding = "latin1",
  on_na = c("drop_frame", "zero"),
  sampling_hz = NULL,
  metadata = list()
)
```

Arguments

`path`	Character scalar. Path to the `.asc` export.
`layout`	A pr_layout object. `NULL` (default) picks a layout from the channel count.
`skip`	Integer. Header lines before the first data line. Default `7L`, the Pliance export format.
`sep`	Character scalar. Literal column delimiter. Default `"\t"`.
`time_col`	Integer. Position of the timestamp column, default `1L`. `NULL` synthesises time from `sampling_hz` instead.
`channel_map`	Device-channel map: a grid-shaped matrix, a bare integer permutation, or `NULL` (default). See *Channel order*.
`na_strings`	Character vector of tokens read as missing. Default `c("", "NA", "___")`.
`encoding`	Character. File encoding, default `"latin1"`.
`on_na`	`"drop_frame"` (default) or `"zero"`. See *Missing values*.
`sampling_hz`	Numeric. Overrides the rate recorded in the header. `NULL` (default) uses the header, or the timestamps when the header is silent.
`metadata`	Named list merged over the parsed metadata, so callers can override any field.

Value

A pr_trial object.

Channel order

A device export lists columns in *device channel* order. Every other pressR function - pr_mask(), pr_calc_cop(), the heatmaps - indexes pressure columns in column-major grid order, the order which(layout$active) produces. For a single-panel device the two agree; for a multi-panel mat they do not. Supplying channel_map (or a layout that carries one, such as pr_layout_saddle_novel()) applies the single permutation pressure[, order, drop = FALSE] that reconciles them.

channel_map may be a grid-shaped matrix whose entry at [row, col] is the device channel sitting at that grid cell, or the equivalent bare integer permutation. When channel_map is NULL and layout carries a channel_order, that order is used; otherwise the columns are left in device order.

Missing values

Pliance writes an unread cell as the literal token ___. With on_na = "drop_frame" (the default) any frame containing one is removed whole, which is what keeps per-frame statistics comparable across a recording; on_na = "zero" keeps the frame and reads the cell as 0. A frame whose *timestamp* is missing is always dropped, since it cannot be placed on the time axis. The counts land in metadata$n_frames_raw and metadata$n_frames_dropped.

See Also

Other Pliance ingest functions: pr_meta_from_filename(), pr_read_dir()

Examples

```
# A miniature export in the Pliance layout: 7 header lines, then data.
tmp <- tempfile(fileext = ".asc")
writeLines(c(
  "Dateiname:  demo.mat\tDatum/Zeit: 01.01.25 09.00",
  "Kalibrationsdatei:  demo_cal",
  "gesamte Messzeit [Sek.]: 0.040\tZeit pro Bild [Sek.]:  0.02000\tMessfrequenz [Hz]:  50",
  "Druckwerte in  kPa",
  "",
  "elektrisch:",
  "Zeit [Sek.]\t1\t2\t3\t4",
  "0.02000\t10\t0\t0\t0",
  "0.04000\t0\t20\t0\t0"
), tmp)

lay <- pr_layout(
  2, 2, matrix(TRUE, 2, 2),
  data.frame(sensor_id = 1:4, row = c(1L, 2L, 1L, 2L),
             col = c(1L, 1L, 2L, 2L),
             x_mm = c(0, 0, 1, 1), y_mm = c(0, 1, 0, 1))
)

# Device channel 1 sits at grid (row 1, col 2), channel 2 at (1, 1).
```

```
cmap <- rbind(c(2L, 1L), c(4L, 3L))
trial <- pr_read_pliance(tmp, layout = lay, channel_map = cmap)
trial$time
trial$pressure
unlink(tmp)
```

---

pr_ref_diabetic_foot *Diabetic Foot Pressure Thresholds*

Description

Clinical thresholds for plantar-pressure risk assessment in patients with diabetic neuropathy.

Usage

```
pr_ref_diabetic_foot()
```

Value

A tibble::tibble with threshold values and sources.

Examples

```
pr_ref_diabetic_foot()
```

---

pr_ref_pci *Pressure Concentration Index Interpretation Bands*

Description

Working interpretation bands for the whole-grid PCI returned by `pr_calc_pci()`.

Usage

```
pr_ref_pci()
```

Details

The first six columns match the other `pr_ref_*` tables (`pr_ref_saddle()`, `pr_ref_diabetic_foot()`, `pr_ref_wheelchair()`), with `threshold` holding the lower edge of each band; `band`, `band_min` and `band_max` are appended so the interval can be used programmatically. `band_max` is `Inf` for the open top band.

Unlike the other reference tables, these bands are **not** taken from a published clinical threshold study. They are the working ranges used to describe whole-grid saddle-mat recordings, and they apply only to `pr_calc_pci(trial, denominator = "grid")`. PCI is dimensionless and scale-free in pressure, but it is *not* comparable across devices with different grid sizes, because the whole-grid mean depends on how much unloaded mat surrounds the contact patch.

Value

A tibble::tibble with columns region, parameter, threshold, unit, interpretation, source, band, band_min, band_max.

See Also

Other whole-grid frame metrics: pr_calc_cop_grid(), pr_calc_loaded_count(), pr_calc_mean_pressure_grid(), pr_calc_pci(), pr_calc_total_pressure(), pr_frame_metrics()

Examples

```
pr_ref_pci()
# Classify one recording:
pci <- pr_calc_pci(pr_example_trial("saddle_horse"))
ref <- pr_ref_pci()
ref$band[pci >= ref$band_min & pci < ref$band_max]
```

---

pr_ref_saddle *Saddle Fit Reference Thresholds*

Description

Returns published threshold values for equine saddle pressure assessment based on peer-reviewed studies.

Usage

```
pr_ref_saddle(source = c("vonpeinen2010", "monkemoller2005", "werner2002"))
```

Arguments

source	Character. Reference source: "vonpeinen2010" (default), "monkemoller2005", or "werner2002".

Value

A tibble::tibble with columns region, parameter, threshold, unit, interpretation, source.

Examples

```
pr_ref_saddle()
```

pr_ref_wheelchair *Wheelchair Seating Pressure Thresholds*

Description

Wheelchair Seating Pressure Thresholds

Usage

```
pr_ref_wheelchair()
```

Value

A tibble::tibble with threshold values and sources.

Examples

```
pr_ref_wheelchair()
```

pr_run_app *Launch the Interactive Pressure Data Explorer*

Description

Opens a Shiny application for importing, analyzing, and visualizing pressure measurement data. The app has six tabs: Import, Heatmap, Regions, Dynamics, Comparison, and Report.

Usage

```
pr_run_app(trial = NULL, ...)
```

Arguments

trial	A pr_trial object to pre-load. If NULL (default), the app starts empty and the user can load data from the Import tab.
...	Additional arguments passed to shiny::runApp().

Value

Invisible NULL. Called for side effects: launches the Shiny application.

Examples

```
if (interactive()) {
  pr_run_app(pr_example_trial("saddle_horse"))
}
```

`pr_sensor_map` *Per-Sensor Statistics Over Time*

Description

Reduces a recording along the time axis: one number per sensor, with the sensor's grid coordinates attached. This is the counterpart of `pr_frame_metrics()`, which reduces along the sensor axis instead.

Usage

```
pr_sensor_map(
  trial,
  statistic = c("mean", "max", "sd", "pti", "pct_zero"),
  threshold = 0
)
```

Arguments

`trial`	A pr_trial object.
`statistic`	Character. One of `"mean"`, `"max"`, `"sd"`, `"pti"` or `"pct_zero"`. See *Details*.
`threshold`	Numeric. Cells at or below this value count as unloaded. Default `0`.

Details

The available statistics, for a sensor read at every frame:

- `"mean"` — mean pressure, unloaded frames included in the denominator.
- `"max"` — the maximum pressure picture (MPP): the peak this sensor ever saw. `pr_sensor_map(trial, "max")$value` is the map `pr_plot_heatmap()` draws.
- `"sd"` — standard deviation over frames (`NA` for a single-frame trial), i.e. how much this sensor's load fluctuated.
- `"pti"` — pressure-time integral, trapezoidal over the trial's own timestamps; identical to `pr_calc_pti()` at the default threshold.
- `"pct_zero"` — fraction of frames in which the sensor was at or below `threshold`. `1` means never loaded, `0` means always loaded; it is the per-sensor duty cycle, and `1 - pct_zero` is the fraction of the recording the sensor was in contact.

`threshold` treats cells at or below it as unloaded, exactly as in `pr_frame_metrics()`: they are zeroed before `mean`, `sd` and `pti` are taken, they cannot raise `max`, and they are what `pct_zero` counts. At the default `threshold = 0` on non-negative data no masking is needed and none is done.

The `sensor` column is the *pressure column index*, taken positionally from `trial$layout$coords_mm`, not the device channel number. For a layout built by `pr_layout_from_index_map()` the two differ, and `pr_channel_order(trial$layout)[sensor]` recovers the device channel. Rows come back in that same column-major grid order, so `$value` can be used as a feature vector directly.

Value

A tibble::tibble with one row per sensor and columns `sensor` (integer), `row`, `col` (integer grid indices) and `value` (numeric).

See Also

`pr_frame_metrics()` for the per-frame reduction, `pr_profile_matrix()` to stack this across a dataset.

Other sensor aggregation functions: `pr_batch_frame_summary()`, `pr_profile_matrix()`

Examples

```
trial <- pr_example_trial("saddle_horse")
mpp <- pr_sensor_map(trial, "max")
mpp[which.max(mpp$value), ]

# The mean map is never above the peak map, sensor for sensor:
avg <- pr_sensor_map(trial, "mean")
all(avg$value <= mpp$value + 1e-9)

# Fraction of the recording each sensor spent in contact:
duty <- 1 - pr_sensor_map(trial, "pct_zero")$value
round(range(duty), 3)
```

---

pr_sensor_quality *Per-Sensor Hardware Quality Across a Cohort*

Description

Cohort-wide quality control for a sensor mat: one row per sensor, summarising how that cell behaved across every recording in the dataset, and a flag naming the failure mode if it has one.

Usage

```
pr_sensor_quality(
  dataset,
  dead_pct_zero = 95,
  saturation_kpa = NULL,
  low_var_sd = 0.1,
  threshold = 0
)
```

Arguments

`dataset`	A pr_dataset object, or a list of pr_trial objects.
`dead_pct_zero`	Numeric. Percentage (0–100) of frames at or below `threshold`, above which a sensor is called dead. Default `95`.

saturation_kpa	Numeric, or NULL (default) to take the layout's declared ceiling layout$pressure_range[2]. See *Trusting the ceiling*.
low_var_sd	Numeric. Lower bound on sd_across_files; below it a sensor is called low-variance. Default 0.1, in the trial's pressure unit.
threshold	Numeric. Cells at or below this value count as unloaded. Default 0.

Details

Each recording contributes four per-sensor numbers — its mean, its standard deviation over frames, its maximum, and the fraction of frames in which it read at or below threshold — and the cohort columns are built from those:

- mean_across_files — mean over recordings of the per-recording mean.
- sd_across_files — mean over recordings of the *within-recording* standard deviation, i.e. how much this cell typically moves during a trial. Despite the name (which is the cohort's own, kept so the table can be compared column-for-column with the published one) this is **not** the standard deviation of the per-recording means. The two are different quantities and they disagree substantially: the between-recording spread mostly measures how differently the mat was loaded from horse to horse, whereas the within-recording spread measures whether the cell responds to load at all — which is what the "Low variance" flag is asking. Recordings of fewer than two frames contribute no standard deviation and are left out of this average.
- max_across_files — the largest reading this cell ever produced, anywhere in the cohort.
- pct_zero_mean — mean over recordings of the percentage of frames at or below threshold. This column is a **percentage** (0–100), matching dead_pct_zero, not the 0–1 fraction that pr_sensor_map(trial, "pct_zero") returns.
- n_files — recordings summarised, the same on every row.

Value

A tibble::tibble with one row per sensor, in the layout's column-major grid order, and columns sensor (integer), row, col (integer grid indices), mean_across_files, sd_across_files, max_across_files, pct_zero_mean (percent), n_files (integer) and flag (character).

Flags

flag takes one of four fixed strings, assigned in this precedence order so a cell that qualifies for more than one gets the most serious:

1. "Dead (>95% zero)" — pct_zero_mean > dead_pct_zero. A cell that is unloaded in almost every frame of every recording is not measuring zero pressure, it is not measuring. Dead cells depress the whole-grid mean (pr_calc_mean_pressure_grid()) and are invisible to the loaded-cells-only pr_calc_mean_pressure().
2. "Saturated (>60 kPa)" — max_across_files >= saturation_kpa. The cell reached the hardware ceiling, so its true peak is unknown and censored from above. Any peak-derived cohort number (pr_calc_pci(), peak_kPa_max) is a lower bound wherever these cells carry the hotspot.
3. "Low variance" — sd_across_files < low_var_sd. The cell responds, but barely: a plausible stuck or half-detached cell that is not zero often enough to read as dead.

4. `"OK"` — none of the above.

The label text is fixed cohort vocabulary and names the study's nominal 60 kPa cut; the comparison actually performed uses `saturation_kpa`.

Trusting the ceiling

With `saturation_kpa = NULL` the cut-off is taken from `layout$pressure_range[2]`. **Check that number before relying on it.** Several built-in layouts carry a nominal range rather than a measured one: `pr_layout_saddle()` declares `c(0, 120)` for `"horse"` and `pr_layout()` defaults to `c(0, 600)`, while the Novel/Pliance saddle mat this cohort was recorded on saturates at **63.75 kPa** (255 x 0.25 kPa). `pr_layout_saddle_novel()` declares that correctly; the others do not, and a ceiling that is too high flags nothing at all. Pass `saturation_kpa` explicitly whenever the layout's own bound is not the device's measured one. A layout that declares `NA` — which `pr_layout_from_index_map()` does unless told otherwise — is an error rather than a guess.

Memory

Recordings are summarised one at a time into fixed-size accumulators, so peak memory is one recording's pressure matrix plus a handful of `n_sensors`-length vectors, not the cohort's. The study cohort is 431 recordings and about 5.5 M frames; holding them all to produce nine columns of 256 rows would cost several gigabytes.

See Also

`pr_sensor_map()` for the single-recording per-sensor statistics these are built from, `pr_batch_frame_summary()` for the per-recording side of the same cohort.

Other sensor quality functions: `pr_calc_gradient()`

Examples

```
ds <- pr_dataset(lapply(1:4, function(s) {
  pr_example_trial("saddle_horse", seed = s)
}))

# The layout declares 0-120 kPa, which this simulated mat never reaches,
# so nothing is flagged as saturated:
q <- pr_sensor_quality(ds)
table(q$flag)

# Lower the cut-off to the cohort's own 99th percentile and the busiest
# cells show up:
cut <- unname(stats::quantile(q$max_across_files, 0.99))
table(pr_sensor_quality(ds, saturation_kpa = cut)$flag)

# The declared ceiling and the pressure this mat actually reached --
# the comparison the *Trusting the ceiling* section asks you to make:
c(declared = ds$trials[[1]]$layout$pressure_range[2],
  observed = max(q$max_across_files))
```

pr_summary *Summarize Trial Pressure Parameters*

Description

Computes a comprehensive set of pressure distribution parameters for an entire trial. Returns a single-row tibble suitable for row-binding across trials in batch analysis.

Usage

```
pr_summary(trial, threshold = 0)
```

Arguments

trial	A pr_trial object.
threshold	Numeric. Threshold for "loaded" sensors. Default 0.

Value

A one-row tibble::tibble with columns:

- mpp: maximum peak pressure (kPa)
- mvp: mean of per-frame mean pressures (kPa)
- max_force: maximum total force (N)
- mean_force: mean total force (N)
- max_contact_area: maximum contact area (cm^2)
- mean_contact_area: mean contact area (cm^2)
- contact_time: total contact duration (s)
- pti_max: maximum PTI across sensors (kPa*s)
- pti_mean: mean PTI across loaded sensors (kPa*s)
- impulse: force-time integral (N*s)
- cop_path_length: total COP trajectory length (mm)
- cop_velocity_mean: mean COP velocity (mm/s)
- cop_range_ap: anterior-posterior COP range (mm)
- cop_range_ml: medial-lateral COP range (mm)

Examples

```
trial <- pr_example_trial("insole")
pr_summary(trial)
```

`pr_symmetry_sensitivity` *Asymmetry Index Across a Family of Masks*

Description

Recomputes `pr_calc_symmetry_map()` for every mask scheme in a list and returns one row per scheme, so that a symmetry claim can be reported together with how much it depends on where the region of interest was drawn.

Usage

```
pr_symmetry_sensitivity(
  trial,
  masks,
  statistic = "mean",
  denominator = "grid",
  balance = TRUE
)
```

Arguments

`trial`	A pr_trial object.
`masks`	A **named** list of mask schemes. Each element is a pr_mask, a logical matrix, or a list of exactly two of those giving the two sides explicitly — the same forms `pr_calc_symmetry_map()` accepts for its own `masks`.
`statistic`	Character. `"mean"` (default), `"max"` or `"loaded"`; see `pr_calc_regional_map()`.
`denominator`	Character. `"grid"` (default) or `"loaded"`; see *Details*.
`balance`	Logical. Trim each scheme to its largest mirror-symmetric subset before computing the index. Default `TRUE`.

Details

A left/right index is only as stable as its mask. Excluding dead sensors, restricting to a band, or thresholding on contact frequency all change which cells are compared, and each of those choices can move the index by more than the effect being reported. Running the family and printing the spread is cheap — the per-sensor map is computed once and reused for every scheme — and it turns an unstated choice into a reported one.

With `balance = TRUE` (the default) each scheme is passed through the mirror trim of `pr_mask_mirror_balance()` first, so every row compares equal, mirror-image sets of sensors and `n_left == n_right` throughout. Set it to `FALSE` to see what the raw masks give; the difference between the two runs is the part of the index that came from the mask rather than from the recording.

Value

A tibble::tibble with one row per scheme and columns `scheme`, `balanced`, `statistic`, `denominator`, `n_left`, `n_right`, `left_value`, `right_value`, `asymmetry_pct`.

See Also

pr_calc_symmetry_map() for a single scheme.

Other regional symmetry functions: pr_calc_cop_masked(), pr_calc_regional_map(), pr_calc_symmetry_map(), pr_mask_mirror_balance(), pr_mask_rowbands()

Examples

```
trial <- pr_example_trial("saddle_horse")
layout <- trial$layout
bands <- pr_mask_rowbands(layout, sides = FALSE)

schemes <- list(
  whole_mat = layout$active,
  cranial = bands$cranial,
  middle = bands$middle,
  caudal = bands$caudal
)
pr_symmetry_sensitivity(trial, schemes)[, c("scheme", "asymmetry_pct")]

# A data-driven mask need not be even-handed. Here the hotspot cells are
# 18 left against 12 right, and the raw index is a quarter of what the
# balanced 12/12 comparison reports.
hot <- matrix(FALSE, layout$grid_rows, layout$grid_cols)
co <- layout$coords_mm
hot[cbind(co$row, co$col)] <- pr_sensor_map(trial, "max")$value > 8
rbind(
  pr_symmetry_sensitivity(trial, list(hotspots = hot), balance = FALSE),
  pr_symmetry_sensitivity(trial, list(hotspots = hot))
)[, c("balanced", "n_left", "n_right", "asymmetry_pct")]
```

---

pr_trial *Create a Pressure Trial Object*

Description

Represents a single pressure measurement recording: a time series of pressure frames measured across a sensor layout.

Usage

```
pr_trial(pressure, time, layout, metadata = list(), sampling_hz = NULL)
```

Arguments

pressure	Numeric matrix of dimensions n_frames x n_sensors. Each row is one time frame, each column is one active sensor.
time	Numeric vector of length n_frames. Timestamps in seconds from recording start.

layout	A pr_layout object defining the sensor arrangement.
metadata	Named list of trial metadata. Expected (optional) fields: subject_id, trial_id, date, condition, system, notes.
sampling_hz	Numeric. Sampling rate in Hz. If NULL, computed from time.

Value

A pr_trial S3 object.

Examples

```
trial <- pr_example_trial("insole")
print(trial)
```

---

pr_validate_dataset *Validate Cohort Homogeneity*

Description

Checks that every trial in a dataset is analysable together: same sensor count, same layout name, non-empty, and free of missing, infinite or negative pressure values. Anything that would make a cohort-level summary meaningless is reported per trial.

Usage

```
pr_validate_dataset(dataset, strict = TRUE)
```

Arguments

dataset	A pr_dataset object, or a list of pr_trial objects.
strict	Logical. If TRUE (default), abort when any problem is found. If FALSE, warn and return the problem table.

Details

The reference sensor count and layout name are the *modal* values across the dataset (ties broken by first appearance), so a single odd recording is flagged rather than the other 430.

Value

A tibble::tibble with one row per problem and columns trial (integer index), trial_label, check, expected, actual and problem. Zero rows means the cohort is homogeneous. When strict = TRUE and problems exist, aborts with condition class pr_dataset_invalid, carrying the same tibble in the condition's problems field.

See Also

Other cohort functions: pr_assert_cohort(), pr_dataset_filter(), pr_design_table()

Examples

```
layout <- pr_layout_mat("16")
mk <- function() {
  pr_trial(matrix(1, 3, layout$n_sensors), time = c(0, 0.02, 0.04),
           layout = layout)
}
pr_validate_dataset(pr_dataset(list(mk(), mk())))

# A mixed-layout cohort is reported, not silently summarised:
odd <- pr_example_trial("insole")
problems <- pr_validate_dataset(pr_dataset(list(mk(), mk(), odd)),
                                strict = FALSE)
problems$check
```

`pr_validate_layout` *Validate a Pressure Sensor Layout*

Description

Checks internal consistency of a pr_layout object.

Usage

```
pr_validate_layout(layout)
```

Arguments

`layout`	A pr_layout object.

Value

Invisibly returns `TRUE` if valid; otherwise throws an informative error.

Examples

```
pr_validate_layout(pr_layout_insole())
```

`pr_validate_summary` *Two-Tier Data Frame Validation*

Description

Runs a set of named predicates over a data frame and returns them as a tibble of results. Rules passed as `hard` abort on failure; rules passed as `soft` warn. Everything is evaluated before anything is signalled, so one failing rule never hides the others.

Usage

```
pr_validate_summary(x, hard = list(), soft = list(), label = NULL)
```

Arguments

`x`	A data frame to validate.
`hard`	Named list of predicates whose failure aborts. Default `list()`.
`soft`	Named list of predicates whose failure warns. Default `list()`.
`label`	Character. Name of the table, used in the warning and error headers. `NULL` (default) uses `"data"`.

Details

The two tiers encode a distinction most validation code loses: some properties are structural (a duplicated key, a negative duration, an unknown condition code) and make every downstream number wrong, while others are worth knowing about but not worth stopping for (an unbalanced cell, an unusually short recording). Collapsing both into `stopifnot()` means either the pipeline halts for cosmetic reasons or the structural checks get commented out.

A rule is a function of the data frame returning either:

- a logical vector — `TRUE` throughout passes; any `FALSE` or `NA` fails, and the message counts the failing elements. This lets a rule be written per row (`function(d) d$n_frames > 0`) or as a single claim (`function(d) anyDuplicated(d$ID) == 0L`).
- a character vector — an empty one passes, and any strings are taken as the problem description. Use this when the rule can say *what* is wrong, not merely that something is.

A rule that throws is recorded as a failure carrying its error message rather than propagated: a validation run should report on a broken rule, not die inside it.

Value

A tibble::tibble with one row per rule and columns `rule`, `level` (`"hard"` or `"soft"`), `passed` (logical) and `message` (`NA` for a passing rule), hard rules first. When a hard rule fails, aborts with condition class `pr_validate_failed`, carrying that same tibble in the condition's `results` field. Soft failures warn with condition class `pr_validate_soft_failed`.

See Also

pr_validate_dataset() for the trial-level homogeneity checks.

Other provenance functions: pr_design_gaps(), pr_factor_contract(), pr_lock_table(), pr_verify_lock()

Examples

```
design <- data.frame(
  ID = c("ID001", "ID001", "ID003"),
  Mode = c("MH", "MS", "MG"),
  n_frames = c(23432L, 10662L, 8000L)
)

res <- pr_validate_summary(
  design,
  hard = list(
    positive_frames = function(d) d$n_frames > 0,
    mode_known = function(d) all(d$Mode %in% c("MG", "MH", "MS"))
  ),
  label = "design"
)
res

# A soft rule warns and is still reported. The real cohort is unbalanced
# across Mode (92 / 169 / 170 recordings), which is worth knowing but is
# not a reason to stop.
unbalanced <- rbind(design, design[2, ])
res2 <- suppressWarnings(pr_validate_summary(
  unbalanced,
  soft = list(balanced = function(d) {
    tab <- table(d$Mode)
    if (length(unique(as.integer(tab))) == 1L) {
      character(0)
    } else {
      paste0("Mode counts differ: ",
             paste(names(tab), as.integer(tab), sep = "=",
                   collapse = ", "))
    }
  }),
  label = "design"
))
res2$message

# A hard failure aborts:
try(pr_validate_summary(
  design,
  hard = list(unique_id = function(d) anyDuplicated(d$ID) == 0L)
))
```

`pr_verify_lock` *Verify a Table Against Its Lock File*

Description

Re-derives the properties `pr_lock_table()` recorded and reports exactly which of them changed: row count, column names, column types, or content.

Usage

```
pr_verify_lock(x, path)
```

Arguments

`x`	A data frame to check.
`path`	Path to a lock file written by `pr_lock_table()`.

Details

The four properties are checked independently, so the report separates failures that a shape check would conflate. A renamed column shows as `columns` changed with `content` unchanged. A row filtered out shows as `rows` changed. An `as.integer()` slipped into a pipeline shows as both `types` and `content`, because the storage type is part of the hashed byte stream. Same shape, same names, different numbers shows as `content` alone — the case a `dim()` check misses entirely.

This function never aborts and never warns on a mismatch: it returns the verdict and prints the report, leaving the caller to decide. Wrap it in `stopifnot()` to make a script fail, or add it as a rule to `pr_validate_summary()`. Errors *are* raised for the caller's own mistakes — a missing, malformed, or wrong-version lock file — because those are not evidence that the data changed.

Value

Invisibly, a single `TRUE`/`FALSE`. The result carries a `"report"` attribute: a four-row tibble::tibble with columns `property` (`"rows"`, `"columns"`, `"types"`, `"content"`), `expected`, `actual` and `passed`.

See Also

`pr_lock_table()` to write the lock file.

Other provenance functions: `pr_design_gaps()`, `pr_factor_contract()`, `pr_lock_table()`, `pr_validate_summary()`

Examples

```
design <- data.frame(
  ID = c("ID001", "ID001", "ID003"),
  Mode = c("MH", "MS", "MG"),
  n_frames = c(23432L, 10662L, 8000L)
)
```

```
path <- tempfile(fileext = ".lock")
pr_lock_table(design, path)

pr_verify_lock(design, path)

# One edited value: same shape, same names, different content.
edited <- design
edited$n_frames[1] <- 23431L
ok <- pr_verify_lock(edited, path)
ok
attr(ok, "report")

unlink(path)
```

Index

∗ **Pliance ingest functions**
  pr_meta_from_filename, 69
  pr_read_dir, 82
  pr_read_pliance, 86
∗ **channel map functions**
  pr_channel_order, 38
  pr_layout_from_index_map, 54
  pr_layout_saddle_novel, 59
∗ **cohort functions**
  pr_assert_cohort, 5
  pr_dataset_filter, 42
  pr_design_table, 44
  pr_validate_dataset, 98
∗ **provenance functions**
  pr_design_gaps, 43
  pr_factor_contract, 49
  pr_lock_table, 61
  pr_validate_summary, 100
  pr_verify_lock, 102
∗ **regional symmetry functions**
  pr_calc_cop_masked, 12
  pr_calc_regional_map, 26
  pr_calc_symmetry_map, 35
  pr_mask_mirror_balance, 65
  pr_mask_rowbands, 66
  pr_symmetry_sensitivity, 96
∗ **sensor aggregation functions**
  pr_batch_frame_summary, 6
  pr_profile_matrix, 79
  pr_sensor_map, 91
∗ **sensor quality functions**
  pr_calc_gradient, 16
  pr_sensor_quality, 92
∗ **temporal structure functions**
  pr_calc_dominant_freq, 14
  pr_calc_phase_map, 23
  pr_calc_spectrum, 30
  pr_calc_stride_cycles, 32
  pr_cop_shape, 39
∗ **whole-grid frame metrics**
  pr_calc_cop_grid, 11
  pr_calc_loaded_count, 18
  pr_calc_mean_pressure_grid, 20
  pr_calc_pci, 21
  pr_calc_total_pressure, 37
  pr_frame_metrics, 51
  pr_ref_pci, 88

make.unique(), *79*
mean(), *24*

plot.pr_layout, 4
plot.pr_layout(), *55*
plot.pr_trial, 5
pr_assert_cohort, 5
pr_assert_cohort(), *42*, *44, 45*, *98*
pr_batch_frame_summary, 6
pr_batch_frame_summary(), *79*, *92*, *94*
pr_batch_summary, 8
pr_calc_contact_area, 9
pr_calc_contact_area(), *19*
pr_calc_contact_time, 9
pr_calc_cop, 10
pr_calc_cop(), *11–13*, *39, 40*, *54*, *59*, *87*
pr_calc_cop_excursion, 10
pr_calc_cop_excursion(), *12, 13*
pr_calc_cop_grid, 11
pr_calc_cop_grid(), *12, 13*, *19*, *21, 22*, *37*, *39, 40*, *52*, *89*
pr_calc_cop_masked, 12
pr_calc_cop_masked(), *27*, *36*, *66, 67*, *97*
pr_calc_cop_path, 13
pr_calc_dominant_freq, 14
pr_calc_dominant_freq(), *24*, *32*, *34*, *40*
pr_calc_force, 15
pr_calc_force(), *37*
pr_calc_gait_cycles, 15
pr_calc_gait_cycles(), *28*, *33*, *73*
pr_calc_gradient, 16

pr_calc_gradient(), *94*
pr_calc_impulse, 18
pr_calc_loaded_count, 18
pr_calc_loaded_count(), *11*, *21, 22*, *37*, *52*, *89*
pr_calc_loaded_rate, 19
pr_calc_loaded_rate(), *19*
pr_calc_mean_pressure, 20
pr_calc_mean_pressure(), *7*, *11*, *19*, *21, 22*, *51*, *93*
pr_calc_mean_pressure_grid, 20
pr_calc_mean_pressure_grid(), *7*, *11*, *19*, *22*, *27*, *31*, *37*, *51, 52*, *89*, *93*
pr_calc_pci, 21
pr_calc_pci(), *7*, *11*, *19*, *21*, *37*, *52*, *88, 89*, *93*
pr_calc_peak_pressure, 22
pr_calc_phase_map, 23
pr_calc_phase_map(), *14*, *32*, *34*, *40*
pr_calc_pti, 25
pr_calc_pti(), *91*
pr_calc_regional, 25
pr_calc_regional(), *27*, *77*
pr_calc_regional_map, 26
pr_calc_regional_map(), *13*, *35, 36*, *66, 67*, *96, 97*
pr_calc_rollover, 28
pr_calc_saddle_bridge, 28
pr_calc_saddle_slip, 29
pr_calc_seat_hotspot, 30
pr_calc_spectrum, 30
pr_calc_spectrum(), *14*, *24*, *33, 34*, *40*
pr_calc_stride_cycles, 32
pr_calc_stride_cycles(), *14*, *23, 24*, *32*, *40*
pr_calc_symmetry_index, 34
pr_calc_symmetry_index(), *36*
pr_calc_symmetry_map, 35
pr_calc_symmetry_map(), *13*, *27*, *66, 67*, *96, 97*
pr_calc_total_pressure, 37
pr_calc_total_pressure(), *11*, *19*, *21, 22*, *31*, *52*, *89*
pr_channel_order, 38
pr_channel_order(), *54, 55*, *60*
pr_cop, *10*, *12*, 39
pr_cop(), *12*
pr_cop_shape, 39
pr_cop_shape(), *14*, *24*, *32*, *34*
pr_dataset, *6–8*, 41, *42*, *44*, *69*, *79*, *82, 83*, *92*, *98*
pr_dataset(), *83*
pr_dataset_filter, 42
pr_dataset_filter(), *6*, *45*, *98*
pr_design_gaps, 43
pr_design_gaps(), *50*, *62*, *101, 102*
pr_design_table, 44
pr_design_table(), *6*, *8*, *42*, *44*, *98*
pr_downsample, 45
pr_example_files, 46
pr_example_trial, 46
pr_export_csv, 47
pr_export_report, 48
pr_factor_contract, 49
pr_factor_contract(), *44*, *62*, *101, 102*
pr_filter_time, 50
pr_frame_metrics, 51
pr_frame_metrics(), *6–8*, *11*, *19*, *21, 22*, *37*, *39, 40*, *89*, *91, 92*
pr_interpolate, 52
pr_layout, *4*, *38*, 53, *54–60*, *63–66*, *68*, *80–83*, *85, 86*, *98, 99*
pr_layout(), *55*, *94*
pr_layout_from_index_map, 54
pr_layout_from_index_map(), *12*, *17*, *38*, *59, 60*, *91*, *94*
pr_layout_glove, 56
pr_layout_insole, 56
pr_layout_list, 57
pr_layout_mat, 57
pr_layout_platform, 58
pr_layout_saddle, 58
pr_layout_saddle(), *67, 68*, *94*
pr_layout_saddle_novel, 59
pr_layout_saddle_novel(), *38*, *55*, *94*
pr_layout_seat, 60
pr_lock_table, 61
pr_lock_table(), *44*, *50*, *101, 102*
pr_mask, *12*, *25, 26*, *35*, 62, *63–65*, *67, 68*, *76*, *81*, *85*, *96*
pr_mask(), *38*, *54*, *87*
pr_mask_apply, 63
pr_mask_default, 64
pr_mask_foot_auto, 64
pr_mask_foot_auto(), *58*
pr_mask_mirror_balance, 65

pr_mask_mirror_balance(), *13*, *27*, *36*, *67*, *96*, *97*
pr_mask_rowbands, 66
pr_mask_rowbands(), *13*, *27*, *36*, *66*, *97*
pr_mask_saddle_6, 67
pr_mask_saddle_6(), *28*, *59*, *67*
pr_mask_symmetry, 68
pr_mask_symmetry(), *29*, *35*, *59*, *67*
pr_merge_trials, 69
pr_meta_from_filename, 69
pr_meta_from_filename(), *83*, *87*
pr_plot_3d, 70
pr_plot_comparison, 71
pr_plot_contact_area, 72
pr_plot_cop, 72
pr_plot_cop_butterfly, 73
pr_plot_foot_report, 73
pr_plot_force_time, 74
pr_plot_heatmap, 74
pr_plot_heatmap(), *5*, *75*, *91*
pr_plot_heatmap_masked, 75
pr_plot_pressure_time, 76
pr_plot_regional_bar, 77
pr_plot_saddle_report, 77
pr_plot_symmetry, 78
pr_profile_matrix, 79
pr_profile_matrix(), *8*, *92*
pr_read_ascii, 80
pr_read_auto, 81
pr_read_csv, 81
pr_read_dir, 82
pr_read_dir(), *70*, *87*
pr_read_forcesensor, 84
pr_read_mask, 85
pr_read_pliance, 86
pr_read_pliance(), *70*, *83*
pr_ref_diabetic_foot, 88
pr_ref_diabetic_foot(), *88*
pr_ref_pci, 88
pr_ref_pci(), *11*, *19*, *21*, *22*, *37*, *52*
pr_ref_saddle, 89
pr_ref_saddle(), *77*, *88*
pr_ref_wheelchair, 90
pr_ref_wheelchair(), *88*
pr_run_app, 90
pr_sensor_map, 91
pr_sensor_map(), *8*, *16*, *17*, *24*, *27*, *79*, *94*
pr_sensor_quality, 92
pr_sensor_quality(), *17*
pr_summary, 95
pr_summary(), *8*
pr_symmetry_sensitivity, 96
pr_symmetry_sensitivity(), *13*, *27*, *36*, *66*, *67*
pr_trial, *5–16*, *18–23*, *25*, *26*, *28–32*, *35*, *37*, *41*, *42*, *44–48*, *50–52*, *63*, *64*, *69*, *71–82*, *84*, *86*, *87*, *90–92*, *95*, *96*, 97, *98*
pr_validate_dataset, 98
pr_validate_dataset(), *6*, *42*, *45*, *101*
pr_validate_layout, 99
pr_validate_layout(), *55*
pr_validate_summary, 100
pr_validate_summary(), *44*, *50*, *62*, *102*
pr_verify_lock, 102
pr_verify_lock(), *44*, *50*, *61*, *62*, *101*

serialize(), *61*
shiny::runApp(), *90*
stopifnot(), *100*, *102*

tibble::tibble, *7*, *8*, *11*, *14*, *16*, *17*, *24*, *26*, *28*, *30*, *31*, *33*, *36*, *40*, *44*, *45*, *49*, *51*, *57*, *61*, *62*, *77*, *88–90*, *92*, *93*, *95*, *96*, *98*, *100*, *102*
tidyr::expand_grid(), *43*
tools::md5sum(), *61*

utils::write.csv(), *61*