Package: mssrscopR (via r-universe)

November 3, 2026

Title Single-Frame Mean-Shift Super-Resolution Structure Maps

Version 0.2.0.9001

Description Computes the single-frame Mean-Shift Super-Resolution ('MSSR') structure map of Torres-Garcia et al. (2022) <doi:10.1038/s41467-022-34693-9> with an independently written implementation that keeps the input orientation, accepts non-square frames and defines the result for flat input. Provides output-grid coordinates, structural masks for measuring the original data, pixel-wise temporal statistics of the per-frame maps of an image sequence (mean, variance and lagged SOFI cumulants as defined in the article's supplement), a self-test against bundled synthetic fixtures and an interactive 'shiny' app for exploring parameters. Results are structural maps, not quantitative intensities; no gain in physical resolution or benefit for any application is implied.

License MIT + file LICENSE

URL https://github.com/CTTIR/mssrscopR, https://cttir.github.io/mssrscopR/, https://cttir.r-universe.dev/mssrscopR

X-schema.org-keywords microscopy, image-processing, mean-shift, temporal-statistics, structural-maps

BugReports https://github.com/CTTIR/mssrscopR/issues

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Imports cli (>= 3.0.0), graphics, grDevices, rlang, stats, utils, withr (>= 2.5.0)

Suggests bslib (>= 0.6.0), callr, digest, jsonlite, knitr, pkgload, png, rmarkdown, shiny (>= 1.8.0), testthat (>= 3.0.0), tiff (>= 0.1-8)

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/Needs/website CTTIR/themakR

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

Date/Publication 2026-11-03 13:18:29 UTC

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

RemoteRef HEAD

RemoteSha 2758f72ccc0c07e6558987e9c0e39700b3ea1588

Contents

inspect_mssr_result . . . 2
mssr . . . 3
mssr_app . . . 6
mssr_batch . . . 8
mssr_coords . . . 9
mssr_estimate_resources . . . 10
mssr_mask . . . 11
mssr_result_io . . . 12
mssr_selftest . . . 13
mssr_settings . . . 14
mssr_temporal . . . 15
mssr_validate_result . . . 19

Index **21**

`inspect_mssr_result` *Inspect a structure-map result*

Description

Inspect a structure-map result

Usage

```
## S3 method for class 'mssr_result'
summary(object, ...)

## S3 method for class 'summary.mssr_result'
print(x, ...)

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

Arguments

`object`, `x`	An `mssr_result`.
`...`	Further arguments. For plotting, passed to `graphics::image()`.

Value

`summary()` returns a named list of class `summary.mssr_result` with role, dimensions, grid, range, diagnostics, parameters and version identities; it prints as a short report. `plot()` draws the stored map in input-pixel coordinates and invisibly returns the result. It does not transform the stored values or infer physical sampling.

Examples

```
r <- mssr(matrix(1:20, 4, 5), fwhm = 2)
summary(r)
plot(r)
```

`mssr` *Single-frame Mean-Shift Super-Resolution*

Description

Computes the Mean-Shift Super-Resolution (MSSR) structure map of one 2D frame (Torres-Garcia et al. 2022). The result is a structural map for display, structure analysis or mask derivation. It is **not** a quantitative intensity: MSSR is nonlinear, suppresses dim structures next to bright ones and can hollow or split broad objects. Take measurements from the original data (see `mssr_mask()`).

Usage

```
mssr(
  x,
  fwhm,
  amp = 1L,
  order = 1L,
  intensity_normalize = TRUE,
  mesh = TRUE,
  grid = c("extent", "endpoint"),
  kernel = c("disk", "square"),
  rounding = c("half_even", "half_up"),
  max_output_pixels = 5e+07,
  max_memory_bytes = NULL
)
```

Arguments

`x`	A plain integer or double matrix (rows = y, columns = x) with finite values and no explicit class. Input dimnames and other attributes are discarded; acquisition metadata are never inferred from attributes.
`fwhm`	Full width at half maximum of the point-spread function, in **input pixels**. Required; there is no default because it is a property of the acquisition. Prefer a measured PSF: a theoretical value such as `0.51 * wavelength / NA` (emission wavelength, divided by the pixel size) describes an ideal widefield system, and aberrations, defocus and pixel size can widen the recorded PSF. Under noise an `fwhm` below the true PSF makes false peaks more likely, while a larger one gives a shallower dip between close sources; when unsure, choose the larger plausible value. An `fwhm` larger than the PSF's FWHM enlarges the neighborhood and gives smoother maps with fewer fragments; choose it per data set and record it. Whatever the value, MSSR maps are visual aids, not a basis for segmentation or quantification. Only the kernel half-size, `0.5 * fwhm * amp` rounded as set by `rounding` and at least 1 (recorded in `parameters$half_size`), enters the computation, so nearby values can give identical results.
`amp`	Amplification factor (whole number from 1 to 64).
`order`	MSSR order (whole number from 0 to 10). Orders above 3 are experimental: bounded high-precision checks found larger errors at order 10. They emit `mssrscopR_warning_unqualified_order`. Checks at orders 0 through 3 cover only the documented synthetic domain, not every input.
`intensity_normalize`	If `TRUE` (the authors' default) the structure map is multiplied by the (amplified) input.
`mesh`	Mesh compensation after amplification (only used if `amp > 1`).
`grid`	`"extent"` (field-of-view-preserving, default) or `"endpoint"` (samples from the first to the last input pixel center, as the authors' code does; for comparison only).
`kernel`	Neighborhood shape: `"disk"` (authors' R code) or `"square"` (authors' Python code).
`rounding`	Rounding of the kernel half-size `0.5 * fwhm * amp`: `"half_even"` (R and Python default) or `"half_up"`.
`max_output_pixels`	Upper limit on output pixels, checked before any computation.
`max_memory_bytes`	Optional positive allocation-estimate ceiling. `NULL` leaves this guard disabled. This is not a hard process-memory cap; see `mssr_estimate_resources()`.

Details

Geometry. The output keeps the input orientation. With `grid = "extent"` the amplified grid preserves the field of view: output pixel centers lie at `(k - 0.5) / amp + 0.5` input pixels, so the spacing is exactly `1 / amp`. Amplification adds samples, not independent observations.

Relation to the authors' code. At `amp = 1` with `intensity_normalize = FALSE`, tested non-degenerate examples agree with the authors' single-frame R function after its final 90-degree rotation is undone (`kernel = "disk"`) or their Python function (`kernel = "square"`) within floating-point tolerance. Extreme numeric ranges and undefined refinement steps can differ (see below). For `amp > 1`, `mssr()` uses the supplement's cubic kernel (Keys cubic convolution, a = -1/2). The authors' R script interpolates with akima's Akima method and their Python script with a SciPy spline, and their R mesh shifts asymmetrically, so the amplified input already differs beyond rounding; MSSR's range normalization and refinement are nonlinear and can enlarge such differences locally. Do not expect pixel-wise agreement with their amplified maps, even with `grid = "endpoint"`, which reproduces their sample positions. With the default `intensity_normalize = TRUE` their R function multiplies the rotated map by the unrotated input, so it agrees with `mssr()` only for frames that a 90-degree rotation leaves unchanged (and fails for non-square frames).

Several channels or bands. `mssr()` processes one 2D frame. Applied to each channel or band of multichannel or spectral data, it gives every channel its own nonlinear structure map, scaled by that channel's maxima. Intensity-weighted outputs (the default) are guaranteed to keep the channels' intensity ratios only when one whole frame, background included, is a scaled copy of another; a different background or another object elsewhere in the frame can change the ratios, and the structure maps (`intensity_normalize = FALSE`) of scaled copies are identical. So do not compare channels, form ratios or indices, or assemble spectra from the outputs; measure them on the original data.

Degenerate input. A flat frame, a frame whose amplified maximum is not positive, a frame without any positive mean shift and a refinement step that leaves no positive value (possible only with signed input) return zeros with `diagnostics$degenerate = TRUE` and a `degenerate_reason`. A refinement step (`order >= 1`) is undefined when the mean-shift map already equals the normalized input, for example for isolated impulses on an exactly zero background. Such a step leaves the map unchanged, is counted in `diagnostics$order_steps_skipped` and gives a warning of class `mssrscopR_warning_skipped_steps`: the result is not the limit for slightly perturbed input, because any positive background makes the step defined and changes the map. The authors' R function returns `NaN` in all of these cases; their Python function returns zeros, which equals `mssr()` for the degenerate frames but not for skipped refinement steps.

Negative values and background. MSSR assumes non-negative intensities. Negative input gives a warning of class `mssrscopR_warning_negative`, and intensity-weighted output can then be negative. Non-negative input is not free of offset effects either: the order-0 structure map (`intensity_normalize = FALSE`) does not change with a constant background, but every refinement step (`order >= 1`) compares the map with the input divided by its maximum, which a background changes, and intensity weighting multiplies by the input. Subtracting a background is therefore a preprocessing choice that changes the result; record it with the result. The range normalizer adapts to the largest local difference at each pixel, so even rounding-level ripple on an otherwise flat frame is stretched to full contrast: only an exactly flat frame is degenerate.

Numeric domain. Input and numeric parameters must use plain integer or double storage, without explicit classes. Convert calibrated or other classed data deliberately before calling. The engine uses internal power-of-two scaling, centered interpolation for non-negative input and weighted local differences to limit overflow, underflow and cancellation. A nonzero input lost during scaling or an overflowing final result raises `mssrscopR_error_numerical`. Rounding already present in the supplied data cannot be recovered. Intensity-weighted output is rescaled to the original numeric units but remains non-quantitative; subnormal output still has limited precision and can round to zero.

Cost. Every order step uses image-wide maxima, so frames must be processed whole. Time grows with the number of output pixels times the number of neighborhood offsets (about `pi * h^2` for the disk kernel and `(2 * h + 1)^2 - 1` for the square kernel, with half-size `h`, recorded in `parameters$half_size`). Calls whose estimated work exceeds 1e10 operations, whose output exceeds `max_output_pixels`, or whose kernel half-size exceeds 1000 pixels are refused before computing.

Value

An object of class `mssr_result`: a list with `values` (numeric matrix), `role` (`"structure"` or `"intensity_weighted_structure"`), `transform`, `parameters`, `engine` and `diagnostics`. `transform` holds `amp`, `grid`, `input_dim`, `output_dim`, the first output pixel center and the spacing (input-pixel units), and `field_of_view`: the input frame's extent from pixel edge 0.5 to `n + 0.5`. Output samples cover it exactly on the extent grid; on the endpoint grid they span only the first to the last input pixel center. The input is not stored. `schema_version` identifies the result contract separately from the numerical engine; `input` records known axes and unknown calibration; `evidence` identifies the numerical-only scope. Qualification diagnostics disclose conservative high-order limits without assigning confidence.

References

Torres-Garcia E, Pinto-Camara R, Linares A, et al. Extending resolution within a single imaging frame. *Nature Communications* 2022;13:7452. doi:10.1038/s41467022346939

See Also

`mssr_coords()`, `mssr_mask()`, `mssr_selftest()`, and `mssr_temporal()` for temporal statistics of a frame sequence

Examples

```
# Two point sources 3 pixels apart, blurred with FWHM 3 px.
g <- expand.grid(r = 1:21, c = 1:25)
spot <- function(r0, c0) exp(-((g$r - r0)^2 + (g$c - c0)^2) / (2 * (3 / 2.3548)^2))
x <- matrix(spot(11, 11) + spot(11, 14) + 0.05, 21, 25)
res <- mssr(x, fwhm = 3, amp = 2)
res
dim(as.matrix(res))
```

---

mssr_app *Interactive MSSR explorer*

Description

Opens a Shiny app for trying `mssr()` on synthetic examples or on an uploaded frame. It shows the input and the structure map side by side on the same input-pixel axes, line profiles, structural masks applied to the original data (`mssr_mask()`) and the fixture self-test (`mssr_selftest()`), and offers the structure map, the mask, a figure and a run record for download. The app calls the exported functions, so its results are the same as in R; it adds no processing of its own.

Usage

```
mssr_app(max_upload_mb = 50, ...)
```

Arguments

max_upload_mb	Maximum upload size in megabytes.
...	Passed to shiny::runApp(), for example port or launch.browser.

Details

Uploads. Delimited text (.csv comma-separated, .tsv or .txt separated by tabs or spaces; numbers only, no header or comments), PNG (needs the png package; values as returned by png::readPNG(), scaled to 0..1) and TIFF (needs the tiff package; stored values, unscaled). Multi-channel images are not converted to gray: choose one channel. For multi-page TIFF choose a page. Files are read by the R process that runs the app (your own session when started with mssr_app()); the app sends nothing elsewhere. Run records include SHA-256 identities of uploaded bytes and decoded matrix content (needs digest). Source filenames and paths are excluded. Retain the original input and preprocessing separately; a digest identifies content, not acquisition authenticity. Compare a retained file with digest::digest(file = path, algo = "sha256", serialize = FALSE). CSV downloads retain 17 significant digits. Text uploads decode decimal samples with jsonlite so finite values near the binary64 limit survive export and re-upload independently of the platform's base R scanner.

Limits. To keep the app responsive, runs whose estimated work exceeds 5e8 operations are refused; call mssr() directly for larger runs. The image header is checked before decoding; image uploads are limited to 5e7 samples, conservatively allowing at least four channels per pixel. Only the selected TIFF page is decoded. Integer TIFF samples keep their stored values; floating-point TIFF samples remain unscaled. The structure map is not a quantitative intensity, and nothing in the app implies a gain in physical resolution. Packed 12-bit, signed-integer, palette, multi-channel planar-separate and volumetric TIFF images are unsupported; prepare an explicit supported 2D frame before uploading. The TIFF preflight accepts classic TIFF, top-left orientation, grayscale black-is-zero or RGB samples, at most 256 pages, 4096 tags per page, 4 MiB of directory/tag metadata and a 256 MiB file. BigTIFF, nested directories and other orientations/color conversions are refused. These are workload guards, not a hard decoder memory or time guarantee. Computation runs in one background R process per app session (needs callr). Cancel stops only that session's job. Incomplete results cannot be downloaded; the previous completed result can be explicitly restored. A job is cancelled when its session closes or after five minutes. The child inherits CPU affinity and uses one numerical thread. Changed settings are marked stale; downloads retain the computed input and settings. Development-source launches also need pkgload. A job is refused when package files, functions or the native library differ from what the R session loaded, instead of running other code.

Value

The value of shiny::runApp(), invisibly; called for its side effect.

Examples

```
if (interactive()) {
```

```
  mssr_app()
}
```

`mssr_batch` *Process and resume independent complete frames*

Description

Processes an explicitly named list of matrices sequentially. Each frame uses the same settings and the public `mssr()` function. Completed files are reused only after input/settings identity, SHA-256 and result readback checks. Changed input or settings creates a new identity and preserves prior output. Failed frames remain in the returned table. No data are downloaded or sent.

Usage

```
mssr_batch(
  frames,
  output_dir,
  settings,
  resume = TRUE,
  max_memory_bytes = 2 * 1024^3,
  max_result_bytes = 256 * 1024^2,
  cancel = NULL
)
```

Arguments

`frames`	A nonempty named list of plain numeric matrices. Names are unique opaque IDs containing only letters, digits, underscore or hyphen.
`output_dir`	Explicit directory for result files and recovery records.
`settings`	Named list of `mssr()` arguments, including `fwhm`; `x` and `max_memory_bytes` are supplied by the runner and cannot be overridden.
`resume`	Reuse matching, verified completed results.
`max_memory_bytes`	Positive estimated-memory budget; default 2 GiB. A frame runs only if its `mssr_estimate_resources()` estimate plus the size of all input frames held in `frames` fits within it.
`max_result_bytes`	Positive serialized-result limit; default 256 MiB.
`cancel`	Optional function with no arguments returning a single logical. `TRUE` marks remaining frames cancelled without running them.

Details

Requires optional packages `digest` and `jsonlite`, as do result files. A directory lock prevents simultaneous writers. A lock left by a terminated process must be reviewed and removed manually once its owner has stopped. Cancellation is checked between frames; it does not interrupt a numerical call already in progress. An interrupted output is retained and a new attempt receives a distinct filename. Keep this output directory local and trusted; hashes establish consistency, not authenticity.

All input matrices are already resident in memory. The budget adds their storage to the per-frame allocation estimate; it is not an OS memory cap. `workers` is deliberately absent: parallel processing remains unqualified.

Value

A data frame with one row per input: ID, status, reuse flag, output path, elapsed seconds and message. Individual JSON records retain warnings, identities, completion state and hashes. Complete results use `.mssr` files.

`mssr_coords` *Output pixel-center coordinates*

Description

Coordinates of the output pixel centers of an `mssr()` result, in input-pixel units (input pixel centers at 1..n). Use them to overlay results on the input grid; amplification adds samples, not independent observations.

Usage

```
mssr_coords(result)
```

Arguments

`result`	An `mssr_result`.

Value

A list with numeric vectors `row` and `col`.

Examples

```
r <- mssr(matrix(runif(20), 4, 5), fwhm = 2, amp = 2)
mssr_coords(r)$col
```

`mssr_estimate_resources` *Estimate whole-frame MSSR resources*

Description

Reports a conservative allocation model before allocating an output frame. The estimate includes a 128 MiB runtime allowance, 24 double buffers per output pixel, six per input pixel, and reflected-index and offset storage. It was calibrated on the peak resident memory of fresh R processes on one Linux machine over a workload ladder of 65,536 to 16,777,216 output pixels (amplification 1 to 4, orders 0 to 3, mesh and intensity weighting on and off, disk and square kernels), where it exceeded every measured peak by 30 to 220 percent; the margin is smallest for order 3 with mesh and weighting, and garbage-collection timing changes peaks from run to run. It is a planning guard, not a hard process-memory limit: other platforms, loaded packages, retained objects, garbage collection and the allocator can change peak RSS, and estimates outside that ladder are extrapolations.

Usage

```
mssr_estimate_resources(
  input_dim,
  fwhm,
  amp = 1L,
  kernel = c("disk", "square"),
  rounding = c("half_even", "half_up"),
  max_output_pixels = 5e+07,
  frames = NULL,
  statistic = NULL
)
```

Arguments

`input_dim`	Two positive whole numbers: input rows and columns.
`fwhm`, `amp`, `kernel`, `rounding`, `max_output_pixels`	As in `mssr()`.
`frames`, `statistic`	Optional: the number of frames and the statistic of a temporal call (see `mssr_temporal()`). Give both or neither.

Details

With `frames` and `statistic`, the list also estimates a call of `mssr_temporal()` (model `temporal_buffers_v4`): `stack_bytes` for the stored per-frame maps, `stored_peak_bytes` and `recompute_peak_bytes` for its two computation paths (the single-frame peak plus the output-sized buffers each path holds, including eight additional exponent/scaling and arithmetic-check work arrays for per-pixel conditioning, the raw temporal mean and one raw frame), and the total neighborhood work of each path. The stored path counts the maps twice (a 100 percent margin): R lets garbage accumulate in proportion to the live heap before it collects, and on long stored stacks the measured peak came close to

twice the bare map size. The input array is read frame by frame and not copied, so it is not counted. The per-frame limits of `mssr()` apply to every frame.

Value

A list with input/output dimensions, pixels, offsets, estimated work, estimated peak bytes, model ID, and limitations; with `frames` and `statistic`, also the temporal estimates described above.

Examples

```
mssr_estimate_resources(c(128, 192), fwhm = 3, amp = 2)
mssr_estimate_resources(c(128, 192), fwhm = 3, amp = 2, frames = 100, statistic = "var")
```

---

mssr_mask *Structural mask on the input grid*

Description

Selects exactly `ceiling(fraction * n)` pixels of a region with the highest MSSR values, on the **input** grid (amplified results are averaged back over each input pixel's children; this needs `grid = "extent"`). Pixels tied at the cut are chosen by a seeded random permutation, without changing the caller's random-number state or depending on its `RNGkind()`. Floating-point noise within eight machine epsilons relative to the quota is snapped to a positive integer before rounding up. Every positive fraction selects at least one pixel. Flat maps can yield masks chosen entirely by ties; such masks are not detected foreground or cell instances.

Usage

```
mssr_mask(result, fraction, region = NULL, seed = 1L)
```

Arguments

`result`	An `mssr_result`.
`fraction`	Fraction of the region to keep, in (0, 1]. Required: a suitable share depends on how much of the map is positive, which differs from frame to frame, so there is no default.
`region`	Optional logical matrix with the input dimensions; defaults to the whole frame.
`seed`	Whole number used for tie-breaking.

Details

Use the mask to measure the **original** data, e.g. `mean(x[mask])`. A mask changes which pixels are measured; it does not make MSSR values quantitative, and selecting pixels by MSSR is not known to improve any downstream analysis. MSSR maps often contain many exact zeros, so check `tie_fraction`: it is the share of the selected pixels that were decided by random tie-breaking (0 when the cut falls between distinct values; only exact ties count). Also check `cut_relative`, the lowest selected score divided by the highest score in the region. Interpolation and rounding leave

many tiny positive values (often below 1e-6 of the maximum) where MSSR found no structure; they rank above exact zeros without counting as ties, so a small `cut_relative` means the mask reaches beyond the structure. A warning of class `mssrscopR_warning_mask_ties` is given when tie-breaking chose more than half of the selected pixels or `cut_relative` is at most 1e-6; the mask is still returned. A `fraction` near the share of positive map values, or a `region`, avoids it. Both mask and coordinate consumers reject nonfinite values or inconsistent grid metadata. This checks the stored matrix/grid contract, not authenticity of the image, parameters or review status.

Value

A logical matrix (input dimensions) with attributes `coverage` (selected share of the region), `tie_fraction`, `cut_relative` (0 when no score in the region is positive) and `n_region`.

Examples

```
x <- matrix(runif(400), 20, 20)
r <- mssr(x, fwhm = 3)
mean(as.matrix(r) > 0)  # share of pixels with any structure
m <- mssr_mask(r, fraction = 0.1)
attr(m, "tie_fraction")
attr(m, "cut_relative")
mean(x[m])
```

---

mssr_result_io *Write and read an exact, bounded result record*

Description

Stores only the documented result contract, with exact binary double values, typed JSON metadata and SHA-256 corruption detection. No R expressions, serialized R objects, source image, source path or acquisition identity are stored. A matching checksum is not proof of origin or scientific validity.

Usage

```
mssr_write_result(result, path, overwrite = FALSE, max_bytes = 64 * 1024^2)

mssr_read_result(path, max_bytes = 64 * 1024^2, max_output_pixels = 5e+07)
```

Arguments

`result`	A validated result from `mssr()`.
`path`	Explicit path to a single result file; its directory must exist.
`overwrite`	Replace an existing file, if the filesystem supports atomic replacement. Defaults to `FALSE`.
`max_bytes`	Maximum total record size in bytes, below 2^31 - 1.
`max_output_pixels`	Maximum decoded matrix size when reading.

Details

Optional packages `jsonlite` and `digest` are required only for these functions. The writer stages and reads back the complete record before installing it. With `overwrite = FALSE`, installation uses an exclusive hard link; filesystems without hard-link support fail without writing a completed destination. With `overwrite = TRUE`, atomic rename must succeed; an existing file is never deleted as a fallback. Symbolic-link file paths are refused.

The reader checks file size, a fixed 1 MiB metadata ceiling, metadata nesting, schema, dimensions, exact payload length and SHA-256 before returning a result. It reads at most `max_bytes` plus one byte before decoding. The record and decoded matrix coexist in memory, so the byte ceiling is not a process memory cap. No compression or external references are accepted.

Value

The writer invisibly returns the normalized file path. The reader returns the stored result without changing its schema or engine identity.

Examples

```
if (requireNamespace("jsonlite", quietly = TRUE) &&
  requireNamespace("digest", quietly = TRUE)) {
  r <- mssr(matrix(1:20, 4, 5), fwhm = 2)
  path <- tempfile(fileext = ".mssr")
  mssr_write_result(r, path)
  identical(mssr_read_result(path), r)
  unlink(path)
}
```

---

mssr_selftest *Check the engine against the bundled synthetic fixtures*

Description

Recomputes every bundled synthetic fixture and compares it with the stored expected output. The fixtures are small non-square, impulse, constant, zero, point-pair, ring and ramp images. The expected outputs were produced by the package's engine definition, so this is a regression check of the installation, not an independent validation of the method or of any application.

Usage

```
mssr_selftest(tolerance = 1e-10)
```

Arguments

`tolerance`	Single non-negative number.

Details

A fixture passes when its dimensions match and `max(abs(got - expected)) <= tolerance * (1 + max(abs(expected)))`.

Value

A data frame with one row per fixture (`fixture_id`, parameters, `max_abs_error`, `passed`), with attribute `all_passed` (`TRUE` or `FALSE`).

Examples

```
st <- mssr_selftest()
attr(st, "all_passed")
```

`mssr_settings` *Extract resolved computation settings from a result*

Description

Validates a result and returns arguments that can be passed to `mssr()` with the retained original input. The grid is included explicitly; derived kernel sizes and resource limits are not computation settings returned here.

Usage

```
mssr_settings(result)
```

Arguments

`result`	A result returned by `mssr()`, `mssr_temporal()` or `mssr_read_result()`.

Details

These are resolved settings, not the original call. For example, `mesh` is `FALSE` when amplification is 1 because it had no effect. Records from a different numerical engine are refused: this helper does not migrate historical engines. A legacy result schema is accepted when its engine matches and its stored contract validates.

Retain the original input, preprocessing and channel/frame selection, acquisition information, and mask region, fraction and seed separately. Result files do not contain them. Set resource budgets explicitly when recomputing. Preserve the package and runtime environment, check engine identity, and compare recomputed values; matching settings alone do not guarantee identical arithmetic across environments or versions. This function needs no optional packages and does not modify the result.

Value

A named list containing `fwhm`, `amp`, `order`, `intensity_normalize`, `mesh`, `grid`, `kernel` and `rounding`. For a temporal result it also starts with `statistic` and `frames` (the frame indices of the original input), plus `sign` for the SOFI statistics, so that it can be passed to `mssr_temporal()` together with the original raw frames (not with an `mssr_stack`, which refuses per-frame settings).

Examples

```
x <- matrix(seq_len(35), 5, 7)
result <- mssr(x, fwhm = 2.5, amp = 2, grid = "endpoint")
replay <- do.call(mssr, c(list(x = x, max_memory_bytes = 1024^3),
                          mssr_settings(result)))
identical(replay$values, result$values)
```

`mssr_temporal` *Temporal statistics of per-frame MSSR maps*

Description

Computes single-frame MSSR (`mssr()`) for every frame of an image sequence and reduces the stack of maps pixel by pixel with one temporal statistic, as in the temporal MSSR (t-MSSR) of Torres-Garcia et al. (2022, Fig. 7a and Supplementary Note 5.2). The output is a temporal statistic of nonlinear structure maps: it is not a quantitative intensity, and no gain in resolution is implied (see **Interpretation**). `mssr_stack()` stores the per-frame maps so that several statistics can be computed without recomputing MSSR.

Usage

```
mssr_temporal(
  x,
  statistic,
  fwhm,
  amp = 1L,
  order = 1L,
  intensity_normalize = TRUE,
  mesh = TRUE,
  grid = c("extent", "endpoint"),
  kernel = c("disk", "square"),
  rounding = c("half_even", "half_up"),
  frames = NULL,
  sign = c("signed", "absolute"),
  background = NULL,
  max_output_pixels = 5e+07,
  max_memory_bytes = NULL
)

mssr_stack(
  x,
  fwhm,
  amp = 1L,
  order = 1L,
  intensity_normalize = TRUE,
  mesh = TRUE,
  grid = c("extent", "endpoint"),
```

```
  kernel = c("disk", "square"),
  rounding = c("half_even", "half_up"),
  frames = NULL,
  max_output_pixels = 5e+07,
  max_memory_bytes = NULL
)
```

Arguments

x	Either a plain numeric array with dimensions rows x columns x frames (rows = y, columns = x) and finite values, or an `mssr_stack` from `mssr_stack()`. For an `mssr_stack`, the per-frame arguments (`fwhm` to `rounding` and `max_output_pixels`) must not be given; the stored maps already fix them.
statistic	One of `"mean"`, `"var"`, `"tpm"`, `"cv"`, `"sofi2"`, `"sofi3"` or `"sofi4"`. Required: none of them is a safe default.
fwhm, amp, order, intensity_normalize, mesh, grid, kernel, rounding, max_output_pixels	Per-frame MSSR settings, exactly as in `mssr()`. `fwhm` is required for raw frames.
frames	Optional contiguous, increasing frame indices; the lags need consecutive frames. Defaults to all frames.
sign	For `"sofi2"`, `"sofi3"` and `"sofi4"`: `"signed"` (default) returns the cumulant, `"absolute"` its absolute value. Other statistics accept only `"signed"`.
background	Optional logical matrix on the input grid that marks signal-free pixels for `diagnostics$mesh_index`. By default these are the pixels whose raw temporal mean is below the median of that mean. Because the mesh index is formed only with `amp > 1` on the extent grid, giving `background` otherwise is an error.
max_memory_bytes	Optional positive allocation-estimate budget (see **Computation**). This is not a hard process-memory cap. For an `mssr_stack`, whose maps already exist, it limits the reduction's buffers.

Details

Definitions. Per output pixel, `x_1, ..., x_M` are the values of the `M` selected frames' MSSR maps, `m` is their mean and `d_t = x_t - m`. Following Supplementary Table S2:

statistic	Value	Minimum `M`
`"mean"`	`m`	1
`"var"`	`(1/M) sum(d_t^2)`	2
`"tpm"`	`(sum(x_t))^2`	1
`"cv"`	`var / m`, and 0 where `m = 0`	2
`"sofi2"`	`(1/(M-1)) sum_{t=1}^{M-1} d_t d_{t+1}`	2
`"sofi3"`	`(1/(M-2)) sum_{t=1}^{M-2} d_t d_{t+1} d_{t+2}`	3
`"sofi4"`	`A(0,1,2,3) - A(0,1)A(2,3) - A(0,2)A(1,3) - A(0,3)A(1,2)`	4

For `"sofi4"`, `A(L)` is the average over `t = 1, ..., M - 3` of the product of `d_{t+l}` for the lags `l` in `L`; all its terms use the same frames.

- `"var"` is the population variance; the sample variance is larger by `M / (M - 1)`.
- `"tpm"` equals `M^2 * mean^2`. For non-negative maps it is a monotone function of the mean and shows nothing that `"mean"` does not; it is offered because every implementation by the authors computes it.
- `"cv"` is the supplement's "CV", literally `Var / Mean`: an index of dispersion, not the standard deviation divided by the mean. It takes the sign of the mean, so it is negative where the mean is negative (signed input). It is 0 where the mean, formed in the scaled range described under **Computation**, is exactly 0; where the mean is tiny compared with the variance the ratio becomes very large, and a ratio beyond the numeric range is refused.
- `"sofi2"` to `"sofi4"` are lagged auto-cumulants of signed deviations. Noise that is independent from frame to frame does not enter their expectation. Orders 3 and 4 change sign with the emitters' duty cycle, so displays usually need `sign = "absolute"`.

The "SOFI" outputs of the authors' Python, MATLAB and Fiji tools are other statistics, so they do not match these values by design. The Python and MATLAB tools use absolute deviations at zero lag, which makes their order 2 the variance without the first frame. The Fiji plugin uses absolute deviations with lags that wrap around the ends of the acquisition. Their variance divides by `M - 1` except in the Python tool. See `vignette("method-concordance")`.

Interpretation. `"var"`, `"cv"` and the SOFI statistics depend on how the emitters blink and on the noise, and every map is a nonlinear function of its frame. Treat the outputs as visual aids:

- Variance, `"cv"` and SOFI of per-frame MSSR maps, and higher MSSR orders, can show two maxima where there is a single emitter.
- When emitters are rarely on and dim, MSSR of nearly empty frames adds maxima to every statistic.
- With `amp > 1`, signal-free regions show a sub-pixel grid with the period `amp`; `diagnostics$mesh_index` reports its size there (see **Value**).
- The article recommends Fourier interpolation for temporal analysis. The package amplifies with the supplement's cubic kernel and mesh compensation instead; Fourier interpolation is not implemented.

No drift, bleaching or background correction is applied. `"var"`, `"cv"` and the SOFI statistics treat slow intensity trends (bleaching, drift, focus changes) as fluctuation, so a large trend inflates them. Correct the frames first, and record what was done; `diagnostics$trend` reports the ratio of the raw intensity at the end and at the start of the selected frames.

Computation. Every statistic is formed from centered deviations in frame order, in a separate power-of-two scaled range for each pixel so that unrelated bright pixels cannot erase dim pixels' representable statistics. The raw temporal mean uses the same conditioning to avoid overflowing an intermediate sum. A nonfinite diagnostic field mean from base R is retried in power-of-two scaled units; finite diagnostic means are left unchanged. Products of up to four deviations are formed in the scaled range. A nonzero frame value, accumulated sum, intermediate product or quotient lost during conditioning or reduction is refused, including a series whose within-pixel dynamic range cannot be kept in the scaled representation. Global rescaling alone cannot repair such within-pixel ratios. This is conservative even when a rounded mean would remain finite; it is not a guarantee against all cancellation error. The scaling is undone at the end; a value that would exceed the numeric range, or a nonzero value that would fall below the smallest representable magnitude, raises `mssrscopR_error_numerical`, and very small results keep only subnormal precision. `"mean"` and

"tpm" need one pass over the frames. The others need the mean first and then a second pass. The per-frame maps are stored for it when they fit (diagnostics$path "stored"), else every frame's MSSR is computed again ("recompute"), which doubles the time. Both paths give the same values. With max_memory_bytes = NULL the maps are stored when they take at most 2 GiB, counted with the estimate's 100 percent margin for stored maps; with a budget, they are stored when the estimate fits it, and a call whose recompute path does not fit is refused. mssr_estimate_resources() reports both estimates; they are checked before any frame is read. The input array is read one frame at a time and never copied as a whole. mssr_stack() stores every map: without max_memory_bytes it refuses (with mssrscopR_error_resource) maps that would take more than 2 GiB with that margin.

Each frame must satisfy the checks of mssr(); errors name the frame. Warnings of the frames' mssr() calls (negative values, skipped refinement steps) are collected and given once per class, with the number of frames concerned. A call-level mssrscopR_warning_unqualified_order is given for orders above 3.

Value

mssr_temporal() returns an mssr_result (result schema 1.1.0) with role "temporal_<statistic>", values on the amplified grid and the same transform as mssr(). parameters holds the per-frame MSSR settings plus statistic, frames (the number of frames used), first_frame (the first frame's index in the original input), sign ("not_applicable" for statistics other than SOFI) and definition. diagnostics counts degenerate_frames, warning_frames and order_steps_skipped over the frames, and gives zero_fraction, negative_fraction, mesh_index, mesh_region, trend and the computation path. mssr_stack() returns an object of class mssr_stack: values (an array of the per-frame maps), transform, parameters (with frames and first_frame), engine, diagnostics (a data frame with one row per frame, including the raw frame's mean input_mean) and input_mean (the raw temporal mean on the input grid).

diagnostics$mesh_index measures the sub-pixel grid on signal-free input pixels only, because where structure is present the index mixes structure and grid. Each such input pixel contributes its whole amp x amp block of output pixels; the index is the range of the mean over the amp x amp sub-pixel phases, divided by the absolute mean of these blocks. For the SOFI statistics it is computed on the absolute values, because signed cumulants average to about zero where there is no signal. It is formed only on the default grid = "extent", where every input pixel holds each phase exactly once; on the endpoint grid the phases drift across the field.

mesh_region names the pixels used: "below_median", the pixels whose raw temporal mean over the selected frames lies strictly below its median; "below_median_stack", the same rule for a frame subset of an mssr_stack, whose stored raw mean covers all of its frames, so the index can differ from that of the raw frames; "background", a supplied mask; or "not_computed" for amp = 1 and the endpoint grid. mesh_pixels counts the input pixels used. mesh_index is NULL when no index is computed, when no pixel qualifies (for example when more than half of the pixels share the lowest raw mean, so that none lies below the median; supply background then), or when the mean of the blocks is 0.

diagnostics$trend is the mean of the raw frames over the whole field in the last tenth of the selected frames divided by that in the first tenth (at least one frame each); NULL when the first is 0. It is reported for every statistic and corrects nothing.

References

Torres-Garcia E, Pinto-Camara R, Linares A, et al. Extending resolution within a single imaging frame. *Nature Communications* 2022;13:7452. doi:10.1038/s41467022346939

Dertinger T, Colyer R, Iyer G, Weiss S, Enderlein J. Fast, background-free, 3D super-resolution optical fluctuation imaging (SOFI). *PNAS* 2009;106:22287-22292. doi:10.1073/pnas.0907866106

See Also

`mssr()`, `mssr_estimate_resources()`, `vignette("temporal")`

Examples

```
# Two blinking emitters, 40 frames, Poisson noise (synthetic).
set.seed(1)
g <- expand.grid(r = 1:16, c = 1:16)
spot <- function(r0, c0) exp(-((g$r - r0)^2 + (g$c - c0)^2) / (2 * (3 / 2.3548)^2))
x <- array(0, c(16, 16, 40))
for (t in 1:40) {
  on <- stats::runif(2) < 0.4
  x[, , t] <- stats::rpois(256, 20 * (on[1] * spot(8, 7) + on[2] * spot(8, 10)) + 2)
}
s <- mssr_stack(x, fwhm = 3)
v <- mssr_temporal(s, "var")
v
m <- mssr_temporal(s, "mean")
all.equal(mssr_temporal(s, "tpm")$values, 40^2 * m$values^2)
```

---

mssr_validate_result *Validate a complete result record*

Description

Checks values, geometry, settings, engine identity and diagnostic consistency. This establishes the stored contract, not image authenticity, numerical recomputation, scientific validation or a calibrated confidence estimate. Coordinate and mask functions retain their narrower matrix/grid validation.

Usage

```
mssr_validate_result(result)
```

Arguments

`result`	A result returned by `mssr()` or `mssr_temporal()`.

Details

Results without a schema field are treated as legacy records; they must still have the original six documented fields. Their engine identity is retained. Schema 1.0.0 adds explicit unknown input calibration and a separate evidence status. Schema 1.1.0 adds temporal records (`mssr_temporal()`), whose role, statistic, sign, frame counts and diagnostics are checked as well; single-frame results keep schema 1.0.0. Unknown schema versions and unexpected fields are refused.

Value

The validated result invisibly, unchanged.

Examples

```
r <- mssr(matrix(1:20, 4, 5), fwhm = 2)
mssr_validate_result(r)
```

Index

graphics::image(), *3*

inspect_mssr_result, 2

mssr, 3
mssr(), *6–12*, *14–16*, *18, 19*
mssr_app, 6
mssr_batch, 8
mssr_coords, 9
mssr_coords(), *6*
mssr_estimate_resources, 10
mssr_estimate_resources(), *4*, *8*, *18, 19*
mssr_mask, 11
mssr_mask(), *3*, *6*
mssr_read_result *(*mssr_result_io*)*, 12
mssr_read_result(), *14*
mssr_result_io, 12
mssr_selftest, 13
mssr_selftest(), *6*
mssr_settings, 14
mssr_stack *(*mssr_temporal*)*, 15
mssr_temporal, 15
mssr_temporal(), *6*, *10*, *14*, *19, 20*
mssr_validate_result, 19
mssr_write_result *(*mssr_result_io*)*, 12

plot.mssr_result *(*inspect_mssr_result*)*, 2
print.summary.mssr_result *(*inspect_mssr_result*)*, 2

shiny::runApp(), *7*
summary.mssr_result *(*inspect_mssr_result*)*, 2