Package: sTiles (via r-universe)

November 16, 2026

Type Package

Title Tile-Based Sparse Cholesky Factorization and Selected Inverse

Version 2026.11.16

Description Interface to the 'sTiles' framework for tile-based sparse Cholesky factorization: log-determinants, selected inverse (marginal variances) and triangular solves, with symbolic reuse so that repeated factorization of matrices sharing one sparsity pattern pays the ordering cost only once, as in a hyperparameter sweep. The compiled glue in this package resolves its symbols at run time against the 'sTiles' solver library ('libstiles'), which is a separate component distributed under its own terms and is not part of this package. Install it once with 'sTiles_install_library()', or point the package at a copy you already have with the 'STILES_LIB' environment variable.

License MIT + file LICENSE

URL https://esmail-abdulfattah.github.io/sTiles/

BugReports https://github.com/esmail-abdulfattah/sTiles/issues

Imports methods, Matrix, tools, utils

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

SystemRequirements the 'sTiles' solver library (libstiles.so on Linux, libstiles.dylib on macOS, libstiles.dll on Windows), obtained with sTiles_install_library() or supplied by the user

Encoding UTF-8

Roxygen list(load = ``source'', roclets = c(``rd''))

Config/roxygen2/version 8.0.0

Repository https://esmail-abdulfattah.r-universe.dev

Date/Publication 2026-11-16 11:01:15 UTC

RemoteUrl https://github.com/esmail-abdulfattah/sTiles

RemoteRef HEAD

RemoteSha edf0bf2105c29ed326e71bfa4209e388837f1326

RemoteSubdir R/sTiles

Contents

print.sTiles . . . 2
print.sTiles_summary . . . 3
sTiles . . . 4
sTiles_analyze . . . 5
sTiles_available . . . 6
sTiles_clean_cache . . . 7
sTiles_close . . . 7
sTiles_factorize . . . 8
sTiles_install_library . . . 9
sTiles_library_path . . . 10
sTiles_logdet . . . 11
sTiles_ordering . . . 11
sTiles_selinv . . . 13
sTiles_selinv_diag . . . 13
sTiles_selinv_elm . . . 14
sTiles_selinv_matrix . . . 15
sTiles_selinv_row . . . 16
sTiles_solve . . . 17
sTiles_summary . . . 18
sTiles_update . . . 19
sTiles_version . . . 20

Index **21**

`print.sTiles` *Print a sTiles factorization object*

Description

Print a sTiles factorization object

Usage

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

Arguments

`x`	An "sTiles" object.
`...`	Ignored, present for consistency with [print()].

Value

'x', invisibly.

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  print(sTiles(Q))
}
```

`print.sTiles_summary` *Print a sTiles summary*

Description

Print a sTiles summary

Usage

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

Arguments

`x`	An object of class "sTiles_summary" from [sTiles_summary()].
`...`	Ignored, present for consistency with [print()].

Value

'x', invisibly.

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  print(sTiles_summary(sTiles(Q)))
}
```

`sTiles` *Factorize a symmetric positive-definite matrix*

Description

One shot: runs the preprocessing ([sTiles_analyze()]) and then the numeric factorization ([sTiles_factorize()]). To time the two phases apart, or to reuse one preprocessing across many sets of values, call those two yourself.

Usage

```
sTiles(
  Q,
  cores = 1L,
  mode = "auto",
  tile_size = 40L,
  inverse = FALSE,
  log_level = -1L
)
```

Arguments

`Q`	A symmetric positive-definite matrix, either a ‘Matrix::sparseMatrix‘ or a base matrix. Only the lower triangle is read.
`cores`	Worker threads (default 1).
`mode`	Tile regime: "auto" (default), "dense", "semisparse" or "sparse".
`tile_size`	Tile size, or -1 to let the solver choose (default 40).
`inverse`	Reserve selected-inverse storage, required by [sTiles_selinv()] and the ‘sTiles_selinv_*()‘ queries (default ‘FALSE‘).
`log_level`	Solver verbosity: -1 silent (default), 0 timing, 1 info, 2 debug, 3 trace.

Value

An object of class "sTiles" wrapping a live factorization.

See Also

[sTiles_logdet()], [sTiles_solve()], [sTiles_selinv()], [sTiles_summary()], [sTiles_close()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles(Q)
```

```
  sTiles_logdet(s)
  sTiles_solve(s, rep(1, 50))
  sTiles_close(s)
}
```

`sTiles_analyze` *Preprocess a matrix: ordering bake-off and tile layout only*

Description

The symbolic phase. It depends only on the sparsity pattern, not on the numeric values, and performs no Cholesky. Follow it with [sTiles_factorize()] for the numeric factorization. Timing the two apart separates the preprocessing cost, paid once per pattern, from the numeric cost, paid once per set of values.

Usage

```
sTiles_analyze(
  Q,
  cores = 1L,
  mode = "auto",
  tile_size = 40L,
  inverse = FALSE,
  log_level = -1L
)
```

Arguments

`Q`	A symmetric positive-definite matrix, either a ‘Matrix::sparseMatrix‘ or a base matrix. Only the lower triangle is read.
`cores`	Worker threads (default 1).
`mode`	Tile regime: "auto" (default), "dense", "semisparse" or "sparse".
`tile_size`	Tile size, or -1 to let the solver choose (default 40).
`inverse`	Reserve selected-inverse storage, required by [sTiles_selinv()] and the ‘sTiles_selinv_*()‘ queries (default ‘FALSE‘).
`log_level`	Solver verbosity: -1 silent (default), 0 timing, 1 info, 2 debug, 3 trace.

Value

An object of class "sTiles", analyzed but not yet factorized.

See Also

[sTiles_factorize()], [sTiles_update()], [sTiles()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles_analyze(Q)
  sTiles_factorize(s)
  sTiles_logdet(s)
  sTiles_close(s)
}
```

`sTiles_available` *Is the sTiles solver library available?*

Description

Reports whether a solver library can be found on this machine, checking the 'STILES_LIB', 'STILES_LIB_DIR' and 'STILES_BINARIES_DIR' environment variables, a copy bundled in the installed package, a development checkout, and the cache filled by [sTiles_install_library()]. It looks at the file system only: it never contacts the network and never loads anything, which makes it the right guard for examples and tests that have to be skipped where no solver is installed.

Usage

```
sTiles_available()
```

Value

'TRUE' when a solver library was found, 'FALSE' otherwise.

See Also

[sTiles_install_library()]

Examples

```
sTiles_available()
```

sTiles_clean_cache *Delete every cached sTiles solver library*

Description

Removes the solver copies that [sTiles_install_library()] placed in the package cache. The blunt instrument for a cache believed to be broken or stale, and the way to make room before installing a different release: call [sTiles_install_library()] afterwards to fetch a fresh one. Takes effect immediately unless this session has already built a matrix with the old solver, in which case the files go but a replacement can only load after restarting R (a loaded solver with live handles cannot be swapped underneath them).

Usage

```
sTiles_clean_cache()
```

Value

The number of cached solvers removed, invisibly.

See Also

[sTiles_install_library()], [sTiles_available()]

Examples

```
## Not run:
sTiles_clean_cache()

## End(Not run)
```

sTiles_close *Free a factorization now*

Description

Releases the solver's memory immediately. Optional: an object that goes out of scope is freed at the next garbage collection anyway. Worth calling in a loop over large matrices, where waiting for the collector means holding several factorizations at once.

Usage

```
sTiles_close(x)
```

Arguments

x	An "sTiles" object.

Value

'NULL', invisibly.

See Also

[sTiles()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles(Q)
  sTiles_close(s)
}
```

`sTiles_factorize` *Numeric Cholesky factorization, reusing the preprocessing*

Description

Runs the numeric phase on an object that [sTiles_analyze()] has already prepared, so the ordering and tile layout are not recomputed.

Usage

```
sTiles_factorize(x, Q = NULL)
```

Arguments

`x`	An "sTiles" object from [sTiles_analyze()] or [sTiles()].
`Q`	Optional: a matrix with the SAME sparsity pattern, whose values to factor. When omitted, the values captured at analyze time are used.

Value

The "sTiles" object, now factorized, invisibly.

See Also

[sTiles_analyze()], [sTiles_update()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles_analyze(Q)
  sTiles_factorize(s)
  sTiles_logdet(s)
  sTiles_close(s)
}
```

`sTiles_install_library` *Install the sTiles solver library*

Description

Downloads the prebuilt solver library that matches this platform from the sTiles project's releases and stores it in the package cache, under ‘tools::R_user_dir("sTiles", "cache")‘. Every later session finds the cached copy without touching the network, so run it again to pick up a newer release. A note says when the cached copy is older than this package.

Usage

```
sTiles_install_library(tag = NULL, variant = NULL, force = FALSE)
```

Arguments

`tag`	Release tag to install, for example "Version_2026.10.19". Defaults to the latest release.
`variant`	Build variant to prefer, for example "armv82-armpl" or "v3-mkl", or "none" for the portable default. Defaults to the best fit for this CPU, falling back to the portable build when the release does not carry the preferred one.
`force`	Re-download even when a matching solver is already cached.

Details

The solver is a separate component under its own license terms (see the NOTICE file in this package) and is deliberately not bundled. Nothing is downloaded unless you call this function or accept the prompt an interactive session shows the first time a solver is needed. To skip the download altogether, point ‘STILES_LIB‘ at a shared object you already have, or ‘STILES_LIB_DIR‘ at the directory holding it. Setting ‘STILES_NO_DOWNLOAD‘ disables the download path entirely.

Value

The path of the installed shared library, invisibly.

Search order

Before anything is downloaded, and on every later call, the package takes the first solver it finds among:

1. ‘STILES_LIB‘, a shared object named outright;
2. ‘STILES_LIB_DIR‘, a directory holding one;
3. ‘STILES_BINARIES_DIR‘, a CI-artifact tree;
4. a ‘binaries/‘ tree above the installed package or the working directory;
5. a copy bundled in the installed package, under ‘solver/‘;
6. ‘lib/‘ in a development checkout above the package;
7. the download cache, newest release first.

See Also

[sTiles_available()], [sTiles_clean_cache()]

Examples

```
## Not run:
sTiles_install_library()
sTiles_install_library(tag = "Version_2026.10.19", variant = "none")

## End(Not run)
```

---

sTiles_library_path *Path of the loaded sTiles solver library*

Description

Loads the solver if this session has not loaded it yet, then reports which file answered. Useful when several builds are installed and you need to know which one is in use.

Usage

```
sTiles_library_path()
```

Value

The absolute path of the loaded shared library, as a string.

See Also

[sTiles_available()], [sTiles_install_library()]

Examples

```
if (sTiles_available()) {
  sTiles_library_path()
}
```

sTiles_logdet *Log-determinant of the factorized matrix*

Description

Returns ‘log(det(Q))‘, computed from the factor as ‘2 * sum(log(diag(L)))‘.

Usage

```
sTiles_logdet(x)
```

Arguments

x	A factorized "sTiles" object.

Value

The log-determinant, a single number.

See Also

[sTiles()], [sTiles_summary()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles(Q)
  sTiles_logdet(s)
  sTiles_close(s)
}
```

sTiles_ordering *Fill-reducing ordering of a symmetric sparse matrix, on its own*

Description

Runs the same candidate bake-off the solver runs before a factorization and returns the winning permutation, without building a solver handle. Which orderings are raced is decided from the size and density of the graph, as the solver decides it; there is no argument to pick one. Only the sparsity pattern of ‘Q‘ matters; its values are ignored.

Usage

```
sTiles_ordering(Q, tile_size = 40L, cores = 1L)
```

Arguments

`Q`	A symmetric matrix, either a ‘Matrix::sparseMatrix‘ or a base matrix. Only the lower triangle is read.
`tile_size`	‘> 0‘ ranks the candidates by the number of occupied ‘tile_size‘ x ‘tile_size‘ blocks in the factor, the tiled solver’s cost model (default 40, as in [sTiles()]). ‘0‘ ranks them by exact nnz(L), which needs a solver built with that ranking; the released builds do not have it and report an error.
`cores`	Candidates are evaluated concurrently, one thread each.

Details

The preprocessing is the solver’s own: nodes whose degree exceeds five times the mean are pulled out of the graph, the rest is ordered, and they are appended at the end, so one dense row cannot fill the whole factor. It is applied exactly as [sTiles_analyze()] applies it, with no switch, so the permutation returned is the one a factorization of ‘Q‘ would use.

Value

A list of class ‘"sTiles_ordering"‘, all indices 1-based:

‘perm‘ ‘perm[old] = new‘, the position of each original node.

‘iperm‘ ‘iperm[new] = old‘; ‘Q[iperm, iperm]‘ is the reordered matrix.

‘winner‘ Name of the ordering that won, e.g. ‘"camd"‘.

‘nblocks‘, ‘range‘, ‘parent‘ The separator tree when the winner is a nested dissection (SCOTCH, METIS): block ‘k‘ spans permuted positions ‘range[k]:(range[k + 1] - 1)‘ and ‘parent[k]‘ is its parent block, ‘0‘ at a root. Minimum-degree orderings and RCM have no such structure and return ‘nblocks = 0‘ with empty ‘range‘ and ‘parent‘.

See Also

[sTiles_analyze()], which runs the same bake-off as the first step of a factorization.

Examples

```
if (sTiles_available()) {
  Q <- Matrix::bandSparse(200, k = c(0, 1),
                          diagonals = list(rep(4, 200), rep(-1, 199)),
                          symmetric = TRUE)
  o <- sTiles_ordering(Q)
  o$winner
  head(o$perm)
  Qp <- Q[o$iperm, o$iperm]     # the matrix as the solver would factor it
}
```

sTiles_selinv *Compute the selected inverse, reusing the current factorization*

Description

Computes Z, the inverse of Q restricted to the pattern of the Cholesky factor, ‘pattern(L + L^T)‘. The object must have been built with ‘inverse = TRUE‘. The call is idempotent, and the computation otherwise happens lazily on the first ‘sTiles_selinv_*()‘ query. Call it explicitly to time the selected inverse on its own, and call it again after each [sTiles_factorize()] to refresh Z for the new values.

Usage

```
sTiles_selinv(x)
```

Arguments

x	A factorized "sTiles" object built with ‘inverse = TRUE‘.

Value

The "sTiles" object, invisibly.

See Also

[sTiles_selinv_matrix()], [sTiles_selinv_diag()], [sTiles_selinv_elm()], [sTiles_selinv_row()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles(Q, inverse = TRUE)
  sTiles_selinv(s)
  head(sTiles_selinv_diag(s))
  sTiles_close(s)
}
```

sTiles_selinv_diag *Diagonal of the selected inverse: the marginal variances*

Description

Returns ‘diag(solve(Q))‘, in the original ordering. Triggers the selected-inverse computation on first use.

Usage

```
sTiles_selinv_diag(x)
```

Arguments

x	A factorized "sTiles" object built with ‘inverse = TRUE‘.

Value

A numeric vector of length ‘n‘, the diagonal of the inverse.

See Also

[sTiles_selinv()], [sTiles_selinv_elm()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles(Q, inverse = TRUE)
  head(sTiles_selinv_diag(s))
  sTiles_close(s)
}
```

---

sTiles_selinv_elm *One entry of the selected inverse*

Description

Returns the selected inverse at position ‘(i, j)‘ when that position lies in the factor pattern, ‘pattern(L + L^T)‘, and exactly 0 outside it. Both triangles are accepted, since Z is symmetric. Triggers the selected-inverse computation on first use.

Usage

```
sTiles_selinv_elm(x, i, j)
```

Arguments

x	A factorized "sTiles" object built with ‘inverse = TRUE‘.
i, j	Row and column, 1-based, in the original ordering.

Value

The entry as a single number, 0 outside the factor pattern.

See Also

[sTiles_selinv()], [sTiles_selinv_row()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles(Q, inverse = TRUE)
  sTiles_selinv_elm(s, 1, 2)
  sTiles_close(s)
}
```

---

sTiles_selinv_matrix *Selected inverse on the pattern of Q, as a sparse matrix*

Description

Returns the entries of ‘solve(Q)‘ at the non-zero positions of ‘Q‘, as a symmetric sparse matrix in the original ordering. Positions outside the pattern of ‘Q‘ are not returned: they read as 0 here, which is not the value of the true inverse there. Triggers the selected-inverse computation on first use, and reflects the latest [sTiles_factorize()].

Usage

```
sTiles_selinv_matrix(x)
```

Arguments

x	A factorized "sTiles" object built with ‘inverse = TRUE‘.

Value

A symmetric [Matrix::dsCMatrix-class] with the pattern of ‘Q‘.

See Also

[sTiles_selinv()], [sTiles_selinv_diag()], [sTiles_selinv_elm()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles(Q, inverse = TRUE)
  Q.inv <- sTiles_selinv_matrix(s)
  Q.inv[1:3, 1:3]
```

```
  sTiles_close(s)
}
```

`sTiles_selinv_row` *Several entries from one row of the selected inverse*

Description

Returns the selected inverse at ‘(node, k)‘ for each ‘k‘ in ‘neighbors‘, the access pattern a graph model wants. Entries outside the factor pattern come back as 0.

Usage

```
sTiles_selinv_row(x, node, neighbors)
```

Arguments

`x`	A factorized "sTiles" object built with ‘inverse = TRUE‘.
`node`	Row index, 1-based, in the original ordering.
`neighbors`	Integer vector of column indices, 1-based.

Value

A numeric vector, one value per entry of ‘neighbors‘.

See Also

[sTiles_selinv()], [sTiles_selinv_elm()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles(Q, inverse = TRUE)
  sTiles_selinv_row(s, 5, c(4, 5, 6))
  sTiles_close(s)
}
```

`sTiles_solve` *Solve a linear system with the factorization*

Description

Solves ‘Q x = b‘, or one half of it, for one or several right-hand sides.

Usage

```
sTiles_solve(x, b, system = c("A", "L", "Lt"))
```

Arguments

`x`	A factorized "sTiles" object.
`b`	Right-hand side, in the original ordering of ‘Q‘: a length-‘n‘ vector, or a matrix with ‘n‘ rows and one column per right-hand side.
`system`	Which system to solve: "A" for ‘Q x = b‘ (default), "L" for ‘C y = b‘, "Lt" for ‘t(C) x = b‘, where ‘Q = C t(C)‘. See the Ordering section.

Value

The solution in the original ordering: a vector when ‘b‘ is a vector, otherwise a matrix with the dimensions of ‘b‘.

Ordering

The solver factors a reordered matrix, ‘Q[p, p] = L t(L)‘, with a fill-reducing permutation ‘p‘ of its own choosing. The solves undo it internally: ‘b‘ and the result are always in the original ordering of ‘Q‘, for every ‘system‘, so nothing has to be permuted by hand.

‘"L"‘ and ‘"Lt"‘ therefore use ‘C‘, which is ‘L‘ with its rows and columns put back in the original ordering. It satisfies ‘Q = C t(C)‘ but is not triangular, and it differs from the factor another library would give. So:

- ‘sTiles_solve(s, sTiles_solve(s, b, "L"), "Lt")‘ equals ‘sTiles_solve(s, b)‘;
- with ‘y <- sTiles_solve(s, b, "L")‘, ‘sum(y^2)‘ is ‘t(b)
- ‘sTiles_solve(s, rnorm(n), "Lt")‘ is a draw from ‘N(0, solve(Q))‘.

See Also

[sTiles()], [sTiles_logdet()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles(Q)
  x <- sTiles_solve(s, rep(1, 50))
  max(abs(as.vector(Q %*% x) - 1))

  ## several right-hand sides, one per column
  X <- sTiles_solve(s, cbind(rep(1, 50), seq_len(50)))
  dim(X)

  ## a draw from N(0, solve(Q))
  draw <- sTiles_solve(s, rnorm(50), system = "Lt")
  sTiles_close(s)
}
```

`sTiles_summary` *Structured summary of a factorization*

Description

Collects the dimensions, the fill, the tile mode, the phase the object is in, and the timings the solver measured. The result is an ordinary list, so single numbers are easy to pull out (‘sTiles_summary(s)$chol_time‘), and it prints as a short report.

Usage

```
sTiles_summary(x)
```

Arguments

x	An "sTiles" object.

Value

An object of class "sTiles_summary": a list with elements ‘n‘, ‘nnz‘, ‘nnz_factor‘, ‘mode‘, ‘cores‘, ‘inverse‘, ‘factored‘, ‘analyze_time‘, ‘chol_time‘, ‘selinv_time‘, ‘version‘ and ‘library‘.

See Also

[sTiles()], [sTiles_version()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles(Q)
  sTiles_summary(s)$nnz_factor
  sTiles_close(s)
}
```

`sTiles_update` *New values, same sparsity pattern: refactorize without reanalyzing*

Description

The ordering and tile layout depend only on WHERE the non-zeros are, so an object built by [sTiles_analyze()] can absorb any number of value updates and pay only the numeric cost each time. This is the loop an iterative method wants.

Usage

```
sTiles_update(x, Q)
```

Arguments

`x`	An "sTiles" object from [sTiles_analyze()].
`Q`	A matrix with the SAME sparsity pattern, whose values to factor.

Value

The "sTiles" object, factorized with the new values, invisibly.

See Also

[sTiles_analyze()], [sTiles_factorize()]

Examples

```
if (sTiles_available()) {
Q <- Matrix::bandSparse(50, k = c(0, 1),
                        diagonals = list(rep(4, 50), rep(-1, 49)),
                        symmetric = TRUE)
  s <- sTiles_analyze(Q)
  sTiles_factorize(s)
  d1 <- sTiles_logdet(s)
  ## same pattern, different values: no new ordering
  sTiles_update(s, Q * 2)
  d2 <- sTiles_logdet(s)
```

```
  c(d1, d2)
  sTiles_close(s)
}
```

`sTiles_version` *Version of the sTiles solver library*

Description

Version of the sTiles solver library

Usage

```
sTiles_version()
```

Value

The solver's version string.

See Also

[sTiles_summary()]

Examples

```
if (sTiles_available()) {
  sTiles_version()
}
```

Index

print.sTiles, 2
print.sTiles_summary, 3

sTiles, 4
sTiles_analyze, 5
sTiles_available, 6
sTiles_clean_cache, 7
sTiles_close, 7
sTiles_factorize, 8
sTiles_install_library, 9
sTiles_library_path, 10
sTiles_logdet, 11
sTiles_ordering, 11
sTiles_selinv, 13
sTiles_selinv_diag, 13
sTiles_selinv_elm, 14
sTiles_selinv_matrix, 15
sTiles_selinv_row, 16
sTiles_solve, 17
sTiles_summary, 18
sTiles_update, 19
sTiles_version, 20