Package ‘fmesher’

June 14, 2024

Type Package

Title Triangle Meshes and Related Geometry Tools

Version 0.1.6

Description Generate planar and spherical triangle meshes, compute finite element calculations for 1- and 2-dimensional flat and curved manifolds with associated basis function spaces, methods for lines and polygons, and transparent handling of coordinate reference systems and coordinate transformation, including 'sf' and 'sp' geometries. The core 'fmesher' library code was originally part of the 'INLA' package, and implements parts of ``Triangulations and Applications'' by Hjelle and Daehlen (2006) <doi:10.1007/3-540-33261-8>.

Depends R (>= 4.0), methods

Imports dplyr, graphics, grDevices, lifecycle, Matrix, rlang, sf, sp (>= 1.6-1), stats, tibble, utils, withr, Rcpp

Suggests ggplot2, inlabru (>= 2.8.0), knitr, testthat (>= 3.0.0), terra, tidyterra, rgl, rmarkdown, splancs, gsl

URL https://inlabru-org.github.io/fmesher/, https://github.com/inlabru-org/fmesher

BugReports https://github.com/inlabru-org/fmesher/issues

License MPL-2.0

Copyright 2010-2023 Finn Lindgren, except src/predicates.cc by Jonathan Richard Shewchuk, 1996

NeedsCompilation yes

RoxygenNote 7.3.1

Encoding UTF-8

Config/testthat/edition 3

Config/testthat/parallel true

SystemRequirements C++17

LinkingTo Rcpp

VignetteBuilder knitr

BuildVignettes true

Collate 'RcppExports.R' 'deprecated.R' 'bary.R' 'bbox.R' 'print.R' 'crs.R' 'data-fmexample.R' 'diameter.R' 'evaluator.R' 'fem.R' 'fm.R' 'fmesher-package.R' 'fmesher.R' 'ggplot.R' 'integration.R' 'lattice_2d.R' 'list.R' 'local.R' 'manifold.R' 'mapping.R' 'matern.R' 'mesh.R' 'mesh_1d.R' 'mesh_2d.R' 'nonconvex_hull.R' 'onload.R' 'plot.R' 'segm.R' 'sf_mesh.R' 'sf_utils.R' 'simplify.R' 'sp_mesh.R' 'split_lines.R' 'tensor.R' 'utils.R'

LazyData true

Author Finn Lindgren [aut, cre, cph] (<https://orcid.org/0000-0002-5833-2011>, Finn Lindgren wrote the main code),
Seaton Andy [ctb] (Andy Seaton constributed features to the sf support),
Suen Man Ho [ctb] (Man Ho Suen contributed features and code structure design for the integration methods),
Fabian E. Bachl [ctb] (Fabian Bachl co-developed precursors of fm_pixels and fm_split_lines in inlabru)

Maintainer Finn Lindgren <finn.lindgren@gmail.com>

Repository CRAN

Date/Publication 2024-06-14 17:20:08 UTC

Contents

fmesher-deprecated . . . 3
fmesher-print . . . 6
fmesher_bary . . . 7
fmesher_fem . . . 8
fmesher_globe_points . . . 8
fmesher_rcdt . . . 9
fmesher_split_lines . . . 10
fmesher_subdivide . . . 10
fmexample . . . 11
fm_as_fm . . . 12
fm_as_lattice_2d . . . 14
fm_as_mesh_1d . . . 15
fm_as_mesh_2d . . . 16
fm_as_segm . . . 17
fm_as_sfc . . . 19
fm_as_tensor . . . 21
fm_bary . . . 22
fm_basis . . . 23
fm_bbox . . . 24
fm_block . . . 27
fm_centroids . . . 30
fm_contains . . . 31
fm_CRS . . . 32

fm_crs . . . 35
fm_crs<- . . . 39
fm_crs_is_identical . . . 41
fm_crs_is_null . . . 42
fm_crs_wkt . . . 43
fm_detect_manifold . . . 46
fm_diameter . . . 47
fm_dof . . . 49
fm_evaluate . . . 49
fm_fem . . . 52
fm_gmrf . . . 53
fm_int . . . 55
fm_is_within . . . 57
fm_lattice_2d . . . 58
fm_list . . . 60
fm_manifold . . . 61
fm_mesh_1d . . . 62
fm_mesh_2d . . . 63
fm_nonconvex_hull . . . 65
fm_nonconvex_hull_inla . . . 67
fm_pixels . . . 69
fm_raw_basis . . . 70
fm_rcdt_2d . . . 72
fm_row_kron . . . 74
fm_segm . . . 75
fm_segm_list . . . 77
fm_simplify . . . 78
fm_split_lines . . . 79
fm_tensor . . . 80
fm_transform . . . 81
fm_vertices . . . 83
geom_fm . . . 84
plot.fm_mesh_2d . . . 86
plot.fm_segm . . . 88
plot_globeproj . . . 90
plot_rgl . . . 91

Index **94**

`fmesher-deprecated` *Deprecated functions in fmesher*

Description

These functions still attempt to do their job, but will be removed in a future version.

Usage

```
fm_spTransform(x, ...)

## Default S3 method:
fm_spTransform(x, crs0 = NULL, crs1 = NULL, passthrough = FALSE, ...)

## S3 method for class 'SpatialPoints'
fm_spTransform(x, CRSobj, passthrough = FALSE, ...)

## S3 method for class 'SpatialPointsDataFrame'
fm_spTransform(x, CRSobj, passthrough = FALSE, ...)

## S3 method for class 'inla.mesh.lattice'
fm_spTransform(x, CRSobj, passthrough = FALSE, ...)

## S3 method for class 'inla.mesh.segment'
fm_spTransform(x, CRSobj, passthrough = FALSE, ...)

## S3 method for class 'inla.mesh'
fm_spTransform(x, CRSobj, passthrough = FALSE, ...)

fm_has_PROJ6()

fm_not_for_PROJ6(fun = NULL)

fm_not_for_PROJ4(fun = NULL)

fm_fallback_PROJ6(fun = NULL)

fm_requires_PROJ6(fun = NULL)

fm_as_sp_crs(x, ...)

fm_sp_get_crs(x)

fm_as_inla_mesh_segment(...)

fm_as_inla_mesh(...)

fm_sp2segment(...)
```

Arguments

`x`	A `sp::Spatial` object
`...`	Potential additional arguments
`crs0`	The source sp::CRS or inla.CRS object
`crs1`	The target sp::CRS or inla.CRS object

passthrough	Default is FALSE. Setting to TRUE allows objects with no CRS information to be passed through without transformation.
CRSobj	The target sp::CRS or inla.CRS object
fun	The name of the function that requires PROJ6. Default: NULL, which uses the name of the calling function.

Details

This function is a convenience method to workaround PROJ4/PROJ6 differences, and the lack of a crs extraction method for Spatial objects. For newer code, use `fm_crs()` instead, that returns `crs` objects, and use fm_CRS() to extract/construct/convert to old style `sp::CRS` objects.

Value

A `CRS` object, or NULL if no valid CRS identified

An `fm_segm` object

An `fm_mesh_2d` object

Functions

- `fm_spTransform()`: **[Deprecated]** (See `fm_transform()` instead) Handle transformation of various inla objects according to coordinate reference systems of `sp::CRS` or `INLA::inla.CRS` class.
- `fm_spTransform(default)`: The default method handles low level transformation of raw coordinates.
- `fm_has_PROJ6()`: Detect whether PROJ6 is available
- `fm_not_for_PROJ6()`: `fm_not_for_PROJ6` is called to warn about using old PROJ4 features even though PROJ6 is available
- `fm_not_for_PROJ4()`: `fm_not_for_PROJ4` is called to give an error when calling methods that are only available for PROJ6
- `fm_fallback_PROJ6()`: Called to warn about falling back to using old PROJ4 methods when a PROJ6 method hasn't been implemented
- `fm_requires_PROJ6()`: Called to give an error when PROJ6 is required but not available
- `fm_as_sp_crs()`: Wrapper for `fm_CRS()` `sp::Spatial` and `sp::CRS` objects.
- `fm_sp_get_crs()`: Wrapper for CRS(projargs) (PROJ4) and CRS(wkt) for `sp::Spatial` objects.
- `fm_as_inla_mesh_segment()`: Conversion to inla.mesh.segment **[Deprecated]** in favour of `fm_as_segm()`.
- `fm_as_inla_mesh()`: Conversion to inla.mesh. **[Deprecated]** in favour of `fm_as_mesh_2d()`.
- `fm_sp2segment()`: **[Deprecated]** in favour of `fm_as_segm()`

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

fm_transform()

Examples

```
if (fm_safe_sp()) {
  s <- sp::SpatialPoints(matrix(1:6, 3, 2), proj4string = fm_CRS("sphere"))
  fm_CRS(s)
}
```

---

fmesher-print *Print objects*

Description

Print objects

Usage

```
## S3 method for class 'fm_segm'
print(x, ..., digits = NULL, verbose = TRUE, newline = TRUE)

## S3 method for class 'fm_segm_list'
print(x, ..., digits = NULL, verbose = FALSE, newline = TRUE)

## S3 method for class 'fm_mesh_2d'
print(x, ..., digits = NULL, verbose = FALSE)

## S3 method for class 'fm_mesh_1d'
print(x, ..., digits = NULL, verbose = FALSE)

## S3 method for class 'fm_bbox'
print(x, ..., digits = NULL, verbose = TRUE, newline = TRUE)

## S3 method for class 'fm_tensor'
print(x, ..., digits = NULL, verbose = FALSE)

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

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

Arguments

x	an object used to select a method.
...	further arguments passed to or from other methods.

digits	a positive integer indicating how many significant digits are to be used for numeric and complex x. The default, NULL, uses `getOption("digits")`.
verbose	logical
newline	logical; if `TRUE` (default), end the printing with `\n`

Value

The input object `x`

Examples

```
fm_bbox(matrix(1:6, 3, 2))
print(fm_bbox(matrix(1:6, 3, 2)), verbose = FALSE)

print(fmexample$mesh)
print(fmexample$boundary_fm)

print(fm_mesh_1d(c(1, 2, 3, 5, 7), degree = 2))
```

`fmesher_bary` *Barycentric coordinate computation*

Description

Locate points and compute triangular barycentric coordinates

Usage

```
fmesher_bary(mesh_loc, mesh_tv, loc, options)
```

Arguments

mesh_loc	numeric matrix; mesh vertex coordinates
mesh_tv	3-column integer matrix with 0-based vertex indices for each triangle
loc	numeric matrix; coordinates of points to locate in the mesh
options	list of triangulation options

Value

A list with vector `t` and matrix `bary`

Examples

```
m <- fmesher_rcdt(list(cet_margin = 1), matrix(0, 1, 2))
b <- fmesher_bary(m$s,
                  m$tv,
                  matrix(c(0.5, 0.5), 1, 2),
                  list())
```

fmesher_fem *Finite element matrix computation*

Description

Construct finite element structure matrices

Usage

```
fmesher_fem(mesh_loc, mesh_tv, fem_order_max, aniso, options)
```

Arguments

mesh_loc	numeric matrix; mesh vertex coordinates
mesh_tv	3-column integer matrix with 0-based vertex indices for each triangle
fem_order_max	integer; the highest operator order to compute
aniso	If non-NULL, a `list(gamma, v)`. Calculates anisotropic structure matrices (in addition to the regular) for γ and v for an anisotropic operator $\nabla \cdot H \nabla$, where $H = \gamma I + vv^\top$. Currently (2023-08-05) the fields need to be given per vertex.
options	list of triangulation options (`sphere_tolerance`)

Value

A list of matrices

Examples

```
m <- fmesher_rcdt(list(cet_margin = 1), matrix(0, 1, 2))
b <- fmesher_fem(m$s, m$tv, fem_order_max = 2, aniso = NULL, options = list())
```

fmesher_globe_points *Globe points*

Description

Create points on a globe

Usage

```
fmesher_globe_points(globe)
```

Arguments

globe	integer; the number of edge subdivision segments, 1 or higher.

Value

A matrix of points on a unit radius globe

Examples

```
fmesher_globe_points(1)
```

---

fmesher_rcdt *Refined Constrained Delaunay Triangulation*

Description

(...)

Usage

```
fmesher_rcdt(
  options,
  loc,
  tv = NULL,
  boundary = NULL,
  interior = NULL,
  boundary_grp = NULL,
  interior_grp = NULL
)
```

Arguments

options	list of triangulation options
loc	numeric matrix; initial points to include
tv	3-column integer matrix with 0-based vertex indices for each triangle
boundary	2-column integer matrix with 0-based vertex indices for each boundary edge constraint
interior	2-column integer matrix with 0-based vertex indices for each interior edge constraint
boundary_grp	integer vector with group labels
interior_grp	integer vector with group labels

Value

A list of information objects for a generated triangulation

Examples

```
m <- fmesher_rcdt(list(cet_margin = 1), matrix(0, 1, 2))
```

fmesher_split_lines *Split lines at triangle edges*

Description

Split a sequence of line segments at triangle edges

Usage

```
fmesher_split_lines(mesh_loc, mesh_tv, loc, idx, options)
```

Arguments

mesh_loc	numeric matrix; mesh vertex coordinates
mesh_tv	3-column integer matrix with 0-based vertex indices for each triangle
loc	numeric coordinate matrix
idx	2-column integer matrix
options	list of triangulation options (sphere_tolerance)

Value

A list of line splitting information objects

See Also

fm_split_lines()

Examples

```
mesh <- fm_mesh_2d(
  boundary = fm_segm(rbind(c(0,0), c(1,0), c(1,1), c(0, 1)), is.bnd = TRUE)
)
splitter <- fm_segm(rbind(c(0.8, 0.2), c(0.2, 0.8)))
segm_split <- fm_split_lines(mesh, splitter)
```

fmesher_subdivide *Split triangles*

Description

Subdivide a mesh with congruent subtriangles

Usage

```
fmesher_subdivide(
  mesh_loc,
  mesh_tv,
  mesh_boundary,
  mesh_interior,
  subdivisions,
  options
)
```

Arguments

mesh_loc	numeric matrix; mesh vertex coordinates
mesh_tv	3-column integer matrix with 0-based vertex indices for each triangle
mesh_boundary	2-column integer matrix with 0-based vertex indices for boundary constraints
mesh_interior	2-column integer matrix with 0-based vertex indices for interior constraints
subdivisions	integer; number of new points along each edge.
options	list of triangulation options (sphere_tolerance)

Value

A list of line splitting information objects

See Also

fm_split_lines()

Examples

```
mesh <- fm_mesh_2d(
  boundary = fm_segm(rbind(c(0,0), c(1,0), c(1,1), c(0, 1)), is.bnd = TRUE)
)
splitter <- fm_segm(rbind(c(0.8, 0.2), c(0.2, 0.8)))
segm_split <- fm_split_lines(mesh, splitter)
```

---

fmexample *Example mesh data*

Description

This is an example data set used for fmesher package examples.

Usage

```
fmexample
```

Format

The data is a list containing these elements:

`loc`: A `matrix` of points.

`loc_sf`: An `sfc` version of `loc`.

`loc_sp`: A `SpatialPoints` version of `loc`.

`boundary_fm`: A `fm_segm_list` of two `fm_segm` objects used in the mesh construction.

`boundary_sf`: An `sfc` list version of `boundary`.

`boundary_sp`: An `SpatialPolygons` list version of `boundary`.

`mesh`: An `fm_mesh_2d()` object.

Source

Generated by `data-raw/fmexample.R`.

Examples

```
if (require(ggplot2, quietly = TRUE)) {
  ggplot() +
    geom_sf(data = fm_as_sfc(fmexample$mesh)) +
    geom_sf(data = fmexample$boundary_sf[[1]], fill = "red", alpha = 0.5)
}
```

---

fm_as_fm *Convert objects to fmesher objects*

Description

Used for conversion from general objects (usually `inla.mesh` and other INLA specific classes) to `fmesher` classes.

Usage

```
fm_as_fm(x, ...)

## S3 method for class 'NULL'
fm_as_fm(x, ...)

## S3 method for class 'fm_mesh_1d'
fm_as_fm(x, ...)

## S3 method for class 'fm_mesh_2d'
fm_as_fm(x, ...)

## S3 method for class 'fm_tensor'
fm_as_fm(x, ...)
```

```
## S3 method for class 'fm_segm'
fm_as_fm(x, ...)

## S3 method for class 'fm_lattice_2d'
fm_as_fm(x, ...)

## S3 method for class 'fm_bbox'
fm_as_fm(x, ...)

## S3 method for class 'crs'
fm_as_fm(x, ...)

## S3 method for class 'CRS'
fm_as_fm(x, ...)

## S3 method for class 'fm_crs'
fm_as_fm(x, ...)

## S3 method for class 'inla.CRS'
fm_as_fm(x, ...)

## S3 method for class 'inla.mesh.1d'
fm_as_fm(x, ...)

## S3 method for class 'inla.mesh'
fm_as_fm(x, ...)

## S3 method for class 'inla.mesh.segment'
fm_as_fm(x, ...)

## S3 method for class 'inla.mesh.lattice'
fm_as_fm(x, ...)
```

Arguments

x	Object to be converted
...	Arguments forwarded to submethods

Value

An object of some `fm_*` class

See Also

Other object creation and conversion: `fm_as_lattice_2d()`, `fm_as_mesh_1d()`, `fm_as_mesh_2d()`, `fm_as_segm()`, `fm_as_sfc()`, `fm_as_tensor()`, `fm_lattice_2d()`, `fm_mesh_1d()`, `fm_mesh_2d()`, `fm_segm()`, `fm_simplify()`, `fm_tensor()`

Examples

```
fm_as_fm(NULL)
```

`fm_as_lattice_2d` *Convert objects to* `fm_lattice_2d`

Description

Convert objects to `fm_lattice_2d`

Usage

```
fm_as_lattice_2d(...)

fm_as_lattice_2d_list(x, ...)

## S3 method for class 'fm_lattice_2d'
fm_as_lattice_2d(x, ...)

## S3 method for class 'inla.mesh.lattice'
fm_as_lattice_2d(x, ...)
```

Arguments

`...`	Arguments passed on to submethods
`x`	Object to be converted

Value

An `fm_lattice_2d` or `fm_lattice_2d_list` object

Functions

- `fm_as_lattice_2d()`: Convert an object to `fm_lattice_2d`.
- `fm_as_lattice_2d_list()`: Convert each element of a list

See Also

Other object creation and conversion: `fm_as_fm()`, `fm_as_mesh_1d()`, `fm_as_mesh_2d()`, `fm_as_segm()`, `fm_as_sfc()`, `fm_as_tensor()`, `fm_lattice_2d()`, `fm_mesh_1d()`, `fm_mesh_2d()`, `fm_segm()`, `fm_simplify()`, `fm_tensor()`

Examples

```
str(fm_as_lattice_2d_list(list(fm_lattice_2d(), fm_lattice_2d())))
```

fm_as_mesh_1d *Convert objects to* fm_segm

Description

Convert objects to fm_segm

Usage

```
fm_as_mesh_1d(x, ...)

fm_as_mesh_1d_list(x, ...)

## S3 method for class 'fm_mesh_1d'
fm_as_mesh_1d(x, ...)

## S3 method for class 'inla.mesh.1d'
fm_as_mesh_1d(x, ...)
```

Arguments

x	Object to be converted
...	Arguments passed on to submethods

Value

An fm_mesh_1d or fm_mesh_1d_list object

Functions

- fm_as_mesh_1d(): Convert an object to fm_mesh_1d.
- fm_as_mesh_1d_list(): Convert each element of a list

See Also

Other object creation and conversion: fm_as_fm(), fm_as_lattice_2d(), fm_as_mesh_2d(), fm_as_segm(), fm_as_sfc(), fm_as_tensor(), fm_lattice_2d(), fm_mesh_1d(), fm_mesh_2d(), fm_segm(), fm_simplify(), fm_tensor()

Examples

```
fm_as_mesh_1d_list(list(fm_mesh_1d(1:4)))
```

fm_as_mesh_2d — *Convert objects to* fm_mesh_2d

Description

Convert objects to fm_mesh_2d

Usage

```
fm_as_mesh_2d(x, ...)

fm_as_mesh_2d_list(x, ...)

## S3 method for class 'fm_mesh_2d'
fm_as_mesh_2d(x, ...)

## S3 method for class 'inla.mesh'
fm_as_mesh_2d(x, ...)

## S3 method for class 'sfg'
fm_as_mesh_2d(x, ...)

## S3 method for class 'sfc_MULTIPOLYGON'
fm_as_mesh_2d(x, ...)

## S3 method for class 'sfc_POLYGON'
fm_as_mesh_2d(x, ...)

## S3 method for class 'sf'
fm_as_mesh_2d(x, ...)
```

Arguments

x	Object to be converted
...	Arguments passed on to submethods

Value

An fm_mesh_2d or fm_mesh_2d_list object

Functions

- fm_as_mesh_2d(): Convert an object to fm_mesh_2d.
- fm_as_mesh_2d_list(): Convert each element of a list

See Also

Other object creation and conversion: fm_as_fm(), fm_as_lattice_2d(), fm_as_mesh_1d(), fm_as_segm(), fm_as_sfc(), fm_as_tensor(), fm_lattice_2d(), fm_mesh_1d(), fm_mesh_2d(), fm_segm(), fm_simplify(), fm_tensor()

Examples

```
fm_as_mesh_2d_list(list(fm_mesh_2d(cbind(2, 1))))
```

---

fm_as_segm *Convert objects to* fm_segm

Description

Convert objects to fm_segm

Usage

```
fm_as_segm(x, ...)

fm_as_segm_list(x, ...)

## S3 method for class 'fm_segm'
fm_as_segm(x, ...)

## S3 method for class 'inla.mesh.segment'
fm_as_segm(x, ...)

## S3 method for class 'sfg'
fm_as_segm(x, ...)

## S3 method for class 'sfc_POINT'
fm_as_segm(x, reverse = FALSE, grp = NULL, is.bnd = TRUE, ...)

## S3 method for class 'sfc_LINESTRING'
fm_as_segm(x, join = TRUE, grp = NULL, reverse = FALSE, ...)

## S3 method for class 'sfc_MULTILINESTRING'
fm_as_segm(x, join = TRUE, grp = NULL, reverse = FALSE, ...)

## S3 method for class 'sfc_POLYGON'
fm_as_segm(x, join = TRUE, grp = NULL, ...)

## S3 method for class 'sfc_MULTIPOLYGON'
fm_as_segm(x, join = TRUE, grp = NULL, ...)

## S3 method for class 'sfc_GEOMETRY'
```

```
fm_as_segm(x, grp = NULL, join = TRUE, ...)

## S3 method for class 'sf'
fm_as_segm(x, ...)

## S3 method for class 'matrix'
fm_as_segm(
  x,
  reverse = FALSE,
  grp = NULL,
  is.bnd = FALSE,
  crs = NULL,
  closed = FALSE,
  ...
)

## S3 method for class 'SpatialPoints'
fm_as_segm(x, reverse = FALSE, grp = NULL, is.bnd = TRUE, closed = FALSE, ...)

## S3 method for class 'SpatialPointsDataFrame'
fm_as_segm(x, ...)

## S3 method for class 'Line'
fm_as_segm(x, reverse = FALSE, grp = NULL, crs = NULL, ...)

## S3 method for class 'Lines'
fm_as_segm(x, join = TRUE, grp = NULL, crs = NULL, ...)

## S3 method for class 'SpatialLines'
fm_as_segm(x, join = TRUE, grp = NULL, ...)

## S3 method for class 'SpatialLinesDataFrame'
fm_as_segm(x, ...)

## S3 method for class 'SpatialPolygons'
fm_as_segm(x, join = TRUE, grp = NULL, ...)

## S3 method for class 'SpatialPolygonsDataFrame'
fm_as_segm(x, ...)

## S3 method for class 'Polygons'
fm_as_segm(x, join = TRUE, crs = NULL, grp = NULL, ...)

## S3 method for class 'Polygon'
fm_as_segm(x, crs = NULL, ...)
```

Arguments

x	Object to be converted.

...	Arguments passed on to submethods
reverse	logical; When TRUE, reverse the order of the input points. Default FALSE
grp	if non-null, should be an integer vector of grouping labels for one for each segment. Default NULL
is.bnd	logical; if TRUE, set the boundary flag for the segments. Default TRUE
join	logical; if TRUE, join input segments with common vertices. Default TRUE
crs	A crs object
closed	logical; whether to treat a point sequence as a closed polygon. Default: FALSE

Value

An fm_segm or fm_segm_list object

Functions

- fm_as_segm(): Convert an object to fm_segm.
- fm_as_segm_list(): Convert each element, making a fm_segm_list object

See Also

c.fm_segm(), c.fm_segm_list(), [.fm_segm_list()

Other object creation and conversion: fm_as_fm(), fm_as_lattice_2d(), fm_as_mesh_1d(), fm_as_mesh_2d(), fm_as_sfc(), fm_as_tensor(), fm_lattice_2d(), fm_mesh_1d(), fm_mesh_2d(), fm_segm(), fm_simplify(), fm_tensor()

Examples

```
fm_as_segm_list(list(
  fm_segm(fmexample$mesh),
  fm_segm(fmexample$mesh, boundary = FALSE)
))

(segm <- fm_segm(fmexample$mesh, boundary = FALSE))
(segm_sfc <- fm_as_sfc(segm))
(fm_as_segm(segm_sfc))
```

---

fm_as_sfc *Conversion methods from mesh related objects to sfc*

Description

Conversion methods from mesh related objects to sfc

Usage

```
fm_as_sfc(x, ...)

## S3 method for class 'inla.mesh'
fm_as_sfc(x, ..., multi = FALSE)

## S3 method for class 'fm_mesh_2d'
fm_as_sfc(x, ..., multi = FALSE)

## S3 method for class 'inla.mesh.segment'
fm_as_sfc(x, ..., multi = FALSE)

## S3 method for class 'fm_segm'
fm_as_sfc(x, ..., multi = FALSE)

## S3 method for class 'sfc'
fm_as_sfc(x, ...)

## S3 method for class 'sf'
fm_as_sfc(x, ...)
```

Arguments

x	An object to be coerced/transformed/converted into another class
...	Arguments passed on to other methods
multi	logical; if TRUE, attempt to a sfc_MULTIPOLYGON, otherwise a set of sfc_POLYGON. Default FALSE

Value

- fm_as_sfc: An sfc_MULTIPOLYGON or sfc_POLYGON object
- fm_as_sfc: An sfc_MULTIPOLYGON or sfc_POLYGON object

Methods (by class)

- fm_as_sfc(inla.mesh): **[Experimental]**
- fm_as_sfc(fm_mesh_2d): **[Experimental]**
- fm_as_sfc(inla.mesh.segment): **[Experimental]**
- fm_as_sfc(fm_segm): **[Experimental]**

See Also

Other object creation and conversion: fm_as_fm(), fm_as_lattice_2d(), fm_as_mesh_1d(), fm_as_mesh_2d(), fm_as_segm(), fm_as_tensor(), fm_lattice_2d(), fm_mesh_1d(), fm_mesh_2d(), fm_segm(), fm_simplify(), fm_tensor()

Examples

```
fm_as_sfc(fmexample$mesh)
fm_as_sfc(fmexample$mesh, multi = TRUE)
```

---

fm_as_tensor *Convert objects to* fm_tensor

Description

Convert objects to fm_tensor

Usage

```
fm_as_tensor(x, ...)

fm_as_tensor_list(x, ...)

## S3 method for class 'fm_tensor'
fm_as_tensor(x, ...)
```

Arguments

x	Object to be converted
...	Arguments passed on to submethods

Value

An fm_tensor object

Functions

- fm_as_tensor(): Convert an object to fm_tensor.
- fm_as_tensor_list(): Convert each element of a list

See Also

Other object creation and conversion: fm_as_fm(), fm_as_lattice_2d(), fm_as_mesh_1d(), fm_as_mesh_2d(), fm_as_segm(), fm_as_sfc(), fm_lattice_2d(), fm_mesh_1d(), fm_mesh_2d(), fm_segm(), fm_simplify(), fm_tensor()

Examples

```
fm_as_tensor_list(list(fm_tensor(list())))
```

fm_bary *Compute barycentric coordinates*

Description

Identify knot intervals or triangles and compute barycentric coordinates

Usage

```
fm_bary(mesh, loc, ...)

## S3 method for class 'fm_mesh_1d'
fm_bary(mesh, loc, method = c("linear", "nearest"), restricted = FALSE, ...)

## S3 method for class 'fm_mesh_2d'
fm_bary(mesh, loc, crs = NULL, ...)

## S3 method for class 'inla.mesh'
fm_bary(mesh, ...)

## S3 method for class 'inla.mesh.1d'
fm_bary(mesh, ...)
```

Arguments

`mesh`	`fm_mesh_1d` or `fm_mesh_2d` object
`loc`	Points for which to identify the containing interval/triangle, and corresponding barycentric coordinates. May be a vector (for 1d) or a matrix of raw coordinates, `sf`, or `sp` point information (for 2d).
`...`	Arguments forwarded to sub-methods.
`method`	character; method for defining the barycentric coordinates, "linear" (default) or "nearest"
`restricted`	logical, used for `method="linear"`. If `FALSE` (default), points outside the mesh interval will be given barycentric weights less than 0 and greater than 1, according to linear extrapolation. If `TRUE`, the barycentric weights are clamped to the (0, 1) interval.
`crs`	Optional crs information for `loc`

Value

A list with elements `t`; either

- vector of triangle indices (triangle meshes),
- matrix of interval knot indices (1D meshes), or
- matrix of lower left box indices (2D lattices),

and `bary`, a matrix of barycentric coordinates.

Methods (by class)

- `fm_bary(fm_mesh_1d)`: Return a list with elements `t` (start and endpoint knot indices) and `bary` (barycentric coordinates), both 2-column matrices.

  For `method = "nearest"`, `t[,1]` contains the index of the nearest mesh knot, and each row of `bary` contains `c(1, 0)`.

- `fm_bary(fm_mesh_2d)`: A list with elements `t` (vector of triangle indices) and `bary` (3-column matrix of barycentric coordinates). Points that were not found give `NA` entries in `t` and `bary`.

Examples

```
str(fm_bary(fmexample$mesh, fmexample$loc_sf))
str(fm_bary(fm_mesh_1d(1:4), seq(0, 5, by = 0.5)))
```

---

fm_basis *Compute mapping matrix between mesh function space and points*

Description

Computes the basis mapping matrix between a function space on a mesh, and locations.

Usage

```
fm_basis(x, ...)

## Default S3 method:
fm_basis(x, loc, ...)

## S3 method for class 'fm_mesh_1d'
fm_basis(x, loc, weights = NULL, derivatives = NULL, ...)

## S3 method for class 'fm_mesh_2d'
fm_basis(x, loc, weights = NULL, derivatives = NULL, ...)

## S3 method for class 'inla.mesh.1d'
fm_basis(x, loc, ...)

## S3 method for class 'inla.mesh'
fm_basis(x, loc, ...)

## S3 method for class 'fm_evaluator'
fm_basis(x, ...)

## S3 method for class 'fm_tensor'
fm_basis(x, loc, weights = NULL, ...)
```

Arguments

x	An object supported by the fm_evaluator() class
...	Currently unused
loc	A set of points of a class supported by fm_evaluator(x, loc = loc)
weights	Optional weight vector to apply (from the left, one weight for each row of the basis matrix)
derivatives	If non-NULL and logical, return a list, optionally including derivative matrices.

Value

A sparseMatrix

For fm_mesh_1d, a matrix, or if derivatives is TRUE, a list with elements

A	The projection matrix, u(loc_i)=sum_j A_ij w_i
d1A, d2A	Derivative weight matrices, du/dx(loc_i)=sum_j dx_ij w_i, etc.

For fm_mesh_2d, a matrix, or if derivatives is TRUE, a list with elements

A	The projection matrix, u(loc_i)=sum_j A_ij w_i
dx, dy, dz	Derivative weight matrices, du/dx(loc_i)=sum_j dx_ij w_i, etc.

For fm_tensor, a matrix

See Also

fm_raw_basis()

Examples

```
# Compute basis mapping matrix
str(fm_basis(fmexample$mesh, fmexample$loc))
```

---

fm_bbox *Bounding box class*

Description

Simple class for handling bounding box information

Usage

```
fm_bbox(...)

## S3 method for class 'list'
fm_bbox(x, ...)

## S3 method for class 'NULL'
fm_bbox(...)

## S3 method for class 'numeric'
fm_bbox(x, ...)

## S3 method for class 'matrix'
fm_bbox(x, ...)

## S3 method for class 'Matrix'
fm_bbox(x, ...)

## S3 method for class 'fm_bbox'
fm_bbox(x, ...)

## S3 method for class 'fm_mesh_1d'
fm_bbox(x, ...)

## S3 method for class 'fm_mesh_2d'
fm_bbox(x, ...)

## S3 method for class 'fm_segm'
fm_bbox(x, ...)

## S3 method for class 'fm_lattice_2d'
fm_bbox(x, ...)

## S3 method for class 'fm_tensor'
fm_bbox(x, ...)

## S3 method for class 'sf'
fm_bbox(x, ...)

## S3 method for class 'sfg'
fm_bbox(x, ...)

## S3 method for class 'sfc'
fm_bbox(x, ...)

## S3 method for class 'bbox'
fm_bbox(x, ...)
```

```
## S3 method for class 'inla.mesh'
fm_bbox(x, ...)

## S3 method for class 'inla.mesh.segment'
fm_bbox(x, ...)

fm_as_bbox(x, ...)

## S3 method for class 'fm_bbox'
x[i]

## S3 method for class 'fm_bbox'
c(..., .join = FALSE)
```

Arguments

`...`	Passed on to sub-methods
`x`	`fm_bbox` object from which to extract element(s)
`i`	indices specifying elements to extract
`.join`	logical; if `TRUE`, concatenate the bounding boxes into a single multi-dimensional bounding box. Default is `FALSE`.

Value

For `c.fm_bbox()`, a `fm_bbox_list` object if `join = FALSE` (the default) or an `fm_bbox` object if `join = TRUE`.

Methods (by class)

- `fm_bbox(list)`: Construct a bounding box from precomputed interval information, stored as a list of 2-vector ranges, `list(xlim, ylim, ...)`.

Methods (by generic)

- `[`: Extract sub-list
- `c(fm_bbox)`: The `...` arguments should be `fm_bbox` objects, or coercible with `fm_as_bbox(list(...))`.

Examples

```
fm_bbox(matrix(1:6, 3, 2))
m <- c(A = fm_bbox(cbind(1, 2)), B = fm_bbox(cbind(3, 4)))
str(m)
str(m[2])
```

fm_block *Blockwise aggregation matrices*

Description

Creates an aggregation matrix for blockwise aggregation, with optional weighting.

Usage

```
fm_block(
  block = NULL,
  weights = NULL,
  log_weights = NULL,
  rescale = FALSE,
  n_block = NULL
)

fm_block_eval(
  block = NULL,
  weights = NULL,
  log_weights = NULL,
  rescale = FALSE,
  n_block = NULL,
  values = NULL
)

fm_block_logsumexp_eval(
  block = NULL,
  weights = NULL,
  log_weights = NULL,
  rescale = FALSE,
  n_block = NULL,
  values = NULL,
  log = TRUE
)

fm_block_weights(
  block = NULL,
  weights = NULL,
  log_weights = NULL,
  rescale = FALSE,
  n_block = NULL
)

fm_block_log_weights(
  block = NULL,
  weights = NULL,
```

```
  log_weights = NULL,
  rescale = FALSE,
  n_block = NULL
)

fm_block_log_shift(block = NULL, log_weights = NULL, n_block = NULL)

fm_block_prep(
  block = NULL,
  log_weights = NULL,
  weights = NULL,
  n_block = NULL,
  values = NULL,
  n_values = NULL,
  force_log = FALSE
)
```

Arguments

`block`	integer vector; block information. If `NULL`, `rep(1L, block_len)` is used, where `block_len` is determined by `length(log_weights)))` or `length(weights)))`. A single scalar is also repeated to a vector of corresponding length to the weights.
`weights`	Optional weight vector
`log_weights`	Optional `log(weights)` vector. Overrides `weights` when non-NULL.
`rescale`	logical; If `TRUE`, normalise the weights by `sum(weights)` or `sum(exp(log_weights))` within each block. Default: `FALSE`
`n_block`	integer; The number of conceptual blocks. Only needs to be specified if it's larger than `max(block)`, or to keep the output of consistent size for different inputs.
`values`	Vector to be blockwise aggregated
`log`	If `TRUE` (default), return log-sum-exp. If `FALSE`, return sum-exp.
`n_values`	When supplied, used instead of `length(values)` to determine the value vector input length.
`force_log`	When `FALSE` (default), passes either `weights` and `log_weights` on, if provided, with `log_weights` taking precedence. If `TRUE`, forces the computation of `log_weights`, whether given in the input or not.

Value

A (sparse) matrix

Functions

- `fm_block()`: A (sparse) matrix of size `n_block` times `length(block)`.
- `fm_block_eval()`: Evaluate aggregation. More efficient alternative to to `as.vector(fm_block(...) %*% values)`.

- `fm_block_logsumexp_eval()`: Evaluate log-sum-exp aggregation. More efficient and numerically stable alternative to to `log(as.vector(fm_block(...) %*% exp(values)))`.
- `fm_block_weights()`: Computes (optionally) blockwise renormalised weights
- `fm_block_log_weights()`: Computes (optionally) blockwise renormalised log-weights
- `fm_block_log_shift()`: Computes shifts for stable blocked log-sum-exp. To compute $\log(\sum_{i;\text{block}_i=k} \exp(v_i)w_i)$ for each block `k`, first compute combined values and weights, and a shift:

  ```
  w_values <- values + fm_block_log_weights(block, log_weights = log_weights)
  shift <- fm_block_log_shift(block, log_weights = w_values)
  ```

  Then aggregate the values within each block:

  ```
  agg <- aggregate(exp(w_values - shift[block]),
                   by = list(block = block),
                   \(x) log(sum(x)))
  agg$x <- agg$x + shift[agg$block]
  ```

  The implementation uses a faster method:

  ```
  as.vector(
    Matrix::sparseMatrix(
      i = block,
      j = rep(1L, length(block)),
      x = exp(w_values - shift[block]),
      dims = c(n_block, 1))
  ) + shift
  ```

- `fm_block_prep()`: Helper function for preparing `block`, `weights`, and `log_weights`, `n_block` inputs.

Examples

```
block <- rep(1:2, 3:2)
fm_block(block)
fm_block(block, rescale = TRUE)
fm_block(block, log_weights = -2:2, rescale = TRUE)
fm_block_eval(
  block,
  weights = 1:5,
  rescale = TRUE,
  values = 11:15
)
fm_block_logsumexp_eval(
  block,
  weights = 1:5,
  rescale = TRUE,
  values = log(11:15),
  log = FALSE
)
```

fm_centroids *Extract triangle centroids from an* fm_mesh_2d

Description

Computes the centroids of the triangles of an fm_mesh_2d() object.

Usage

```
fm_centroids(x, format = NULL)
```

Arguments

x	An fm_mesh_2d or inla.mesh object.
format	character; "sf", "df", "sp"

Value

An sf, data.frame, or SpatialPointsDataFrame object, with the vertex coordinates, and a .triangle column with the triangle indices.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

fm_vertices()

Examples

```
if (require("ggplot2", quietly = TRUE)) {
  vrt <- fm_centroids(fmexample$mesh, format = "sf")
  ggplot() +
    geom_sf(data = fm_as_sfc(fmexample$mesh)) +
    geom_sf(data = vrt, color = "red")
}
```

fm_contains *Check which mesh triangles are inside a polygon*

Description

Wrapper for the sf::st_contains() (previously sp::over()) method to find triangle centroids or vertices inside sf or sp polygon objects

Usage

```
fm_contains(x, y, ...)

## S3 method for class 'Spatial'
fm_contains(x, y, ...)

## S3 method for class 'sf'
fm_contains(x, y, ...)

## S3 method for class 'sfc'
fm_contains(x, y, ..., type = c("centroid", "vertex"))
```

Arguments

x	geometry (typically an sf or sp::SpatialPolygons object) for the queries
y	an fm_mesh_2d() or inla.mesh object
...	Passed on to other methods
type	the query type; either 'centroid' (default, for triangle centroids), or 'vertex' (for mesh vertices)

Value

List of vectors of triangle indices (when type is 'centroid') or vertex indices (when type is 'vertex'). The list has one entry per row of the sf object. Use unlist(fm_contains(...)) if the combined union is needed.

Author(s)

Haakon Bakka, <bakka@r-inla.org>, and Finn Lindgren <finn.lindgren@gmail.com>

Examples

```
if (TRUE &&
  fm_safe_sp()) {
  # Create a polygon and a mesh
  obj <- sp::SpatialPolygons(
    list(sp::Polygons(
      list(sp::Polygon(rbind(
```

```
        c(0, 0),
        c(50, 0),
        c(50, 50),
        c(0, 50)
      ))),
      ID = 1
    )),
    proj4string = fm_CRS("longlat_globe")
  )
  mesh <- fm_rcdt_2d_inla(globe = 2, crs = fm_crs("sphere"))

  ## 3 vertices found in the polygon
  fm_contains(obj, mesh, type = "vertex")

  ## 3 triangles found in the polygon
  fm_contains(obj, mesh)

  ## Multiple transformations can lead to slightly different results due to edge cases
  ## 4 triangles found in the polygon
  fm_contains(
    obj,
    fm_transform(mesh, crs = fm_crs("mollweide_norm"))
  )
}
```

---

fm_CRS *Create a coordinate reference system object*

Description

Creates either a CRS object or an inla.CRS object, describing a coordinate reference system

Usage

```
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'fm_CRS'
is.na(x)

## S3 method for class 'crs'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'fm_crs'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'Spatial'
fm_CRS(x, oblique = NULL, ...)
```

```
## S3 method for class 'fm_CRS'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'SpatVector'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'SpatRaster'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'sf'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'sfc'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'sfg'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'fm_mesh_2d'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'fm_lattice'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'fm_segm'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'matrix'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'CRS'
fm_CRS(x, oblique = NULL, ...)

## Default S3 method:
fm_CRS(
  x,
  oblique = NULL,
  projargs = NULL,
  doCheckCRSArgs = NULL,
  args = NULL,
  SRS_string = NULL,
  ...
)

## S3 method for class 'inla.CRS'
is.na(x)

## S3 method for class 'inla.CRS'
```

```
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'inla.mesh'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'inla.mesh.lattice'
fm_CRS(x, oblique = NULL, ...)

## S3 method for class 'inla.mesh.segment'
fm_CRS(x, oblique = NULL, ...)
```

Arguments

x	Object to convert to CRS or to extract CRS information from.
oblique	Vector of length at most 4 of rotation angles (in degrees) for an oblique projection, all values defaulting to zero. The values indicate (longitude, latitude, orientation, orbit), as explained in the Details section for `fm_crs()`.
...	Additional parameters, passed on to sub-methods.
projargs	Either 1) a projection argument string suitable as input to `sp::CRS`, or 2) an existing `CRS` object, or 3) a shortcut reference string to a predefined projection; run `names(fm_wkt_predef())` for valid predefined projections. (projargs is a compatibility parameter that can be used for the default `fm_CRS()` method)
doCheckCRSArgs	ignored.
args	An optional list of name/value pairs to add to and/or override the PROJ4 arguments in `projargs`. `name=value` is converted to `"+name=value"`, and `name=NA` is converted to `"+name"`.
SRS_string	a WKT2 string defining the coordinate system; see `sp::CRS`. This takes precedence over `projargs`.

Details

The first two elements of the `oblique` vector are the (longitude, latitude) coordinates for the oblique centre point. The third value (orientation) is a counterclockwise rotation angle for an observer looking at the centre point from outside the sphere. The fourth value is the quasi-longitude (orbit angle) for a rotation along the oblique observers equator.

Simple oblique: `oblique=c(0, 45)`

Polar: `oblique=c(0, 90)`

Quasi-transversal: `oblique=c(0, 0, 90)`

Satellite orbit viewpoint: `oblique=c(lon0-time*v1, 0, orbitangle, orbit0+time*v2)`, where `lon0` is the longitude at which a satellite orbit crosses the equator at `time=0`, when the satellite is at an angle `orbit0` further along in its orbit. The orbital angle relative to the equatorial plane is `orbitangle`, and `v1` and `v2` are the angular velocities of the planet and the satellite, respectively. Note that "forward" from the satellite's point of view is "to the right" in the projection.

When `oblique[2]` or `oblique[3]` are non-zero, the resulting projection is only correct for perfect spheres.

Value

Either an sp::CRS object or an inla.CRS object, depending on if the coordinate reference system described by the parameters can be expressed with a pure sp::CRS object or not.

An S3 inla.CRS object is a list, usually (but not necessarily) containing at least one element:

crs	The basic sp::CRS object

Functions

- is.na(fm_CRS): Check if a fm_CRS has NA crs information and NA obliqueness
- is.na(inla.CRS): Check if a inla.CRS has NA crs information and NA obliqueness

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

fm_crs(), sp::CRS(), fm_crs_wkt, fm_sp_get_crs(), fm_crs_is_identical()

Examples

```
crs1 <- fm_CRS("longlat_globe")
crs2 <- fm_CRS("lambert_globe")
crs3 <- fm_CRS("mollweide_norm")
crs4 <- fm_CRS("hammer_globe")
crs5 <- fm_CRS("sphere")
crs6 <- fm_CRS("globe")
```

---

fm_crs *Obtain coordinate reference system object*

Description

Obtain an sf::crs or fm_crs object from a spatial object, or convert crs information to construct a new sf::crs object.

Usage

```
fm_crs(x, oblique = NULL, ..., crsonly = deprecated())

fm_crs_oblique(x)

## S3 method for class 'fm_crs'
st_crs(x, ...)

## S3 method for class 'fm_crs'
```

```
x$name

## Default S3 method:
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'crs'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'fm_crs'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'fm_CRS'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'character'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'Spatial'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'SpatVector'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'SpatRaster'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'sf'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'sfc'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'sfg'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'fm_mesh_2d'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'fm_lattice_2d'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'fm_segm'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'fm_list'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'matrix'
```

```
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'fm_list'
fm_CRS(x, oblique = NULL, ...)

fm_wkt_predef()

## S3 method for class 'inla.CRS'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'inla.mesh'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'inla.mesh.lattice'
fm_crs(x, oblique = NULL, ...)

## S3 method for class 'inla.mesh.segment'
fm_crs(x, oblique = NULL, ...)
```

Arguments

`x`	Object to convert to `crs` or to extract `crs` information from. If `character`, a string suitable for `sf::st_crs(x)`, or the name of a predefined `wkt` string from "names(fm_wkt_predef())'.
`oblique`	Numeric vector of length at most 4 of rotation angles (in degrees) for an oblique projection, all values defaulting to zero. The values indicate (longitude, latitude, orientation, orbit), as explained in the Details section below. When `oblique` is non-NULL, used to override the obliqueness parameters of a `fm_crs` object. When `NA`, remove obliqueness from the object, resulting in a return class of `sf::st_crs()`. When `NULL`, pass though any oblique information in the object, returning an `fm_crs()` object if needed.
`...`	Additional parameters. Not currently in use.
`crsonly`	**[Deprecated]** logical; if TRUE, remove `oblique` information from `fm_crs` objects and return a plain `crs` object instead. For `crsonly = TRUE`, use `oblique = NA` instead. For `crsonly = FALSE`, use default, NULL, or non-NA `oblique`.
`name`	element name

Details

The first two elements of the `oblique` vector are the (longitude, latitude) coordinates for the oblique centre point. The third value (orientation) is a counter-clockwise rotation angle for an observer looking at the centre point from outside the sphere. The fourth value is the quasi-longitude (orbit angle) for a rotation along the oblique observers equator.

Simple oblique: `oblique=c(0, 45)`

Polar: `oblique=c(0, 90)`

Quasi-transversal: `oblique=c(0, 0, 90)`

Satellite orbit viewpoint: `oblique=c(lon0-time*v1, 0, orbitangle, orbit0+time*v2)`, where `lon0` is the longitude at which a satellite orbit crosses the equator at `time=0`, when the satellite is at an angle `orbit0` further along in its orbit. The orbital angle relative to the equatorial plane is `orbitangle`, and `v1` and `v2` are the angular velocities of the planet and the satellite, respectively. Note that "forward" from the satellite's point of view is "to the right" in the projection.

When `oblique[2]` or `oblique[3]` are non-zero, the resulting projection is only correct for perfect spheres.

Value

Either an `sf::crs` object or an `fm_crs` object, depending on if the coordinate reference system described by the parameters can be expressed with a pure `crs` object or not.

A `crs` object (`sf::st_crs()`) or a `fm_crs` object. An S3 `fm_crs` object is a list with elements `crs` and `oblique`.

`fm_wkt_predef` returns a WKT2 string defining a projection

Methods (by class)

- `fm_crs(fm_list)`: returns a list of 'crs' objects, one for each list element

Methods (by generic)

- `st_crs(fm_crs)`: `st_crs(x, ...)` is equivalent to `fm_crs(x, ... oblique = NA)` when `x` is a `fm_crs` object.
- `$`: For a `fm_crs` object `x`, `x$name` calls the accessor method for the `crs` object inside it. If `name` is "crs", the internal crs object itself is returned. If `name` is "oblique", the internal oblique angle parameter vector is returned.

Functions

- `fm_crs_oblique()`: Return `NA` for object with no oblique information, and otherwise a length 4 numeric vector.
- `fm_CRS(fm_list)`: returns a list of 'CRS' objects, one for each list element

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

`sf::st_crs()`, `fm_crs_wkt`

fm_crs_is_null

`fm_crs<-()`, `fm_crs_oblique<-()`

Examples

```
crs1 <- fm_crs("longlat_globe")
crs2 <- fm_crs("lambert_globe")
crs3 <- fm_crs("mollweide_norm")
crs4 <- fm_crs("hammer_globe")
crs5 <- fm_crs("sphere")
crs6 <- fm_crs("globe")
names(fm_wkt_predef())
```

---

fm_crs<- *Assignment operators for crs information objects*

Description

Assigns new crs information.

Usage

```
fm_crs(x) <- value

fm_crs_oblique(x) <- value

## S3 replacement method for class 'NULL'
fm_crs(x) <- value

## S3 replacement method for class 'NULL'
fm_crs_oblique(x) <- value

## S3 replacement method for class 'fm_segm'
fm_crs(x) <- value

## S3 replacement method for class 'fm_list'
fm_crs(x) <- value

## S3 replacement method for class 'fm_mesh_2d'
fm_crs(x) <- value

## S3 replacement method for class 'fm_lattice_2d'
fm_crs(x) <- value

## S3 replacement method for class 'sf'
fm_crs(x) <- value

## S3 replacement method for class 'sfg'
fm_crs(x) <- value

## S3 replacement method for class 'sfc'
```

```
fm_crs(x) <- value

## S3 replacement method for class 'Spatial'
fm_crs(x) <- value

## S3 replacement method for class 'crs'
fm_crs_oblique(x) <- value

## S3 replacement method for class 'CRS'
fm_crs_oblique(x) <- value

## S3 replacement method for class 'fm_CRS'
fm_crs_oblique(x) <- value

## S3 replacement method for class 'fm_crs'
fm_crs_oblique(x) <- value

## S3 replacement method for class 'fm_segm'
fm_crs_oblique(x) <- value

## S3 replacement method for class 'fm_mesh_2d'
fm_crs_oblique(x) <- value

## S3 replacement method for class 'fm_lattice_2d'
fm_crs_oblique(x) <- value

## S3 replacement method for class 'inla.CRS'
fm_crs_oblique(x) <- value
```

Arguments

x	Object to assign crs information to
value	For `fm_crs<-()`, object supported by `fm_crs(value)`. For `fm_crs_oblique<-()`, `NA` or a numeric vector, see the `oblique` argument for `fm_crs()`. For assignment, `NULL` is treated as `NA`.

Value

The modified object

Functions

- `fm_crs(x) <- value`: Automatically converts the input value with `fm_crs(value)`, `fm_crs(value, oblique = NA)`, `fm_CRS(value)`, or `fm_CRS(value, oblique = NA)`, depending on the type of `x`.
- `fm_crs_oblique(x) <- value`: Assigns new `oblique` information.

See Also

fm_crs()

Examples

```
x <- fm_segm()
fm_crs(x) <- fm_crs("+proj=longlat")
fm_crs(x)$proj4string
```

---

fm_crs_is_identical *Check if two CRS objects are identical*

Description

Check if two CRS objects are identical

Usage

```
fm_crs_is_identical(crs0, crs1, crsonly = FALSE)

fm_identical_CRS(crs0, crs1, crsonly = FALSE)
```

Arguments

crs0, crs1	Two sf::crs, sp::CRS, fm_crs or inla.CRS objects to be compared.
crsonly	logical. If TRUE and any of crs0 and crs1 are fm_crs or inla.CRS objects, extract and compare only the sf::crs or sp::CRS aspects. Default: FALSE

Value

logical, indicating if the two crs objects are identical in the specified sense (see the crsonly argument)

Functions

- fm_identical_CRS(): **[Deprecated]** by fm_crs_is_identical().

See Also

fm_crs(), fm_CRS(), fm_crs_is_null()

Examples

```
crs0 <- crs1 <- fm_crs("longlat_globe")
fm_crs_oblique(crs1) <- c(0, 90)
print(c(
  fm_crs_is_identical(crs0, crs0),
  fm_crs_is_identical(crs0, crs1),
  fm_crs_is_identical(crs0, crs1, crsonly = TRUE)
))
```

---

fm_crs_is_null *Check if a crs is NULL or NA*

Description

Methods of checking whether various kinds of CRS objects are `NULL` or `NA`. Logically equivalent to either `is.na(fm_crs(x))` or `is.na(fm_crs(x, oblique = NA))`, but with a short-cut pre-check for `is.null(x)`.

Usage

```
fm_crs_is_null(x, crsonly = FALSE)

## S3 method for class 'fm_crs'
is.na(x)
```

Arguments

`x`	An object supported by `fm_crs(x)`
`crsonly`	For crs objects with extended functionality, such as `fm_crs()` objects with `oblique` information, `crsonly = TRUE` only checks the plain CRS part.

Value

logical

Functions

- `fm_crs_is_null()`: Check if an object is or has `NULL` or `NA` CRS information. If not `NULL`, `is.na(fm_crs(x))` is returned. This allows the input to be e.g. a proj4string or epsg number, since the default `fm_crs()` method passes its argument on to `sf::st_crs()`.
- `is.na(fm_crs)`: Check if a `fm_crs` has `NA` crs information and `NA` obliqueness

See Also

`fm_crs()`, fm_CRS(), `fm_crs_is_identical()`

Examples

```
fm_crs_is_null(NULL)
fm_crs_is_null(27700)
fm_crs_is_null(fm_crs())
fm_crs_is_null(fm_crs(27700))
fm_crs_is_null(fm_crs(oblique = c(1, 2, 3, 4)))
fm_crs_is_null(fm_crs(oblique = c(1, 2, 3, 4)), crsonly = TRUE)
fm_crs_is_null(fm_crs(27700, oblique = c(1, 2, 3, 4)))
fm_crs_is_null(fm_crs(27700, oblique = c(1, 2, 3, 4)), crsonly = TRUE)
```

---

fm_crs_wkt *Handling CRS/WKT*

Description

Get and set CRS object or WKT string properties.

Usage

```
fm_wkt_is_geocent(wkt)

fm_crs_is_geocent(crs)

fm_wkt_get_ellipsoid_radius(wkt)

fm_crs_get_ellipsoid_radius(crs)

fm_ellipsoid_radius(x)

## Default S3 method:
fm_ellipsoid_radius(x)

## S3 method for class 'character'
fm_ellipsoid_radius(x)

fm_wkt_set_ellipsoid_radius(wkt, radius)

fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'character'
fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'CRS'
fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'fm_CRS'
```

```
fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'crs'
fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'fm_crs'
fm_ellipsoid_radius(x) <- value

fm_crs_set_ellipsoid_radius(crs, radius)

fm_wkt_unit_params()

fm_wkt_get_lengthunit(wkt)

fm_wkt_set_lengthunit(wkt, unit, params = NULL)

fm_crs_get_lengthunit(crs)

fm_crs_set_lengthunit(crs, unit)

fm_length_unit(x)

## Default S3 method:
fm_length_unit(x)

## S3 method for class 'character'
fm_length_unit(x)

fm_length_unit(x) <- value

## S3 replacement method for class 'character'
fm_length_unit(x) <- value

## S3 replacement method for class 'CRS'
fm_length_unit(x) <- value

## S3 replacement method for class 'fm_CRS'
fm_length_unit(x) <- value

## S3 replacement method for class 'crs'
fm_length_unit(x) <- value

## S3 replacement method for class 'fm_crs'
fm_length_unit(x) <- value

fm_wkt(crs)

fm_proj4string(crs)
```

```
fm_crs_get_wkt(crs)

fm_wkt_tree_projection_type(wt)

fm_wkt_projection_type(wkt)

fm_crs_projection_type(crs)

fm_crs_bounds(crs, warn.unknown = FALSE)

## S3 replacement method for class 'inla.CRS'
fm_ellipsoid_radius(x) <- value

## S3 replacement method for class 'inla.CRS'
fm_length_unit(x) <- value
```

Arguments

wkt	A WKT2 character string
crs	An sf::crs, sp::CRS, fm_crs or inla.CRS object
x	crs object to extract value from or assign values in
radius	numeric; The new radius value
value	Value to assign
unit	character, name of a unit. Supported names are "metre", "kilometre", and the aliases "meter", "m", International metre", "kilometer", and "km", as defined by fm_wkt_unit_params or the params argument. (For legacy PROJ4 use, only "m" and "km" are supported)
params	Length unit definitions, in the list format produced by fm_wkt_unit_params(), Default: NULL, which invokes fm_wkt_unit_params()
wt	A parsed wkt tree, see fm_wkt_as_wkt_tree()
warn.unknown	logical, default FALSE. Produce warning if the shape of the projection bounds is unknown.

Value

For fm_wkt_unit_params, a list of named unit definitions

For fm_wkt_get_lengthunit, a list of length units used in the wkt string, excluding the ellipsoid radius unit.

For fm_wkt_set_lengthunit, a WKT2 string with altered length units. Note that the length unit for the ellipsoid radius is unchanged.

For fm_crs_get_lengthunit, a list of length units used in the wkt string, excluding the ellipsoid radius unit. (For legacy PROJ4 code, the raw units from the proj4string are returned, if present.)

For fm_length_unit<-, a crs object with altered length units. Note that the length unit for the ellipsoid radius is unchanged.

Functions

- `fm_wkt()`: Returns a WKT2 string, for any input supported by `fm_crs()`.
- `fm_proj4string()`: Returns a proj4 string, for any input supported by `fm_crs()`.
- `fm_crs_get_wkt()`: **[Deprecated]** Use `fm_wkt()` instead.
- `fm_wkt_tree_projection_type()`: Returns "longlat", "lambert", "mollweide", "hammer", "tmerc", or `NULL`
- `fm_wkt_projection_type()`: See `fm_wkt_tree_projection_type`
- `fm_crs_projection_type()`: See `fm_wkt_tree_projection_type`
- `fm_crs_bounds()`: Returns bounds information for a projection, as a list with elements `type` ("rectangle" or "ellipse"), `xlim`, `ylim`, and `polygon`.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

`fm_crs()`

Examples

```
c1 <- fm_crs("globe")
fm_crs_get_lengthunit(c1)
c2 <- fm_crs_set_lengthunit(c1, "m")
fm_crs_get_lengthunit(c2)
```

---

fm_detect_manifold *Detect manifold type*

Description

Detect if a 2d object is on "R2", "S2", or "M2"

Usage

```
fm_detect_manifold(x)

fm_crs_detect_manifold(x)

## S3 method for class 'crs'
fm_detect_manifold(x)

## S3 method for class 'CRS'
fm_detect_manifold(x)
```

```
## S3 method for class 'numeric'
fm_detect_manifold(x)

## S3 method for class 'matrix'
fm_detect_manifold(x)

## S3 method for class 'fm_mesh_2d'
fm_detect_manifold(x)
```

Arguments

x	Object to investigate

Value

A string containing the detected manifold classification

Functions

- `fm_crs_detect_manifold()`: Detect if a crs is on "R2" or "S2" (if `fm_crs_is_geocent(crs)` is `TRUE`). Returns `NA_character_` if the crs is NULL or NA.

Examples

```
fm_detect_manifold(1:4)
fm_detect_manifold(rbind(c(1, 0, 0), c(0, 1, 0), c(1, 1, 0)))
fm_detect_manifold(rbind(c(1, 0, 0), c(0, 1, 0), c(0, 0, 1)))
```

---

fm_diameter *Diameter bound for a geometric object*

Description

Find an upper bound to the convex hull of a point set

Usage

```
fm_diameter(x, ...)

## S3 method for class 'matrix'
fm_diameter(x, manifold = NULL, ...)

## S3 method for class 'sf'
fm_diameter(x, ...)

## S3 method for class 'sfg'
fm_diameter(x, ...)
```

```
## S3 method for class 'sfc'
fm_diameter(x, ...)

## S3 method for class 'fm_lattice_2d'
fm_diameter(x, ...)

## S3 method for class 'fm_segm'
fm_diameter(x, ...)

## S3 method for class 'fm_mesh_2d'
fm_diameter(x, ...)

## S3 method for class 'fm_mesh_1d'
fm_diameter(x, ...)

## S3 method for class 'inla.mesh.1d'
fm_diameter(x, ...)

## S3 method for class 'inla.mesh.segment'
fm_diameter(x, ...)

## S3 method for class 'inla.mesh.lattice'
fm_diameter(x, ...)

## S3 method for class 'inla.mesh'
fm_diameter(x, ...)
```

Arguments

`x`	A point set as an $n \times d$ matrix, or an `fm_mesh_2d/1d/sf` related object.
`...`	Additional parameters passed on to the submethods.
`manifold`	Character string specifying the manifold type. Default is to treat the point set with Euclidean R^d metrics. Use `manifold="S2"` for great circle distances on the unit sphere (this is set automatically for `fm_fmesh_2d` objects).

Value

A scalar, upper bound for the diameter of the convex hull of the point set.

Author(s)

Finn Lindgren finn.lindgren@gmail.com

Examples

```
fm_diameter(matrix(c(0, 1, 1, 0, 0, 0, 1, 1), 4, 2))
```

fm_dof — *Function spece degrees of freedom*

Description

Obtain the degrees of freedom of a function space, i.e. the number of basis functions it uses.

Usage

```
fm_dof(x)

## S3 method for class 'fm_mesh_1d'
fm_dof(x)

## S3 method for class 'fm_mesh_2d'
fm_dof(x)

## S3 method for class 'fm_tensor'
fm_dof(x)
```

Arguments

x	A function space object, such as `fm_mesh_1d()` or `fm_mesh_2d()`

Value

An integer

Examples

```
fm_dof(fmexample$mesh)
```

fm_evaluate — *Methods for projecting to/from mesh objects*

Description

Calculate evaluation information and/or evaluate a function defined on a mesh or function space.

Usage

```
fm_evaluate(...)

## Default S3 method:
fm_evaluate(mesh, field, ...)

## S3 method for class 'fm_evaluator'
fm_evaluate(projector, field, ...)

fm_evaluator(...)

## S3 method for class 'fm_mesh_2d'
fm_evaluator(mesh, loc = NULL, lattice = NULL, crs = NULL, ...)

## S3 method for class 'fm_mesh_1d'
fm_evaluator(mesh, loc = NULL, xlim = mesh$interval, dims = 100, ...)

## S3 method for class 'fm_tensor'
fm_evaluator(x, loc, ...)

fm_evaluator_lattice(
  mesh,
  xlim = NULL,
  ylim = NULL,
  dims = c(100, 100),
  projection = NULL,
  crs = NULL,
  ...
)

## S3 method for class 'inla.mesh'
fm_evaluator(mesh, loc = NULL, lattice = NULL, crs = NULL, ...)

## S3 method for class 'inla.mesh.1d'
fm_evaluator(mesh, loc = NULL, xlim = mesh$interval, dims = 100, ...)
```

Arguments

`...`	Additional arguments passed on to methods.
`mesh`	An `inla.mesh` or `inla.mesh.1d` object.
`field`	Basis function weights, one per mesh basis function, describing the function to be evaluated at the projection locations
`projector`	An `fm_evaluator` object.
`loc`	Projection locations. Can be a matrix, `SpatialPoints`, `SpatialPointsDataFrame`, `sf`, `sfc`, or `sfg` object.
`lattice`	An `fm_lattice_2d()` object.
`crs`	An optional CRS or inla.CRS object associated with `loc` and/or `lattice`.

xlim	X-axis limits for a lattice. For R2 meshes, defaults to covering the domain.
dims	Lattice dimensions.
x	fm_tensor() object
ylim	Y-axis limits for a lattice. For R2 meshes, defaults to covering the domain.
projection	One of c("default", "longlat", "longsinlat", "mollweide").

Value

A vector or matrix of the evaluated function

An fm_evaluator object

Methods (by class)

- fm_evaluate(default): The default method calls proj = fm_evaluator(mesh, ...), followed by fm_evaluate(proj, field).

Functions

- fm_evaluate(): Returns the field function evaluated at the locations determined by an fm_evaluator object. fm_evaluate(mesh, field = field, ...) is a shortcut to fm_evaluate(fm_evaluator(mesh, ...), field = field).
- fm_evaluator(): Returns an fm_evaluator list object with evaluation information. The proj element contains a mapping matrix A and a logical vector ok, that indicates which locations were mappable to the input mesh. For fm_mesh_2d and inla.mesh input, proj also contains a matrix bary and vector t, with the barycentric coordinates within the triangle each input location falls in.
- fm_evaluator(fm_mesh_2d): The ... arguments are passed on to fm_evaluator_lattice() if no loc or lattice is provided.
- fm_evaluator_lattice(): Creates an fm_lattice_2d() object, by default covering the input mesh.
- fm_evaluator(inla.mesh): The ... arguments are passed on to fm_evaluator_lattice() if no loc or lattice is provided.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

fm_mesh_2d(), fm_mesh_1d(), fm_lattice_2d()

Examples

```
if (TRUE) {
  n <- 20
  loc <- matrix(runif(n * 2), n, 2)
  mesh <- fm_rcdt_2d_inla(loc, refine = list(max.edge = 0.05))
```

```
  proj <- fm_evaluator(mesh)
  field <- cos(mesh$loc[, 1] * 2 * pi * 3) * sin(mesh$loc[, 2] * 2 * pi * 7)
  image(proj$x, proj$y, fm_evaluate(proj, field))
}

# if (require("ggplot2") &&
#  require("ggpolypath")) {
#  ggplot() +
#    gg(data = fm_as_sfc(mesh), col = field)
# }
```

`fm_fem` *Compute finite element matrices*

Description

(...)

Usage

```
fm_fem(mesh, order = 2, ...)

## S3 method for class 'fm_mesh_1d'
fm_fem(mesh, order = 2, ...)

## S3 method for class 'fm_mesh_2d'
fm_fem(mesh, order = 2, aniso = NULL, ...)

## S3 method for class 'inla.mesh.1d'
fm_fem(mesh, order = 2, ...)

## S3 method for class 'inla.mesh'
fm_fem(mesh, order = 2, ...)

## S3 method for class 'fm_tensor'
fm_fem(mesh, order = 2, ...)
```

Arguments

`mesh`	`fm_mesh_1d` or other supported mesh class object
`order`	integer
`...`	Currently unused
`aniso`	If non-NULL, a `list(gamma, v)`. Calculates anisotropic structure matrices (in addition to the regular) for γ and v for an anisotropic operator $\nabla \cdot H \nabla$, where $H = \gamma I + vv^\top$. Currently (2023-08-05) the fields need to be given per vertex.

Value

`fm_fem.fm_mesh_1d`: A list with elements `c0`, `c1`, `g1`, `g2`. When `mesh$degree == 2`, also `g01`, `g02`, and `g12`.

`fm_fem.fm_mesh_2d`: A list with elements `c0`, `c1`, `g1`, `va`, `ta`, and more if `order > 1`. When `aniso` is non-NULL, also `g1aniso` matrices, etc.

`fm_fem.fm_tensor`: A list with elements `cc`, `g1`, `g2`.

Examples

```
str(fm_fem(fmexample$mesh))
```

---

fm_gmrf — *SPDE, GMRF, and Matérn process methods*

Description

[Experimental] Methods for SPDEs and GMRFs.

Usage

```
fm_matern_precision(x, alpha, rho, sigma)

fm_matern_sample(x, alpha = 2, rho, sigma, n = 1, loc = NULL)

fm_covariance(Q, A1 = NULL, A2 = NULL, partial = FALSE)

fm_sample(n, Q, mu = 0, constr = NULL)
```

Arguments

`x`	A mesh object, e.g. from `fm_mesh_1d()` or `fm_mesh_2d()`.
`alpha`	The SPDE operator order. The resulting smoothness index is `nu = alpha - dim / 2`.
`rho`	The Matérn range parameter (scale parameter `kappa = sqrt(8 * nu) / rho`)
`sigma`	The nominal Matérn std.dev. parameter
`n`	The number of samples to generate
`loc`	locations to evaluate the random field, compatible with `fm_evaluate(x, loc = loc, field = ...)`
`Q`	A precision matrix
`A1, A2`	Matrices, typically obtained from `fm_basis()` and/or `fm_block()`.

partial **[Experimental]** If TRUE, compute the partial inverse of Q, i.e. the elements of the inverse corresponding to the non-zero pattern of Q. (Note: This can be done efficiently with the Takahashi recursion method, but to avoid an RcppEigen dependency this is currently disabled, and a slower method is used until the efficient method is reimplemented.)

mu Optional mean vector

constr Optional list of constraint information, with elements A and e. Should only be used for a small number of exact constraints.

Value

fm_matern_sample() returns a matrix, where each column is a sampled field. If loc is NULL, the fm_dof(mesh) basis weights are given. Otherwise, the evaluated field at the nrow(loc) locations loc are given (from version 0.1.4.9001)

Functions

- fm_matern_precision(): Construct the (sparse) precision matrix for the basis weights for Whittle-Matérn SPDE models. The boundary behaviour is determined by the provided mesh function space.
- fm_matern_sample(): Simulate a Matérn field given a mesh and covariance function parameters, and optionally evaluate at given locations.
- fm_covariance(): Compute the covariance between "A1 x" and "A2 x", when x is a basis vector with precision matrix Q.
- fm_sample(): Generate n samples based on a sparse precision matrix Q

Examples

```
library(Matrix)
mesh <- fm_mesh_1d(-20:120, degree = 2)
Q <- fm_matern_precision(mesh, alpha = 2, rho = 15, sigma = 1)
x <- seq(0, 100, length.out = 601)
A <- fm_basis(mesh, x)
plot(x,
  as.vector(Matrix::diag(fm_covariance(Q, A))),
  type = "l",
  ylab = "marginal variances"
)

plot(x,
  fm_evaluate(mesh, loc = x, field = fm_sample(1, Q)[, 1]),
  type = "l",
  ylab = "process sample"
)
```

fm_int *Multi-domain integration*

Description

Construct integration points on tensor product spaces

Usage

```
fm_int(domain, samplers = NULL, ...)

## S3 method for class 'list'
fm_int(domain, samplers = NULL, ...)

## S3 method for class 'numeric'
fm_int(domain, samplers = NULL, name = "x", ...)

## S3 method for class 'character'
fm_int(domain, samplers = NULL, name = "x", ...)

## S3 method for class 'factor'
fm_int(domain, samplers = NULL, name = "x", ...)

## S3 method for class 'SpatRaster'
fm_int(domain, samplers = NULL, name = "x", ...)

## S3 method for class 'fm_lattice_2d'
fm_int(domain, samplers = NULL, name = "x", ...)

## S3 method for class 'fm_mesh_1d'
fm_int(domain, samplers = NULL, name = "x", int.args = NULL, ...)

## S3 method for class 'fm_mesh_2d'
fm_int(
  domain,
  samplers = NULL,
  name = NULL,
  int.args = NULL,
  format = NULL,
  ...
)

## S3 method for class 'inla.mesh.lattice'
fm_int(domain, samplers = NULL, name = "x", ...)

## S3 method for class 'inla.mesh.1d'
fm_int(domain, samplers = NULL, name = "x", int.args = NULL, ...)
```

```
## S3 method for class 'inla.mesh'
fm_int(
  domain,
  samplers = NULL,
  name = NULL,
  int.args = NULL,
  format = NULL,
  ...
)
```

Arguments

domain	Functional space specification; single domain or a named list of domains
samplers	For single domain `fm_int` methods, an object specifying one or more subsets of the domain, and optional weighting in a `weight` variable. For `fm_int.list`, a list of sampling definitions, where data frame elements may contain information for multiple domains, in which case each row represent a separate tensor product integration subspace.
...	Additional arguments passed on to other methods
name	For single-domain methods, the variable name to use for the integration points. Default 'x'
int.args	List of arguments passed to line and integration methods. • `method`: "stable" (to aggregate integration weights onto mesh nodes) or "direct" (to construct a within triangle/segment integration scheme without aggregating onto mesh nodes) • `nsub1`, `nsub2`: integers controlling the number of internal integration points before aggregation. Points per triangle: `(nsub2+1)^2`. Points per knot segment: `nsub1`
format	character; determines the output format, as either "sf" (default when the sampler is `NULL`) or "sp". When `NULL`, determined by the sampler type.

Value

A `data.frame`, `tibble`, `sf`, or `SpatialPointsDataFrame` of 1D and 2D integration points, including a `weight` column and `.block` column.

Methods (by class)

- `fm_int(list)`: Multi-domain integration
- `fm_int(numeric)`: Discrete double or integer space integration
- `fm_int(character)`: Discrete character space integration
- `fm_int(factor)`: Discrete factor space integration
- `fm_int(SpatRaster)`: `SpatRaster` integration. Not yet implemented.
- `fm_int(fm_lattice_2d)`: `fm_lattice_2d` integration. Not yet implemented.

- `fm_int(fm_mesh_1d)`: `fm_mesh_1d` integration. Supported samplers:
  - `NULL` for integration over the entire domain;
  - A length 2 vector defining an interval;
  - A 2-column matrix with a single interval in each row;
  - A tibble with a named column containing a matrix, and optionally a `weight` column.
- `fm_int(fm_mesh_2d)`: `fm_mesh_2d` integration. Any sampler class with an associated `fm_int_mesh_2d()` method is supported.

Examples

```
# Integration on the interval (2, 3.5) with Simpson's rule
ips <- fm_int(fm_mesh_1d(0:4), samplers = cbind(2, 3.5))
plot(ips$x, ips$weight)

# Create integration points for the two intervals [0,3] and [5,10]

ips <- fm_int(
  fm_mesh_1d(0:10),
  matrix(c(0, 3, 5, 10), nrow = 2, byrow = TRUE)
)
plot(ips$x, ips$weight)

# Convert a 1D mesh into integration points
mesh <- fm_mesh_1d(seq(0, 10, by = 1))
ips <- fm_int(mesh, name = "time")
plot(ips$time, ips$weight)

if (require("ggplot2", quietly = TRUE)) {
  #' Integrate on a 2D mesh with polygon boundary subset
  ips <- fm_int(fmexample$mesh, fmexample$boundary_sf[[1]])
  ggplot() +
    geom_sf(data = fm_as_sfc(fmexample$mesh, multi = TRUE), alpha = 0.5) +
    geom_sf(data = fmexample$boundary_sf[[1]], fill = "red", alpha = 0.5) +
    geom_sf(data = ips, aes(size = weight)) +
    scale_size_area()
}

ips <- fm_int(
  fm_mesh_1d(0:10, boundary = "cyclic"),
  rbind(c(0, 3), c(5, 10))
)
plot(ips$x, ips$weight)
```

`fm_is_within` *Query if points are inside a mesh*

Description

Queries whether each input point is within a mesh or not.

Usage

```
fm_is_within(x, y, ...)

## Default S3 method:
fm_is_within(x, y, ...)
```

Arguments

x	A set of points of a class supported by `fm_evaluator(y, loc = x)`
y	An `inla.mesh`
...	Currently unused

Value

A logical vector

Examples

```
all(fm_is_within(fmexample$loc, fmexample$mesh))
```

---

fm_lattice_2d *Make a lattice object*

Description

Construct a lattice grid for `fm_mesh_2d()`

Usage

```
fm_lattice_2d(...)

## Default S3 method:
fm_lattice_2d(
  x = seq(0, 1, length.out = 2),
  y = seq(0, 1, length.out = 2),
  z = NULL,
  dims = if (is.matrix(x)) {
     dim(x)
 } else {
     c(length(x), length(y))
 },
  units = NULL,
  crs = NULL,
  ...
)
```

Arguments

...	Passed on to submethods
x	vector or grid matrix of x-values
y	vector of grid matrix of y-values
z	if x is a matrix, a grid matrix of z-values
dims	the size of the grid, length 2 vector
units	One of `c("default", "longlat", "longsinlat", "mollweide")` or NULL (equivalent to `"default"`).
crs	An optional `fm_crs`, `sf::st_crs`, or `sp::CRS` object

Value

An `fm_lattice_2d` object with elements

dims integer vector

x x-values

y y-values

loc matrix of `(x, y)` values

segm `fm_segm` object

crs `fm_crs` object or `NULL`

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

`fm_mesh_2d()`

Other object creation and conversion: `fm_as_fm()`, `fm_as_lattice_2d()`, `fm_as_mesh_1d()`, `fm_as_mesh_2d()`, `fm_as_segm()`, `fm_as_sfc()`, `fm_as_tensor()`, `fm_mesh_1d()`, `fm_mesh_2d()`, `fm_segm()`, `fm_simplify()`, `fm_tensor()`

Examples

```
lattice <- fm_lattice_2d(
  seq(0, 1, length.out = 17),
  seq(0, 1, length.out = 10)
)

## Use the lattice "as-is", without refinement:
mesh <- fm_rcdt_2d_inla(lattice = lattice, boundary = lattice$segm)
mesh <- fm_rcdt_2d_inla(lattice = lattice, extend = FALSE)

## Refine the triangulation, with limits on triangle angles and edges:
mesh <- fm_rcdt_2d(
  lattice = lattice,
```

```
  refine = list(max.edge = 0.08),
  extend = FALSE
)

## Add an extension around the lattice, but maintain the lattice edges:
mesh <- fm_rcdt_2d(
  lattice = lattice,
  refine = list(max.edge = 0.08),
  interior = lattice$segm
)

## Only add extension:
mesh <- fm_rcdt_2d(lattice = lattice, refine = list(max.edge = 0.08))
```

`fm_list` *Handle lists of fmesher objects*

Description

Methods for constructing and manipulating `fm_list` objects.

Usage

```
fm_list(x, ..., .class_stub = NULL)

fm_as_list(x, ..., .class_stub = NULL)

## S3 method for class 'fm_list'
c(...)

## S3 method for class 'fm_list'
x[i]
```

Arguments

`x`	`fm_list` object from which to extract element(s)
`...`	Arguments passed to each individual conversion call.
`.class_stub`	character; class stub name of class to convert each list element to. If `NULL`, uses `fm_as_fm` and auto-detects if the resulting list has consistent class, and then adds that to the class list. If non-null, uses `paste0("fm_as_", .class_stub)` for conversion, and verifies that the resulting list has elements consistent with that class.
`i`	indices specifying elements to extract

Value

An `fm_list` object, potentially with `fm_{class_stub}_list` added.

Methods (by generic)

- `c(fm_list)`: The `...` arguments should be coercible to `fm_list` objects.
- `[`: Extract sub-list

Functions

- `fm_list()`: Convert each element of a list, or convert a single non-list object and return in a list
- `fm_as_list()`: Convert each element of a list, or convert a single non-list object and return in a list

Examples

```
fm_as_list(list(fmexample$mesh, fm_segm_join(fmexample$boundary_fm)))
```

---

fm_manifold *Query the mesh manifold type*

Description

Extract a manifold definition string, or a logical for matching manifold type

Usage

```
fm_manifold(x, type = NULL)

fm_manifold_type(x)

fm_manifold_dim(x)
```

Arguments

`x`	An object with `manifold` information, or a character string
`type`	`character`; if `NULL` (the default), returns the manifold definition string. If `character`, returns `TRUE` if the manifold type of `x` matches at least one of the character vector elements.

Value

`fm_manifold()`: Either logical (matching manifold type yes/no), or character (the stored manifold, when `is.null(type)` is `TRUE`)

`fm_manifold_type()`: character or NULL; "M", "R", "S", or "T"

`fm_manifold_dim()`: integer or NULL

Examples

```
fm_manifold(fmexample$mesh)
fm_manifold_type(fmexample$mesh)
fm_manifold_dim(fmexample$mesh)
```

`fm_mesh_1d` *Make a 1D mesh object*

Description

Create a `fm_mesh_1d` object.

Usage

```
fm_mesh_1d(
  loc,
  interval = range(loc),
  boundary = NULL,
  degree = 1,
  free.clamped = FALSE,
  ...
)
```

Arguments

`loc`	B-spline knot locations.
`interval`	Interval domain endpoints.
`boundary`	Boundary condition specification. Valid conditions are `c('neumann', 'dirichlet', 'free', 'cyclic')`. Two separate values can be specified, one applied to each endpoint.
`degree`	The B-spline basis degree. Supported values are 0, 1, and 2.
`free.clamped`	If `TRUE`, for `'free'` boundaries, clamp the basis functions to the interval endpoints.
`...`	Additional options, currently unused.

Value

An `fm_mesh_1d` object

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

Other object creation and conversion: fm_as_fm(), fm_as_lattice_2d(), fm_as_mesh_1d(), fm_as_mesh_2d(), fm_as_segm(), fm_as_sfc(), fm_as_tensor(), fm_lattice_2d(), fm_mesh_2d(), fm_segm(), fm_simplify(), fm_tensor()

Examples

```
if (require("ggplot2")) {
  m <- fm_mesh_1d(c(1, 2, 3, 5, 8, 10),
    boundary = c("neumann", "free"),
    degree = 2
  )
  ggplot() +
    geom_fm(data = m, xlim = c(0.5, 10.5))
}
```

---

fm_mesh_2d *Make a 2D mesh object*

Description

Make a 2D mesh object

Usage

```
fm_mesh_2d(...)

fm_mesh_2d_inla(
  loc = NULL,
  loc.domain = NULL,
  offset = NULL,
  n = NULL,
  boundary = NULL,
  interior = NULL,
  max.edge = NULL,
  min.angle = NULL,
  cutoff = 1e-12,
  max.n.strict = NULL,
  max.n = NULL,
  plot.delay = NULL,
  crs = NULL,
  ...
)
```

Arguments

...	Currently passed on to fm_mesh_2d_inla
loc	Matrix of point locations to be used as initial triangulation nodes. Can alternatively be a sf, sfc, SpatialPoints or SpatialPointsDataFrame object.
loc.domain	Matrix of point locations used to determine the domain extent. Can alternatively be a SpatialPoints or SpatialPointsDataFrame object.
offset	The automatic extension distance. One or two values, for an inner and an optional outer extension. If negative, interpreted as a factor relative to the approximate data diameter (default=-0.10???)
n	The number of initial nodes in the automatic extensions (default=16)
boundary	one or more (as list) of fm_segm() objects, or objects supported by fm_as_segm()
interior	one object supported by fm_as_segm()
max.edge	The largest allowed triangle edge length. One or two values.
min.angle	The smallest allowed triangle angle. One or two values. (Default=21)
cutoff	The minimum allowed distance between points. Point at most as far apart as this are replaced by a single vertex prior to the mesh refinement step.
max.n.strict	The maximum number of vertices allowed, overriding min.angle and max.edge (default=-1, meaning no limit). One or two values, where the second value gives the number of additional vertices allowed for the extension.
max.n	The maximum number of vertices allowed, overriding max.edge only (default=-1, meaning no limit). One or two values, where the second value gives the number of additional vertices allowed for the extension.
plot.delay	If logical TRUE or a negative numeric value, activates displaying the result after each step of the multi-step domain extension algorithm.
crs	An optional fm_crs(), sf::crs or sp::CRS object

Value

An inla.mesh object.

Functions

- fm_mesh_2d_inla(): Legacy method for INLA::inla.mesh.2d() Create a triangle mesh based on initial point locations, specified or automatic boundaries, and mesh quality parameters.

INLA compatibility

For mesh and curve creation, the fm_rcdt_2d_inla(), fm_mesh_2d_inla(), and fm_nonconvex_hull_inla() methods will keep the interface syntax used by INLA::inla.mesh.create(), INLA::inla.mesh.2d(), and INLA::inla.nonconvex.hull() functions, respectively, whereas the fm_rcdt_2d(), fm_mesh_2d(), and fm_nonconvex_hull() interfaces may be different, and potentially change in the future.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

fm_rcdt_2d(), fm_mesh_2d(), fm_delaunay_2d(), fm_nonconvex_hull(), fm_extensions(), fm_refine()

Other object creation and conversion: fm_as_fm(), fm_as_lattice_2d(), fm_as_mesh_1d(), fm_as_mesh_2d(), fm_as_segm(), fm_as_sfc(), fm_as_tensor(), fm_lattice_2d(), fm_mesh_1d(), fm_segm(), fm_simplify(), fm_tensor()

Examples

```
fm_mesh_2d_inla(boundary = fm_extensions(cbind(2, 1), convex = 1, 2))
```

---

fm_nonconvex_hull *Compute an extension of a spatial object*

Description

Constructs a potentially nonconvex extension of a spatial object by performing dilation by convex + concave followed by erosion by concave. This is equivalent to dilation by convex followed by closing (dilation + erosion) by concave.

Usage

```
fm_nonconvex_hull(x, ...)

## S3 method for class 'sfc'
fm_nonconvex_hull(
  x,
  convex = -0.15,
  concave = convex,
  preserveTopology = TRUE,
  dTolerance = NULL,
  crs = fm_crs(x),
  ...
)

fm_extensions(x, convex = -0.15, concave = convex, dTolerance = NULL, ...)

## S3 method for class 'matrix'
fm_nonconvex_hull(x, ...)

## S3 method for class 'sf'
fm_nonconvex_hull(x, ...)

## S3 method for class 'Spatial'
fm_nonconvex_hull(x, ...)
```

```
## S3 method for class 'sfg'
fm_nonconvex_hull(x, ...)
```

Arguments

x	A spatial object
...	Arguments passed on to the fm_nonconvex_hull() sub-methods
convex	numeric vector; How much to extend
concave	numeric vector; The minimum allowed reentrant curvature. Default equal to convex
preserveTopology	logical; argument to sf::st_simplify()
dTolerance	If not zero, controls the dTolerance argument to sf::st_simplify(). The default is pmin(convex, concave) / 40, chosen to give approximately 4 or more subsegments per circular quadrant.
crs	Options crs object for the resulting polygon

Details

Morphological dilation by convex, followed by closing by concave, with minimum concave curvature radius concave. If the dilated set has no gaps of width between

$$2\text{convex}(\sqrt{1 + 2\text{concave}/\text{convex}} - 1)$$

and 2concave, then the minimum convex curvature radius is convex.

The implementation is based on the identity

$$\text{dilation}(a)\&\text{closing}(b) = \text{dilation}(a + b)\&\text{erosion}(b)$$

where all operations are with respect to disks with the specified radii.

When convex, concave, or dTolerance are negative, fm_diameter * abs(...) is used instead.

Differs from sf::st_buffer(x, convex) followed by sf::st_concave_hull() (available from GEOS 3.11) in how the amount of allowed concavity is controlled.

Value

fm_nonconvex_hull() returns an extended object as an sfc polygon object (regardless of the x class).

fm_extensions() returns a list of sfc objects.

Functions

- fm_nonconvex_hull(): Basic nonconvex hull method.
- fm_extensions(): Constructs a potentially nonconvex extension of a spatial object by performing dilation by convex + concave followed by erosion by concave. This is equivalent to dilation by convex followed by closing (dilation + erosion) by concave.

INLA compatibility

For mesh and curve creation, the fm_rcdt_2d_inla(), fm_mesh_2d_inla(), and fm_nonconvex_hull_inla() methods will keep the interface syntax used by INLA::inla.mesh.create(), INLA::inla.mesh.2d(), and INLA::inla.nonconvex.hull() functions, respectively, whereas the fm_rcdt_2d(), fm_mesh_2d(), and fm_nonconvex_hull() interfaces may be different, and potentially change in the future.

References

Gonzalez and Woods (1992), Digital Image Processing

See Also

fm_nonconvex_hull_inla()

Examples

```
inp <- matrix(rnorm(20), 10, 2)
out <- fm_nonconvex_hull(inp, convex = 1)
plot(out)
points(inp, pch = 20)
if (TRUE) {
  inp <- sf::st_as_sf(as.data.frame(matrix(1:6, 3, 2)), coords = 1:2)
  bnd <- fm_extensions(inp, convex = c(0.75, 2))
  plot(fm_mesh_2d(boundary = bnd, max.edge = c(0.25, 1)), asp = 1)
}
```

---

fm_nonconvex_hull_inla

Non-convex hull computation

Description

Legacy method for INLA::inla.nonconvex.hull()

Usage

```
fm_nonconvex_hull_inla(
  x,
  convex = -0.15,
  concave = convex,
  resolution = 40,
  eps = NULL,
  eps_rel = NULL,
  crs = NULL,
  ...
)

fm_nonconvex_hull_inla_basic(
```

```
  x,
  convex = -0.15,
  resolution = 40,
  eps = NULL,
  crs = NULL
)
```

Arguments

x	A spatial object
convex	numeric vector; How much to extend
concave	numeric vector; The minimum allowed reentrant curvature. Default equal to convex
resolution	The internal computation resolution. A warning will be issued when this needs to be increased for higher accuracy, with the required resolution stated.
eps, eps_rel	The polygonal curve simplification tolerances used for simplifying the resulting boundary curve. See fm_simplify_helper() for details.
crs	Options crs object for the resulting polygon
...	Unused.

Details

Requires splancs::nndistF()

Value

fm_nonconvex_hull_inla() returns an fm_segm/inla.mesh.segment object, for compatibility with inla.nonconvex.hull().

Functions

- fm_nonconvex_hull_inla_basic(): Special method for convex = 0.

INLA compatibility

For mesh and curve creation, the fm_rcdt_2d_inla(), fm_mesh_2d_inla(), and fm_nonconvex_hull_inla() methods will keep the interface syntax used by INLA::inla.mesh.create(), INLA::inla.mesh.2d(), and INLA::inla.nonconvex.hull() functions, respectively, whereas the fm_rcdt_2d(), fm_mesh_2d(), and fm_nonconvex_hull() interfaces may be different, and potentially change in the future.

See Also

fm_nonconvex_hull()

Other nonconvex inla legacy support: fm_segm_contour_helper(), fm_simplify_helper()

Examples

```
fm_nonconvex_hull_inla(cbind(0, 0), convex = 1)
```

fm_pixels *Generate lattice points covering a mesh*

Description

Generate terra, sf, or sp lattice locations

Usage

```
fm_pixels(
  mesh,
  dims = c(150, 150),
  xlim = NULL,
  ylim = NULL,
  mask = TRUE,
  format = "sf",
  minimal = TRUE,
  nx = deprecated(),
  ny = deprecated()
)
```

Arguments

mesh	An fm_mesh_2d object
dims	A length 2 integer vector giving the dimensions of the target lattice.
xlim, ylim	Length 2 numeric vectors of x- and y- axis limits. Defaults taken from the range of the mesh or mask; see minimal.
mask	If logical and TRUE, remove pixels that are outside the mesh. If mask is an sf or Spatial object, only return pixels covered by this object.
format	character; "sf", "terra" or "sp"
minimal	logical; if TRUE (default), the default range is determined by the minimum of the ranges of the mesh and mask, otherwise only the mesh.
nx	**[Deprecated]** Number of pixels in x direction, or a numeric vector of x-values
ny	**[Deprecated]** Number of pixels in y direction, or a numeric vector of y-values

Value

sf, SpatRaster, or SpatialPixelsDataFrame covering the mesh or mask.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

Examples

```
if (require("ggplot2", quietly = TRUE)) {
  dims <- c(50, 50)
  pxl <- fm_pixels(
    fmexample$mesh,
    dims = dims,
    mask = fmexample$boundary_sf[[1]],
    minimal = TRUE
  )
  pxl$val <- rnorm(NROW(pxl)) +
    fm_evaluate(fmexample$mesh, pxl, field = 2 * fmexample$mesh$loc[, 1])
  ggplot() +
    geom_tile(
      data = pxl,
      aes(geometry = geometry, fill = val),
      stat = "sf_coordinates"
    ) +
    geom_sf(data = fm_as_sfc(fmexample$mesh), alpha = 0.2)
}


if (require("ggplot2", quietly = TRUE) &&
  require("terra", quietly = TRUE) &&
  require("tidyterra", quietly = TRUE)) {
  pxl <- fm_pixels(fmexample$mesh,
    dims = c(50, 50), mask = fmexample$boundary_sf[[1]],
    format = "terra"
  )
  pxl$val <- rnorm(NROW(pxl) * NCOL(pxl))
  pxl <-
    terra::mask(
      pxl,
      mask = pxl$.mask,
      maskvalues = c(FALSE, NA),
      updatevalue = NA
    )
  ggplot() +
    geom_spatraster(data = pxl, aes(fill = val)) +
    geom_sf(data = fm_as_sfc(fmexample$mesh), alpha = 0.2)
}
```

`fm_raw_basis` *Basis functions for mesh manifolds*

Description

Calculate basis functions on `fm_mesh_1d()` or `fm_mesh_2d()`, without necessarily matching the default function space of the given mesh object.

Usage

```
fm_raw_basis(
  mesh,
  type = "b.spline",
  n = 3,
  degree = 2,
  knot.placement = "uniform.area",
  rot.inv = TRUE,
  boundary = "free",
  free.clamped = TRUE,
  ...
)
```

Arguments

mesh	An fm_mesh_1d() or fm_mesh_2d() object.
type	b.spline (default) for B-spline basis functions, sph.harm for spherical harmonics (available only for meshes on the sphere)
n	For B-splines, the number of basis functions in each direction (for 1d meshes n must be a scalar, and for planar 2d meshes a 2-vector). For spherical harmonics, n is the maximal harmonic order.
degree	Degree of B-spline polynomials. See fm_mesh_1d().
knot.placement	For B-splines on the sphere, controls the latitudinal placements of knots. "uniform.area" (default) gives uniform spacing in sin(latitude), "uniform.latitude" gives uniform spacing in latitudes.
rot.inv	For spherical harmonics on a sphere, rot.inv=TRUE gives the rotationally invariant subset of basis functions.
boundary	Boundary specification, default is free boundaries. See fm_mesh_1d() for more information.
free.clamped	If TRUE and boundary is "free", the boundary basis functions are clamped to 0/1 at the interval boundary by repeating the boundary knots. See fm_mesh_1d() for more information.
...	Unused

Value

A matrix with evaluated basis function

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

fm_mesh_1d(), fm_mesh_2d(), fm_basis()

Examples

```
loc <- rbind(c(0, 0), c(1, 0), c(1, 1), c(0, 1))
mesh <- fm_mesh_2d(loc, max.edge = 0.15)
basis <- fm_raw_basis(mesh, n = c(4, 5))

proj <- fm_evaluator(mesh, dims = c(10, 10))
image(proj$x, proj$y, fm_evaluate(proj, basis[, 7]), asp = 1)

if (interactive() && require("rgl")) {
  plot_rgl(mesh, col = basis[, 7], draw.edges = FALSE, draw.vertices = FALSE)
}
```

---

fm_rcdt_2d *Refined Constrained Delaunay Triangulation*

Description

Computes a refined constrained Delaunay triangulation on R2 or S2.

Usage

```
fm_rcdt_2d(...)

fm_rcdt_2d_inla(
  loc = NULL,
  tv = NULL,
  boundary = NULL,
  interior = NULL,
  extend = (missing(tv) || is.null(tv)),
  refine = FALSE,
  lattice = NULL,
  globe = NULL,
  cutoff = 1e-12,
  quality.spec = NULL,
  crs = NULL,
  ...
)

fm_delaunay_2d(loc, crs = NULL, ...)
```

Arguments

`...`	Currently passed on to `fm_mesh_2d_inla` or converted to `fmesher_rcdt()` options.
`loc`	Input coordinates that should be part of the mesh. Can be a matrix, `sf`, `sfc`, `SpatialPoints`, or other object supported by `fm_unify_coords()`.

`tv` Initial triangulation, as a N-by-3 index vector into `loc`

`boundary, interior`
Objects supported by `fm_as_segm()`. If `boundary` is `numeric`, `fm_nonconvex_hull(loc, convex = boundary)` is used.

`extend` `logical` or `list` specifying whether to extend the data region, with parameters

list("n") the number of edges in the extended boundary (default=16)

list("offset") the extension distance. If negative, interpreted as a factor relative to the approximate data diameter (default=-0.10)

Setting to `FALSE` is only useful in combination `lattice` or `boundary`.

`refine` `logical` or `list` specifying whether to refine the triangulation, with parameters

list("min.angle") the minimum allowed interior angle in any triangle. The algorithm is guaranteed to converge for `min.angle` at most 21 (default=21)

list("max.edge") the maximum allowed edge length in any triangle. If negative, interpreted as a relative factor in an ad hoc formula depending on the data density (default=`Inf`)

list("max.n.strict") the maximum number of vertices allowed, overriding `min.angle` and `max.edge` (default=-1, meaning no limit)

list("max.n") the maximum number of vertices allowed, overriding `max.edge` only (default=-1, meaning no limit)

`lattice` An `fm_lattice_2d` object, generated by `fm_lattice_2d()`, specifying points on a regular lattice.

`globe` If non-NULL, an integer specifying the level of subdivision for global mesh points, used with `fmesher_globe_points()`

`cutoff` The minimum allowed distance between points. Point at most as far apart as this are replaced by a single vertex prior to the mesh refinement step.

`quality.spec` List of vectors of per vertex `max.edge` target specification for each location in `loc`, `boundary/interior` (`segm`), and `lattice`. Only used if refining the mesh.

`crs` Optional crs object

Value

An `fm_mesh_2d` object

Functions

- `fm_rcdt_2d_inla()`: Legacy method for the `INLA::inla.mesh.create()` interface
- `fm_delaunay_2d()`: Construct a plain Delaunay triangulation.

INLA compatibility

For mesh and curve creation, the `fm_rcdt_2d_inla()`, `fm_mesh_2d_inla()`, and `fm_nonconvex_hull_inla()` methods will keep the interface syntax used by `INLA::inla.mesh.create()`, `INLA::inla.mesh.2d()`, and `INLA::inla.nonconvex.hull()` functions, respectively, whereas the `fm_rcdt_2d()`, `fm_mesh_2d()`, and `fm_nonconvex_hull()` interfaces may be different, and potentially change in the future.

Examples

```
(m <- fm_rcdt_2d_inla(
  boundary = fm_nonconvex_hull(cbind(0, 0), convex = 5)
))

fm_delaunay_2d(matrix(rnorm(30), 15, 2))
```

`fm_row_kron` *Row-wise Kronecker products*

Description

Takes two Matrices and computes the row-wise Kronecker product. Optionally applies row-wise weights and/or applies an additional 0/1 row-wise Kronecker matrix product.

Usage

```
fm_row_kron(M1, M2, repl = NULL, n.repl = NULL, weights = NULL)
```

Arguments

`M1`	A matrix that can be transformed into a sparse Matrix.
`M2`	A matrix that can be transformed into a sparse Matrix.
`repl`	An optional index vector. For each entry, specifies which replicate the row belongs to, in the sense used in `INLA::inla.spde.make.A`
`n.repl`	The maximum replicate index, in the sense used in `INLA::inla.spde.make.A()`.
`weights`	Optional scaling weights to be applied row-wise to the resulting matrix.

Value

A `Matrix::sparseMatrix` object.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

Examples

```
fm_row_kron(rbind(c(1, 1, 0), c(0, 1, 1)), rbind(c(1, 2), c(3, 4)))
```

fm_segm *Make a spatial segment object*

Description

Make a spatial segment object

Usage

```
fm_segm(...)

## Default S3 method:
fm_segm(loc = NULL, idx = NULL, grp = NULL, is.bnd = TRUE, crs = NULL, ...)

## S3 method for class 'fm_segm'
fm_segm(..., grp = NULL, grp.default = 0L, is.bnd = NULL)

## S3 method for class 'fm_segm_list'
fm_segm(x, grp = NULL, grp.default = 0L, ...)

fm_segm_join(x, grp = NULL, grp.default = 0L, is.bnd = NULL)

fm_segm_split(x, grp = NULL, grp.default = 0L)

## S3 method for class 'inla.mesh.segment'
fm_segm(..., grp.default = 0)

## S3 method for class 'inla.mesh'
fm_segm(x, ...)

## S3 method for class 'fm_mesh_2d'
fm_segm(x, boundary = TRUE, grp = NULL, ...)

fm_is_bnd(x)

fm_is_bnd(x) <- value
```

Arguments

`...`	Passed on to submethods
`loc`	Matrix of point locations, or `SpatialPoints`, or `sf`/`sfc` point object.
`idx`	Segment index sequence vector or index pair matrix. The indices refer to the rows of `loc`. If `loc==NULL`, the indices will be interpreted as indices into the point specification supplied to `fm_rcdt_2d()`. If `is.bnd==TRUE`, defaults to linking all the points in `loc`, as `c(1:nrow(loc),1L)`, otherwise `1:nrow(loc)`.

grp	When joining segments, use these group labels for segments instead of the original group labels.
is.bnd	TRUE if the segments are boundary segments, otherwise FALSE.
crs	An optional fm_crs(), sf::st_crs() or sp::CRS() object
grp.default	If grp.default is NULL, use these group labels for segments with NULL group.
x	Mesh to extract segments from
boundary	logical; if TRUE, extract the boundary segments, otherwise interior constrain segments.
value	logical

Value

An fm_segm or fm_segm_list object

Methods (by class)

- fm_segm(fm_segm): Join multiple fm_segm objects into a single fm_segm object. If is.bnd is non-NULL, it overrides the input segment information. Otherwise, it checks if the inputs are consistent.
- fm_segm(fm_segm_list): Join fm_segm objects from a fm_segm_list into a single fm_segm object. Equivalent to fm_segm_join(x)
- fm_segm(fm_mesh_2d): Extract the boundary or interior segments of a 2d mesh. If grp is non-NULL, extracts only segments matching the matching the set of groups given by grp.

Functions

- fm_segm(): Create a new fm_segm object.
- fm_segm_join(): Join multiple fm_segm objects into a single fm_segm object. If is.bnd is non-NULL, it overrides the segment information. Otherwise it checks for consistency.
- fm_segm_split(): Split an fm_segm object by grp into an fm_segm_list object, optionally keeping only some groups.

See Also

Other object creation and conversion: fm_as_fm(), fm_as_lattice_2d(), fm_as_mesh_1d(), fm_as_mesh_2d(), fm_as_segm(), fm_as_sfc(), fm_as_tensor(), fm_lattice_2d(), fm_mesh_1d(), fm_mesh_2d(), fm_simplify(), fm_tensor()

Examples

```
fm_segm(rbind(c(0, 0), c(1, 0), c(1, 1), c(0, 1)), is.bnd = FALSE)
fm_segm(rbind(c(0, 0), c(1, 0), c(1, 1), c(0, 1)), is.bnd = TRUE)

fm_segm_join(fmexample$boundary_fm)

fm_segm(fmexample$mesh, boundary = TRUE)
fm_segm(fmexample$mesh, boundary = FALSE)
```

fm_segm_list *Methods for fm_segm lists*

Description

fm_segm lists can be combined into fm_segm_list list objects.

Usage

```
## S3 method for class 'fm_segm'
c(...)

## S3 method for class 'fm_segm_list'
c(...)

## S3 method for class 'fm_segm_list'
x[i]
```

Arguments

...	Objects to be combined.
x	fm_segm_list object from which to extract element(s)
i	indices specifying elements to extract

Value

A fm_segm_list object

Methods (by generic)

- c(fm_segm_list): The ... arguments should be coercible to fm_segm_list objects.
- [: Extract sub-list

Functions

- c(fm_segm): The ... arguments should be fm_segm objects, or coercible with fm_as_segm_list(list(...)).

See Also

fm_as_segm_list()

Examples

```
m <- c(A = fm_segm(1:2), B = fm_segm(3:4))
str(m)
str(m[2])
```

`fm_simplify` *Recursive curve simplification.*

Description

[Experimental] Simplifies polygonal curve segments by joining nearly co-linear segments.

Uses a variation of the binary splitting Ramer-Douglas-Peucker algorithm, with an ellipse of half-width `eps` ellipse instead of a rectangle, motivated by prediction ellipse for Brownian bridge.

Usage

```
fm_simplify(x, eps = NULL, eps_rel = NULL, ...)
```

Arguments

`x`	An `fm_segm()` object.
`eps`	Absolute straightness tolerance. Default `NULL`, no constraint.
`eps_rel`	Relative straightness tolerance. Default `NULL`, no constraint.
`...`	Currently unused.

Details

Variation of Ramer-Douglas-Peucker. Uses width epsilon ellipse instead of rectangle, motivated by prediction ellipse for Brownian bridge.

Value

The simplified `fm_segm()` object.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

References

Ramer, Urs (1972). "An iterative procedure for the polygonal approximation of plane curves". *Computer Graphics and Image Processing*. **1** (3): 244–256. doi:10.1016/S0146664X(72)800170

Douglas, David; Peucker, Thomas (1973). "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". *The Canadian Cartographer*. **10** (2): 112–122. doi:10.3138/FM576770U75U7727

See Also

Other object creation and conversion: `fm_as_fm()`, `fm_as_lattice_2d()`, `fm_as_mesh_1d()`, `fm_as_mesh_2d()`, `fm_as_segm()`, `fm_as_sfc()`, `fm_as_tensor()`, `fm_lattice_2d()`, `fm_mesh_1d()`, `fm_mesh_2d()`, `fm_segm()`, `fm_tensor()`

Examples

```
theta <- seq(0, 2 * pi, length.out = 1000)
(segm <- fm_segm(cbind(cos(theta), sin(theta)),
  idx = seq_along(theta)
))
(segm1 <- fm_simplify(segm, eps_rel = 0.1))
(segm2 <- fm_simplify(segm, eps_rel = 0.2))
plot(segm)
lines(segm1, col = 2)
lines(segm2, col = 3)

(segm <- fm_segm(cbind(theta, sin(theta * 4)),
  idx = seq_along(theta)
))
(segm1 <- fm_simplify(segm, eps_rel = 0.1))
(segm2 <- fm_simplify(segm, eps_rel = 0.2))
plot(segm)
lines(segm1, col = 2)
lines(segm2, col = 3)
```

---

fm_split_lines *Split lines at triangle edges*

Description

Compute intersections between line segments and triangle edges, and filter out segment of length zero.

Usage

```
fm_split_lines(mesh, ...)

## S3 method for class 'fm_mesh_2d'
fm_split_lines(mesh, segm, ...)

## S3 method for class 'inla.mesh'
fm_split_lines(mesh, ...)
```

Arguments

mesh	An fm_mesh_2d or inla.mesh object
...	Unused.
segm	An fm_segm() object with segments to be split

Value

An fm_segm() object with the same crs as the mesh, with an added field origin, that for each new segment gives the originator index into to original segm object for each new line segment.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

Examples

```
mesh <- fm_mesh_2d(
  boundary = fm_segm(rbind(c(0, 0), c(1, 0), c(1, 1), c(0, 1)), is.bnd = TRUE)
)
splitter <- fm_segm(rbind(c(0.8, 0.2), c(0.2, 0.8)))
segm_split <- fm_split_lines(mesh, splitter)

plot(mesh)
lines(splitter)
points(segm_split$loc)
```

---

fm_tensor *Make a tensor product function space*

Description

[Experimental] Tensor product function spaces. The interface and object storage model is experimental and may change.

Usage

```
fm_tensor(x, ...)
```

Arguments

x	list of function space objects, such as fm_mesh_2d().
...	Currently unused

Value

A fm_tensor or fm_tensor_list object. Elements of fm_tensor:

fun_spaces fm_list of function space objects

manifold character; manifold type summary. Regular subset of Rd "Rd", if all function spaces have type "R", torus connected "Td" if all function spaces have type "S", and otherwise "Md" In all cases, d is the sum of the manifold dimensions of the function spaces.

See Also

Other object creation and conversion: fm_as_fm(), fm_as_lattice_2d(), fm_as_mesh_1d(), fm_as_mesh_2d(), fm_as_segm(), fm_as_sfc(), fm_as_tensor(), fm_lattice_2d(), fm_mesh_1d(), fm_mesh_2d(), fm_segm(), fm_simplify()

Examples

```
m <- fm_tensor(list(
  space = fmexample$mesh,
  time = fm_mesh_1d(1:5)
))
m2 <- fm_as_tensor(m)
m3 <- fm_as_tensor_list(list(m, m))
c(fm_dof(m$fun_spaces$space) * fm_dof(m$fun_spaces$time), fm_dof(m))
str(fm_evaluator(m, loc = list(space = cbind(0, 0), time = 2.5)))
str(fm_basis(m, loc = list(space = cbind(0, 0), time = 2.5)))
str(fm_fem(m))
```

---

fm_transform *Object coordinate transformation*

Description

Handle transformation of various inla objects according to coordinate reference systems of `crs` (from `sf::st_crs()`), `fm_crs`, `sp::CRS` or `INLA::inla.CRS` class.

Usage

```
fm_transform(x, crs, ...)

## Default S3 method:
fm_transform(x, crs, ..., crs0 = NULL)

## S3 method for class 'NULL'
fm_transform(x, crs, ...)

## S3 method for class 'matrix'
fm_transform(x, crs, ..., passthrough = FALSE, crs0 = NULL)

## S3 method for class 'sf'
fm_transform(x, crs, ..., passthrough = FALSE)

## S3 method for class 'sfc'
fm_transform(x, crs, ..., passthrough = FALSE)

## S3 method for class 'sfg'
fm_transform(x, crs, ..., passthrough = FALSE)

## S3 method for class 'Spatial'
fm_transform(x, crs, ..., passthrough = FALSE)

## S3 method for class 'fm_mesh_2d'
fm_transform(x, crs = fm_crs(x), ..., passthrough = FALSE, crs0 = fm_crs(x))
```

```
## S3 method for class 'fm_lattice_2d'
fm_transform(x, crs = fm_crs(x), ..., passthrough = FALSE, crs0 = fm_crs(x))

## S3 method for class 'fm_segm'
fm_transform(x, crs = fm_crs(x), ..., passthrough = FALSE, crs0 = fm_crs(x))

## S3 method for class 'fm_list'
fm_transform(x, crs, ...)

## S3 method for class 'inla.mesh'
fm_transform(x, crs = fm_crs(x), ...)

## S3 method for class 'inla.mesh.lattice'
fm_transform(x, crs, ...)

## S3 method for class 'inla.mesh.segment'
fm_transform(x, crs, ...)
```

Arguments

x	The object that should be transformed from it's current CRS to a new CRS
crs	The target crs object
...	Potential additional arguments
crs0	The source crs object for spatial classes without crs information
passthrough	Default is FALSE. Setting to TRUE allows objects with no CRS information to be passed through without transformation. Use with care!

Value

A transformed object, normally of the same class as the input object.

See Also

fm_CRS()

Examples

```
fm_transform(
  rbind(c(0, 0), c(0, 90)),
  crs = fm_crs("sphere"),
  crs0 = fm_crs("longlat_norm")
)
```

`fm_vertices` *Extract vertex locations from an* `fm_mesh_2d`

Description

Extracts the vertices of an `fm_mesh_2d` object.

Usage

```
fm_vertices(x, format = NULL)
```

Arguments

`x`	An `fm_mesh_2d` object.
`format`	character; `"sf"`, `"df"`, `"sp"`

Value

An `sf`, `data.frame`, or `SpatialPointsDataFrame` object, with the vertex coordinates, and a `.vertex` column with the vertex indices.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

`fm_centroids()`

Examples

```
if (require("ggplot2", quietly = TRUE)) {
  vrt <- fm_vertices(fmexample$mesh, format = "sf")
  ggplot() +
    geom_sf(data = fm_as_sfc(fmexample$mesh)) +
    geom_sf(data = vrt, color = "red")
}
```

geom_fm *ggplot2 geomes for fmesher related objects*

Description

[Experimental]

geom_fm is a generic function for generating geomes from various kinds of fmesher objects, e.g. fm_segm and fm_mesh_2d. The function invokes particular methods which depend on the class of the data argument. Requires the ggplot2 package.

Note: geom_fm is not yet a "proper" ggplot2 geom method; the interface may therefore change in the future.

Usage

```
geom_fm(mapping = NULL, data = NULL, ...)

## S3 method for class 'fm_mesh_2d'
geom_fm(
  mapping = NULL,
  data = NULL,
  mapping_int = NULL,
  mapping_bnd = NULL,
  defs_int = NULL,
  defs_bnd = NULL,
  ...,
  crs = NULL
)

## S3 method for class 'fm_segm'
geom_fm(mapping = NULL, data = NULL, ..., crs = NULL)

## S3 method for class 'fm_mesh_1d'
geom_fm(
  mapping = NULL,
  data = NULL,
  ...,
  xlim = NULL,
  basis = TRUE,
  knots = TRUE,
  derivatives = FALSE,
  weights = NULL
)
```

Arguments

mapping	an object for which to generate a geom.

data	an object for which to generate a geom.
...	Arguments passed on to the geom method.
mapping_int	aes for interior constraint edges.
mapping_bnd	aes for boundary edges.
defs_int	additional settings for interior constraint edges.
defs_bnd	additional settings for boundary edges.
crs	Optional crs to transform the object to before plotting.
xlim	numeric 2-vector; specifies the interval for which to compute functions. Default is data$interval
basis	logical; if TRUE (default), show the spline basis functions
knots	logical; if TRUE (default), show the spline knot locations
derivatives	logical; if TRUE (not default), draw first order derivatives instead of function values
weights	numeric vector; if provided, draw weighted basis functions and the resulting weighted sum.

Value

A combination of ggplot2 geoms.

Methods (by class)

- geom_fm(fm_mesh_2d): Converts an fm_mesh_2d() object to sf with fm_as_sfc() and uses geom_sf to visualize the triangles and edges.
- geom_fm(fm_segm): Converts an fm_segm() object to sf with fm_as_sfc() and uses geom_sf to visualize it.
- geom_fm(fm_mesh_1d): Evaluates and plots the basis functions defined by an fm_mesh_1d() object.

Examples

```
ggplot() +
  geom_fm(data = fmexample$mesh)


m <- fm_mesh_2d(
  cbind(10, 20),
  boundary = fm_extensions(cbind(10, 20), c(25, 65)),
  max.edge = c(4, 10),
  crs = fm_crs("+proj=longlat")
)
ggplot() +
  geom_fm(data = m)
ggplot() +
  geom_fm(data = m, crs = fm_crs("epsg:27700"))
```

```
# Compute a mesh vertex based function on a different grid
px <- fm_pixels(fm_transform(m, fm_crs("mollweide_globe")))
px$fun <- fm_evaluate(m,
  loc = px,
  field = sin(m$loc[, 1] / 5) * sin(m$loc[, 2] / 5)
)
ggplot() +
  geom_tile(aes(geometry = geometry, fill = fun),
    data = px,
    stat = "sf_coordinates"
  ) +
  geom_fm(
    data = m, alpha = 0.2, linewidth = 0.05,
    crs = fm_crs("mollweide_globe")
  )



m <- fm_mesh_1d(c(1, 2, 4, 6, 10), boundary = c("n", "d"), degree = 2)
ggplot() +
  geom_fm(data = m, weights = c(4, 2, 4, -1))


m <- fm_mesh_1d(
  c(1, 2, 3, 5, 7),
  boundary = c("dirichlet", "neumann"),
  degree = 2
)
ggplot() +
  geom_fm(data = m)
```

---

plot.fm_mesh_2d *Draw a triangulation mesh object*

Description

Plots an fm_mesh_2d() object using standard graphics.

Usage

```
## S3 method for class 'fm_mesh_2d'
lines(x, ..., add = TRUE)

## S3 method for class 'fm_mesh_2d'
plot(
  x,
  col = "white",
  t.sub = seq_len(nrow(x$graph$tv)),
```

```
  add = FALSE,
  lwd = 1,
  xlim = range(x$loc[, 1]),
  ylim = range(x$loc[, 2]),
  main = NULL,
  size = 1,
  draw.vertices = FALSE,
  vertex.color = "black",
  draw.edges = TRUE,
  edge.color = rgb(0.3, 0.3, 0.3),
  draw.segments = draw.edges,
  rgl = deprecated(),
  visibility = "front",
  asp = 1,
  axes = FALSE,
  xlab = "",
  ylab = "",
  ...
)
```

Arguments

x	An fm_mesh_2d() object.
...	Further graphics parameters, interpreted by the respective plotting systems.
add	If TRUE, adds to the current plot instead of starting a new one.
col	Color specification. A single named color, a vector of scalar values, or a matrix of RGB values. Requires rgl=TRUE.
t.sub	Optional triangle index subset to be drawn.
lwd	Line width for triangle edges.
xlim	X-axis limits.
ylim	Y-axis limits.
main	Deprecated.
size	argument cex for vertex points.
draw.vertices	If TRUE, draw triangle vertices.
vertex.color	Color specification for all vertices.
draw.edges	If TRUE, draw triangle edges.
edge.color	Color specification for all edges.
draw.segments	If TRUE, draw boundary and interior constraint edges more prominently.
rgl	Deprecated
visibility	If "front" only display mesh faces with normal pointing towards the camera.
asp	Aspect ratio for new plots. Default 1.
axes	logical; whether axes should be drawn on the plot. Default FALSE.
xlab, ylab	character; labels for the axes.

Value

None

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

plot.fm_segm(), plot_rgl.fm_mesh_2d()

Examples

```
mesh <- fm_rcdt_2d(globe = 10)
plot(mesh)

mesh <- fm_mesh_2d(cbind(0, 1), offset = c(1, 1.5), max.edge = 0.5)
plot(mesh)
```

---

plot.fm_segm *Draw* fm_segm *objects.*

Description

Draws a fm_segm() object with generic or rgl graphics.

Usage

```
## S3 method for class 'fm_segm'
plot(x, ..., add = FALSE)

## S3 method for class 'fm_segm'
lines(
  x,
  loc = NULL,
  col = NULL,
  colors = c("black", "blue", "red", "green"),
  add = TRUE,
  xlim = NULL,
  ylim = NULL,
  rgl = FALSE,
  asp = 1,
  axes = FALSE,
  xlab = "",
  ylab = "",
  visibility = "front",
  ...
```

```
)

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

## S3 method for class 'fm_segm_list'
lines(x, ...)
```

Arguments

x	An fm_segm() object.
...	Additional parameters, passed on to graphics methods.
add	If TRUE, add to the current plot, otherwise start a new plot.
loc	Point locations to be used if x$loc is NULL.
col	Segment color specification.
colors	Colors to cycle through if col is NULL.
xlim, ylim	X and Y axis limits for a new plot.
rgl	If TRUE, use rgl for plotting.
asp	Aspect ratio for new plots. Default 1.
axes	logical; whether axes should be drawn on the plot. Default FALSE.
xlab, ylab	character; labels for the axes.
visibility	If "front" only display mesh faces with normal pointing towards the camera.

Value

None

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

fm_segm(), plot.fm_mesh_2d

Examples

```
plot(fm_segm(fmexample$mesh, boundary = TRUE))
lines(fm_segm(fmexample$mesh, boundary = FALSE), col = 2)
```

plot_globeproj *Plot a globeproj object*

Description

Plot a globeproj object

Usage

```
plot_globeproj(
  x,
  xlim = NULL,
  ylim = NULL,
  outline = TRUE,
  graticule = c(24, 12),
  tissot = c(12, 6),
  asp = 1,
  add = FALSE,
  ...
)
```

Arguments

x	A globeproj object
xlim, ylim	The x- and y-axis limits
outline	logical
graticule	The number of graticules (n-long, n-lat) to compute
tissot	The number of Tissot indicatrices (n-long, n-lat) to compute
asp	the aspect ratio. Default = 1
add	logical; If TRUE, add to existing plot. Default: FALSE
...	Additional parameters passed on to other methods

Value

Nothing

Author(s)

Finn Lindgren

Examples

```
proj <- old_globeproj("moll", orient = c(0, 0, 45))
plot_globeproj(proj, graticule = c(24, 12), add = FALSE, asp = 1, lty = 2, lwd = 0.5)
```

plot_rgl *Low level triangulation mesh plotting*

Description

Plots a triangulation mesh using `rgl`.

Usage

```
plot_rgl(x, ...)

lines_rgl(x, ..., add = TRUE)

## S3 method for class 'fm_segm'
lines_rgl(
  x,
  loc = NULL,
  col = NULL,
  colors = c("black", "blue", "red", "green"),
  ...,
  add = TRUE
)

## S3 method for class 'fm_mesh_2d'
plot_rgl(
  x,
  col = "white",
  color.axis = NULL,
  color.n = 512,
  color.palette = cm.colors,
  color.truncate = FALSE,
  alpha = NULL,
  lwd = 1,
  specular = "black",
  draw.vertices = TRUE,
  draw.edges = TRUE,
  draw.faces = TRUE,
  draw.segments = draw.edges,
  size = 2,
  edge.color = rgb(0.3, 0.3, 0.3),
  t.sub = seq_len(nrow(x$graph$tv)),
  visibility = "",
  S = deprecated(),
  add = FALSE,
  ...
)
```

```
## S3 method for class 'fm_segm'
plot_rgl(x, ..., add = FALSE)

## S3 method for class 'fm_segm_list'
plot_rgl(x, ...)

## S3 method for class 'fm_segm_list'
lines_rgl(x, ...)
```

Arguments

x	A fm_mesh_2d() object
...	Additional parameters passed to and from other methods.
add	If TRUE, adds to the current plot instead of starting a new one.
loc	Point locations to be used if x$loc is NULL.
col	Segment color specification.
colors	Colors to cycle through if col is NULL.
color.axis	The min/max limit values for the color mapping.
color.n	The number of colors to use in the color palette.
color.palette	A color palette function.
color.truncate	If TRUE, truncate the colors at the color axis limits.
alpha	Transparency/opaqueness values. See rgl.material.
lwd	Line width for edges. See rgl.material.
specular	Specular color. See rgl.material.
draw.vertices	If TRUE, draw triangle vertices.
draw.edges	If TRUE, draw triangle edges.
draw.faces	If TRUE, draw triangles.
draw.segments	If TRUE, draw boundary and interior constraint edges more prominently.
size	Size for vertex points.
edge.color	Edge color specification.
t.sub	Optional triangle index subset to be drawn.
visibility	If "front" only display mesh faces with normal pointing towards the camera.
S	Deprecated.

Value

An rgl device identifier, invisibly.

Author(s)

Finn Lindgren <finn.lindgren@gmail.com>

See Also

plot.fm_mesh_2d()

Examples

```
if (interactive() && require("rgl")) {
  mesh <- fm_rcdt_2d(globe = 10)
  plot_rgl(mesh, col = mesh$loc[, 1])
}
```

Index

∗ **datasets**
fmexample, 11
∗ **fm_as**
fm_as_sfc, 19
∗ **nonconvex inla legacy support**
fm_nonconvex_hull_inla, 67
∗ **object creation and conversion**
fm_as_fm, 12
fm_as_lattice_2d, 14
fm_as_mesh_1d, 15
fm_as_mesh_2d, 16
fm_as_segm, 17
fm_as_sfc, 19
fm_as_tensor, 21
fm_lattice_2d, 58
fm_mesh_1d, 62
fm_mesh_2d, 63
fm_segm, 75
fm_simplify, 78
fm_tensor, 80
[.fm_bbox *(fm_bbox)*, 24
[.fm_list *(fm_list)*, 60
[.fm_segm_list *(fm_segm_list)*, 77
[.fm_segm_list(), *19*
$.fm_crs *(fm_crs)*, 35

c.fm_bbox *(fm_bbox)*, 24
c.fm_bbox(), *26*
c.fm_list *(fm_list)*, 60
c.fm_segm *(fm_segm_list)*, 77
c.fm_segm(), *19*
c.fm_segm_list *(fm_segm_list)*, 77
c.fm_segm_list(), *19*
class, *84*

fm_as_bbox *(fm_bbox)*, 24
fm_as_fm, 12, *14, 15*, *17*, *19–21*, *59*, *63*, *65*, *76*, *78*, *80*
fm_as_inla_mesh *(fmesher-deprecated)*, 3
fm_as_inla_mesh_segment *(fmesher-deprecated)*, 3
fm_as_lattice_2d, *13*, 14, *15*, *17*, *19–21*, *59*, *63*, *65*, *76*, *78*, *80*
fm_as_lattice_2d_list *(fm_as_lattice_2d)*, 14
fm_as_list *(fm_list)*, 60
fm_as_mesh_1d, *13, 14*, 15, *17*, *19–21*, *59*, *63*, *65*, *76*, *78*, *80*
fm_as_mesh_1d_list *(fm_as_mesh_1d)*, 15
fm_as_mesh_2d, *13–15*, 16, *19–21*, *59*, *63*, *65*, *76*, *78*, *80*
fm_as_mesh_2d(), *5*
fm_as_mesh_2d_list *(fm_as_mesh_2d)*, 16
fm_as_segm, *13–15*, *17*, 17, *20, 21*, *59*, *63*, *65*, *76*, *78*, *80*
fm_as_segm(), *5*, *64*, *73*
fm_as_segm_list *(fm_as_segm)*, 17
fm_as_segm_list(), *77*
fm_as_sfc, *13–15*, *17*, *19*, 19, *21*, *59*, *63*, *65*, *76*, *78*, *80*
fm_as_sfc(), *85*
fm_as_sp_crs *(fmesher-deprecated)*, 3
fm_as_tensor, *13–15*, *17*, *19, 20*, 21, *59*, *63*, *65*, *76*, *78*, *80*
fm_as_tensor_list *(fm_as_tensor)*, 21
fm_bary, 22
fm_basis, 23
fm_basis(), *53*, *71*
fm_bbox, 24
fm_block, 27
fm_block(), *53*
fm_block_eval *(fm_block)*, 27
fm_block_log_shift *(fm_block)*, 27
fm_block_log_weights *(fm_block)*, 27
fm_block_logsumexp_eval *(fm_block)*, 27
fm_block_prep *(fm_block)*, 27
fm_block_weights *(fm_block)*, 27
fm_centroids, 30

fm_centroids(), *83*
fm_contains, 31
fm_covariance (fm_gmrf), 53
fm_CRS, 32
fm_crs, 35
fm_CRS(), *5*, *41*, *42*, *82*
fm_crs(), *5*, *34*, *35*, *40–42*, *46*, *64*
fm_CRS.fm_list (fm_crs), 35
fm_crs.fm_list (fm_crs), 35
fm_crs.fm_mesh_2d (fm_crs), 35
fm_crs.fm_segm (fm_crs), 35
fm_crs.inla.CRS (fm_crs), 35
fm_crs.inla.mesh (fm_crs), 35
fm_crs.matrix (fm_crs), 35
fm_crs.sf (fm_crs), 35
fm_crs.sfc (fm_crs), 35
fm_crs.sfg (fm_crs), 35
fm_crs.Spatial (fm_crs), 35
fm_crs.SpatRaster (fm_crs), 35
fm_crs.SpatVector (fm_crs), 35
fm_crs<-, 39
fm_crs_bounds (fm_crs_wkt), 43
fm_crs_detect_manifold (fm_detect_manifold), 46
fm_crs_get_ellipsoid_radius (fm_crs_wkt), 43
fm_crs_get_lengthunit (fm_crs_wkt), 43
fm_crs_get_wkt (fm_crs_wkt), 43
fm_crs_is_geocent (fm_crs_wkt), 43
fm_crs_is_identical, 41
fm_crs_is_identical(), *35*, *42*
fm_crs_is_null, 42
fm_crs_is_null(), *41*
fm_crs_oblique (fm_crs), 35
fm_crs_oblique<- (fm_crs<-), 39
fm_crs_projection_type (fm_crs_wkt), 43
fm_crs_set_ellipsoid_radius (fm_crs_wkt), 43
fm_crs_set_lengthunit (fm_crs_wkt), 43
fm_crs_wkt, *35*, *38*, 43
fm_delaunay_2d (fm_rcdt_2d), 72
fm_delaunay_2d(), *65*
fm_detect_manifold, 46
fm_diameter, 47
fm_dof, 49
fm_ellipsoid_radius (fm_crs_wkt), 43
fm_ellipsoid_radius<- (fm_crs_wkt), 43
fm_evaluate, 49
fm_evaluator (fm_evaluate), 49
fm_evaluator(), *24*
fm_evaluator_lattice (fm_evaluate), 49
fm_extensions (fm_nonconvex_hull), 65
fm_extensions(), *65*
fm_fallback_PROJ6 (fmesher-deprecated), 3
fm_fem, 52
fm_gmrf, 53
fm_has_PROJ6 (fmesher-deprecated), 3
fm_identical_CRS (fm_crs_is_identical), 41
fm_int, 55
fm_int_mesh_2d(), *57*
fm_is_bnd (fm_segm), 75
fm_is_bnd<- (fm_segm), 75
fm_is_within, 57
fm_lattice_2d, *13–15*, *17*, *19–21*, 58, *63*, *65*, *76*, *78*, *80*
fm_lattice_2d(), *50*, *51*, *73*
fm_length_unit (fm_crs_wkt), 43
fm_length_unit<- (fm_crs_wkt), 43
fm_list, 60
fm_manifold, 61
fm_manifold_dim (fm_manifold), 61
fm_manifold_type (fm_manifold), 61
fm_matern_precision (fm_gmrf), 53
fm_matern_sample (fm_gmrf), 53
fm_mesh_1d, *13–15*, *17*, *19–21*, *59*, 62, *65*, *76*, *78*, *80*
fm_mesh_1d(), *49*, *51*, *70*, *71*, *85*
fm_mesh_2d, *13–15*, *17*, *19–21*, *59*, *63*, 63, *76*, *78*, *80*
fm_mesh_2d(), *12*, *30*, *31*, *49*, *51*, *58*, *59*, *64*, *65*, *67*, *68*, *70*, *71*, *73*, *80*, *85–87*
fm_mesh_2d_inla (fm_mesh_2d), 63
fm_mesh_2d_inla(), *64*, *67*, *68*, *73*
fm_nonconvex_hull, 65
fm_nonconvex_hull(), *64–68*, *73*
fm_nonconvex_hull_inla, 67
fm_nonconvex_hull_inla(), *64*, *67*, *68*, *73*
fm_nonconvex_hull_inla_basic (fm_nonconvex_hull_inla), 67
fm_not_for_PROJ4 (fmesher-deprecated), 3
fm_not_for_PROJ6 (fmesher-deprecated), 3
fm_pixels, 69
fm_proj4string (fm_crs_wkt), 43
fm_raw_basis, 70

fm_raw_basis(), *24*
fm_rcdt_2d, 72
fm_rcdt_2d(), *64, 65*, *67, 68*, *73*, *75*
fm_rcdt_2d_inla (fm_rcdt_2d), 72
fm_rcdt_2d_inla(), *64*, *67, 68*, *73*
fm_refine(), *65*
fm_requires_PROJ6 (fmesher-deprecated), 3
fm_row_kron, 74
fm_sample (fm_gmrf), 53
fm_segm, *13–15*, *17*, *19–21*, *59*, *63*, *65*, 75, *78*, *80*
fm_segm(), *64*, *78, 79*, *85*, *88, 89*
fm_segm_contour_helper, *68*
fm_segm_join (fm_segm), 75
fm_segm_list, 77
fm_segm_split (fm_segm), 75
fm_simplify, *13–15*, *17*, *19–21*, *59*, *63*, *65*, *76*, 78, *80*
fm_simplify_helper, *68*
fm_simplify_helper(), *68*
fm_sp2segment (fmesher-deprecated), 3
fm_sp_get_crs (fmesher-deprecated), 3
fm_sp_get_crs(), *35*
fm_split_lines, 79
fm_split_lines(), *10, 11*
fm_spTransform (fmesher-deprecated), 3
fm_tensor, *13–15*, *17*, *19–21*, *59*, *63*, *65*, *76*, *78*, 80
fm_tensor(), *51*
fm_transform, 81
fm_transform(), *5, 6*
fm_unify_coords(), *72*
fm_vertices, 83
fm_vertices(), *30*
fm_wkt (fm_crs_wkt), 43
fm_wkt(), *46*
fm_wkt_as_wkt_tree(), *45*
fm_wkt_get_ellipsoid_radius (fm_crs_wkt), 43
fm_wkt_get_lengthunit (fm_crs_wkt), 43
fm_wkt_is_geocent (fm_crs_wkt), 43
fm_wkt_predef (fm_crs), 35
fm_wkt_projection_type (fm_crs_wkt), 43
fm_wkt_set_ellipsoid_radius (fm_crs_wkt), 43
fm_wkt_set_lengthunit (fm_crs_wkt), 43
fm_wkt_tree_projection_type (fm_crs_wkt), 43
fm_wkt_unit_params (fm_crs_wkt), 43
fmesher-deprecated, 3
fmesher-print, 6
fmesher_bary, 7
fmesher_fem, 8
fmesher_globe_points, 8
fmesher_globe_points(), *73*
fmesher_rcdt, 9
fmesher_rcdt(), *72*
fmesher_split_lines, 10
fmesher_subdivide, 10
fmexample, 11

geom_fm, 84
globeproj, *90*

is.na.fm_CRS (fm_CRS), 32
is.na.fm_crs (fm_crs_is_null), 42
is.na.inla.CRS (fm_CRS), 32

lines.fm_mesh_2d (plot.fm_mesh_2d), 86
lines.fm_segm (plot.fm_segm), 88
lines.fm_segm_list (plot.fm_segm), 88
lines_rgl (plot_rgl), 91

plot.fm_mesh_2d, 86, *89*
plot.fm_mesh_2d(), *93*
plot.fm_segm, 88
plot.fm_segm(), *88*
plot.fm_segm_list (plot.fm_segm), 88
plot_globeproj, 90
plot_rgl, 91
plot_rgl.fm_mesh_2d(), *88*
print.fm_bbox (fmesher-print), 6
print.fm_CRS (fmesher-print), 6
print.fm_crs (fmesher-print), 6
print.fm_mesh_1d (fmesher-print), 6
print.fm_mesh_2d (fmesher-print), 6
print.fm_segm (fmesher-print), 6
print.fm_segm_list (fmesher-print), 6
print.fm_tensor (fmesher-print), 6

sf::st_contains(), *31*
sf::st_crs(), *38*
sp::CRS(), *35*
st_crs.fm_crs (fm_crs), 35