Package 'polyCub'

October 24, 2014

Title Cubature over Polygonal Domains

Version 0.5-1

Date 2014-10-24

Description A package providing methods for cubature (numerical integration)
over polygonal domains. Currently, four cubature methods are implemented:
the two-dimensional midpoint rule as a simple wrapper around as.im.function() from package spatstat (Baddeley and Turner, 2005),the product Gauss cubature by Sommariva and Vianello (2007),an adaptive cubature for isotropic functions via line integrate() along the boundary (Meyer and Held, 2014),and quasi-exact methods specific to the integration of the bivariate Gaussian density over polygonal and circular domains
(based on formulae from the Abramowitz and Stegun (1972) handbook).
For cubature over simple hypercubes, the packages ``cubature'' and
``R2Cuba'' are more appropriate.

License GPL-2

URL https://github.com/WastlM/polyCub

BugReports https://github.com/WastlM/polyCub/issues

Depends R (>= 2.15.0), methods, sp

Imports grDevices, graphics, stats, spatstat

Suggests lattice, testthat, mvtnorm, statmod, rgeos, gpclib

Author Sebastian Meyer [aut, cre, trl],Michael Hoehle [ths]

Maintainer Sebastian Meyer <Sebastian.Meyer@ifspm.uzh.ch>

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-10-24 13:52:14

R topics documented:

`polyCub-package` *Cubature over Polygonal Domains*

Description

The R package **polyCub** provides methods for **cubature** (numerical integration) **over polygonal domains**. The function `polyCub()` is the main entry point of the package. It is a wrapper around the specific cubature methods listed below.

Details

`polyCub.midpoint`**:** Two-dimensional midpoint rule. Polygons are converted to binary pixel images using the `as.im.function` method from package **spatstat** (Baddeley and Turner, 2005). The integral is then obtained as the sum over (pixel area * f(pixel midpoint)).

`polyCub.SV`**:** Product Gauss cubature as proposed by Sommariva and Vianello (2007).

`polyCub.iso`**:** Efficient adaptive cubature for *isotropic* functions via line `integrate()` along the polygon boundary, see Meyer and Held (2014, Supplement B, Section 2.4).

`polyCub.exact.Gauss`**:** Quasi-exact method specific to the integration of the *bivariate Gaussian density* over polygonal domains. It is based on formulae from Chapter 26 of the Abramowitz and Stegun (1972) handbook, i.e. triangulation of the polygonal domain (using `tristrip` of package **gpclib**) and appropriate evaluations of `pmvnorm` from package **mvtnorm**. Note that there is also a function `circleCub.Gauss` to perform integration of the *isotropic* Gaussian density over *circular domains*.

See Section 3.2 of Meyer (2010) for a more detailed description and benchmark experiment of some of the above cubature methods (and others).

References

Abramowitz, M. and Stegun, I. A. (1972). Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. New York: Dover Publications.

Baddeley, A. and Turner, R. (2005). **spatstat**: an R package for analyzing spatial point patterns. *Journal of Statistical Software*, **12** (6), 1-42.

Meyer, S. (2010). Spatio-Temporal Infectious Disease Epidemiology based on Point Processes. Master's Thesis, LMU Munich. Available as http://epub.ub.uni-muenchen.de/11703/.

Meyer, S. and Held, L. (2014). Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639.
DOI-Link: http://dx.doi.org/10.1214/14-AOAS743, arXiv:1308.5115

Sommariva, A. and Vianello, M. (2007). Product Gauss cubature over polygons based on Green's integration formula. *Bit Numerical Mathematics*, **47** (2), 441-453.

See Also

The packages **cubature** and **R2Cuba**, which are more appropriate for cubature over simple hypercubes.

`checkintrfr` *Check the Integral of* $rf\_r(r)$

Description

This function is auxiliary to `polyCub.iso`. The (analytical) integral of $rf_r(r)$ from 0 to R is checked against a numeric approximation using `integrate` for various values of the upper bound R. A warning is issued if inconsistencies are found.

Usage

```
checkintrfr(intrfr, f, ..., center, control = list(), rs = numeric(0L),
  tolerance = control$rel.tol)
```

Arguments

`intrfr`	analytical antiderivative of $rf_r(r)$ from 0 to `R` (first argument, not necessarily named `"R"`, must be vectorized). If missing, `intrfr` is approximated numerically using `integrate` configured with `control`.
`f`	a two-dimensional real function. As its first argument it must take a coordinate matrix, i.e., a numeric matrix with two columns, and it must return a numeric vector of length the number of coordinates.
`...`	further arguments for `f` or `intrfr`.
`center`	numeric vector of length 2, the center of isotropy.
`control`	list of arguments passed to `integrate`, the quadrature rule used for the line integral along the polygon boundary.

rs	numeric vector of upper bounds for which to check the validity of intrfr. If it has length 0, no checks are performed.
tolerance	of all.equal.numeric when comparing intrfr results with numerical integration. Defaults to the relative tolerance used for integrate.

Value

The intrfr function. If it was not supplied, its quadrature version using integrate is returned.

`circleCub.Gauss` *Integration of the Isotropic Gaussian Density over Circular Domains*

Description

This function calculates the integral of the bivariate, isotropic Gaussian density (i.e. $\Sigma =$ `sd^2*diag(2)`) over circular domains via the cumulative distribution function of the (non-central) Chi-Squared distribution (pchisq), cp. Formula 26.3.24 in Abramowitz and Stegun (1972).

Usage

```
circleCub.Gauss(center, r, mean, sd)
```

Arguments

center	numeric vector of length 2 (center of the circle).
r	numeric (radius of the circle). Several radii may be supplied.
mean	numeric vector of length 2 (mean of the bivariate Gaussian density).
sd	numeric (common standard deviation of the isotropic Gaussian density in both dimensions).

Value

The integral value (one for each supplied radius).

Note

The non-centrality parameter of the evaluated chi-squared distribution equals the squared distance between the mean and the center. If this becomes too large, the result becomes inaccurate, see pchisq.

References

Abramowitz, M. and Stegun, I. A. (1972). Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. New York: Dover Publications.

Examples

```
circleCub.Gauss(center=c(1,2), r=3, mean=c(4,5), sd=6)

if (requireNamespace("mvtnorm") && gpclibPermit()) {
  ## compare with cubature over a polygonal approximation of a circle
  disc.poly <- spatstat::disc(radius=3, centre=c(1,2), npoly=32)
  polyCub.exact.Gauss(disc.poly, mean=c(4,5), Sigma=6^2*diag(2))
}
```

`coerce-sp-methods` *Coerce* "Polygons" *to* "owin"

Description

Package **polyCub** also implements `coerce`-methods (`as(object, Class)`) to convert "Polygons" (or "SpatialPolygons" or "Polygon") to "owin".

Author(s)

Sebastian Meyer

`gpclibPermit` **gpclib** *Licence Acceptance*

Description

Similar to the handling in package **maptools**, these functions explicitly accept the restricted **gpclib** licence (commercial use prohibited) and report its acceptance status, respectively. **gpclib** functionality is only required for `polyCub.exact.Gauss`.

Usage

```
gpclibPermit()

gpclibPermitStatus()
```

owin2gpc *Conversion from and to the "gpc.poly" Class*

Description

Package **polyCub** implements converters between the classes "owin" of package **spatstat** and "gpc.poly" of package **rgeos** (originally from **gpclib**). Support for the "gpc.poly" class was dropped from **spatstat** as of version 1.34-0.

Usage

```
owin2gpc(object)

gpc2owin(object, ...)
```

Arguments

object	an object of class "gpc.poly" or "owin", respectively.
...	further arguments passed to owin.

Value

The converted polygon of class "gpc.poly" or "owin", respectively. If neither package **rgeos** nor **gpclib** are available, owin2gpc will just return the pts slot of the "gpc.poly" (no formal class) with a warning.

Note

The converter owin2gpc requires the package **rgeos** (or **gpclib**) for the formal class definition of a "gpc.poly". It will produce vertices ordered according to the **sp** convention, i.e. clockwise for normal boundaries and anticlockwise for holes, where, however, the first vertex is *not* repeated!

Author(s)

Sebastian Meyer

See Also

xylist, and the package **rgeos** for conversions of "gpc.poly" objects from and to **sp**'s "SpatialPolygons" class.

plotpolyf *Plot Polygonal Domain on Image of Bivariate Function*

Description

Produces a combined plot of a polygonal domain and an image of a bivariate function, using either `lattice::levelplot` or `image`.

Usage

```
plotpolyf(polyregion, f, ..., npixel = 100, cuts = 15,
  col = rev(heat.colors(cuts + 1)), lwd = 3, xlim = NULL, ylim = NULL,
  use.lattice = TRUE, print.args = list())
```

Arguments

`polyregion`	a polygonal domain. The following classes are supported: `"owin"`, `"gpc.poly"`, `"SpatialPolygons"`, `"Polygons"`, and `"Polygon"` (for these we have an internal `xylist` method).
`f`	a two-dimensional real function. As its first argument it must take a coordinate matrix, i.e., a numeric matrix with two columns, and it must return a numeric vector of length the number of coordinates.
`...`	further arguments for `f`.
`npixel`	numeric vector of length 1 or 2 setting the number of pixels in each dimension.
`cuts`	number of cut points in the z dimension. The range of function values will be divided into `cuts+1` levels.
`col`	colour vector used for the function levels.
`lwd`	line width of the polygon edges.
`xlim,ylim`	numeric vectors of length 2 setting the axis limits. `NULL` means using the bounding box of `polyregion`.
`use.lattice`	logical indicating if **lattice** graphics (`levelplot`) should be used.
`print.args`	a list of arguments passed to `print.trellis` for plotting the produced `"trellis"` object (given `use.lattice = TRUE`). The latter will be returned without explicit `printing` if `print.args` is not a list.

Author(s)

Sebastian Meyer

Examples

```
### a polygonal domain
data("letterR", package="spatstat")

### f: isotropic exponential decay
fr <- function(r, rate=1) dexp(r, rate=rate)
fcenter <- c(2,3)
f <- function (s, rate=1) fr(sqrt(rowSums(t(t(s)-fcenter)^2)), rate=rate)

### plot
plotpolyf(letterR, f, use.lattice=FALSE)
plotpolyf(letterR, f, use.lattice=TRUE)
```

---

plot_polyregion *Plots a Polygonal Domain (of Various Classes)*

Description

Plots a Polygonal Domain (of Various Classes)

Usage

```
plot_polyregion(polyregion, lwd = 2, add = FALSE)
```

Arguments

polyregion	a polygonal domain. The following classes are supported: "owin", "gpc.poly", "SpatialPolygons", "Polygons", and "Polygon" (for these we have an internal xylist method).
lwd	line width of the polygon edges.
add	logical. Add to existing plot?

---

polyCub *Wrapper Function for the Various Cubature Methods*

Description

Instead of calling one of the specific cubature methods of this package, the wrapper function polyCub may be used together with the method argument.

Usage

```
polyCub(polyregion, f, method = c("SV", "midpoint", "iso", "exact.Gauss"),
  ..., plot = FALSE)
```

Arguments

polyregion	a polygonal integration domain. The supported classes depend on the specific method, however, the "owin" class from package **spatstat** works for all methods, as well should a "gpc.poly" polygon (but see the comments in help("coerce-methods")).
f	two-dimensional function to be integrated. As its first argument the function must take a coordinate matrix, i.e. a numeric matrix with two columns. For the "exact.Gauss" method, f is ignored since it is specific to the bivariate normal density.
method	choose one of the implemented cubature methods (partial argument matching is applied), see help("polyCub-package") for an overview. Defaults to using the product Gauss cubature implemented in polyCub.SV.
...	arguments of f or of the specific method.
plot	logical indicating if an illustrative plot of the numerical integration should be produced.

Value

The approximated integral of f over polyregion.

See Also

Other polyCub.methods: .polyCub.iso, polyCub.iso; polyCub.SV; polyCub.exact.Gauss; polyCub.midpoint

Examples

```
### Short comparison of the various cubature methods

## 2D-function to integrate (here: isotropic zero-mean Gaussian density)
f <- function (s, sigma = 5) exp(-rowSums(s^2)/2/sigma^2) / (2*pi*sigma^2)

## simple polygonal integration domain
disc.owin <- spatstat::disc(radius=5, centre=c(3,2), npoly=8)

## plot image of the function and integration domain
plotpolyf(disc.owin, f, xlim=c(-8,8), ylim=c(-8,8))


### Two-dimensional midpoint rule

testmidpoint <- function (eps, main=paste("2D midpoint rule with eps =",eps))
{
    plotpolyf(disc.owin, f, xlim=c(-8,8), ylim=c(-8,8), use.lattice=FALSE)
    ## add evaluation points to plot
    with(spatstat::as.mask(disc.owin, eps=eps),
         points(expand.grid(xcol, yrow), col=m, pch=20))
    polyCub.midpoint(disc.owin, f, eps=eps)
}
testmidpoint(5)
testmidpoint(3)
```

```
testmidpoint(0.5)
testmidpoint(0.2)


### Product Gauss cubature using an increasing number of nodes

for (nGQ in c(1:5,10,20,60)) {
    cat("nGQ =", sprintf("%2i",nGQ), ": ",
        format(polyCub.SV(disc.owin, f, nGQ=nGQ), digits=16),
        "\n")
}

## 'rotation' affects location of nodes
opar <- par(mfrow=c(1,2))
polyCub.SV(disc.owin, f, nGQ=2, rotation=FALSE, plot=TRUE)
polyCub.SV(disc.owin, f, nGQ=2, rotation=TRUE, plot=TRUE)
par(opar)


### Line integration along the boundary for isotropic functions

polyCub.iso(disc.owin, f, center=c(0,0))   # see ?polyCub.iso


### Quasi-exact cubature of the bivariate Gaussian density
### using gpclib::tristrip and mvtnorm::pmvnorm()

if (requireNamespace("mvtnorm") && gpclibPermit()) {
    print(polyCub.exact.Gauss(disc.owin, mean=c(0,0), Sigma=5^2*diag(2),
                              plot=TRUE), digits=16)
}
```

`polyCub.exact.Gauss` *Quasi-Exact Cubature of the Bivariate Normal Density*

Description

Integration is based on triangulation of the polygonal domain and formulae from the Abramowitz and Stegun (1972) handbook (Section 26.9, Example 9, pp. 956f.). This method is quite cumbersome because the A&S formula is only for triangles where one vertex is the origin (0,0). For each triangle of the `tristrip` we have to check in which of the 6 outer regions of the triangle the origin (0,0) lies and adapt the signs in the formula appropriately. (AOB+BOC-AOC) or (AOB-AOC-BOC) or (AOB+AOC-BOC) or (AOC+BOC-AOB) or .... However, the most time consuming step is the evaluation of `pmvnorm`.

Usage

```
polyCub.exact.Gauss(polyregion, mean = c(0, 0), Sigma = diag(2),
  plot = FALSE)
```

Arguments

polyregion	a "gpc.poly" polygon or something that can be coerced to this class, e.g., an "owin" polygon (converted via owin2gpc and – given **rgeos** is available – "SpatialPolygons" also work.
mean,Sigma	mean and covariance matrix of the bivariate normal density to be integrated.
plot	logical indicating if an illustrative plot of the numerical integration should be produced. Note that the polyregion will be shifted and scaled.

Value

The integral of the bivariate normal density over polyregion. Two attributes are appended to the integral value:

nEval	number of triangles over which the standard bivariate normal density had to be integrated, i.e. number of calls to pmvnorm and pnorm, the former of which being the most time-consuming operation.
error	Approximate absolute integration error steming from the error introduced by the nEval pmvnorm evaluations. For this reason, the cubature method is in fact only quasi-exact (as is the pmvnorm function).

Note

The package **gpclib** (which is required to produce the tristrip, since this is not yet implemented in **rgeos**) has a restricted license (commercial use prohibited). It has to be accepted explicitly via gpclibPermit() prior to using polyCub.exact.Gauss.

References

Abramowitz, M. and Stegun, I. A. (1972). Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. New York: Dover Publications.

See Also

circleCub.Gauss for quasi-exact cubature of the isotropic Gaussian density over a circular domain.

Other polyCub.methods: .polyCub.iso, polyCub.iso; polyCub.SV; polyCub.midpoint; polyCub

Examples

```
# see example(polyCub)
```

polyCub.iso *Cubature of Isotropic Functions over Polygonal Domains*

Description

Conducts numerical integration of a two-dimensional isotropic function $f(x,y) = f_r(||(x,y) - \boldsymbol{\mu}||)$, with μ being the center of isotropy, over a polygonal domain. It internally solves a line integral along the polygon boundary using `integrate` where the integrand requires the antiderivative of $rf_r(r)$), which ideally is analytically available and supplied to the function as argument `intrfr`. The two-dimensional integration problem thereby reduces to an efficient adaptive quadrature in one dimension. See Meyer and Held (2014, Supplement B, Section 2.4) for mathematical details.

`.polyCub.iso` is a "bare-bone" version of `polyCub.iso`.

Usage

```
polyCub.iso(polyregion, f, intrfr, ..., center, control = list(),
  check.intrfr = FALSE, plot = FALSE)

.polyCub.iso(polys, intrfr, ..., center, control = list(),
  .witherror = FALSE)
```

Arguments

`polyregion`	a polygonal domain. The following classes are supported: `"owin"`, `"gpc.poly"`, `"SpatialPolygons"`, `"Polygons"`, and `"Polygon"` (for these we have an internal `xylist` method).
`f`	a two-dimensional real function. As its first argument it must take a coordinate matrix, i.e., a numeric matrix with two columns, and it must return a numeric vector of length the number of coordinates.
`intrfr`	analytical antiderivative of $rf_r(r)$ from 0 to `R` (first argument, not necessarily named `"R"`, must be vectorized). If missing, `intrfr` is approximated numerically using `integrate` configured with `control`.
`...`	further arguments for `f` or `intrfr`.
`center`	numeric vector of length 2, the center of isotropy.
`control`	list of arguments passed to `integrate`, the quadrature rule used for the line integral along the polygon boundary.
`check.intrfr`	logical (or numeric vector) indicating if (for which `r`'s) the supplied `intrfr` function should be checked against a numeric approximation. This check requires `f` to be specified. If `TRUE`, the set of test `r`'s defaults to a `seq` of length 20 from 1 to the maximum absolute x or y coordinate of any edge of the `polyregion`.
`plot`	logical indicating if an image of the function should be plotted together with the polygonal domain, i.e., `plotpolyf(polyregion, f, ...)`.
`polys`	something like `owin$bdry`, but see `xylist`.
`.witherror`	logical indicating if an upperbound for the absolute integration error should be attached as an attribute to the result?

Value

The approximate integral of the isotropic function `f` over `polyregion`.
If the `intrfr` function is provided (which is assumed to be exact), an upperbound for the absolute integration error is appended to the result as attribute `"abs.error"`. It equals the sum of the absolute errors reported by all `integrate` calls (there is one for each edge of `polyregion`).

Author(s)

Sebastian Meyer

The basic mathematical formulation of this efficient integration for radially symmetric functions was ascertained with great support by Emil Hedevang (2013), Dept. of Mathematics, Aarhus University, Denmark.

References

Hedevang, E. (2013). Personal communication at the Summer School on Topics in Space-Time Modeling and Inference (May 2013, Aalborg, Denmark).

Meyer, S. and Held, L. (2014). Power-law models for infectious disease spread. *The Annals of Applied Statistics*, **8** (3), 1612-1639.
DOI-Link: http://dx.doi.org/10.1214/14-AOAS743, arXiv:1308.5115

See Also

Other polyCub.methods: `polyCub.SV`; `polyCub.exact.Gauss`; `polyCub.midpoint`; `polyCub`

Examples

```
## we use the example polygon and f (exponential decay) from
example(plotpolyf)

## numerical approximation of 'intrfr'
(intISOnum <- polyCub.iso(letterR, f, center=fcenter))

## analytical 'intrfr' (recall: f_r(r)=dexp(r), we need int_0^R r*f(r) dr)
intrfr <- function (R, rate=1) pgamma(R, 2, rate) / rate
(intISOana <- polyCub.iso(letterR, intrfr=intrfr, center=fcenter))

stopifnot(all.equal(intISOana, intISOnum, check.attributes=FALSE))


### polygon area: f(r) = 1, f(x,y) = 1, center does not really matter

intrfr.const <- function (R) R^2/2
(area.ISO <- polyCub.iso(letterR, intrfr=intrfr.const, center=c(0,0)))

stopifnot(all.equal(spatstat::area.owin(letterR),
                    area.ISO,
                    check.attributes=FALSE))
## the hole is subtracted correctly
```

`polyCub.midpoint` *Two-Dimensional Midpoint Rule*

Description

The surface is converted to a binary pixel image using the `as.im.function` method from package **spatstat** (Baddeley and Turner, 2005). The integral under the surface is then approximated as the sum over (pixel area * f(pixel midpoint)).

Usage

```
polyCub.midpoint(polyregion, f, ..., eps = NULL, dimyx = NULL,
  plot = FALSE)
```

Arguments

`polyregion`	a polygonal integration domain. It can be any object coercible to the **spatstat** class `"owin"` (via `as.owin`).
`f`	a two-dimensional real function. As its first argument it must take a coordinate matrix, i.e., a numeric matrix with two columns, and it must return a numeric vector of length the number of coordinates.
`...`	further arguments for `f`.
`eps`	width and height of the pixels (squares), see `as.mask`.
`dimyx`	number of subdivisions in each dimension, see `as.mask`.
`plot`	logical indicating if an illustrative plot of the numerical integration should be produced.

Value

The approximated value of the integral of `f` over `polyregion`.

References

Baddeley, A. and Turner, R. (2005). **spatstat**: an R package for analyzing spatial point patterns. *Journal of Statistical Software*, **12** (6), 1-42.

See Also

Other polyCub.methods: `.polyCub.iso`, `polyCub.iso`; `polyCub.SV`; `polyCub.exact.Gauss`; `polyCub`

Examples

```
# see example(polyCub)
```

`polyCub.SV` *Product Gauss Cubature over Polygonal Domains*

Description

Product Gauss cubature over polygons as proposed by Sommariva and Vianello (2007).

Usage

```
polyCub.SV(polyregion, f, ..., nGQ = 20, alpha = NULL, rotation = FALSE,
  engine = "C", plot = FALSE)
```

Arguments

`polyregion`	a polygonal domain. The following classes are supported: `"owin"`, `"gpc.poly"`, `"SpatialPolygons"`, `"Polygons"`, and `"Polygon"` (for these we have an internal `xylist` method).
`f`	a two-dimensional real function (or `NULL` to only compute nodes and weights). As its first argument it must take a coordinate matrix, i.e., a numeric matrix with two columns, and it must return a numeric vector of length the number of coordinates.
`...`	further arguments for `f`.
`nGQ`	degree of the one-dimensional Gauss-Legendre quadrature rule (default: 20) as implemented in function `gauss.quad` of package **statmod**. Nodes and weights up to `nGQ=60` are cached in **polyCub**, for larger degrees **statmod** is required.
`alpha`	base-line of the (rotated) polygon at $x = \alpha$ (see Sommariva and Vianello (2007) for an explication). If `NULL` (default), the midpoint of the x-range of each polygon is chosen if no `rotation` is performed, and otherwise the x-coordinate of the rotated point `"P"` (see `rotation`). If `f` has its maximum value at the origin $(0, 0)$, e.g., the bivariate Gaussian density with zero mean, `alpha = 0` is a reasonable choice.
`rotation`	logical (default: `FALSE`) or a list of points `"P"` and `"Q"` describing the preferred direction. If `TRUE`, the polygon is rotated according to the vertices `"P"` and `"Q"`, which are farthest apart (see Sommariva and Vianello, 2007). For convex polygons, this rotation guarantees that all nodes fall inside the polygon.
`engine`	character string specifying the implementation to use. Up to **polyCub** version 0.4-3, the two-dimensional nodes and weights were computed by R functions and these are still available by setting `engine = "R"`. The new C-implementation is now the default (`engine = "C"`) and requires approximately 30% less computation time. The special setting `engine = "C+reduce"` will discard redundant nodes at (0,0) with zero weight resulting from edges on the base-line $x = \alpha$ or orthogonal to it. This extra cleaning is only worth its cost for computationally intensive functions `f` over polygons which really have some edges on the baseline or parallel to the x-axis. Note that the old R implementation does not have such unset zero nodes and weights.

plot — logical indicating if an illustrative plot of the numerical integration should be produced.

Value

The approximated value of the integral of f over polyregion.
In the case f = NULL, only the computed nodes and weights are returned in a list of length the number of polygons of polyregion, where each component is a list with nodes (a numeric matrix with two columns), weights (a numeric vector of length nrow(nodes)), the rotation angle, and alpha.

Author(s)

Sebastian Meyer
The product Gauss cubature is based on the original MATLAB implementation polygauss by Sommariva and Vianello (2007), which is available under the GNU GPL (>=2) license from http://www.math.unipd.it/~alvise/software.html.

References

Sommariva, A. and Vianello, M. (2007). Product Gauss cubature over polygons based on Green's integration formula. *Bit Numerical Mathematics*, **47** (2), 441-453.

See Also

Other polyCub.methods: .polyCub.iso, polyCub.iso; polyCub.exact.Gauss; polyCub.midpoint; polyCub

Examples

```
# see example(polyCub)
```

---

xylist — *Convert Various Polygon Classes to a Simple List of Vertices*

Description

Different packages concerned with spatial data use different polygon specifications, which sometimes becomes very confusing (see Details below). To be compatible with the various polygon classes, package **polyCub** uses an S3 class "xylist", which represents polygons by their core feature only, a list of lists of vertex coordinates (see the "Value" section below). The generic function xylist can deal with the following polygon classes:

- "owin" from package **spatstat**
- "gpc.poly" from package **rgeos** (or **gpclib**)
- "Polygons" from package **sp** (as well as "Polygon" and "SpatialPolygons")

The (somehow useless) default xylist-method does not perform any transformation but only ensures that the polygons are not closed (first vertex not repeated).

Usage

```
xylist(object, ...)

## S3 method for class 'owin'
xylist(object, ...)

## S3 method for class 'gpc.poly'
xylist(object, ...)

## S3 method for class 'SpatialPolygons'
xylist(object, reverse = TRUE, ...)

## S3 method for class 'Polygons'
xylist(object, reverse = TRUE, ...)

## S3 method for class 'Polygon'
xylist(object, reverse = TRUE, ...)

## Default S3 method:
xylist(object, ...)
```

Arguments

`object`	an object of one of the supported spatial classes.
`...`	(unused) argument of the generic.
`reverse`	logical (`TRUE`) indicating if the vertex order of the **sp** classes should be reversed to get the `xylist`/`owin` convention.

Details

Different packages concerned with spatial data use different polygon specifications with respect to:

- do we repeat the first vertex?
- which direction represents holes?

Package overview:

sp: *Repeat* first vertex at the end (closed), anticlockwise = hole, clockwise = normal boundary

spatstat: do *not repeat* first vertex, anticlockwise = normal boundary, clockwise = hole. This convention is also used in `xylist`.

gpclib: Unfortunately, there seems to be no convention for the specification of polygons of class `"gpc.poly"`.

Value

Applying `xylist` to a polygon object, one gets a simple list, where each component (polygon) is a list of `"x"` and `"y"` coordinates. These represent vertex coordinates following **spatstat**'s `"owin"` convention (anticlockwise order without repeating any vertex). The opposite vertex order can be retained for the **sp**-classes by the non-default use with `reverse=FALSE`.

Author(s)

Sebastian Meyer

Index