Package ‘polyCub’

February 7, 2019

**Title**  Cubature over Polygonal Domains

**Version**  0.7.1

**Date**  2019-02-07

**Description**  Numerical integration of continuously differentiable functions \( f(x,y) \) over simple closed polygonal domains. The following cubature methods are implemented:
- the simple two-dimensional midpoint rule (wrapping 'spatstat' functions),
- adaptive cubature for radially symmetric functions via line integrate() along the polygon boundary (Meyer and Held, 2014, <doi:10.1214/14-AOAS743>, Supplement B),
- and integration of the bivariate Gaussian density based on polygon triangulation.

For simple integration along the axes, the ‘cubature’ package is more appropriate.

**License**  GPL-2

**URL**  https://github.com/bastistician/polyCub

**BugReports**  https://github.com/bastistician/polyCub/issues

**Depends**  R (>= 2.15.0), methods

**Imports**  grDevices, graphics, stats, sp (>= 1.0-11)

**Suggests**  spatstat, lattice, testthat, mvtnorm, statmod, rgeos, gpclib, cubature, knitr, rmarkdown, microbenchmark

**RoxygenNote**  6.1.1

**VignetteBuilder**  knitr, rmarkdown

**NeedsCompilation**  yes

**Author**  Sebastian Meyer [aut, cre, trl]  
  <https://orcid.org/0000-0002-1791-9449>,
  Leonhard Held [ths],
  Michael Hoehle [ths]
polyCub-package

Maintainer  Sebastian Meyer <seb.meyer@fau.de>
Repository  CRAN
Date/Publication  2019-02-07 16:30:02 UTC

R topics documented:

polyCub-package .................................................. 2
checkintrfr .......................................................... 3
circleCub.Gauss .................................................. 4
coeerce-gpc-methods ........................................... 5
coeerce-sp-methods ............................................. 6
gpclibPermit ..................................................... 7
plotpolyf ............................................................ 7
plot_polyregion .................................................. 9
polyCub ............................................................. 9
polyCub.exact.Gauss ............................................ 10
polyCub.iso ........................................................ 12
polyCub.midpoint ............................................... 14
polyCub.SV ........................................................ 16
xystlist ............................................................ 18

Index 21

---

polyCub-package  Cubature over Polygonal Domains

Description

The R package polyCub implements cubature (numerical integration) over polygonal domains. It solves the problem of integrating a continuously differentiable function \( f(x, y) \) over simple closed polygons.

Details

polyCub provides the following cubature methods, which can either be called explicitly or via the generic polyCub function:

polyCub.SV: General-purpose product Gauss cubature (Sommariva and Vianello, 2007)
polyCub midpoint: Simple two-dimensional midpoint rule based on as.im.function from spatstat (Baddeley and Turner, 2005)
polyCub.iso: Adaptive cubature for radially symmetric functions via line integrate() along the polygon boundary (Meyer and Held, 2014, Supplement B, Section 2.4).
polyCub.exact.Gauss: Accurate (but slow) integration of the bivariate Gaussian density based on polygon triangulation (via tristrip from gpclib) and (numerous) evaluations of cumulative densities (via pmvnorm from package mvtnorm). Note that there is also a function circleCub.Gauss to integrate the isotropic Gaussian density over a circular domain.
checkintrfr

A more detailed description and benchmark experiment of the above cubature methods can be found in the vignette("polyCub") and in Meyer (2010, Section 3.2).

References


DOI-Link: https://doi.org/10.1214/14-AOAS743, arXiv:1308.5115


DOI-Link: https://doi.org/10.1007/s10543-007-0131-2

See Also

vignette("polyCub")

For the special case of a rectangular domain along the axes (e.g., a bounding box), the cubature package is more appropriate.

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

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

Arguments

intrfr       a function(R, ...), which implements the (analytical) antiderivative of rf_r(r) from 0 to R. The first argument must be vectorized but not necessarily named R. If intrfr is missing, it will be approximated numerically via integrate(function(r, ...)
    r * f(cbind(x0 + r, y0), ...), 0, R, ..., control=control).
    where c(x0, y0) is the center of isotropy. Note that f will not be checked for isotropy.
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 a circular domain via the cumulative distribution function pchisq of the (non-central) Chi-Squared distribution (Abramowitz and Stegun, 1972, Formula 26.3.24).

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


Examples

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

if (requireNamespace("mvtnorm") && gpclibPermit() && requireNamespace("spatstat")) {
  ## 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-gpc-methods  Conversion between polygonal "owin" and "gpc.poly"

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, ...)

as.owin.gpc.poly(W, ...)

Arguments

object an object of class "gpc.poly" or "owin", respectively.
...
W an object of class "gpc.poly".

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.

Examples
if (gpclibpermit() && require("spatstat")) {
    ## use example polygons from
    example(plotpolyf, ask = FALSE)

    letterR   # a simple "xylist"
    letterR.owin <- owin(poly = letterR)
    letterR.gpc_from_owin <- owin2gpc(letterR.owin)
    letterR.xylist_from_gpc <- xylist(letterR.gpc_from_owin)
    stopifnot(all.equal(letterR.owin, lapply(letterR.xylist_from_gpc, "]", 1:2)))

    letterR.owin_from_gpc <- as.owin(letterR.gpc_from_owin)
    stopifnot(all.equal(letterR.owin, letterR.owin_from_gpc))
}

coerce-sp-methods  Coerce "SpatialPolygons" to "owin"

Description
Package polyCub implements coerce-methods (as(object, Class)) to convert "SpatialPolygons" (or "Polygons" or "Polygon") to "owin". They are also available as as.owin.* functions to support polyCub.midpoint. However, these are no registered S3 methods for as.owin, since package spatstat is optional. Note that the maptools package contains an alternative implementation of coercion from "SpatialPolygons" to "owin" (and reverse), and R will use the S4 coerce-method that was loaded last, and prefer the as.owin.SpatialPolygons S3-method exported from maptools if attached.

Usage
as.owin.SpatialPolygons(W, ...)

as.owin.Polygons(W, ...)

as.owin.Polygon(W, ...)  

Arguments
W an object of class "SpatialPolygons", "Polygons", or "Polygon".
... further arguments passed to owin.
gpclibPermit

Author(s)
Sebastian Meyer

Examples

```r
if (require("spatstat") && require("sp")) {
  diamond <- list(x = c(1,2,1,0), y = c(1,2,3,2)) # anti-clockwise
  diamond.owin <- owin(poly = diamond)
  diamond.sp <- Polygon(lapply(diamond, rev)) # clockwise
  diamond.owin_from_sp <- as(diamond.sp, "owin")
  stopifnot(all.equal(diamond.owin, diamond.owin_from_sp))

  ## similarly works for Polygons and SpatialPolygons
  diamond.Ps <- as(diamond.sp, "Polygons")
  stopifnot(identical(diamond.owin, as.owin(diamond.Ps)))
  diamond.SpPs <- SpatialPolygons(list(diamond.Ps))
  stopifnot(identical(diamond.owin, as.owin(diamond.SpPs)))
}
```

Description

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

Usage

- `gpclibPermit()`
- `gpclibPermitStatus()`

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

```r
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())
```
plotpolyf

Arguments

polyregion a polygonal domain. The following classes are supported: "owin" from package spatstat, "gpc.poly" from rgeos (or gpclib), as well as "SpatialPolygons", "Polygons", and "Polygon" from package sp. (For these classes, polyCub knows how to get an xylist.)
f a two-dimensional real-valued 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 color 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 (a rounded version of spatstat.data::letterR$bdry)
letterR <- list(
  list(x = c(3.9, 3.8, 3.7, 3.5, 3.4, 3.5, 3.7, 3.8, 3.8, 3.7,
          3.7, 3.5, 3.3, 2, 2, 2.7, 2.7, 2.9, 3, 3.3, 3.9),
   y = c(0.7, 1.1, 1.3, 1.7, 1.8, 1.9, 2.1, 2.3, 2.5, 2.8, 3,
          3.2, 3.3, 3.3, 0.7, 0.7, 1.7, 1.7, 1.5, 0.7, 0.6)),
  list(x = c(2.6, 2.6, 3, 3.1, 3.2, 3.1, 3.1, 3),
   y = c(2.2, 2.7, 2.7, 2.6, 2.5, 2.4, 2.3, 2.2))
)

### 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" from package spatstat, "gpc.poly" from rgeos (or gpclib), as well as "SpatialPolygons", "Polygons", and "Polygon" from package sp. (For these classes, polyCub knows how to get an xylist.)

lwd  line width of the polygon edges.

add  logical. Add to existing plot?

polyCub  

Wrapper Function for the Various Cubature Methods

Description

The wrapper function polyCub can be used to call specific cubature methods via its method argument. It calls polyCub.SV by default, which implements general-purpose product Gauss cubature.

Usage

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

Arguments

polyregion  a polygonal domain. The following classes are supported: "owin" from package spatstat, "gpc.poly" from rgeos (or gpclib), as well as "SpatialPolygons", "Polygons", and "Polygon" from package sp. (For these classes, polyCub knows how to get an xylist.)

f  a two-dimensional real-valued function to be integrated over polyregion. 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.

For the "exact.Gauss" method, f is ignored since it is specific to the bivariate normal density.
The bivariate Gaussian density can be integrated based on a triangulation of the (transformed) polygonal domain, using 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

\[
\text{polyCub.exact.Gauss}(\text{polyregion, mean = c(0, 0), Sigma = diag(2), plot = FALSE})
\]

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 transformed (shifted and scaled).
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 stemming 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).

The package `gpclib` is required to produce the `tristrip`, since this is not implemented in `rgeos` (as of version 0.3-25). The restricted license of `gpclib` (commercial use prohibited) has to be accepted explicitly via `gpclibPermit()` prior to using `polycub.exact.Gauss`.

**References**


**See Also**

- `circleCub.Gauss` for quasi-exact cubature of the isotropic Gaussian density over a circular domain.
- Other `polyCub`-methods: `polyCub.SV`, `polyCub.iso`, `polyCub.midpoint`, `polyCub`

**Examples**

```r
## a 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)

## a simple polygon as integration domain
hexagon <- list(
  list(x = c(7.33, 7.33, 3, -1.33, -1.33, 3),
   y = c(-0.5, 4.5, 7, 4.5, -0.5, -3))
)

## quasi-exact integration based on gpclib::tristrip() and mvtnorm::pmvnorm()
if (requireNamespace("mvtnorm") && gpclibPermit()) {
  hexagon.gpc <- new("gpc.poly", pts = lapply(hexagon, c, list(hole = FALSE)))
  plotpolyf(hexagon.gpc, f, xlim = c(-8,8), ylim = c(-8,8))
  print(polycub.exact.Gauss(hexagon.gpc, mean = c(0,0), Sigma = 5^2*diag(2),
    plot = TRUE), digits = 16)
}
```
Description

polyCub.iso numerically integrates a radially symmetric function \( f(x, y) = f_r(||(x, y) - \mu||) \), with \( \mu \) being the center of isotropy, over a polygonal domain. It internally approximates a line integral along the polygon boundary using \texttt{integrate}. The integrand requires the antiderivative of \( r f_r(r) \), which should be supplied as argument \texttt{intrfr} (if it is only required if \texttt{check.intrfr=}TRUE). The two-dimensional integration problem thereby reduces to an efficient adaptive quadrature in one dimension. If \texttt{intrfr} is not available analytically, polyCub.iso can use a numerical approximation (meaning \texttt{integrate} within \texttt{integrate}), but the general-purpose cubature method \texttt{polyCub.SV} might be more efficient in this case. See Meyer and Held (2014, Supplement B, Section 2.4) for mathematical details.

\texttt{.polyCub.iso} is a “bare-bone” version of \texttt{polyCub.iso}.

Usage

\begin{verbatim}
polyCub.iso(polyregion, f, intrfr, ..., center, control = list(),
check.intrfr = FALSE, plot = FALSE)

.polyCub.iso(polys, intrfr, ..., center, control = list(),
.witherror = FALSE)
\end{verbatim}

Arguments

polyregion a polygonal domain. The following classes are supported: "owin" from package \texttt{spatstat}, "gpc.poly" from \texttt{rgeos} (or \texttt{gpclib}), as well as "SpatialPolygons", "Polygons", and "Polygon" from package \texttt{sp}. (For these classes, \texttt{polyCub} knows how to get an \texttt{xylist}.)

f a two-dimensional real-valued 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 a function(R, ...), which implements the (analytical) antiderivative of \( r f_r(r) \) from 0 to R. The first argument must be vectorized but not necessarily named R. If \texttt{intrfr} is missing, it will be approximated numerically via \texttt{integrate(function(r, ...)} \( r * f(\text{cbind}(x0 + r, y0), \ldots), 0, R, \ldots, \text{control} = \text{control} \), where \( c(x0, y0) \) is the center of isotropy. Note that \texttt{f} will not be checked for isotropy.

... further arguments for \texttt{f} or \texttt{intrfr}.

center numeric vector of length 2, the center of isotropy.

control list of arguments passed to \texttt{integrate}, the quadrature rule used for the line integral along the polygon boundary.
polyCub.iso

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.

.witheerror  logical indicating if an upper bound 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 upper bound 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).
DOI-Link: https://doi.org/10.1214/14-AOAS743, arXiv:1308.5115

See Also

system.file("include", "polyCubAPI.h", package = "polyCub") for a full C-implementation of this cubature method (for a single polygon). The corresponding C-routine polyCub_iso can be used by other R packages, notably surveillance, via ‘LinkingTo: polyCub’ (in the ‘DESCRIPTION’) and ‘include <polyCubAPI.h>’ (in suitable ‘/src’ files). Note that the intrfr function must then also be supplied as a C-routine. An example can be found in the package tests.
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' (not recommended)
(intIS0num <- polyCub.iso(letterR, f, center = fcenter))
## 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**

```r
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 a corresponding as.owin-method. Note that this includes polygons of the classes "gpc.poly" and "SpatialPolygons", because polyCub defines methods as.owin.gpc.poly and as.owin.SpatialPolygons, respectively.
- `f` a two-dimensional real-valued 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.
polyCub.midpoint

- `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

### See Also
Other polyCub-methods: `polyCub.SV`, `polyCub.exact.Gauss`, `polyCub.iso`, `polyCub`

### Examples
```r
## a 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)

## a simple polygon as integration domain
hexagon <- list(
  list(x = c(7.33, 7.33, 3, -1.33, -1.33, 3),
       y = c(-0.5, 4.5, 7, 4.5, -0.5, -3))
)

if (require("spatstat")) {
  hexagon.owin <- owin(poly = hexagon)

  show_midpoint <- function(eps)
  {
    plotpolyf(hexagon.owin, f, xlim = c(-8, 8), ylim = c(-8, 8),
              use.lattice = FALSE)
    ## add evaluation points to plot
    with(as.mask(hexagon.owin, eps = eps),
         points(expand.grid(xcol, yrow), col = t(m), pch = 20))
    title(main = paste("2D midpoint rule with eps =", eps))
  }

  ## show nodes (eps = 0.5)
  show_midpoint(0.5)

  ## show pixel image (eps = 0.5)
  polyCub.midpoint(hexagon.owin, f, eps = 0.5, plot = TRUE)

  ## use a decreasing pixel size (increasing number of nodes)
  for (eps in c(0.5, 0.3, 0.1))
    cat(sprintf("eps = %.2f
", eps),
         polyCub.midpoint(hexagon.owin, f, eps = eps))
```
polyCub.Nsv

Product Gauss Cubature over Polygonal Domains

Description

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

Usage

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

Arguments

polyregion a polygonal domain. The following classes are supported: "owin" from package spatstat, "gpc.poly" from rgeos (or gpclib), as well as "SpatialPolygons", "Polygons", and "Polygon" from package sp. (For these classes, polyCub knows how to get an xylist.)

f a two-dimensional real-valued function to be integrated over polyregion (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.

gnengine 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 \( \text{R} \) implementation does not have such unset zero nodes 
and weights.

\textbf{plot} logical indicating if an illustrative plot of the numerical integration should be 
produced.

\textbf{Value} 

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

\textbf{Author(s)} 

Sebastian Meyer 

These R and C implementations of product Gauss cubature are based on the original \texttt{MATLAB} 
implementation \texttt{polygauss} by Sommariva and Vianello (2007), which is available under the GNU 
GPL (>=2) license from \url{http://www.math.unipd.it/~alvise/software.html}.

\textbf{References} 

DOI-Link: \url{https://doi.org/10.1007/s10543-007-0131-2}

\textbf{See Also} 

Other \texttt{polyCub}-methods: \texttt{polyCub.exact.Gauss}, \texttt{polyCub.iso}, \texttt{polyCub.midpoint}, \texttt{polyCub}

\textbf{Examples} 

\begin{verbatim}
## a 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)

## a simple polygon as integration domain
hexagon <- list(
  list(x = c(7.33, 7.33, 3, -1.33, -1.33, 3),
       y = c(-0.5, 4.5, 7, 4.5, -0.5, -3))
)

## image of the function and integration domain
plotpolyf(hexagon, f, xlim = c(-8,8), ylim = c(-8,8))

## use a degree of nGQ = 3 and show the corresponding nodes
polyCub.SV(hexagon, f, nGQ = 3, plot = TRUE)

## extract nodes and weights
\end{verbatim}
### 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)
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

```r
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

*Topic hplot
  plotpolyf, 7
*Topic math
  circleCub.Gauss, 4
  polyCub, 9
  polyCub.exact.Gauss, 10
  polyCub.iso, 12
  polyCub.midpoint, 14
  polyCub.SV, 16
*Topic methods
  coerce-gpc-methods, 5
  coerce-sp-methods, 6
  xylist, 18
*Topic spatial
  circleCub.Gauss, 4
  coerce-gpc-methods, 5
  coerce-sp-methods, 6
  polyCub, 9
  polyCub.exact.Gauss, 10
  polyCub.iso, 12
  polyCub.midpoint, 14
  polyCub.SV, 16
  xylist, 18
  .polyCub.iso (polyCub.iso), 12

  all.equal.numeric, 4
  as.im.function, 2, 14
  as.mask, 14, 15
  as.owin, 6, 14
  as.owin.gpc.poly, 14
  as.owin.gpc.poly (coerce-gpc-methods), 5
  as.owin.Polygon (coerce-sp-methods), 6
  as.owin.Polygons (coerce-sp-methods), 6
  as.owin.SpatialPolygons, 14
  as.owin.SpatialPolygons (coerce-sp-methods), 6

  checkintrfr, 3
  circleCub.Gauss, 2, 4, 11

  coerce, Polygon, owin-method (coerce-sp-methods), 6
  coerce, Polygon, Polygons-method (coerce-sp-methods), 6
  coerce, Polygons, owin-method (coerce-sp-methods), 6
  coerce, SpatialPolygons, owin-method (coerce-sp-methods), 6
  coerce-gpc-methods, 5
  coerce-sp-methods, 6

gauss.quad, 16
 gpc.poly, 5, 8–10, 12, 16, 18
 gpc2owin (coerce-gpc-methods), 5
 gpclibPermit, 7, 11
 gpclibPermitStatus (gpclibPermit), 7

image, 7
 integrate, 2–4, 12, 13

lattice::levelplot, 7
 levelplot, 8

owin, 5, 6, 8, 9, 12, 14, 16, 18
 owin2gpc, 10
 owin2gpc (coerce-gpc-methods), 5

pchisq, 4
 plot_polyregion, 9
 plotpolyf, 7, 13
 pmvnorm, 2, 10, 11
 pnorm, 11
 polyCub, 2, 9, 11, 13, 15, 17
 polyCub-package, 2
 polyCub.exact.Gauss, 2, 7, 10, 10, 13, 15, 17
 polyCub.iso, 2, 3, 10, 11, 12, 15, 17
 polyCub.midpoint, 2, 6, 10, 11, 13, 14, 17
 polyCub.SV, 2, 9–13, 15, 16
 Polygon, 6, 8, 9, 12, 16, 19
 Polygons, 6, 8, 9, 12, 16, 19
 print.trellis, 8
seq, 13
SpatialPolygons, 6, 8, 9, 12, 14, 16, 19
tristrip, 2, 10
xylist, 6, 8, 9, 12, 13, 16, 18