Package 'SphericalCubature'

September 16, 2017

Type Package

Title Numerical Integration over Spheres and Balls in n-Dimensions; Multivariate Polar Coordinates

Version 1.4

Date 2017-09-16

Author John P. Nolan, American University

Maintainer John P. Nolan <jpnolan@american.edu>

Depends R (>= 3.0), cubature, SimplicialCubature, mvmesh, abind

Description Provides several methods to integrate functions over the unit sphere and ball in n-dimensional Euclidean space. Routines for converting to/from multivariate polar/spherical coordinates are also provided.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2017-09-16 19:15:42 UTC

R topics documented:

SphericalCubature-package

Numerical integration over spheres and balls in n-dimensions; multivariate polar/spherical coordinates

Description

Provides functions to integrate a function f(x)=f(x[1],...,x[n]) over the unit sphere and balls in n-dimensional Euclidean space:

$$\int_S f(s)ds \qquad \text{and} \qquad \int_B f(x)dx,$$

where the first integral is over the unit sphere S, an (n-1) dimensional surface, and the second integral is over the unit ball B, an n dimensional solid.

There are three classes of methods:

1. exact methods for polynomials in any dimension (fast)
2. a method due to Stroud for smooth integrands on the sphere (in dimensions n=3,4,...,16) (slower)
3. adaptive methods for integrands with different behavior in different regions (slowest)

Methods 2 and 3 are approximations: like any numerical quadrature algorithm, they may give inaccurate results if the integrand changes abruptly on a small region. This happens even in one dimension, and is more difficult to find and deal with in higher dimensions. (One attempt to handle this difficulty is the 'split' versions of the adaptive methods, functions `adaptIntegrateSpherePolarSplit` and `adaptIntegrateBallPolarSplit`, where one can split the region of integration based on knowledge of the integrand.)

Version 1.1 of this package introduces new methods to integrate over spheres. Earlier versions used only polar coordinate representations of the sphere. Now one can use both polar representaions and triangulations of the sphere. The latter has advantages in some cases: it avoids the problems with polar coordinates giving regions that are sometimes rectangles and sometimes triangles (which occurs at the poles), triangles can be approximately equal area in any dimension, etc. While adding these new routines, names became confusing. Apologies to anyone who has trouble because of this, but it seems better in the long run to explicitly name functions based on their approach. Hence `adaptIntegrateSphere` has been renamed `adaptIntegrateSpherePolar` to indicate that it uses polar coordinates, while the new function `adaptIntegrateSphereTri` uses spherical triangles.

An explicit goal was to get beyond the cases where n=2 or n=3, so some efficiency has been sacrificed. In all the methods, the higher the dimension n, the longer the compute time. For methods 2 and 3, compute times get noticeable when n > 5. One application that motivated this package required the ability to work reliably with integrands that have sharp spikes. That requires some sort of adaptive technique, with the possibility of telling the integration algorithm where the spikes are.

The package also includes functions to convert to/from polar coordinates in higher dimensions.

This is a second attempt to provide methods for integrating over spheres and balls in multiple dimensions. One possible improvement is speed: coding routines in C would give a significant

increase in speed. Another possible extension is to include other multivariate integration methods, e.g. the package R2cuba. This may provide a way to approximate higher dimensional integrals in some cases, if the integrand is well behaved. Vector valued integrands are not supported in most functions (but are in `adaptIntegrateSphereTri`); this could be incorporated in the future.

Please let me know if you find any mistakes. I will try to fix bugs promptly.

Constructive comments for improvements are welcome; actually implementing any suggestions will be dependent on time constraints.

Version history:

- 1.0.0 (2013-05-16) original package
- 1.0.1 (2013-05-24) fix mistake in `adaptIntegrateBallPolarSplit`, fix example in `integratePolynomialSphere`, add more documentation
- 1.0.2 (2013-12-18) add function `adaptIntegrateSphereTri3d` to integrate over spherical triangles in 3-dimensions
- 1.1 (2016-05-14) add function `adaptIntegrateSphereTri` to integrate over hyperspherical triangles in n-dimensions.
- 1.2 (2016-07-23) improve `adaptIntegrateSphereTri` where the integration over octants worked, but integrals over other subdivisions did not. New version works over any subdivision that doesn't cross into different octants (this is checked). Minor changes to documentation were made and more checks on input values were added.
- 1.3 (2017-09-16) Improve changes in version 1.2: remove the restricition on simplices in `adaptIntegrateSphereTri`: the input simplices are analyzed and if a simplex is in more than one orthant, it is automatically subdivided, giving a list of simplices that exactly cover the same part of the sphere and respect each orthant. Fix adaptIntegrateSphericalCubatureTri to correctly pass optional arguments to the integrand function. Change the word "octant" to "orthant" throughout the code to stress that the code works in higher dimensions.

Author(s)

John P. Nolan

Maintainer: John P. Nolan <jpnolan@american.edu>

This research was supported by an agreement with Cornell University, Operations Research & Information Engineering, under contract W911NF-12-1-0385 from the Army Research Development and Engineering Command.

See Also

`integrateSpherePolynomial`, `integrateBallPolynomial`, `integrateSphereStroud11`, `sphereArea`, `ballVolume`, `polar2rect`, `rect2polar`, `adaptIntegrateSpherePolar`, `adaptIntegrateSpherePolarSplit`, `adaptIntegrateSphereTri`, `adaptIntegrateSphereTri3d`, `adaptIntegrateBallPolar`, `adaptIntegrateBallPolarSp`

Examples

```
#  integral should just be the area of sphere in n dimensions
f1 <- function( x ) { return(1.0) }
n <- 3
sphereArea( n )
```

```
integrateSphereStroud11( f1, n )
p <- list(coef=1.0,k=matrix( rep(0L,n), nrow=1,ncol=n))
integrateSpherePolynomial( p )
adaptIntegrateSpherePolar( f1, n )$integral

# test of polynomial integration
f2 <- function( x ) { return(x[1]^2) }
sphereArea(n)/n # exact answer
integrateSphereStroud11( f2, n )
p <- list(coef=1.0,k=matrix( c(2L,rep(0L,n-1)), nrow=1) )
integrateSpherePolynomial( p )
adaptIntegrateSpherePolar( f2, n )$integral
```

`adaptIntegrateSpherePolar` *Adaptive integration over sphere or ball in n-dimensions*

Description

Approximate the integral over the sphere or ball in n-dimensions using polar coordinates. Can also integrate over sectors of the sphere/ball, see details. These functions will be slow, but may be necessary to get accurate answers if the integrand function f(x) is not smooth. If the integrand changes rapidly in certain regions, the basic routines `adaptIntegrateSpherePolar` and codeadaptIntegrateBallPolar will likely miss these abrupt changes and give inaccurate results. For cases where the location of the rapid changes are known, the functions `adaptIntegrateSpherePolarSplit` and codeadaptIntegrateBallPolarSplit allow you to split the region of integration and capture those changes.

Usage

```
adaptIntegrateSpherePolar(f, n, lowerLimit = rep(0, n - 1),
    upperLimit = c(rep(pi, n - 2), 2 * pi), tol = 1e-05, ...)
adaptIntegrateSpherePolarSplit(f, n, xstar, width = 0, lowerLimit = rep(0, n - 1),
    upperLimit = c(rep(pi, n - 2), 2 * pi), tol = 1e-05, ...)

adaptIntegrateBallPolar(f, n, lowerLimit = rep(0, n - 1),
    upperLimit = c(rep(pi, n - 2), 2 * pi), R = c(0, 1), tol = 1e-05, ...)
adaptIntegrateBallPolarSplit(f, n, xstar, width = 0, lowerLimit = rep(0, n - 1),
    upperLimit = c(rep(pi, n - 2), 2 * pi), R = c(0, 1), tol = 1e-05, ...)
```

Arguments

`f`	Integrand function f(x)=f(x[1],...,x[n]).
`n`	dimension of the space. The sphere is an (n-1) dimensional manifold inside n-space, the ball is an n-dimensional solid.
`lowerLimit`	Polar angular coordinates of lower limit

upperLimit	Polar angular coordinates of upper limit
tol	tolerance, the desired accuracy of the result. The functions try to get abs(exact-integral) < tol
...	optional arguments passed to f. If used, these should be specified with a tag, e. g. param1=7
R	a numeric vector of length 2, integration is performed over the region with R[1] < radius < R[2].
xstar	(n x m) matrix whose columns give the directions where the integrand changes quickly, where the region will be subdivided to focus on that region. (The length of a column vector is not used, just it's direction.)
width	width of 'splitting regions', a vector of length m. If it is of length 1, then that value is repeated for each j in 1:m. If width[j]=0, the angular region is split just at the points given by the columns xstar[,j]. If width[j] > 0, then angular region is split at an angle plus and minus width[j].

Details

Approximate the integral of f(x) over (part of) the sphere or ball in n-space. The approach is simplistic: reparameterize the region in polar coordinates. For the sphere, this makes the region of integration a hyper-rectangle in dimension (n-1) in the angle space (here the radius is fixed: R=1). For the ball, the polar representation in terms of angles and radius gives an region of integration that is an n dimensional hyper-rectangle.

The region of integration can be a subset of the sphere/ball by specifying a patch/sector in polar coordinates. To integrate over a subregion, bounds for the polar integration have be specified. For example, in two dimensions, you can integrate over the top half of the circle by specifying lowerLimit=0.0 and upperLimit=pi to `adaptIntegrateSpherePolar`. Likewise for the ball, to integrate over the part of the annulus with inner radius .2 and outer radius .7 that is in the first quadrant, specify lowerLimit=0.0, upperLimit=pi/2, R=c(.2,.7).

Value

For `adaptIntegrateSpherePolar` and `adaptIntegrateBallPolar`, the function returns a list containing several fields. There is always a field

integral	Giving the approximate value of the integral.

The other fields depend on the dimension: when n=2, the other fields are what is returned by the function integrate( ) in base R; when n > 2, the other fields are the fields returned by function `adaptIntegrate( )` in package cubature.

For `adaptIntegrateSpherePolarSplit` and `adaptIntegrateBallPolarSplit`, a single value is returned. (This is because these functions make multiple calls to the adaptive integration routine and the results of each call are not saved.

See Also

`polar2rect`, `rect2polar`

Examples

```
f1 <- function( x ) { return(x[1]^2+3*x[2]+exp(x[3])) }
n <- 3
adaptIntegrateSpherePolar( f1, n )
adaptIntegrateSpherePolarSplit( f1, n, xstar=matrix(c(1,1,1),nrow=3) )
adaptIntegrateBallPolar( f1, n )
adaptIntegrateBallPolarSplit( f1, n, xstar=matrix(c(1,1,1),nrow=3) )

# test of adaptive integration with deliberate splitting
# function f3 has a sharp spike in the direction (1,2),
# elsewhere it has value 1
f3 <- function( x ) {
  x0 <- c(1.0,2.0)/sqrt(5.0)
  dist <- sqrt(sum( (x-x0)^2) )
  y <- 10-5000*dist
  y <- 1 + max(y,0)
  return(y)  }

# no splitting: this straightforward attempt at integration misses
# the spike and sees the integrand as =1 everywhere, so returns the arclengh 2*pi
n <- 2
adaptIntegrateSpherePolar( f3, n )

# deliberate splitting at specified points, but still misses spike
# default width=0 splits the region of integration from [0,2*pi] to [0,a] U [a,2*pi],
# where tan(a)=2/1.
xstar <- matrix( c(1.0,2.0,-1.0,1.0), nrow=2 )
adaptIntegrateSpherePolarSplit( f3, n, xstar=xstar )

# deliberate splitting around specified points, 'smart' choice of width gets the spike
# Here the region of integration is split into [0,a-.01] U [a-.01,a+.01] U [a+.01,2*pi]
adaptIntegrateSpherePolarSplit( f3, n, xstar=xstar, width=c(0.01,0.01) )
```

`adaptIntegrateSphereTri` *Adaptive integration over spherical triangles*

Description

Adaptively integrate a function over a set of spherical triangles. Function `adaptIntegrateSphereTri` uses hyperspherical triangles and works in n-dimensions; it uses function `adaptIntegrateSimplex` in package SimplicialCubature, which is based on code of Alan Genz. `adaptIntegrateSphereTri3d` works only in 3-dimensions and is described in the paper by N. Boal and F-J. Sayas; it is not as sophisticated and is slower than `adaptIntegrateSphereTri`, but can be useful and is self contained.

Usage

```
adaptIntegrateSphereTri( f, S, fDim=1L, maxEvals=20000L, absError=0.0,
    tol=1.0e-5, integRule=3L, partitionInfo=FALSE, ...  )
```

```
adaptIntegrateSphereTri3d( f, S, maxRef=50, relTol=0.001, maxTri=50000, gamma=1.5 )
adaptIntegrateSphereTriI0( f, K )
adaptIntegrateSphereTriI1( f, K )
adaptIntegrateSphereTriSubdivideK( K )
```

Arguments

f	function f defined on the sphere
S	array of hyperspherical triangles, dim(S)=c(n,n,nS). Columns of S should be points on the unit sphere: sum(S[,i,j]^2)=1. Execution will be faster if every simplex S[,,j] is contained within any single orthant. This will happend automatically if function Orthants is used to generate orthants, or if S is a tessellation coming from function UnitSphere in package mvmesh. If one or more simplices interect multiple orthants, the simplices will automatically be subdivided so that each subsimplex is in a single orthant.
fDim	integer dimension of the integrand function
maxEvals	maximum number of evaluations allowed
absError	desired absolute error
tol	desired relative tolerance
integRule	integration rule to use in call to function adsimp
partitionInfo	if TRUE, return the final partition after subdivision
...	optional arguments to function f(x,...)
maxRef	maximum number of refinements allowed
relTol	desired relative tolerance
maxTri	maximum number of triangles allowed while refining
gamma	threshold parameter for when a triangle is subdivided, should be >= 1
K	a single spherical triangle in 3 space, dim(K)=c(3,3)

Details

adaptIntegrateSphereTri3d takes as input a function f defined on the unit sphere in 3-dimensions and a list of spherical triangles K0 and attempts to integrate f over the part of the unit sphere described by K0. The triangles in K0 should individually contained in a hemisphere. The algorithm estimates the integral over each triangle, then estimates error on each triangle, and subdivides triangles with large errors. This is repeated until either the desired accuracy is achieved, or there are too many subdivisions.

Functions adaptIntegrateSphereI0, adaptIntegrateSphereI1, and adaptIntegrateSphereSubdivideK are internal functions that compute respectively I0 for a spherical triangle, I1 for a spherical triangle, and subdivide spherical triangle K into 4 smaller spherical triangles (using midpoints of each side).

adaptIntegrateSphereTri takes as input a function f defined on the unit sphere in n-dimensions and a list of hyperspherical triangles S and attempts to integrate f over (part of) the unit sphere

described by S. It uses the R package SimplicialCubature to evaluate the integrals. The hyperspherical triangles in S should individually be contained in a hemisphere. The algorithm estimates the integral over each triangle, then estimates error on each triangle, and subdivides triangles with large errors. This is repeated until either the desired accuracy is achieved, or there are too many subdivisions. This function is more general than adaptIntegrateSphereTri3d in two ways: (1) it works in dimension n >= 2, and (2) it allows vector integrands f. It also is generaly faster than adaptIntegrateSphereTri. Use the function Orthants to get a rough triangulation of the sphere; see the examples below. For finer resolution triangulation can be obtained from UnitSphere in R package mvmesh. For description of adaptIntegrateSphereTri3d see www.unizar.es/galdeano/actas_pau/PDFVIII/pp61-69.pdf.

Value

A list containing

status	a string describing result, ideally it should be "success", otherwise it is an error/warning message.
integral	approximation to the value of the integral
I0	vector of approximate integral over each triangle in K
numRef	number of refinements
nk	number of triangles in K
K	array of spherical triangles after subdivision, dim(K)=c(3,3,nk)
est.error	estimated error
subsimplices	if partitionInfo=TRUE, this gives an array of subsimplices, see function adsimp for more details.
subsimplicesIntegral	if partitionInfo=TRUE, this array gives estimated values of each component of the integral on each subsimplex, see function adsimp for more details.
subsimplicesAbsError	if partitionInfo=TRUE, this array gives estimated values of the absolute error of each component of the integral on each subsimplex, see function adsimp for more details.
subsimplicesVolume	if partitionInfo=TRUE, vector of m-dim. volumes of subsimplices; this is not d-dim. volume if m < n.

Examples

```
# test of polynomial integration f(s)=s[1]^2
f <- function( s ) { return( s[1]^2 ) }
n <- 3

# integrate over whole sphere
S <- Orthants( n )
a <- adaptIntegrateSphereTri( f, S )
b <- adaptIntegrateSphereTri3d( f, S )
# exact answer, adaptIntegrateSphereTri approximation, adaptIntegrateSphereTri3d approximation
sphereArea(n)/n; a$integral; b$integral
```

```
# integrate over first orthant only
S <- Orthants( n, positive.only=TRUE )
a <- adaptIntegrateSphereTri( f, S )
b <- adaptIntegrateSphereTri3d( f, S )
# exact answer, adaptIntegrateSphereTri approximation, adaptIntegrateSphereTri3d approximation
sphereArea(n)/(8*n); a$integral; b$integral
```

`integrateSpherePolynomial` *Integration of polynomials over sphere or ball.*

Description

Exact integration of polynomial over sphere or ball in n-dimensions.

Usage

```
integrateSpherePolynomial(p, valueOnly = TRUE)
integrateBallPolynomial(p, R = c(0, 1))
```

Arguments

`p`	a list specifying the coefficients and powers of the polynomial. See details below
`valueOnly`	boolean saying whether to return only the value of the integral, or return both the value and a intermediate terms. These intermediate terms are used by integrateBallPolynomial( ).
`R`	inner and outer radius of the annular region: R[1] <= radius <= R[2].

Details

Compute the exact integral over the sphere in n dimensions of a polynomial
p(x[1],...,x[n])=sum (coef[i] * x[1]^k[i,1] * ... * x[n]^k[i,n]),
where the sum is over i=1,...,m. The polynomial is specified as a list p with fields

- coef, an m-vector of doubles
- k, an (m x n) matrix of integers

m and n are given implicitly in the sizes of these arrays output is normally just a number, the value of the integral. If integrateSpherePolynomial is called with valueOnly=FALSE, a list with two fields:

- integral, a double containing the value of the integral
- term, a vector of length m of values used in function IntegratePolynomialBall( )

Value

integrateSpherePolynomial( ) normally just returns a value of the integral, but if valueOnly=FALSE, it will return a list containing the value and intermediate terms. These intermediate terms correspond to the integral of each monomial term in the polynomial; they are used by integrateBallPolynomial( ).

integrateBallPolynomial( ) returns just the value of the integral.

References

Method is from How to Integrate a Polynomial over a Sphere, by G. Folland (2001), MAA Monthly 108, pg. 446-448.

Examples

```
n <- 3
# specify the polynomial p(x) = 1.0 * x[1]^2 * x[2]^0 * x[3]^0 + 7.0 * x[1]^0 * x[2]^3 * x[3]
p <- list(coef=c(1.0,7.0),k=matrix( c(2L,0L,0L,0L,3L,0L), byrow=TRUE, nrow=2) )
integrateSpherePolynomial( p )
integrateBallPolynomial( p )

# compare to adaptive integration
f4 <- function( x ) { return( x[1]^2 + 7*x[2]^2*x[3] ) }
adaptIntegrateSpherePolar( f4, n )$integral
adaptIntegrateBallPolar( f4, n )$integral
```

`integrateSphereStroud11`

Integrate a function over the sphere in n-dimensions.

Description

Approximate the integral of a function f(x)=f(x[1],...,x[n]) over the unit sphere in n-space using Stroud's method of degree 11.

Usage

```
integrateSphereStroud11(f, n, ...)
```

Arguments

`f`	function f(x)=f(x[1],...,x[n]) to integrate
`n`	dimension of the space, implemented for n in the range 3:16.
`...`	optional arguments passed to f( ). If these are specified, they should be labeled with a tag, e.g. param1=3.4

Details

This method works if the integrand is smooth. If the function changes rapidly, adaptive integration can be tried as described in 'See Also' below.

Value

A single number, the approximation to the integral.

References

Stroud integration and related functions, adapted from fortran code by John Burkhart found at http://people.sc.fsu.edu/~jburkardt/f77_src/stroud/stroud.html
Based on the book by A. H. Stroud, Approximate Calculation of multiple integrals, 1971, page 296-297.

See Also

adaptIntegrateSpherePolar, adaptIntegrateBallPolar, adaptIntegrateSphereTri

Examples

```
f2 <- function( x ) { return(x[1]^2) }
integrateSphereStroud11( f2, n=3 )
```

---

rect2polar *n-dimensional polar coordinate transformations*

Description

Convert between polar and rectangular coordinates in n-dimensions. The point (x[1],...,x[n]) in rectangular coordinates corresponds to the point (r,phi[1],...,phi[n-1]) in polar coordinates.

Usage

```
polar2rect(r, phi)
rect2polar(x)
```

Arguments

r	a vector of radii of length m.
phi	a (n-1) x m matrix of angles.
x	(n x m) matrix, with column j being the point in n-dimensional space.

Details

n dimensional polar coordinates are given by the following:
rectangular x=(x[1],...,x[n]) corresponds to polar (r,phi[1],...,phi[n-1]) by
x[1] = r*cos(phi[1])
x[2] = r*sin(phi[1])*cos(phi[2])
x[3] = r*sin(phi[1])*sin(phi[2])*cos
...
x[n-1]= r*sin(phi[1])*sin(phi[2])*...*sin(phi[n-2])*cos(phi[n-1])
x[n] = r*sin(phi[1])*sin(phi[2])*...*sin(phi[n-2])*sin(phi[n-1])

Here phi[1],...,phi[n-2] in [0,pi), and phi[n-1] in [0,2*pi). For multivariate integration, the Jacobian of the above tranformation is J(phi) = r^(n-1) * prod(sin(phi[1:(n-2)])^((n-2):1)); note that phi[n-1] does not appear in the Jacobian.

Value

For polar2rect( ), an (n x m) matrix of rectangular coordinates.

For rect2polar( ), a list with fields:

r	a vector of length m containing the radii
phi	an (n x m) matrix of angles

Examples

```
x <- matrix( 1:9, nrow=3 )
x
a <- rect2polar( x )
a
polar2rect( a$r, a$phi )
```

---

sphereArea *Surface area of spheres, volumes of balls in n-dimensions.*

Description

Calculates the (n-1) dimensional surface area of a sphere and the n dimensional volume of a ball in n-space.

Usage

```
sphereArea(n, R = 1)
ballVolume(n, R = 1)
```

Arguments

n	Dimension of the space.
R	Radius of the sphere/ball.

Value

Single number that is the area of the sphere/volume of the ball.

Examples

```
sphereArea(n=5)
ballVolume(n=5)
```

SphericalMisc *Miscellaneous internal functions used by SphericalCubature package.*

Description

These functions are not intended for general use, they are only listed here to document their existence.

adaptIntegrateCheck is used by the adaptive integration functions to check input parameters, partitionRegion is used by the 'split' versions of the the adaptive integration functions, nextGraySubset is used by IntegrateSphereStroud11, nextMultiIndex is used by adaptive integration functions.

Usage

```
adaptIntegrateSphereCheck( n, lowerLimit, upperLimit, R, xstar, width )
partitionRegion( xstar, width, lowerLimit, upperLimit )
nextGraySubset( gray.list )
nextMultiIndex( j, size )
CheckUnitVectors( S, eps=1.0e-14 )
Orthants( n, positive.only=FALSE )
SubdivideSphereTriByOrthant( S, eps=1.0e-14 )
```

Arguments

n	dimension of the space
lowerLimit	lower angular limit for integration region
upperLimit	upper angular limit for integration region
R	inner and outer radii for integration region
xstar	directions where function changes rapidly
width	width of subdivisions
gray.list	list used by Stroud integration
j	current multi-index
size	length of multi-index
S	a matrix or array specifying simplices
eps	tolerance used in test for unit vectors
positive.only	logical; if TRUE, only the first orthant is returned, if FALSE, all 2^n orthants are returned

Details

`Orthant` returns an array of simplices in the V representation; each one gives the points on the axes that are on the boundary of one orthant. `CheckSphereTri` performs validity checks on the simplices in S.

Index