Package ‘SimplicialCubature’

May 16, 2016

Type Package
Title Integration of Functions Over Simplices
Version 1.2
Date 2016-05-14
Author John P. Nolan, with parts adapted from Fortran and matlab code by Alan Genz
Maintainer John P. Nolan <jpnolan@american.edu>
Depends R (>= 3.0)
Description Provides methods to integrate functions over m-dimensional simplices in n-dimensional Euclidean space. There are exact methods for polynomials and adaptive methods for integrating an arbitrary function. Dirichlet probabilities are calculated in certain cases.
License GPL (>= 2)
NeedsCompilation no
Repository CRAN
Date/Publication 2016-05-16 15:41:03

R topics documented:

SimplicialCubature-package ............................................. 2
adaptIntegrateSimplex ..................................................... 3
adsimp ................................................................. 5
CanonicalSimplex ........................................................... 7
definePoly ............................................................... 8
grnmol ................................................................. 9
integrateSimplexPolynomial ........................................... 10
LasserreAvrachenkov .................................................... 11

Index 13
SimplicialCubature-package

Numerical integration of functions over simplices

Description

Provides methods to evaluate integrals of the form

$$\int_S f(x) dx,$$

where S is a simplex (or a union of simplices) in n-space and f(x) is a function defined on S. The function f(x) may be vector valued and the simplices can be m-dimensional simplices, 1 <= m <= n. For example, if m=n-1, the package will evaluate a surface area integral; if m=1, the package will evaluate a line integral.

There are exact methods for polynomials and adaptive methods for integrating an arbitrary function. The two main functions are:

- `adaptIntegrateSimplex` - integrate a general (possibly vector valued) function over a simplex using the method of Genz and Cools.
- `integrateSimplexPolynomial` - integrate a single polynomial exactly over a simplex using either the Grundmann-Moller method or the Lasserre-Avrachenkov method.

The naming of the functions, arguments, and return values deliberately mimics that in the CRAN packages `cubature` (for integrating over hyper-rectangles) and `sphericalcubature` (for integrating over spheres and balls).

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:

- 1.0.1 original package
- 1.1 fix an incorrect test for dimension=1 in SimplexVolume; add functions SimplexSurfaceArea.
  Rename function UnitSimplex to UnitSimplexV to eliminate name conflict with function in package mvmesh.
- 1.2 miscellaneous small changes.

Author(s)

John P. Nolan, with R translations of adsimp code by Alan Genz

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.
adaptIntegrateSimplex

References

V. Baldoni, N. Berline, J. A. De Loera, M. Koppe, and M. Vergene, How to integrate a polynomial over a simplex, Mathematics of Computation, 80, 297-325 (2011)


A. Grundmann and H.M. Moller, Invariant Integration Formulas for the n-Simplex by Combinatorial Methods, SIAM Journal on Numerical Analysis, 15, 282-289 (1978)


See Also

adaptIntegrateSimplex, integrateSimplexPolynomial

adaptIntegrateSimplex  Integrate a general function over a simplex

Description

Adaptive integration of a function $f(x)$ of $n$ variables over an $m$-dimensional simplex $S$, $1 \leq m \leq n$. More generally, $f$ can be a vector valued function and $S$ can be a list of simplices.

Usage

adaptIntegrateSimplex(f, S, fDim = 1L, maxEvals = 10000L, absError = 0, tol = 1e-05,
integRule = 3L, partitionInfo = FALSE, ...)
integrate.vector.fn( intervals, fDim, f, maxEvals, absError, tol, partitionInfo=FALSE )
original.coordinates( u, S )

Arguments

f  a function of $n$-variables (where $n$ is determined by $S$) or a vector valued function (if $fDim > 1$).
S  a simplex or list of simplices that specify the region of integration. A single simplex $S$ is given by an $n \times (m+1)$ matrix, where $n$ is the dimension of the underlying space and $m$ is the dimension of the simplex, $1 \leq m \leq n$. In this case, the columns $S[.1]...,S[.m+1]$ are the vertices of the $m$-dimensional simplex. If $S$ is an $n \times (m+1) \times k$ array, then the region of integration is the union of the simplices $S[.,1]...,S[.,k]$, each of the above form.
fDim  integer dimension of the integrand function.
maxEvals  integer maximum number of function evaluations allowed
absError  requested absolute error in the computation of the integral
tol requested relative error in the computation of the integral
integRule integer in the range 1:4 specifying degree of integration rule: a (2*integRule+1) degree integration rule is used in function adsimp.
partitionInfo if FALSE, then only the results of the computations are returned. If TRUE, then partition information is also returned for the final subdivision of the region. This will require more memory, but sometimes that information can be useful for other purposes.
intervals optional arguments to integrand function f(x,...)
u (1 x 2 x k) array of intervals for univariate integration

Details

If m=n, then an R translation of Alan Genz’s function adsimp(...) is used to evaluate the n-dimensional integral. It works by adaptively splitting the region of integration into finer partitions, always splitting the simplex with the largest estimated error.

If 1 < m < n, then the integral is evaluated by mapping the m-simplex in R^n to the canonical simplex in m-dimensional space, using function adsimp on that ‘full’ m-dimensional integral, and correcting with the Jacobian of the transformation.

If m=1, the built-in R function integrate (based on QUADPACK 1-dimensional adaptive quadrature) is used to evaluate the line integral. Since it does not provide access to the final subdivision, partitionInfo=TRUE in the univariate case returns the original partition information. So, if a fine partition is desired in the m=1 case, start with a fine partition.

Value

A list containing:

integral estimated value of the integral, it is a vector if fDim > 1
estAbsError estimated absolute error
functionEvaluations count of number of function evaluations
returnCode integer status: returnCode=0 is a successful return; non-zero error values are described by next variable
message text message explaining returnCode; "OK" for normal return
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.
Note

No check is done on the simplices to see that they are disjoint.

When m > 1 and fDim > 1, adsimp uses the same grid for each coordinate of f.

When m=1, adsimp does not handle the line integral case, and the built-in R function integrate is used to evaluate the integral. In this 1-dimensional case, no partition information is available (integrate does not provide that information) and if fDim > 1, the components of the integral are evaluated independently, with an upper limit of maxEvals function evaluations for each component. This means that (a) a different grid may be used for each component, and (b) the return variable functionEvaluations is the sum of the number of function evaluations for each component; it may be up to maxEvals*fDim.

References


Examples

```
n <- 4
S <- CanonicalSimplex( n )
f1 <- function( x ) { x[1]^3 }
adaptIntegrateSimplex( f1, S ) # correct answer 0.00119047619
str( adaptIntegrateSimplex( f1, S, partitionInfo=TRUE ) ) # same result, with more info returned

# test with vector valued integrand
f2 <- function( x ) { c(x[1]^3,x[3]^4) }
adaptIntegrateSimplex( f2, S, fDim=2 ) # correct answer 0.00119047619 0.005952380952

# test with vector valued integrand and extra arguments
f3 <- function( x, extra.arg ) { extra.arg*c(x[1]^3,x[3]^4) } # multiple of f2 above
adaptIntegrateSimplex( f3, S, fDim=2, extra.arg=100 ) # correct answer 0.119047619 0.05952380952

# integrate over lower dimensional simplices
adaptIntegrateSimplex( f1, UnitSimplexV(4) ) # answer = 0.01666667

f4 <- function(x) { 1 }
# 2-dim integral, exact answer area of unit simplex = sqrt(3)/2 = 0.8660254...
adaptIntegrateSimplex( f4, UnitSimplexV(3) )

# line integral over diamond in 2-dim, exact answer=arclength*4*sqrt(2)=5.656854...
S4 <- array( c(1,0,0,1, 0,1, -1,0, -1,0, 0,-1, 0,-1, 1,0), dim=c(2,2,4) )
adaptIntegrateSimplex( f4, S4 )
adaptIntegrateSimplex( f4, S4, partitionInfo=TRUE )
```

adsimp

Internal functions for adaptIntegrateSimplex.
Description

adsimp is a translation of Alan Genz’s matlab program adsimp.m to adaptively integrate over a simplex. The other functions listed below are all called by adsimp. These functions are used internally; use at your own risk.

Usage

```plaintext
adsimp(ND, VRTS, NF, F, MXFS, EA, ER, KEY, partitionInfo = FALSE )
adsimp.return.message( rcode )
SMPCHC(ND, NF, MXFS, EA, ER, SBS, KEY)
SMPDFS(ND, NF, F, TOP, SBS, VRTS)
SMPRMS(N, KEY)
SMPRUL(ND, VRTS, VOL, NF, F, G, W, PTS)
SMPSAD(ND, NF, F, MXFS, EA, ER, KEY, RCLS, SBS, VRTS, partitionInfo )
SMPSMS(N, VERTEX, NF, F, G)
```

Arguments

- **ND, N** dimension of the space
- **VRTX, VERTEX** array specifying the simplices
- **NF** dimension of the function; F(x) has NF coordinates
- **F** a function of ND variables, value F(x) has NF coordinates
- **rcode** an integer return code from adsimp
- **MXFS** maximum number of function evaluations allowed
- **EA** requested absolute error
- **ER** requested relative error
- **KEY** integration rule
- **partitionInfo** TRUE or FALSE, controls whether or not the function returns information about the final partition. See the comments in function adaptIntegrateSimplex().
- **SBS** number of subsimplices
- **TOP** pointer to a subsimplex
- **VOL** volume of a simplex
- **G** generators for integration rule
- **W** weights for an integration rule
- **PTS** points in an integration rule
- **RCLS** number of terms in an integration rule

Value

Not meant to be used directly, these functions are generally called from function adaptIntegrateSimplex(...).

See Also

- `adaptIntegrateSimplex`
**CanonicalSimplex**

*Internal functions for defining/working with simplices.*

**Description**

These are utility functions that are useful when defining/working with simplices in n-dimensional space.

**Usage**

- `CanonicalSimplex(n)`
- `UnitSimplexV(n)`
- `SimplexVolume(S)`
- `SimplexSurfaceArea(S3)`
- `JacobianS2Canonical(S2)`

**Arguments**

- **n**
  - positive integer giving the dimension of the space
- **S**
  - an n x (n+1) matrix specifying a single n-dimensional simplex; the columns $S[,1]$,.., $S[,n+1]$ give the vertices of the simplex.
- **S2**
  - an n x (m+1) matrix specifying a single m-dimensional simplex, with m <= n; the columns $S2[,1]$,.., $S2[,m+1]$ give the vertices of the simplex.
- **S3**
  - an n x n matrix specifying a single (n-1)-dimensional simplex inside n-dimensional space; the columns $S3[,1]$,.., $S3[,n]$ give the vertices of the simplex.

**Value**

Let $e[j]$ be the j-th standard unit basis vector. `CanonicalSimplex(n)` gives the simplex with columns being vertices of the canonical simplex in n-dimensions: the n-dim. simplex with vertices $(0,0,...,0)$ and $e[1]$,....,$e[n]$. A vector $(u[1],...,u[n])$ is in the canonical simplex if $0 \leq u[i] \leq 1$ for all i and $\sum(u) \leq 1$. `UnitSimplexV(n)` gives the vertices (V-representation) of the unit simplex, namely $e[1]$,....,$e[n]$. A vector $(u[1],...,u[n])$ is in the unit simplex if $0 \leq u[i] \leq 1$ for all i and $\sum(u) = 1$. `SimplexVolume(S)` returns the n-dim. volume of S and `SimplexSurfaceArea(S3)` computes the (n-1)-dim. surface area of S3. `JacobianS2Canonical(S2)` returns the Jacobian of the transformation from an m-dim. simplex S2 to the m-dim. canonical simplex.

**Examples**

- `CanonicalSimplex(3)`
- `UnitSimplexV(3)`
- `SimplexVolume( CanonicalSimplex(3) )`
- `SimplexSurfaceArea( UnitSimplexV(3) )`
- `JacobianS2Canonical( UnitSimplexV(3) )`
definePoly

Define, evaluate, or print a polynomial

Description

Utility functions to work with a multivariate polynomial.

Usage

definePoly(coef, k)
printPoly( p, num.digits )
evalPoly( x, p, useTerm=rep(TRUE, length(p$coef) ) )

Arguments

ccoef: a vector of coefficients, one for each term of p(x)
k: a matrix of (non-negative, integer) powers
p: a polynomial, defined by definePoly
num.digits: number of digits to print for the coefficients of the polynomial
x: a (n x m) matrix, with columns containing the vectors where the polynomial should be evaluated
useTerm: vector of boolean values: if useTerm[i]=TRUE, term i is included in the evaluation; if useTerm[i]=FALSE, term i is not included.

Details

These are utility functions for use with integrateSimplexPolynomial. definePoly is used to define a polynomial:

\[ p(x) = \sum_{i=1}^{\text{length(coef)}} \text{coef}_i x_1^{k[i,1]} x_2^{k[i,2]} \cdots x_n^{k[i,n]} \]

printPoly prints a polynomial in human readable form.
evalPoly evaluates a polynomial at each of the vectors x[,1],x[,2],...,x[,m]. The optional argument useTerm is for internal use.
See example below.

Value

For definePoly, a list is returned. That list can be used by integrateSimplexPolynomial, printPoly, or evalPoly.
For printPoly, nothing is returned, but a human readable format is printed on the console.
For evalPoly, a vector of m values: y[i] = p(x[,i]).
Note

The internal definition of a polynomial may change in the future.

See Also

integrateSimplexPolynomial

Examples

```r
p1 <- definePoly( c(3,5), matrix(c(3,0,0,0,2,1,4), nrow=2, ncol=4, byrow=TRUE ) )
printPoly(p1)
evalPoly( c(1,3,1,2) , p1 ) # f(1,3,1,2) = 723
```

---

### grnmol

**Grundmann-Moller integration of a function over a simplex**

**Description**

Computes an approximation to the integral of a function \( f(x) \) over a simplex \( S \). This is an R translation of the matlab function grnmol.m which was written by Alan Genz.

**Usage**

```r
grnmol(f,V,s)
```

**Arguments**

- **f**
  - a (real-valued) function \( f \) that can be evaluated at all points in \( V \).

- **V**
  - a single simplex, specified by an \((n \times (n+1))\) matrix. The columns \( V[,1] ,...,V[,n+1] \) are the vertices of the simplex.

- **s**
  - a positive integer specifying the order of the rule used

**Details**

The Grundmann-Moller algorithm approximates the integral of \( f(x) \) over the simplex \( V \). When \( f(x) \) is a polynomial, and \( s \) is large enough, the integral is exact. This function is called by integrateSimplexPolynomial.

**Value**

- **Q**
  - a vector of length \( s+1 \), with \( Q[i] \) the \( i \)-th degree approximate value of the integral

- **nv**
  - number of function evaluations used

**References**

See reference by Grundmann and Moller in SimplicialCubature-package.


Examples

```r
f <- function( x ) { x[1]^2*x[4]^5 }
grnmol( f, CanonicalSimplex(4), s=4 )
```

---

**integrateSimplexPolynomial**

*Exact integration of a polynomial over a simplex*

**Description**

Computes the exact integral of a polynomial \( p(x) \) over an \( m \)-dimensional simplex \( S \) in \( n \)-dimensional space, \( 1 \leq m \leq n \). The methods are exact for polynomials, no approximation is used. The only inaccuracies possible are in the floating point evaluation of knots, coefficients, evaluation of the polynomial, sums, and products.

**Usage**

```r
integrateSimplexPolynomial(p, S, method="GM")
```

**Arguments**

- `p` a single polynomial, defined through function `definePoly`.
- `S` Either a single simplex, specified by an \( n \) x \( (m+1) \) matrix with the columns \( S[,1],...,S[,n+1] \) giving the vertices of the simplex, or a \( n \) x \( (m+1) \) x \( k \) array with \( S[..,1],...,S[..,k] \) each a single simplex as described above.
- `method` either "GM" (for the Grundmann-Moller method) or "LA" (for the Lasserre-Avrenchenkov) method

**Details**

If `method="GM"`, the Grundmann-Moller method is used; it is exact for polynomials (because the function chooses a rule of high enough degree for the degree of the polynomial \( p(x) \)). This is faster, requiring fewer function evaluations. This method works for \( n \geq 1 \) and \( 1 \leq m \leq n \).

If `method="LA"`, the algorithm splits the polynomial into terms that are homogeneous of degree \( q \), uses the method of Lasserre and Avrachenkov to exactly integrate those terms, and sums over all degrees. This method is slower, requiring more function evaluations. The degree of the polynomial has more effect on execution time than the number of terms or number of variables. This method only works with \( n > 1 \) and \( m=n \).

**Value**

- `integral` value of the integral
- `functionEvaluations` number of function evaluations used
References
See references in SimplicialCubature-package.

Examples

```r
S <- CanonicalSimplex(4) # 4-dim. simplex
p1 <- definePoly(1.0, matrix(c(2,0,0,5), nrow=1))
printPoly(p1)
# same as example for function grnmol(), but explicitly using the fact that the integrand
# function is a polynomial, and automatic selection of the order of the integration rule
integrateSimplexPolynomial( p1, S, method="GM" )
integrateSimplexPolynomial( p1, S, method="LA" )

p2 <- definePoly(c(5,-6), matrix(c(3,1,0,0,0,0,7), nrow=2, byrow=TRUE))
printPoly(p2)
integrateSimplexPolynomial( p2, S, method="GM" ) # correct answer -1.352814e-05
integrateSimplexPolynomial( p2, S, method="LA" ) # correct answer -1.352814e-05

# integrate random polynomials and random simplices in different dimensions
for (n in 3:5) {
  S <- matrix(rnorm(n*(n+1)), nrow=n, ncol=n+1)
p.rand <- definePoly(rnorm(n), matrix(c(4, rep(0,n-1)), nrow=1))
  #printPoly(p.rand)
tmp1 <- integrateSimplexPolynomial( p.rand, S, method="GM" )
tmp2 <- integrateSimplexPolynomial( p.rand, S, method="LA" )
cat("n=" , n, " GM integral=" , tmp1$integral, " functionEvaluations=" , tmp1$functionEvaluations,
  " LA integral=" , tmp2$integral, " functionEvaluations=" , tmp2$functionEvaluations,"\n")
}
```

LasserreAvrachenkov

**Internal functions for integrateSimplexPolynomial.**

Description
LasserreAvrachenkov implements the exact integration formula for a homogeneous polynomial \( p \) of degree \( q \). The other functions are helper functions for that.

Usage

LasserreAvrachenkov(q, p, useTerm, S)
nextIndexLA(current.n, b)
nextIntBaseLA(current.n, b)
Arguments

- \( q \) degree of the polynomial \( p \)
- \( p \) polynomial obtained by calling definePoly
- \( \text{useTerm} \) vector of booleans, telling which terms are homogeneous of degree \( q \)
- \( S \) an \( n \times (n+1) \) matrix specifying a single simplex; the columns \( S[,1], \ldots, S[,n+1] \) give the vertices of the simplex.
- \( \text{current.n} \) vector of integers giving the base \( b \) representation of a (non-negative) integer
- \( b \) base used for the base \( b \) representation of an integer

Value

Not meant to be called externally.

References

See Lasserre and Avrachenkov, Baldoni, et al., and Konerth references in SimplicialCubature-package.

See Also

integrateSimplexPolynomial
# Index

* Topic **Numerical mathematics**  
  SimplicialCubature-package, 2

* Topic **cubature**  
  SimplicialCubature-package, 2

* Topic **multivariate integration**  
  SimplicialCubature-package, 2

adaptIntegrateSimplex, 2, 3, 6  
adsimp, 5

CanonicalSimplex, 7

definePoly, 8

evalPoly(definePoly), 8

gnmmol, 9

integrate.vector.fn  
  (adaptIntegrateSimplex), 3

integrateSimplexPolynomial, 2, 3, 9, 10, 12

JacobianS2Canonical (CanonicalSimplex), 7

LasserreAvrachenkov, 11

nextIndexLA(LasserreAvrachenkov), 11

nextIntBaseB(LasserreAvrachenkov), 11

original.coordinates  
  (adaptIntegrateSimplex), 3

printPoly(definePoly), 8

SimplexSurfaceArea(CanonicalSimplex), 7

SimplexVolume(CanonicalSimplex), 7

SimplicialCubature  
  (SimplicialCubature-package), 2

SimplicialCubature-package, 2

SMPCHC (adsimp), 5

SMPDFS (adsimp), 5

SMPRMS (adsimp), 5

SMPRUL (adsimp), 5

SMPSAD (adsimp), 5

SMPSMS (adsimp), 5

UnitSimplexV(CanonicalSimplex), 7