Package 'cubature'

July 2, 2014

Type Package

Title Adaptive multivariate integration over hypercubes

Version 1.1-2

Date 2013-02-24

Author C code by Steven G. Johnson, R by Balasubramanian Narasimhan

Maintainer Balasubramanian Narasimhan <naras@stat.stanford.edu>

Description Adaptive multivariate integration over hypercubes

License GPL (>= 2)

NeedsCompilation yes

Repository CRAN

Date/Publication 2013-02-25 07:34:36

R topics documented:

`cubature-package` *Cubature is a package for adaptive multidimensional integration over hypercubes*

Description

Cubature is a package for adaptive multidimensional integration over hypercubes. It is a wrapper around the pure C, GPLed implementation by Steven G. Johnson available from the URL http://ab-initio.mit.edu/wiki/index.php/Cubature.

Details

Package:	cubature
Type:	Package
Version:	1.0
Date:	2009-12-17
License:	GPL V2 or later
LazyLoad:	yes

There is only one function in the package called adaptIntegrate.

Author(s)

C code by Steven G. Johnson, R by Balasubramanian Narasimhan

Maintainer: Balasubramanian Narasimhan<naras@stat.stanford.edu>

References

See http://ab-initio.mit.edu/wiki/index.php/Cubature

`adaptIntegrate` *Adaptive multivariate integration over hypercubes*

Description

The function performs adaptive multidimensional integration (cubature) of (possibly) vector-valued integrands over hypercubes.

Usage

```
adaptIntegrate(f, lowerLimit, upperLimit, ..., tol = 1e-05, fDim = 1,
               maxEval = 0, absError=0, doChecking=FALSE)
```

Arguments

`f`	The function (integrand) to be integrated
`lowerLimit`	The lower limit of integration, a vector for hypercubes
`upperLimit`	The upper limit of integration, a vector for hypercubes
`...`	All other arguments passed to the function f
`tol`	The maximum tolerance, default 1e-5.
`fDim`	The dimension of the integrand, default 1, bears no relation to the dimension of the hypercube
`maxEval`	The maximum number of function evaluations needed, default 0 implying no limit
`absError`	The maximum absolute error tolerated
`doChecking`	A flag to be a bit anal about checking inputs to C routines. A FALSE value results in approximately 9 percent speed gain in our experiments. Your mileage will of course vary. Default value is FALSE.

Details

The function merely calls Johnson's C code and returns the results. The original C code by Johnson was modified for use with R memory allocation functions and a helper function does the callback.

One can specify a maximum number of function evaluations (default is 0 for no limit). Otherwise, the integration stops when the estimated error is less than the absolute error requested, or when the estimated error is less than tol times the integral, in absolute value.

Value

The returned value is a list of three items:

`integral`	the value of the integral
`error`	the estimated relative error
`functionEvaluations`	the number of times the function was evaluated
`returnCode`	the actual integer return code of the C routine

Author(s)

Balasubramanian Narasimhan

References

See http://ab-initio.mit.edu/wiki/index.php/Cubature.

Examples

```
## Test function 0
## Compare with original cubature result of
## ./cubature_test 2 1e-4 0 0
## 2-dim integral, tolerance = 0.0001
## integrand 0: integral = 0.708073, est err = 1.70943e-05, true err = 7.69005e-09
## #evals = 17

testFn0 <- function(x) {
  prod(cos(x))
}

adaptIntegrate(testFn0, rep(0,2), rep(1,2), tol=1e-4)

M_2_SQRTPI <- 2/sqrt(pi)

## Test function 1
## Compare with original cubature result of
## ./cubature_test 3 1e-4 1 0
## 3-dim integral, tolerance = 0.0001
## integrand 1: integral = 1.00001, est err = 9.67798e-05, true err = 9.76919e-06
## #evals = 5115

testFn1 <- function(x) {
```

```
  scale = 1.0
  val = 0
  dim = length(x)
  val = sum (((1-x) / x)^2)
  scale = prod(M_2_SQRTPI/x^2)
  exp(-val) * scale
}

adaptIntegrate(testFn1, rep(0, 3), rep(1, 3), tol=1e-4)

##
## Test function 2
## Compare with original cubature result of
## ./cubature_test 2 1e-4 2 0
## 2-dim integral, tolerance = 0.0001
## integrand 2: integral = 0.19728, est err = 1.97261e-05, true err = 4.58316e-05
## #evals = 166141

testFn2 <- function(x) {
  ## discontinuous objective: volume of hypersphere
  radius = as.double(0.50124145262344534123412)
  ifelse(sum(x*x) < radius*radius, 1, 0)
}

adaptIntegrate(testFn2, rep(0, 2), rep(1, 2), tol=1e-4)

##
## Test function 3
## Compare with original cubature result of
## ./cubature_test 3 1e-4 3 0
## 3-dim integral, tolerance = 0.0001
## integrand 3: integral = 1, est err = 0, true err = 2.22045e-16
## #evals = 33

testFn3 <- function(x) {
  prod(2*x)
}

adaptIntegrate(testFn3, rep(0,3), rep(1,3), tol=1e-4)

##
## Test function 4 (Gaussian centered at 1/2)
## Compare with original cubature result of
## ./cubature_test 2 1e-4 4 0
## 2-dim integral, tolerance = 0.0001
## integrand 4: integral = 1, est err = 9.84399e-05, true err = 2.78894e-06
## #evals = 1853

testFn4 <- function(x) {
  a = 0.1
  s = sum((x-0.5)^2)
  (M_2_SQRTPI / (2. * a))^length(x) * exp (-s / (a * a))
}
```

```
adaptIntegrate(testFn4, rep(0,2), rep(1,2), tol=1e-4)


##
## Test function 5 (double Gaussian)
## Compare with original cubature result of
## ./cubature_test 3 1e-4 5 0
## 3-dim integral, tolerance = 0.0001
## integrand 5: integral = 0.999994, est err = 9.98015e-05, true err = 6.33407e-06
## #evals = 59631

testFn5 <- function(x) {
  a = 0.1
  s1 = sum((x-1/3)^2)
  s2 = sum((x-2/3)^2)
  0.5 * (M_2_SQRTPI / (2. * a))^length(x) * (exp(-s1 / (a * a)) + exp(-s2 / (a * a)))
}

adaptIntegrate(testFn5, rep(0,3), rep(1,3), tol=1e-4)

##
## Test function 6 (Tsuda's example)
## Compare with original cubature result of
## ./cubature_test 4 1e-4 6 0
## 4-dim integral, tolerance = 0.0001
## integrand 6: integral = 0.999998, est err = 9.99685e-05, true err = 1.5717e-06
## #evals = 18753

testFn6 <- function(x) {
  a = (1+sqrt(10.0))/9.0
  prod(a/(a+1)*((a+1)/(a+x))^2)
}

adaptIntegrate(testFn6, rep(0,4), rep(1,4), tol=1e-4)


##
## Test function 7
##   test integrand from W. J. Morokoff and R. E. Caflisch, "Quasi=
##   Monte Carlo integration," J. Comput. Phys 122, 218-230 (1995).
##   Designed for integration on [0,1]^dim, integral = 1. */
## Compare with original cubature result of
## ./cubature_test 3 1e-4 7 0
## 3-dim integral, tolerance = 0.0001
## integrand 7: integral = 1.00001, est err = 9.96657e-05, true err = 1.15994e-05
## #evals = 7887

testFn7 <- function(x) {
  n <- length(x)
  p <- 1/n
  (1+p)^n * prod(x^p)
}
```

```
adaptIntegrate(testFn7, rep(0,3), rep(1,3), tol=1e-4)


## Example from web page
## http://ab-initio.mit.edu/wiki/index.php/Cubature
##
## f(x) = exp(-0.5(euclidean_norm(x)^2)) over the three-dimensional
## hyperbcube [-2, 2]^3
## Compare with original cubature result
testFnWeb <-  function(x) {
  exp(-0.5*sum(x^2))
}

adaptIntegrate(testFnWeb, rep(-2,3), rep(2,3), tol=1e-4)

## Test function I.1d from
## Numerical integration using Wang-Landau sampling
## Y. W. Li, T. Wust, D. P. Landau, H. Q. Lin
## Computer Physics Communications, 2007, 524-529
## Compare with exact answer: 1.63564436296
##
I.1d <- function(x) {
  sin(4*x) *
    x * ((x * ( x * (x*x-4) + 1) - 1))
}

adaptIntegrate(I.1d, -2, 2, tol=1e-7)

## Test function I.2d from
## Numerical integration using Wang-Landau sampling
## Y. W. Li, T. Wust, D. P. Landau, H. Q. Lin
## Computer Physics Communications, 2007, 524-529
## Compare with exact answer: -0.01797992646
##
## Test function I.2d from
## Numerical integration using Wang-Landau sampling
## Y.W. Li, T. Wust, D.P. Landau, H.Q. Lin
## Computer Physics Communications, 2007 524-529
## Compare with exact answer: -0.01797992646
##
I.2d <- function(x) {
  x1 = x[1]
  x2 = x[2]
  sin(4*x1+1) * cos(4*x2) * x1 * (x1*(x1*x1)^2 - x2*(x2*x2 - x1) +2)
}

adaptIntegrate(I.2d, rep(-1, 2), rep(1, 2), maxEval=10000)

##
## Example of multivariate normal integration borrowed from
## package mvtnorm (on CRAN) to check ... argument
## Compare with output of
```

```
## pmvnorm(lower=rep(-0.5, m), upper=c(1,4,2), mean=rep(0, m), corr=sigma, alg=Miwa())
##     0.3341125.  Blazing quick as well!  Ours is, not unexpectedly, much slower.
##
dmvnorm <- function (x, mean, sigma, log = FALSE) {
    if (is.vector(x)) {
        x <- matrix(x, ncol = length(x))
    }
    if (missing(mean)) {
        mean <- rep(0, length = ncol(x))
    }
    if (missing(sigma)) {
        sigma <- diag(ncol(x))
    }
    if (NCOL(x) != NCOL(sigma)) {
        stop("x and sigma have non-conforming size")
    }
    if (!isSymmetric(sigma, tol = sqrt(.Machine$double.eps),
        check.attributes = FALSE)) {
        stop("sigma must be a symmetric matrix")
    }
    if (length(mean) != NROW(sigma)) {
        stop("mean and sigma have non-conforming size")
    }
    distval <- mahalanobis(x, center = mean, cov = sigma)
    logdet <- sum(log(eigen(sigma, symmetric = TRUE, only.values = TRUE)$values))
    logretval <- -(ncol(x) * log(2 * pi) + logdet + distval)/2
    if (log)
        return(logretval)
    exp(logretval)
}

m <- 3
sigma <- diag(3)
sigma[2,1] <- sigma[1, 2] <- 3/5 ; sigma[3,1] <- sigma[1, 3] <- 1/3
sigma[3,2] <- sigma[2, 3] <- 11/15
adaptIntegrate(dmvnorm, lower=rep(-0.5, m), upper=c(1,4,2),
                        mean=rep(0, m), sigma=sigma, log=FALSE,
               maxEval=10000)
```

Index