Package 'cubature'

July 19, 2017

Type Package

Title Adaptive Multivariate Integration over Hypercubes

Version 1.3-11

VignetteBuilder knitr

URL https://github.com/bnaras/cubature

Description R wrapper around the cubature C library of
Steven G. Johnson for adaptive multivariate integration over hypercubes.
This version provides both hcubature and pcubature routines in addition
to a vector interface that results in substantial speed gains.

License GPL-3

LinkingTo Rcpp

Imports Rcpp

NeedsCompilation yes

RoxygenNote 6.0.1

Suggests testthat, knitr, mvtnorm, R2Cuba, benchr

Author Balasubramanian Narasimhan [aut, cre],
Manuel Koller [ctb],
Steven G. Johnson [aut]

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

Repository CRAN

Date/Publication 2017-07-19 16:27:43 UTC

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.

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>

hcubature *Adaptive multivariate integration over hypercubes (hcubature and pcubature)*

Description

The function performs adaptive multidimensional integration (cubature) of (possibly) vector-valued integrands over hypercubes. The function includes a vector interface where the integrand may be evaluated at several hundred points in a single call.

Usage

```
hcubature(f, lowerLimit, upperLimit, ..., tol = 1e-05, fDim = 1,
  maxEval = 0, absError = 0, doChecking = FALSE,
  vectorInterface = FALSE, norm = c("INDIVIDUAL", "PAIRED", "L2", "L1",
  "LINF"))

pcubature(f, lowerLimit, upperLimit, ..., tol = 1e-05, fDim = 1,
```

```
maxEval = 0, absError = 0, doChecking = FALSE,
vectorInterface = FALSE, norm = c("INDIVIDUAL", "PAIRED", "L2", "L1",
"LINF"))
```

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. Note that the actual number of function evaluations performed is only approximately guaranteed not to exceed this number.
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.
vectorInterface	A flag that indicates whether to use the vector interface and is by default FALSE. See details below
norm	For vector-valued integrands, norm specifies the norm that is used to measure the error and determine convergence properties. See below.

Details

The function merely calls Johnson's C code and returns the results.

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, or the maximum number of iterations is reached (see parameter info below), whichever is earlier.

For compatibility with earlier versions, the adaptIntegrate function is an alias for the underlying hcubature function which uses h-adaptive integration. Otherwise, the calling conventions are the same.

We highly recommend referring to the vignette to achieve the best results!

The hcubature function is the h-adaptive version that recursively partitions the integration domain into smaller subdomains, applying the same integration rule to each, until convergence is achieved.

The p-adaptive version, pcubature, repeatedly doubles the degree of the quadrature rules until convergence is achieved, and is based on a tensor product of Clenshaw-Curtis quadrature rules. This algorithm is often superior to h-adaptive integration for smooth integrands in a few (<=3) dimensions, but is a poor choice in higher dimensions or for non-smooth integrands. Compare with hcubature which also takes the same arguments.

The vector interface requires the integrand to take a matrix as its argument. The return value should also be a matrix. The number of points at which the integrand may be evaluated is not under user control: the integration routine takes care of that and this number may run to several hundreds. We strongly advise vectorization; see vignette.

The `norm` argument is irrelevant for scalar integrands and is ignored. Given vectors v and e of estimated integrals and errors therein, respectively, the `norm` argument takes on one of the following values:

`INDIVIDUAL` Convergence is achieved only when each integrand (each component of v and e) individually satisfies the requested error tolerances

`L1`, `L2`, `LINF` The absolute error is measured as $|e|$ and the relative error as $|e|/|v|$, where $|...|$ is the L_1, L_2, or $_\infty$ norm, respectively

`PAIRED` Like `INDIVIDUAL`, except that the integrands are grouped into consecutive pairs, with the error tolerance applied in an L_2 sense to each pair. This option is mainly useful for integrating vectors of complex numbers, where each consecutive pair of real integrands is the real and imaginary parts of a single complex integrand, and the concern is only the error in the complex plane rather than the error in the real and imaginary parts separately

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

Examples

```
## Not run:
## 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))
}

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

pcubature(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) {
  val <- sum (((1-x) / x)^2)
  scale <- prod(M_2_SQRTPI/x^2)
  exp(-val) * scale
}

hcubature(testFn1, rep(0, 3), rep(1, 3), tol=1e-4)
pcubature(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(0.5012414526234453412 3412)
  ifelse(sum(x*x) < radius*radius, 1, 0)
}

hcubature(testFn2, rep(0, 2), rep(1, 2), tol=1e-4)
pcubature(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)
}

hcubature(testFn3, rep(0,3), rep(1,3), tol=1e-4)
pcubature(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))
}

hcubature(testFn4, rep(0,2), rep(1,2), tol=1e-4)
pcubature(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)))
}

hcubature(testFn5, rep(0,3), rep(1,3), tol=1e-4)
pcubature(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)
}

hcubature(testFn6, rep(0,4), rep(1,4), tol=1e-4)
pcubature(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)
}

hcubature(testFn7, rep(0,3), rep(1,3), tol=1e-4)
pcubature(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))
}

hcubature(testFnWeb, rep(-2,3), rep(2,3), tol=1e-4)
pcubature(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))
}

hcubature(I.1d, -2, 2, tol=1e-7)
pcubature(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
##
##
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)
}

hcubature(I.2d, rep(-1, 2), rep(1, 2), maxEval=10000)
pcubature(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
hcubature(dmvnorm, lower=rep(-0.5, m), upper=c(1,4,2),
                        mean=rep(0, m), sigma=sigma, log=FALSE,
                maxEval=10000)
pcubature(dmvnorm, lower=rep(-0.5, m), upper=c(1,4,2),
                        mean=rep(0, m), sigma=sigma, log=FALSE,
                maxEval=10000)
```

```
## End(Not run)
```

Index