Package 'cubature'

December 18, 2018

Type Package

Title Adaptive Multivariate Integration over Hypercubes

Version 2.0.3

VignetteBuilder knitr

SystemRequirements GNU make

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

BugReports https://github.com/bnaras/cubature/issues

Description R wrappers around the cubature C library of Steven G. Johnson for adaptive multivariate integration over hypercubes and the Cuba C library of Thomas Hahn for deterministic and Monte Carlo integration. Scalar and vector interfaces for cubature and Cuba routines are provided; the vector interfaces are highly recommended as demonstrated in the package vignette.

License GPL-3

Encoding UTF-8

LinkingTo Rcpp

Imports Rcpp

NeedsCompilation yes

RoxygenNote 6.1.1

Suggests testthat, knitr, mvtnorm, benchr

Author Balasubramanian Narasimhan [aut, cre],
Manuel Koller [ctb],
Steven G. Johnson [aut],
Thomas Hahn [aut],
Annie Bouvier [aut],
Kiên Kiêu [aut],
Simen Gaure [ctb]

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

Repository CRAN

Date/Publication 2018-12-18 11:00:10 UTC

R topics documented:

cubature-package 2
cubintegrate 2
cuhre 4
default_args 7
divonne 7
hcubature 12
suave 18
vegas 22

Index **26**

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

Description

Cubature is a package for adaptive and monte-carlo multidimensional integration over hypercubes. It is a wrapper around the pure C, GPLed implementations by Steven G. Johnson (cubature) and Thomas Hahn (Cuba) libraries.

Author(s)

C code by Steven G. Johnson and Thomas Hahn, R by Balasubramanian Narasimhan, Manuel Koller, Simen Gaure, Kiên Kiêu, and Annie Bouvier

Maintainer: Balasubramanian Narasimhannaras@stat.stanford.edu

---

`cubintegrate` *Unified Cubature Integration Interface*

Description

Integrate a function within specified limits using method specified. Further arguments specific to method as well as other arguments to f may be passed. For defaults used in each method, see help on the method or `default_args`'s.

Usage

```
cubintegrate(f, lower, upper, fDim = 1, method = c("hcubature",
  "pcubature", "cuhre", "divonne", "suave", "vegas"), relTol = 1e-05,
  absTol = 1e-12, maxEval = 10^6, nVec = 1L, ...)
```

Arguments

f	The function (integrand) to be integrated. Can be vectorized version, but the additional arguments ... must indicate via either vectorInterface = TRUE for hcubature and pcubature, or a value for nVec. See details on each method.
lower	The lower limit of integration, a vector for hypercubes.
upper	The upper limit of integration, a vector for hypercubes.
fDim	The number of components of f, default 1, bears no relation to the dimension of the hypercube over which integration is performed.
method	the method to use should be one of "hcubature", "pcubature", "cuhre", "divonne", "suave" or "vegas".
relTol	The maximum tolerance, default 1e-5.
absTol	the absolute tolerance, default 1e-12.
maxEval	The maximum number of function evaluations needed, default 10^6. Note that the actual number of function evaluations performed is only approximately guaranteed not to exceed this number.
nVec	the number of vectorization points for Cuba C library, default 1, but can be set to an integer > 1 for vectorization, for example, 1024. The function f above needs to handle the vector of points appropriately; see vignette examples. Unlike Cuba, the cubature C library manages the number of points on its own and can vary between calls. Therefore, any value for nVec greater than one implies vectorization for a cubature method.
...	All other arguments which may include integration method specific parameters and those for f. Unrecognized parameters for integration method are presumed to be intended for f and so processed.

Value

The returned value is a list of items: -

integral	the value of the integral
-	
error	the estimated relative error for cubature; for Cuba it is the estimated absolute error
neval	the number of times the function was evaluated
-	
returnCode	the actual integer return code of the C routine; a non-zero value usually indicates problems; further interpretation depends on method
-	
nregions	for Cuba routines, the actual number of subregions needed
prob	the χ^2-probability (not the χ^2-value itself!) that error is not a reliable estimate of the true integration error.

See Also

default_args, hcubature, pcubature, cuhre, vegas, suave, divonne

Examples

```
I.1d <- function(x) {
  sin(4*x) *
    x * ((x * ( x * (x*x-4) + 1) - 1))
}
I.1d_v <- function(x) {
   matrix(apply(x, 2, function(z)
       sin(4 * z) *
       z * ((z * ( z * (z * z - 4) + 1) - 1))),
       ncol = ncol(x))
}
cubintegrate(f = I.1d, lower = -2, upper = 2, method = "pcubature")
cubintegrate(f = I.1d, lower = -2, upper = 2, method = "cuhre", flags=list(verbose = 2))
cubintegrate(f = I.1d_v, lower = -2, upper = 2, method = "hcubature", nVec = 2L)
cubintegrate(f = I.1d_v, lower = -2, upper = 2, method = "cuhre", nVec = 128L)
```

---

cuhre *Integration by a Deterministic Iterative Adaptive Algorithm*

Description

Implement a deterministic algorithm for multidimensional numerical integration. Its algorithm uses one of several cubature rules in a globally adaptive subdivision scheme. The subdivision algorithm is similar to suave's.

Usage

```
cuhre(f, nComp = 1L, lowerLimit, upperLimit, ..., relTol = 1e-05,
  absTol = 1e-12, minEval = 0L, maxEval = 10^6,
  flags = list(verbose = 0L, final = 1L, keep_state = 0L, level = 0L),
  key = 0L, nVec = 1L, stateFile = NULL)
```

Arguments

f	The function (integrand) to be integrated. For cuhre, it can be something as simple as a function of a single argument, say x.
nComp	The number of components of f, default 1, bears no relation to the dimension of the hypercube over which integration is performed.
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.
relTol	The maximum tolerance, default 1e-5.

`absTol` the absolute tolerance, default 1e-12.

`minEval` the minimum number of function evaluations required

`maxEval` The maximum number of function evaluations needed, default 10^6. Note that the actual number of function evaluations performed is only approximately guaranteed not to exceed this number.

`flags` flags governing the integration. The list here is exhaustive to keep the documentation and invocation uniform, but not all flags may be used for a particular method as noted below. List components:

`verbose` encodes the verbosity level, from 0 (default) to 3. Level 0 does not print any output, level 1 prints reasonable information on the progress of the integration, level 2 also echoes the input parameters, and level 3 further prints the subregion results.

`final` when 0, all sets of samples collected on a subregion during the various iterations or phases contribute to the final result. When 1, only the last (largest) set of samples is used in the final result.

`smooth` Applies to Suave and Vegas only. When 0, apply additional smoothing to the importance function, this moderately improves convergence for many integrands. When 1, use the importance function without smoothing, this should be chosen if the integrand has sharp edges.

`keep_state` when nonzero, retain state file if argument `stateFile` is non-null, else delete `stateFile` if specified.

`load_state` Applies to Vegas only. Reset the integrator's state even if a state file is present, i.e. keep only the grid. Together with `keep_state` this allows a grid adapted by one integration to be used for another integrand.

`level` applies only to Divonne, Suave and Vegas. When `0`, Mersenne Twister random numbers are used. When nonzero Ranlux random numbers are used, except when `rngSeed` is zero which forces use of Sobol quasi-random numbers. Ranlux implements Marsaglia and Zaman's 24-bit RCARRY algorithm with generation period p, i.e. for every 24 generated numbers used, another p-24 are skipped. The luxury level for the Ranlux generator may be encoded in `level` as follows:

Level 1 (p = 48) gives very long period, passes the gap test but fails spectral test

Level 2 (p = 97) passes all known tests, but theoretically still defective

Level 3 (p = 223) any theoretically possible correlations have very small chance of being observed

Level 4 (p = 389) highest possible luxury, all 24 bits chaotic

Levels 5-23 default to 3, values above 24 directly specify the period p.

Note that Ranlux's original level 0, (mis)used for selecting Mersenne Twister in Cuba, is equivalent to `level` = 24.

`key` the quadrature rule key: `key = 7, 9, 11, 13` selects the cubature rule of degree key. Note that the degree-11 rule is available only in 3 dimensions, the degree-13 rule only in 2 dimensions. For other values, including the default 0, the rule is the degree-13 rule in 2 dimensions, the degree-11 rule in 3 dimensions, and the degree-9 rule otherwise.

`nVec` the number of vectorization points, default 1, but can be set to an integer > 1 for vectorization, for example, 1024 and the function f above needs to handle the vector of points appropriately. See vignette examples.

`stateFile` the name of an external file. Vegas can store its entire internal state (i.e. all the information to resume an interrupted integration) in an external file. The state file is updated after every iteration. If, on a subsequent invocation, Vegas finds a file of the specified name, it loads the internal state and continues from the point it left off. Needless to say, using an existing state file with a different integrand generally leads to wrong results. Once the integration finishes successfully, i.e. the prescribed accuracy is attained, the state file is removed. This feature is useful mainly to define 'check-points' in long-running integrations from which the calculation can be restarted.

Details

See details in the documentation.

Value

A list with components:

nregions the actual number of subregions needed

neval the actual number of integrand evaluations needed

returnCode if zero, the desired accuracy was reached, if -1, dimension out of range, if 1, the accuracy goal was not met within the allowed maximum number of integrand evaluations.

integral vector of length `nComp`; the integral of `integrand` over the hypercube

error vector of length `nComp`; the presumed absolute error of `integral`

prob vector of length `nComp`; the χ^2-probability (not the χ^2-value itself!) that `error` is not a reliable estimate of the true integration error.

References

J. Berntsen, T. O. Espelid (1991) An adaptive algorithm for the approximate calculation of multiple integrals. *ACM Transactions on Mathematical Software*, **17**(4), 437-451.

T. Hahn (2005) CUBA-a library for multidimensional numerical integration. *Computer Physics Communications*, **168**, 78-95.

See http://www.feynarts.de/cuba/

See Also

vegas, suave, divonne

Examples

```
integrand <- function(arg) {
  x <- arg[1]
  y <- arg[2]
```

```
  z <- arg[3]
  ff <- sin(x)*cos(y)*exp(z);
return(ff)
} # End integrand

NDIM <- 3
NCOMP <- 1
cuhre(f = integrand,
      lowerLimit = rep(0, NDIM),
      upperLimit = rep(1, NDIM),
      relTol = 1e-3, absTol= 1e-12,
      flags = list(verbose = 2, final = 0))
```

`default_args` *Default arguments for each integration method*

Description

Since each method has a different set of parameters, this function returns the default values of all parameters that can be modified and passed to integration routines.

Usage

```
default_args()
```

Value

a named list of parameters for each method.

Examples

```
default_args()
```

`divonne` *Integration by Stratified Sampling for Variance Reduction*

Description

Divonne works by stratified sampling, where the partioning of the integration region is aided by methods from numerical optimization.

Usage

```
divonne(f, nComp = 1L, lowerLimit, upperLimit, ..., relTol = 1e-05,
  absTol = 1e-12, minEval = 0L, maxEval = 10^6,
  flags = list(verbose = 0L, final = 1L, keep_state = 0L, level = 0L),
  rngSeed = 0L, nVec = 1L, key1 = 47L, key2 = 1L, key3 = 1L,
  maxPass = 5L, border = 0, maxChisq = 10, minDeviation = 0.25,
  xGiven = NULL, nExtra = 0L, peakFinder = NULL, stateFile = NULL)
```

Arguments

f	The function (integrand) to be integrated as in cuhre. Optionally, the function can take an additional arguments in addition to the variable being integrated: - cuba_phase. The last argument, phase, indicates the integration phase: 0. sampling of the points in xgiven, 1. partitioning phase, 2. final integration phase, 3. refinement phase. This information might be useful if the integrand takes long to compute and a sufficiently accurate approximation of the integrand is available. The actual value of the integral is only of minor importance in the partitioning phase, which is instead much more dependent on the peak structure of the integrand to find an appropriate tessellation. An approximation which reproduces the peak structure while leaving out the fine details might hence be a perfectly viable and much faster substitute when cuba_phase < 2. In all other instances, phase can be ignored and it is entirely admissible to define the integrand without it. which is the
nComp	The number of components of f, default 1, bears no relation to the dimension of the hypercube over which integration is performed.
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.
relTol	The maximum tolerance, default 1e-5.
absTol	the absolute tolerance, default 1e-12.
minEval	the minimum number of function evaluations required
maxEval	The maximum number of function evaluations needed, default 10^6. Note that the actual number of function evaluations performed is only approximately guaranteed not to exceed this number.
flags	flags governing the integration. The list here is exhaustive to keep the documentation and invocation uniform, but not all flags may be used for a particular method as noted below. List components: verbose encodes the verbosity level, from 0 (default) to 3. Level 0 does not print any output, level 1 prints reasonable information on the progress of the integration, level 2 also echoes the input parameters, and level 3 further prints the subregion results. final when 0, all sets of samples collected on a subregion during the various iterations or phases contribute to the final result. When 1, only the last (largest) set of samples is used in the final result.

`smooth` Applies to Suave and Vegas only. When 0, apply additional smoothing to the importance function, this moderately improves convergence for many integrands. When 1, use the importance function without smoothing, this should be chosen if the integrand has sharp edges.

`keep_state` when nonzero, retain state file if argument `stateFile` is non-null, else delete `stateFile` if specified.

`load_state` Applies to Vegas only. Reset the integrator's state even if a state file is present, i.e. keep only the grid. Together with `keep_state` this allows a grid adapted by one integration to be used for another integrand.

`level` applies only to Divonne, Suave and Vegas. When `0`, Mersenne Twister random numbers are used. When nonzero Ranlux random numbers are used, except when `rngSeed` is zero which forces use of Sobol quasi-random numbers. Ranlux implements Marsaglia and Zaman's 24-bit RCARRY algorithm with generation period p, i.e. for every 24 generated numbers used, another p-24 are skipped. The luxury level for the Ranlux generator may be encoded in `level` as follows:

Level 1 (p = 48) gives very long period, passes the gap test but fails spectral test

Level 2 (p = 97) passes all known tests, but theoretically still defective

Level 3 (p = 223) any theoretically possible correlations have very small chance of being observed

Level 4 (p = 389) highest possible luxury, all 24 bits chaotic

Levels 5-23 default to 3, values above 24 directly specify the period p.

Note that Ranlux's original level 0, (mis)used for selecting Mersenne Twister in Cuba, is equivalent to `level` = 24.

`rngSeed` seed, default 0, for the random number generator. Note the articulation with `level` settings for `flag`

`nVec` the number of vectorization points, default 1, but can be set to an integer > 1 for vectorization, for example, 1024 and the function f above needs to handle the vector of points appropriately. See vignette examples.

`key1` integer that determines sampling in the partitioning phase: `key1 = 7, 9, 11, 13` selects the cubature rule of degree `key1`. Note that the degree-11 rule is available only in 3 dimensions, the degree-13 rule only in 2 dimensions. For other values of `key1`, a quasi-random sample of $n = |key1|$ points is used, where the sign of `key1` determines the type of sample, `key1 = 0`, use the default rule. `key1 > 0`, use a Korobov quasi-random sample, `key1 < 0`, use a Sobol quasi-random sample if `flags$seed` is zero, otherwise a "standard" sample (Mersenne Twister) pseudo-random sample

`key2` integer that determines sampling in the final integration phase: same as `key1`, but here

$$n = |key2|$$

determines the number of points, $n > 39$, sample n points, $n < 40$, sample n `nneed` points, where `nneed` is the number of points needed to reach the prescribed accuracy, as estimated by Divonne from the results of the partitioning phase.

key3 — integer that sets the strategy for the refinement phase: key3 = 0, do not treat the subregion any further. key3 = 1, split the subregion up once more. Otherwise, the subregion is sampled a third time with key3 specifying the sampling parameters exactly as key2 above.

maxPass — integer that controls the thoroughness of the partitioning phase: The partitioning phase terminates when the estimated total number of integrand evaluations (partitioning plus final integration) does not decrease for maxPass successive iterations. A decrease in points generally indicates that Divonne discovered new structures of the integrand and was able to find a more effective partitioning. maxPass can be understood as the number of "safety" iterations that are performed before the partition is accepted as final and counting consequently restarts at zero whenever new structures are found.

border — the relative width of the border of the integration region. Points falling into the border region will not be sampled directly, but will be extrapolated from two samples from the interior. Use a non-zero border if the integrand subroutine cannot produce values directly on the integration boundary. The relative width of the border is identical in all the dimensions. For example, set border=0.1 for a border of width equal to 10% of the width of the integration region.

maxChisq — the maximum χ^2 value a single subregion is allowed to have in the final integration phase. Regions which fail this χ^2 test and whose sample averages differ by more than min.deviation move on to the refinement phase.

minDeviation — a bound, given as the fraction of the requested error of the entire integral, which determines whether it is worthwhile further examining a region that failed the χ^2 test. Only if the two sampling averages obtained for the region differ by more than this bound is the region further treated.

xGiven — a matrix (nDim, nGiven). A list of nGiven points where the integrand might have peaks. Divonne will consider these points when partitioning the integration region. The idea here is to help the integrator find the extrema of the integrand in the presence of very narrow peaks. Even if only the approximate location of such peaks is known, this can considerably speed up convergence.

nExtra — the maximum number of extra points the peak-finder subroutine will return. If nextra is zero, peakfinder is not called and an arbitrary object may be passed in its place, e.g. just 0.

peakFinder — the peak-finder subroutine. This R function is called whenever a region is up for subdivision and is supposed to point out possible peaks lying in the region, thus acting as the dynamic counterpart of the static list of points supplied in xgiven. It is expected to be declared as peakFinder <- function(bounds, nMax) where bounds is a matrix of dimension (2, nDim) which contains the lower (row 1) and upper (row 2) bounds of the subregion. The returned value should be a matrix (nX, nDim) where nX is the actual number of points (should be less or equal to nMax).

stateFile — the name of an external file. Vegas can store its entire internal state (i.e. all the information to resume an interrupted integration) in an external file. The state file is updated after every iteration. If, on a subsequent invocation, Vegas finds a file of the specified name, it loads the internal state and continues from the point it left off. Needless to say, using an existing state file with a different integrand

generally leads to wrong results. Once the integration finishes successfully, i.e. the prescribed accuracy is attained, the state file is removed. This feature is useful mainly to define 'check-points' in long-running integrations from which the calculation can be restarted.

Details

Divonne uses stratified sampling for variance reduction, that is, it partitions the integration region such that all subregions have an approximately equal value of a quantity called the spread (volume times half-range).

See details in the documentation.

Value

A list with components:

nregions the actual number of subregions needed

neval the actual number of integrand evaluations needed

returnCode if zero, the desired accuracy was reached, if -1, dimension out of range, if 1, the accuracy goal was not met within the allowed maximum number of integrand evaluations.

integral vector of length `nComp`; the integral of `integrand` over the hypercube

error vector of length `nComp`; the presumed absolute error of `integral`

prob vector of length `nComp`; the χ^2-probability (not the χ^2-value itself!) that `error` is not a reliable estimate of the true integration error.

References

J. H. Friedman, M. H. Wright (1981) A nested partitioning procedure for numerical multiple integration. *ACM Trans. Math. Software*, **7**(1), 76-92.

J. H. Friedman, M. H. Wright (1981) User's guide for DIVONNE. SLAC Report CGTM-193-REV, CGTM-193, Stanford University.

T. Hahn (2005) CUBA-a library for multidimensional numerical integration. *Computer Physics Communications*, **168**, 78-95.

See Also

cuhre, suave, vegas

Examples

```
integrand <- function(arg, phase) {
  x <- arg[1]
  y <- arg[2]
  z <- arg[3]
  ff <- sin(x)*cos(y)*exp(z);
return(ff)
}
divonne(integrand, relTol=1e-3,  absTol=1e-12, lowerLimit = rep(0, 3), upperLimit = rep(1, 3),
```

```
        flags=list(verbose = 2),  key1= 47)

# Example with a peak-finder function
nDim <- 3L
peakf <- function(bounds, nMax) {
#  print(bounds) # matrix (ndim,2)
  x <- matrix(0, ncol = nMax, nrow = nDim)
   pas <- 1 / (nMax - 1)
   # 1ier point
   x[, 1] <- rep(0, nDim)
   # Les autres points
   for (i in 2L:nMax) {
      x[, i] <- x[, (i - 1)] + pas
    }
  x
} #end peakf

divonne(integrand, relTol=1e-3,  absTol=1e-12,
        lowerLimit = rep(0, 3), upperLimit = rep(1, 3),
        flags=list(verbose = 2),  peakFinder = peakf, nExtra = 4L)
```

`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`	As of version 2.0, this flag is ignored and will be dropped in forthcoming versions
`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 four 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(0.50124145262344534123412)
  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)
```

---

suave *Integration with SUbregion-Adaptive Vegas Algorithm*

Description

Suave uses vegas-like importance sampling combined with a globally adaptive subdivision strategy: Until the requested accuracy is reached, the region with the largest error at the time is bisected in the dimension in which the fluctuations of the integrand are reduced most. The number of new samples in each half is prorated for the fluctuation in that half.

Usage

```
suave(f, nComp = 1L, lowerLimit, upperLimit, ..., relTol = 1e-05,
  absTol = 1e-12, minEval = 0L, maxEval = 10^6,
  flags = list(verbose = 0L, final = 1L, smooth = 0L, keep_state = 0L,
  level = 0L), rngSeed = 0L, nVec = 1L, nNew = 1000L, nMin = 50L,
  flatness = 50, stateFile = NULL)
```

Arguments

f	The function (integrand) to be integrated as in cuhre. Optionally, the function can take two additional arguments in addition to the variable being integrated: - cuba_weight which is the weight of the point being sampled, - cuba_iter the current iteration number. The function author may choose to use these in any appropriate way or ignore them altogether.
nComp	The number of components of f, default 1, bears no relation to the dimension of the hypercube over which integration is performed.
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.
relTol	The maximum tolerance, default 1e-5.
absTol	the absolute tolerance, default 1e-12.
minEval	the minimum number of function evaluations required
maxEval	The maximum number of function evaluations needed, default 10^6. Note that the actual number of function evaluations performed is only approximately guaranteed not to exceed this number.
flags	flags governing the integration. The list here is exhaustive to keep the documentation and invocation uniform, but not all flags may be used for a particular method as noted below. List components:

verbose encodes the verbosity level, from 0 (default) to 3. Level 0 does not print any output, level 1 prints reasonable information on the progress of the integration, level 2 also echoes the input parameters, and level 3 further prints the subregion results.

final when 0, all sets of samples collected on a subregion during the various iterations or phases contribute to the final result. When 1, only the last (largest) set of samples is used in the final result.

smooth Applies to Suave and Vegas only. When 0, apply additional smoothing to the importance function, this moderately improves convergence for many integrands. When 1, use the importance function without smoothing, this should be chosen if the integrand has sharp edges.

`keep_state` when nonzero, retain state file if argument `stateFile` is non-null, else delete `stateFile` if specified.

`load_state` Applies to Vegas only. Reset the integrator's state even if a state file is present, i.e. keep only the grid. Together with `keep_state` this allows a grid adapted by one integration to be used for another integrand.

`level` applies only to Divonne, Suave and Vegas. When `0`, Mersenne Twister random numbers are used. When nonzero Ranlux random numbers are used, except when `rngSeed` is zero which forces use of Sobol quasi-random numbers. Ranlux implements Marsaglia and Zaman's 24-bit RCARRY algorithm with generation period p, i.e. for every 24 generated numbers used, another p-24 are skipped. The luxury level for the Ranlux generator may be encoded in `level` as follows:

Level 1 (p = 48) gives very long period, passes the gap test but fails spectral test

Level 2 (p = 97) passes all known tests, but theoretically still defective

Level 3 (p = 223) any theoretically possible correlations have very small chance of being observed

Level 4 (p = 389) highest possible luxury, all 24 bits chaotic

Levels 5-23 default to 3, values above 24 directly specify the period p.

Note that Ranlux's original level 0, (mis)used for selecting Mersenne Twister in Cuba, is equivalent to `level` = 24.

`rngSeed` seed, default 0, for the random number generator. Note the articulation with `level` settings for `flag`

`nVec` the number of vectorization points, default 1, but can be set to an integer > 1 for vectorization, for example, 1024 and the function f above needs to handle the vector of points appropriately. See vignette examples.

`nNew` the number of new integrand evaluations in each subdivision.

`nMin` the minimum number of samples a former pass must contribute to a subregion to be considered in that region's compound integral value. Increasing nmin may reduce jumps in the χ^2 value.

`flatness` the parameter p, or the type of norm used to compute the fluctuation of a sample. This determines how prominently "outliers," i.e. individual samples with a large fluctuation, figure in the total fluctuation, which in turn determines how a region is split up. As suggested by its name, flatness should be chosen large for "flat" integrands and small for "volatile" integrands with high peaks. Note that since flatness appears in the exponent, one should not use too large values (say, no more than a few hundred) lest terms be truncated internally to prevent overflow.

`stateFile` the name of an external file. Vegas can store its entire internal state (i.e. all the information to resume an interrupted integration) in an external file. The state file is updated after every iteration. If, on a subsequent invocation, Vegas finds a file of the specified name, it loads the internal state and continues from the point it left off. Needless to say, using an existing state file with a different integrand generally leads to wrong results. Once the integration finishes successfully, i.e. the prescribed accuracy is attained, the state file is removed. This feature is useful mainly to define 'check-points' in long-running integrations from which the calculation can be restarted.

Details

See details in the documentation.

Value

A list with components:

nregions the actual number of subregions needed

neval the actual number of integrand evaluations needed

returnCode if zero, the desired accuracy was reached, if -1, dimension out of range, if 1, the accuracy goal was not met within the allowed maximum number of integrand evaluations.

integral vector of length `nComp`; the integral of `integrand` over the hypercube

error vector of length `nComp`; the presumed absolute error of `integral`

prob vector of length `nComp`; the χ^2-probability (not the χ^2-value itself!) that `error` is not a reliable estimate of the true integration error.

References

T. Hahn (2005) CUBA-a library for multidimensional numerical integration. *Computer Physics Communications*, **168**, 78-95.

See Also

`cuhre`, `divonne`, `vegas`

Examples

```
integrand <- function(arg) {
  x <- arg[1]
  y <- arg[2]
  z <- arg[3]
  ff <- sin(x)*cos(y)*exp(z);
return(ff)
} # end integrand
suave(integrand, lowerLimit = rep(0, 3), upperLimit = rep(1, 3),
             relTol=1e-3,  absTol=1e-12,
             flags=list(verbose=2, final=0))
```

vegas *Integration by a Monte Carlo Algorithm*

Description

Implement a Monte Carlo algorithm for multidimensional numerical integration. This algorithm uses importance sampling as a variance-reduction technique. Vegas iteratively builds up a piecewise constant weight function, represented on a rectangular grid. Each iteration consists of a sampling step followed by a refinement of the grid.

Usage

```
vegas(f, nComp = 1L, lowerLimit, upperLimit, ..., relTol = 1e-05,
  absTol = 1e-12, minEval = 0L, maxEval = 10^6,
  flags = list(verbose = 0L, final = 1L, smooth = 0L, keep_state = 0L,
  load_state = 0L, level = 0L), rngSeed = 12345L, nVec = 1L,
  nStart = 1000L, nIncrease = 500L, nBatch = 1000L, gridNo = 0L,
  stateFile = NULL)
```

Arguments

f	The function (integrand) to be integrated as in cuhre. Optionally, the function can take two additional arguments in addition to the variable being integrated: - cuba_weight which is the weight of the point being sampled, - cuba_iter the current iteration number. The function author may choose to use these in any appropriate way or ignore them altogether.
nComp	The number of components of f, default 1, bears no relation to the dimension of the hypercube over which integration is performed.
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.
relTol	The maximum tolerance, default 1e-5.
absTol	the absolute tolerance, default 1e-12.
minEval	the minimum number of function evaluations required
maxEval	The maximum number of function evaluations needed, default 10^6. Note that the actual number of function evaluations performed is only approximately guaranteed not to exceed this number.
flags	flags governing the integration. The list here is exhaustive to keep the documentation and invocation uniform, but not all flags may be used for a particular method as noted below. List components:
	verbose encodes the verbosity level, from 0 (default) to 3. Level 0 does not print any output, level 1 prints reasonable information on the progress of the integration, level 2 also echoes the input parameters, and level 3 further prints the subregion results.

`final` when 0, all sets of samples collected on a subregion during the various iterations or phases contribute to the final result. When 1, only the last (largest) set of samples is used in the final result.

`smooth` Applies to Suave and Vegas only. When 0, apply additional smoothing to the importance function, this moderately improves convergence for many integrands. When 1, use the importance function without smoothing, this should be chosen if the integrand has sharp edges.

`keep_state` when nonzero, retain state file if argument `stateFile` is non-null, else delete `stateFile` if specified.

`load_state` Applies to Vegas only. Reset the integrator's state even if a state file is present, i.e. keep only the grid. Together with `keep_state` this allows a grid adapted by one integration to be used for another integrand.

`level` applies only to Divonne, Suave and Vegas. When `0`, Mersenne Twister random numbers are used. When nonzero Ranlux random numbers are used, except when `rngSeed` is zero which forces use of Sobol quasi-random numbers. Ranlux implements Marsaglia and Zaman's 24-bit RCARRY algorithm with generation period p, i.e. for every 24 generated numbers used, another p-24 are skipped. The luxury level for the Ranlux generator may be encoded in `level` as follows:

Level 1 (p = 48) gives very long period, passes the gap test but fails spectral test

Level 2 (p = 97) passes all known tests, but theoretically still defective

Level 3 (p = 223) any theoretically possible correlations have very small chance of being observed

Level 4 (p = 389) highest possible luxury, all 24 bits chaotic

Levels 5-23 default to 3, values above 24 directly specify the period p.

Note that Ranlux's original level 0, (mis)used for selecting Mersenne Twister in Cuba, is equivalent to `level` = 24.

`rngSeed` seed, default 0, for the random number generator. Note the articulation with `level` settings for `flag`

`nVec` the number of vectorization points, default 1, but can be set to an integer > 1 for vectorization, for example, 1024 and the function f above needs to handle the vector of points appropriately. See vignette examples.

`nStart` the number of integrand evaluations per iteration to start with.

`nIncrease` the increase in the number of integrand evaluations per iteration. The j-th iteration evaluates the integrand at nStart+(j-1)*nincrease points.

`nBatch` Vegas samples points not all at once, but in batches of a predetermined size, to avoid excessive memory consumption. `nbatch` is the number of points sampled in each batch. Tuning this number should usually not be necessary as performance is affected significantly only as far as the batch of samples fits into the CPU cache.

`gridNo` an integer. Vegas may accelerate convergence to keep the grid accumulated during one integration for the next one, if the integrands are reasonably similar to each other. Vegas maintains an internal table with space for ten grids for this purpose. If `gridno` is a number between 1 and 10, the grid is not discarded

at the end of the integration, but stored in the respective slot of the table for a future invocation. The grid is only re-used if the dimension of the subsequent integration is the same as the one it originates from. In repeated invocations it may become necessary to flush a slot in memory. In this case the negative of the grid number should be set. Vegas will then start with a new grid and also restore the grid number to its positive value, such that at the end of the integration the grid is again stored in the indicated slot.

`stateFile` the name of an external file. Vegas can store its entire internal state (i.e. all the information to resume an interrupted integration) in an external file. The state file is updated after every iteration. If, on a subsequent invocation, Vegas finds a file of the specified name, it loads the internal state and continues from the point it left off. Needless to say, using an existing state file with a different integrand generally leads to wrong results. Once the integration finishes successfully, i.e. the prescribed accuracy is attained, the state file is removed. This feature is useful mainly to define 'check-points' in long-running integrations from which the calculation can be restarted.

Details

See details in the documentation.

Value

A list with components:

nregions the actual number of subregions needed

neval the actual number of integrand evaluations needed

returnCode if zero, the desired accuracy was reached, if -1, dimension out of range, if 1, the accuracy goal was not met within the allowed maximum number of integrand evaluations.

integral vector of length `nComp`; the integral of `integrand` over the hypercube

error vector of length `nComp`; the presumed absolute error of `integral`

prob vector of length `nComp`; the χ^2-probability (not the χ^2-value itself!) that `error` is not a reliable estimate of the true integration error.

References

G. P. Lepage (1978) A new algorithm for adaptive multidimensional integration. *J. Comput. Phys.*, **27**, 192-210.

G. P. Lepage (1980) VEGAS - An adaptive multi-dimensional integration program. Research Report CLNS-80/447. Cornell University, Ithaca, N.-Y.

T. Hahn (2005) CUBA-a library for multidimensional numerical integration. *Computer Physics Communications*, **168**, 78-95.

See Also

`cuhre`, `suave`, `divonne`

Examples

```
integrand <- function(arg, weight) {
  x <- arg[1]
  y <- arg[2]
  z <- arg[3]
  ff <- sin(x)*cos(y)*exp(z);
return(ff)
} # end integrand
vegas(integrand, lowerLimit = rep(0, 3), upperLimit = rep(1, 3),
             relTol=1e-3,  absTol=1e-12,
             flags=list(verbose=2, final=0))
```

Index

∗Topic **math**
cuhre, 4
divonne, 7
hcubature, 12
suave, 18
vegas, 22
∗Topic **package**
cubature-package, 2

adaptIntegrate *(*hcubature*)*, 12

cubature *(*cubature-package*)*, 2
cubature-package, 2
cubintegrate, 2
cuhre, *4*, 4, *8*, *11*, *19*, *21*, *22*, *24*

default_args, *2*, *4*, 7
divonne, *4*, *6*, 7, *21*, *24*

hcubature, *4*, 12

pcubature, *4*
pcubature *(*hcubature*)*, 12

suave, *4*, *6*, *11*, 18, *24*

vegas, *4*, *6*, *11*, *19*, *21*, 22