

# The adapt Package

February 16, 2008

**Title** adapt – multidimensional numerical integration

**Version** 1.0-4

**Author** FORTRAN by Alan Genz, S by Mike Meyer, R by Thomas Lumley and Martin Maechler

**Description** Adaptive Quadrature in up to 20 dimensions

**Depends**

**License** Unclear (Fortran) – code in Statlib’s ./S/adapt

**Maintainer** Thomas Lumley <tlumley@u.washington.edu>

## R topics documented:

adapt . . . . . 1

**Index** . . . . . 4

---

adapt *Adaptive Numerical Integration in 2–20 Dimensions*

---

## Description

Integrates a scalar function over a multidimensional rectangle, i.e., computes

$$\int_l^u \text{functn}(t) d^n t$$

where  $l$  =lower,  $u$  =upper and  $n$  =ndim. Infinite rectangles are not allowed, and ndim must be between 2 and 20.

## Usage

```
adapt(ndim, lower, upper, minpts = 100, maxpts = NULL, functn, eps = 0.01, ...)
```

**Arguments**

|                     |   |
|---------------------|---|
| <code>ndim</code>   | the dimension of the integral, and i.e. number  |
| <code>lower</code>  | vector of at least length <code>ndim</code> of the lower bounds on the integral.  |
| <code>upper</code>  | vector of at least length <code>ndim</code> of the upper bounds on the integral.  |
| <code>minpts</code> | the minimum number of function evaluations.   |
| <code>maxpts</code> | the maximum number of function evaluations or <code>NULL</code> per default, see <i>Details</i> .   |
| <code>functn</code> | an <b>R</b> function which should take a single vector argument and possibly some parameters and return the function value at that point. <code>functn</code> must return a single numeric value. |
| <code>eps</code>    | the desired accuracy for the relative error.  |
| <code>...</code>    | other parameters to be passed to <code>functn</code>  |

**Details**

This is modified from Mike Meyer's *S* code. The functions just call A.C. Genz's fortran ADAPT subroutine to do all of the calculations. A work array is allocated within the C/Fortran code.

The Fortran function has been modified to use double precision, for compatibility with **R**. It only works in two or more dimensions; for one-dimensional integrals use the `integrate` function in the base package.

Setting `maxpts` to `NULL` asks the function to keep doubling `maxpts` (starting at  $\max(\text{minpts}, 500, r(\text{ndim}))$ ) until the desired precision is achieved or **R** runs out of memory. Note that the necessary number of evaluations typically grows exponentially with the dimension `ndim`, and the underlying code requires  $\text{maxpts} \geq r(\text{ndim})$  where  $r(d) = 2^d + 2d(d + 3) + 1$ .

**Value**

A list of `class` "integration" with components

|                     |  |
|---------------------|--|
| <code>value</code>  | the estimated integral   |
| <code>relerr</code> | the estimated relative error; $< \text{eps}$ argument if the algorithm converged properly.                       |
| <code>minpts</code> | the actual number of function evaluations  |
| <code>ifail</code>  | an error indicator. If <code>ifail</code> is not equal to 0, the function warns the user of the error condition. |

**See Also**

`integrate`

**Examples**

```
## Example of p - dimensional spherical normal distribution:
ir2pi <- 1/sqrt(2*pi)
fred <- function(z) { ir2pi^length(z) * exp(-0.5 * sum(z * z)) }

adapt(2, lo = c(-5,-5), up = c(5,5), functn = fred)
```

```
adapt(2, lo = c(-5,-5), up = c(5,5), functn = fred, eps = 1e-4)
adapt(2, lo = c(-5,-5), up = c(5,5), functn = fred, eps = 1e-6)
## adapt "sees" function ~= constantly 0 --> wrong result
adapt(2, lo = c(-9,-9), up = c(9,9), functn = fred)
## fix by using much finer initial grid:
adapt(2, lo = c(-9,-9), up = c(9,9), functn = fred, min = 1000)
adapt(2, lo = c(-9,-9), up = c(9,9), functn = fred, min = 1000, eps = 1e-6)

il <- print(integrate(dnorm, -2, 2))$value

## True values for the following example:
il ^ c(3,5)

for(p in c(3,5)) {
  cat("\np = ", p, "\n-----\n")
  f.lo <- rep(-2., p)
  f.up <- rep(+2., p)
  ## not enough evaluations:
  print(adapt(p, lo=f.lo, up=f.up, max=100*p, functn = fred))
  ## enough evaluations:
  print(adapt(p, lo=f.lo, up=f.up, max=10^p, functn = fred))
  ## no upper limit; p=3: 7465 points, ie 5 attempts (on an Athlon/gcc/g77):
  print(adapt(p, lo=f.lo, up=f.up, functn = fred, eps = 1e-5))
}
```

# Index

\*Topic **math**

  adapt, 1

\*Topic **utilities**

  adapt, 1

adapt, 1

class, 2

integrate, 2

print.integration(*adapt*), 1