### Package ‘round’

January 4, 2021

**Version** 0.20-0  
**Date**  2021-01-04  
**Title** Rounding to Decimal Digits  
**Description** Decimal rounding is non-trivial in binary arithmetic. ISO standard round to even is more rare than typically assumed as most decimal fractions are not exactly representable in binary. Our roundX() versions explore differences between current and potential future versions of round() in R. Further, provides (some partly related) C99 math lib functions not in base R.

**Imports** stats  
**Suggests** grDevices, graphics, Matrix, lattice, gmp, knitr, rmarkdown  
**License** AGPL (>= 3) | file LICENSE  
**Encoding** UTF-8  
**KeepSource** TRUE  
**VignetteBuilder** knitr  
**URL** https://gitlab.com/mmaechler/round/  
**BugReports** https://gitlab.com/mmaechler/round/-/issues

### R topics documented:

- mathC99........................................................................... 1  
- randI................................................................................. 3  
- roundX............................................................................. 4

**Index**  8

---

**mathC99**  
* C99 ’math’ Library Functions (where Not in Standard aka ’Base’ R)

### Description

Provides simple R versions of those C99 “math lib” / “libmath” / “libm” functions that are not (yet) in standard (aka ‘base’ R).
Usage

logB(x) # C's logb(x), numeric integer-valued "log2".
# R's logb() is defined as "log wrt base"
ilogb(x) # == logB(), but of *type* integer

fpclassify(x)
isnormal(x)
nearbyint(x)
signbit(x)

nextafter(x, y)
nexttoward(x, y)

Arguments

x, y numeric vector(s); will be recycled to common length.

Value

a numeric (double or integer) vector of the same (or recycled) length of x (and y where appropriate) with the values of <fn>(x) for the corresponding C99 libmath function <fn>.

Author(s)

Martin Maechler

References


See Also

sqrt, log, exp, Trig;
floor, ceiling, trunc;
is.finite, is.na.

Examples

x <- (1:20)*pi
stopifnot(ilogb (x) == logB (x), is.integer(ilogb(x)),
ilogb(-x) == logB(-x), is.double ( logB(x))))
cbind(x, "2^il(x)"= 2^logB(x), ilogb = ilogb(x), signbit = signbit(x),
fpclassify = fpclassify(x), isnormal = isnormal(x))

x <- c(2^-10:22, rexp(1000));
summary(x / 2^ilogb(x)) # in [1, 2) interval
stopifnot(nearbyint(x) == round(x))

nextafter(-0, +0)
nextafter(+0, 1)
nextafter(+0, -1)
nextafter(-Inf, -1)
nextafter(-Inf, 0)
Description

Create \( n \) random integer valued numbers all with a specified number of digits \( d \).

Usage

\[
\text{randI}(n, d)
\]

Arguments

\( n \) numeric sample size, i.e., \text{length}() of result.

\( d \) a positive integer, giving the exact number of digits the resulting numbers must have.

Details

This is based on \text{runif}() and not \text{sample}(), which for now also makes it less \text{R} version dependent.

Value

A numeric vector of length \( n \) of numbers \( N \) where each \( N \) has exactly \( d \) digits; equivalently,

\[
10^{d-1} \leq N_i < 10^d,
\]

and every \( N_i \) appears with the same probability \( 1/(9 \cdot 10^{d-1}) \)

Author(s)

Martin Maechler

See Also

Uniform random numbers \text{runif}; Random number generators, seeds, etc: \text{RNG}.

Examples

\[
\begin{align*}
\text{plot}( & \text{table(randI(1e6, 2)))} ; \text{abline(h = 1e6 / (9*10^(2 - 1)), lty=2, col="gray70")} \\
& \text{chisq.test(T2)} \ # \text{typically not at all significant} \\
& \text{T3 <- table(randI(1e6, 3))} \\
& \text{chisq.test(T3)} \\
& \text{stopifnot(exprs = \{} \\
& \quad \text{identical( 10:99 , as.integer(names(T2)))} \\
& \quad \text{identical(100:999, as.integer(names(T3)))} \\
& \})
\end{align*}
\]
roundX (x, digits, version = roundVersions, trace = 0)
roundAll(x, digits, versions = roundVersions)
round_r3(x, d, info=FALSE, check=TRUE)
roundVersions # "sprintf" "r0.C" "r1.C" "r1a.C" "r2.C" "r3.C" "r3d.C" "r3"

Arguments
x numeric vector
digits, d integer number (for d) or numeric vector.
version a character string specifying the version of rounding. Must match roundVersions (via match.arg).
trace integer; if positive, the corresponding computations should be “traced” (possibly proportionally to the value of trace); currently only implemented for version = "r3.C".
versions a character vector, a subset of roundVersions.
info logical specifying if round_r3(*) should result in a list with components "r": the rounded x,
"D": the difference \((xu - x) - (x - xd)\), where xd and xu are the round down and
up versions of x,
"e": the modulo-2 remainder of \(\text{floor}(x \times 10^d)\), which determines rounding
to even (only) in case \(D = 0\).
check logical indicating if x and digits should be checked for validity. Is set to FALSE when used in roundX() (or roundAll), as the checks happen before round_r3() is called.

Details
Rounding to decimal digits is non-trivial in binary arithmetic. ISO standard “round to even”, see round()’s (help page), is more rare than typically assumed as most decimal fractions are not exactly representable in binary double precision numbers.

Decimal rounding is well defined when digits = 0, and calls the (C99 standard) C library function nearbyint() (which provide in this package as well, for completeness): round(x) is (R level) equivalent to round(x, digits = 0) and is also equivalent to (R and C level) nearbyint(x) which is defined to return the closest integer number (as double) and in the case of “doubt”, where both integer number neighbours are of the same distance, i.e., distance 0.5 the famous “round to even” strategy is used, such that, e.g., round(0:7 + 0.5) == c(0,2,2,4,4,6,6,8).
The following strategy / algorithms are used for the different `roundVersions`; note that we only consider the crucial case digits > 0 in the following description:

"sprintf": diverts the operation to `sprintf("%.f", digits, x)` which in turn diverts to the corresponding C library function `sprintf()`; consequently may be platform dependent (though we have not yet seen differences from what we get by the most widely used GNU ‘glibc’ library, [https://www.gnu.org/software/libc/](https://www.gnu.org/software/libc/)). This version does not work with negative digits, returning NA with a warning there.

"r0.C": a (too much) simplified version of R’s "r1.C", just skipping the whole integer part computations; this was the first patch proposal in R-bugs’ report PR#17668. However, this completely breaks down in extreme cases.

"r1.C": the version of `round()` as in R 3.6.2 and earlier. It first removes the integer part(s) of x, then rounds and re-adds the integer part.

"r1a.C": a slightly improved version of "r1.C", notably for |digits| > 308.

"r2.C": the version of `round()` as added to ‘R-devel’ (the development version of R) with ‘svn’ revision . . . . . . . It does not remove and re-add the integer part(s) of x but ensures that no unnecessary overflow to +/-Inf or underflow to 0 happens when numbers are multiplied and divided by 10^d.

"r2a.C": a slightly improved version of "r2.C", notably for large negative digits.

"r3": (R level) implementation of “correct” rounding, rounding to the nearest double precision number (with “round to even” in case of equal distance) as seen in the function definition of `round_r3()`. Note that info=TRUE is only applied when when the digits d fulfill |d| <= 308.

"r3.C": a C translation of "r3", using long double for intermediate computations which is particularly convenient for digits 308 < d < 324 as overflow is not a possible then.

"r3d.C": a version of "r3.C", only using double precision, and hence typically fast and less platform dependent, and also more often identical to "r3".

**Value**

`roundX()` returns a numeric vector (of length of recycled x and digits, i.e., typically (when digits is of length one) of length(x).

`round_r3()` is the workhorse of `roundX(...,version = "r3")`; it vectorizes in x but needs `length(d) == 1`.

`round Versions` is a character vector of the versions available for `roundX()`.

`roundAll()` applies `roundX()` for all versions, returning a matrix if one of x or digits is not of length one.

**Author(s)**

Martin Maechler (R Core for version "r1.C")

**References**

Wikipedia, Rounding, notably "Round half to even": [https://en.wikipedia.org/wiki/Rounding#Round_half_to_even](https://en.wikipedia.org/wiki/Rounding#Round_half_to_even)

**See Also**

`round`, also `signif` which is relatively sophisticated (also by code from M.M.).
Examples

roundVersions

round (55.55, 1)
roundX(55.55, 1, "r3")

## round() with all roundVersions; quite simple (w/ recycling!)
roundAll # shows the function’s definition

roundAll(55.55, 1)
roundAll(55.555, 2)
roundAll(55.5555, 3)
roundAll(55.55555, 4)
roundAll(55.555555, 5)
roundAll(55.5555555, 6)

## other "controversial" cases
rEx <- cbind( x = c(10.7775, 12.345, 9.18665), digits = c( 3 , 2 , 4 ))

resEx <- matrix(, length(roundVersions), nrow(rEx),
               dimnames = list(roundVersions, as.character(rEx[,"x"])))
for(i in 1:nrow(rEx))
  resEx[,i] <- roundAll(rEx[[i,"x"]], digits = rEx[[i,"digits"]])

resEx # r0.C & r2* agree and differ from the r1*;
# "r3*" is close to "r2*" but not for 12.345
## The parts of "r3":
r3rE <- sapply(1:nrow(rEx), function(i)
  round_r3(rEx[[i,"x"]], rEx[[i,"digits"]], info=TRUE))
colnames(r3rE) <- sapply(rEx[,"x"], format)
r3rE # rounding to even when D=0, but not when D < 0

## "Deterministic" Simulation - few digits only:
long <- interactive() # save time/memory e.g. when checking
I <- if(long) 0:9999 else 0:999
Ix <- I + 0.5
ndI <- 1L + as.integer(log10(pmax(1,I))) # number of (decimal) digits of I
nd2 <- outer(ndI, if(long) -3:4 else -2:3,
               \ + \)
x <- c(t( Ix / (10^nd2) ))
nd2 <- c(t( nd2 ))
x <- x [nd2 > 0]
nd2 <- nd2[nd2 > 0]
rx <- roundAll(x, digits = nd2)

formatF <- function(.) format(. , scientific=FALSE, drop0trailing=TRUE)
rownames(rx) <- formatF(x)
options(width = 123)
noquote(cbind(d = nd2, formatF(rx))[1:140,])
## -- The first cases already show a diverse picture; sprintf() a bit as outlier

## Error, assuming "r3" to be best, as it *does* really go to nearest:
Err <- rx - rx[, "r3"]
## careful : allowing small "noise" differences:
tErr <- abs(Err) > 1e-3* 10^-nd2 # "truly" differing from "r3"
colSums(tErr) # -- > old R "r1*" is best here, then sprintf (among non-r3):
For F30 Linux 64-bit (gcc), and this selection of cases, r0+r2 are worst; r1 is best

<table>
<thead>
<tr>
<th>Case</th>
<th>Format</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
<th>Value 6</th>
<th>Value 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0.C</td>
<td>C</td>
<td>15559</td>
<td>1167</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r1.C</td>
<td>C</td>
<td>19778</td>
<td>1457</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r1a.C</td>
<td>C</td>
<td>14078</td>
<td>1290</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r2.C</td>
<td>C</td>
<td>19778</td>
<td>1457</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r2a.C</td>
<td>C</td>
<td>19778</td>
<td>1457</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r3.C</td>
<td>C</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r3d.C</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r3</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

if(long) { # Q: where does "r3.C" differ from "r3"? A: in 10 cases; 8 "real"

i3D <- which(Err[, "r3.C"] != 0)
print(cbind(d = nd2[i3D], formatF(rx[i3D,])), quote=FALSE)
print.table(zapsmall(Err[i3D,]), zero.print = ".")# differences (not very small ones!)
}

Visualization of error happening (FIXME: do zapsmall()-like not count "noise")
cumErr <- apply(tErr[, colnames(rx) != "r3"], 2L, cumsum)
matPm <- function(y) {
  matplot(y=, type = "l", lwd = 2, xlab = "i", ylab = deparse(substitute(y)))
  abline(h = 0, lty=2, col="gray")
  legend("topleft", legend = setdiff(roundVersions, "r3"),
         col = 1:6, lty = 1:5, lwd = 2, bty = "n")
}
matPm(head(cumErr, 100)) # sprintf seems worst
matPm(head(cumErr, 250)) # now r0+2 is worst, sprintf best
matPm(head(cumErr, 1000)) # now sprintf clearly worst again
matPm(head(cumErr, 2000)) # 0r/r2 best sprintf catching up
if(long) {
  matPm(head(cumErr, 5000)) # now sprintf clearly worst again
  matPm(head(cumErr, 100000)) # now r0+2 is worst, r1 best
  matPm( cumErr )
}

same_cols <- function(m) all(m == m[,1])
stopifnot(same_cols(Err[, c("r0.C", "r2.C", "r2a.C")]))
stopifnot(same_cols(Err[, c("r1.C", "r1a.C")]))
if(FALSE) # not in 'long' case, see above
  stopifnot(same_cols(Err[, c("r3", "r3.C", "r3d.C")]))

sp <- search()
if(long & require("Matrix") { # fails, bug in lattice? useRaster = !dev.interactive(TRUE) & (nrow(m) >= 2^12),
  showSp <- function(m) print(image(as(m, "sparseMatrix"), aspect = 4,
                                 fails, bug in lattice? useRaster = !dev.interactive(TRUE) & (nrow(m) >= 2^12),
                                 border.col = if(nrow(m) < 1e3) adjustcolor(1, 1/2) else NA))
  showSp(head(Err, 100))
  showSp(head(Err, 1000))
  showSp(Err)
  showSp(Err != 0) # B&W version ..
  if(!any(sp == "package:Matrix") detach("package:Matrix")
}

More digits random sample simulation tend go against "sprintf";
# see ../tests/ and also the vignette
Index

* Rounding
  roundX, 4
* arithmetic
  mathC99, 1
* arith
  roundX, 4
* distribution
  randI, 3
* math
  mathC99, 1
  ceiling, 2
  character, 4, 5
  double, 2, 4
  exp, 2
  floor, 2
  fpclassify (mathC99), 1
  ilogb (mathC99), 1
  Inf, 5
  integer, 2
  is.finite, 2
  is.na, 2
  isnormal (mathC99), 1
  length, 3
  list, 4
  log, 2
  logB (mathC99), 1
  logical, 4
  match.arg, 4
  mathC99, 1
  NA, 5
  nearbyint, 4
  nearbyint (mathC99), 1
  nextafter (mathC99), 1
  nexttoward (mathC99), 1
  numeric, 2, 3
  randI, 3
  RNG, 3
  round, 4, 5
  round_r3, 5
  round_r3 (roundX), 4
  roundAll (roundX), 4
  roundVersions (roundX), 4
  roundX, 4
  runif, 3
  sample, 3
  signbit (mathC99), 1
  signif, 5
  sprintf, 5
  sqrt, 2
  Trig, 2
  trunc, 2
  warning, 5