Package ‘gmp’

October 1, 2020

Version 0.6-1

Date 2020-10-01

Title Multiple Precision Arithmetic

Author Antoine Lucas, Immanuel Scholz, Rainer Boehme <rb-gmp@reflex-studio.de>,
Sylvain Jasson <Sylvain.Jasson@inrae.fr>,
Martin Maechler <maechler@stat.math.ethz.ch>

Maintainer Antoine Lucas <antoinelucas@gmail.com>

Description Multiple Precision Arithmetic (big integers and rationals,
prime number tests, matrix computation), ``arithmetic without limitations''
using the C library GMP (GNU Multiple Precision Arithmetic).

Depends R (>= 3.5.0)

Imports methods

Suggests Rmpfr, MASS, round

SystemRequirements gmp (>= 4.2.3)

License GPL (>= 2)

BuildResaveData no

LazyDataNote not available, as we use data/*.R *and* our classes

NeedsCompilation yes

URL https://forgemia.inra.fr/sylvain.jasson/gmp

Repository CRAN

Date/Publication 2020-10-01 14:30:02 UTC

R topics documented:

apply 2
asNumeric 3
BernoulliQ 4
Bigq 5
bigq 6
Bigq operators 8

bigz . . . 9
bigz operators . . . 12
cumsum . . . 14
extract . . . 16
Extremes . . . 17
factorialZ . . . 18
factorization . . . 19
formatN . . . 20
frexpZ . . . 21
gcd.bigz . . . 22
gcdex . . . 23
gmp-ifiworkarounds . . . 24
gmp.utils . . . 24
is.whole . . . 25
isprime . . . 26
lucnum . . . 27
matrix . . . 28
modulus . . . 30
nextprime . . . 31
Oakley . . . 32
powm . . . 33
Random . . . 34
Relational Operator . . . 35
roundQ . . . 35
sizeinbase . . . 37
solve.bigz . . . 38
Stirling . . . 39

Index **42**

`apply` *Apply Functions Over Matrix Margins (Rows or Columns)*

Description

These are S3 `methods` for `apply()` which we re-export as S3 generic function. They “overload” the `apply()` function for big rationals (`"bigq"`) and big integers (`"bigz"`).

Usage

```
## S3 method for class 'bigz'
apply(X, MARGIN, FUN, ...)
## S3 method for class 'bigq'
apply(X, MARGIN, FUN, ...)
```

Arguments

`X`	a matrix of class bigz or bigq, see e.g., `matrix.bigz`.
`MARGIN`	1: apply function to rows; 2: apply function to columns
`FUN`	`function` to be applied
`...`	(optional) extra arguments for `FUN()`, as e.g., in `lapply`.

Value

The `bigz` and `bigq` methods return a vector of class `"bigz"` or `"bigq"`, respectively.

Author(s)

Antoine Lucas

See Also

`apply`; `lapply` is used by our `apply()` method.

Examples

```
x <- as.bigz(matrix(1:12,3))
apply(x,1,min)
apply(x,2,max)

x <- as.bigq(x ^ 3, d = (x + 3)^2)
apply(x,1, min)
apply(x,2, sum)
## now use the "..." to pass  na.rm=TRUE :
x[2,3] <- NA
apply(x,1, sum)
apply(x,1, sum, na.rm = TRUE)
```

---

asNumeric — *Coerce to 'numeric', not Loosing Dimensions*

Description

a number-like object is coerced to type (typeof) `"numeric"`, keeping `dim` (and maybe `dimnames`) when present.

Usage

```
asNumeric(x)
```

Arguments

`x`	a "number-like" object, e.g., big integer (`bigz`), or `mpfr`, notably including matrices and arrays of such numbers.

Value

an R object of type (typeof) "numeric", a matrix or array if x had non-NULL dimension dim().

Methods

signature(x = "ANY") the default method, which is the identity for numeric array.

signature(x = "bigq") the method for big rationals.

signature(x = "bigq") the method for big integers.

Note that package **Rmpfr** provides methods for its own number-like objects.

Author(s)

Martin Maechler

See Also

as.numeric coerces to both "numeric" and to a vector, whereas asNumeric() should keep dim (and other) attributes.

Examples

```
m <- matrix(1:6, 2,3)
stopifnot(identical(m, asNumeric(m)))# remains matrix

(M <- as.bigz(m) / 5) ##-> "bigq" matrix
asNumeric(M) # numeric matrix
stopifnot(all.equal(asNumeric(M), m/5))
```

BernoulliQ *Exact Bernoulli Numbers*

Description

Return the n-th Bernoulli number B_n, (or B_n^+, see the reference), where $B_1 = +\frac{1}{2}$.

Usage

```
BernoulliQ(n, verbose = getOption("verbose", FALSE))
```

Arguments

n	integer *vector*, $n \geq 0$.
verbose	logical indicating if computation should be traced.

Value

a big rational (class "bigq") vector of the Bernoulli numbers B_n.

Author(s)

Martin Maechler

References

https://en.wikipedia.org/wiki/Bernoulli_number

See Also

`Bernoulli` in **Rmpfr** in arbitrary precision via Riemann's ζ function.

`Bern(n)` in **DPQ** uses standard (double precision) R arithmetic for the n-th Bernoulli number.

Examples

```
(Bn0.10 <- BernoulliQ(0:10))
```

`Bigq` *Relational Operators*

Description

Binary operators which allow the comparison of values in atomic vectors.

Usage

```
## S3 method for class 'bigq'
sign(x)

## S3 method for class 'bigq'
e1 < e2
## S3 method for class 'bigq'
e1 <= e2
## S3 method for class 'bigq'
e1 == e2
## S3 method for class 'bigq'
e1 >= e2
## S3 method for class 'bigq'
e1 > e2
## S3 method for class 'bigq'
e1 != e2
```

Arguments

`x, e1, e2`	Object or vector of class `bigq`

Examples

```
x <- as.bigq(8000,21)
x  < 2 * x
```

---

bigq *Large sized rationals*

Description

Class "bigq" encodes rationals encoded as ratios of arbitrary large integers (via GMP). A simple S3 class (internally a raw vector), it has been registered as formal (S4) class (via setOldClass), too.

Usage

```
as.bigq(n, d = 1)
## S3 method for class 'bigq'
as.character(x, b=10,...)
## S3 method for class 'bigq'
as.double(x,...)
as.bigz.bigq(a, mod=NA)
is.bigq(x)
## S3 method for class 'bigq'
is.na(x)
## S3 method for class 'bigq'
print(x, quote=FALSE, initLine = TRUE, ...)
denominator(x)
numerator(x)
NA_bigq_
c_bigq(L)
```

Arguments

n,d	either integer, numeric or string value (String value: either starting with 0x for hexadecimal, 0b for binary or without prefix for decimal values. Any format error results in 0). n stands for numerator, d for denominator.
a	an element of class "bigq"
mod	optional modulus to convert into biginteger
x	a "rational number" (vector), of class "bigq".
b	base: from 2 to 36
...	additional arguments passed to methods
quote	(for printing:) logical indicating if the numbers should be quoted (as characters are); the default used to be TRUE (implicitly) till 2011.

initLine
: (for printing:) logical indicating if an **init**ial line (with the class and length or dimension) should be printed.

L
: a list where each element contains "bigq" numbers, for c_bigq(), this allows something like an sapply() for "bigq" vectors, see sapplyQ() in the examples below.

Details

as.bigq(x) when x is numeric (aka double precision) calls the 'GMP' function mpq_set_d() which is documented to be *exact* (every finite double precision number is a rational number).

as.bigz.bigq() returns the smallest integers not less than the corresponding rationals bigq.

NA_bigq_ is computed on package load time as as.bigq(NA).

Value

An R object of (S3) class "bigq" representing the parameter value.

Author(s)

Antoine Lucas

Examples

```
x <- as.bigq(21,6)
x
# 7 / 2
# Wow ! result is simplified.

y <- as.bigq(5,3)

# addition works !
x + y

# You can even try multiplication, division...
x * y / 13

# and, since May 2012,
x ^ 20
stopifnot(is.bigq(x), is.bigq(x + y),
  x ^ 20 == as.bigz(7)^20 / 2^20)

# convert to string, double
as.character(x)
as.double(x)

stopifnot( is.na(NA_bigq_) )

# Depict the "S4-class" bigq, i.e., the formal (S4) methods:
if(require("Rmpfr")) # mostly interesting there
  showMethods(class="bigq")
```

```
# an  sapply() version that works for big rationals "bigq":
sapplyQ <- function(X, FUN, ...) c_bigq(lapply(X, FUN, ...))

# dummy example showing it works (here):
qq <- as.bigq(1, 1:999)
q1 <- sapplyQ(qq, function(q) q^2)
stopifnot( identical(q1, qq^2) )
```

Bigq operators   *Basic arithmetic operators for large rationals*

Description

Addition, subtraction, multiplication, division, and absolute value for large rationals, i.e. "bigq" class R objects.

Usage

```
add.bigq(e1, e2)
## S3 method for class 'bigq'
e1 + e2

sub.bigq(e1, e2=NULL)
## S3 method for class 'bigq'
e1 - e2

mul.bigq(e1, e2)
## S3 method for class 'bigq'
e1 * e2

div.bigq(e1, e2)
## S3 method for class 'bigq'
e1 / e2

## S3 method for class 'bigq'
e1 ^ e2

## S3 method for class 'bigq'
abs(x)
```

Arguments

e1,e2, x	of class "bigq", or (e1 and e2) integer or string from an integer

Details

Operators can be use directly when the objects are of class "bigq": a + b, a * b, etc, and a ^ n, where n must be coercable to a biginteger ("bigz").

Value

A bigq class representing the result of the arithmetic operation.

Author(s)

Immanuel Scholz and Antoine Lucas

Examples

```
## 1/3 + 1 = 4/3 :
as.bigq(1,3) + 1

r <- as.bigq(12, 47)
stopifnot(r ^ 3 == r*r*r)
```

---

bigz *Large Sized Integer Values*

Description

Class "bigz" encodes arbitrarily large integers (via GMP). A simple S3 class (internally a `raw` vector), it has been registered as formal (S4) class (via `setOldClass`), too.

Usage

```
as.bigz(a, mod = NA)
NA_bigz_
## S3 method for class 'bigz'
as.character(x, b = 10, ...)

is.bigz(x)
## S3 method for class 'bigz'
is.na(x)
## S3 method for class 'bigz'
print(x, quote=FALSE, initLine = is.null(modulus(x)), ...)
c_bigz(L)
```

Arguments

`a`	either `integer`, `numeric` (i.e., `double`) or `character` vector. If character: the strings either start with `0x` for hexadecimal, `0b` for binary, `0` for octal, or without a `0*` prefix for decimal values. Formatting errors are signalled as with `stop`.
`b`	base: from 2 to 36
`x`	a "big integer number" (vector), of class "bigz".
`...`	additional arguments passed to methods

`mod` an integer, numeric, string or bigz of the internal modulus, see below.

`quote` (for printing:) logical indicating if the numbers should be quoted (as characters are); the default used to be `TRUE` (implicitly) till 2011.

`initLine` (for printing:) logical indicating if an **init**ial line (with the class and length or dimension) should be printed. The default prints it for those cases where the class is not easily discernable from the print output.

`L` a `list` where each element contains `"bigz"` numbers, for `c_bigz()`, this allows something like an `sapply()` for `"bigz"` vectors, see `sapplyZ()` in the examples.

Details

Bigz's are integers of arbitrary, but given length (means: only restricted by the host memory). Basic arithmetic operations can be performed on bigzs as addition, subtraction, multiplication, division, modulation (remainder of division), power, multiplicative inverse, calculating of the greatest common divisor, test whether the integer is prime and other operations needed when performing standard cryptographic operations.

For a review of basic arithmetics, see `add.bigz`.

Comparison are supported, i.e., `"=="`, `"!="`, `"<"`, `"<="`, `">"`, and `">="`.

`NA_bigz_` is computed on package load time as `as.bigz(NA)`.

Objects of class `"bigz"` may have a "modulus", accessible via `modulus()`, currently as an attribute `mod`. When the object has such a modulus m, arithmetic is performed *"modulo m"*, mathematically "within the ring Z/mZ". For many operations, this means

```
result <- mod.bigz(result, m)  ## == result %% m
```

is called after performing the arithmetic operation and the result will have the attribute `mod` set accordingly. This however does not apply, e.g., for `/`, where $a/b := ab^{-1}$ and b^{-1} is the *multiplicate inverse* of b with respect to ring arithmetic, or `NA` with a warning when the inverse does not exist. The warning can be turned off via `options("gmp:warnModMismatch" = FALSE)`

Powers of bigzs can only be performed, if either a modulus is going to be applied to the result bigz or if the exponent fits into an integer value. So, if you want to calculate a power in a finite group ("modulo c"), for large c do not use `a ^ b %% c`, but rather `as.bigz(a,c) ^ b`.

The following rules for the result's modulus apply when performing arithmetic operations on `bigz`s:

- If none of the operand has a modulus set, the result will not have a modulus.
- If both operands have a different modulus, the result will not have a modulus, except in case of `mod.bigz`, where the second operand's value is used.
- If only one of the operands has a modulus or both have a common (the same), it is set and used for the arithmetic operations, except in case of `mod.bigz`, where the second operand's value is used.

Value

An R object of (S3) class `"bigz"`, representing the argument (`x` or `a`).

Note

```
x <- as.bigz(1234567890123456789012345678901234567890)
```

will not work as R converts the number to a double, losing precision and only then convert to a "bigz" object.

Instead, use the syntax

```
x <- as.bigz("1234567890123456789012345678901234567890")
```

Author(s)

Immanuel Scholz

References

The GNU MP Library, see https://gmplib.org

Examples

```
## 1+1=2
a <- as.bigz(1)
a + a

## Two non-small Mersenne primes:
two <- as.bigz(2)
p1 <- two^107 -1 ; isprime(p1); p1
p2 <- two^127 -1 ; isprime(p2); p2

stopifnot( is.na(NA_bigz_) )

## Calculate c = x^e mod n
## ------------------------------------------------------------------------
x <- as.bigz("0x123456789abcdef") # my secret message
e <- as.bigz(3) # something smelling like a dangerous public RSA exponent
(n <- p1 * p2) #  a product of two primes
as.character(n, b=16)# as both primes were Mersenne's..

## recreate the three numbers above [for demo below]:
n. <- n; x. <- x; e. <- e # save
Rev <- function() { n <<- n.; x <<- x.; e <<- e.}

# first way to do it right
modulus(x) <- n
c <- x ^ e ; c ; Rev()

# similar second way (makes more sense if you reuse e) to do it right
modulus(e) <- n
c2 <- x ^ e
stopifnot(identical(c2, c), is.bigz(c2)) ; Rev()
```

```
# third way to do it right
c3 <- x ^ as.bigz(e, n) ; stopifnot(identical(c3, c))

# fourth way to do it right
c4 <- as.bigz(x, n) ^ e ; stopifnot(identical(c4, c))

# WRONG! (although very beautiful. Ok only for very small 'e' as here)
cc <- x ^ e %% n
cc == c

# Return result in hexa
as.character(c, b=16)

# Depict the "S4-class" bigz, i.e., the formal (S4) methods:
if(require("Rmpfr")) # mostly interesting there
  showMethods(class="bigz")

# an  sapply() version that works for big integers "bigz":
sapplyZ <- function(X, FUN, ...) c_bigz(lapply(X, FUN, ...))

# dummy example showing it works (here):
zz <- as.bigz(3)^(1000+ 1:999)
z1 <- sapplyZ(zz, function(z) z^2)
stopifnot( identical(z1, zz^2) )
```

---

bigz operators   *Basic Arithmetic Operators for Large Integers ("bigz")*

Description

Addition, substraction, multiplication, (integer) division, remainder of division, multiplicative inverse, power and logarithm functions.

Usage

```
add.bigz(e1, e2)
sub.bigz(e1, e2 = NULL)
mul.bigz(e1, e2)
div.bigz(e1, e2)
divq.bigz(e1,e2) ## ==  e1 %/% e2
mod.bigz(e1, e2) ## ==  e1 %%  e2
## S3 method for class 'bigz'
abs(x)

inv.bigz(a, b,...)## == (1 / a) (modulo b)

pow.bigz(e1, e2,...)## == e1 ^ e2

## S3 method for class 'bigz'
```

```
log(x, base=exp(1))
## S3 method for class 'bigz'
log2(x)
## S3 method for class 'bigz'
log10(x)
```

Arguments

`x`	bigz, integer or string from an integer
`e1, e2, a,b`	bigz, integer or string from an integer
`base`	base of the logarithm; base e as default
`...`	Additional parameters

Details

Operators can be used directly when objects are of class bigz: a + b, log(a), etc.

For details about the internal modulus state, *and* the rules applied for arithmetic operations on big integers with a modulus, see the `bigz` help page.

`a / b` = `div(a,b)` returns a rational number unless the operands have a (matching) modulus where `a * b^-1` results.
`a %/% b` (or, equivalently, `divq(a,b)`) returns the quotient of simple *integer* division (with truncation towards zero), possibly re-adding a modulus at the end (but *not* using a modulus like in `a / b`).

`r <-inv.bigz(a,m)`, the multiplicative inverse of `a` modulo m, corresponds to `1/a` or `a ^-1` from above *when* `a` has modulus `m`. Note that a not always has an inverse modulo m, in which case `r` will be `NA` with a warning that can be turned off via

```
options("gmp:warnNoInv" = FALSE)
```

.

Value

Apart from `/` (or `div`), where rational numbers (`bigq`) may result, these functions return an object of class `"bigz"`, representing the result of the arithmetic operation.

Author(s)

Immanuel Scholz and Antoine Lucas

References

The GNU MP Library, see https://gmplib.org

Examples

```
# 1+1=2
as.bigz(1) + 1
as.bigz(2)^10
as.bigz(2)^200

# if my.large.num.string is set to a number, this returns the least byte
(my.large.num.string <- paste(sample(0:9, 200, replace=TRUE), collapse=""))
mod.bigz(as.bigz(my.large.num.string), "0xff")

# power exponents can be up to MAX_INT in size, or unlimited if a
# bigz's modulus is set.
pow.bigz(10,10000)

## Modulo 11,   7 and 8 are inverses :
as.bigz(7, mod = 11) * 8 ## ==>  1  (mod 11)
inv.bigz(7, 11)## hence, 8
a <- 1:10
(i.a <- inv.bigz(a, 11))
d <- as.bigz(7)
a %/% d  # = divq(a, d)
a %%  d  # = mod.bigz (a, d)

(ii <- inv.bigz(1:10, 16))
## with 5 warnings (one for each NA)
op <- options("gmp:warnNoInv" = FALSE)
i2 <- inv.bigz(1:10, 16) # no warnings
(i3 <- 1 / as.bigz(1:10, 16))
i4 <- as.bigz(1:10, 16) ^ -1
stopifnot(identical(ii, i2),
  identical(as.bigz(i2, 16), i3),
  identical(i3, i4))
options(op)# revert previous options' settings

stopifnot(inv.bigz(7, 11) == 8,
          all(as.bigz(i.a, 11) * a == 1),
          identical(a %/% d, divq.bigz(1:10, 7)),
          identical(a %%  d, mod.bigz (a, d))
 )
```

---

cumsum *(Cumulative) Sums, Products of Large Integers and Rationals*

Description

Theses are methods to 'overload' the sum(), cumsum() and prod() functions for big rationals and big integers.

Usage

```
## S3 method for class 'bigz'
cumsum(x)
## S3 method for class 'bigq'
cumsum(x)
## S3 method for class 'bigz'
sum(..., na.rm = FALSE)
## S3 method for class 'bigq'
sum(..., na.rm = FALSE)
## S3 method for class 'bigz'
prod(..., na.rm = FALSE)
## S3 method for class 'bigq'
prod(..., na.rm = FALSE)
```

Arguments

`x, ...`	R objects of class `bigz` or `bigq` or 'simple' numbers.
`na.rm`	logical indicating if missing values (`NA`) should be removed before the computation.

Value

return an element of class bigz or bigq.

Author(s)

Antoine Lucas

See Also

`apply`

Examples

```
x <- as.bigz(1:12)
cumsum(x)
prod(x)
sum(x)

x <- as.bigq(1:12)
cumsum(x)
prod(x)
sum(x)
```

`extract` *Extract or Replace Parts of an Object*

Description

Operators acting on vectors, arrays and lists to extract or replace subsets.

Usage

```
## S3 method for class 'bigz'
x[i=NULL, j=NULL, drop = TRUE]
## S3 method for class 'bigq'
x[i=NULL, j=NULL, drop = TRUE]

##_______ In the following, only the bigq method is mentioned: ______



## S3 method for class 'bigq'
c(..., recursive = FALSE)
## S3 method for class 'bigq'
rep(x, times=1, length.out=NA, each=1, ...)
```

Arguments

`x`	R object of class `"bigz"` or `"bigq"`, respectively.
`...`	further arguments, notably for `c()`.
`i,j`	indices, see standard R subsetting and subassignment.
`drop`	logical, unused here.
`times, length.out, each`	integer; typically only *one* is specified; for more see `rep` (standard R, package **base**).
`recursive`	unused here

Note

Unlike standard matrices, `x[i]` and `x[i,]` do the same.

Examples

```
a <- as.bigz(123)
## indexing "outside" --> extends the vectors (filling with NA)
a[2] <- a[1]
a[4] <- -4

## create a vector of 3 a
```

```
c(a,a,a)

## repeate a 5 times
rep(a,5)

## with matrix
m <- matrix.bigz(1:6,3)

## these do the same:
m[1,]
m[1]
m[-c(2,3),]
m[-c(2,3)]
m[c(TRUE,FALSE,FALSE)]

##_modification on matrix
m[2,-1] <- 11
```

`Extremes` *Extrema (Maxima and Minima)*

Description

We provide S3 `methods` for `min` and `max` for big rationals (bigq) and big integers (biqz); consequently, `range()` works as well.

Similarly, S4 methods are provided for `which.min()` and `which.max()`.

Usage

```
## S3 method for class 'bigz'
max(..., na.rm=FALSE)
## S3 method for class 'bigq'
max(..., na.rm=FALSE)
## S3 method for class 'bigz'
min(..., na.rm=FALSE)
## S3 method for class 'bigq'
min(..., na.rm=FALSE)

## S4 method for signature 'bigz'
which.min(x)


## S4 method for signature 'bigq'
which.max(x)
```

Arguments

`x`	a "big integer" (`bigz`) or "big rational" (`bigq`) vector.
`...`	numeric arguments
`na.rm`	a logical indicating whether missing values should be removed.

Value

an object of class "`bigz`" or "`bigq`".

Author(s)

Antoine Lucas

See Also

`max` etc in **base**.

Examples

```
 x <- as.bigz(1:10)
 max(x)
 min(x)
 range(x) # works correctly via default method
 x <- x[c(7:10,6:3,1:2)]
 which.min(x) ## 9
 which.max(x) ## 4

 Q <- as.bigq(1:10, 3)
 max(Q)
 min(Q)
 (Q <- Q[c(6:3, 7:10,1:2)])
 stopifnot(which.min(Q) == which.min(asNumeric(Q)),
           which.max(Q) == which.max(asNumeric(Q)))

stopifnot(range(x) == c(1,10), 3*range(Q) == c(1,10))
```

`factorialZ` *Factorial and Binomial Coefficient as Big Integer*

Description

Efficiently compute the factorial $n!$ or a binomial coefficient $\binom{n}{k}$ as big integer (class `bigz`).

Usage

```
factorialZ(n)
chooseZ(n, k)
```

Arguments

n	non-negative integer (vector), for `factorialZ`. For `chooseZ`, may be a `bigz` big integer, also negative.
k	non-negative integer vector.

Value

a vector of big integers, i.e., of class `bigz`.

See Also

`factorial` and `gamma` in base R;

Examples

```
factorialZ(0:10)# 1 1 2 6 ... 3628800
factorialZ(0:40)# larger
factorialZ(200)

n <- 1000
f1000 <- factorialZ(n)
stopifnot(1e-15 > abs(as.numeric(1 - lfactorial(n)/log(f1000))))

system.time(replicate(8, f1e4 <<- factorialZ(10000)))
nchar(as.character(f1e4))# 35660 ... (too many to even look at ..)

chooseZ(1000, 100:102)# vectorizes
chooseZ(as.bigz(2)^120, 10)
n <- c(50,80,100)
k <- c(20,30,40)
## currently with an undesirable warning: % from methods/src/eval.c  _FIXME_
stopifnot(chooseZ(n,k) == factorialZ(n) / (factorialZ(k)*factorialZ(n-k)))
```

`factorization` *Factorize a number*

Description

Give all primes numbers to factor the number

Usage

```
factorize(n)
```

Arguments

n	Either integer, numeric or string value (String value: ither starting with `0x` for hexadecimal, `0b` for binary or without prefix for decimal values.) Or an element of class bigz.

Details

The factorization function uses the Pollard Rho algorithm.

Value

Vector of class bigz.

Author(s)

Antoine Lucas

References

The GNU MP Library, see https://gmplib.org

Examples

```
factorize(34455342)
```

---

formatN *Format Numbers Keeping Classes Distinguishable*

Description

Format (generalized) numbers in a way that their `class`es are distinguishable. Contrary to `format()` which uses a common format for all elements of `x`, here, each entry is formatted individually.

Usage

```
formatN(x, ...)
## Default S3 method:
formatN(x, ...)
## S3 method for class 'integer'
formatN(x, ...)
## S3 method for class 'double'
formatN(x, ...)
## S3 method for class 'bigz'
formatN(x, ...)
## S3 method for class 'bigq'
formatN(x, ...)
```

Arguments

`x`	any R object, typically "number-like".
`...`	potentially further arguments passed to methods.

Value

a character vector of the same `length` as `x`, each entry a representation of the corresponding entry in `x`.

Author(s)

Martin Maechler

See Also

`format`, including its (sophisticated) default method; `as.character`.

Examples

```
## Note that each class is uniquely recognizable from its output:
formatN(    -2:5)# integer
formatN(0 + -2:5)# double precision
formatN(as.bigz(-2:5))
formatN(as.bigq(-2:5, 4))
```

`frexpZ` *Split Number into Fractional and Exponent of 2 Parts*

Description

Breaks the number `x` into its binary significand ("fraction") $d \in [0.5, 1)$ and ex, the integral exponent for 2, such that $x = d \cdot 2^{ex}$.

If `x` is zero, both parts (significand and exponent) are zero.

Usage

```
frexpZ(x)
```

Arguments

`x`	integer or big integer (`bigz`).

Value

a `list` with the two components

`d`	a numeric vector whose absolute values are either zero, or in $[\frac{1}{2}, 1)$.
`exp`	an integer vector of the same length; note that `exp == 1 + floor(log2(x))`, and hence always `exp > log2(x)`.

Author(s)

Martin Maechler

See Also

log2, etc; for bigz objects built on (the C++ equivalent of) `frexp()`, actually GMP's '`mpz_get_d_2exp()`'.

Examples

```
frexpZ(1:10)
## and confirm :
with(frexpZ(1:10),  d * 2^exp)
x <- rpois(1000, lambda=100) * (1 + rpois(1000, lambda=16))
X <- as.bigz(x)
stopifnot(all.equal(x, with(frexpZ(x), d* 2^exp)),
          1+floor(log2(x)) == (fx <- frexpZ(x)$exp),
          fx == frexpZ(X)$exp,
          1+floor(log2(X)) == fx
)
```

`gcd.bigz` *Greatest Common Divisor (GCD) and Least Common Multiple (LCM)*

Description

Compute the greatest common divisor (GCD) and least common multiple (LCM) of two (big) integers.

Usage

```
## S3 method for class 'bigz'
gcd(a, b)
lcm.bigz(a, b)
```

Arguments

`a,b`	Either integer, numeric, bigz or a string value; if a string, either starting with `0x` for hexadecimal, `0b` for binary or without prefix for decimal values.

Value

An element of class bigz

Author(s)

Antoine Lucas

References

The GNU MP Library, see https://gmplib.org

See Also

gcdex

Examples

```
gcd.bigz(210,342) # or also
lcm.bigz(210,342)
a <- 210 ; b <- 342
stopifnot(gcd.bigz(a,b) * lcm.bigz(a,b) == a * b)

## or
(a <- as.bigz("82696155787249022588"))
(b <- as.bigz("65175989479756205392"))
gcd(a,b) # 4
stopifnot(gcd(a,b) * lcm.bigz(a,b) == a * b)
```

---

gcdex *Compute Bezoult Coefficient*

Description

Compute g,s,t as $as + bt = g = gcd(a, b)$. s and t are also known as Bezoult coefficients.

Usage

```
gcdex(a, b)
```

Arguments

a,b	either integer, numeric, character string, or of class "bigz"; If a string, either starting with "0x" for hexadecimal, "0b" for binary or without prefix for decimal values.

Value

a class "bigz" vector of length 3 with (long integer) values g, s, t.

Author(s)

Antoine Lucas

References

The GNU MP Library, see https://gmplib.org

See Also

gcd.bigz

Examples

```
gcdex(342,654)
```

---

gmp-ifiworkarounds *Base Functions in 'gmp'-ified Versions*

Description

Functions from **base** etc which need a *copy* in the **gmp** namespace so they correctly dispatch.

Usage

```
outer(X, Y, FUN = "*", ...)
```

Arguments

`X, Y, FUN, ...`	See **base** package help: `outer`.

See Also

`outer`.

Examples

```
twop <- as.bigz(2)^(99:103)
(mtw <- outer(twop, 0:2))
stopifnot(
   identical(dim(mtw), as.integer(c(5,3)))
 ,
   mtw[,1] == 0
 ,
   identical(as.vector(mtw[,2]), twop)
)
```

---

gmp.utils *GMP Number Utilities*

Description

`gmpVersion()` returns the version of the GMP library which **gmp** is currently linked to.

Usage

```
gmpVersion()
```

References

The GNU MP Library, see https://gmplib.org

Examples

```
gmpVersion()
```

---

is.whole *Whole ("Integer") Numbers*

Description

Check which elements of `x[]` are integer valued aka "whole" numbers.

Usage

```
is.whole(x)
## Default S3 method:
is.whole(x)
## S3 method for class 'bigz'
is.whole(x)
## S3 method for class 'bigq'
is.whole(x)
```

Arguments

x	any R vector

Value

logical vector of the same length as `x`, indicating where `x[.]` is integer valued.

Author(s)

Martin Maechler

See Also

`is.integer(x)` (**base** package) checks for the *internal* mode or class; not if `x[i]` are integer valued.

The `is.whole()` method for "mpfr" numbers.

Examples

```
is.integer(3) # FALSE, it's internally a double
is.whole(3)   # TRUE
## integer valued complex numbers  (two FALSE) :
is.whole(c(7, 1 + 1i, 1.2, 3.4i, 7i))
is.whole(factorialZ(20)^(10:12)) ## "bigz" are *always* whole numbers
q <- c(as.bigz(36)^50 / as.bigz(30)^40, 3, factorialZ(30:31), 12.25)
is.whole(q) # F T T T F
```

`isprime` *Determine if number is (very probably) prime*

Description

Determine whether the number n is prime or not, with *three* possible answers:

2: n is prime,

1: n is probably prime (without beeing certain),

0: n is composite.

Usage

```
isprime(n, reps = 40)
```

Arguments

`n`	integer number, to be tested.
`reps`	integer number of primality testing repeats.

Details

This function does some trial divisions, then some Miller-Rabin probabilistic primary tests. `reps` controls how many such tests are done, 5 to 10 is already a resonable number. More will reduce the chances of a composite being returned as "probably prime".

Value

`0`	n is not prime
`1`	n is probably prime
`2`	n is prime

Author(s)

Antoine Lucas

References

The GNU MP Library, see https://gmplib.org

See Also

nextprime, factorize.

Note that for "small" n, which means something like $n < 10'000'000$, non-probabilistic methods (such as factorize()) are fast enough. For example, primes in package **sfsmisc**.

Examples

```
isprime(210)
isprime(71)

# All primes numbers from 1 to 100
t <- isprime(1:99)
(1:99)[t > 0]

table(isprime(1:10000))# 0 and 2 : surely prime or not prime

primes <- function(n) {
  ## all primes <= n
  stopifnot(length(n) == 1, n <= 1e7) # be reasonable
  p <- c(2L, as.integer(seq(3, n, by=2)))
  p[isprime(p) > 0]
}

## quite quickly, but for these small numbers
## still slower than e.g., sfsmisc::primes()
system.time(p100k <- primes(100000))

## The first couple of Mersenne primes:
p.exp <- primes(1000)
Mers <- as.bigz(2) ^ p.exp - 1
isp.M <- sapply(seq_along(Mers), function(i) isprime(Mers[i], reps=256))
cbind(p.exp, isp.M)[isp.M > 0,]
Mers[isp.M > 0]
```

---

lucnum — *Compute Fibonacci and Lucas numbers*

Description

fibnum compute n-th Fibonacci number. fibnum2 compute (n-1)-th and n-th Fibonacci number. lucnum compute n-th lucas number. lucnum2 compute (n-1)-th and n-th lucas number.

Fibonacci numbers are define by: $F_n = F_{n-1} + F_{n-2}$ Lucas numbers are define by: $L_n = F_n + 2F_{n-1}$

Usage

```
fibnum(n)
fibnum2(n)
lucnum(n)
lucnum2(n)
```

Arguments

`n`	Integer

Value

Fibonacci numbers and Lucas number.

Author(s)

Antoine Lucas

References

The GNU MP Library, see https://gmplib.org

Examples

```
fibnum(10)
fibnum2(10)
lucnum(10)
lucnum2(10)
```

`matrix` *Matrix manipulation with gmp*

Description

Overload of "all" standard tools useful for matrix manipulation adapted to large numbers.

Usage

```
## S3 method for class 'bigz'
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL, mod = NA,...)

is.matrixZQ(x)

## S3 method for class 'bigz'
x %*% y
## S3 method for class 'bigq'
x %*% y
## S3 method for class 'bigq'
crossprod(x, y=NULL,...)
## S3 method for class 'bigz'
tcrossprod(x, y=NULL,...)
## ..... etc
```

Arguments

`data`	an optional data vector
`nrow`	the desired number of rows
`ncol`	the desired number of columns
`byrow`	logical. If `FALSE` (the default), the matrix is filled by columns, otherwise the matrix is filled by rows.
`dimnames`	not implemented for `"bigz"` or `"bigq"` matrices.
`mod`	optional modulus (when `data` is `"bigz"`).
`...`	Not used
`x,y`	numeric, `bigz`, or `bigq` matrices or vectors.

Details

The extract function (`"["`) is the same use for vector or matrix. Hence, `x[i]` returns the same values as `x[i,]`. This is not considered a feature and may be changed in the future (with warnings).

All matrix multiplications should work as with numeric matrices.

Special features concerning the `"bigz"` class: the modulus can be

Unset: Just play with large numbers

Set with a vector of size 1: Example: `matrix.bigz(1:6,nrow=2,ncol=3,mod=7)` This means you work in Z/nZ, for the whole matrix. It is the only case where the `%*%` and `solve` functions will work in Z/nZ.

Set with a vector smaller than data: Example: `matrix.bigz(1:6,nrow=2,ncol=3,mod=1:5)`. Then, the modulus is repeated to the end of data. This can be used to define a matrix with a different modulus at each row.

Set with same size as data: Modulus is defined for each cell

Value

`matrix()`: A matrix of class `"bigz"` or `"bigq"`.

`is.matrixZQ()`: `TRUE` or `FALSE`.

`dim()`, `ncol()`, etc: integer or `NULL`, as for simple matrices.

Author(s)

Antoine Lucas

See Also

Solving a linear system: `solve.bigz`. `matrix`

Examples

```
V <- as.bigz(v <- 3:7)
crossprod(V)# scalar product
(C <- t(V))
stopifnot(dim(C) == dim(t(v)), C == v,
          dim(t(C)) == c(length(v), 1),
          crossprod(V) == sum(V * V),
         tcrossprod(V) == outer(v,v),
          identical(C, t(t(C))),
          is.matrixZQ(C), !is.matrixZQ(V), !is.matrixZQ(5)
)

## a matrix
x <- diag(1:4)
## invert this matrix
(xI <- solve(x))

## matrix in Z/7Z
y <- as.bigz(x,7)
## invert this matrix (result is *different* from solve(x)):
(yI <- solve(y))
stopifnot(yI %*% y == diag(4),
          y %*% yI == diag(4))

## matrix in Q
z  <- as.bigq(x)
## invert this matrix (result is the same as solve(x))
(zI <- solve(z))

stopifnot(abs(zI - xI) <= 1e-13,
          z %*% zI == diag(4),
          identical(crossprod(zI), zI %*% t(zI))
         )

A <- matrix(2^as.bigz(1:12), 3,4)
for(a in list(A, as.bigq(A, 16), factorialZ(20), as.bigq(2:9, 3:4))) {
  a.a <- crossprod(a)
  aa. <- tcrossprod(a)
  stopifnot(identical(a.a, crossprod(a,a)),
      identical(a.a, t(a) %*% a)
            ,
            identical(aa., tcrossprod(a,a)),
    identical(aa., a %*% t(a))
     )
}# {for}
```

`modulus` *Modulus of a Big Integer*

Description

The modulus of a bigz number a is "unset" when a is a regular integer, $a \in Z$). Or the modulus can be set to m which means $a \in Z/m \cdot Z$), i.e., all arithmetic with a is performed 'modulo m'.

Usage

```
modulus(a)
modulus(a) <- value
```

Arguments

a	R object of class "bigz"
value	integer number or object of class "bigz".

Examples

```
x <- as.bigz(24)
modulus(x) # NULL, i.e. none

# x element of Z/31Z :
modulus(x) <- 31
x+x  #  48 |-> (17 %% 31)
10*x # 240 |-> (23 %% 31)
x31 <- x

# reset modulus to "none":
modulus(x) <- NA; x; x. <- x
x <- x31
modulus(x) <- NULL; x

stopifnot(identical(x,            as.bigz(24)), identical(x, x.),
          identical(modulus(x31), as.bigz(31)))
```

---

nextprime *Next Prime Number*

Description

Return the next prime number, say p, with $p > n$.

Usage

```
nextprime(n)
```

Arguments

n	Integer

Details

This function uses probabilistic algorithm to identify primes. For practical purposes, it is adequate, the chance of a composite passing will be extremely small.

Value

A (probably) prime number

Author(s)

Antoine Lucas

References

The GNU MP Library, see https://gmplib.org

See Also

isprime and its references and examples.

Examples

```
nextprime(14)
## still very fast:
(p <- nextprime(1e7))
## to be really sure { isprime()  gives "probably prime" } :
stopifnot(identical(p, factorize(p)))
```

Oakley — *RFC 2409 Oakley Groups - Parameters for Diffie-Hellman Key Exchange*

Description

RFC 2409 standardizes global unique prime numbers and generators for the purpose of secure asymmetric key exchange on the Internet.

Usage

```
data(Oakley1)
data(Oakley2)
```

Value

Oakley1 returns an object of class bigz for a 768 bit Diffie-Hellman group. The generator is stored as value with the respective prime number as modulus attribute.

Oakley2 returns an object of class bigz for a 1024 bit Diffie-Hellman group. The generator is stored as value with the respective prime number as modulus attribute.

References

The Internet Key Exchange (RFC 2409), Nov. 1998

Examples

```
packageDescription("gmp") # {possibly useful for debugging}

data(Oakley1)
(M1 <- modulus(Oakley1))
isprime(M1)# '1' : "probably prime"
sizeinbase(M1)#  232 digits (was 309 in older version)
```

---

powm *Exponentiation function*

Description

This function return $x^y mod n$.

This function return $x^y mod n$ pow.bigz do the same when modulus is set.

Usage

```
powm(x, y, n)
```

Arguments

x	Integer or big integer - possibly a vector
y	Integer or big integer - possibly a vector
n	Integer or big integer - possibly a vector

Value

A bigz class representing the parameter value.

Author(s)

A. L.

See Also

pow.bigz

Examples

```
powm(4,7,9)


x = as.bigz(4,9)
x ^ 7
```

`Random` *Generate a random number*

Description

Generate a uniformly distributed random number in the range 0 to $2^{size} - 1$, inclusive.

Usage

```
urand.bigz(nb=1,size=200, seed = 0)
```

Arguments

`nb`	Integer: number of random numbers to be generated (size of vector returned)
`size`	Integer: number will be generated in the range 0 to $2^{size} - 1$
`seed`	Bigz: random seed initialisation

Value

A biginteger of class bigz.

Author(s)

Antoine Lucas

References

'`mpz_urandomb`' from the GMP Library, see https://gmplib.org

Examples

```
# Integers are differents
urand.bigz()
urand.bigz()
urand.bigz()

# Integers are the same
urand.bigz(seed="234234234324323")
urand.bigz(seed="234234234324323")

# Vector
urand.bigz(nb=50,size=30)
```

`Relational Operator` *Relational Operators*

Description

Binary operators which allow the comparison of values in atomic vectors.

Usage

```
## S3 method for class 'bigz'
sign(x)
## S3 method for class 'bigz'
e1 == e2
## S3 method for class 'bigz'
 e1 < e2
## S3 method for class 'bigz'
e1 >= e2
```

Arguments

`x, e1, e2`	R object (vector or matrix-like) of class "`bigz`".

See Also

`mod.bigz` for arithmetic operators.

Examples

```
x <- as.bigz(8000)
x ^ 300 < 2 ^x

sign(as.bigz(-3:3))
sign(as.bigq(-2:2, 7))
```

`roundQ` *Rounding Big Rationals ("bigq") to Decimals*

Description

Rounding big rationals (of class "bigq", see `as.bigq()`) to decimal `digits` is strictly based on a (optionally choosable) definition of rounding to integer, i.e., `digits = 0`, the default method of which we provide as `round0()`.

The users typically just call `round(x,digits)` as elsewhere, and the `round()` method will call `round(x,digits,round0=round0)`.

Usage

```
round0(x)

roundQ(x, digits = 0, r0 = round0)

## S3 method for class 'bigq'
round(x, digits = 0)
```

Arguments

`x`	vector of big rationals, i.e., of `class` "bigq".
`digits`	integer number of decimal digits to round to.
`r0`	a `function` of one argument which implements a version of `round(x,digits=0)`. The default for `roundQ()` is to use our `round0()` which implements "round to even", as base R's `round`.

Value

`round0()` returns a vector of big integers, i.e., "bigz" classed.

`roundQ(x,digits,round0)` returns a vector of big rationals, "bigq", as `x`.

`round.bigq` is very simply defined as `function(x,digits) roundQ(x,digits)` .

Author(s)

Martin Maechler, ETH Zurich

References

The vignette "*Exact Decimal Rounding via Rationals*" from CRAN package **round**,

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

See Also

`round` for (double precision) numbers in base R; `roundX` from CRAN package **round**.

Examples

```
qq <- as.bigq((-21:31), 10)
noquote(cbind(as.character(qq), asNumeric(qq)))
round0(qq) # Big Integer ("bigz")
## corresponds to R's own "round to even" :
stopifnot(round0(qq) == round(asNumeric(qq)))
round(qq) # == round(qq, 0): the same as round0(qq) *but* Big Rational ("bigq")

halfs <- as.bigq(1,2) + -5:12
```

```
## round0() is simply
round0 <- function (x) {
    nU <- as.bigz.bigq(xU <- x + as.bigq(1, 2)) # traditional round: .5 rounded up
    if(any(I <- is.whole.bigq(xU))) { # I <==>  x == <n>.5 : "hard case"
        I[I] <- .mod.bigz(nU[I], 2L) == 1L # rounded up is odd  ==> round *down*
        nU[I] <- nU[I] - 1L
    }
    nU
}

## 's' for simple: rounding as you learned in school:
round0s <- function(x) as.bigz.bigq(x + as.bigq(1, 2))

cbind(halfs, round0s(halfs), round0(halfs))


## roundQ() is simply
roundQ <- function(x, digits = 0, r0 = round0) {
    ## round(x * 10^d) / 10^d --  vectorizing in both (x, digits)
    p10 <- as.bigz(10) ^ digits # class: if(all(digits >= 0)) "bigz" else "bigq"
    r0(x * p10) / p10
}
```

`sizeinbase` *Compute size of a bigz in a base*

Description

Return an approximation to the number of character the integer X would have printed in base b. The approximation is never too small.

In case of powers of 2, function gives exact result.

Usage

```
sizeinbase(a, b=10)
```

Arguments

`a`	big integer, i.e. "`bigz`"
`b`	base

Value

integer of the same length as `a`: the size, i.e. number of digits, of each `a[i]`.

Author(s)

Antoine Lucas

References

The GNU MP Library, see https://gmplib.org

Examples

```
sizeinbase(342434, 10)# 6 obviously

Iv <- as.bigz(2:7)^500
sizeinbase(Iv)
stopifnot(sizeinbase(Iv)       == nchar(as.character(Iv)),
          sizeinbase(Iv, b=16) == nchar(as.character(Iv, b=16)))
```

---

solve.bigz *Solve a system of equation*

Description

This generic function solves the equation $a\% * \%x = b$ for x, where b can be either a vector or a matrix.

If a and b are rational, return is a rational matrix.

If a and b are big integers (of class bigz) solution is in Z/nZ if there is a common modulus, or a rational matrix if not.

Usage

```
## S3 method for class 'bigz'
solve(a, b, ...)
## S3 method for class 'bigq'
solve(a, b, ...)
```

Arguments

`a,b`	A element of class bigz or bigq
`...`	Unused

Details

It uses the Gauss and trucmuch algo … (to be detailled).

Value

If a and b are rational, return is a rational matrix.

If a and b are big integers (of class bigz) solution is in Z/nZ if there is a common modulus, of a rational matrix if not.

Author(s)

Antoine Lucas

See Also

solve

Examples

```
x <- matrix(1:4,2,2)  ## standard solve :
solve(x)

q <- as.bigq(x) ## solve with rational
solve(q)

z <- as.bigz(x)
modulus(z) <- 7  ## solve in Z/7Z :
solve(z)

b <- c(1,3)
solve(q,b)
solve(z,b)

## Inversion of ("non-trivial") rational matrices :

A <- rbind(c(10, 1,  3),
           c( 4, 2, 10),
           c( 1, 8,  2))
(IA.q <- solve(as.bigq(A))) # fractions..
stopifnot(diag(3) == A %*% IA.q)# perfect

set.seed(5); B <- matrix(round(9*runif(5^2, -1,1)), 5)
B
(IB.q <- solve(as.bigq(B)))
stopifnot(diag(5) == B %*% IB.q, diag(5) == IB.q %*% B,
          identical(B, asNumeric(solve(IB.q))))
```

Stirling — *Eulerian and Stirling Numbers of First and Second Kind*

Description

Compute Eulerian numbers and Stirling numbers of the first and second kind, possibly vectorized for all k “at once”.

Usage

```
Stirling1(n, k)
Stirling2(n, k, method = c("lookup.or.store", "direct"))
Eulerian (n, k, method = c("lookup.or.store", "direct"))

Stirling1.all(n)
Stirling2.all(n)
Eulerian.all (n)
```

Arguments

`n`	positive integer (`0` is allowed for `Eulerian()`).
`k`	integer in `0:n`.
`method`	for `Eulerian()` and `Stirling2()`, string specifying the method to be used. `"direct"` uses the explicit formula (which may suffer from some cancelation for "large" `n`).

Details

Eulerian numbers:
$A(n, k) =$ the number of permutations of 1,2,...,n with exactly k ascents (or exactly k descents).

Stirling numbers of the first kind:
s(n,k) = (-1)^n-k times the number of permutations of 1,2,...,n with exactly k cycles.

Stirling numbers of the second kind:
$S_n^{(k)}$ is the number of ways of partitioning a set of n elements into k non-empty subsets.

Value

$A(n, k)$, $s(n, k)$ or $S(n, k) = S_n^{(k)}$, respectively.

`Eulerian.all(n)` is the same as `sapply(0:(n-1),Eulerian,n=n)` (for $n > 0$),
`Stirling1.all(n)` is the same as `sapply(1:n,Stirling1,n=n)`, and
`Stirling2.all(n)` is the same as `sapply(1:n,Stirling2,n=n)`, but more efficient.

Note

For typical double precision arithmetic,
`Eulerian*(n,*)` overflow (to `Inf`) for $n \geq 172$,
`Stirling1*(n,*)` overflow (to $\pm$`Inf`) for $n \geq 171$, and
`Stirling2*(n,*)` overflow (to `Inf`) for $n \geq 220$.

Author(s)

Martin Maechler ("direct": May 1992)

References

Eulerians:

NIST Digital Library of Mathematical Functions, 26.14: https://dlmf.nist.gov/26.14

Stirling numbers:

Abramowitz and Stegun 24,1,4 (p. 824-5 ; Table 24.4, p.835); Closed Form : p.824 "C."

NIST Digital Library of Mathematical Functions, 26.8: https://dlmf.nist.gov/26.8

See Also

`chooseZ` for the binomial coefficients.

Examples

```
Stirling1(7,2)
Stirling2(7,3)

stopifnot(
 Stirling1.all(9) == c(40320, -109584, 118124, -67284, 22449, -4536, 546, -36, 1)
 ,
 Stirling2.all(9) == c(1, 255, 3025, 7770, 6951, 2646, 462, 36, 1)
 ,
 Eulerian.all(7) == c(1, 120, 1191, 2416, 1191, 120, 1)
)
```

Index

!=.bigq (Bigq), 5
!=.bigz (Relational Operator), 35
∗ **GCD**
    gcd.bigz, 22
∗ **LCM**
    gcd.bigz, 22
∗ **Rounding**
    roundQ, 35
∗ **arithmetic**
    Stirling, 39
∗ **arith**
    apply, 2
    asNumeric, 3
    Bigq, 5
    bigq, 6
    Bigq operators, 8
    bigz, 9
    bigz operators, 12
    cumsum, 14
    extract, 16
    Extremes, 17
    factorialZ, 18
    factorization, 19
    frexpZ, 21
    gcd.bigz, 22
    gcdex, 23
    gmp.utils, 24
    isprime, 26
    lucnum, 27
    matrix, 28
    modulus, 30
    nextprime, 31
    powm, 33
    Random, 34
    Relational Operator, 35
    roundQ, 35
    sizeinbase, 37
    solve.bigz, 38
∗ **character**
    formatN, 20
∗ **data**
    Oakley, 32
∗ **math**
    is.whole, 25
∗ **methods**
    asNumeric, 3
∗ **misc**
    gmp-ifiworkarounds, 24
∗ **print**
    formatN, 20
*.bigq (Bigq operators), 8
*.bigz (bigz operators), 12
+.bigq (Bigq operators), 8
+.bigz (bigz operators), 12
-.bigq (Bigq operators), 8
-.bigz (bigz operators), 12
/.bigq (Bigq operators), 8
/.bigz (bigz operators), 12
<.bigq (Bigq), 5
<.bigz (Relational Operator), 35
<=.bigq (Bigq), 5
<=.bigz (Relational Operator), 35
==.bigq (Bigq), 5
==.bigz (Relational Operator), 35
>.bigq (Bigq), 5
>.bigz (Relational Operator), 35
>=.bigq (Bigq), 5
>=.bigz (Relational Operator), 35
[.bigq (extract), 16
[.bigz (extract), 16
[<-.bigq (extract), 16
[<-.bigz (extract), 16
[[.bigq (extract), 16
[[.bigz (extract), 16
[[<-.bigq (extract), 16
[[<-.bigz (extract), 16
%*% (matrix), 28
%/%.bigz (bigz operators), 12

%%.bigz (bigz operators), 12
^.bigq (Bigq operators), 8
^.bigz (bigz operators), 12

abs.bigq (Bigq operators), 8
abs.bigz (bigz operators), 12
add.bigq (Bigq operators), 8
add.bigz, *10*
add.bigz (bigz operators), 12
apply, *2*, 2, *3*, *15*
array, *4*
as.bigq, *35*
as.bigq (bigq), 6
as.bigz (bigz), 9
as.bigz.bigq (bigq), 6
as.character, *21*
as.character.bigq (bigq), 6
as.character.bigz (bigz), 9
as.double.bigq (bigq), 6
as.double.bigz (bigz), 9
as.matrix.bigq (matrix), 28
as.matrix.bigz (matrix), 28
as.numeric, *4*
as.vector.bigq (matrix), 28
as.vector.bigz (matrix), 28
asNumeric, 3
asNumeric,ANY-method (asNumeric), 3
asNumeric,bigq-method (asNumeric), 3
asNumeric,bigz-method (asNumeric), 3
asNumeric-methods (asNumeric), 3

Bernoulli, *5*
BernoulliQ, 4
biginteger_as (bigz), 9
biginteger_as_character (bigz), 9
Bigq, 5
bigq, *5*, 6, *8*, *13*, *18*
Bigq operators, 8
bigq-class (bigq), 6
bigz, *3*, *8*, 9, *13*, *18*, *19*, *21*, *22*, *29*, *31*, *32*, *35*, *37*
bigz operators, 12
bigz-class (bigz), 9

c.bigq (extract), 16
c.bigz (extract), 16
c_bigq (bigq), 6
c_bigz (bigz), 9
cbind.bigq (matrix), 28
cbind.bigz (matrix), 28
character, *9*
chooseZ, *41*
chooseZ (factorialZ), 18
class, *20*, *36*
crossprod (matrix), 28
cumsum, *14*, 14

denominator (bigq), 6
denominator<- (bigq), 6
dim, *3*, *4*
dim.bigq (matrix), 28
dim.bigz (matrix), 28
dim<-.bigq (matrix), 28
dim<-.bigz (matrix), 28
dimnames, *3*
div.bigq (Bigq operators), 8
div.bigz (bigz operators), 12
divq.bigz (bigz operators), 12
double, *7*, *9*

Eulerian (Stirling), 39
extract, 16
Extremes, 17

factorial, *19*
factorialZ, 18
factorization, 19
factorize, *27*
factorize (factorization), 19
fibnum (lucnum), 27
fibnum2 (lucnum), 27
format, *20*, *21*
formatN, 20
frexp (frexpZ), 21
frexpZ, 21
function, *3*, *36*

gamma, *19*
gcd (gcd.bigz), 22
gcd.bigz, 22, *23*
gcdex, *23*, 23
gmp-ifiworkarounds, 24
gmp.utils, 24
gmpVersion (gmp.utils), 24

integer, *9*
inv (bigz operators), 12
is.bigq (bigq), 6

is.bigz (bigz), 9
is.integer, *25*
is.matrixZQ (matrix), 28
is.na.bigq (bigq), 6
is.na.bigz (bigz), 9
is.whole, *25*, 25
isprime, 26, *32*

lapply, *3*
lcm.bigz (gcd.bigz), 22
lcm.default (gcd.bigz), 22
length, *21*
length.bigq (extract), 16
length.bigz (extract), 16
length<-.bigq (extract), 16
length<-.bigz (extract), 16
list, *7*, *10*, *21*
log.bigz (bigz operators), 12
log10.bigz (bigz operators), 12
log2, *22*
log2.bigz (bigz operators), 12
lucnum, 27
lucnum2 (lucnum), 27

matrix, *4*, 28, *29*
matrix.bigz, *3*
max, *17*, *18*
max.bigq (Extremes), 17
max.bigz (Extremes), 17
methods, *2*, *17*
min, *17*
min.bigq (Extremes), 17
min.bigz (Extremes), 17
mod.bigz, *10*, *35*
mod.bigz (bigz operators), 12
modulus, *10*, 30
modulus<- (modulus), 30
mpfr, *3*
mul.bigq (Bigq operators), 8
mul.bigz (bigz operators), 12

NA, *10*, *13*, *15*
NA_bigq_ (bigq), 6
NA_bigz_ (bigz), 9
ncol.bigq (matrix), 28
ncol.bigz (matrix), 28
nextprime, *27*, 31
nrow.bigq (matrix), 28
nrow.bigz (matrix), 28
numerator (bigq), 6
numerator<- (bigq), 6
numeric, *4*, *7*, *9*

Oakley, 32
Oakley1 (Oakley), 32
Oakley2 (Oakley), 32
outer, *24*
outer (gmp-ifiworkarounds), 24

pow (bigz operators), 12
pow.bigq (Bigq operators), 8
pow.bigz, *33*
powm, 33
primes, *27*
print.bigq (bigq), 6
print.bigz (bigz), 9
prod, *14*
prod.bigq (cumsum), 14
prod.bigz (cumsum), 14

Random, 34
range, *17*
raw, *6*, *9*
rbind.bigq (matrix), 28
rbind.bigz (matrix), 28
Relational Operator, 35
rep, *16*
rep.bigq (extract), 16
rep.bigz (extract), 16
round, *36*
round.bigq (roundQ), 35
round0 (roundQ), 35
roundQ, 35
roundX, *36*

sapply, *7*, *10*
setOldClass, *6*, *9*
sign.bigq (Bigq), 5
sign.bigz (Relational Operator), 35
sizeinbase, 37
solve, *29*, *39*
solve.bigq (solve.bigz), 38
solve.bigz, *29*, 38
Stirling, 39
Stirling1 (Stirling), 39
Stirling2 (Stirling), 39
stop, *9*
sub.bigq (Bigq operators), 8

sub.bigz *(*bigz operators*)*, 12
sum, *14*
sum.bigq *(*cumsum*)*, 14
sum.bigz *(*cumsum*)*, 14

t.bigq *(*matrix*)*, 28
t.bigz *(*matrix*)*, 28
tcrossprod *(*matrix*)*, 28
TRUE, *29*
typeof, *3*, *4*

urand.bigz *(*Random*)*, 34

vector, *4*

which.max, *17*
which.max,bigq-method *(*Extremes*)*, 17
which.max,bigz-method *(*Extremes*)*, 17
which.min, *17*
which.min,bigq-method *(*Extremes*)*, 17
which.min,bigz-method *(*Extremes*)*, 17