Package ‘gmp’

February 20, 2015

Version 0.5-12
Date 2014-07-28
Title Multiple Precision Arithmetic
Author Antoine Lucas, Immanuel Scholz, Rainer Boehme <rb-gmp@reflex-studio.de>,
Sylvain Jasson <jasson@toulouse.inra.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).
Imports methods
Suggests Rmpfr
SystemRequirements gmp (>= 4.2.3)
License GPL
BuildResaveData no
LazyDataNote not available, as we use data/*.R *and* our classes
URL http://mulcyber.toulouse.inra.fr/projects/gmp
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-07-28 21:53:29

R topics documented:

  apply .......................................................... 2
  asNumeric ...................................................... 3
  Bigq ........................................................... 4
  bigq ............................................................ 5
  Bigq operators ................................................ 7
  bigz ............................................................. 8
  bigz operators ............................................... 11
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

```r
## 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`
asNumeric

**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**

```r
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**

```r
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))

---

**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
**bigq**  

**Arguments**  

\( \text{x, e1, e2} \) Object or vector of class **bigq**

**Examples**

\[
\begin{align*}
\text{x} & \leftarrow \text{as.bigq}(8000,21) \\
\text{x} & < 2 \times \text{x}
\end{align*}
\]

---

**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**

\[
\begin{align*}
\text{as.bigq(n, d = 1)} \\
\text{## S3 method for class 'bigq'} \\
\text{as.character(x, b=10,...)} \\
\text{## S3 method for class 'bigq'} \\
\text{as.double(x,...)} \\
\text{as.bigz.bigq(a, mod=NA)} \\
\text{is.bigq(x)} \\
\text{## S3 method for class 'bigq'} \\
\text{is.na(x)} \\
\text{## S3 method for class 'bigq'} \\
\text{print(x, quote=FALSE, initLine = TRUE, ...)} \\
\text{denominator(x)} \\
\text{numerator(x)}
\end{align*}
\]

**Arguments**

\[
\begin{align*}
\text{n,d} & \text{ either integer, numeric or string value (String value: either starting with 0x for hexadecimal, 0b for binary or without prefix for decimal values. Any format error results in 0). n stands for numerator, d for denominator} \\
\text{a} & \text{ an element of class "bigq"} \\
\text{mod} & \text{ optional modulus to convert into biginteger} \\
\text{x} & \text{ a “rational number” (vector), of class "bigq".} \\
\text{b} & \text{ base: from 2 to 36} \\
\text{...} & \text{ additional arguments passed to methods} \\
\text{quote} & \text{ (for printing:) logical indicating if the numbers should be quoted (as characters are); the default used to be TRUE (implicitly) till 2011.}
\end{align*}
\]
initLine  
(for printing:) logical indicating if an initial line (with the class and length or dimension) should be printed.

Details

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

Value

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

Author(s)

Antoine Lucas

References

http://mulcyber.toulouse.inra.fr/projects/gmp/

Examples

\begin{verbatim}
  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)

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

```r
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
References

http://mulcyber.toulouse.inra.fr/projects/gmp/

Examples

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

r <- as.bigz(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

```r
as.bigz(a, mod = NA)
## 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)), ...)
```

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 `initial` 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.
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 ">=".

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 \( \mathbb{Z}/m\mathbb{Z} \)”. For many operations, this means

\[
\text{result} \leftarrow \text{mod.bigz}(\text{result}, m) \quad \text{###} \quad \text{result} \bmod m
\]

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 multiplicative 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 \( a^b \bmod c \), but rather as.bigz(a, c) ^ b.

The following rules for the result’s modulus apply when performing arithmetic operations on bigzs:

- 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 \( \mathcal{R} \) object of (S3) class "bigz", representing the argument \( (x \) or \( a) \).

Note

\[
x \leftarrow \text{as.bigz}(1234567890123456789012345678901234567890)
\]

will not work as \( \mathcal{R} \) converts the number to a double, losing precision and only then convert to a "bigz" object.

Instead, use the syntax

\[
x \leftarrow \text{as.bigz}("123456789012345678901234567890")
\]
Author(s)
Immanuel Scholz

References
The GNU MP Library, see http://gmplib.org

Examples

```r
## 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

## 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..

## re create 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)
```

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

```r
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 = \text{div}(a, b) \text{ returns a rational number unless the operands have a (matching) modulus where a \* b^{-1} results.}

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

\text{options("gmp:warnNoInv" = FALSE)}

\textbf{Value}

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

\textbf{Author(s)}

Immanuel Scholz and Antoine Lucas

\textbf{References}

The GNU MP Library, see \url{http://gmplib.org}

\textbf{Examples}

```r
# 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 # = \text{divq}(a, d)
a \%\% d # = \text{mod.bigz}(a, d)

(ii <- inv.bigz(1:10, 16))
```
cumsum

(Cumulative) Sums, Products of Large Integers and Rationals

Description

These are methods to ‘overload’ the \texttt{sum()}, \texttt{cumsum()} and \texttt{prod()} functions for big rationals and big integers.

Usage

```r
## 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

- \texttt{x, \ldots} \texttt{R} objects of class \texttt{bigz} or \texttt{bigq} or ‘simple’ numbers.
- \texttt{na.rm} logical indicating if missing values (\texttt{NA}) should be removed before the computation.

Value

return an element of class \texttt{bigz} or \texttt{bigq}. 

```r
## 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)))
```
Author(s)
Antoine Lucas

See Also
apply

Examples

```r
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
```r
## 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.
Note

Unlike standard matrices, \texttt{x[i]} and \texttt{x[,]} do the same.

Examples

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

## create a vector of 3 a
c(a,a,a)

## repeat 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 \texttt{methods} for \texttt{min} and \texttt{max} for big rationals (\texttt{bigq}) and big integers (\texttt{biqz}); consequently, \texttt{range()} works as well.

Similarly, S4 methods are provided for \texttt{which.min()} and \texttt{which.max()}.
Usage

```r
## 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

return an element of class bigz or bigq.

Author(s)

Antoine Lucas

See Also

max

Examples

```r
x <- as.bigz(1:10)
max(x)
min(x)
rng(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)])
```
factorialZ

Description

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

Usage

\[
\text{factorialZ}(n) \\
\text{chooseZ}(n, k)
\]

Arguments

\( n \quad \text{non-negative integer (vector), for factorialZ. For chooseZ, may be a bigz big integer, also negative.} \)
\( k \quad \text{non-negative integer vector.} \)

Value

a vector of big integers, i.e., of class \texttt{bigz}.

See Also

\texttt{factorial} and \texttt{gamma} in base \texttt{R};

Examples

\[
\text{factorialZ}(0:10) \# 1 \ 1 \ 2 \ 6 \ldots \ 3628800 \\
\text{factorialZ}(0:40) \# \text{larger} \\
\text{factorialZ}(200)
\]

\( n \leftarrow \text{1000} \\
\text{fl000} \leftarrow \text{factorialZ}(n) \\
\text{stopifnot}(1e-15 > \text{abs(\text{as.numeric}(1 - \text{factorialZ}(n) / \log(\text{fl000})))))}
\]

\text{system.time(replicate(8, f1e4 \leftarrow \text{factorialZ}(10000)))} \\
\text{nchar(\text{as.character(f1e4))}) \# 35660 \ldots \text{(too many to even look at ..)}
\]

\( \text{chooseZ}(1000, 100:102) \# \text{vectorizes} \\
\text{chooseZ(\text{as.bigz}(2)^120, 10)} \\
\text{n} \leftarrow \text{c(50,80,100)} \\
\text{k} \leftarrow \text{c(20,30,40)} \\
\text{## currently with an undesirable warning: \% from methods/src/eval.c \_\_FIXME\_} \\
\text{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: either 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 http://gmplib.org

Examples

factorize(34455342)
**Format Numbers Keeping Classes Distinguishable**

**Description**

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

**Usage**

```r
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'
```

**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**

```r
## 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

\[
\text{frexpZ}(x)
\]

Arguments

\( x \)  
integer or big integer (\texttt{bigz}).

Value

a list with the two components

\[
\begin{align*}
\text{d} & \quad \text{a numeric vector whose absolute values are either zero, or in } [\frac{1}{2}, 1). \\
\text{exp} & \quad \text{an integer vector of the same length; note that } \text{exp} = 1 + \text{floor}(\log_2(x)), \\
& \quad \text{and hence always } \text{exp} > \log_2(x).
\end{align*}
\]

Author(s)

Martin Maechler

See Also

\texttt{log2}, etc; for \texttt{bigz} objects built on (the C++ equivalent of) \texttt{frexp()}, actually GMP’s ‘\texttt{mpz_get_d_2exp()}’.

Examples

\[
\begin{align*}
\text{frexpZ}(1:10) \\
\text{## and confirm:} \\
\text{with(frexpZ}(1:10), d \times 2^{\text{exp}}) \\
x \leftarrow \text{rpois}(1000, \text{lambda}=100) \times (1 + \text{rpois}(1000, \text{lambda}=10)) \\
X \leftarrow \text{as.bigz}(x) \\
\text{stopifnot(all.equal(x, with(frexpZ}(x), d(\times \text{2}^{\text{exp}})), \\
1 + \text{floor} (\log_2(x)) = (fx \leftarrow \text{frexpZ}(x)\$\text{exp}, \\
fx \leftarrow \text{frexpZ}(X)\$\text{exp}, \\
1 + \text{floor} (\log_2(X)) = fx)
\end{align*}
\]
**gcd.bigz**

*Greatest Common Divisor, Least Common Multiple*

**Description**

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

**Usage**

```r
## 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 [http://gmplib.org](http://gmplib.org)

**See Also**

`gcdex`

**Examples**

```r
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)
```
gcdex

Compute Bezout coefficient

Description

Compute \( g, s, t \) as \( as + bt = g = \text{gcd}(a, b) \). \( s \) and \( t \) are also known as Bezout coefficients.

Usage

\[
gcdex(a, b)
\]

Arguments

\( a, b \)

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

Value

3 values:

\( g, s, t \)

Elements of class bigz

Author(s)

Antoine Lucas

References

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

See Also

gcd.bigz

Examples

\[
gcdex(342, 654)
\]
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 http://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
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 being 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. \( \text{reps} \) 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 http://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

```r
isprime(210)
isprime(71)

# All primes numbers from 1 to 100
t <- isprime(1:99)
(1:99)[t > 0]
table(isprime(1:100000))# 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 http://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.
matrix

Usage

```r
## 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 $\mathbb{Z}/n\mathbb{Z}$, for the whole matrix. It is the only case where the `%%` and `solve` functions will work in $\mathbb{Z}/n\mathbb{Z}$.

**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

```r
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(S))

## 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(z*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)
```

### modulus

**Modulus of a Big Integer**

**Description**

The modulus of a bigz number \( a \) is “unset” when \( a \) is a regular integer, \( a \in \mathbb{Z} \). Or the modulus can be set to \( m \) which means \( a \in \mathbb{Z}/m \cdot \mathbb{Z} \), i.e., all arithmetic with \( a \) is performed ‘modulo m’.

**Usage**

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

**Arguments**

- **a**
  - \( \text{R object of class "bigz"} \)
- **value**
  - integer number or object of class "bigz".

**Examples**

```r
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)))
```
**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 [http://gmplib.org](http://gmplib.org)

**See Also**

`isprime` and its references and examples.

**Examples**

```r
nextprime(14)
## still very fast:
(p <- nextprime(107))
## 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
Oakley1
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"
szineinbase(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)
Random

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)

\[
x = \text{as.bigz}(4,9)
\]
\[
x ^ 7
\]

Description

Generate a uniformly distributed random number in the range 0 to \(2^{\text{size}} - 1\), 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 \(2^{\text{size}} - 1\)
seed  Bigz: random seed initialisation

Value

A biginteger of class bigz.
Relational Operator

Author(s)
Antoine Lucas

References
'mpz_urandomb' from the GMP Library, see http://gmplib.org

Examples

# Integers are different
urand.bigz()
urand.bigz()
urand.bigz()

# Integers are the same
urand.bigz(seed="23423423423423")
urand.bigz(seed="23423423423423")

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

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

\[
\begin{align*}
& x \leftarrow \text{as.bigz}(8000) \\
& x \cdot 300 < 2 \cdot x \\
& \text{sign(as.bigz(-3:3))} \\
& \text{sign(as.bigq(-2:2, 7))}
\end{align*}
\]

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

\[
\text{sizeinbase}(a, b=10)
\]

Arguments

\[
\begin{align*}
a & \quad \text{big integer, i.e. "bigz"} \\
\text{b} & \quad \text{base}
\end{align*}
\]

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 http://gmplib.org

Examples

\[
\begin{align*}
& \text{sizeinbase}(342434, 10) \# 6 \text{ obviously} \\
& \text{Iv} \leftarrow \text{as.bigz}(2:7) \cdot 500 \\
& \text{sizeinbase(Iv)} \\
& \text{stopifnot(sizeinbase(Iv)} \quad == \quad \text{nchar(as.character(Iv)),} \\
& \text{sizeinbase(Iv, b=16) \quad == \quad nchar(as.character(Iv, b=16))}
\end{align*}
\]
solve.bigz  Solve a system of equation

Description

This generic function solves the equation \( a \% \times x = b \) 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 \( \mathbb{Z}/n\mathbb{Z} \) if there is a common modulus, or a rational matrix if not.

Usage

```r
## 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 truncamuch 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 \( \mathbb{Z}/n\mathbb{Z} \) 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)

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 $1, 2, \ldots, n$ 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 $1, 2, \ldots, n$ with exactly $k$ cycles.

Stirling numbers of the second kind:
$S_n^{(k)} =$ 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($(0:(n-1)), \text{Eulerian, } n=n)$ (for $n > 0$),
Stirling1.all$(n)$ is the same as sapply($(1:n), \text{Stirling1, } n=n)$, and
Stirling2.all$(n)$ is the same as sapply($(1:n), \text{Stirling2, } n=n)$, but more efficient.

Note

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

Author(s)

Martin Maechler ("direct": May 1992)

References

Eulerians:

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

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), 4
!=.bigz (Relational Operator), 33
*Topic arithmetic
    Stirling, 36
*Topic arith
    apply, 2
    asNumeric, 3
    Bigq, 4
    bigq, 5
    Bigq operators, 7
    bigz, 8
    bigz operators, 11
    cumsum, 13
    extract, 14
    Extremes, 15
    factorialZ, 17
    factorization, 18
    frexpZ, 20
    gcd.bigz, 21
    gcdex, 22
    gmp.utils, 23
    isprime, 24
    lucnum, 25
    matrix, 26
    modulus, 29
    nextprime, 30
    powm, 31
    Random, 32
    Relational Operator, 33
    sizeinbase, 34
    solve.bigz, 35
*Topic character
    formatN, 19
*Topic data
    Oakley, 31
*Topic math
    is.whole, 23
*Topic methods
    asNumeric, 3

*Topic print
    formatN, 19
* .bigq (Bigq operators), 7
* .bigz (bigz operators), 11
+ .bigq (Bigq operators), 7
+ .bigz (bigz operators), 11
-. .bigq (Bigq operators), 7
-. .bigz (bigz operators), 11
/. .bigq (Bigq operators), 7
/. .bigz (bigz operators), 11
< .bigq (Bigq), 4
<. .bigz (Relational Operator), 33
<= .bigq (Bigq), 4
<=. .bigz (Relational Operator), 33
== .bigq (Bigq), 4
==. .bigz (Relational Operator), 33
>. .bigq (Bigq), 4
>. .bigz (Relational Operator), 33
>= .bigq (Bigq), 4
>=. .bigz (Relational Operator), 33
[. .bigq (extract), 14
[. .bigz (extract), 14
[<= .bigq (extract), 14
[<=. .bigz (extract), 14
[[. .bigq (extract), 14
[[. .bigz (extract), 14
[[<=. .bigq (extract), 14
[[<=. .bigz (extract), 14
%%% (matrix), 26
%/%.bigz (bigz operators), 11
%%.bigz (bigz operators), 11
^. .bigq (Bigq operators), 7
^. .bigz (bigz operators), 11
abs.bigq (Bigq operators), 7
abs.bigz (bigz operators), 11
add.bigq (Bigq operators), 7
add.bigz, 9
add.bigz (bigz operators), 11
apply, 2, 2, 3, 14
array, 3
as.bigq (bigq), 5
as.bigz (bigz), 8
as.bigz.bigq (bigq), 5
as.character, 19
as.character.bigq (bigq), 5
as.character.bigz (bigz), 8
as.double.bigq (bigq), 5
as.double.bigz (bigz), 8
as.matrix.bigq (matrix), 26
as.matrix.bigz (matrix), 26
as.numeric, 4
as.vector.bigq (matrix), 26
as.vector.bigz (matrix), 26
asNumeric, 3
asNumeric, ANY-method (asNumeric), 3
asNumeric, bigq-method (asNumeric), 3
asNumeric, bigz-method (asNumeric), 3
asNumeric-methods (asNumeric), 3
biginteger.as (bigz), 8
biginteger.as.character (bigz), 8
Bigq, 4
bigq, 5, 5, 7, 12
Bigq operators, 7
bigq-class (bigq), 5
bigz, 3, 7, 8, 11, 17, 20, 21, 27, 29, 31, 33, 34
bigz operators, 11
bigz-class (bigz), 8
c.bigq (extract), 14
c.bigz (extract), 14
cbind.bigq (matrix), 26
cbind.bigz (matrix), 26
character, 8
chooseZ, 37
chooseZ (factorialZ), 17
class, 19
crossprod (matrix), 26
cumsum, 13, 13
denominator (bigq), 5
denominator<= (bigq), 5
dim, 3, 4
dim.bigq (matrix), 26
dim.bigz (matrix), 26
dim<=.bigq (matrix), 26
dim<=.bigz (matrix), 26
dimnames, 3
div.bigq (Bigq operators), 7
div.bigz (bigz operators), 11
divq.bigz (bigz operators), 11
double, 8
Eulerian (Stirling), 36
extract, 14
Extremes, 15
factorial, 17
factorialZ, 17
factorization, 18
factorize, 25
factorize (factorization), 18
fibnum (lucnum), 25
fibnum2 (lucnum), 25
format, 19
formatN, 19
frexp (frexpZ), 20
frexpZ, 20
function, 2
gamma, 17
gcd (gcd.bigz), 21
gcd.bigz, 21, 22
gcdex, 21, 22
gmp.utils, 23
gmpVersion (gmp.utils), 23
integer, 8
inv (bigz operators), 11
is.bigq (bigq), 5
is.bigz (bigz), 8
is.integer, 24
is.matrixZQ (matrix), 26
is.na.bigq (bigq), 5
is.na.bigz (bigz), 8
is.whole, 23, 24
isprime, 24, 30
lapply, 2, 3
lcm (gcd.bigz), 21
length, 19
length.bigq (extract), 14
length.bigz (extract), 14
length<=.bigq (extract), 14
length<=.bigz (extract), 14
list, 20
log.bigz (bigz operators), 11
INDEX

log10.bigz (bigz operators), 11
log2, 20
log2.bigz (bigz operators), 11
lucnum, 25
lucnum2 (lucnum), 25

matrix, 3, 26, 28
matrix.bigz, 2
max, 15, 16
max.bigq (Extremes), 15
max.bigz (Extremes), 15
methods, 2, 15
min, 15
min.bigq (Extremes), 15
min.bigz (Extremes), 15
mod.bigz, 9, 33
mod.bigz (bigz operators), 11
modulus, 9, 29
modulus<-(modulus), 29
mpfr, 3
mul.bigq (Bigq operators), 7
mul.bigz (bigz operators), 11

NA, 9, 12, 13
ncol.bigq (matrix), 26
ncol.bigz (matrix), 26
nextprime, 25, 30
nrow.bigq (matrix), 26
nrow.bigz (matrix), 26
numerator (bigq), 5
numerator<-(bigq), 5
numeric, 4, 8

Oakley, 31
Oakley1 (Oakley), 31
Oakley2 (Oakley), 31

pow (bigz operators), 11
pow.bigq (Bigq operators), 7
pow.bigz, 32
powm, 31
primes, 25
print.bigq (bigq), 5
print.bigz (bigz), 8
prod, 13
prod.bigq (cumsum), 13
prod.bigz (cumsum), 13

Random, 32

range, 15
raw, 5, 8
rbind.bigq (matrix), 26
rbind.bigz (matrix), 26
Relational Operator, 33
rep, 15
rep.bigq (extract), 14
rep.bigz (extract), 14

setOldClass, 5, 8
sign.bigq (Bigq), 4
sign.bigz (Relational Operator), 33
sizeinbase, 34
solve, 27, 35
solve.bigq (solve.bigz), 35
solve.bigz, 28, 35
Stirling, 36
Stirling1 (Stirling), 36
Stirling2 (Stirling), 36
stop, 8
sub.bigq (Bigq operators), 7
sub.bigz (bigz operators), 11
sum, 13
sum.bigq (cumsum), 13
sum.bigz (cumsum), 13

t.bigq (matrix), 26
t.bigz (matrix), 26
tcrossprod (matrix), 26
TRUE, 28
typeof, 3

urand.bigz (Random), 32

vector, 4

which.max, 15
which.max, bigq-method (Extremes), 15
which.max, bigz-method (Extremes), 15
which.min, 15
which.min, bigq-method (Extremes), 15
which.min, bigz-method (Extremes), 15