

Package ‘gmp’

October 27, 2009

Version 0.4-11

Date 2009-10-27

Title Multiple Precision Arithmetic

Author Antoine Lucas, Immanuel Scholz, Rainer Boehme <rb-gmp@reflex-studio.de>, Sylvain Jasson <jasson@toulouse.inra.fr>

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.

Depends R (>= 2.0.0)

SystemRequirements gmp (>= 4.1.4)

License GPL

URL <http://mulcyber.toulouse.inra.fr/projects/gmp>

Repository CRAN

Date/Publication 2009-10-27 19:17:37

R topics documented:

apply	2
Bigq	3
bigq	3
Bigq operators	5
bigz	6
bigz operators	8
cumsum	9
Extract	10
Extremes	11
factorization	12
gcd.bigz	13

gcdex	14
isprime	15
lucnum	16
matrix	17
modulus	18
nextprime	19
Oakley	20
powm	20
Random	21
Relational Operator	22
sizeinbase	23
solve.bigz	23

Index	25
--------------	-----------

apply	<i>apply over a matrix</i>
-------	----------------------------

Description

Theses functions overload apply function for big rationals and big integers

Usage

```
apply(X, MARGIN, FUN)
```

Arguments

X	A matrix of class bigz or bigq
MARGIN	1: apply function on rows; 2: apply function of columns
FUN	function to be applied

Value

return a vector of class bigz or bigq.

Author(s)

Antoine Lucas

See Also

[apply](#)

Examples

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

x = as.bigq(matrix(1:12,3))
apply(x,1,min)
apply(x,2,max)
```

Bigq

Relational Operators

Description

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

Usage

```
sign.bigq(x)
```

Arguments

x Object or vector of class bigq

Examples

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

bigq

Large sized rationals

Description

Type class supporting arithmetic operations on very large rationals.

Usage

```
as.bigq(n, d = 1)
as.character.bigq(x,b=10,...)
as.double.bigq(x,...)
as.bigz.bigq(a,mod=NA)
is.na.bigq(x)
print.bigq(x,...)
denominator(x)
numerator(x)
```

Arguments

<code>n, d</code>	Either integer, numeric or string value (String value: either starting with <code>0x</code> for hexadecimal, <code>0b</code> for binary or without prefix for decimal values. Any format error results in 0). <code>n</code> stands for numerator, <code>d</code> for denominator
<code>a</code>	An element of class <code>bigq</code>
<code>mod</code>	Optional modulus to convert into biginteger
<code>x</code>	Numeric value
<code>b</code>	Base: from 2 to 32
<code>...</code>	Additional parameters

Details

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

Value

A `bigq` class representing the parameter value.

Author(s)

Antoine Lucas

References

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

Examples

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

y <- as.bigq(5,3)

# addition work !
x + y

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

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

Description

Addition, subtraction, multiplication, division.

Usage

```
add.bigq(a, b)
sub.bigq(a, b=NULL)
mul.bigq(a, b)
div.bigq(a, b)
abs.bigq(x)
```

Arguments

<code>a, b, x</code>	bigq, integer or string from an integer
<code>...</code>	Additional parameters

Details

Operators can be use directly when objects are of class bigq: $a + b$, $a * b$, etc.

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

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

bigz

*Large sized integer values***Description**

Type class supporting arithmetic operations on very large integers.

Usage

```
as.bigz(a, mod = NA)
as.character.bigz(x, b=10, ...)
as.double.bigz(x, ...)
is.na.bigz(x)
print.bigz(x, ...)
```

Arguments

a	Either integer, numeric or string value (String value: either starting with 0x for hexadecimal, 0b for binary, 0 for octal or without prefix for decimal values. Any format error results return an Error message)
b	Base: from 2 to 32
x	Numeric value
...	Additional parameters
mod	An integer, numeric, string or bigz of the internal modulus, see below.

Details

Bigzs are integers of infinite, 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 things that comes in need when performing standard cryptographic operations.

For a review of basic arithmetics, see "[add.bigz](#)".

The most important logical operators are supported, such as "=", "!", "<", "<=", ">", and ">=".

Objects of class "[bigz](#)" may have an attribute `mod` which specifies a modulus that is applied after each arithmetic operation. If the result is going to have a modulus,

```
result = mod.bigz(result, modulus)
```

is called after performing the arithmetic operation and the result will have the attribute `mod` set accordingly.

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, don't use `a ^ b %% c`, but use `as.bigz(a, c) ^ b`, instead.

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 operands value is used.
- If only one of the operands has a modulus or both have a common (the same), it is set and applied to the result, except in case of mod.bigz, where the second operands value is used.

Value

A bigz class representing the parameter value.

Note

`x = as.bigz(12345678901234567890123456789012345678901234567890)` will not work as integer will be first converted to a double and then converted to a "bigz" element.

Correct syntax: `x = as.bigz("1234567890123456789012345678901234567890")`

Author(s)

Immanuel Scholz

References

GNU MP Library see <http://swox.com/gmp>

Examples

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

## Not run:
## 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 <- as.bigz("0x4378a27...") # probably a product of two primes

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

# similar second way (maybe more sensefull if you reuse e) to do it right
modulus(e) <- n
c <- x ^ e

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

# WRONG! (although very beautiful. Maybe ok for small examples)
c <- x ^ e

# Return result in hexa
```

```
as.character(c,b=16)
## End(Not run)
```

bigz operators *Basic arithmetic operators for large integers*

Description

Addition, subtraction, multiplication, division, remainder of division, multiplicative inverse, power and logarithm functions.

Usage

```
add.bigz(a, b)
sub.bigz(a, b = NULL)
mul.bigz(a, b)
div.bigz(a, b)
divq.bigz(a,b)
mod.bigz(a, b)
abs.bigz(x)
inv.bigz(a, b, ...)
inv(a, ...)
pow.bigz(a, b, ...)
pow(a, ...)
log.bigz(x, base=exp(1))
log2.bigz(a)
log10.bigz(a)
```

Arguments

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

Details

For details about the internal modulus state, see the manpage of "bigz".

div or "/" return a rational number; divq or "%/%" return the quotient of division.

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

Value

A bigz class representing the result of the arithmetic operation.

Author(s)

Immanuel Scholz and Antoine Lucas

References

Gnu MP Library see <http://swox.com/gmp>

Examples

```
# 1+1=2
as.bigz(1) + 1

# if my_large_number_string is set to a number, it returns the least byte
## Not run:
mod.bigz(as.bigz(my_large_number_string), "0xff")
## End(Not run)

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

cumsum

Cummulative, product

Description

Theses functions overload cumsum and prod function for big rationals and big integers

Usage

```
cumsum.bigz(...)  
cumsum.bigq(...)  
prod(...)
```

Arguments

... an element of class bigz or bigq.

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)
```

```
x = as.bigq(1:12)
cumsum(x)
prod(x)
```

 Extract

Extract or Replace Parts of an Object

Description

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

Usage

```
c.bigz(..., recursive = FALSE)
rep.bigz(x, times, ...)
c.bigq(..., recursive = FALSE)
rep.bigq(x, times, ...)
```

Arguments

x	Object or vector of class bigz
...	Additional parameters
times	Integer
recursive	Unused

Note

Unlike standard matrix, operator `x[i]` & `x[i,]` do the same.

Examples

```
a <- as.bigz(123)
a[2 ] <- a[ 1]

## 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)
```

```
## this 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

Returns the maxima and minima of the input values.

Description

Theses functions overload min and max function for big rationals and big integers

Usage

```
max.bigz(..., na.rm=FALSE)
max.bigq(..., na.rm=FALSE)
min.bigz(..., na.rm=FALSE)
min.bigq(..., na.rm=FALSE)
```

Arguments

... 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

```
x = as.bigz(1:10)
##_call max.bigz => return 10
max(x)
min(x)

x = as.bigq(1:10)
##_call max.bigq
max(x)
min(x)
```

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

GNU MP Library see <http://swox.com/gmp>

Examples

```
factorize(34455342)
```

`gcd.bigz`*Great common divisor, Least common multiple*

Description

Great common divisor and least common multiple.

Usage

```
gcd.bigz(a, b)
lcm.bigz(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

An element of class `bigz`

Author(s)

Antoine Lucas

References

GNU MP Library see <http://swox.com/gmp>

See Also

[gcdex](#)

Examples

```
gcd.bigz(210, 342)
lcm.bigz(210, 342)
```

`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 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

GNU MP Library see <http://swox.com/gmp>

See Also

[gcd.bigz](#)

Examples

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

isprime *Determine if number is prime*

Description

Determine whether the number is prime or not. 2: number is prime, 1, number is probably prime (without being certain), 0 number is composite.

Usage

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

Arguments

n	Integer number, to be tested
reps	Integer, number of 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 a reasonable number. More will reduce the chances of a composite being returned as "probably prime".

Value

0	Number is not prime
1	Number is probably prime
2	Number is prime

Author(s)

Antoine Lucas

References

GNU MP Library see <http://swox.com/gmp>

See Also

[nextprime](#)

Examples

```
isprime(210)
isprime(71)

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

Gnu MP Library see <http://swox.com/gmp>

Examples

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

matrix

*Matrix manipulation with gmp***Description**

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

Usage

```
matrix.bigz(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL, mod = NA)
```

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
mod	Optional modulus
...	Not used

Details

Extract function is the same use for vector or matrix. Then, `x[i]` return same value as `x[i,]`.

Special features concerning bigz class: 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 mean you work in $\mathbb{Z}/n\mathbb{Z}$, for the whole matrix. It is the only case where `%*%` 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 modulus is repeated to the end of data. This can be used to define matrix with a different modulus at each row

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

Value

A matrix of class bigz or bigq

Author(s)

Antoine Lucas

See Also

Solving linear algebra system `solve.bigz`; `matrix`

Examples

```
## a matrix
x=diag(1:4)
## invert this matrix
solve(x)

## matrix in Z/7Z
y = as.bigz(x,7)
## invert this matrix (result is not the same as solve(x))
solve(y)

## matrix in Q
z = as.bigq(x)
## invert this matrix (result is the same as solve(x))
solve(z)
```

modulus

Modulus

Description

Access to modulus of a bigz

Usage

```
modulus(a, value)
```

Arguments

a	Object of class bigz
value	Object of class bigz

Examples

```
x <- as.bigz(24)
modulus(x)

# x element of Z/31Z
modulus(x) <- 31
x+x
# (6)
```

nextprime	<i>Next prime number</i>
-----------	--------------------------

Description

Return next prime number greater than n

Usage

```
nextprime(n)
```

Arguments

n	Integer
---	---------

Details

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

Value

A (probably) prime number

Author(s)

Antoine Lucas

References

Gnu MP Library see <http://swox.com/gmp>

See Also

[isprime](#)

Examples

```
nextprime(14)
```

Oakley	<i>RFC 2409 Oakley Groups - Standardized Parameters for Diffie-Hellman Key Exchange</i>
--------	---

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

```
data(Oakley1)
isprime(modulus(Oakley1))
```

powm	<i>Exponentiation function</i>
------	--------------------------------

Description

This function return $x^y \bmod n$

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

Details

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

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 Gnu MP Library see <http://swox.com/gmp>

Examples

```
# Integers are different
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

```
sign.bigz(x)
```

Arguments

x Object or vector of class bigz

Examples

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

sizeinbase *Compute size of a bigz in a base*

Description

Return size (number of digits) written in base b.

Usage

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

Arguments

a	Integer
b	base

Value

Integer: size (number of digits).

Author(s)

Antoine Lucas

References

GNU MP Library see <http://swox.com/gmp>

Examples

```
sizeinbase(342434, 10)
```

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 $\mathbb{Z}/n\mathbb{Z}$ if there is a common modulus, or a rational matrix if not.

Usage

```
solve.bigz(a, b, ...)  
solve.bigq(a, b, ...)
```

Arguments

<code>a, b</code>	A element of class bigz or bigq
<code>...</code>	Unused

Details

It uses the Gauss and trucmuch algo ... to be detailed.

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  $\mathbb{Z}/7\mathbb{Z}$   
solve(z)  
  
b = c(1,3)  
  
solve(q,b)  
  
solve(z,b)
```

Index

`!=.bigq(Bigq)`, 3
`!=.bigz(Relational Operator)`, 22
***Topic arith**
 `apply`, 2
 `Bigq`, 3
 `bigq`, 3
 `Bigq operators`, 5
 `bigz`, 6
 `bigz operators`, 8
 `cumsum`, 9
 `Extract`, 10
 `Extremes`, 11
 `factorization`, 12
 `gcd.bigz`, 13
 `gcdex`, 14
 `isprime`, 15
 `lucnum`, 16
 `matrix`, 17
 `modulus`, 18
 `nextprime`, 19
 `powm`, 20
 `Random`, 21
 `Relational Operator`, 22
 `sizeinbase`, 23
 `solve.bigz`, 23
***Topic data**
 `Oakley`, 20
 `*.bigq(Bigq operators)`, 5
 `*.bigz(bigz operators)`, 8
 `+.bigq(Bigq operators)`, 5
 `+.bigz(bigz operators)`, 8
 `-.bigq(Bigq operators)`, 5
 `-.bigz(bigz operators)`, 8
 `/.bigq(Bigq operators)`, 5
 `/.bigz(bigz operators)`, 8
 `<.bigq(Bigq)`, 3
 `<.bigz(Relational Operator)`, 22
 `<=.bigq(Bigq)`, 3
 `<=.bigz(Relational Operator)`, 22
 `==.bigq(Bigq)`, 3
 `==.bigz(Relational Operator)`, 22
 `>.bigq(Bigq)`, 3
 `>.bigz(Relational Operator)`, 22
 `>=.bigq(Bigq)`, 3
 `>=.bigz(Relational Operator)`, 22
 `[.bigq(Extract)`, 10
 `[.bigz(Extract)`, 10
 `[<-.bigq(Extract)`, 10
 `[<-.bigz(Extract)`, 10
 `[[.bigq(Extract)`, 10
 `[[.bigz(Extract)`, 10
 `[[<-.bigq(Extract)`, 10
 `[[<-.bigz(Extract)`, 10
 `%*%(matrix)`, 17
 `%/.bigq(Bigq operators)`, 5
 `%/.bigz(bigz operators)`, 8
 `%%.bigz(bigz operators)`, 8
 `^.bigz(bigz operators)`, 8
 `abs.bigq(Bigq operators)`, 5
 `abs.bigz(bigz operators)`, 8
 `add.bigq(Bigq operators)`, 5
 `add.bigz`, 6
 `add.bigz(bigz operators)`, 8
 `apply`, 2, 2, 9
 `as.bigq(bigq)`, 3
 `as.bigz(bigz)`, 6
 `as.bigz.bigq(bigq)`, 3
 `as.character.bigq(bigq)`, 3
 `as.character.bigz(bigz)`, 6
 `as.double.bigq(bigq)`, 3
 `as.double.bigz(bigz)`, 6
 `as.matrix.bigq(matrix)`, 17
 `as.matrix.bigz(matrix)`, 17
 `as.vector.bigq(matrix)`, 17
 `as.vector.bigz(matrix)`, 17
 `Bigq`, 3
 `bigq`, 3

- Bigq operators, 5
- bigz, 6, 6, 20
- bigz operators, 8
- c.bigq (*Extract*), 10
- c.bigz (*Extract*), 10
- cbind.bigq (*matrix*), 17
- cbind.bigz (*matrix*), 17
- cumsum, 9
- denominator (*bigq*), 3
- denominator<- (*bigq*), 3
- dim.bigq (*matrix*), 17
- dim.bigz (*matrix*), 17
- dim<-.bigq (*matrix*), 17
- dim<-.bigz (*matrix*), 17
- div.bigq (*Bigq operators*), 5
- div.bigz (*bigz operators*), 8
- divq.bigz (*bigz operators*), 8
- Extract, 10
- Extremes, 11
- factorization, 12
- factorize (*factorization*), 12
- fibnum (*lucnum*), 16
- fibnum2 (*lucnum*), 16
- gcd (*gcd.bigz*), 13
- gcd.bigz, 13, 14
- gcdex, 13, 14
- inv (*bigz operators*), 8
- is.na.bigq (*bigq*), 3
- is.na.bigz (*bigz*), 6
- isprime, 15, 19
- lcm (*gcd.bigz*), 13
- length.bigq (*Extract*), 10
- length.bigz (*Extract*), 10
- length<-.bigq (*Extract*), 10
- length<-.bigz (*Extract*), 10
- log.bigz (*bigz operators*), 8
- log10.bigz (*bigz operators*), 8
- log2.bigz (*bigz operators*), 8
- lucnum, 16
- lucnum2 (*lucnum*), 16
- matrix, 17, 17
- max, 11
- max.bigq (*Extremes*), 11
- max.bigz (*Extremes*), 11
- min.bigq (*Extremes*), 11
- min.bigz (*Extremes*), 11
- mod.bigz, 7
- mod.bigz (*bigz operators*), 8
- modulus, 18
- modulus<- (*modulus*), 18
- mul.bigq (*Bigq operators*), 5
- mul.bigz (*bigz operators*), 8
- ncol.bigq (*matrix*), 17
- ncol.bigz (*matrix*), 17
- nextprime, 15, 19
- nrow.bigq (*matrix*), 17
- nrow.bigz (*matrix*), 17
- numerator (*bigq*), 3
- numerator<- (*bigq*), 3
- Oakley, 20
- Oakley1 (*Oakley*), 20
- Oakley2 (*Oakley*), 20
- pow (*bigz operators*), 8
- pow.bigz, 21
- powm, 20
- print.bigq (*bigq*), 3
- print.bigz (*bigz*), 6
- prod (*cumsum*), 9
- Random, 21
- rbind.bigq (*matrix*), 17
- rbind.bigz (*matrix*), 17
- Relational Operator, 22
- rep.bigq (*Extract*), 10
- rep.bigz (*Extract*), 10
- sign.bigq (*Bigq*), 3
- sign.bigz (*Relational Operator*), 22
- sizeinbase, 23
- solve, 24
- solve.bigq (*solve.bigz*), 23
- solve.bigz, 17, 23
- sub.bigq (*Bigq operators*), 5
- sub.bigz (*bigz operators*), 8
- t.bigq (*matrix*), 17
- t.bigz (*matrix*), 17
- urand.bigz (*Random*), 21