Package ‘numbers’

February 15, 2013

Type Package

Title Number-theoretic Functions

Version 0.3-3

Date 2012-11-20

Author Hans W Borchers

Maintainer HwB <hwborchers@googlemail.com>

Depends R (>= 2.11.1)

Description Provides some number-theoretic functions for R.

License GPL (>= 3)

Repository CRAN

Repository/R-Forge/Project optimist

Repository/R-Forge/Revision 252

Repository/R-Forge/DateTimeStamp 2012-11-20 22:36:55

Date/Publication 2012-11-21 12:11:09

NeedsCompilation no

R topics documented:

numbers-package ........................................ 2
agm ......................................................... 3
chinese remainder theorem ................................ 5
contFrac ................................................... 6
coprime .................................................... 7
div ........................................................... 7
droplet_e .................................................... 8
eyptian_complete ........................................... 9
Index

numbers-package Number-theoretic Functions for R

Description

More about what it does (maybe more than one line) ~~ A concise (1-5 lines) description of the package ~~

Details

Package: numbers
Type: Package
Version: 0.2.9
Date: 2012-10-02
License: GPL (>= 3)
**Description**

The arithmetic-geometric mean of positive numbers.

**Usage**

\[
\text{agm}(a, b, \text{maxiter} = 25, \text{tol} = \text{.Machine$double$eps}^{(1/2)})
\]

**Arguments**

- `a, b`: Positive numbers.
- `maxiter`: Maximum number of iterations.
- `tol`: Tolerance; stops when \( |a-b| < \text{tol} \).

**Details**

The arithmetic-geometric mean is defined as the common limit of the two sequences

\[
a_{n+1} = \frac{a_n + b_n}{2}
\]

\[
b_{n+1} = \sqrt{a_n b_n}
\]

**Value**

Returns one value, the mean of the last two values `a, b`.

**Note**

The AGM is also defined for negative values `a` and/or `b`, but then the complex square root has to be taken.

**References**


**See Also**

Arithmetic, geometric, and harmonic mean.
Examples

```r
## Gauss constant
1 / agm(1, sqrt(2), tol = 1e-15)$agm # 0.8346268416747073

## Calculate the (elliptic) integral 2/pi \[ \int_0^1 dt / \sqrt{1 - t^4} \]

f <- function(t) 1 / sqrt(1-t^4)
2 / pi * integrate(f, 0, 1)$value

## Calculate pi with quadratic convergence (modified AGM)
# See algorithm 2.1 in Borwein and Borwein
y <- sqrt(sqrt(2))
x <- (y+1/y)/2
p <- 2+sqrt(2)
for (i in 1:6){
  cat(format(p, digits=16), "\n")
  p <- p * (1+x) / (1+y)
  s <- sqrt(x)
y <- (y*s + 1/s) / (1+y)
x <- (s+1/s)/2
}

## Not run:
## Calculate pi with arbitrary precision using the Rmpfr package
require("Rmpfr")
vpa <- function(., d = 32) mpfr(., precBits = 4*d)
# Function to compute \pi to d decimal digits accuracy, based on the
# algebraic-geometric mean, correct digits are doubled in each step.
agm_pi <- function(d) {
  a <- vpa(1, d)
b <- 1/sqrt(vpa(2, d))
s <- 1/vpa(4, d)
p <- 1
n <- ceiling(log2(d));
for (k in 1:n) {
  c <- (a+b)/2
  b <- sqrt(a*b)
s <- s - p * (c-a)^2
  p <- 2 * p
  a <- c
}
return(a^2/s)
}
d <- 64
pia <- agm_pi(d)
print(pia, digits = d)
# 3.141592653589793238462643383279502884197169399375105820974944592
# 3.1415926535897932384626433832795028841971693993751058209749445923 exact
```

## End(Not run)
Description

Executes the Chinese Remainder Theorem (CRT).

Usage

\texttt{chinese(a, m)}

Arguments

\begin{itemize}
  \item \texttt{a} sequence of integers, same length as \texttt{m}.
  \item \texttt{m} sequence of integers, relatively prime to each other.
\end{itemize}

Details

The Chinese Remainder Theorem says that given integers \(a_i\) and natural numbers \(m_i\), relatively prime (i.e., coprime) to each other, there exists a unique solution \(x = x_i\) such that the following system of linear modular equations is satisfied:

\[ x_i = a_i \mod m_i, \quad 1 \leq i \leq n \]

More generally, a solution exists if the following condition is satisfied:

\[ a_i = a_j \mod \gcd(m_i, m_j) \]

This version of the CRT is not yet implemented.

Value

Returns the (unique) solution of the system of modular equalities as an integer between 0 and \(M = \prod(m)\).

See Also

\texttt{extGCD}

Examples

\begin{verbatim}
  m <- c(3, 4, 5)
  a <- c(2, 3, 1)
  chinese(a, m) #=> 11

  # ... would be sufficient
  # m <- c(50, 210, 154)
  # a <- c(44, 34, 132)
  # x = 4444
\end{verbatim}
Description
Evaluate a continuous fraction or generate one.

Usage
contFrac(x, tol = 1e-6)

Arguments
x a numeric scalar or vector.
tol tolerance; default 1e-6 to make a nicer appearance for pi.

Details
If x is a scalar its continuous fraction will be generated up to the accuracy prescribed in tol. If it is of length greater 1, the function assumes this is a continuous fraction and computes its value.
For implementation contfrac uses the representation of continuous fractions through 2-by-2 matrices, i.e. the recursion formula.

Value
Either a numeric value, or a list with components cf, numeric vector representing the continuous fraction \([b_0; b_1, \ldots, b_{n-1}]\); rat, the rational number as a vector with (numerator, denominator); and prec, the difference between x and the value of the continuous fraction.

Note
This function is not vectorized.

References

See Also
ratFarey

Examples
contFrac(pi)
contFrac(c(3, 7, 15, 1))
### coprime

**coprime**  

**Coprinality**

**Description**
Determine whether two numbers are coprime, i.e. do not have a common prime divisor.

**Usage**
coprime(n,m)

**Arguments**
n, m integer scalars

**Details**
Two numbers are coprime iff their greatest common divisor is 1.

**Value**
Logical, being TRUE if the numbers are coprime.

**See Also**
GCD

**Examples**
coprime(46368, 75/zero.noslash25)  # Fibonacci numbers are relatively prime to each other  
coprime(1/zero.noslash/zero.noslash1, 1334)

### div

**Integer Division**

**Description**
Integer division.

**Usage**
div(n, m)

**Arguments**
n numeric vector (preferably of integers)  
m integer vector (positive, zero, or negative)
div(n, m) is integer division, that is discards the fractional part, with the same effect as n %%/% m. It can be defined as floor(n/m).

Value
A numeric (integer) value or vector/matrix.

See Also
mod, rem

Examples
```r
div(c(-5:5), 5)
div(c(-5:5), -5)
div(c(1, -1), 0)  #=> Inf -Inf
div(0, c(0, 1))  #=> NaN 0
```

---

**droplet_e**

**Droplet for e**

Description
Generates digits for the Euler number e.

Usage
droplet_e(n)

Arguments

- **n**
  number of digits after the decimal point; should not exceed 1000 much as otherwise it will be very slow.

Details
Based on a formula discovered by S. Rabinowitz and S. Wagon.

Value
String containing “2.718281828...” with n digits after the decimal point.

References
Egyptian Fractions - Complete Search

Description

Generate all Egyptian fractions of length 2 and 3.

Usage

egyptian_complete(a, b)

Arguments

a, b: integers, a != 1, a < b and a, b relatively prime.

Details

For a rational number 0 < a/b < 1, generates all Egyptian fractions of length 2 and three, that is finds integers x1, x2, x3 such that

\[
a/b = 1/x_1 + 1/x_2 + 1/x_3
\]

Value

No return value, all solutions found will be printed to the console.

References

http://www.ics.uci.edu/~eppstein/numth/egypt/

See Also

egyptian_methods

Examples

egyptian_complete(6, 7)  # 1/2 + 1/3 + 1/42
egyptian_complete(8, 11) # no solution found

# TODO
# 2/9 = 1/9 + 1/10 + 1/90 is not recognized
# 13/30 = 1/3 + 1/(3*4) + 1/(3*5) ="-

Examples
egyptian_methods

**Description**
Generate Egyptian fractions with specialized methods.

**Usage**
```
egyptian_methods(a, b)
```

**Arguments**
a, b integers, a ≠ 1, a < b and a, b relatively prime.

**Details**
For a rational number 0 < a/b < 1, generates Egyptian fractions that is finds integers x1, x2, ..., xk such that
\[ a/b = 1/x_1 + 1/x_2 + \ldots + 1/x_k \]
using the following methods:
- 'greedy'
- Fibonacci-Sylvester
- Golomb (same as with Farey sequences)
- continued fractions (not yet implemented)

**Value**
No return value, all solutions found will be printed to the console.

**References**
http://www.ics.uci.edu/~eppstein/numth/egypt/

**See Also**
egyptian_complete

**Examples**
```
egyptian_methods(8, 11)
# 8/11 = 1/2 + 1/5 + 1/37 + 1/4070 (Fibonacci-Sylvester)
# 8/11 = 1/2 + 1/6 + 1/21 + 1/77 (Golomb-Farey)
# Other solutions
# 8/11 = 1/2 + 1/8 + 1/11 + 1/88
# 8/11 = 1/2 + 1/12 + 1/22 + 1/121
```
Euler's Phi Function

Description
Euler's Phi function (aka Euler's 'totient' function).

Usage
\texttt{eulersPhi(n)}

Arguments
\texttt{n} Positive integer.

Details
The \textit{phi} function is defined to be the number of positive integers less than or equal to \texttt{n} that are \textit{coprime} to \texttt{n}, i.e. have no common factors other than 1.

Value
Natural number, the number of coprime integers \(\leq n\).

Note
Works well up to \(10^9\).

See Also
\texttt{factorize, sigma}

Examples
\begin{verbatim}
eulersPhi(9973) == 9973 - 1  # for prime numbers
eulersPhi(3^10) == 3^9 * (3 - 1)  # for prime powers
eulersPhi(12*35) == eulersPhi(12) * eulersPhi(35)  # TRUE if coprime
\end{verbatim}

\begin{verbatim}
## Not run:
x <- 1:100; y <- sapply(x, eulersPhi)
plot(1:100, y, type="l", col="blue",
     xlab="n", ylab="phi(n)", main="Euler's totient function")
points(1:100, y, col="blue", pch=20)
grid()
## End(Not run)
\end{verbatim}
**extGCD**

*Extended Euclidean Algorithm*

**Description**

The extended Euclidean algorithm computes the greatest common divisor and solves Bezout’s identity.

**Usage**

`extGCD(a, b)`

**Arguments**

- `a, b` integer scalars

**Details**

The extended Euclidean algorithm not only computes the greatest common divisor $d$ of $a$ and $b$, but also two numbers $n$ and $m$ such that $d = na + mb$.

This algorithm provides an easy approach to computing the modular inverse.

**Value**

A numeric vector of length three, `c(d, n, m)`, where $d$ is the greatest common divisor of $a$ and $b$, and $n$ and $m$ are integers such that $d = na + mb$.

**Note**

There is also a shorter, more elegant recursive version for the extended Euclidean algorithm. For R the procedure suggested by Blankinship appeared more appropriate.

**References**


**See Also**

- `GCD`

**Examples**

```r
extGCD(12, 10)  # Fibonacci numbers are relatively prime to each other
```
factorize

**Prime Factors**

Description

Returns a vector containing the prime factors of \( n \).

Usage

\[
\text{factorize}(n)
\]

Arguments

\[ n \quad \text{nonnegative integer} \]

Details

Computes the prime factors of \( n \) in ascending order, each one as often as its multiplicity requires, such that \( n = \prod \text{factorize}(n) \).

Value

Vector containing the prime factors of \( n \).

See Also

`isPrime`, `Primes`

Examples

\[
\begin{align*}
\text{factorize}(1002001) & \quad # \text{7 7 11 11 13 13} \\
\text{factorize}(65537) & \quad # \text{is prime} \\
& \quad # \text{Euler's calculation} \\
\text{factorize}(2^{32} + 1) & \quad # 641 6700417
\end{align*}
\]

fibonacci

**Fibonacci and Lucas Series**

Description

Generates single Fibonacci numbers or a Fibonacci sequence; generates a Lucas series based on the Fibonacci series.

Usage

\[
\begin{align*}
\text{fibonacci}(n, \text{sequence} = \text{TRUE}) \\
\text{lucas}(n)
\end{align*}
\]
Arguments

\- \text{\texttt{\texttt{n}}} \quad \text{integer.}
\- \text{\texttt{sequence}} \quad \text{logical; default: True.}

Details

Generates the \texttt{n}-th Fibonacci number, or the whole Fibonacci sequence from the first to the \texttt{n}-th number. Generates only the sequence of Lucas numbers.

The recursive version (sequence=\texttt{FALSE}) is extremely slow for values \texttt{n}>=30. To get the \texttt{n}-th Fibonacci number for larger \texttt{n} values, use \texttt{fibonacci(n)[n]}.

Value

A single integer, or a vector of integers.

Examples

\begin{verbatim}
fibonacci(0)      \quad \# 1
fibonacci(2)      \quad \# 2
fibonacci(2, sequence = TRUE)  \quad \# 1 2

lucas(0)         \quad \# 2
lucas(2)         \quad \# 1 3

# Golden ratio
F <- fibonacci(25, sequence = TRUE)  \quad \# \ldots 75025 121393
f25 <- F[25]/F[24]  \quad \# 1.618033989
phi <- (sqrt(5) + 1)/2
abs(f25 - phi)  \quad \# 7.945178e-11

# Fibonacci numbers w/o iteration
fibo <- function(n) {
  phi <- (sqrt(5) + 1)/2
  fib <- (phi^((n+1)) - (1-phi)^((n+1))) / (2*phi - 1)
  round(fib)
}
fibo(30:33) \quad \# 1346269 2178309 3524578 572887

luca <- function(n) {
  phi <- (sqrt(5) + 1)/2
  floor(phi^n + 1/2)
}
luca(30:33) \quad \# 1860498 3010349 4870847 7881196

# Compare recursive with iterative approach:
# system.time(F30 <- fibonacci(30, sequence = FALSE) \quad \# user: 17.075 s
# system.time(F30 <- fibonacci(30, sequence = TRUE)[30]) \quad \# user: 0.006 s
\end{verbatim}
GCD, LCM

GCD and LCM Integer Functions

Description

Greatest common divisor and least common multiple

Usage

GCD(n, m)
LCM(n, m)
mGCD(x)
mLCM(x)

Arguments

n, m integer scalars.

x a vector of integers.

Details

Computation based on the Euclidean algorithm without using the extended version.
mGCD (the multiple GCD) computes the greatest common divisor for all numbers in the integer vector x together.

Value

A numeric (integer) value.

Note

The following relation is always true:

n * m = GCD(n, m) * LCM(n, m)

See Also

extGCD, coprime

Examples

GCD(12, 10)
GCD(46368, 75025) # Fibonacci numbers are relatively prime to each other

LCM(12, 10)
LCM(46368, 75025) # = 46368 * 75025

mGCD(c(2, 3, 5, 7) * 11)
Hermite normal form

\texttt{mGCD(c(2*3, 3*5, 5*7))}
\texttt{mLCM(c(2, 3, 5, 7) * 11)}
\texttt{mLCM(c(2*3, 3*5, 5*7))}

\textbf{Description}

Hermite normal form over integers (in column-reduced form).

\textbf{Usage}

\texttt{hermiteNF(A)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{A} integer matrix.
\end{itemize}

\textbf{Details}

An \(mxn\)-matrix of rank \(r\) with integer entries is said to be in Hermite normal form if:

(i) the first \(r\) columns are nonzero, the other columns are all zero;
(ii) The first \(r\) diagonal elements are nonzero and \(d[i-1]\) divides \(d[i]\) for \(i = 2,...,r\).
(iii) All entries to the left of nonzero diagonal elements are non-negative and strictly less than the corresponding diagonal entry.

The lower-triangular Hermite normal form of \(A\) is obtained by the following three types of column operations:

(i) exchange two columns
(ii) multiply a column by \(-1\)
(iii) Add an integral multiple of a column to another column

\(U\) is the unitary matrix such that \(AU = H\), generated by these operations.

\textbf{Value}

List with two matrices, the Hermite normal form \(H\) and the unitary matrix \(U\).

\textbf{Note}

Another normal form often used in this context is the Smith normal form.

\textbf{References}

isIntpower

Powers of Integers

Is an integer a power of another integer?

Description

Determine whether p is the power of an integer.

Examples

n <- 4; m <- 5
A = matrix(c(9, 6, 0, -8, 0,-5, -8, 0, 0, 0, 0, 0, 0, 0,-5, 0), n, m, byrow = TRUE)

Hnf <- hermiteNF(A); Hnf
# $H = 1 2 0 0 0
# 28 36 84 0 0
# -35 -45 -105 0 0
# $U = 11 14 32 0 0
# 7 9 21 0 0
# 0 0 0 0 1

r <- 3
H <- Hnf$H; U <- Hnf$U
all(H == A %*% U) #=> TRUE

Examples

n <- 4; m <- 5
A = matrix(c(9, 6, 0, -8, 0,-5, -8, 0, 0, 0, 0, 0, 0, 0,-5, 0), n, m, byrow = TRUE)

Hnf <- hermiteNF(A); Hnf
# $H = 1 2 0 0 0
# 28 36 84 0 0
# -35 -45 -105 0 0
# $U = 11 14 32 0 0
# 7 9 21 0 0
# 0 0 0 0 1

r <- 3
H <- Hnf$H; U <- Hnf$U
all(H == A %*% U) #=> TRUE

## Example: Compute integer solution of A x = b
# H = A * U, thus H * U^-1 * x = b, or H * y = b
b <- as.matrix(c(-11, -21, 16, -2/zero.noslash))

y <- numeric(m)
for (i in 2:r)
  y[i] <- (b[i] - sum(H[i, 1:(i-1)] * y[1:(i-1)])) / H[i, i]
# special solution:
xs <- U %*% y
# and the general solution is xs + U * c(0, 0, 0, a, b), or
# in other words the basis are the m-r vectors c(0,...,0, 1, ...).
# If the special solution is not integer, there are no integer solutions.
Usage

\texttt{isIntpower(p)}

Arguments

\texttt{p} \hspace{1cm} \text{any integer number.}

Details

Determines whether \( p \) is the power of an integer and returns a tuple \((n, m)\) such that \( p = n^m \) where \( m \) is as small as possible. E.g., if \( p \) is prime it returns \((p, 1)\).

Value

A 2-vector of integers.

Examples

\begin{verbatim}
isIntpower(1)    # 1 1
isIntpower(15)   # 15 1
isIntpower(17)   # 17 1
isIntpower(64)   # 8 2
isIntpower(36)   # 6 2
isIntpower(100)  # 10 2
\end{verbatim}

\texttt{## Not run:}

\begin{verbatim}
for (p in 5^7:7^5) {
    pp <- isIntpower(p)
    if (pp[2] != 1) cat(p, ":	", pp, "\n")
}
\texttt{## End(Not run)}

\end{verbatim}

\[
\begin{array}{lll}
\text{isNatural} & \text{Natural Number} \\
\end{array}
\]

Description

Natural number type.

Usage

\texttt{isNatural(n)}

Arguments

\texttt{n} \hspace{1cm} \text{any numeric number.}

Details

Returns \texttt{TRUE} for natural (or: whole) numbers between 1 and \( 2^{53}-1 \).
isPrime

Value

Boolean

Examples

\[
\text{IsNatural} \leftarrow \text{Vectorize(isNatural)}
\]
\[
\text{IsNatural}(c(-1, 0, 1, 5.1, 10, 2^{53}-1, 2^{53}, \text{Inf}, \text{NA}))
\]

<table>
<thead>
<tr>
<th>isPrime</th>
<th>isPrime Property</th>
</tr>
</thead>
</table>

Description

Vectorized version, returning for a vector or matrix of positive integers a vector of the same size containing 1 for the elements that are prime and 0 otherwise.

Usage

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

Arguments

\[x\] vector or matrix of nonnegative integers

Details

Given an array of positive integers returns an array of the same size of 0 and 1, where the \(i\) indicates a prime number in the same position.

Value

array of elements 0, 1 with 1 indicating prime numbers

See Also

factorize, Primes

Examples

\[
x \leftarrow \text{matrix}(1:10, \text{nrow}=10, \text{ncol}=10, \text{byrow}=\text{TRUE})
x \times \text{isPrime}(x)
\]

# Find first prime number octett:
\[
\text{octett} \leftarrow c(0, 2, 6, 8, 30, 32, 36, 38) - 19
\]
while (TRUE) {
    \[
    \text{octett} \leftarrow \text{octett} + 210
    \]
    if (all(isPrime(octett))) {
        \[
        \text{cat(octett, "\n", sep=" ")}
        \]
        break
    }
Modulo Operator

Description
Modulo operator.

Usage

mod(n, m)

Arguments

n numeric vector (preferably of integers)
m integer vector (positive, zero, or negative)

Details

mod(n, m) is the modulo operator and returns $n \mod m$. mod(n, 0) is n, and the result always has the same sign as m.

Value

a numeric (integer) value or vector/matrix

Note

The following relation is fulfilled (for m $\neq 0$):

$mod(n, m) = n - m \times floor(n/m)$

See Also

rem, div

Examples

mod(c(-5:5), 5)
mod(c(-5:5), -5)
mod(0, 1) #=> 0
mod(1, 0) #=> 1
**modinv**

*Modular Inverse*

**Description**

Computes the modular inverse of \( n \) modulo \( m \).

**Usage**

\[ \text{modinv}(n, m) \]

**Arguments**

\( n, m \) integer scalars

**Details**

The modular inverse of \( n \) modulo \( m \) is the unique natural number \( 0 < n_0 < m \) such that \( n \times n_0 = 1 \mod m \).

**Value**

a natural number smaller \( m \), if \( n \) and \( m \) are coprime, else NA.

**See Also**

extGCD

**Examples**

\[ \text{modinv}(5, 1001) \Rightarrow 801, \text{ as } 5 \times 801 = 4005 = 1 \mod 1001 \]

\[ \text{Modinv} \leftarrow \text{Vectorize}(\text{modinv}, \"n\") \]

\[ ((1:10) \times \text{Modinv}(1:10, 11)) \mod 11 \Rightarrow 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \]

---

**modlin**

*Modular Linear Equation Solver*

**Description**

Solves the modular equation \( a \times x = b \mod n \).

**Usage**

\[ \text{modlin}(a, b, n) \]
modpower

Arguments

a, b, n integer scalars

Details

Solves the modular equation \( a \times x = b \mod n \). This equation is solvable if and only if \( \gcd(a, n) \mid b \).

The function uses the extended greatest common divisor approach.

Value

Returns a vector of integer solutions.

See Also

extGCD

Examples

\[
\begin{align*}
\text{modlin}(14, 3, 100) & \quad \# 95\ 45 \\
\text{modlin}(3, 4, 5) & \quad \# 3 \\
\text{modlin}(3, 5, 6) & \quad \# [] \\
\text{modlin}(3, 6, 9) & \quad \# 2\ 5\ 8 \\
\end{align*}
\]

modpower

\textit{Power Function modulo } m

Description

Calculates powers and orders modulo \( m \).

Usage

\[
\begin{align*}
\text{modpower}(n, k, m) \\
\text{modorder}(n, m)
\end{align*}
\]

Arguments

\( n, k, m \) Natural numbers, \( m \geq 1 \).

Details

\text{modpower} calculates \( n \) to the power of \( k \) modulo \( m \).

Uses modular exponentiation, as described in the Wikipedia article.

\text{modorder} calculates the order of \( n \) in the multiplicative group modulo \( m \). \( n \) and \( m \) must be coprime.

Uses brute force, trick to use binary expansion and square is not more efficient in an R implementation.
modpower

Value

Natural number.

Note

This function is not vectorized.

See Also

primroot

Examples

```r
modpower(2, 100, 7)  #=> 2
modpower(3, 100, 7)  #=> 4
modorder(7, 17)     #=> 16, i.e. 7 is a primitive root mod 17
```

## Gauss' table of primitive roots modulo prime numbers < 100

```r
proots <- c(2, 2, 3, 2, 2, 6, 5, 10, 10, 10, 2, 2, 10, 17, 5, 5,
            6, 28, 10, 10, 26, 10, 10, 5, 12, 62, 5, 29, 11, 50, 30, 10)
P <- Primes(100)
for (i in seq(along=P)) {
  cat(P[i], "\t", modorder(proots[i], P[i]), proots[i], "\t", "\n")
}
```

## Not run:

## Lehmann's primality test

```r
lehmann_test <- function(n, ntry = 25) {
  if (!is.numeric(n) || ceiling(n) != floor(n) || n < 0)
    stop("Argument 'n' must be a natural number")
  if (n >= 9e7)
    stop("Argument 'n' should be smaller than 9e7.")

  if (n < 2) return(FALSE)
  else if (n == 2) return(TRUE)
  else if (n > 2 && n %% 2 == 0) return(FALSE)

  k <- floor(ntry)
  if (k < 1) k <- 1
  if (k > n-2) a <- 2:(n-1)
  else a <- sample(2:(n-1), k, replace = FALSE)

  for (i in 1:length(a)) {
    m <- modpower(a[i], (n-1)/2, n)
    if (m != 1 && m != n-1) return(FALSE)
  }
  return(TRUE)
}
```

## Examples

```r
for (i in seq(1001, 1011, by = 2))
  if (lehmann_test(i)) cat(i, "\n")
```
moebius

## Moebius Function

### Description

The classical Moebius and Mertens functions in number theory.

### Usage

```r
moebius(n)
mertens(n)
```

### Arguments

- **n**: Positive integer.

### Details

- **moebius(n)** is +1 if `n` is a square-free positive integer with an even number of prime factors, or +1 if there are an odd of prime factors. It is 0 if `n` is not square-free.
- **mertens(n)** is the aggregating summary function, that sums up all values of `moebius` from 1 to `n`.

### Value

- **For moebius**, 0, 1 or -1, depending on the prime decomposition of `n`.
- **For mertens** the values will very slowly grow.

### Note

Works well up to `10^9`, but will become very slow for the Mertens function.

### See Also

- `factorize`, `eulersPhi`
Examples

\[
\begin{align*}
\text{sapply(1:16, moebius)} \\
\text{sapply(1:16, mertens)} \\
\end{align*}
\]

## Not run:
\[
\begin{align*}
x & \leftarrow 1:50; y \leftarrow \text{sapply}(x, \text{moebius}) \\
\text{plot} & (c(1, 50), c(-3, 3), \text{type} = \text{"n"}) \\
\text{grid} & () \\
\text{points} & (1:50, y, \text{pch=18, col} = \text{"blue"}) \\

x & \leftarrow 1:100; y \leftarrow \text{sapply}(x, \text{mertens}) \\
\text{plot} & (c(1, 100), c(-5, 3), \text{type} = \text{"n"}) \\
\text{grid} & () \\
\text{lines} & (1:100, y, \text{col} = \text{"red", type} = \text{"s"}) \\
\end{align*}
\]
## End(Not run)

---

**nextPrime**

**Next Prime**

Description

Find the next prime above \( n \).

Usage

\[
\text{nextPrime}(n)
\]

Arguments

\( n \)  

natural number.

Details

\text{nextPrime} \text{ finds the next prime number greater than } n, \text{ while } \text{previousPrime} \text{ finds the next prime number below } n. \text{ In general the next prime will occur in the interval } [n+1,n+\log(n)] .

\text{In double precision arithmetic integers are represented exactly only up to } 2^{53} - 1, \text{ therefore this is the maximal allowed value.}

Value

Integer.

See Also

\text{Primes, isPrime}

Examples

\[
\begin{align*}
p & \leftarrow \text{nextPrime}(1e+6) \quad \# \text{ 1000003} \\
\text{isPrime(p)} & \quad \# \text{ TRUE}
\end{align*}
\]
**omega**  
*Number of Prime Factors*

**Description**
Sum of all exponents of prime factors in the prime decomposition.

**Usage**

omega(n)  
Omega(n)

**Arguments**

n  
Positive integer.

**Details**

Compute the number of prime factors of \( n \) resp. the sum of their exponents in the prime decomposition.

\((-1)^{\Omega(n)}\) is the Liouville function.

**Value**

Natural number.

**Note**
Works well up to \( 10^9 \).

**See Also**

sigma

**Examples**

omega(2*3*5*7*11*13*17*19) #=> 8  
Omega(2 \times 3^2 \times 5^3 \times 7^4) #=> 10
**previousPrime**

**Previous Prime**

---

**Description**

Find the next prime below n.

**Usage**

```
previousPrime(n)
```

**Arguments**

- **n**: natural number.

**Details**

`previousPrime` finds the next prime number smaller than n, while `nextPrime` finds the next prime number below n. In general the previous prime will occur in the interval \([n-1, n-\log(n)]\).

In double precision arithmetic integers are represented exactly only up to \(2^{53} - 1\), therefore this is the maximal allowed value.

**Value**

Integer.

**See Also**

`Primes`, `isPrime`

**Examples**

```
p <- previousPrime(1e+6) # 999983
isPrime(p) # TRUE
```

---

**Primes**

**Prime Numbers**

---

**Description**

Generate a list of prime numbers less or equal n, resp. between n1 and n2.

**Usage**

```
Primes(n1, n2 = NULL)
```
**Arguments**

\( n_1, n_2 \) natural numbers with \( n_1 \leq n_2 \).

**Details**

The list of prime numbers up to \( n \) is generated using the "sieve of Erasthostenes". This approach is reasonably fast, but may require a lot of main memory when \( n \) is large.

Primes computes first all primes up to \( \sqrt{n_2} \) and then applies a refined sieve on the numbers from \( n_1 \) to \( n_2 \), thereby drastically reducing the need for storing long arrays of numbers.

In double precision arithmetic integers are represented exactly only up to \( 2^{53} - 1 \), therefore this is the maximal allowed value.

**Value**

vector of integers representing prime numbers

**See Also**

`isPrime`, `factorize`

**Examples**

```
Primes(1000)
Primes(1949, 2019)
```

```
## Not run:
## Appendix: Logarithmic Integrals and Prime Numbers (C.F.Gauss, 1846)

library('gsl')
# 'European' form of the logarithmic integral
Li <- function(x) expint_Ei(log(x)) - expint_Ei(log(2))

# No. of primes and logarithmic integral for 10^i, i=1..12
i <- 1:12; N <- 10^i
# piN <- numeric(12)
# for (i in 1:12) piN[i] <- length(primes(10^i))
piN <- c(4, 25, 168, 1229, 9592, 78498, 664579,
        5761455, 50847534, 455052511, 4118054813, 37607912018)
cbind(i, piN, round(Li(N)), round((Li(N)-piN)/piN, 6))
```

```
# i    pi(10^i)    Li(10^i) rel.err
#----------
#  1     4      5 0.280109
#  2    25     29 0.163239
#  3   168    177 0.050979
#  4  1229   1245 0.013094
#  5  9592   9629 0.003833
#  6  78498  78627 0.001637
#  7  664579 664917 0.000509
#  8  5761455 5762208 0.000131
#  9  50847534 50849234 0.000033
```
primroot

## Description
Find the smallest primitive root modulo m.

## Usage
```r
primroot(m)
```

## Arguments
- `m` A prime integer.

## Details
For every prime number \( m \) there exists a natural number \( n \) that generates the field \( F_m \), i.e. \( n, n^2, ..., n^{m-1} \mod(m) \) are all different.

The computation here is all brute force. As most primitive roots are relatively small, it is still reasonable fast.

One trick is to factorize \( m - 1 \) and test only for those prime factors. In R this is not more efficient as factorization also takes some time.

## Value
A natural number if \( m \) is prime, else \( NA \).

## Note
This function is not vectorized.

## References

## See Also
`modpower`, `modorder`
Examples

```r
P <- Primes(100)
R <- c()
for (p in P) {
  R <- c(R, primroot(p))
}
cbind(P, R)  # 7 is the biggest prime root here (for p=71)
```

---

**ratFarey**  
**Farey Approximation**

Description

Rational approximation of real numbers through Farey fractions.

Usage

```r
ratFarey(x, n, upper = TRUE)
```

Arguments

- `x`  
  real number.
- `n`  
  integer, highest allowed denominator in a rational approximation.
- `upper`  
  logical; shall the Farey fraction be greater than `x`.

Details

Rational approximation of real numbers through Farey fractions, i.e. find for `x` the nearest fraction in the Farey series of rational numbers with denominator not larger than `n`.

Value

Returns a vector with two natural numbers, nominator and denominator.

References


See Also

`contFrac`
Examples

ratFarey(pi, 100) # 22/7 0.0013
ratFarey(pi, 100, upper = FALSE) # 311/99 0.0002
ratFarey(-pi, 100) # -22/7
ratFarey(pi - 3, 70) # pi ~= 3 + (3/8)^2
ratFarey(pi, 1000) # 355/113
ratFarey(pi, 10000, upper = FALSE) # id.
ratFarey(pi, 1e5, upper = FALSE) # 312689/99532 - pi ~= 3e-11

ratFarey(4/5, 5) # 4/5
ratFarey(4/5, 4) # 1/1
ratFarey(4/5, 4, upper = FALSE) # 3/4

Description

Integer remainder function.

Usage

rem(n, m)

Arguments

n numeric vector (preferably of integers)
m must be a scalar integer (positive, zero, or negative)

Details

rem(n, m) is the same modulo operator and returns \( n \mod m \). mod(n, 0) is NaN, and the result always has the same sign as n (for n != m and m != 0).

Value

a numeric (integer) value or vector/matrix

See Also

mod, div

Examples

rem(c(-5:5), 5)
rem(c(-5:5), -5)
rem(0, 1) #=> 0
rem(1, 1) #=> 0 (always for n == m)
rem(1, 0) # NA (should be NaN)
rem(0, 0) #=> NaN
Description

Sum of powers of all divisors of a natural number.

Usage

\[
\text{sigma}(n, k = 1, \text{proper} = \text{FALSE})
\]
\[
\text{tau}(n)
\]

Arguments

- **n**: Positive integer.
- **k**: Numeric scalar, the exponent to be used.
- **proper**: Logical; if TRUE, n will not be considered as a divisor of itself; default: FALSE.

Details

Total sum of all integer divisors of \( n \) to the power of \( k \), including 1 and \( n \).

For \( k=0 \) this is the number of divisors, for \( k=1 \) it is the sum of all divisors of \( n \).

tau is Ramanujan’s tau function, here computed using sigma(., 5) and sigma(., 11).

A number is called refactorable, if tau(n) divides n, for example n=12 or n=18.

Value

Natural number, the number or sum of all divisors.

Note

Works well up to \( 10^9 \).

References


See Also

- factorize
twinPrimes

Examples

```r
sapply(1:16, sigma, k = 0)
sapply(1:16, sigma, k = 1)
sapply(1:16, sigma, proper = TRUE)
```

---

### twinPrimes

Twin Primes

#### Description

Generate a list of twin primes between \( n_1 \) and \( n_2 \).

#### Usage

```r
twinPrimes(n1, n2)
```

#### Arguments

- `n1`, `n2` natural numbers with \( n_1 \leq n_2 \).

#### Details

`twinPrimes` uses `Primes` and uses `diff` to find all twin primes in the given interval.

In double precision arithmetic integers are represented exactly only up to \( 2^{53} - 1 \), therefore this is the maximal allowed value.

#### Value

Returns a \( nx2 \)-matrix, where \( n \) is the number of twin primes found, and each twin tuple fills one row.

#### See Also

- `Primes`

#### Examples

```r
twinPrimes(1e6+1, 1e6+1001)
```
Description

Generates the Zeckendorf representation of an integer as a sum of Fibonacci numbers.

Usage

zeck(n)

Arguments

n integer.

Details

According to Zeckendorf's theorem from 1972, each integer can be uniquely represented as a sum of Fibonacci numbers such that no two of these are consecutive in the Fibonacci sequence.

The computation is simply the greedy algorithm of finding the highest Fibonacci number below n, subtracting it and iterating.

Value

List with components fibs the Fibonacci numbers that add up to n, and inds their indices in the Fibonacci sequence.

Examples

zeck(10)  #=> 2 + 8 = 10
zeck(100) #=> 3 + 8 + 89 = 100
zeck(1000) #=> 13 + 987 = 1000
Index

agm, 3
chinese, 17
chinese (chinese remainder theorem), 5
chinese remainder theorem, 5
contFrac, 6
coprime, 7, 15
div, 7, 20, 31
droplet_e, 8
egyptian_complete, 9, 10
egyptian_methods, 9, 10
eulersPhi, 11, 24
extGCD, 5, 12, 15, 21, 22
factorize, 11, 13, 19, 24, 28, 32
fibonacci, 13
GCD, 7, 12
GCD (GCD, LCM), 15
GCD, LCM, 15
Hermite normal form, 16
HermiteNF (Hermite normal form), 16
isIntpower, 17
isNatural, 18
isPrime, 13, 19, 25, 27, 28
LCM (GCD, LCM), 15
lucas (fibonacci), 13
mertens (moebius), 24
mGCD (GCD, LCM), 15
mLCM (GCD, LCM), 15
mod, 8, 20, 31
modinv, 21
modlin, 21
modorder, 29
modorder (modpower), 22
modpower, 22, 29
moebius, 24
nextPrime, 25
numbers-package, 2
Omega (omega), 26
omega, 26
previousPrime, 27
Primes, 13, 19, 25, 27, 27, 33
primroot, 23, 29
ratFarey, 6, 30
rem, 8, 20, 31
sigma, 11, 26, 32
tau (sigma), 32
twinPrimes, 33
zech, 34

35