

# Package ‘MiscMath’

April 13, 2025

**Type** Package

**Title** Miscellaneous Mathematical Tools

**Version** 1.1

**Description** Some basic math calculators for finding angles for triangles and for finding the greatest common divisor of two numbers and so on.

**LazyLoad** true

**License** GPL (>= 2)

**Depends** randomForest, numbers

**NeedsCompilation** yes

**Author** W.J. Braun [aut, cre]

**Maintainer** W.J. Braun <john.braun@ubc.ca>

**Repository** CRAN

**Date/Publication** 2025-04-13 05:30:01 UTC

## Contents

MiscMath-package . . . . .	2
DecToBin . . . . .	2
EratosthenesSieve . . . . .	3
gcd . . . . .	3
IntDecToBin . . . . .	4
LawofCosines . . . . .	5
LawofSines . . . . .	5
Matpower . . . . .	6
MaxRunLength . . . . .	7
mCracker . . . . .	8
microLASSO . . . . .	9
rAlias . . . . .	9
rlincong . . . . .	10
rMWC . . . . .	11
RNGtest . . . . .	11
rngV . . . . .	12
shuffle . . . . .	13

**Index****14**

---

MiscMath-package      *Miscellaneous Mathematical Tools and Facts*

---

**Description**

A random assortment of elementary mathematical formulas and calculators.

**Examples**

```
# Find the greatest common divisor of 57, 93 and 117.
gcd(c(57, 93, 117))
```

---

DecToBin      *Convert Decimal to Binary*

---

**Description**

Convert a given decimal constant in the interval (0, 1) to the corresponding binary representation.

**Usage**

```
DecToBin(x, m = 32, format = "character")
```

**Arguments**

x	a numeric vector of values in the interval (0, 1)
m	a numeric constant specifying the number of binary digits to use in the output
format	a character string constant specifying the form of the output

**Details**

Default format is as a character string. Alternatives are plain which prints results to the device, and vector which output a binary vector.

**Value**

a vector containing the binary representation

**Examples**

```
x <- c(.81, .57, .333)
DecToBin(x) # character output
DecToBin(x, format="vector") # binary vector output
DecToBin(x, format="plain")
```

---

EratosthenesSieve	<i>Sieve of Eratosthenes</i>
-------------------	------------------------------

---

**Description**

The sieve of Eratosthenes is an ancient method for listing all prime numbers up to a given value  $n$ .

**Usage**

EratosthenesSieve( $n$ )

**Arguments**

$n$  a numeric vector consisting of a single positive integer.

**Details**

The algorithm scans through the vector from 2 through  $n$ , eliminating all multiples of 2, then eliminating all multiples of the next smallest integer (3), and so on, until only the prime numbers less than  $n$  remain.

**Value**

a numeric vector containing all primes less than  $n$ .

**Examples**

EratosthenesSieve(100)

---

gcd	<i>Greatest Common Divisor</i>
-----	--------------------------------

---

**Description**

Greatest common divisor or factor for all elements of a positive-integer-valued vector.

**Usage**

gcd( $x$ )

**Arguments**

$x$  a numeric vector consisting of at least two positive integer values.

**Details**

The gcd is calculated using the Euclidean algorithm applied to successive pairs of the elements of `x`.

**Value**

a numeric constant containing the greatest common divisor.

**Examples**

```
x <- c(81, 57, 333)
gcd(x)
```

---

`IntDecToBin`*Binary Expansion of Positive Integers*

---

**Description**

Convert positive integers to their corresponding binary representation.

**Usage**

```
IntDecToBin(x, m = 31)
```

**Arguments**

<code>x</code>	an integer vector
<code>m</code>	a numeric constant specifying the number of binary digits to use in the output

**Value**

a matrix containing the binary representations

**Examples**

```
x <- c(81, 57, 333)
IntDecToBin(as.integer(x))
```

---

LawofCosines	<i>Law of Cosines</i>
--------------	-----------------------

---

**Description**

Use of the ancient law of cosines to determine the angle between two sides of a triangle, given lengths of all three sides. This is a generalization of Pythagoras' Theorem.

**Usage**

```
LawofCosines(sides)
```

**Arguments**

sides            a numeric vector of length 3, containing the side lengths.

**Value**

a numeric constant giving the angle in between the sides corresponding to the first two components in sides. Result is expressed in degrees.

**Examples**

```
LawofCosines(c(3, 4, 5))
```

---

LawofSines	<i>Law of Sines</i>
------------	---------------------

---

**Description**

Use of the ancient law of sines to determine the angle opposite one side of a triangle, given the length of the opposite side as well as the angle opposite another side whose length is also known. Alternatively, one can find the length of the side opposite a given angle.

**Usage**

```
LawofSines(sides, angles, findAngle)
```

**Arguments**

sides            a numeric vector of length 1 or 2, containing the side lengths.  
angles           a numeric vector of length 1 or 2, containing the angles (in degrees).  
findAngle        a logical constant

**Details**

If `findAngle` is `TRUE`, `sides` should be of length 2 and the function will compute angle opposite the side with length given by the second element of `sides`. Otherwise, `angles` should be of length 2, and the function will calculate the length of the side opposite the angle corresponding to the second element of `angles`.

**Value**

a numeric constant giving the angle opposite the given side, if `findAngle` is `TRUE`, or giving the length of the side opposite the given angle, if `findAngle` is `FALSE`.

**Examples**

```
LawofSines(c(3, 4), 50) # find angle opposite the side of length 4.  
LawofSines(3, c(70, 50), findAngle = FALSE) # find length of side opposite the 50 degree angle
```

---

Matpower

*Matrix Power*

---

**Description**

Implementation of efficient algorithm to compute the  $p$ th power of a matrix.

**Usage**

```
Matpower(X, p)
```

**Arguments**

<code>X</code>	numeric square matrix
<code>p</code>	integer: nonnegative exponent.

**Value**

a matrix containing the  $p$ th power of `X`.

**References**

Golub and Van Loan (1983) *Matrix Computations*. Algorithm 11.2-1. p. 393.

---

MaxRunLength	<i>Maximum Run Length</i>
--------------	---------------------------

---

**Description**

Calculate the maximum run length of 0's in a binary vector.

**Usage**

```
MaxRunLength(x)
```

**Arguments**

x                    a binary vector

**Value**

the maximum run length of 0's

**Examples**

```
x <- c(0L, 1L, 1L, 1L, 0L, 0L, 1L, 1L, 0L, 0L, 1L)
MaxRunLength(x) # should be 2
MaxRunLength(1L - x) # should be 3
# Test of Mersenne Twister

RNGkind("Mers") # ensure that default generator is in use
N <- 10000
x <- runif(N)
y <- DecToBin(x, format = "vector", m = 40)
MaxHeadRunsM <- apply(y, 1, MaxRunLength) # 0 run lengths (Heads)
MaxTailRunsM <- apply(1-y, 1, MaxRunLength) # 1 run lengths (Tails)
# distributions of Max 0 run lengths and Max 1 run lengths should match
boxplot(MaxHeadRunsM, MaxTailRunsM, axes=FALSE, main="Maximum Run Length")
axis(side=1, at=c(1, 2), label=c("Heads", "Tails"))
axis(2)
box()

# Comparison with Wichmann-Hill Generator

RNGkind("Wich")
x <- runif(N)
y <- DecToBin(x, format = "vector", m = 40)
MaxHeadRunsW <- apply(y, 1, MaxRunLength)
MaxTailRunsW <- apply(1-y, 1, MaxRunLength)
boxplot(MaxHeadRunsW, MaxTailRunsW, axes=FALSE, main="Maximum Run Length")
axis(side=1, at=c(1, 2), label=c("Heads", "Tails"))
axis(2)
box()
RNGkind("Mers") # restore default generator
```

---

mCracker

*Modulus Cracker*


---

### Description

Infers the modulus  $m$  for a congruential random number generator.

### Usage

```
mCracker(U, par = 1e6, maxit = 100)
```

### Arguments

U	a numeric vector consisting of $n$ (say 10000) uniform(0,1) pseudorandom numbers of the form $x_1/m, x_2/m, \dots, x_n/m$ .
par	an integer guess as to an upper bound on the smallest integer in the sequence $x_1, x_2, \dots, x_n$ .
maxit	maximum number of iterations allowed.

### Details

Basic idea: Let  $x(1)$  denote the minimum order statistic in  $x_1, x_2, \dots, x_n$ . Then the set  $(x_1/m)/(x(1)/m) \cdot (1:par)$  must contain at least one integer, and  $m$  is in that set, if  $par$  has been set correctly.

### Value

a list consisting of

m	the integer value of $m$
firstInteger	the minimum order statistic of the set $x_1, x_2, \dots, x_n$

### Examples

```
# set.seed(33663)
x <- runif(1000000)
Y <- mCracker(x)$m
log(Y, 2) # should be 32
```



---

`microLASSO`*Simplest Case of LASSO Regression*

---

**Description**

Simple linear regression estimators for slope, intercept and noise standard deviation with absolute value penalty on slope.

**Usage**

```
microLASSO(x, y, lambda)
```

**Arguments**

<code>x</code>	a numeric vector of covariate values
<code>y</code>	a numeric vector of response values
<code>lambda</code>	a numeric constant which should be nonnegative

**Value**

a list consisting of

<code>Coefficients</code>	a numeric vector containing intercept and slope estimates
<code>RMSE</code>	a numeric constant containing the (penalized) maximum likelihood estimate of the noise standard deviation

**Examples**

```
x <- runif(30)
y <- x + rnorm(30)
microLASSO(x, y, lambda = 0.5)
```

---

`rAlias`*Alias Method for Generating Discrete Random Variates*

---

**Description**

Efficient method for generating discrete random variates from any distribution with a finite number (N) of states. The method is described in detail in Section 10.1 of the given reference.

**Usage**

```
rAlias(n, P)
```

**Arguments**

n                    numeric, constant number of variates to be simulated  
P                    numeric, probability mass function, assuming states from 1 through N

**Value**

numeric vector of containing n simulated values from the given discrete distribution

**References**

S. Ross (1990) A Course in Simulation, MacMillan.

**Examples**

```
x <- rAlias(n = 100, P = c(1:5)/15)
table(x)/100
```

---

rlincong

*Linear Congruential Generator*

---

**Description**

Basic implementation of the linear congruential random number generator.

**Usage**

```
rlincong(n, seed, par)
```

**Arguments**

n                    numeric: number of variates to generate.  
seed                numeric: initial seed.  
par                 an integer vector containing parameters for the generator: a, cc, m.

**Value**

a vector of n uniform random numbers

**Examples**

```
x <- rlincong(1000, 6976, c(171, 0, 30269))
summary(x)
```

---

`rMWC`*Multiply-With-Carry Random Number Generator*

---

**Description**

Basic implementation of the multiply-with-carry generator.

**Usage**

```
rMWC(n, par)
```

**Arguments**

`n` numeric: number of variates to generate.  
`par` an integer vector containing parameters for the generator: X, C, A, B.

**Value**

a vector of `n` uniform random numbers

**References**

Marsaglia, G. (2003) Random Number Generators. *Journal of Modern Applied Statistical Methods*. 2(1):2.

**Examples**

```
x <- rMWC(58, c(5, 3, 6, 10))  
summary(x)
```

---

`RNGtest`*Pseudorandom Number Testing via Random Forest*

---

**Description**

Given a sequence of pseudorandom numbers, this function constructs a random forest prediction model for successive values, based on previous values up to a given lag. The ability of the random forest model to predict future values is inversely related to the quality of the sequence as an approximation to locally random numbers.

**Usage**

```
RNGtest(u, m=5)
```

**Arguments**

u                    numeric, a vector of pseudorandom numbers to test  
 m                    numeric, number of lags to test

**Value**

Side effect is a two way layout of graphs showing effectiveness of prediction on a training and a testing subset of data. Good predictions indicate a poor quality sequence.

**Author(s)**

W. John Braun

**Examples**

```
x <- runif(200)
RNGtest(x, m = 4)
```

---

 rngV

---

*Vectorized Congruential Random Number Generator*


---

**Description**

Basic vectorized implementation of the linear congruential generator for simulating uniform random numbers on the interval (0, 1).

**Usage**

```
rngV(n, seed, par)
```

**Arguments**

n                    numeric: number of variates to generate.  
 seed                numeric: initial seed.  
 par                 an integer vector containing parameters for the generator: a, cc, m, L.

**Value**

a vector of n uniform random numbers

**References**

Anderson, S.L. (1990) Random number generators on vector supercomputers and other advanced architectures. *SIAM Review*, 32(2):221-251.

**Examples**

```
x <- rngV(1000, 6976, c(171, 0, 30269, 10))
summary(x)
```

---

`shuffle`*Shuffling Algorithm*

---

**Description**

Implementation of a simple shuffling algorithm that can be used to randomly permute a given set of simulated random numbers.

**Usage**

```
shuffle(n, k = 100, x = runif(n))
```

**Arguments**

<code>n</code>	numeric: number of variates to be output.
<code>k</code>	numeric: a tuning parameter for the shuffler.
<code>x</code>	a vector containing a sequence to be randomly permuted with the shuffler.

**Value**

a numeric vector

# Index

- \* **graphics**
    - rAlias, 9
    - RNGtest, 11
  - \* **math**
    - DecToBin, 2
    - EratosthenesSieve, 3
    - gcd, 3
    - IntDecToBin, 4
    - LawofCosines, 5
    - LawofSines, 5
    - Matpower, 6
    - MaxRunLength, 7
    - mCracker, 8
    - rIncong, 10
    - rMWC, 11
    - rngV, 12
    - shuffle, 13
  - \* **models**
    - microLASSO, 9
  - \* **package**
    - MiscMath-package, 2
- DecToBin, 2
- EratosthenesSieve, 3
- gcd, 3
- IntDecToBin, 4
- LawofCosines, 5
- LawofSines, 5
- Matpower, 6
- MaxRunLength, 7
- mCracker, 8
- microLASSO, 9
- MiscMath (MiscMath-package), 2
- MiscMath-package, 2
- rAlias, 9
- rIncong, 10
- rMWC, 11
- RNGtest, 11
- rngV, 12
- shuffle, 13