## NAME

Math::BigRat - Arbitrary big rational numbers

## SYNOPSIS

```
use Math::BigRat;

my $x = Math::BigRat->new('3/7'); $x += '5/9';

print $x->bstr(),"\n";
print $x ** 2,"\n";

my $y = Math::BigRat->new('inf');
print "$y ", ($y->is_inf ? 'is' : 'is not') , " infinity\n";

my $z = Math::BigRat->new(144); $z->bsqrt();
```

## DESCRIPTION

Math::BigRat complements Math::BigInt and Math::BigFloat by providing support for arbitrary big rational numbers.

### MATH LIBRARY

You can change the underlying module that does the low-level math operations by using:

```
use Math::BigRat try => 'GMP';
```

Note: This needs Math::BigInt::GMP installed.

The following would first try to find Math::BigInt::Foo, then Math::BigInt::Bar, and when this also fails, revert to Math::BigInt::Calc:

```
use Math::BigRat try => 'Foo,Math::BigInt::Bar';
```

If you want to get warned when the fallback occurs, replace "try" with "lib":

```
use Math::BigRat lib => 'Foo,Math::BigInt::Bar';
```

If you want the code to die instead, replace "try" with "only":

```
use Math::BigRat only => 'Foo,Math::BigInt::Bar';
```

## METHODS

Any methods not listed here are derived from Math::BigFloat (or Math::BigInt), so make sure you check these two modules for further information.

### new()

```
$x = Math::BigRat->new('1/3');
```

Create a new Math::BigRat object. Input can come in various forms:

```
$x = Math::BigRat->new(123);    # scalars
$x = Math::BigRat->new('inf');    # infinity
$x = Math::BigRat->new('123.3');   # float
$x = Math::BigRat->new('1/3');    # simple string
$x = Math::BigRat->new('1 / 3');   # spaced
```

```
$x = Math::BigRat->new('1 / 0.1');   # w/ floats
$x = Math::BigRat->new(Math::BigInt->new(3));  # BigInt
$x = Math::BigRat->new(Math::BigFloat->new('3.1')); # BigFloat
$x = Math::BigRat->new(Math::BigInt::Lite->new('2')); # BigLite

# You can also give D and N as different objects:
$x = Math::BigRat->new(
 Math::BigInt->new(-123),
 Math::BigInt->new(7),
 );   # => -123/7
```

### numerator()

```
$n = $x->numerator();
```

Returns a copy of the numerator (the part above the line) as signed BigInt.

### denominator()

```
$d = $x->denominator();
```

Returns a copy of the denominator (the part under the line) as positive BigInt.

### parts()

```
($n,$d) = $x->parts();
```

Return a list consisting of (signed) numerator and (unsigned) denominator as BigInts.

### numify()

```
my $y = $x->numify();
```

Returns the object as a scalar. This will lose some data if the object cannot be represented by a normal Perl scalar (integer or float), so use *as_int()* or *as_float()* instead.

This routine is automatically used whenever a scalar is required:

```
my $x = Math::BigRat->new('3/1');
@array = (0,1,2,3);
$y = $array[$x];  # set $y to 3
```

### as_int()/as_number()

```
$x = Math::BigRat->new('13/7');
print $x->as_int(),"\n";  # '1'
```

Returns a copy of the object as BigInt, truncated to an integer.

`as_number()` is an alias for `as_int()`.

### as_float()

```
$x = Math::BigRat->new('13/7');
print $x->as_float(),"\n";  # '1'

$x = Math::BigRat->new('2/3');
print $x->as_float(5),"\n";  # '0.66667'
```

Returns a copy of the object as BigFloat, preserving the accuracy as wanted, or the default of 40 digits.

This method was added in v0.22 of Math::BigRat (April 2008).

### as_hex()

```
$x = Math::BigRat->new('13');
print $x->as_hex(),"\n";  # '0xd'
```

Returns the BigRat as hexadecimal string. Works only for integers.

### as_bin()

```
$x = Math::BigRat->new('13');
print $x->as_bin(),"\n";  # '0x1101'
```

Returns the BigRat as binary string. Works only for integers.

### as_oct()

```
$x = Math::BigRat->new('13');
print $x->as_oct(),"\n";  # '015'
```

Returns the BigRat as octal string. Works only for integers.

### from_hex()/from_bin()/from_oct()

```
my $h = Math::BigRat->from_hex('0x10');
my $b = Math::BigRat->from_bin('0b10000000');
my $o = Math::BigRat->from_oct('020');
```

Create a BigRat from an hexadecimal, binary or octal number in string form.

### length()

```
$len = $x->length();
```

Return the length of $x in digits for integer values.

### digit()

```
print Math::BigRat->new('123/1')->digit(1); # 1
print Math::BigRat->new('123/1')->digit(-1); # 3
```

Return the N'ths digit from X when X is an integer value.

### bnorm()

```
$x->bnorm();
```

Reduce the number to the shortest form. This routine is called automatically whenever it is needed.

### bfac()

```
$x->bfac();
```

Calculates the factorial of $x. For instance:

```
print Math::BigRat->new('3/1')->bfac(),"\n"; # 1*2*3
print Math::BigRat->new('5/1')->bfac(),"\n"; # 1*2*3*4*5
```

Works currently only for integers.

### bround()/round()/bfround()

Are not yet implemented.

### bmod()

```
use Math::BigRat;
my $x = Math::BigRat->new('7/4');
my $y = Math::BigRat->new('4/3');
print $x->bmod($y);
```

Set $x to the remainder of the division of $x by $y.

### bneg()

```
$x->bneg();
```

Used to negate the object in-place.

### is_one()

```
print "$x is 1\n" if $x->is_one();
```

Return true if $x is exactly one, otherwise false.

### is_zero()

```
print "$x is 0\n" if $x->is_zero();
```

Return true if $x is exactly zero, otherwise false.

### is_pos()/is_positive()

```
print "$x is >= 0\n" if $x->is_positive();
```

Return true if $x is positive (greater than or equal to zero), otherwise false. Please note that '+inf' is also positive, while 'NaN' and '-inf' aren't.

`is_positive()` is an alias for `is_pos()`.

### is_neg()/is_negative()

```
print "$x is < 0\n" if $x->is_negative();
```

Return true if $x is negative (smaller than zero), otherwise false. Please note that '-inf' is also negative, while 'NaN' and '+inf' aren't.

`is_negative()` is an alias for `is_neg()`.

### is_int()

```
print "$x is an integer\n" if $x->is_int();
```

Return true if $x has a denominator of 1 (e.g. no fraction parts), otherwise false. Please note that '-inf', 'inf' and 'NaN' aren't integer.

### is_odd()

```
print "$x is odd\n" if $x->is_odd();
```

Return true if $x is odd, otherwise false.

### is_even()

```
print "$x is even\n" if $x->is_even();
```

Return true if $x is even, otherwise false.

### bceil()

```
$x->bceil();
```

Set $x to the next bigger integer value (e.g. truncate the number to integer and then increment it by one).

### bfloor()

```
$x->bfloor();
```

Truncate $x to an integer value.

### bsqrt()

```
$x->bsqrt();
```

Calculate the square root of $x.

### broot()

```
$x->broot($n);
```

Calculate the N'th root of $x.

### badd()/bmul()/bsub()/bdiv()/bdec()/binc()

Please see the documentation in *Math::BigInt*.

### copy()

```
my $z = $x->copy();
```

Makes a deep copy of the object.

Please see the documentation in *Math::BigInt* for further details.

### bstr()/bsstr()

```
my $x = Math::BigInt->new('8/4');
print $x->bstr(),"\n";   # prints 1/2
print $x->bsstr(),"\n";   # prints 1/2
```

Return a string representing this object.

### bacmp()/bcmp()

Used to compare numbers.

Please see the documentation in *Math::BigInt* for further details.

### blsft()/brsft()

Used to shift numbers left/right.

Please see the documentation in *Math::BigInt* for further details.

### bpow()

```
$x->bpow($y);
```

Compute $x ** $y.

Please see the documentation in *Math::BigInt* for further details.

### bexp()

```
$x->bexp($accuracy);  # calculate e ** X
```

Calculates two integers A and B so that A/B is equal to `e ** $x`, where e is Euler's number.

This method was added in v0.20 of Math::BigRat (May 2007).

See also *blog()*.

### bnok()

```
$x->bnok($y);     # x over y (binomial coefficient n over k)
```

Calculates the binomial coefficient n over k, also called the "choose" function. The result is equivalent to:

```
( n )      n!
| - |  = -------
( k )    k!(n-k)!
```

This method was added in v0.20 of Math::BigRat (May 2007).

### config()

```
use Data::Dumper;

print Dumper ( Math::BigRat->config() );
print Math::BigRat->config()->{lib},"\n";
```

Returns a hash containing the configuration, e.g. the version number, lib loaded etc. The following hash keys are currently filled in with the appropriate information.

```
key             RO/RW   Description
                        Example
============================================================
lib             RO      Name of the Math library
                        Math::BigInt::Calc
lib_version     RO      Version of 'lib'
                        0.30
class           RO      The class of config you just called
                        Math::BigRat
version         RO      version number of the class you used
                        0.10
upgrade         RW      To which class numbers are upgraded
                        undef
downgrade       RW      To which class numbers are downgraded
                        undef
precision       RW      Global precision
                        undef
accuracy        RW      Global accuracy
```

```
                                undef
round_mode      RW              Global round mode
                                even
div_scale       RW              Fallback accuracy for div
                                40
trap_nan        RW              Trap creation of NaN (undef = no)
                                undef
trap_inf        RW              Trap creation of +inf/-inf (undef = no)
                                undef
```

By passing a reference to a hash you may set the configuration values. This works only for values that a marked with a `RW` above, anything else is read-only.

### objectify()

This is an internal routine that turns scalars into objects.

## BUGS

Some things are not yet implemented, or only implemented half-way:

inf handling (partial)

NaN handling (partial)

rounding (not implemented except for bceil/bfloor)

$x ** $y where $y is not an integer

bmod(), blog(), bmodinv() and bmodpow() (partial)

## LICENSE

This program is free software; you may redistribute it and/or modify it under the same terms as Perl itself.

## SEE ALSO

*Math::BigFloat* and *Math::Big* as well as *Math::BigInt::Pari* and *Math::BigInt::GMP*.

See *http://search.cpan.org/search?dist=bignum* for a way to use Math::BigRat.

The package at *http://search.cpan.org/search?dist=Math%3A%3ABigRat* may contain more documentation and examples as well as testcases.

## AUTHORS

(C) by Tels *http://bloodgate.com/* 2001 - 2009.

Currently maintained by Jonathan "Duke" Leto <jonathan@leto.net> *http://leto.net*