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

Math with the numbers is done (by default) by a module called Math::BigInt::Calc. This is equivalent to saying:

```
use Math::BigRat lib => 'Calc';
```

You can change this by using:

```
use Math::BigRat lib => 'BitVect';
```

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

Calc.pm uses as internal format an array of elements of some decimal base (usually 1e7, but this might be different for some systems) with the least significant digit first, while BitVect.pm uses a bit vector of base 2, most significant bit first. Other modules might use even different means of representing the numbers. See the respective module documentation for further details.

Currently the following replacement libraries exist, search for them at CPAN:

```
Math::BigInt::BitVect
Math::BigInt::GMP
Math::BigInt::Pari
Math::BigInt::FastCalc
```

METHODS

Any methods not listed here are dervied 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.

as_int()

```
$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_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.

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.

blog()

Is not yet implemented.

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.

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()

```
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()

```
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.

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 acccuracy 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.

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::BitVect*, *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 - 2005.