NAME

bigrat - Transparent BigNumber/BigRational support for Perl

SYNOPSIS

```
use bigrat;

print 2 + 4.5,"\n"; # BigFloat 6.5
print 1/3 + 1/4,"\n"; # produces 7/12

{
no bigrat;
print 1/3,"\n"; # 0.33333...
}

# Import into current package:
use bigrat qw/hex oct/;
print hex("0x1234567890123490"),"\n";
print oct("01234567890123490"),"\n";
```

DESCRIPTION

All operators (including basic math operations) are overloaded. Integer and floating-point constants are created as proper BigInts or BigFloats, respectively.

Other than bignum, this module upgrades to Math::BigRat, meaning that instead of 2.5 you will get 2+1/2 as output.

Modules Used

`bigrat` is just a thin wrapper around various modules of the Math::BigInt family. Think of it as the head of the family, who runs the shop, and orders the others to do the work.

The following modules are currently used by bignum:

```
Math::BigInt::Lite (for speed, and only if it is loadable)
Math::BigInt
Math::BigFloat
Math::BigRat
```

Math Library

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

```
use bigrat lib => 'Calc';
```

You can change this by using:

```
use bignum lib => 'GMP';
```

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

Using `lib` warns if none of the specified libraries can be found and Math::BigInt did fall back to one of the default libraries. To suppress this warning, use `try` instead:

```
use bignum try => 'GMP';
```

If you want the code to die instead of falling back, use `only` instead:

```
use bignum only => 'GMP';
```

Please see respective module documentation for further details.

Sign

The sign is either '+', '-', 'NaN', '+inf' or '-inf'.

A sign of 'NaN' is used to represent the result when input arguments are not numbers or as a result of 0/0. '+inf' and '-inf' represent plus respectively minus infinity. You will get '+inf' when dividing a positive number by 0, and '-inf' when dividing any negative number by 0.

Methods

Since all numbers are not objects, you can use all functions that are part of the BigInt or BigFloat API. It is wise to use only the *bxxx()* notation, and not the *fxxx()* notation, though. This makes you independent on the fact that the underlying object might morph into a different class than BigFloat.

inf()

A shortcut to return Math::BigInt->*binf()*. Useful because Perl does not always handle bareword `inf` properly.

NaN()

A shortcut to return Math::BigInt->*bnan()*. Useful because Perl does not always handle bareword `NaN` properly.

e

```
# perl -Mbigrat=e -wle 'print e'
```

Returns Euler's number `e`, aka *exp(1)*.

PI

```
# perl -Mbigrat=PI -wle 'print PI'
```

Returns PI.

bexp()

```
bexp($power,$accuracy);
```

Returns Euler's number `e` raised to the appropriate power, to the wanted accuracy.

Example:

```
# perl -Mbigrat=bexp -wle 'print bexp(1,80)'
```

bpi()

```
bpi($accuracy);
```

Returns PI to the wanted accuracy.

Example:

```
# perl -Mbigrat=bpi -wle 'print bpi(80)'
```

upgrade()

Return the class that numbers are upgraded to, is in fact returning `$Math::BigInt::upgrade`.

in_effect()

```
use bigrat;

print "in effect\n" if bigrat::in_effect; # true
{
no bigrat;
print "in effect\n" if bigrat::in_effect; # false
}
```

Returns true or false if `bigrat` is in effect in the current scope.

This method only works on Perl v5.9.4 or later.

MATH LIBRARY

Math with the numbers is done (by default) by a module called

Caveat

But a warning is in order. When using the following to make a copy of a number, only a shallow copy will be made.

```
$x = 9; $y = $x;
$x = $y = 7;
```

If you want to make a real copy, use the following:

```
$y = $x->copy();
```

Using the copy or the original with overloaded math is okay, e.g. the following work:

```
$x = 9; $y = $x;
print $x + 1, " ", $y,"\n"; # prints 10 9
```

but calling any method that modifies the number directly will result in **both** the original and the copy being destroyed:

```
$x = 9; $y = $x;
print $x->badd(1), " ", $y,"\n"; # prints 10 10

$x = 9; $y = $x;
print $x->binc(1), " ", $y,"\n"; # prints 10 10

$x = 9; $y = $x;
print $x->bmul(2), " ", $y,"\n"; # prints 18 18
```

Using methods that do not modify, but testthe contents works:

```
$x = 9; $y = $x;
$z = 9 if $x->is_zero(); # works fine
```

See the documentation about the copy constructor and `=` in overload, as well as the documentation in BigInt for further details.

Options

bignum recognizes some options that can be passed while loading it via use. The options can (currently) be either a single letter form, or the long form. The following options exist:

a or accuracy

This sets the accuracy for all math operations. The argument must be greater than or equal to zero. See Math::BigInt's *bround()* function for details.

```
perl -Mbigrat=a,50 -le 'print sqrt(20)'
```

Note that setting precision and accuracy at the same time is not possible.

p or precision

This sets the precision for all math operations. The argument can be any integer. Negative values mean a fixed number of digits after the dot, while a positive value rounds to this digit left from the dot. 0 or 1 mean round to integer. See Math::BigInt's *bfround()* function for details.

```
perl -Mbigrat=p,-50 -le 'print sqrt(20)'
```

Note that setting precision and accuracy at the same time is not possible.

t or trace

This enables a trace mode and is primarily for debugging bignum or Math::BigInt/Math::BigFloat.

l or lib
Load a different math lib, see ‘‘MATH LIBRARY’’.

```
perl -Mbigrat=l,GMP -e 'print 2 ** 512'
```

Currently there is no way to specify more than one library on the command line. This means the following does not work:

```
perl -Mbignum=l,GMP,Pari -e 'print 2 ** 512'
```

This will be hopefully fixed soon ;)

hex
Override the built-in *hex()* method with a version that can handle big numbers. This overrides it by exporting it to the current package. Under Perl v5.10.0 and higher, this is not so necessary, as *hex()* is lexically overridden in the current scope whenever the bigrat pragma is active.

oct
Override the built-in *oct()* method with a version that can handle big numbers. This overrides it by exporting it to the current package. Under Perl v5.10.0 and higher, this is not so necessary, as *oct()* is lexically overridden in the current scope whenever the bigrat pragma is active.

v or version
This prints out the name and version of all modules used and then exits.

```
perl -Mbigrat=v
```

CAVEATS

in_effect()
This method only works on Perl v5.9.4 or later.

hex()/oct()
`bigint` overrides these routines with versions that can also handle big integer values. Under Perl prior to version v5.9.4, however, this will not happen unless you specifically ask for it with the two import tags ‘‘hex’’ and ‘‘oct’’ - and then it will be global and cannot be disabled inside a scope with ‘‘no bigint’’:

```
use bigint qw/hex oct/;

print hex("0x1234567890123456");
{
no bigint;
print hex("0x1234567890123456");
}
```

The second call to *hex()* will warn about a non-portable constant.

Compare this to:

```
use bigint;

# will warn only under Perl older than v5.9.4
print hex("0x1234567890123456");
```

EXAMPLES

```
perl -Mbigrat -le 'print sqrt(33)'
perl -Mbigrat -le 'print 2*255'
perl -Mbigrat -le 'print 4.5+2*255'
perl -Mbigrat -le 'print 3/7 + 5/7 + 8/3'
perl -Mbigrat -le 'print 12->is_odd()';
perl -Mbignum=l,GMP -le 'print 7 ** 7777'
```

LICENSE

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

SEE ALSO

Especially bignum.

Math::BigFloat, Math::BigInt, Math::BigRat and Math::Big as well as Math::BigInt::Pari and Math::BigInt::GMP.

AUTHORS

(C) by Tels <http://bloodgate.com/> in early 2002 - 2007.