

GNU MPC

Ruby bindings to the GNU MPC library

Edition 0.1.1

9 October 2012

written by Sam Rawlins

with extensive quoting from the GNU MPC manual

This manual describes how to use the `gnu_mpc` Ruby gem, which provides bindings to GNU MPC, “a C library for the arithmetic of complex numbers with arbitrarily high precision and correct rounding of the result.”

Copyright 2012 Sam Rawlins

Apache 2.0 License <http://www.apache.org/licenses/LICENSE-2.0>

Contents

GNU MPC	1
Introduction to GNU MPC	4
Introduction to the gnu_mpc gem	5
Installing the gnu_mpc gem	5
Prerequisites	5
Installing	5
Testing the gnu_mpc gem	6
MPC and mpc gem basics	6
Classes	6
Standard Options Hash Keys	6
Precision of Returned Values	6
Complex Functions	7
Initializing, Assigning Complex Numbers	7
Converting Complex Numbers	8
String and Stream Input and Output	8
Complex Comparison	8
Projection and Decomposing	8
Basic Arithmetic	9
Power Functions and Logarithm	10
Trigonometric Functions	11
Miscellaneous Complex Functions	12
Advanced Functions	12
Internals	12

Introduction to GNU MPC

GNU MPC is a C library for the arithmetic of complex numbers with arbitrarily high precision and correct rounding of the result. It extends the principles of the IEEE-754 standard for fixed precision real floating point numbers to complex numbers, providing well-defined semantics for every operation. At the same time, speed of operation at high precision is a major design goal.

The library is built upon and follows the same principles as Gnu Mpfr. It is written by Andreas Enge, Mickaël Gastineau, Philippe Théveny and Paul Zimmermann and is distributed under the Gnu Lesser General Public License, either version 3 of the licence, or (at your option) any later version. The Gnu Mpc library has been registered in France by the Agence pour la Protection des Programmes on 2003-02-05 under the number IDDN FR 001 060029 000 R P 2003 000 10000.

Introduction to the gnu_mpc gem

The gnu_gmp Ruby gem is a Ruby library that provides bindings to GNU MPC. The gem is incomplete, and will likely only include a subset of the GMP functions. It is built as a C extension for Ruby, interacting with mpc.h. The gnu_mpc gem is not endorsed or supported by GNU or the MPC team. The gnu_mpc gem also does not ship with MPC; that must be compiled separately.

Installing the gnu_mpc gem

Prerequisites

OK. First, we've got a few requirements. To install the gnu_mpc gem, you need one of the following versions of Ruby:

- (MRI) Ruby 1.8.7 - tested seriously.
- (MRI) Ruby 1.9.3 - tested seriously.
- (REE) Ruby 1.8.7 - tested lightly.
- (RBX) Rubinius 1.1 - tested lightly.

As you can see, only Matz's Ruby Interpreter (MRI) is seriously supported.

Next is the platform, the combination of the architecture (processor) and OS. As far as I can tell, if you can compile MPC and Ruby on a given platform, you can use the mpc gem there too. Please report problems with that hypothesis.

Lastly, MPC. MPC must be compiled and working. "And working" means you ran `make check` after compiling MPC, and it 'check's out. The following version of MPC have been tested:

- MPC 1.0.1

That's all. I don't intend to test any older versions.

Installing

You may clone the mpc gem's git repository with:

```
git clone git://github.com/srawlins/gnu_mpc.git
```

Or you may install the gem from <http://rubygems.org>:

```
gem install gnu_mpc
```

At this time, the gem self-compiles. If required libraries cannot be found, you may compile the C extensions manually with:

```
cd <gnu_mpc gem directory>/ext
ruby extconf.rb
make
```

There shouldn't be any errors, or warnings.

Testing the gnu_mpc gem

Testing the gnu_mpc gem is quite simple. The testing framework uses RSpec.

```
bundle && rspec
```

MPC and mpc gem basics

Classes

The gnu_mpc gem includes the class `MPC`. There are also some constants within `MPC`:

`MPC::MPC_VERSION` - The version of MPC linked into the mpc gem

`MPC::MPC_RNDxy` - Rounding mode where x is the rounding mode for the real part and y is the rounding mode for the imaginary part.

Standard Options Hash Keys

Unless otherwise noted, each MPC instance method will accept an options hash for optional arguments, after any required arguments. For example, since `MPC#pow` requires one argument but `MPC#sin` requires none, options hashes can be supplied like so:

```
MPC.new([1,1]).pow(2, {})
MPC.new([1,1]).sin({})
```

Here are the keys typically accepted by an MPC instance method.

`:rnd`, `:round`, `:rounding`, `:rounding_mode` rounding mode for the operation

`:prec`, `:precision` precision used when initializing the return value; This precision will be used for both the real and imaginary parts of the returned complex value.

`:real_prec`, `:real_precision` precision used for the real part of the return value

`:imag_prec`, `:imag_precision` precision used for the imaginary part of the return value

Precision of Returned Values

Unless otherwise noted, the precision of a value returned by an MPC instance method follows certain standards. By default, a method will create and return an object with the MPFR default precision (which can be changed with `GMP::F.default_prec=`). The user can also pass in a preferred precision in one of three ways:

- Most methods will accept a `precision` parameter in their ordered argument list, which will be used for both the real and imaginary parts of the return object.
- Most methods accept an options hash as well. The user can pass in a preferred precision using the `:prec` or `:precision` keys, like so:

```
MPC.new(5).sin(:prec => 64)
MPC.new([7,11]).exp(:precision => 128)
```

- The user can also specify the precision of the real part, and the imaginary part, individually, using the options hash as well. The `:real_prec`, (or `:real_precision`) and `:imag_prec` (or `:imag_precision`) keys can be used:

```
MPC.new(72).sin(:real_prec => 17, :imag_prec => 53)
```

The broad `:prec` and `:precision` values are parsed before the real- and imaginary-specific values, so that

```
my_usual_options = {:rounding => MPC::MPC_RNDZZ, :prec => 64}
MPC.new(42).tan(my_usual_options.merge(:prec_real => 128))
```

will return the tangent of 42 with a precision of 64 on the real part and a precision of 128 on the imaginary part.

Complex Functions

Initializing, Assigning Complex Numbers

new

`MPC.new` → *integer*
`MPC.new(numeric = 0)` → *integer*
`MPC.new(str, base = 0)` → *integer*

This method creates a new MPC integer. It typically takes one optional argument for the value of the integer. This argument can be one of several classes. If the first argument is a String, then a second argument, the base, may be optionally supplied. Here are some examples:

```
MPC.new           #=> 0 (default)
MPC.new(1)        #=> 1 (Ruby Fixnum)
MPC.new("127")    #=> 127 (Ruby String)
MPC.new("FF", 16)  #=> 255 (Ruby String with base)
MPC.new("1Z", 36)  #=> 71 (Ruby String with base)
MPC.new(4294967296) #=> 4294967296 (Ruby Bignum)
MPC.new(GMP::Z.new(31)) #=> 31 (GMP Integer)
```

Converting Complex Numbers

String and Stream Input and Output

Complex Comparison

Projection and Decomposing

real	$\text{MPC\#real}()$	$\rightarrow GMP::F$
	$\text{MPC\#real}(\text{rounding_mode} = \text{MPC::MPC_RNDNN},$ $\text{precision} = \text{mpfr_default_precision})$	$\rightarrow GMP::F$
	Return the real part of the receiver, rounded according to <i>rounding_mode</i> .	
imag	$\text{MPC\#imag}()$	$\rightarrow GMP::F$
	$\text{MPC\#imag}(\text{rounding_mode} = \text{MPC::MPC_RNDNN},$ $\text{precision} = \text{mpfr_default_precision})$	$\rightarrow GMP::F$
	Return the imaginary part of the receiver, rounded according to <i>rounding_mode</i> .	
arg	$\text{MPC\#arg}()$	$\rightarrow GMP::F$
	$\text{MPC\#arg}(\text{rounding_mode} = \text{MPC::MPC_RNDNN},$ $\text{precision} = \text{mpfr_default_precision})$	$\rightarrow GMP::F$
	Return the argument of the receiver, rounded according to <i>rounding_mode</i> .	
proj	$\text{MPC\#proj}()$	$\rightarrow MPC$
	$\text{MPC\#proj}(\text{rounding_mode} = \text{MPC::MPC_RNDNN},$ $\text{precision} = \text{mpfr_default_precision})$	$\rightarrow MPC$
	Return the projection of the receiver onto the Riemann sphere, rounded according to <i>rounding_mode</i> .	

Basic Arithmetic

add, +	MPC#add(<i>op2</i>) → <i>complex</i> MPC#add(<i>op2</i> , <i>rounding_mode</i> = MPC::MPC_RNDNN) → <i>complex</i> <i>op1</i> + <i>op2</i> → <i>complex</i>
---------------	---

Return the sum of the receiver (*op1*) and *op2*, rounded according to *rounding_mode*. *op2* may be a Fixnum, GMP::Z, Bignum, or GMP::F.

```

a = MPC.new(GMP::F("3.1416", 12)) #=> (3.1416 +0)
a + 0                               #=> (3.1416 +0)
a + 1                               #=> (4.1406 +0)
a - -7                              #=> (-3.8584 +0)
a + GMP::Z(1024)                    #=> (1.0270e+3 +0)
a + 2**66                           #=> (7.3787e+19 +0)
a + GMP::F("3.1416", 12)           #=> (6.2832 +0)

```

sub, -	MPC#sub(<i>op2</i>) → <i>complex</i> MPC#sub(<i>op2</i> , <i>rounding_mode</i> = MPC::MPC_RNDNN) → <i>complex</i> <i>op1</i> - <i>op2</i> → <i>complex</i>
---------------	---

Return the difference between the receiver (*op1*) and *op2*, rounded according to *rounding_mode*. *op2* may be a Fixnum, GMP::Z, Bignum, or GMP::F.

```

a = MPC.new(GMP::F("3.1416", 12)) #=> (3.1416 +0)
a - 0                               #=> (3.1416 +0)
a - 1                               #=> (2.1416 +0)
a - -7                              #=> (1.0141e+1 +0)
a - GMP::Z(1024)                    #=> (-1.0208e+3 +0)
a - 2**66                           #=> (-7.3787e+19 +0)
a - GMP::F("3.1416", 12)           #=> (+0 +0)

```

mul, *	MPC#mul(<i>op2</i>) → <i>complex</i> MPC#mul(<i>op2</i> , <i>rounding_mode</i> = MPC::MPC_RNDNN) → <i>complex</i> <i>op1</i> * <i>op2</i> → <i>complex</i>
---------------	---

Return the product of the receiver (*op1*) and *op2*, rounded according to *rounding_mode*. *op2* may be a Fixnum, GMP::Z, Bignum, or GMP::F.

```

a = MPC.new(GMP::F("3.1416", 12)) #=> (3.1416 +0)
a * 0                               #=> (+0 +0)
a * 1                               #=> (3.1416 +0)
a * -7                              #=> (-2.1992e+1 -0)
a * GMP::Z(1024)                    #=> (3.2170e+3 +0)
a * 2**66                           #=> (2.3181e+20 +0)
a * GMP::F("3.1416", 12)           #=> (9.8711 +0)

```

div, /	MPC#div(<i>op2</i>) → <i>complex</i>
	MPC#div(<i>op2</i> , <i>rounding_mode</i> = MPC::MPC_RNDNN) → <i>complex</i>
	<i>op1/op2</i> → <i>complex</i>
<hr/> Return the quotient of the receiver (<i>op1</i>) and <i>op2</i> , rounded according to <i>rounding_mode</i> . <i>op2</i> may be a Fixnum, GMP::Z, Bignum, or GMP::F.	
<pre> a = MPC.new(GMP::F("3.1416", 12)) #=> (3.1416 +0) a / 0 #=> (Inf NaN) a / 1 #=> (3.1416 +0) a / -7 #=> (-4.4885e-1 -0) a / GMP::Z(1024) #=> (3.0680e-3 +0) a / 2**66 #=> (4.9088e-2 +0) a / GMP::F("3.1416", 12) #=> (1.0000 +0) </pre>	

Power Functions and Logarithm

sqrt	MPC#sqrt() → <i>complex</i>
	MPC#sqrt(<i>rounding_mode</i> = MPC::MPC_RNDNN, <i>precision</i> = <i>mpfr_default_precision</i>) → <i>complex</i>
	MPC#sqrt(<i>options_hash</i>) → <i>complex</i>
<hr/> Return the square root of the receiver, rounded according to <i>rounding_mode</i> . The returned value has a non-negative real part, and if its real part is zero, a non-negative imaginary part.	
pow	MPC#pow() → <i>complex</i>
	<i>Not implemented yet.</i>
exp	MPC#exp() → <i>complex</i>
	MPC#exp(<i>rounding_mode</i> = MPC::MPC_RNDNN, <i>precision</i> = <i>mpfr_default_precision</i>) → <i>complex</i>
	MPC#exp(<i>options_hash</i>) → <i>complex</i>
<hr/> Return the exponential of the receiver, rounded according to <i>rounding_mode</i> .	
log	MPC#log() → <i>complex</i>
	MPC#log(<i>rounding_mode</i> = MPC::MPC_RNDNN, <i>precision</i> = <i>mpfr_default_precision</i>) → <i>complex</i>
	MPC#log(<i>options_hash</i>) → <i>complex</i>
<hr/> Return the natural logarithm of the receiver, rounded according to <i>rounding_mode</i> . The principal branch is chosen, with the branch cut on the negative real axis, so that the imaginary part of the result lies in $[-\pi, \pi]$ and $[-\pi/\log(10), \pi/\log(10)]$ respectively.	
log10	MPC#log10() → <i>complex</i>
	MPC#log10(<i>rounding_mode</i> = MPC::MPC_RNDNN, <i>precision</i> = <i>mpfr_default_precision</i>) → <i>complex</i>
	MPC#log10(<i>options_hash</i>) → <i>complex</i>
<hr/> Return the base-10 logarithm of the receiver, rounded according to <i>rounding_mode</i> . The principal branch is chosen, with the branch cut on the negative real axis, so that the imaginary part of the result lies in $[-\pi, \pi]$ and $[-\pi/\log(10), \pi/\log(10)]$ respectively.	

Trigonometric Functions

sin	MPC#sin()	→ complex
	MPC#sin(<i>rounding_mode</i> = MPC::MPC_RNDNN,	
	<i>precision</i> = mpfr_default_precision)	→ complex
	MPC#sin(<i>options_hash</i>)	→ complex
Return the sine of the receiver, rounded according to <i>rounding_mode</i> .		
cos	MPC#cos()	→ complex
	MPC#cos(<i>rounding_mode</i> = MPC::MPC_RNDNN,	
	<i>precision</i> = mpfr_default_precision)	→ complex
	MPC#cos(<i>options_hash</i>)	→ complex
Return the cosine of the receiver, rounded according to <i>rounding_mode</i> .		
sin_cos	MPC#sin_cos()	→ complex
	Not implemented yet.	
tan	MPC#tan()	→ complex
	MPC#tan(<i>rounding_mode</i> = MPC::MPC_RNDNN,	
	<i>precision</i> = mpfr_default_precision)	→ complex
	MPC#tan(<i>options_hash</i>)	→ complex
Return the tangent of the receiver, rounded according to <i>rounding_mode</i> .		
sinh	MPC#sinh()	→ complex
	MPC#sinh(<i>rounding_mode</i> = MPC::MPC_RNDNN,	
	<i>precision</i> = mpfr_default_precision)	→ complex
	MPC#sinh(<i>options_hash</i>)	→ complex
Return the hyperbolic sine of the receiver, rounded according to <i>rounding_mode</i> .		
cosh	MPC#cosh()	→ complex
	MPC#cosh(<i>rounding_mode</i> = MPC::MPC_RNDNN,	
	<i>precision</i> = mpfr_default_precision)	→ complex
	MPC#cosh(<i>options_hash</i>)	→ complex
Return the hyperbolic cosine of the receiver, rounded according to <i>rounding_mode</i> .		
tanh	MPC#tanh()	→ complex
	MPC#tanh(<i>rounding_mode</i> = MPC::MPC_RNDNN,	
	<i>precision</i> = mpfr_default_precision)	→ complex
	MPC#tanh(<i>options_hash</i>)	→ complex
Return the hyperbolic tangent of the receiver, rounded according to <i>rounding_mode</i> .		

asin	MPC#asin()	→ <i>complex</i>
	MPC#asin(<i>rounding_mode</i> = MPC::MPC_RNDNN, <i>precision</i> = <i>mpfr_default_precision</i>)	→ <i>complex</i>
	MPC#asin(<i>options_hash</i>)	→ <i>complex</i>
	Return the inverse sine of the receiver, rounded according to <i>rounding_mode</i> .	
acos	MPC#acos()	→ <i>complex</i>
	MPC#acos(<i>rounding_mode</i> = MPC::MPC_RNDNN, <i>precision</i> = <i>mpfr_default_precision</i>)	→ <i>complex</i>
	MPC#acos(<i>options_hash</i>)	→ <i>complex</i>
	Return the inverse cosine of the receiver, rounded according to <i>rounding_mode</i> .	
atan	MPC#atan()	→ <i>complex</i>
	MPC#atan(<i>rounding_mode</i> = MPC::MPC_RNDNN, <i>precision</i> = <i>mpfr_default_precision</i>)	→ <i>complex</i>
	MPC#atan(<i>options_hash</i>)	→ <i>complex</i>
	Return the inverse tangent of the receiver, rounded according to <i>rounding_mode</i> .	

Miscellaneous Complex Functions

Advanced Functions

Internals