

MPFR CX

The Multiple Precision Real and Complex Polynomial Library
Version 0.3.1
September 2010

Andreas Enge

Copyright © 2007, 2008, 2009, 2010 Andreas Enge

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.

MPFRCX Copying Conditions

The MPFRCX Library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version, see the file `COPYING.LIB`.

The MPFRCX Library is distributed in the hope that it will be useful, but **WITHOUT ANY WARRANTY**; without even the implied warranty of **MERCHANTABILITY** or **FITNESS FOR A PARTICULAR PURPOSE**. See the GNU Lesser General Public License for more details.

1 Introduction to MPFRCX

MPFRCX is a portable library written in C for arithmetic of polynomials with arbitrary precision real or complex floating point coefficients. It is based on the GNU MP, the MPFR and the MPC libraries.

2 Installing MPFRCX

To build MPFRCX, you first have to install GNU MP, MPFR and MPC on your computer. For MPC, the minimal supported version is 0.7; the minimally required versions of GNU MP and MPFR depend on the MPC version used. You need a C compiler, preferably GCC, but any reasonable compiler should work. And you need a standard Unix ‘make’ program, plus some other standard Unix utility programs.

Here are the steps needed to install the library on Unix systems:

1. ‘tar xzf mpfrcx-0.3.1.tar.gz’
2. ‘cd mpfrcx-0.3.1’
3. ‘./configure’

if GMP, MPFR and MPC are installed into standard directories, that is, directories that are searched by default by the compiler and the linking tools.

‘./configure --with-gmp=<gmp_install_dir>’

is used to indicate a different location where GMP is installed.

‘./configure --with-mpfr=<mpfr_install_dir>’

is used to indicate a different location where MPFR is installed.

‘./configure --with-mpc=<mpc_install_dir>’

is used to indicate a different location where MPC is installed.

‘./configure --with-gmp=<gmp_install_dir> --with-mpfr=<mpfr_install_dir>
--with-mpc=<mpc_install_dir>’

is used to indicate different locations of GMP, MPFR and MPC.

Warning! Do not use these options if you have CPPFLAGS and/or LDFLAGS containing a -I or -L option with a directory that contains a GMP header or library file, as these options just add -I and -L options to CPPFLAGS and LDFLAGS *after* the ones that are currently declared, so that DIR will have a lower precedence. Also, this may not work if DIR is a system directory.

4. ‘make’

This compiles MPFRCX in the working directory.

5. ‘make check’

This will make sure MPFRCX was built correctly.

If you get error messages, please report them to ‘andreas.enge@inria.fr’ (See [Chapter 3 \[Reporting Bugs\]](#), page 5, for information on what to include in useful bug reports).

6. ‘make install’

This will copy the file ‘mpfrcx.h’ to the directory ‘/usr/local/include’, the file ‘libmpfrcx.a’ to the directory ‘/usr/local/lib’, and the file ‘mpfrcx.info’ to the directory ‘/usr/local/share/info’ (or if you passed the ‘--prefix’ option to ‘configure’, using the prefix directory given as argument to ‘--prefix’ instead of ‘/usr/local’). Note: you need write permissions on these directories.

2.1 Other ‘make’ Targets

There are some other useful make targets:

- ‘mpfrcx.pdf’ or ‘pdf’ inside the ‘doc’ subdirectory
Create a PDF version of the manual, in ‘mpfrcx.pdf’.

- `'mpfrcx.html'` or `'html'` inside the `'doc'` subdirectory
Create an HTML version of the manual, in several pages in the directory `'mpfrcx.html'`; if you want only one output HTML file, then type `'makeinfo --html --no-split mpfrcx.texi'` instead.
- `'clean'`
Delete all object files and archive files, but not the configuration files.
- `'distclean'`
Delete all files not included in the distribution.
- `'uninstall'`
Delete all files copied by `'make install'`.

3 Reporting Bugs

If you think you have found a bug in the MPFRCX library, please investigate and report it.

There are a few things you should take into account when you compose your bug report. Please send us a minimal test case that makes it possible for us to reproduce the bug. Include instructions on how to run the test case, and why the result differs from what you would expect.

Please include compiler version information in your bug report. This can be extracted using `'cc -V'` on some machines, or, if you are using gcc, `'gcc -v'`. Also, include the output from `'uname -a'`.

Send your bug report to: `'andreas.enge@inria.fr'`.

Beware that the MPFRCX library is in a very early development stage, and some functions, while working correctly, may be terribly inefficient. You might want to send an e-mail to the above address if you are interested in one of the more exotic functions to enquire about its status.

4 MPFRCX Basics

MPFRCX provides types and functions for working with univariate polynomials, taking as coefficients either real or complex floating point numbers of arbitrary precision. The functions are collected in the library ‘`libmpfrcx.a`’ and declared in the header file ‘`mpfrcx.h`’.

4.1 Nomenclature and Types

A *real polynomial* is a polynomial whose coefficients are of type `mpfr_t`. The C data type for such objects is `mpfrx_t`. All coefficients are supposed to have the same floating point precision. Besides its list of coefficients, a variable of type `mpfrx_t` contains the degree of the polynomial as an `int` and the precision of its coefficients as an `mp_prec_t`. If the degree of a polynomial increases, its list of coefficients is lengthened accordingly; on the other hand, if the degree decreases, the memory allocated to the now superfluous coefficients is not freed, unless explicitly requested by a call to `mpfrx_realloc`, see [Section 5.1 \[Initialisation Functions\]](#), page 7.

A *complex polynomial* is a polynomial whose coefficients are of type `mpc_t`. The C data type for such objects is `mpcx_t`. All coefficients are supposed to have the same floating point precision, and this both for their real and their imaginary parts. Otherwise, complex polynomials behave like real ones.

When calling the functions described in the following, arguments of type `mpfrx_ptr` or `mpfrx_srcptr` stand for arbitrary variables of type `mpfrx_t`; the former may be modified by the function, the latter not. The same holds for `mpcx_ptr` and `mpcx_srcptr`.

Notice that unlike for operations with real numbers of type `mpfr_t` and complex numbers of type `mpc_t`, there are no rounding modes for operations with polynomials and no precise semantics; polynomial arithmetic is realised by calls to functions on the coefficients, which may accumulate rounding errors.

4.2 Function Classes

Functions and macros working with real polynomials begin with `mpfrx_`, those treating complex polynomials begin with `mpcx_`. For the time being, there are no mixed operations.

4.3 MPFRCX Variable Conventions

As a general rule, all MPFRCX functions expect output arguments before input arguments, in analogy with GMP, MPFR and MPC.

MPFRCX allows you to use the same variable for both input and output in the same expression. For example, the main function for multiplication of real polynomials, `mpfrx_mul`, can be used like this: `mpfrx_mul (f, f, f)` to replace f by its square.

Before you can assign to an MPFRCX variable, you need to initialise it by calling one of the special initialisation functions. When you are done with a variable, you need to clear it out, using one of the functions for that purpose.

A variable should be initialised only once; after a variable has been initialised, values may be assigned to it any number of times.

5 Functions

All publicly visible functions exist with the same behaviour for real or complex polynomials; their names start with `mpfrx_` resp. `mpcx_`. For the time being, the only specific functions concern the fast Fourier transform and are internal to the library.

5.1 Initialisation Functions

An `mpfrx_t` or `mpcx_t` object must be initialised before storing the first value in it, using the function `mpfrx_init` or `mpcx_init`.

`void mpfrx_init (mpfrx_ptr f, const int size, const mp_prec_t prec)` [Function]

`void mpcx_init (mpcx_ptr f, const int size, const mp_prec_t prec)` [Function]

Initialise f with initial space for $size$ coefficients of precision $prec$, and set it to zero. It is possible to assign a polynomial with more than $size$ coefficients to f later on; the size of f is then increased automatically. Beware that $size = d + 1$ coefficients are needed to store a polynomial of degree d .

`void mpfrx_clear (mpfrx_ptr z)` [Function]

`void mpcx_clear (mpcx_ptr z)` [Function]

Free the space currently occupied by z . Make sure to call this function on each variable precisely once.

`void mpfrx_realloc (mpfrx_ptr f, const int size)` [Function]

`void mpcx_realloc (mpcx_ptr f, const int size)` [Function]

Changes the number of coefficients stored in f to $size$, which may be more or less than (or equal to) the previous size, while preserving the precision of the coefficients. If f still fits (that is, its degree is at most $size - 1$), its value is preserved, otherwise, it is replaced by 0.

5.2 Assignment Functions

These functions assign new values to already initialised polynomials (see [Section 5.1 \[Initialisation Functions\]](#), page 7).

`void mpfrx_set (mpfrx_ptr h, mpfrx_srcptr f)` [Function]

`void mpcx_set (mpcx_ptr h, mpcx_srcptr f)` [Function]

Set the value of h from f . The precision of h is preserved, and the coefficients of f are rounded if the target precision is lower.

5.3 Combined Initialisation and Assignment Functions

`void mpfrx_init_set (mpfrx_ptr h, mpfrx_srcptr op)` [Function]

`void mpcx_init_set (mpcx_ptr h, mpcx_srcptr op)` [Function]

Initialise h with the same precision as f and set its value from f .

5.4 Access Functions

`int mpfrx_get_prec (mpfrx_srcptr f)` [Macro]

`int mpcx_get_prec (mpcx_srcptr f)` [Macro]

Return the common precision of the coefficients of f .

`void mpfrx_set_prec (mpfrx_ptr f, const mp_prec_t prec)` [Function]

`void mpcx_set_prec (mpcx_ptr f, const mp_prec_t prec)` [Function]

Set the precision of the coefficients of f to $prec$ and replace f by the zero polynomial. The effect is similar to a call to `mpfrx_clear` resp. `mpcx_clear` followed by a call to `mpfrx_init` resp. `mpcx_init`, but the number of coefficients in the polynomial is kept.

```
int mpfrx_get_deg (mpfrx_srcptr f) [Macro]
int mpcx_get_deg (mpcx_srcptr f) [Macro]
```

Return the degree of f , which is -1 for the zero polynomial.

```
void mpfrx_set_deg (mpfrx_ptr f, const int deg) [Function]
void mpcx_set_deg (mpcx_ptr f, const int deg) [Function]
```

Set the degree of f to deg while preserving the coefficients. If the degree increases, the new coefficients are set to NaN and need to be set manually before computing with the variable, see `mpfrx_set_coeff` and `mpcx_set_coeff`. If necessary, new coefficients are allocated.

```
mpfr_ptr mpfrx_get_coeff (mpfrx_srcptr f, const unsigned int i) [Function]
mpc_ptr mpcx_get_coeff (mpcx_srcptr f, const unsigned int i) [Function]
```

Return a pointer to coefficient i of f , or NULL if the degree of f is less than i .

```
void mpfrx_set_coeff (mpfrx_ptr f, const unsigned int i, mpfr_srcptr [Function]
                    coeff)
```

```
void mpcx_set_coeff (mpcx_ptr f, const unsigned int i, mpc_srcptr coeff) [Function]
```

Set the coefficient i of f to $coeff$. If the current degree of f is smaller than i , then the degree of f is set to i ; intermediate coefficients are set to NaN.

5.5 Comparison and Miscellaneous Functions

```
int mpfrx_cmp (mpfrx_srcptr f, mpfr_srcptr g) [Function]
int mpcx_cmp (mpcx_srcptr f, mpcx_srcptr g) [Function]
```

Return 0 if f equals g and -1 if not. The coefficients of f and g are compared one by one; so even if the two polynomials have different precisions, they may be recognised as equal.

```
void mpfrx_urandom (mpfrx_ptr f, int deg, gmp_rand_state_t state) [Function]
void mpcx_urandom (mpcx_ptr f, int deg, gmp_rand_state_t state) [Function]
```

If $deg < 0$, set f to be the 0 polynomial. Otherwise, generate a uniformly distributed random degree between 0 and deg (inclusive), and a random polynomial of this degree. Each coefficient is chosen uniformly at random in the unit interval $[0, 1]$ resp. the unit square $[0, 1] \times [0, 1]$; except for the leading coefficient, which cannot be 0.

$state$ is a `gmp_randstate_t` structure which should be created using the GMP `rand_init` function, see the GMP manual.

```
const int MPFRXC_VERSION_MAJOR [Macro]
const int MPFRXC_VERSION_MINOR [Macro]
const int MPFRXC_VERSION_PATCHLEVEL [Macro]
const char* MPFRXC_VERSION_STRING [Macro]
```

The major, minor and patchlevel version number of the library. These are concatenated and separated by '.' to form the version string; '-dev' is added to the version string of development snapshots.

```
const char * mpfrcx_get_version (void) [Function]
```

Return the MPFRXC version as a null-terminated string.

5.6 Input and Output Functions

The following function writes a polynomial to an output stream. When using it, you need to include 'stdio.h' before 'mpfrcx.h'.

`size_t mpfrx_out_str (FILE* stream, int base, size_t n_digits, mpfrx_srcptr f)` [Function]

`size_t mpcx_out_str (FILE* stream, int base, size_t n_digits, mpcx_srcptr f)` [Function]

Output f to $stream$ using the given $base$ and the given number of digits n_digits for the coefficients. If n_digits is 0, then a suitable number of digits is chosen so that reading the polynomial into a variable of the same precision as f yields the same polynomial again (this input function needs yet to be written...).

The output starts with an opening bracket ' $($ ', followed by the degree and a list of coefficients in decreasing order of degree (separated by spaces) and a closing bracket ' $)$ '.

Return the number of written characters.

5.7 Basic Polynomial Arithmetic

`void mpfrx_add (mpfrx_ptr h, mpfrx_srcptr f, mpfrx_srcptr g)` [Function]

`void mpcx_add (mpcx_ptr h, mpcx_srcptr f, mpcx_srcptr g)` [Function]

Set h to $f + g$.

`void mpfrx_sub (mpfrx_ptr h, mpfrx_srcptr f, mpfrx_srcptr g)` [Function]

`void mpfrx_si_sub (mpfrx_ptr h, const long int f, mpfrx_srcptr g)` [Function]

`void mpcx_sub (mpcx_ptr h, mpcx_srcptr f, mpcx_srcptr g)` [Function]

`void mpcx_si_sub (mpcx_ptr h, const long int f, mpcx_srcptr g)` [Function]

Set h to $f - g$.

`void mpfrx_neg (mpfrx_ptr h, mpfrx_srcptr f)` [Function]

`void mpcx_neg (mpcx_ptr h, mpcx_srcptr f)` [Function]

Set h to $-f$.

`void mpfrx_mul (mpfrx_ptr h, mpfrx_srcptr f, mpfrx_srcptr g)` [Function]

`void mpcx_mul (mpcx_ptr h, mpcx_srcptr f, mpcx_srcptr g)` [Function]

Set h to $f * g$.

`void mpfrx_rem (mpfrx_ptr r, mpfrx_srcptr f, mpfrx_srcptr g)` [Function]

`void mpcx_rem (mpcx_ptr r, mpcx_srcptr f, mpcx_srcptr g)` [Function]

Set r to the remainder of f divided by g .

5.8 High-Level Functions

`void mpfrx_reconstruct (mpfrx_ptr h, mpfr_t* factors, int no, int verbose)` [Function]

`void mpcx_reconstruct (mpcx_ptr h, mpcx_t* factors, int no, int verbose)` [Function]

Computes the product of the first no polynomials in the array $factors$ and stores it in h (which may be one of the elements of $factors$). If the boolean parameter $verbose$ is `true`, then intermediate timing information is output on screen.

`void mpfrx_multieval (mpfr_ptr* values, mpfr_srcptr* args, int no, mpfr_t f)` [Function]

`void mpcx_multieval (mpc_ptr* values, mpc_srcptr* args, int no, mpcx_t f)` [Function]

Evaluate the polynomial f in the first no elements of the array $args$, and store the values in the first no entries of $values$ (that must exist and already be initialised).

```
void mpfrx_root (mpfr_ptr root, mpfr_srcptr f) [Function]
void mpcx_root (mpc_ptr root, mpc_srcptr f) [Function]
```

Computes a root of f . The variable $root$ is supposed to contain an initial approximation, that is refined via Newton iterations until it does not change any more. No special care is taken to avoid infinite loops.

Contributors

The main developer of the MPFRCX library is Andreas Enge.

References

- Torbjörn Granlund et al. `gmp` – GNU multiprecision library. Version 4.2, <http://gmplib.org/>.
- Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Philippe Théveny and Paul Zimmermann. `mpfr` – A library for multiple-precision floating-point computations with exact rounding. Version 2.3.1, <http://www.mpfr.org/>.
- Andreas Enge, Philippe Théveny and Paul Zimmermann. `mpc` – A library for multiprecision complex arithmetic with exact rounding. Version 0.7, <http://mpc.multiprecision.org/>.

Concept Index

A

Access Functions	7
Assignment Functions	7

B

Basic Polynomial Arithmetic	9
-----------------------------------	---

C

Combined Initialisation and Assignment Functions	7
Comparison Functions	8
Complex polynomial	6
Conditions for copying MPFRGX	1
Copying conditions	1

F

Functions	7
-----------------	---

H

High-Level Functions	9
----------------------------	---

I

I/O functions	8
Initialisation Functions	7
Input functions	8
Installation	3

L

'libmpfrcx.a'	6
---------------------	---

M

Miscellaneous Functions	8
'mpfrcx.h'	6

O

Output functions	8
------------------------	---

R

Real polynomial	6
Reporting bugs	5

Function and Type Index

C

const char* MPFRCX_VERSION_STRING..... 8
 const int MPFRCX_VERSION_MAJOR..... 8
 const int MPFRCX_VERSION_MINOR..... 8
 const int MPFRCX_VERSION_PATCHLEVEL..... 8

I

int mpcx_get_deg..... 8
 int mpcx_get_prec..... 7
 int mpfrx_get_deg..... 8
 int mpfrx_get_prec..... 7

M

mpcx_add..... 9
 mpcx_clear..... 7
 mpcx_cmp..... 8
 mpcx_get_coeff..... 8
 mpcx_init..... 7
 mpcx_init_set..... 7
 mpcx_mul..... 9
 mpcx_multieval..... 9
 mpcx_neg..... 9
 mpcx_out_str..... 9
 mpcx_realloc..... 7
 mpcx_reconstruct..... 9
 mpcx_rem..... 9
 mpcx_root..... 10
 mpcx_set..... 7

mpcx_set_coeff..... 8
 mpcx_set_deg..... 8
 mpcx_set_prec..... 7
 mpcx_si_sub..... 9
 mpcx_sub..... 9
 mpcx_t..... 6
 mpcx_urandom..... 8
 mpfrx_get_version..... 8
 mpfrx_add..... 9
 mpfrx_clear..... 7
 mpfrx_cmp..... 8
 mpfrx_get_coeff..... 8
 mpfrx_init..... 7
 mpfrx_init_set..... 7
 mpfrx_mul..... 9
 mpfrx_multieval..... 9
 mpfrx_neg..... 9
 mpfrx_out_str..... 9
 mpfrx_realloc..... 7
 mpfrx_reconstruct..... 9
 mpfrx_rem..... 9
 mpfrx_root..... 10
 mpfrx_set..... 7
 mpfrx_set_coeff..... 8
 mpfrx_set_deg..... 8
 mpfrx_set_prec..... 7
 mpfrx_si_sub..... 9
 mpfrx_sub..... 9
 mpfrx_t..... 6
 mpfrx_urandom..... 8

Table of Contents

MPFRCX Copying Conditions	1
1 Introduction to MPFRCX	2
2 Installing MPFRCX	3
2.1 Other ‘make’ Targets	3
3 Reporting Bugs	5
4 MPFRCX Basics	6
4.1 Nomenclature and Types	6
4.2 Function Classes	6
4.3 MPFRCX Variable Conventions	6
5 Functions	7
5.1 Initialisation Functions	7
5.2 Assignment Functions	7
5.3 Combined Initialisation and Assignment Functions	7
5.4 Access Functions	7
5.5 Comparison and Miscellaneous Functions	8
5.6 Input and Output Functions	8
5.7 Basic Polynomial Arithmetic	9
5.8 High-Level Functions	9
Contributors	11
References	12
Concept Index	13
Function and Type Index	14

