MPFRCX

Multiple Precision Real and Complex Polynomial Library
Edition 0.6.1
April 2021

Andreas Enge

This manual is for MPFRCX, a library for the arithmetic of polynomials with multiple precision real or complex floating point coefficients, version 0.6.1 of April 2021.

MPFRCX Copying Conditions

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 1.0; 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-0.6.1.tar.gz`'
2. '`cd mpfrcx-0.6.1`'
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 `docsubdirectory`

  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]
`void mpcx_set_frx` (*mpcx_ptr* `h`, *mpfrx_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.

`void mpfrx_swap` (*mpfrx_ptr* `f`, *mpfrx_ptr* `g`) [Function]
`void mpcx_swap` (*mpcx_ptr* `f`, *mpcx_ptr* `g`) [Function]

Swap the contents of the variables *f* and *g*. If their coefficients do not have the same precision, precisions are swapped as well. The effect is thus not the same as obtained by several calls to `mpfrx_set` or `mpcx_set`, respectively, with an additional temporary variable, which would keep the respective precisions of *f* and *g* unchanged.

5.3 Combined Initialisation and Assignment Functions

`void mpfrx_init_set` (*mpfrx_ptr* `h`, *mpfrx_srcptr* `f`) [Function]
`void mpcx_init_set` (*mpcx_ptr* `h`, *mpcx_srcptr* `f`) [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_prc_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* `coeff`) [Function]
`void mpcx_set_coeff` (*mpcx_ptr* `f`, *const unsigned int* `i`, *mpc_scrptr* `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`, *mpfrx_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 MPFRCX_VERSION_MAJOR` [Macro]
`const int MPFRCX_VERSION_MINOR` [Macro]
`const int MPFRCX_VERSION_PATCHLEVEL` [Macro]

`const char* MPFRCX_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 MPFRCX 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 mpfrx_mul_fr` (*mpfrx_ptr* `h`, *mpfrx_srcptr* `f`, *mpfr_srcptr* `g`) [Function]

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

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

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

`void mpcx_mul_c` (*mpcx_ptr* `h`, *mpcx_srcptr* `f`, *mpc_srcptr* `g`) [Function]

`void mpcx_mul_fr` (*mpcx_ptr* `h`, *mpcx_srcptr* `f`, *mpfr_srcptr* `g`) [Function]

`void mpcx_mul_si` (*mpcx_ptr* `h`, *mpcx_srcptr* `f`, *const long int* `g`) [Function] `void mpcx_mul_ui` (*mpcx_ptr* `h`, *mpcx_srcptr* `f`, *const unsigned long int* `g`) [Function]

Set h to $f * g$.

`void mpfrx_mul_x` (*mpfrx_ptr* `h`, *mpfrx_srcptr* `f`, *const unsigned int* `n`) [Function]
`void mpcx_mul_x` (*mpcx_ptr* `h`, *mpcx_srcptr* `f`, *const unsigned int* `n`) [Function]

Set h to $f * x^n$

`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.

`void mpfrcx_real` (*mpfrx_ptr* `h`, *mpcx_srcptr* `f`) [Function]
`void mpfrcx_imag` (*mpfrx_ptr* `h`, *mpcx_srcptr* `f`) [Function]

Set h to the real or the imaginary part of f, respectively. The prefix `mpfrcx` indicates the mixed type of the operations, with an `mpcx` argument and an `mpfrx` result.

5.8 Higher Level Functions

`void mpfrx_eval` (*mpfr_ptr* `r`, *mpfrx_srcptr* `f`, *mpfr_srcptr* `x`) [Function]
`void mpcx_eval` (*mpc_ptr* `r`, *mpcx_srcptr* `f`, *mpc_srcptr* `x`) [Function]

Set r to the value f(x) of f at x, obtained using a Horner scheme.

`void mpfrx_derive` (*mpfrx_ptr* `h`, *mpfrx_srcptr* `f`) [Function]
`void mpcx_derive` (*mpcx_ptr* `h`, *mpcx_srcptr* `f`) [Function]

Set h to the derivative of f.

`void mpfrx_root` (*mpfr_ptr* `root`, *mpfrx_srcptr* `f`) [Function]
`void mpcx_root` (*mpc_ptr* `root`, *mpcx_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.

5.9 Tree Based Functions

The following functions implement asymptotically fast operations on arrays of polynomials, usually through the use of subproduct trees. Such a tree is a binary tree constructed from an array of polynomials by storing these polynomials in the leaves of the tree. Each parent node contains the product of the child nodes, so that the root of the tree contains the product of all the leaves. Internally, subproduct trees are variables of types `mpfrx_tree_t` and `mpcx_tree_t`. (Analogously to the situation with polynomials, in the following, `mpfrx_tree_ptr` and `mpfcx_tree_ptr` are used for variables of type `mpfrx_tree_t` and `mpcx_tree_t` that may be modified by the function, and `mpfrx_tree_srcptr` and `mpfcx_tree_srcptr` for variables that are not modified.)

`void mpfrx_tree_init` (*mpfrx_tree_ptr* `t`, *int* `no`, *mpfr_prec_t* `prec`) [Function]
`void mpcx_tree_init` (*mpcx_tree_ptr* `t`, *int* `no`, *mpfr_prec_t* `prec`) [Function]

Initialises *t* as a subproduct tree with *no* leaves in which all polynomials stored in the nodes will have precision *prec*. All nodes are initialised by a call to `mpfrx_init` or `mpcx_init`, respectively.

void **mpfrx_tree_clear** (*mpfrx_tree_ptr* `t`) [Function]
void **mpcx_tree_clear** (*mpcx_tree_ptr* `t`) [Function]

Frees the subproduct tree referenced by *t*, and all the polynomials stored in its nodes by calls to `mpfrx_clear` or `mpcx_clear`, respectively.

void **mpfrx_subproducttree** (*mpfrx_tree_ptr* `t`, *mpfrx_t* `*factors`) [Function]
void **mpcx_subproducttree** (*mpcx_tree_ptr* `t`, *mpcx_t* `*factors`) [Function]

Computes the subproduct tree *t* whose leaves contains the polynomials in the array *factors*. The variable *t* needs to have been initialised by a call to `mpfrx_tree_init` or `mpcx_tree_init`, respectively, and *factors* needs to contain at least as many elements as there are leaves in *t*. (If there are more elements in *factors*, the last ones are ignored.)

So a typical usage is a call to `mpfrx_tree_init`, followed by a call to `mpfrx_subproducttree`, and finally a call to `mpfrcx_tree_clear`.

factors is not modified by the function.

void **mpfrx_tree_get_root** (*mpfrx_ptr* `f`, *mpfrx_tree_srcptr* `t`) [Function]
void **mpcx_tree_get_root** (*mpcx_ptr* `f`, *mpcx_tree_srcptr* `t`) [Function]

Assigns the root of the tree *t* to *f*.

For instance, if *t* has been obtained by a call to `mpfrx_subproducttree` or `mpcx_subproducttree`, this retrieves the product of all polynomials on the leaves.

void **mpfrx_reconstruct** (*mpfrx_ptr* `h`, *mpfrx_t** `factors`, *int* `no`) [Function]
void **mpcx_reconstruct** (*mpcx_ptr* `h`, *mpcx_t** `factors`, *int* `no`) [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*; apart from that, *factors* is not modified).

The same effect could be obtained by a call to `mpfrx_subproducttree` or `mpcx_subproducttree`, respectively, followed by `mpfrx_tree_get_root` or `mpcx_tree_get_root`, respectively. But to save space by a factor of O(log(`no`)), this function uses a separate implementation.

void **mpfrx_hecke** (*mpfrx_ptr* `rop`, *mpfrx_tree_srcptr* `subproducts`, *mpfrx_t* `*vals`) [Function]
void **mpcx_hecke** (*mpcx_ptr* `rop`, *mpcx_tree_srcptr* `subproducts`, *mpcx_t* `*vals`) [Function]

Assume that *t* has been created (by a call to `mpfrx_subproducttree` or `mpcx_subproducttree`, respectively) with the elements of the array *factors* on its leaves, and let *f* be the product of all the elements in *factors* (or, equivalently, the polynomial at the root of *subproducttree*). Then the function computes Σ *vals*[i]**f*/*factors*[i] and returns the result in *rop*. It can be used to compute the Hecke representation of algebraic numbers, whence its name.

void **mpfrx_product_and_hecke** (*mpfrx_t* `*rop`, *mpfrx_t* `**vals`, *int* `no_pols`, *int* `no_factors`) [Function]
void **mpcx_product_and_hecke** (*mpcx_t* `*rop`, *mpcx_t* `**vals`, *int* `no_pols`, *int* `no_factors`) [Function]

Combines one call to `mpfrx_subproducttree` (resp. `mpcx_subproducttree`) and one or more calls to `mpfrx_hecke` (resp. `mpcx_hecke`) into one function, which allows to not store the subproduct tree and thus to conserve memory without computational overhead. The function behaves as if a subproduct tree were created from *vals*[0], which needs to contain *no_factors* elements; the root of the tree is returned in *rop*[0]. For *i* from 1 to *no_pols-1*, it then behaves as if it called `mpfrx_hecke` (resp. `mpcx_hecke`) with this subproduct tree and *vals*[i], which needs to also contain *no_factors* values; the result of the operation is stored in *rop*[i].

void `mpfrx_multieval` (*mpfr_t** **values**, *mpfr_t** **args**, *int* **no**, *mpfrx_t* **f**) [Function]
void `mpcx_multieval` (*mpc_t** **values**, *mpc_t** **args**, *int* **no**, *mpcx_t* **f**) [Function]

Evaluates 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 initalised).

The following functions are convenience functions; they operate more or less like their counterparts without the suffix `_from_roots` (see the individual functions for small differences), but instead of a list of polynomials, they take as input a list of roots, which are interpreted as linear polynomials with this root.

void `mpfrx_reconstruct_from_roots` (*mpfrx_ptr* **h**, *mpfr_t** **roots**, *int* **no**) [Function]
void `mpcx_reconstruct_from_roots` (*mpcx_ptr* **h**, *mpc_t** **roots**, *int* **no**) [Function]

Compute the polynomial *h* of degree *no* that has the elements of *roots* as roots (potentially with multiplicity).

void `mpfrx_subproducttree_from_roots` (*mpfrx_tree_ptr* **t**, *mpfr_t* ***roots**, *int* **no**) [Function]
void `mpcx_subproducttree_from_roots` (*mpcx_tree_ptr* **t**, *mpc_t* ***roots**, *int* **no**) [Function]

Unlike their counterparts without the `_from_roots` suffix, these functions also initialise the subproduct tree to the correct size *no*; so they should be called with an uninitialised argument *t*, which needs to be cleared explicitly later.

This inconsistency is motivated by the corresponding behaviour of `mpfrcx_subproducttree_from_roots`, described below.

void `mpfrx_hecke_from_roots` (*mpfrx_ptr* **rop**, *mpfrx_tree_srcptr* **subproducts**, *mpfr_t* ***vals**) [Function]
void `mpcx_hecke_from_roots` (*mpcx_ptr* **rop**, *mpcx_tree_srcptr* **subproducts**, *mpc_t* ***vals**) [Function]

The functions assume that *subproducts* has been generated (for instance by a call to `mpfrx_subproducttree_from_roots` or `mpcx_subproducttree_from_roots`) from monic linear polynomials; then *vals* corresponds to an array of constant polynomials, which are simply passed as complex numbers.

void `mpfrx_product_and_hecke_from_roots` (*mpfrx_t* ***rop**, *mpfr_t* ****vals**, *int* **no_pols**, *int* **no_factors**) [Function]
void `mpcx_product_and_hecke_from_roots` (*mpcx_t* ***rop**, *mpc_t* ****vals**, *int* **no_pols**, *int* **no_factors**) [Function]

The functions work in the same way as `mpfrx_products_and_hecke` and `mpcx_products_and_hecke`, except that the elements in `vals[0]` are interpreted as linear polynomials with the elements as roots, and the other elements in `vals` as constant polynomials. Otherwise said, they work like a call to `mpcx_subproducttree_from_roots` or `mpfrx_subproducttree_from_roots` with `vals [0]`, followed by calls to `mpcx_hecke_from_roots` or `mpfrx_hecke_from_roots` with `vals [i]` for *i* from 1 to *no_pols*−1.

The following functions go one step further: They work with real polynomials given by their complex roots; this “mixed” nature of the operations is indicated by the prefix `mpfrcx` instead of `mpfrx` or `mpcx`. To use these functions, one needs to indicate whether a root is real or complex, and in the latter case, how the roots are paired. So together with an array *roots* of elements of type `mpc_t`, one needs to pass an array of integers *conjugates* of the same length, where `conjugates [i] == j` if and only if `roots [j]` is the complex conjugate of `roots [i]`. In particular, `conjugates [i] == i` if and only if `roots [i]` is in fact real. To save memory space, only the first element of each complex-conjugate pair needs to be set, that is, the one with `conjugates [i] >= i`. Internally, the functions work with linear or quadratic real polynomials obtained by regrouping the roots. For a numerical example, see the following function description.

`void mpfrcx_reconstruct_from_roots` (*mpfrx_ptr* `h`, *mpc_t** `roots`, *int** `conjugates`, *int* `no`) [Function]

Compute the polynomial *h* of degree *no* from its roots given in *roots* and paired, as seen above, through *conjugates*.

For instance, the function may be called with `no == 3`, `roots == { (1.2599 0), (-0.6299 1.0911), NULL }` and `conjugates == { 1, 3, 2 }` ; it then computes the result X^3-2 (modulo some rounding errors) as the product of the linear real polynomial $X - 1.2599$ and the quadratic real polynomial $X^2 + 1.2599X + 1.5874$.

`void mpfrcx_subproducttree_from_roots` (*mpfrx_ptr* `h`, *mpc_t** `roots`, *int** `conjugates`, *int* `no`) [Function]

The function initialises the subproducttree to the correct size, which depends not only on *no*, but on the exact number n_1 of real roots and n_2 of pairs of complex-conjugate roots. The tree has $n_1 + n_2$ leaves, with n_1 linear and n_2 quadratic polynomials. The function then computes the subproduct tree, with a polynomial of degree *no*$= n_1 + 2n_2$ at its root.

`void mpfrcx_hecke_from_roots` (*mpfrx_ptr* `h`, *mpfrx_tree_srcptr* `subproducts`, *mpc_t* *`vals`, *mpc_t* *`roots`, *int* *`conjugates`) [Function]

The function works like `mpcx_hecke_from_roots`, returning a real polynomial in *h*, assuming that *vals* is an array of real values and pairs of complex-conjugate values corresponding to the roots in *roots*, paired according to the same array *conjugates*. It is necessary to pass *roots* again for constructing the linear real interpolation polynomial for a pair of complex-conjugate values without factoring the corresponding quadratic polynomial on a leaf of *subproducts*.

`void mpfrcx_product_and_hecke_from_roots` (*mpfrx_t* *`rop`, *mpc_t* **`vals`, *int* *`conjugates`, *int* `no_pols`, *int* `no_factors`) [Function]

The function works exactly like `mpcx_product_and_hecke_from_roots`, except that the result is known to be real since all elements of *vals* are paired up into complex-conjugate pairs according to *conjugates*.

5.10 Functions for Galois Field Towers

The functions in this section decompose a solvable Galois number field extension into a tower of extensions, following *Andreas Enge and François Morain: Fast Decomposition of Polynomials with Known Galois Group*. They have been written in the context of complex multiplication of elliptic curves, for either Hilbert or ray class fields of imaginary-quadratic number fields, or their real subfields, but could be applied in other contexts; abelian extensions of the rationals come to mind, but in principle also Galois extensions of higher degree number fields can be handled.

The general setting is as follows: Let L/Q, where Q is the field of rational numbers, be a Galois number field with Galois group $G = (g_0, ..., g_{h-1})$. We assume that a fixed complex embedding of L has been chosen, and that L is given by the corresponding embedding of a generating element x and the action of the Galois group on this element; in other words, by an ordered list $(x_0, ..., x_{h-1})$ of complex numbers such that $x_i = x^{g_i}$. Moreover, we assume the knowledge of a normal series for G, that is, $1 = H_{l-1} < H_{l-2} < ... < H_0 < H_{-1} = G$ such that each subgroup H_i is normal in H_{i-1}. Let L_i be the fixed field of H_i, such that $L_{l-1} = L$ and $L_{-1} = Q$. Then we obtain a Galois tower in which L/L_i has group H_i, the extension L_i/Q has group G/H_i, and the extension L_i/L_{i-1} has group H_{i-1}/H_i. For instance, in the abelian case, such a normal series exists such that each quotient H_{i-1}/H_i is cyclic of prime order. In the case of a ray class field L over an imaginary-quadratic number field K, the Galois group G is a semi-direct product of the (abelian) ray class group and the group of order 2 generated by the complex conjugation automorphism. One may then choose all the H_i down to H_0 such that they do not contain complex conjugation, that is, as subgroups of the ray class group; then $L_0 = K$, so that alongside the extension L/Q, we have also decomposed the Galois extension L/K.

For the decomposition to work, we furthermore assume an ordering of the Galois group elements $(g_0, ..., g_{h-1})$ (or, more precisely, of the complex embeddings $(x_0, ..., x_{h-1})$ this ordering induces) that is compatible with the decomposition series: $g_0 = \mathrm{id}$, and the first $|H_{i-1}|$ entries are the concatenation of the $|H_{i-1}|/|H_i|$ cosets of length $|H_i|$ of H_i in H_{i-1}. (This implies that in particular the first $|H_i|$ entries contain H_i itself.)

Galois field towers are stored in variables of type `mpcx_tower_t`; again, pointers to modifiable or unmodifiable towers exist, of types `mpcx_tower_ptr` and `mpcx_tower_srcptr`, respectively. The type is defined as a (one-dimensional array of a) C `struct` containing the following fields:

```
typedef struct {
   int levels;
   int* d;
   int deg;
   mpcx_t** W;
}
```

Here, `levels` is the length of the normal series (called l above). The array entry `d[i]` stores the relative degree of L_i/L_{i-1}. The variable `deg` stores the total degree of the extension L/Q (called h above), which is also the product of all the `d[i]`. Finally `W[i]` for `i` from `1` up to `levels-1` holds (after computation) the relative minimal polynomial of a generator of L_i/L_{i-1}, given by its Hecke representation. Otherwise said, if the absolute extension L_{i-1}/Q has the minimal polynomial V_{i-1} in the variable X_{i-1}, then `W[i]` stores the minimal polynomial of X_i, multiplied by the derivative V'_{i-1}. It is thus a (non-monic, with leading coefficient V'_{i-1}) polynomial of degree `d[i]` in the variable X_i, the coefficients of which are polynomials in the variable X_{i-1}, stored in `W[i][0]` up to `W[i][d[i]]`.

To be consistent with this, the variable `W[0]` would have to hold V_0 as an array of `d[0]` constant polynomials in the variable X_0; instead, we chose to use an array of length 1 for `W[0]` and to let its unique entry `W[0][0]` directly hold V_0.

`void mpcx_tower_init` (*mpcx_tower_ptr* `twr`, *int* `l`, *int** `d`, *mpfr_prec_t* `prec`) [Function]

Initialises *twr* to hold a Galois tower, where *l* as above is the length of the Galois group decomposition, *d* the sequence of relative degress as explained above, and *prec* is the common precision used for all subsequent computations.

`void mpcx_tower_clear` (*mpcx_tower_ptr* `twr`) [Function]

Clears *twr* and all its components.

`void mpcx_tower_decomposition` (*mpcx_tower_ptr* `twr`, *mpc_t* *`roots`) [Function]

Given an initialised Galois tower *twr*, and a generating element of the extension L/Q together with its conjugates under the Galois group in an order compatible with the normal series as explained above in *roots*, computes absolute and relative extensions corresponding to the subfield tower and stores them in *twr*. The precisions of the elements in *roots* should match the precision with which *twr* has been initialised.

The function chooses as generators "small" elementary-symmetric expressions under the action of the relative Galois groups; for instance, if possible the relative trace is preferred, then sums of products of two roots, and so on. The question whether an element generates a subfield is answered on a heuristic basis, since equality of elements cannot be rigorously determined in the presence of rounding errors; so there is a very small risk of the function not finding a generator of a subfield.

Notice that the defining polynomials of all field extensions should have rational coefficients (or even integral rational coefficients if the generating element in `roots[0]` is an algebraic integer), but that the identification is not carried out by the function, and that all polynomials are stored with complex coefficients. So this function may in fact also be called when the base field L_{-1} is not the field of rational numbers, as long as a suitable function is called afterwards that obtains the symbolic expressions in L_{-1} of the coefficients of the generated polynomials from their image under the fixed complex embedding of L, which also induces a fixed embedding of L_{-1}.

The following variants of the data structures and functions treat a generalisation to an important non-Galois case, that of the real subfields of class fields of imaginary-quadratic number fields. These fields do not form Galois towers, but being the "projection" under complex conjugation, they behave very similarly to extensions of Q that have subgroups of the class group as Galois groups. The mathematical and algorithmic details are described in *Andreas Enge and François Morain: Fast Decomposition of Polynomials with Known Galois Group*.

Roughly speaking, all occurring polynomials are real, and their roots split into real roots and complex-conjugate pairs of complex roots. Since comparison of floating point numbers with rounding errors is a thorny problem, the user is expected to provide the functions with an indication of which roots are paired up; in the targeted application, this information can generally be derived symbolically from the class group.

Such field towers are stored in variables of type `mpfrx_tower_t`, with pointers `mpfrx_ptr` to modifiable and `mpfrx_srcptr` to unmodifiable towers. The type is defined as a (one-dimensional array of a) C `struct` containing the following fields:

```
typedef struct {
   int levels;
   int* d;
   int deg;
   mpfrx_t** W;
}
```

The only difference to `mpcx_tower_t` is that the polynomials are real instead of complex.

`void mpfrx_tower_init` (*mpfrx_tower_ptr* `twr`, *int* `l`, *int** `d`, *mpfr_prec_t* `prec`) [Function]

`void mpfrx_tower_clear` (*mpfrx_tower_ptr* `twr`) [Function]

These functions behave exactly like their complex counterparts.

`void` **`mpfrcx_tower_decomposition`** (*mpfrx_tower_ptr* `twr`, *mpc_t* `*roots`, *int* `*conjugates`) [Function]

The function behaves exactly like its complex counterpart `mpcx_tower_decomposition`. Together with the list *roots* of complex roots of the class polynomial, in an order compatible with the tower field structure, it expects an array *conjugates* that designates the pairing of the roots: `conjugates [i] == j` if and only if `roots [j]` is the complex conjugate of `roots [i]`. In particular, `conjugates [i] == i` if and only if `roots [i]` is in fact real. To save memory space, only the first element of each complex-conjugate pair needs to be set, that is, the one with `conjugates [i] >= i`.

The prefix `mpfrcx` of the function name indicates the "mixed" nature of the operation: While the results are real, the input values are still of complex type (reflecting that the number fields L_i all have at least one real embedding, but need not be totally real).

Contributors

The main developer of the MPFRCX library is Andreas Enge. The functions for Galois field towers are co-authored by Jared Asuncion.

References

- Torbjörn Granlund et al. `gmp` – GNU multiprecision library. Version 5.0.2, `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 3.1.0, `http://www.mpfr.org/`
- Andreas Enge, Mickaël Gastineau, Philippe Théveny and Paul Zimmermann. `mpc` – A library for multiprecision complex arithmetic with exact rounding. Version 0.9, `http://mpc.multiprecision.org/`
- Andreas Enge and François Morain. Fast Decomposition of Polynomials with Known Galois Group. In Marc Fossorier, Tom Høholdt and Alain Poli (eds.): Applied Algebra, Algebraic Algorithms and Error-Correcting Codes — AAECC-15, vol. 2643 of Lecture Notes in Computer Science, Springer-Verlag, Berlin 2003, pp. 254–264 Extended version: `https://www.math.u-bordeaux.fr/~aenge/papers/galois_long.ps.gz`

Concept Index

A

Access Functions . . . 8
Assignment Functions . . . 7

B

Basic Polynomial Arithmetic . . . 9

C

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

F

Field tower . . . 13
Functions . . . 7

G

Galois field . . . 13

H

Higher Level Functions . . . 10

I

I/O functions . . . 9
Initialisation Functions . . . 7
Input functions . . . 9
Installation . . . 3

L

`libmpfrcx.a` . . . 6

M

Miscellaneous Functions . . . 8
`mpfrcx.h` . . . 6

O

Output functions . . . 9

R

Real polynomial . . . 6
Reporting bugs . . . 5

S

Subproduct tree . . . 10

T

Tree Based Functions . . . 10

Function 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 8
int mpfrx_get_deg 8
int mpfrx_get_prec 8

M

mpcx_add 9
mpcx_clear 7
mpcx_cmp 8
mpcx_derive 10
mpcx_eval 10
mpcx_get_coeff 8
mpcx_hecke 11
mpcx_hecke_from_roots 12
mpcx_init 7
mpcx_init_set 7
mpcx_mul 9
mpcx_mul_c 9
mpcx_mul_fr 9
mpcx_mul_si 9
mpcx_mul_ui 9
mpcx_mul_x 10
mpcx_multieval 12
mpcx_neg 9
mpcx_out_str 9
mpcx_product_and_hecke 11
mpcx_product_and_hecke_from_roots 12
mpcx_realloc 7
mpcx_reconstruct 11
mpcx_reconstruct_from_roots 12
mpcx_rem 10
mpcx_root 10
mpcx_set 7
mpcx_set_coeff 8
mpcx_set_deg 8
mpcx_set_frx 7
mpcx_set_prec 8
mpcx_si_sub 9
mpcx_sub 9
mpcx_subproducttree 11
mpcx_subproducttree_from_roots 12
mpcx_swap 7
mpcx_t 6
mpcx_tower_clear 14
mpcx_tower_decomposition 15
mpcx_tower_init 14
mpcx_tree_clear 11
mpcx_tree_get_root 11
mpcx_tree_init 10
mpcx_urandom 8
mpfrcx_get_version 9
mpfrcx_hecke_from_roots 13
mpfrcx_imag 10
mpfrcx_product_and_hecke_from_roots 13
mpfrcx_real 10
mpfrcx_reconstruct_from_roots 13
mpfrcx_subproducttree_from_roots 13
mpfrcx_tower_decomposition 16
mpfrx_add 9
mpfrx_clear 7
mpfrx_cmp 8
mpfrx_derive 10
mpfrx_eval 10
mpfrx_get_coeff 8
mpfrx_hecke 11
mpfrx_hecke_from_roots 12
mpfrx_init 7
mpfrx_init_set 7
mpfrx_mul 9
mpfrx_mul_fr 9
mpfrx_mul_si 9
mpfrx_mul_ui 9
mpfrx_mul_x 10
mpfrx_multieval 12
mpfrx_neg 9
mpfrx_out_str 9
mpfrx_product_and_hecke 11
mpfrx_product_and_hecke_from_roots 12
mpfrx_realloc 7
mpfrx_reconstruct 11
mpfrx_reconstruct_from_roots 12
mpfrx_rem 10
mpfrx_root 10
mpfrx_set 7
mpfrx_set_coeff 8
mpfrx_set_deg 8
mpfrx_set_prec 8
mpfrx_si_sub 9
mpfrx_sub 9
mpfrx_subproducttree 11
mpfrx_subproducttree_from_roots 12
mpfrx_swap 7
mpfrx_t 6
mpfrx_tower_clear 15
mpfrx_tower_init 15
mpfrx_tree_clear 11
mpfrx_tree_get_root 11
mpfrx_tree_init 10
mpfrx_urandom 8

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

   The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its

Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled

"Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

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 8
- 5.5 Comparison and Miscellaneous Functions 8
- 5.6 Input and Output Functions 9
- 5.7 Basic Polynomial Arithmetic 9
- 5.8 Higher Level Functions 10
- 5.9 Tree Based Functions 10
- 5.10 Functions for Galois Field Towers 13

Contributors 17

References 18

Concept Index 19

Function Index 20

Appendix A GNU Free Documentation License 21