CM

Complex Multiplication
Version 0.3
March 2016

Andreas Enge

Table of Contents

1 Introduction

The CM software implements the construction of ring class fields of imaginary quadratic number fields and of elliptic curves with complex multiplication via floating point approximations. It consists of libraries that can be called from within a C program and of executable command line applications. For the implemented algorithms, see [En09a], page 8.

Given an imaginary quadratic discriminant D<0, the associated ring class field is generated by the values of modular functions in special arguments taken from the quadratic field Q(sqrt D); these values are called *singular values* or *class invariants*. Depending on D, different modular functions need to be chosen; we call the suitable ones *class functions*. CM implements (to a greater or lesser extent) all major class invariants described in the literature.

Licence

2 Installation

2.1 Instructions

CM relies on a number of external libraries, which need to be installed before compiling CM: GNU MP (see [Gr09], page 8, version 4.3.2 or higher), GNU MPFR (see [HaLePeZi09], page 8, version 3.0.0 or higher), MPC (see [EnThZi09], page 8, version 1.0.0 or higher), MPFRCX (see [En09b], page 8, version 0.4 or higher) and PARI (see [Pa10], page 8, version 2.7.0 or higher, compiled with GMP as the arithmetic kernel). Compilation of CM needs a standards compliant C compiler (preferably GCC).

These are the steps needed to install CM, provided that the required libraries are installed in standard locations:

1. '`tar xzf cm-0.3.tar.gz`'
2. '`cd cm-0.3`'
3. '`./configure`'
4. '`make`'

   This compiles CM.
5. '`make check`'

   This performs a few tests to check that CM has been built correctly.

   If you get error messages, please report them to the mailing list (cf. Section 2.4 [Mailing list], page 3).
6. '`make install`'

   This copies the executable applications into the directory `/usr/local/bin`, the header files into `/usr/local/include`, the library files into `/usr/local/lib`, the data files into subdirectories of `/usr/local/share/cm` (see Section 2.3 [Data], page 3) and the file `cm.info` into `/usr/local/share/info`.

   It is possible to pass the option '`--prefix=/my/install/directory`' to the '`./configure`' step above, so that all files go into subdirectories of `/my/install/directory` instead of `/usr/local`.

If auxiliary libraries are to be found in non-standard locations, these need to be passed in the '`./configure`' step above by adding parameters

- '`--with-gmp=<gmp_install_dir>`',
- '`--with-mpfr=<mpfr_install_dir>`',
- '`--with-mpc=<mpc_install_dir>`',
- '`--with-mpfrcx=<mpfrcx_install_dir>`' and
- '`--with-pari=<pari_install_dir>`'.

For an exhaustive list of configuration parameters, execute '`./configure --help`'.

2.2 Documentation

Besides the texinfo documentation obtained by a simple invocation of '`make`', the commands '`make dvi`', '`make ps`', '`make pdf`' and '`make html`' create the documentation in the corresponding formats.

2.3 Data

Some parameterised families of class functions need additional data (namely, *modular polynomials*), which depend on the parameter value, to deduce the equation of an elliptic curve from the class polynomial. A few modular polynomials for double eta quotients are provided and stored in `/usr/local/share/cm/df` (or the subdirectory `share/cm/df` of the installation directory provided with the '`--prefix`' configuration option, respectively); those for Atkin functions can be found in `/usr/local/share/cm/af`, and so on.

An infinite amount of data is needed to handle all possible discriminants with a given family of class functions, and already covering all moderately sized discriminants would require gigabytes of data. So only a very small sample of modular polynomials is currently distributed; if you need more, please write to the mailing list (cf. Section 2.4 [Mailing list], page 3).

2.4 Mailing list

Bug reports should go to the mailing list `cme-discuss@lists.gforge.inria.fr`, which, as its name indicates, may also be used for discussions around the software.

3 Applications

At the time being, CM comes with two sample applications, '`cm`' and '`classpol`'. Both take the same command line arguments. Being given a negative discriminant D, passed to the program through the parameter '`-d`' directly followed (without separating space) by the absolute value `|D|` of the discriminant, '`cm`' computes and outputs a cryptographically suitable elliptic curve with complex multiplication by D. The program chooses a 200 bit prime p and computes a curve E : Y^2 = X^3 + a*X + b over Fp of "almost prime" cardinality, that is, a cardinality that is either itself prime or twice or four times a prime depending on D (prime cardinalities may only be reached for D = 5 (mod 8)). On the other hand, '`classpol`' computes and outputs only the class polynomial without an associated elliptic curve.

In normal operation, the programs output only the result of their computations. For more details and progress reports, use the command line parameter '`-v`'.

By default, both programs use the j-function as class function, which works for arbitrary discriminants, but is a poor choice for efficiency reasons since it generates very large class polynomials. It is recommended to specify alternative class invariants, which, for `|D|` tending to infinity, allow to gain a constant factor in the size of the computed objects. A class invariant is selected by the parameter '`-i`', directly followed (without space) by a word characterising a class function or a parameterised family of such functions; in the latter case, an optimal parameter value is computed by the program. The following class functions are implemented:

- j: The default choice.
- doubleeta: Double eta quotients whose levels are the product of two (not necessarily distinct) primes p1 and p2. The class polynomial is asymptotically smaller by a *height factor* of 12*psi(p1*p2) / ((p1-1)*(p2-1)), where psi(p1*p2) = (p1+1)*(p2+1) if p1 and p2 are distinct and psi(p1^2) = p1*(p1+1). This factor is at least 12 and may reach 37 in the optimal case p1=2 and p2=73. Since CM has originally been written to test these class functions, the code is particularly optimised for them. It is always possible to find a suitable parameter combination p1 and p2, but the required data may not be available to deduce the elliptic curve; in this case, do not hesitate to make a request on the mailing list (cf. Section 2.3 [Data], page 3).
- gamma3, gamma2: Weber class functions with height factors 2 and 3, respectively; of little practical interest.
- weber: Class functions derived from Weber's functions, with height factors between 6 and 72.
- atkin: Optimal functions for X0+(N) with N=71 (height factor 36), N=131 (height factor 33), N=59 (height factor 30) or N=47 (height factor 24). While yielding a nice gain in the size of the computed objects, these functions are slow to evaluate numerically and thus not really attractive for the floating point approach implemented in CM. The implementation has not been optimised and should rather be seen as a proof of concept.
- simpleeta: Simple eta quotients; currently not working properly since the determination of optimal parameters is still work in progress.
- multieta: Multiple eta quotients; currently not working properly since the determination of optimal parameters is still work in progress.

Example invocations

- '`cm -d108708`'

  Computes a curve with complex multiplication by D=-108708 over a prime field of 200 bit and with cardinality twice a prime of 199 bit.

- '`cm -d108708 -idoubleeta`'

  Computes a curve with complex multiplication by D=-108708 over the same prime field and of the same cardinality, but the class polynomial computed using a double eta quotient has coefficients of 151 bit instead of 5874 bit. The optimal parameter combination p1=2 and p2=193 is determined automatically.

- '`cm -d2717700 -idoubleeta`'

  Computes a curve with complex multiplication by D=-2717700=-25*108708. Since this corresponds to the non-maximal order of conductor 5 in the same quadratic field, the finite prime field and the cardinality are the same. However, the degree as well as the coefficients of the class polynomial are six times bigger.

- '`classpol -d108708 -idoubleeta -v`'

  Computes the class polynomial for the discriminant -108708 using a double eta quotient, providing details of the computation.

If you experience difficulties creating your own application using the libraries described in Chapter 4 [Libraries], page 6, or are missing functionalities, please do not hesitate to use the mailing list, cf. Section 2.4 [Mailing list], page 3.

4 Libraries

The code of CM is organised in libraries to make the different functions accessible to external applications. With a few exceptions, the names of all publicly accessible, non-static functions and types start by `cm_` to create a name space proper to CM.

4.1 `cm_class`

The library `cm_class` contains the main code needed for computing class polynomials and elliptic curves with prescribed complex multiplication. For the time being, only the interface needed to implement the applications of Chapter 3 [Applications], page 4, is made public in `cm_class.h`. Further internal functions, such as the computation of the algebraic conjugates of a class invariant, can be found in `cm_class-impl.h` and may be made public in the future.

`int_cl_t` [Type]
This signed 64 bit integer type is used for discriminants.

`cm_class_t` [Type]
This structural type holds information related to a class polynomial, from the discriminant and class function set at initialisation to computed data such as class numbers and the class polynomial.

`void cm_class_init` (*cm_class_t* ***c***, *int_cl_t* ***d***, *char* ***inv***, *bool* ***verbose***) [Function]
Sets the discriminant of *c* to *disc* and the class function to *inv*. The variable *inv* may take one of the predefined constants `CM_INVARIANT_J`, `CM_INVARIANT_DOUBLEETA`, `CM_INVARIANT_GAMMA3`, `CM_INVARIANT_GAMMA2`, `CM_INVARIANT_WEBER`, `CM_INVARIANT_SIMPLEETA`, `CM_INVARIANT_MULTIETA` or `CM_INVARIANT_ATKIN`, corresponding to the class functions enumerated in Chapter 3 [Applications], page 4. For a parameterised family of class functions, a suitable parameter value is computed if it exists; otherwise, the program execution is aborted. Moreover, the class number is computed and space is allocated accordingly for the class polynomial.

If *verbose* is set to `true`, some information is printed on screen during execution.

`void cm_class_compute_minpoly` (*cm_class_t* ***c***, *bool* ***checkpoints***, *bool* ***write***, *bool* ***print***, *bool* ***verbose***) [Function]
Takes a previously initialised `cm_class_t` object *c* and computes the corresponding class polynomial. If *print* is `true`, the polynomial is printed on screen, if *write* is `true`, it is written to disc into the subdirectory `cp`.

The variable *checkpoints* is only relevant for record computations; if set to `true`, intermediate results are written to disk and read in again if the program execution is interrupted and resumed later.

If *verbose* is set to `true`, some information is printed on screen during execution.

`void cm_class_clear` (*cm_class_t* ***c***) [Function]
This is the counterpart to `cm_class_init` and should be called once the class polynomial is not needed any more to free the allocated space.

`void cm_curve_compute_curve` (*int_cl_t* ***d***, *char* ***inv***, *int* ***fieldsize***, *const char** ***modpoldir***, *bool* ***readwrite***, *bool* ***print***, *bool* ***verbose***) [Function]
Computes a cryptographically suitable curve with complex multiplication by *d* over some prime field of *fieldsize* bits, using the class function designated by *inv*.

If *readwrite* is set to `true`, the function tries to read the class polynomial from a file; if the file is not present, the class polynomial is computed and written to a file.

If *print* is set to `true`, the j-invariant and coefficients a and b of the curve are printed on screen, if *verbose* is set to `true`, additional progress information is printed.

The variable *modpoldir* holds the base directory of the modular polynomials needed to derive elliptic curves from roots of class polynomials; the modular polynomials themelves are stored in subdirectories `df`, `af` and so on of the base directory. In the standard installation, this is `/usr/local/share/cm`.

4.2 cm_common

This library is split off from `cm_class` and contains functions that can be seen as internal to the class polynomial computation, but that can also be useful in a broader context (notably, the computation of modular polynomials).

4.3 mpfpx

This library is used for low degree polynomials over prime fields, the characteristic of which is stored in a global variable. It is in a very rudimentary state.

References

- [En09a] Andreas Enge. The complexity of class polynomial computation via floating point approximations. *Mathematics of Computation* 78 (266), 2009, pp. 1089–1107
- [En09b] Andreas Enge. `mpfrcx` – A library for the arithmetic of univariate polynomials over arbitrary precision real or complex numbers. INRIA. Version 0.3, 2010, http://mpfrcx.multiprecision.org/
- [EnThZi09] Andreas Enge, Philippe Théveny and Paul Zimmermann. `mpc` – A library for multiprecision complex arithmetic with exact rounding. INRIA. Version 0.8, 2009, http://mpc.multiprecision.org/
- [Gr09] Torbjörn Granlund et al. `gmp` – GNU multiprecision library. Version 4.3.1, 2009, http://gmplib.org/
- [HaLePeZi09] Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Paul Zimmermann et al. `mpfr` – A library for multiple-precision floating-point computations with exact rounding. Version 2.4.1, 2009, http://www.mpfr.org/
- [Pa10] The PARI group. `PARI/GP`. Version 2.3.5, 2010, http://pari.math.u-bordeaux.fr/

Index