CMAT DOCUMENTATION

K. R. MATTHEWS

5th July 2010

1 cmat installation

`cmat` is an exact arithmetic calculator program, written in ANSI C. It is designed to perform many of the standard arithmetical operations that can be carried out exactly, without approximations, on matrices and polynomials whose coefficients are either rational numbers, complex rational numbers, or elements of $\mathbf{Z}_p$ – the finite field of p elements, where p is a prime less than `R0`$= 2^{16} = 65536$.

The file `config.h` contains default paths which are set up for the Linux version on the assumption that the four executable programs and the `*.dat` files live in the current directory "."

To change the number `M0` of stored global variables of each arithmetical type, the file `integer.h` should be edited and the programs recompiled.

2 Documentation

1. **Files**. There are various `*.h` files:

   (a) `integer.h` contains declarations of various arithmetic data structures:

   - (i) MPI, MPR, MPCI, MPCR;
   - (ii) MPMATI, MPMATR, MPMATR, MPMATCR, MPMATm;
   - (iii) POLYI, POLYR, POLYR, POLYCR, POLYm;
   - (iv) MPMATPI, MPMATPR, MPMATPCI, MPMATPCR, MPMATPm.

   as well as definitions of constants `R0, T0, M0, Z0, Y0, BUFLEN` and some abbreviations.

   (b) `cmat.h` contains declarations of functions defined in files `util.c`, `spawn.c` (Linux) and `menu.c`, as well as external declarations of variables used in the files.

   (c) `config.h` contains system dependent #defines.

   (d) `primes.h` contains an array of global variables corresponding to the first 2048 primes, together with a two–dimensional global array which provides a table–lookup for the inverses mod m of all residues a, $1 \leq a \leq m$, $(a, m) = 1$, where $m \leq 99$.

(e) `fun.h` contains declarations of all functions in all .c files residing in all three directories.

There are various `*.dat` files:

(a) `info.dat` contains information about `cmat`;

(b) `menuR.dat` contains the menus for `cmatr`;

(c) `menuCR.dat` contains the menus for `cmatcr`;

(d) `menum.dat` contains the menus for `cmatm`.

Under Linux, `.cmatrc` sets up the directory paths: `cmat=.` for the executable programs and `datpath=.` for the `*.dat` files.

If the variable `DEBUG` in integer.h is defined, an overall count of the number of bytes remaining as unfreed memory is printed on exiting the component programs `cmatr`, `cmatcr` and `cmatm`. Care has been taken to ensure this number should always be zero. If not, then this indicates that space was not freed at some stage in at least one of the algorithms; consequently an unwelcome buildup of used memory could result if the program is run repeatedly.

`parse.y` is a yacc file which produces a file parse.c for use in parsing simple arithmetic expressions suitable for use as formulae for the (i, j)th element of a matrix. This was initially used to produce parse.c.

2. **Makefiles**. There are two types: Makefile (Linux) and `Makefile_Windows` (Windows).

To compile under Linux, type `make -f Makefile all`, while to compile under cygwyn, type `nmake -f Makefile_Windows all`. This assumes one has `nmake.exe` present and the `lib` facility.

3. **The arithmetic data types**.

The basic structure is the MPI (multiple precision integer) M which has three components: an unsigned integer M.D, a variable array of M.D+1 unsigned integers:

$$\text{M.V}[0], \ldots, \text{M.V}[\texttt{M.D}],$$

where each of these array elements must be less than $\texttt{R0} = 2^{16} = 65536$. There is a third structure component M.S, which records the sign. If m is the integer represented by the MPI M, there is a corresponding representation of m to base `R0`:

$$m = \text{M.S} \sum_{i=0}^{\text{M.D}} \text{M.V}[i](\texttt{R0})^i.$$

A rational number is represented by a structure M called an MPR. Here M.N and M.D are pointers to MPI's representing the numerator and denominator, respectively. The denominator must be positive and the numerator and denominator relatively prime.

A complex rational number m is represented by a structure M called an MPCR. Here M.R and M.I are pointers to MPR's representing the real and imaginary parts of m.

Polynomials are represented by linked lists along the lines of Scientific Pascal by H. Flanders:

POLYI, POLYR, POLYCI, POLYCR, POLYm,

each terminated by the NULL pointer. Only the non–zero terms of the polynomial are recorded. The zero polynomial is represented by the NULL pointer.

Matrices of scalars are represented by structures:

MPMATI, MPMATR, MPMATCI, MPMATCR, MPMATm.

If M is such a structure, the components M.R and M.C record the number of rows and columns, while the component M.V corresponds to a two– dimensional array of scalars of variable size.

There are also correspondingly matrices with polynomial elements:

MPMATPI, MPMATPR, MPMATPCI, MPMATPCR, MPMATPm.

It is important to remember that for the polynomial structure POLYm and matrices MPMATm and MPMATPm, the coefficients are always understood to be unsigned integers less than a prime p (the modulus), with $p <$ `R0`. It is the user's responsibility to keep track of the various moduli used and to perform calculations with arithmetic objects only having the same prime modulus.

4. **Other Global Variables**.

Up to `M0` $= 30$ (say) objects of each type can be created and stored as global variables:

1. rational numbers: R[0], …, R[`M0` $-$ 1];
2. matrices with rational coefficients:
   RM[0], …, RM[`M0` $-$ 1];

3. polynomials with rational coefficients:
   $\mathrm{PR}[0], \ldots, \mathrm{PR}[\texttt{MO} - 1]$;

4. matrices with polynomial rational coefficients: $\mathrm{PRM}[0], \ldots, \mathrm{PRM}[\texttt{MO} - 1]$;

5. complex rational numbers: $\mathrm{CR}[0], \ldots, \mathrm{CR}[\texttt{MO} - 1]$;

6. matrices with complex rational coefficients:
   $\mathrm{CRM}[0], \ldots, \mathrm{CRM}[\texttt{MO} - 1]$;

7. polynomials with complex rational coefficients:
   $\mathrm{PCR}[0], \ldots, \mathrm{PCR}[\texttt{MO} - 1]$;

8. matrices with polynomial complex rational coefficients:
   $\mathrm{PCRM}[0], \ldots, \mathrm{PCRM}[\texttt{MO} - 1]$;

9. matrices with coefficients (mod p): $\mathrm{mM}[0], \ldots, \mathrm{mM}[\texttt{MO} - 1]$;

10. polynomials with coefficients (mod p): $\mathrm{Pm}[0], \ldots, \mathrm{Pm}[\texttt{MO} - 1]$;

11. matrices with polynomial coefficients (mod p):
    $\mathrm{PmM}[0], \ldots, \mathrm{PmM}[\texttt{MO} - 1]$;

These arrays always contain the appropriate zero object until replaced by either an object entered by the user or one resulting from a calculation.

5. **Saving for future sessions**.

When each of the programs `cmatr`, `cmatcr`, `cmatm` is run, the following files are respectively opened in the user's current directory:

(i) cR, cPR, cRM, cPRM;

(ii) cCR, cPCR, cCRM, cPCRM;

(iii) cPm, cmM, cPmM.

If these files do not already exist, they are created, with contents corresponding to appropriate zero arithmetic objects. At the end of each computing session, the user is given the option of saving data for further re–use in future sessions.

6. **Memory bank**. We use a memory banking algorithm implemented by Peter Adams. Previously, MPI's and other data structures were malloced and freed whenever they were required, which proved to be very time consuming. About 50% of the execution time was spent in calls to malloc() and free().

Now, rather than freeing when we have finished using them, we place them on a list of free MPI's (and corresponding lists for other data structures). When another MPI is required, if there is one of an appropriate size on the list, it can be used, rather than calling malloc().

Any suggestions for improvement, as well as uncovering of bugs, should be reported to the author,
Keith Matthews
http://www.numbertheory.org/keith.html

5th July 2010