

**FM**

---

The Fourier-Motzkin Library  
Edition 0.2, for FM 0.5.0  
November 1st 2008

**Louis-Noël Pouchet**

---

This manual is dedicated to FM version 0.5.0, a library for manipulating rational and integer polyhedra.

Copyright © 2006-2008 Louis-Noël Pouchet.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 published by the Free Software Foundation. To receive a copy of the GNU Free Documentation License, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

# Table of Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b> .....               | <b>1</b> |
| 1.1      | Installation of FM .....                | 1        |
| <b>2</b> | <b>FM Software</b> .....                | <b>3</b> |
| 2.1      | Input file design .....                 | 3        |
| 2.2      | Output file design .....                | 3        |
| 2.3      | Running the binaries .....              | 3        |
| <b>3</b> | <b>FM Library</b> .....                 | <b>5</b> |
| 3.1      | A first solver example .....            | 5        |
| 3.2      | A first lexmin example .....            | 5        |
| 3.3      | Option flags for the solver .....       | 5        |
| 3.4      | Available solver methods .....          | 6        |
| 3.5      | Scaling with automatic compaction ..... | 7        |
| 3.6      | PIP related functions .....             | 7        |
| 3.7      | Integer types .....                     | 7        |
| <b>4</b> | <b>References</b> .....                 | <b>9</b> |



# 1 Introduction

FM is a library dedicated to manipulating Q-polyhedra, and especially those representing the projection of a given system of inequalities. The projection is computed with an improved version of the Fourier-Motzkin algorithm. The library offers features such as:

- A redundancy-aware C implementation of the Fourier-Motzkin projection algorithm.
- A lexicographic min/max computation (for Q and Z polyhedra).
- A lot of convenience functions to manipulate Q-polyhedra.

**Communication:** one group is available for subscription [fm-announce](#), to receive announces about releases of the software. Please contact the author directly for any question.

**API Documentation:** There is a Doxygen documentation of the API available in `doc/html/doc.tar.gz`.

**License:** Starting version 0.5.0, FM is released under the terms of the GNU Lesser General Public License version 3 and later. Before version 0.5.0 FM was released under the terms of the GNU GPL v2 and later license.

## 1.1 Installation of FM

The installation of FM follows the classical scheme. You can specify the path of the optional dependence of FM (PIPLib) to the configure script with `--with-piplib1`.

```
$> tar xzf fm-0.5.0.tar.gz
$> cd fm-0.5.0
$> ./configure --prefix=/path/to/be/installed
$> make
```

The `make check` command can be used to make a test run of FM.

```
$> make check
```

The `make install` command can be used to install the library.

```
$> make install
```

---

<sup>1</sup> The configure script will automatically detect the location of dependencies if `-prefix=path/to/dir` is specified and all dependencies were installed in `path/to/dir`



## 2 FM Software

This chapter describes briefly the input/output of FM binaries.

### 2.1 Input file design

System must be supplied in a PolyLib-like formatted file. It correspond to the `s_fm_system_t` structure used in the library.

As an example:

```
# ----- Sample -----
# Commented lines should start with '#'. Blank lines are ignored.

# Supply here the number of lines and columns
4 4

# Then the matrix should have exactly the same number of lines and
# columns as indicated below. A number can be a rational, with the
# syntax SX/Y (no spaces, S = sign (- or + or nothing), X and Y =
# integer), or an integer. Numbers must be separated with at least one
# space (' ').
#
# Remember in PolyLib format, first number is '0' if the line is equal
# to 0, and is a '1' if the line is greater or equal to 0.

#   x1   x2   1
1   -2   11   3
1    3   -2  -5
1 -1/1  -3  8/2
1    2    0  -3

# ----- ■
```

### 2.2 Output file design

The results are presented as 'balls', the common representation of Fourier-Motzkin projection algorithm. At index 0, you have the positive ( $x_0 \geq \dots$ ) and negative conditions ( $x_0 \leq \dots$ ) for the first variable of the system. It corresponds to the `s_fm_solution_t` structure of the library. An example is given below.

### 2.3 Running the binaries

Then the binary `fm-solver` (in `src/`) can be called:

```
$> src/fm-solver doc/sample.in
At index 0
- Positives:
[1 1 -49/29]
```

```

- Negatives:
[1 -1 53/17]
At index 1
- Positives:
[1 -2/11 1 3/11]
- Negatives:
[1 3/2 -1 -5/2]
[1 -1/3 -1 4/3]

```

it is read:

```

x0 >= 49/29
x0 <= 53/17 (or -x0 + 53/17 >= 0)
-2/11.x0 + x1 >= -3/11
1/3.x0 + x1 <= 4/3

```

Optionally, a second argument can be given, and the result will be stored in this filename:

```
$> src/fm-solver doc/sample.in sample.out
```

If the system has no solution, (Empty solution) is outputted.

One can desire to compute the lexico-smallest set of values for a system. This function is provided with the binary `fm-minlexico`. Three arguments are to be given: the filename storing the system to solve, a bit (0 or 1) to set if you want rational or integral values, and the value to be used to replace -infinite where the lower bound is -infinite:

```
$> src/fm-solver doc/sample.in 0 -42
[49/29 1/29]
```

If no rational or integral solution exists, [(null)] is outputted.

```
$> src/fm-solver doc/sample.in 1 -42
[(null)]
```

## 3 FM Library

The library provides a set of includes, installed in `$prefix/includes/fm`.

### 3.1 A first solver example

In order to use the solver, one is encouraged to do the following:

```
#include <fm/system.h>
#include <fm/solution.h>
#include <fm/solver.h>

// Declare the input system.
s_fm_system_t* system;
// Allocate and fill the system, or read it from a stream
system = fm_system_read (stream);
// Declare the output solution.
s_fm_solution_t* solution;
// Solve the given system. Use FM_SOLVER_AUTO_SIMPLIFY to help the solver
// to automatically scale
solution = fm_solver (system, FM_SOLVER_FAST | FM_SOLVER_VERBOSE);
// Print the output solution.
fm_solution_print (stream);
// Be clean.
fm_system_free (system);
fm_solution_free (solution);
```

### 3.2 A first lexmin example

In order to get the lexico-smallest set of values, one is encouraged to do the following:

```
#include <fm/system.h>
#include <fm/solution.h>
#include <fm/solver.h>

s_fm_solution_t* solution;
solution = fm_solver (system, FM_SOLVER_FAST);
// lexico_smallest will be a NULL-terminated array.
s_fm_rational_t** lexico_smallest;
lexico_smallest = fm_solver_minlexico (solution, FM_MINLEXICO_INT);
```

### 3.3 Option flags for the solver

- `FM_SOLVER_FAST`: The default solver option that should be passed. Provides simple redundancy elimination.
- `FM_SOLVER_NORMALIZE_EQ`: Converts equalities in the input system to two inequalities. Although this option is optional, the user is encouraged to use it when providing equalities in the input system.
- `FM_SOLVER_DEBUG`: Performs extra checks during computation.

- `FM_SOLVER_SIMPLIFY`: Enable full redundancy elimination, at each step of the projection. Requires PIP and one of the 2 following options.
- `FM_SOLVER_REDREC_DESCENDANT`: Performs the Le Fur “descendant” order for eliminating constraints.
- `FM_SOLVER_REDREC_IRIGOIN`: Performs the Le Fur “Irigoin” order for eliminating constraints. This is the most efficient.
- `FM_SOLVER_AUTO_SIMPLIFY`: Automatically change on the fly the solver options to increase scalability. If the number of constraints exceeds a given threshold, more aggressive redundancy reduction is used.
- `FM_SOLVER_VERBOSE`: Verbose output.

More than one option can be passed at the same time thanks to the `|` operator.

### 3.4 Available solver methods

- `s_fm_solution_t* fm_solver (s_fm_system_t* system, int solver_type)`  
Performs the full projection of the input system.
- `s_fm_solution_t* fm_solver_solution_at (s_fm_system_t* system, int solver_type, unsigned last)`  
Performs the full projection of the input system, and returns only the 1..last first columns.
- `s_fm_solution_t* fm_solver_solution_to (s_fm_system_t* system, int solver_type, unsigned to)`  
Performs a partial projection of the input system of `n` variables: columns `to..n` are projected on columns `1..n-1`.
- `s_fm_rational_t** fm_solver_minlexico(s_fm_solution_t* sol, z_type_t min, int is_integral)`  
Returns the lexmin of the solution set, and `min` if it is -infinite.
- `s_fm_rational_t** fm_solver_maxlexico(s_fm_solution_t* sol, z_type_t max, int is_integral)`  
Returns the lexmax of the solution set, and `max` if it is infinite.
- `void fm_solver_compute_min (s_fm_rational_t** lb, s_fm_list_t* l, s_fm_vector_t* vect, unsigned idx, int is_int)`  
Compute the lower bound `lb` of the variable at index `idx`, given a list `l` of inequalities to respect, and the values of the variable `1..idx-1` stored in the vector `vect`. This function holds only if the full projection has been performed before.
- `void fm_solver_compute_max (s_fm_rational_t** ub, s_fm_list_t* l, s_fm_vector_t* vect, unsigned idx, int is_int)`  
Same for the upper bound `ub`.
- `s_fm_solution_t* fm_solver_linind (s_fm_system_t* A)`  
Extract the replacement vectors for all possible variables, given a system of equalities.
- `int fm_solver_gauss (s_fm_system_t* A)`  
Performs Gaussian elimination on the input system of equalities.

### 3.5 Scaling with automatic compaction

One of the strength of FM is its multi-level redundancy elimination. To avoid inserting a redundant constraint in a `s_fm_solution_t`, the function `fm_solution_add_unique` can be used, provided the inequality has been normalized with `fm_vector_normalize_idx`.

Sometimes, some dimensions of the input system may be fixed to a given value (or equal to a linear combination of other dimensions), while others are standard variables. A significant performance gain can be obtained during the projection if it is performed only on the “variable” dimensions. To silently manipulate this compacted polyhedron, we provide the `s_fm_compsol_t` structure. It can be initialized from a `s_fm_system_t` with the function `fm_compsol_init_sys(s_fm_system_t*)` or from a `s_fm_solution_t` with the function `fm_compsol_init_sol(s_fm_solution_t*)`. The outputted structure has a field `poly` which contains the polyhedron of non-fixed dimensions. The following example illustrates how to leverage this representation to perform projection cleverly:

```
s_fm_system_t* to_solve = fm_system_read (stream);
cs_tosolve = fm_compsol_init_sys (to_solve);
fm_system_free (to_solve);
to_solve = fm_solution_to_system (cs_tosolve->poly);
fm_solution_free (cs_tosolve->poly);
cs_tosolve->poly = fm_solver (to_solve,
                             FM_SOLVER_FAST | FM_SOLVER_AUTO_SIMPLIFY);
solution = fm_compsol_expand (cs_tosolve);
fm_compsol_free (cs_tosolve);
```

Note the out-of-compsol function `fm_compsol_expand` which returns a `s_fm_solution_t*`, as well as the converter `fm_solution_to_system`.

### 3.6 PIP related functions

Some of the redundancy elimination functions of FM requires the use of PIPLib, the Parametric Integer Programming library. Typically, existence of a point can be checked thanks to the `fm_piptools_check_int(s_fm_solution_t* poly, s_fm_vector_t* point)` which realizes a call to PIP.

### 3.7 Integer types

The FM library uses ‘long int’ as default type for integers. If one needs to modify this type, alter `fm/include/common.h` file (change the typedef for `z_type_t`, and update the `STRING_MODIFIER` accordingly).



## 4 References

- [1] Louis-Noël Pouchet. When iterative optimization meets the polyhedral model: one-dimensional date. Master thesis, University of Paris-Sud 11, France, September 2006.
- [2] Louis-Noël Pouchet, Cédric Bastoul, Albert Cohen and Nicolas Vasilache. Iterative optimization in the polyhedral model: Part I, one-dimensional time. In *IEEE/ACM Fifth International Symposium on Code Generation and Optimization (CGO'07)*, pp 144–156, IEEE Computer Society Press, San Jose, California, March 2007.
- [3] Louis-Noël Pouchet, Cédric Bastoul, Albert Cohen and John Cavazos. Iterative optimization in the polyhedral model: Part II, multidimensional time. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'08)*, pp 90–100, ACM Press, Tucson, Arizona, June 2008.

