How to solve multivariate polynomial systems with MSOLVE?

Contents

1 Introduction

MSOLVE is a C library for solving multivariate polynomial systems of equations. It relies on computer algebra, a.k.a. symbolic computation, algorithms to compute *algebraic* representations of the solution set from which many, if not all, informations can be extracted.

Solving polynomial systems with MSOLVE is *global* by contrast to *local* numerical routines. The use of computer algebra methods allow also the user to bypass classical numerical issues encountered by numerical methods for polynomial system solving.

MSOLVE relies mainly on Gröbner bases algorithms (see below for a some basic definitions and properties). It is highly optimized, uses AVX2 vectorization instructions and multi-threading.

It uses the GMP library (handling multi-precision integers) and the FLINT library (handling arithmetics of univariate polynomials).

MSOLVE can be downloaded from

https://msolve.lip6.fr

where binaries (for x86 processors runing Linux operating systems) and the source files are provided.

MSOLVE is designed for 64 bit architectures, with AVX2 instructions.

2 Preliminaries

Polynomial systems arise in a wide range of applications in engineering and computing sciences. What it means to "solve" a polynomial system mainly depends on the application, the *domain* where the coefficients lie and the properties of the solution set.

For instance, in many applications of engineering sciences, the coefficients lie in the *field* of rational numbers $\mathbb{Q}$. Sometimes, these coefficients appear as *floating point* numbers but since floating point arithmetics is not fully convenient to handle reliably non-linear systems, the end-user may coerce these coefficients into rational numbers. There, using continued fractions is quite useful as it allows one to obtain reliable approximations with integers of small *height* (hence small length when represented with bits).

All in all, one ends up with polynomial equations in the polynomial ring $\mathbb{Q}[x_1, \ldots, x_n]$, i.e. the set of polynomials with coefficients in $\mathbb{Q}$ and involving indeterminates $x_1, \ldots, x_n$.

In such applications, the end-user will most of the time expect informations on the solutions with coordinates in the field of real numbers $\mathbb{R}$ (for instance in robotics or biology) or in the field of complex numbers $\mathbb{C}$ (for instance in signal theory).

When the number of solutions such systems in $\mathbb{C}^n$ is finite (one says that such systems have dimension at most zero), then it is well-known that the coordinates of the solutions can be parametrized by an *algebraic number* at which one evaluates a rational fraction. In other words, there exist univariate polynomials $(w, v_1, \ldots, v_n)$ in $\mathbb{Q}[t]$ such that w is square-free and the solution set in $\mathbb{C}^n$ can be written as

$$\left\{ \left(\frac{v_1(\vartheta)}{w'(\vartheta)}, \ldots, \frac{v_n(\vartheta)}{w'(\vartheta)} \right) \quad | \quad w(\vartheta) = 0 \right\}$$

where $w' = \frac{\partial w}{\partial t}$. Note that, with such an encoding (which is called a rational parametrization of the solution set), one can approximate at arbitrary precision the solutions of the input system (by isolate the ones of the univariate polynomial w). Note also that, since w is square-free, w and w' are co-prime and the denominator in the above parametrization can be replaced by a constant.

Remark that the total number of solutions in $\mathbb{C}^n$ equals the degree of w. Hence, when the input system has no complex solution, w is a non-zero constant. Also, the number of real roots to the input system is the number of real roots of w.

Note also that this encoding loses the number of solutions counted with *multiplicities* ; this number is called the *degree* of the input system (which always dominates the number of complex solutions).

For systems in $\mathbb{Q}[x_1, \ldots, x_n]$, MSOLVE is able to decide if they have finitely many complex solutions and, in that case, returns a rational parametrization of the solution set in $\mathbb{C}^n$. It also counts and isolates the real roots of the polynomial w. We refer to Section 4 for details on how MSOLVE encodes such outputs.

When the input system has infinitely many complex solutions, MSOLVE can output a *Gröbner basis* of the *ideal* generated by all input polynomials ; this is a family of polynomials which is obtained by taking appropriate algebraic combinations of the input polynomials and from which one can extract several properties of the solution set, in particular its *dimension* and its *degree* (see Section 5).

For applications in cryptology and/or coding theory, polynomial systems lie in some polynomial ring $\boldsymbol{F}[x_1, \ldots, x_n]$ where $\boldsymbol{F}$ is some finite field. In that situation, MSOLVE only supports prime field $\frac{\mathbb{Z}}{p\mathbb{Z}}$ with $p < 2^{31}$. Denoting by $\overline{\boldsymbol{F}}$ an algebraic closure of $\boldsymbol{F}$, such systems have dimension at most 0 when they have finitely many solutions in $\overline{\boldsymbol{F}}^n$, else they have positive dimension. For systems of dimension at most zero, an above rational parametrization can be computed when the characteristic of the field is large enough.

3 Input file format

Consider the following polynomial system of equations

$$\begin{aligned} x + 2y + 2z - 1 &= 0 \\ x^2 + 2y^2 + 2z^2 - x &= 0 \\ 2xy + 2yz - y &= 0 \end{aligned}$$

in $\mathbb{Q}[x, y, z]$.

In order to solve it with MSOLVE one simply produces a file with the following content

```
x,y,z
0
x+2*y+2*z-1,
x^2+2*y^2+2*z^2-x,
2*x*y+2*y*z-y
```

Hence the structure of input files to MSOLVE is as follows:

1. the first line contains the variables of the input system, separated with a comma ;
2. the second line contains the characteristic of the field over which computations are performed ;

3. the next lines contain polynomials, in expanded form, separated by a comma (except the last one) and with a line break.

When one wants to solve this system over $\frac{\mathbb{Z}}{65521\mathbb{Z}}$ one just replaces 0 by 65521 in the second line. Note that in the positive characteristic case the coefficients used should be smaller or equal to $2^{31} - 1$.

4 Solving polynomial systems of dimension at most zero

Let $\mathbb{K}$ be a field and $\overline{\mathbb{K}}$ be an algebraic closure of $\mathbb{K}$. We consider polynomials $(f_1, \ldots, f_s)$ in $\mathbb{K}[x_1, \ldots, x_n]$ and V be the set of solutions to the system

$$f_1 = \cdots = f_s = 0$$

in $\overline{\mathbb{K}}^n$.

When V is empty in $\overline{\mathbb{K}}[x_1, \ldots, x_n]$, the output of MSOLVE is

```
[1, []]
```

When V is finite, say V is the union of the ℓ points $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_\ell$ in $\overline{\mathbb{K}}^n$ with $\boldsymbol{\alpha}_i = (\alpha_{i,1}, \ldots, \alpha_{i,n})$.

When the input system has finitely many solutions in $\overline{\mathbb{K}}^n$, the output is encoded as follows:

```
[dim, [deg, nbsols, form, vars, [lw, lwp, param]]]
```

where

- `dim` is 0 (which indicates that the input system has finitely many solutions);
- `deg` is the number of solutions counted with multiplicities in $\overline{\mathbb{K}}^n$;
- `nbsols` is the number of solutions in $\overline{\mathbb{K}}^n$;
- `form` is the linear form λ
- `vars` is the list of the variables which are parametrized;
- `lw` is the encoding of the eliminating polynomial w;
- `lwp` is the encoding of the denominator used in the rational parametrization;
- `param` is the list of the output parametrizations as described above ; the i-th one corresponds to the i-th element of `vars`.

5 Solving polynomial systems of positive dimension

6 Julia interface to MSOLVE

The Julia interface to MSOLVE is part of the official Julia package `GroebnerBasis.jl`. You can install the package via the following commands inside a Julia session:

```
using Pkg
Pkg.add("GroebnerBasis")
```

Once the package is loaded via `using GroebnerBasis` one can call the function `msolve()` which returns, if any, the solutions of the given input system of multivariate polynomials. The most common calling convention is as follows:

```
solutions = msolve(I).
```

where

- `I` is of type `Singular.sideal` and
- `solutions` is of type `Array{Array{Rational{Int64},1},1}`.

The most common options for calling `msolve()` in Julia are:

- `info_level` with values `0` (no information printing; default), `1` (slight information printing on comptutational status) or `2` (full information printing also on intermediate steps),
- `la_option` for the linear algebra variant to be chosen inside F4: `2` for exact linear algebra and tracing multi modular computations (default) or `44` for probabilistic linear algebra with independent modular computations;
- `precision` for the bit precision with which the solutions are computed from the rational parametrization. Default is `64`.
- `get_param` for returning, besides the solutions, also the rational parametrization for the solution set. Default is `false`.

So using `msolve()` with probabilistic linear algebra, the most verbose information printout and a precision of 80 one would call

```
solutions = msolve(I, la_option=44, info_level=2, precision=80).
```

Returning also the rational parametrization of the solution set one would call

```
param, solutions = msolve(I, get_param=true).
```

Having the rational parametrization `param` one can further solve trying, for example, higher

precision:

```
newly_computed_solutions = solve_rational_parametrization(param)
```

You can get a full list of options for `msolve()` in Julia by typing inside Julia

```
?  msolve()
```

7 Maple interface to MSOLVE

The Maple interface to MSOLVE can be found on the MSOLVE homepage or in the MSOLVE binary package. Having loaded the interface one can call the function `MSolveRealRoots()` in the following way:

```
results = MSolveRealRoots(F, vars, msolve_binary, in_file, out_file, get_parametrization)
```

where `F` denotes a polynomial system in variables `vars`, `msolve_binary` gives the path to the binary (default is "`../binary/msolve`"), `in_file` is the name of the input file for MSOLVE (default is "`/tmp/in.ms`"), `out_file` is the name of the output file for MSOLVE (default is "`/tmp/out.ms`"), and `get_parametrization` defines if the rational parametrization of the solution set is also returned (default is `0`, i.e. the parametrization is not returned). Depending on parameter `get_parametrization` `results` consists only of the solutions or it is an array consisting of the rational parametrization as first entry and the solutions as second entry.

8 Sage interface to MSOLVE

The Sage interface to MSOLVE can be found on the MSOLVE homepage or in the MSOLVE binary package. Starting Sage we load the interface:

```
load("msolve-to-sage-file-interface.sage").
```

Defining a multivariate polynomial ring `R` and a list `F` of polynomials in `R` one can solve the multivariate polynomial system defined by `F` via calling

```
results = MSolveRealRoots(
F, mspath="/path/to/msolve/binary", v=verbosity, p=get_parametrization)
```

Parameter `mspath` states the path to the msolve binary (default is "`../binary/msolve`"). The verbosity level can be set to `0` (no information printing; default), `1` (slight information printing on computational status) or `2` (full information printing also on intermediate steps). The `p` parameter can be set to `0` (rational parametrization of solution set is not returned) or `1` (returns also the rational parametrization of the solution set). Depending on parameter `p` `results` consists

only of the solutions or it is an array consisting of the rational parametrization as first entry and the solutions as second entry.

9 Credits

The main developers of MSOLVE are Jérémy Berthomieu, Christian Eder and Mohab Safey El Din. It relies on original implementations of Faugère's F_4 algorithm [1] as well as Faugère and Mou's Sparse-FGLM algorithm [2]. We are grateful to Huu Phuoc Le and Jorge Garcia Fontan for a preliminary version of the Maple interface as well as Rémi Prébet for a preliminary version of the Sage interface.

References

[1] J.-C. Faugère. "A new efficient algorithm for computing Gröbner bases (F4)". In: *Journal of Pure and Applied Algebra* 139.1-3 (1999), pp. 61–88.

[2] J.-C. Faugère and C. Mou. "Sparse FGLM algorithms". In: *Journal of Symbolic Computation* 80 (2017), pp. 538–569.