Numerical View of Polynomial Systems

Abhishek Bhardwaj
Supervisor: Professor Markus Hegland

Australian National University

June 13, 2018

Polynomials in Geometry & Algebra

Given a set of polynomials $\{p_1, \ldots, p_n\} \subseteq \mathbb{R}[x_1, \ldots, x_k]$, consider the problem of finding $\mathbf{x} = (x_1, \ldots, x_k)$ such that,

$$p_1(\mathbf{x}) = 0,$$

$$\vdots$$

$$p_n(\mathbf{x}) = 0.$$

There are many descriptions to the solution of this problem.

Polynomials in Geometry & Algebra

The solutions $\mathbf{x}$ form a *Variety*

$$V(\{p_1, \ldots, p_n\}) = \{\mathbf{x} \quad | \quad p_1(\mathbf{x}) = \cdots = p_n(\mathbf{x}) = 0\}$$

Polynomials in Geometry & Algebra

The solutions $\mathbf{x}$ form a *Variety*

$$V(\{p_1, \ldots, p_n\}) = \{\mathbf{x} \quad | \quad p_1(\mathbf{x}) = \cdots = p_n(\mathbf{x}) = 0\}$$

Example

Taking $p_1(x_1, x_2) = x_1^2 + x_2^2 - 2$, and $p_2(x_1, x_2) = x_1 + x_2$, gives us

$$V(\{p_1, p_2\}) = \{(-1, 1), (1, -1)\}.$$

Polynomials in Geometry & Algebra

A more generic description is that of the *Ideal* $I \subseteq \mathbb{R}[\mathbf{x}]$, generated by $\{p_1, \ldots, p_n\}$;

$$I = \langle p_1, \ldots, p_n \rangle = \left\{ \sum_1^n q_i p_i \quad | \quad q_1, \ldots, q_n \in \mathbb{R}[\mathbf{x}] \right\}$$

Polynomials in Geometry & Algebra

A more generic description is that of the *Ideal* $I \subseteq \mathbb{R}[\mathbf{x}]$, generated by $\{p_1, \ldots, p_n\}$;

$$I = \langle p_1, \ldots, p_n \rangle = \left\{ \sum_1^n q_i p_i \quad | \quad q_1, \ldots, q_n \in \mathbb{R}[\mathbf{x}] \right\}$$

Question

Is $\{p_1, \ldots, p_n\}$ the simplest set of generators of I?

Gröbner Basis

There are in fact many "simpler" generators of I.

Gröbner Basis

A Gröbner basis for an ideal I is a set G, such that for every $p \in I$, there exist $g_1, \ldots, g_k \in G$ and $h_1, \ldots, h_k \in \mathbb{R}[\mathbf{x}]$ such that

$$p = h_1 g_1 + \cdots + h_k g_k.$$

Gröbner Basis

There are in fact many "simpler" generators of I.

Gröbner Basis

A Gröbner basis for an ideal I is a set G, such that for every $p \in I$, there exist $g_1, \ldots, g_k \in G$ and $h_1, \ldots, h_k \in \mathbb{R}[\mathbf{x}]$ such that

$$p = h_1 g_1 + \cdots + h_k g_k.$$

How does this help?

Buchberger's Algorithm

Input: A set of polynomials F which generate the ideal I.
Output: A Gröbner Basis G.

1 Set $G := F$.

Buchberger's Algorithm

Input: A set of polynomials F which generate the ideal I.
Output: A Gröbner Basis G.

1. Set $G := F$.
2. For every $f_i, f_j \in G$, denote by g_i the leading term of f_i with respect to the given ordering, and by a_{ij} the least common multiple of g_i and g_j.

Buchberger's Algorithm

Input: A set of polynomials F which generate the ideal I.
Output: A Gröbner Basis G.

1. Set $G := F$.
2. For every $f_i, f_j \in G$, denote by g_i the leading term of f_i with respect to the given ordering, and by a_{ij} the least common multiple of g_i and g_j.
3. Choose two polynomials in G and let $S_{ij} = (a_{ij}/g_i)f_i - (a_{ij}/g_j)f_j$.

Buchberger's Algorithm

Input: A set of polynomials F which generate the ideal I.
Output: A Gröbner Basis G.

1. Set $G := F$.
2. For every $f_i, f_j \in G$, denote by g_i the leading term of f_i with respect to the given ordering, and by a_{ij} the least common multiple of g_i and g_j.
3. Choose two polynomials in G and let $S_{ij} = (a_{ij}/g_i)f_i - (a_{ij}/g_j)f_j$.
4. Reduce (completely) S_{ij}, with the multivariate division algorithm relative to the set G. If the result is non-zero, add it to G.

Buchberger's Algorithm

Input: A set of polynomials F which generate the ideal I.
Output: A Gröbner Basis G.

1. Set $G := F$.
2. For every $f_i, f_j \in G$, denote by g_i the leading term of f_i with respect to the given ordering, and by a_{ij} the least common multiple of g_i and g_j.
3. Choose two polynomials in G and let $S_{ij} = (a_{ij}/g_i)f_i - (a_{ij}/g_j)f_j$.
4. Reduce (completely) S_{ij}, with the multivariate division algorithm relative to the set G. If the result is non-zero, add it to G.
5. Repeat steps 1-4 until all possible pairs are considered, including those involving the new polynomials added in step 4.

Buchberger's Algorithm

Input: A set of polynomials F which generate the ideal I.
Output: A Gröbner Basis G.

1. Set $G := F$.
2. For every $f_i, f_j \in G$, denote by g_i the leading term of f_i with respect to the given ordering, and by a_{ij} the least common multiple of g_i and g_j.
3. Choose two polynomials in G and let $S_{ij} = (a_{ij}/g_i)f_i - (a_{ij}/g_j)f_j$.
4. Reduce (completely) S_{ij}, with the multivariate division algorithm relative to the set G. If the result is non-zero, add it to G.
5. Repeat steps 1-4 until all possible pairs are considered, including those involving the new polynomials added in step 4.
6. Output G

The Matricial Representation

Rewrite the polynomials in matrix form

$$C = \begin{array}{c} \\ p_1 \\ \vdots \\ p_{n-1} \\ p_n \end{array} \begin{array}{c} \begin{array}{cccccccccc} x_1^d \ldots x_k^d & \ldots & x_1^d & \ldots & x_{k-1} & x_k^d & \ldots & x_k & 1 \end{array} \\ \begin{pmatrix} * & \ldots & * & \ldots & * & * & \ldots & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & \ldots & * & \ldots & * & * & \ldots & * & * \\ * & \ldots & * & \ldots & * & * & \ldots & * & * \end{pmatrix} \end{array}.$$

The Matricial Representation

Rewrite the polynomials in matrix form

$$C = \begin{array}{c} \\ p_1 \\ \vdots \\ p_{n-1} \\ p_n \end{array} \begin{array}{c} \begin{array}{ccccccccc} x_1^d \dots x_k^d & \dots & x_1^d & \dots & x_{k-1} & x_k^d & \dots & x_k & 1 \end{array} \\ \begin{pmatrix} * & \dots & * & \dots & * & * & \dots & * & * \\ \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & \dots & * & \dots & * & * & \dots & * & * \\ * & \dots & * & \dots & * & * & \dots & * & * \end{pmatrix} \end{array}.$$

Let

$$\mathbf{x^k} = \begin{pmatrix} x_1^d \dots x_k^d & \dots & x_1^2 & \dots & x_k^d & \dots & 1 \end{pmatrix},$$

then

$$\begin{cases} p_1(\mathbf{x}) = 0 \\ \vdots \\ p_n(\mathbf{x}) = 0 \end{cases} \Leftrightarrow C\mathbf{x^k} = 0$$

The Matricial Representation

Buchberger's algorithm takes the following form. Take two polynomials in the system;

$$\begin{array}{c} \\ p_i \\ p_j \end{array} \begin{array}{c} \begin{array}{cccccccc} c_{12} & & c_1 & & c_2 & & & \end{array} \\ \begin{pmatrix} 0 & 0 & \dots & 1 & \dots & * & \dots & * & * \\ 0 & 0 & \dots & 0 & \dots & 1 & \dots & * & * \end{pmatrix} \end{array}$$

The Matricial Representation

Buchberger's algorithm takes the following form. Take two polynomials in the system;

$$\begin{array}{c} \\ p_i \\ p_j \end{array} \begin{array}{c} \begin{array}{cccccccc} c_{12} & & c_1 & & c_2 & & & \end{array} \\ \begin{pmatrix} 0 & 0 & \dots & 1 & \dots & * & \dots & * & * \\ 0 & 0 & \dots & 0 & \dots & 1 & \dots & * & * \end{pmatrix} \end{array}$$

Compute the relevant LCM of the leading terms, and modify the rows to obtain

$$\begin{array}{c} \\ \tilde{p}_i \\ \tilde{p}_j \end{array} \begin{array}{c} \begin{array}{cccccccc} c_{12} & & c_1 & c_2 & & & & \end{array} \\ \begin{pmatrix} 0 & 1 & \dots & * & \dots & * & \dots & * & * \\ 0 & 1 & \dots & * & \dots & * & \dots & * & * \end{pmatrix}, \end{array}$$

the S-polynomial, S_{ij} is the difference $\tilde{p}_i - \tilde{p}_j$.

The Matricial Representation

Update the system to

$$C = \begin{array}{c} \\ p_1 \\ \vdots \\ p_n \\ S_{i,j} \end{array} \begin{array}{c} \begin{array}{cccccccc} x_1^d & \dots & x_k^d & \dots & x_1^d & \dots & x_k^d & \dots \end{array} \\ \begin{pmatrix} * & \dots & * & \dots & * & \dots & * & * \\ \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & \dots & * & \dots & * & \dots & * & * \\ * & \dots & * & \dots & * & \dots & * & * \end{pmatrix} \end{array} \begin{array}{c} x_k \\ \\ 1 \end{array}.$$

The Matricial Representation

Update the system to

$$C = \begin{array}{c} \\ p_1 \\ \vdots \\ p_n \\ S_{i,j} \end{array} \begin{array}{c} \begin{array}{cccccccc} x_1^d \ldots x_k^d & \ldots & x_1^d & \ldots & x_k^d & \ldots & x_k & 1 \end{array} \\ \begin{pmatrix} * & \ldots & * & \ldots & * & \ldots & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & \ldots & * & \ldots & * & \ldots & * & * \\ * & \ldots & * & \ldots & * & \ldots & * & * \end{pmatrix} \end{array}.$$

Repeat until

$$C = \begin{array}{c} \begin{array}{ccccccccc} x_1^d \ldots x_k^d & \ldots & x_1^d & \ldots & x_{k-1} & x_k^d & \ldots & x_k & 1 \end{array} \\ \begin{pmatrix} * & \ldots & * & \ldots & * & * & \ldots & * & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ * & \ldots & * & \ldots & * & * & \ldots & * & * \\ 0 & \ldots & 0 & \ldots & 0 & * & \ldots & * & * \end{pmatrix} \end{array}.$$

Updating C

Notice that we may write

$$\mathbf{x^k} = \begin{pmatrix} x_k^d & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_k^0 & 0 & \dots & 0 \\ 0 & x_k^d & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & x_k^0 & \dots & 0 \\ 0 & 0 & \dots & x_k^d \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & x_k^0 \end{pmatrix} \mathbf{x^{k-1}}$$

Updating C

Which leads to

$$C\mathbf{x^k} = C \begin{pmatrix} x_k^d & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ x_k^0 & 0 & \dots & 0 \\ 0 & x_k^d & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & x_k^0 & \dots & 0 \\ 0 & 0 & \dots & x_k^d \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & x_k^0 \end{pmatrix} \mathbf{x^{k-1}}$$

Solving for $\mathbf{x}$

Numerically solving for the last variable at each step, we can iterate the process to obtain

$$C\mathbf{x^k} = C_k C_{k-1} \dots C_2 \mathbf{x^1}.$$

Initial Results

Let $p = 10^6$ and consider the system of equations

$$f_1 = x^p + y^p,$$
$$f_2 = x^p.$$

The Gröbner Basis is clearly $\{x^p, y^p\}$.
Our software takes 3.58 ms and less than 0.01 MB of memory to solve this. On the other hand SymPy takes 6.04 s and 168.7 MB.

Initial Results

Let $q = 10^4$ and consider

$$f_1 = x^{2q}z^q + x^q y^q z^q + x^q y^q,$$
$$f_2 = x^{2q}y^{2q}z^{2q} + x^{2q}x^q + x^q y^q,$$
$$f_3 = x^q y^q z^q + xz^{2q} + z^q$$

For this kind of high degree system, our software performs much better taking only 326 ms and less than 0.01 MB memory, while SymPy takes 625 ms and 5.74 MB.

Gaussian Quadrature

Consider the quadrature equations with constant weights

$$\begin{pmatrix} x_1 & x_2 & x_3 & \dots & x_k & a_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^k & x_2^k & x_3^k & \vdots & x_k^k & a_k \end{pmatrix}.$$

Gaussian Quadrature

Consider the quadrature equations with constant weights

$$\begin{pmatrix} x_1 & x_2 & x_3 & \dots & x_k & a_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^k & x_2^k & x_3^k & \vdots & x_k^k & a_k \end{pmatrix}.$$

Instead of using the S-polynomial, we can eliminate x_1 from the higher order equations by using the first one.

Gaussian Quadrature

Question

Can Buchberger's algorithm be simplified for problems with special structure?

Conjecture

If we use $\tilde{p}_i(x)$ to eliminate x_i from $\tilde{p}_j(x)$ (with $i < j$), then the leading term of $\tilde{p}_j(x)$ is divisible by the leading term of $\tilde{p}_i(x)$ at every step of the elimination.

Future Work

1. Parallelizing procedures in the algorithms
2. Ensuring the sparsity of C
3. Symmetric systems
   i. Higher dimensional quadrature
   ii. Orthogonal polynomials
   iii. Simplifying the algorithm
4. Generalizations
   i. Tensors and tensor calculus
   ii. Non-commutative polynomial systems
   iii. Semi-algebraic systems

Thank you.