— Inverting Matrices Modulo Zero Dimensional Regular Chains —

Marc Moreno Maza and Paul Vrbik

Zero Dimensional Regular Chains

Regular Chains play a fundamental role in polynomial system solving. Namely, they can encode the generic points of the irreducible components of algebraic varieties. [3].

Of particular interest in practice is when these varieties are zero dimensional (i.e. finite). For instance, the authors of [1] have developed a probabilistic and modular algorithm for solving zero-dimensional polynomial systems with rational coefficients. Their algorithm requires to **invert polynomial matrices modulo regular chains**. For sufficiently large problems, this operation is a bottleneck, mainly due to memory consumption when testing the invertibility of an element modulo a regular chain.

The Leverrier-Faddeev Algorithm

The **Leverrier-Faddeev** [2] algorithm is a method for finding a matrix inverse that only does one division but requires repeated matrix multiplication.

Consider the characteristic polynomial of the $m \times m$ matrix $\mathbf{A}$:

$$p(\lambda) = \det(\lambda \mathbf{I} - \mathbf{A}) = \lambda^m - a_1\lambda^{m-1} - \cdots - a_{m-1}\lambda - a_m.$$

An expression for the inverse of $\mathbf{A}$ is given by evaluating $p(\mathbf{A})$, multiplying by $\mathbf{A}^{-1}$ and re-arranging terms:

$$\begin{aligned} 0 &= \mathbf{A}^m - a_1\mathbf{A}^{m-1} - \cdots - a_{m-1}\mathbf{A} - a_m \\ \mathbf{A}^{-1}a_m &= \mathbf{A}^{m-1} - a_1\mathbf{A}^{m-2} - \cdots - a_{m-1} \\ \mathbf{A}^{-1} &= \left(\mathbf{A}^{m-1} - \sum_{i=1}^{m-1} a_i \mathbf{A}^{n-i-1}\right) a_m^{-1}. \end{aligned} \qquad (1)$$

a_k can be obtained successively by $a_k = \frac{1}{k}\left(s_k - \sum_{i=1}^{k-1} s_{k-i}a_i\right)$, where $s_k = \text{trace}(\mathbf{A}^k)$ and $a_1 = s_1$. Thus, in order to find the inverse of $\mathbf{A}$, the only ring division we must do is by $\det(\mathbf{A})$.

Optimizations

Calculate the s_k's by "baby step giant step".

Store $M_0, M_1, M_2, \ldots, M_\ell = \mathbf{A}^0, \mathbf{A}, \mathbf{A}^2, \ldots, \mathbf{A}^\ell$ where $\ell = \lfloor\sqrt{m}\rfloor$.

Generate $N_0, N_1, N_2, \ldots, N_k = \mathbf{A}^0, \mathbf{A}^{\ell+1}, \mathbf{A}^{2(\ell+1)}, \ldots, \mathbf{A}^{2k(\ell+1)}$ on the fly (repeatedly multiplying by $\mathbf{A}^{\ell+1}$, without storing).

Get the traces in blocks by $\text{tr}(M_iN_j) = \text{tr}(\mathbf{A}^i\mathbf{A}^{(\ell+1)\cdot j}) = \text{tr}(\mathbf{A}^{i+(\ell+1)\cdot j})$ taking $0 \le i, j \le \ell$. For example if $n = 8$ with $\ell = \lfloor\sqrt{8}\rfloor = 2$ do $\{\text{tr}(\mathbf{A}^0\mathbf{A}^0), \ldots, \text{tr}(\mathbf{A}^0\mathbf{A}^2)\}$ $\{\text{tr}(\mathbf{A}^3\mathbf{A}^0), \ldots, \text{tr}(\mathbf{A}^3\mathbf{A}^2)\}$ $\{\text{tr}((\mathbf{A}^3\mathbf{A}^3)\mathbf{A}^0), \ldots, \text{tr}((\mathbf{A}^3\mathbf{A}^3)\mathbf{A}^2)\}$.

The complexity is given by (NUMBER OF $\times$'S FOR THE N'S AND M'S) + (NUMBER OF $\times$'S FOR THE TRACES) $= 2m^3\sqrt{m} + m^3 = m^3(1 + 2\sqrt{m})$ $\times$'s.

Expand (1) by expressing $p(\mathbf{A})$ in NESTED FORM as

$$p(\mathbf{A}) = \left(\cdots\left(\left(\left(\sum_{i=0}^{t-1} a_iM_{t-1-i}\right)N_1 + \sigma(0)\right)N_1 + \sigma(1)\right)N_1 + \cdots\right)N_1 + \sigma\left(\frac{m-1-\ell-t}{\ell+1}\right)$$

with $t \equiv m \bmod (\ell+1)$ and $\sigma(k) = \sum_{i=t+k\ell+k}^{t+k\ell+k+\ell} a_iM_{(m-i-1) \bmod (\ell+1)}$

The complexity is given by the matrix multiplications required to do $\sigma(k)$ and $\sum_{i=0}^{t-1} a_iM_{t-1-i}$ which amounts to $m^3\left(\frac{m+1+d-t}{d+1}\right)$ coefficient multiplications.

We express this as a function of m using $s = m \bmod (d+1) \le d$ and $d \le \sqrt{m}$.

$$m^3 \cdot \frac{m+1+d-t}{d+1} < m^3\left(\frac{m+1+\sqrt{m}}{1+\sqrt{m}}\right) < m^3\left(\frac{m}{\sqrt{m}}+1\right) < \boldsymbol{O}\left(\boldsymbol{m^3\sqrt{m}}\right)$$

Acknowledgements

NSERC CRSNG — Maplesoft — MITACS

Using Leverrier-Faddeev Recursively

Use Leverrier-Faddeev algorithm to **find a_m^{-1} recursively**. For $\mathbf{T}$ a zero dimensional regular chain with coefficients in the field $\mathbb{K}$, define the linear map:

$$\begin{aligned} m_f : \mathbb{K}[x_1, \ldots, x_{n-1}][x_n] &\mapsto (\mathbb{K}[x_1, \ldots, x_{n-1}]/\langle T_1, \ldots, T_{n-1}\rangle)[x_n]/\langle T_n\rangle \\ \alpha &\mapsto f\alpha \end{aligned}$$

such that $m_f([g]) = [f] \cdot [g] = [fg]$ (or more simply: $m_f(g) = \overline{fg}^{\mathbf{T}}$).

Since $\mathbb{K}[x_1, \ldots, x_n]/\mathbf{T}$ is finite dimensional it has a finite monomial basis B. We can thus represent m_f by its matrix with respect to this basis.

The multiplication matrix satisfies $m_f \cdot m_g = m_{fg}$ and thus **we can find the inverse of a_m by inverting its corresponding multiplication matrix**.

Space Complexity

For Leverrier-Faddeev. Let $F(m, [d_1, \ldots, d_n])$ be the number of field elements required to invert an $m \times m$ matrix modulo a regular chain $T = \langle T_1, \ldots, T_n\rangle \subset \mathbb{K}[x_1, \ldots, x_n]$ with $d_i = \text{degree}_{x_i}(T_i)$. Assuming completely dense input we have

$$\begin{aligned} F(m, [d_1, \ldots, d_n]) &= \sqrt{m} \cdot m \cdot m \cdot d_1 \cdots d_n && \text{input and } M_i\text{'s} \\ &+ m \cdot d_1 \cdots d_n && \text{traces} \\ &+ F(d_n, [d_1, \ldots, d_{n-1}]) && \text{recursive call} \\ &+ m \cdot m \cdot d_1 \cdots d_n && \text{expansion} \end{aligned}$$

Letting $\sigma = \sum \text{degree}_{x_i}(T_i)$ and $\delta = \prod \text{degree}_{x_i}(T_i)$ we can bound the above recurrence by $O(m^{2.5}\delta + \delta\sigma^{1.5})$. Adding the space required for field multiplication gives a space complexity of $\boldsymbol{O(2^n\delta + m^{2.5}\delta + \delta\sigma^{1.5})}$ **field elements**.

For GCD based Algorithm. Here one follows the method of Bareiss testing invertibility by using an Euclidean-like algorithm. In [4] the space complexity for this is given by (setting $\delta_i = \prod_{j=1}^{i} d_i$ and otherwise reusing the above notation) $\boldsymbol{2m^2\delta + O(2^n n^2)\sum_{i=2}^{n}\left(d_i^{i-2} \cdot \delta_i\right)}$ **field elements**.

Experimental Results

We compare two approaches: recursive Leverrier-Faddeev algorithm and the existing (Bareiss based) method. We choose a random dense regular chain $\boldsymbol{T} \subset \mathbb{F}_p[x_1, \ldots, x_n]$ with $\text{degree}(\boldsymbol{T_i}) = \boldsymbol{6}$, varying $\boldsymbol{n}$ and $\boldsymbol{p = 962592769}$. Our matrix is a random (invertible) $\boldsymbol{m \times m}$ matrix with dense entries from $\mathbb{F}_p[x_1, \ldots, x_n]/\langle \boldsymbol{T}\rangle$.

		Recursive Lev-Fad					Bareiss	
Vars	Matrix Size	Time	Trace	Inv	Exp	Space	Time	Space
3	11×11	157.34s	0.06%	2.74%	97.21%	0.10GB	1102.310s	0.18GB
4	7×7	408.15s	37.65%	10.56%	51.80%	0.11GB	—	4.0GB
5	1×1	800.43s	19.24%	60.91%	19.85%	0.41GB	*	>4.0GB

The columns "Trace", "Inv" and "Exp" show the proportion of the time spent calculating the s_k's, the inverses, and expansion of the nested form (respectively). "—" means computation was cut off (after 1 hour) due to over 90% memory usage. "*" means that MAPLE ran out of memory.

[1] X. Dahan, M. Moreno Maza, É. Schost, W. Wu, and Y. Xie. Lifting techniques for triangular decompositions. In *ISSAC'05*, pages 108–115. ACM Press, 2005.

[2] D. K. Faddeev and V. N. Faddeeva. *Computational Methods of Linear Algebra*. Freeman, San Francisco, 1963.

[3] M. Kalkbrener. A generalized Euclidean algorithm for computing triangular representations of algebraic varieties. *J. Symb. Comp.*, 15:143–167, 1993.

[4] Xin Li, Marc Moreno Maza, and Wei Pan. Computations modulo regular chains. In *Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, ISSAC '09, pages 239–246, New York, NY, USA, 2009. ACM.