Kronecker Representation

Solving a polynomial system in the sense of Kronecker means developing techniques to compute ***with*** the roots [6]. We start by following the exposition in [7], introducing the concept of a geometric resolution [4]. With Newton-Hensel lifting we compute power series solutions.

1 Geometric Resolution

Given a set V of d distinct solutions in n-space, their number equal to D when counted with multiplicities. A geometric resolution of V consists of

1. a linear form $\ell(\mathbf{x}) = u_0 + u_1x_1 + u_2x_2 + \cdots + u_nx_n$,
2. polynomials $q_1, p_1, p_2, \ldots, p_n \in \mathbb{Q}[T]$ such that
   (a) for all $\mathbf{z}_j, \mathbf{z}_k \in V$, $j \neq k$: $\ell(\mathbf{z}_j) \neq \ell(z_k)$,
   (b) $q(T) = \prod_{i=1}^{D} (T - \ell(\mathbf{z}_i))$,
   (c) for $j = 1, 2, \ldots, n$: $\deg(p_j) \leq D-1$ and $V = \{ \ (p_1(r), p_2(r), \ldots, p_n(r)) \mid r \in \mathbb{C} : q(r) = 0 \ \}$.

For a fixed linear form ℓ, this description is unique so we speak then of the geometric resolution of V associated to ℓ. The linear form separates the points in V and plays the same role as in the rational univariate representation.

Consider [1, Example 4.3.7]:

$$f(x_1, x_2) = \begin{cases} x_1^2 + (x_2 - 1)^2 - 1 = 0 \\ x_1^2 - x_2 = 0. \end{cases} \tag{1}$$

The system has four solutions, counted with multiplicity, there is a double solution at $(0, 0)$. We see that x_1 separates the zeroes, but that x_2 does not. For $\ell = x_1$, the geometric resolution of $V = f^{-1}(\mathbf{0})$ consists of

$$\ell(\mathbf{x}) = x_1, \quad q(T) = T^2(T-1)(T+1), \quad p_1(T) = T, \quad p_2(T) = T^2. \tag{2}$$

The geometric interpretation of this solution corresponds to looking at the algebraic curves defined by f_1 and f_2, intersected with the line $x_1 = T$. Consider first the circle $f_1 = x_1^2 + (x_2 - 1)^2 - 1 = 0$, for $x_1 = T$ and then the computation of $q(T)$ is done with a resultant, eliminating x_2 from $f_1(T, x_2)$ and $f_2(T, x_2)$. In Maple:

```
> resultant(T^2 + (x-1)^2 - 1, T^2 - x,x);
```

returns $-T^2 + T^4 = T^2(T-1)(T+1)$. Then p_1 and p_2 follow from $x_1 = T$ and $x_1^2 - x_2 = 0$. According to the Kronecker philosophy of solving we compute with roots modulo $q(T)$ and replace T^4 by T^2.

The Kronecker representation (or Kronecker parametrization) of a finite set V of solutions has the form

$$q(T) = 0 \quad \begin{cases} \frac{\partial q}{\partial T} x_1 = p_1(T) \\ \frac{\partial q}{\partial T} x_2 = p_2(T) \\ \qquad \vdots \\ \frac{\partial q}{\partial T} x_n = p_n(T) \end{cases} \tag{3}$$

where q, p_1, $p_2, \ldots, p_n$ are polynomials of degree bounded by $D = \#V$.

2 Camera Motion from Point Matches

The application we consider comes from computer vision, see [2] and [3]. The problem is to compute the displacement of a camera between two positions in a static environment.

Consider a point A viewed by two camera positions giving images $\mathbf{a}$ and $\mathbf{a}'$ in their respective coordinate frames (x, y, z) and (x', y', z'), via a rotation R and translation $\mathbf{t}$. The condition that the images come from the same point seen from two different camera positions is equivalent to the coplanarity of the vectors $\mathbf{a}$, $\mathbf{t}$, and $R\mathbf{a}'_i + \mathbf{t}$, expressed in [2] by

$$\mathbf{a}_i^T(\mathbf{t} \times (R\mathbf{a}'_i + \mathbf{t})) = \mathbf{a}_i^T(\mathbf{t} \times R\mathbf{a}'_i) = 0, \quad \text{for } i = 1, 2, \ldots, 5, \tag{4}$$

where $\times$ denotes the cross product. For five points, the number of solutions to this problem is finite and equals ten, as shown in [3].

Using a quaternion formulation, in [2], a system of 5 equations in 6 unknowns (two 3-vectors $\mathbf{q}$ and $\mathbf{d}$) is derived:

$$\begin{cases} 1 - \mathbf{d}^T\mathbf{q} = 0 \\ (\mathbf{a}_i^T\mathbf{q})(\mathbf{d}^T\mathbf{a}'_i) + \mathbf{a}_i^T\mathbf{a}'_i + (\mathbf{a}_i \times \mathbf{q})^T\mathbf{a}'_i + (\mathbf{a}_i \times \mathbf{q})^T(\mathbf{d} \times \mathbf{a}'_i) + \mathbf{a}_i(\mathbf{d} \times \mathbf{a}'_i) = 0, \ i = 1, 2, \ldots, 5. \end{cases} \tag{5}$$

This system is bilinear in $\mathbf{q}$ and $\mathbf{d}$ and has a 2-homogeneous Bézout bound of 20. As pointed out in [2], all solutions can be real.

3 The Newton-Hensel Method

Lemma 3.1 *Consider $f_1, f_2, \ldots, f_n \in \mathbb{Q}[\mathbf{t}, \mathbf{x}]$, $\mathbf{t} = (t_1, t_2, \ldots, t_m)$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and J_f is the Jacobian matrix of $f = (f_1, f_2, \ldots, f_n)$ with respect to $\mathbf{x}$. Given $(\boldsymbol{\tau}, \mathbf{z}) \in \mathbb{C}^m \times \mathbb{C}^n$ such that $f(\boldsymbol{\tau}, \mathbf{z}) = \mathbf{0}$ and $\det(J_f(\boldsymbol{\tau}, \mathbf{z})) \neq 0$. Then there is a unique n-tuple of formal power series $S \in (\mathbb{C}[[\mathbf{t} - \boldsymbol{\tau}]])^n$ such that*

1. *$S(\boldsymbol{\tau}) = \mathbf{z}$, and*
2. *$f(\mathbf{t}, S) = 0$.*

The proof of the lemma uses the Newton-Hensel operator:

$$N_f(\mathbf{x}) = \mathbf{x} - [J_f(\mathbf{x})]^{-1} f(\mathbf{x}). \tag{6}$$

By the assumption $\det(J_f(\boldsymbol{\tau}, \mathbf{z})) \neq 0$, the matrix $J_f(\mathbf{t}, \mathbf{x})$ is invertible at $\mathbf{x} = \mathbf{z}$ for $\mathbf{t} = \boldsymbol{\tau}$. If we set $S^{(0)} = \mathbf{z}$, then the Newton-Hensel operator defines via $S^{(k+1)} = N_f(S^{(k)})$ a sequence of power series. The sequence converges quadratically in the sense that if the error term in $S^{(k)}$ is $O((\mathbf{t} - \boldsymbol{\tau})^p)$ then the error term in $S^{(k+1)}$ is $O((\mathbf{t} - \boldsymbol{\tau})^{2p})$.

To invert a power series $f \in \mathbb{C}[[t]]$, we may apply the following identity:

$$f^{-1} = \frac{1}{1 - (1 - f)} = \sum_{k=0}^{\infty} (1 - f)^k \tag{7}$$

truncating the infinite series at some order d. Note that if we are computing a polynomial of degree d, once we have d correct terms of a series expansion, we know we can truncate the result without errors.

Consider for example:

$$f(x_1, x_2, t) = \begin{cases} x_1^2 + (x_2 - 1)^2 - 1 + t^2 = 0 \\ x_1^2 - x_2 + t = 0 \end{cases} \tag{8}$$

where for $t = 0$ we have the solution $(x_1 = 1, x_2 = 1)$. With Maple we can symbolically compute the inverse of the Jacobian matrix and execute Newton's method.

```
> f := [x[1]^2 + (x[2]-1)^2 - 1 + t^2, x[1]^2 - x[2] + t]:
> J := Matrix([seq([seq(diff(f[i],x[j]),j=1..2)],i=1..2)]):
> Jinv := J^(-1);
> sNf := Vector([x[1],x[2]]) - Jinv.Vector(f):
> newton := (a,b) -> subs(x[1]=a,x[2]=b,sNf):
> X[0] := [1,1]:
  for k from 1 to 4 do
     X[k] := newton(X[k-1][1],X[k-1][2]);
     s[k] := [series(X[k][1],t=0,8),series(X[k][2],t=0,8)];
  end do:
```

After each Newton step, we develop the solution as a series around $t = 0$:

$$S^{(0)} = [1, 1] \tag{9}$$

$$S^{(1)} = \left[1 - \frac{1}{2}t^2, 1 + t - t^2\right] \tag{10}$$

$$S^{(2)} = \left[1 - t^2 + 2t^3 - \frac{47}{8}t^4 + 16t^5 - \frac{703}{16}t^6 + 120t^7 + O(t^8), 1 + t - 2t^2 + 4t^3 - 11t^4 + \cdots\right] \tag{11}$$

$$S^{(3)} = \left[1 - t^2 + 2t^3 - \frac{13}{2}t^4 + 22t^5 - \frac{161}{2}t^6 + 307t^7 + O(t^8), 1 + t - 2t^2 + 4t^3 - 12t^4 + \cdots\right] \tag{12}$$

$$S^{(4)} = \left[1 - t^2 + 2t^3 - \frac{13}{2}t^4 + 22t^5 - \frac{161}{2}t^6 + 307t^7 + O(t^8), 1 + t - 2t^2 + 4t^3 - 12t^4 + \cdots\right] \tag{13}$$

To save space, the series for x_2 are truncated. Observe the quadratic convergence, typical for Newton's method.

The Maple instructions in the execution of Newton's method are of course not efficient. In practice one would fix the order of the power series and compute with the series directly. Adding and multiplying power series follows the normal rules of polynomials. To invert a power series, the identity $1/(1-x) = 1+x+x^2+\cdots$ is useful.

With the Newton-Hensel operator we extend the solutions for specific values of the parameters. This extension process is often called *Hensel lifting*. In the geometric resolution, the parameter t is a new variable, say x_3, as we lift the isolated points to a solution curve and then intersect the curve with the next equation f_3.

4 Straight-Line Programs and Complexity

Following [7], the encoding of polynomials as straight-line programs is defined as follows. For n indeterminates $x_1, x_2, \ldots, x_n$ with values in $\mathbb{Q}$ and $R \in \mathbb{N}$, a sequence $s = (q_1, q_2, \ldots, q_R)$ is *a straight-line program* (slp for short) if each q_k, for $k = 1, 2, \ldots, R$ satisfies the following two conditions:

1. $q_k \in \mathbb{Q} \cup \{ x_1, x_2, \ldots, x_n \}$, or
2. $\exists k_1, k_2 < k$ and $* \in \{+, -, \cdot, \div\}$: $q_k = q_{k_1} * q_{k_2}$.

We say that s is a *division-free* slp if $q_k = q_{k_1} \div q_{k_2} \Rightarrow q_{k_2} \in \mathbb{Q} \setminus \{0\}$.

The length of a slp indicates the cost to evaluate the polynomial it encodes. Polynomials with few monomials with nonzero coefficient have a short length, but not all short slp's are sparse. Consider for example $(x + y)^{1024}$.

To represent a solution set V with $\#V = D$, a geometric resolution needs polynomials of degree D. With a quadratically converging Newton-Hensel lifting about $\log_2(D)$ steps are needed to have D correct terms in the series. An important factor in the cost to compute a geometric resolution is the length of the slp's to evaluate the polynomials in the system.

Expressing the complexity of problems in symbolic computation in terms of the straight-line programs is very useful. Consider for example the determinant of a matrix. Expanding an n-by-n matrix of indeterminates results in a polynomial with $n!$ monomials. But with Gaussian elimination we can evaluate a determinant with $O(n^3)$ operations.

5 Forward and Reverse Modes

Consider for example

$$f(x_1, x_2, \ldots, x_{10}) = x_1 \times x_2 \times x_3 \times x_4 \times x_5 \times x_6 \times x_7 \times x_8 \times x_9 \times x_{10}. \tag{14}$$

It takes 9 multiplications to evaluate f. Suppose we want to evaluate its gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n})$. The symbolic computation of the gradient leads to expression swell:

$$\begin{aligned}
\tfrac{\partial f}{\partial x_1} &= x_2 \times x_3 \times x_4 \times x_5 \times x_6 \times x_7 \times x_8 \times x_9 \times x_{10} \\
\tfrac{\partial f}{\partial x_2} &= x_1 \times x_3 \times x_4 \times x_5 \times x_6 \times x_7 \times x_8 \times x_9 \times x_{10} \\
&\vdots \\
\tfrac{\partial f}{\partial x_{10}} &= x_1 \times x_2 \times x_3 \times x_4 \times x_5 \times x_6 \times x_7 \times x_8 \times x_9
\end{aligned} \tag{15}$$

Not only is the expression swell too expensive for memory, but it is also clear that using these expressions to evaluate the gradient is not the most efficient way to compute the gradient. This example (attributed to Speelpenning in [5]) motivated the reverse mode of algorithmic differentiation.

The code to evaluate both the function $x_1 \times x_2 \times \cdots \times x_n$ and its gradient is below:

$v_0 := 1$
for i from 1 to n do
  $v_i := v_{i-1} \times x_i$
$y := v_n$
$w_n := 1$
for i from n downto 1 do
  $z_i := w_i \times v_{i-1}$
  $w_{i-1} := w_i \times x_i$

The value of the function is returned in y and the components of the gradient are in the final values of z_i for i from 1 to n. The cost to evaluate both the function and its gradient is $3n$.

To verify whether the code above indeed computes what is needed, we trace the evaluation for $n = 3$, recording for each step the values of all variables in a table:

i	v_0	v_1	v_2	v_3	y	w_3	w_2	w_1	w_0	z_3	z_2	z_1
	1											
1	1	x_1										
2	1	x_1	x_1x_2									
3	1	x_1	x_1x_2	$x_1x_2x_3$								
	1	x_1	x_1x_2	$x_1x_2x_3$	$x_1x_2x_3$							
	1	x_1	x_1x_2	$x_1x_2x_3$	$x_1x_2x_3$	1						
3	1	x_1	x_1x_2	$x_1x_2x_3$	$x_1x_2x_3$	1	x_3			x_1x_2		
2	1	x_1	x_1x_2	$x_1x_2x_3$	$x_1x_2x_3$	1	x_3	x_3x_2		x_1x_2	x_1x_3	
1	1	x_1	x_1x_2	$x_1x_2x_3$	$x_1x_2x_3$	1	x_3	x_3x_2	$x_3x_2x_1$	x_1x_2	x_1x_3	x_3x_2

6 Exercises

1. Apply Newton's method to $h(t,x) = t^2 + x^2 - 1 = 0$ to compute a power series $x(t)$, starting with $x(0) = 1$, using Maple or Sage. Do three steps and verify the quadratic convergence. Compare the result with the Taylor series of $\sqrt{1-t^2}$ about $t = 0$.

2. As in the previous exercise, consider $h(t,x) = t^2 + x^2 - 1 = 0$. Instead of the conceptual execution of Newton's method, work directly with power series, fixing the order to $O(t^8)$, using Maple or Sage.

3. For the system in (8), to make the computation of the power series solution more efficient, do the following:

   (a) Instead of working with the symbolic inverse of the Jacobian matrix, describe the algorithm via solving a linear system to compute the update to the series solution.

   (b) Work out the more efficient version of the algorithm using Maple and Sage. Verify that it produces the same results on (8).

References

[1] C. Durvye. *Algorithmes pour la décomposition primaire des idéaux polynomiaux de dimension nulle donnés en évaluation*. PhD thesis, Université de Versailles - Saint-Quentin, Yvelines, Paris, 2008.

[2] I.Z. Emiris. *Sparse Elimination and Applications in Kinematics*. PhD thesis, Computer Science Division, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, 1994.

[3] O.D. Faugeras and S. Maybank. Motion from point matches: multiplicity of solutions. *International Journal of Computer Vision*, 4(3):225–246, 1990.

[4] M. Giusti, G. Lecerf, and B. Salvy. A Gröbner free alternative for polynomial system solving. *J. Complexity*, 17(1):154–211, 2001.

[5] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2000.

[6] T. Mora. *Solving Polynomial Equation Systems I: the Kronecker-Duval Philosophy*, volume 88 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 2003.

[7] J. Sabia. Algorithms and their complexities. In *Solving Polynomial Equations. Foundations, Algorithms and Applications*, volume 14 of *Algorithms and Computation in Mathematics*, pages 241–268. Springer–Verlag, 2005.