Kronecker Representation

1. Geometric Resolution
   - representing a finite set of solutions

2. A Problem in Computer Vision
   - camera motion from point matches

3. The Newton-Hensel Method
   - lifting a solution
   - computing with power series

4. Straight-Line Programs (slp)
   - expressing complexity in term of slps
   - forward and reverse modes

MCS 563 Lecture 14
Analytic Symbolic Computation
Jan Verschelde, 14 February 2014
Kronecker Representation

1. Geometric Resolution
   - representing a finite set of solutions

2. A Problem in Computer Vision
   - camera motion from point matches

3. The Newton-Hensel Method
   - lifting a solution
   - computing with power series

4. Straight-Line Programs (slp)
   - expressing complexity in term of slps
   - forward and reverse modes
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 $\ell(x) = u_0 + u_1 x_1 + u_2 x_2 + \cdots + u_n x_n$,

2. polynomials $q_1, p_1, p_2, \ldots, p_n \in \mathbb{Q}[T]$ such that
   
   1. for all $z_j, z_k \in V, j \neq k$: $\ell(z_j) \neq \ell(z_k)$,
   
   2. $q(T) = \prod_{i=1}^{D} (T - \ell(z_i))$,

   3. for $j = 1, 2, \ldots, n$: $\deg(p_j) \leq D - 1$ and

   $V = \{ (p_1(r), p_2(r), \ldots, p_n(r)) \mid r \in \mathbb{C} : q(r) = 0 \}$.

We speak of the geometric resolution of $V$ associated to $\ell$.

$\ell$ works as in the rational univariate representation.
Consider
\[ f(x_1, x_2) = \begin{cases} x_1^2 + (x_2 - 1)^2 - 1 = 0 \\ x_1^2 - x_2 = 0. \end{cases} \]

4 solutions, counted with multiplicity, \((0, 0)\) is double.

We see that \(x_1\) separates the zeroes, but that \(x_2\) does not.

For \(\ell = x_1\), the geometric resolution of \(V = f^{-1}(0)\) is
\[ \ell(x) = x_1, q(T) = T^2(T - 1)(T + 1), p_1(T) = T, p_2(T) = T^2. \]

The geometric interpretation of this solution corresponds to looking at the algebraic curves defined by \(f_1\) and \(f_2\), intersected with the line \(x_1 = T\).
using resultants

Consider \( f_1 = x_1^2 + (x_2 - 1)^2 - 1 = 0 \), 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:

\[
> \text{resultant}(T^2 + (x-1)^2 - 1, T^2 - x,x);
\]

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

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

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

1. Geometric Resolution
   - representing a finite set of solutions

2. A Problem in Computer Vision
   - camera motion from point matches

3. The Newton-Hensel Method
   - lifting a solution
   - computing with power series

4. Straight-Line Programs (slp)
   - expressing complexity in term of slps
   - forward and reverse modes
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 $a$ and $a'$ in their respective coordinate frames $(x, y, z)$ and $(x', y', z')$, via a rotation $R$ and translation $t$.

The condition that the images come from the same point seen from two different camera positions is equivalent to the coplanarity of the vectors $a, t, \text{ and } Ra' + t$, expressed by

$$a_i^T(t \times (Ra'_i + t)) = a_i^T(t \times Ra'_i) = 0, \quad \text{for } i = 1, 2, \ldots, 5,$$

where $\times$ denotes the cross product. For 5 points, the number of solutions is finite and equals 10.
Single point seen by two camera positions:

![Diagram of single point seen by two camera positions](image-url)
Using a quaternion formulation, we get a system of 5 equations in 6 unknowns, two 3-vectors \( q \) and \( d \):

\[
\begin{align*}
1 - d^T q &= 0 \\
(a_i^T q)(d^T a'_i) + a_i^T a'_i + (a_i \times q)^T a'_i \\
+ (a_i \times q)^T (d \times a'_i) + a_i (d \times a'_i) &= 0, \quad i = 1, 2, \ldots, 5.
\end{align*}
\]

This system is bilinear in \( q \) and \( d \) and has a 2-homogeneous Bézout bound of 20. All solutions can be real.
Kronecker Representation

1. Geometric Resolution
   - representing a finite set of solutions

2. A Problem in Computer Vision
   - camera motion from point matches

3. The Newton-Hensel Method
   - lifting a solution
   - computing with power series

4. Straight-Line Programs (slp)
   - expressing complexity in term of slps
   - forward and reverse modes
the Hensel Lemma

Lemma (Hensel)

Consider \( f_1, f_2, \ldots, f_n \in \mathbb{Q}[t, x] \), \( t = (t_1, t_2, \ldots, t_m) \), \( x = (x_1, x_2, \ldots, x_n) \) and \( J_f \) is the Jacobian matrix of \( f = (f_1, f_2, \ldots, f_n) \) with respect to \( x \).

Given \( (\tau, z) \in \mathbb{C}^m \times \mathbb{C}^n \) such that \( f(\tau, z) = 0 \) and \( \det(J_f(\tau, z)) \neq 0 \).

Then there is a unique \( n \)-tuple of formal power series \( S \in (\mathbb{C}[[t - \tau]])^n \) such that

1. \( S(\tau) = z \), and
2. \( f(t, S) = 0 \).
inverting power series

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

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

truncating the infinite series at some order $d$.

If we are computing a polynomial of degree $d$, once we have $d$ terms of the expansion, the truncation is without error.
The proof of the lemma uses the Newton-Hensel operator:

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

By the assumption \( \det(J_f(\tau, z)) \neq 0 \), the matrix \( J_f(t, x) \) is invertible at \( x = z \) for \( t = \tau \).

If we set \( S^{(0)} = z \), 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 \( O((t - \tau)^p) \) then the error term in \( S^{(k+1)} \) is \( O((t - \tau)^{2p}) \).
Kronecker Representation

1. Geometric Resolution
   - representing a finite set of solutions

2. A Problem in Computer Vision
   - camera motion from point matches

3. The Newton-Hensel Method
   - lifting a solution
   - computing with power series

4. Straight-Line Programs (slp)
   - expressing complexity in term of slps
   - forward and reverse modes
an example in Maple

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

where for \( t = 0 \) we have the solution \( (x_1 = 1, x_2 = 1) \).

\[
    f := [x[1]^2+(x[2]-1)^2-1+t^2, x[1]^2-x[2]+t]:
\]

\[
    J := Matrix([seq([seq(diff(f[i],x[j]),j=1..2)],
                   i=1..2))]):
\]

\[
    Jinv := J^(-1);
\]

\[
    sNf := Vector([x[1],x[2]]) - Jinv.Vector(f):
\]

\[
    newton := (a,b) \rightarrow \text{subs}(x[1]=a,x[2]=b,sNf):
\]

\[
    X[0] := [1,1]:
\]

for \( k \) from 1 to 4 do
    \[
    X[k] := \text{newton}(X[k-1][1],X[k-1][2]);
    \]
    \[
    s[k] := [\text{series}(X[k][1],t=0,8),\text{series}(X[k][2],
                   t=0,8)];
    \]
end do:
quadratic convergence

\[ S^{(0)} = [1, 1] \]
\[ S^{(1)} = \left[ 1 - \frac{1}{2} t^2, 1 + t - t^2 \right] \]
\[ S^{(2)} = \left[ 1 - t^2 + 2 t^3 - \frac{47}{8} t^4 + 16 t^5 - \frac{703}{16} t^6 + 120 t^7 + O(t^8), \right. \]
\[ \left. 1 + t - 2 t^2 + 4 t^3 - 11 t^4 + \cdots \right] \]
\[ S^{(3)} = \left[ 1 - t^2 + 2 t^3 - \frac{13}{2} t^4 + 22 t^5 - \frac{161}{2} t^6 + 307 t^7 + O(t^8), \right. \]
\[ \left. 1 + t - 2 t^2 + 4 t^3 - 12 t^4 + \cdots \right] \]
Kronecker Representation

1. Geometric Resolution
   • representing a finite set of solutions

2. A Problem in Computer Vision
   • camera motion from point matches

3. The Newton-Hensel Method
   • lifting a solution
   • computing with power series

4. Straight-Line Programs (slp)
   • expressing complexity in term of slps
   • forward and reverse modes
We encode polynomials as straight-line 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 $(x + y)^{1024}$.

Expressing the complexity of problems in symbolic computation in terms of the straight-line programs is useful.

Consider for example the determinant of a matrix. Expanding an $n$-by-$n$ matrix of indeterminates results in a polynomial with $n!$ monomials. With Gaussian elimination we evaluate a determinant with $O(n^3)$ operations.
Kronecker Representation

1. Geometric Resolution
   - representing a finite set of solutions

2. A Problem in Computer Vision
   - camera motion from point matches

3. The Newton-Hensel Method
   - lifting a solution
   - computing with power series

4. Straight-Line Programs (slp)
   - expressing complexity in term of slps
   - forward and reverse modes
the example of Speelpenning

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

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:

\[
\frac{\partial f}{\partial x_1} = x_2 \times x_3 \times x_4 \times x_5 \times x_6 \times x_7 \times x_8 \times x_9 \times x_{10} \\
\frac{\partial f}{\partial x_2} = x_1 \times x_3 \times x_4 \times x_5 \times x_6 \times x_7 \times x_8 \times x_9 \times x_{10} \\
\vdots \\
\frac{\partial f}{\partial x_{10}} = x_1 \times x_2 \times x_3 \times x_4 \times x_5 \times x_6 \times x_7 \times x_8 \times x_9
\]

Not only is the expression swell too expensive for memory, but using these expressions to evaluate the gradient is not the most efficient way to compute the gradient.
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:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$v_0$</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$x_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$x_1$</td>
<td>$x_1 x_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$x_1$</td>
<td>$x_1 x_2$</td>
<td>$x_1 x_2 x_3$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>$x_1$</td>
<td>$x_1 x_2$</td>
<td>$x_1 x_2 x_3$</td>
<td>$x_1 x_2 x_3$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>$w_3$</th>
<th>$w_2$</th>
<th>$w_1$</th>
<th>$w_0$</th>
<th>$z_3$</th>
<th>$z_2$</th>
<th>$z_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$x_3$</td>
<td></td>
<td></td>
<td>$x_1 x_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$x_3$</td>
<td>$x_3 x_2$</td>
<td></td>
<td>$x_1 x_2$</td>
<td>$x_1 x_3$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$x_3$</td>
<td>$x_3 x_2$</td>
<td>$x_3 x_2 x_1$</td>
<td>$x_1 x_2$</td>
<td>$x_1 x_3$</td>
<td>$x_3 x_2$</td>
</tr>
</tbody>
</table>
Summary + Exercises

Newton’s method on power series is a symbolic method to produce approximate results.

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.
one more exercise

For the system

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

to make the computation of the power series solution more efficient, do the following:

1. 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.

2. Work out the more efficient version of the algorithm using Maple and Sage. Verify that it produces the same results.