Reverse engineering using computational algebra

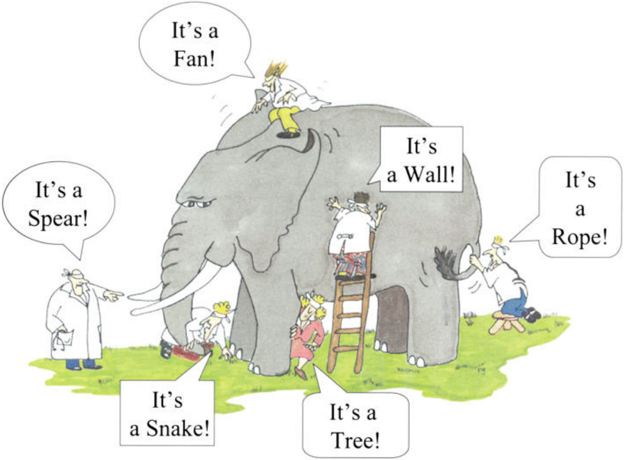
Matthew Macauley

Department of Mathematical Sciences
Clemson University
http://www.math.clemson.edu/~macaule/

Math 4500, Spring 2015

The blind men and the elephant

An old parable from India tells of several blind men who try to determine what an elephant looks like just by touch.



The blind men are trying to reverse engineer an elephant from just a few data points.

Inferring a Boolean network model (elephant) from data (observations)

Consider a Boolean network model on n nodes, with update function $F\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$. There are 2^n input states.

Suppose we don't know the actual function F, but through experimental data, we are able to observe several transitions:

$$\begin{array}{ccccccc}
\mathbf{s}_1 = (s_{11}, s_{12}, \dots, s_{1n}) & & \mathbf{s}_2 = (s_{21}, \dots, s_{2n}) & & \cdots & & \mathbf{s}_m = (s_{m1}, \dots, s_{mn}) \\
\downarrow & & \downarrow & & & & \downarrow \\
\mathbf{t}_1 = (t_{11}, t_{12}, \dots, t_{1n}) & & \mathbf{t}_2 = (t_{21}, \dots, t_{2n}) & & \cdots & & \mathbf{t}_m = (t_{m1}, \dots, t_{mn})
\end{array}$$

Reverse engineering

Start with experimental data (observations) and reconstruct the model (elephant). The two main features are:

(i) the network topology, or wiring diagram,

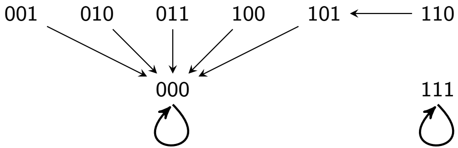
(ii) the Boolean functions at each node: $F = (f_1, \dots, f_n)$.

Inferring a Boolean network model (elephant) from data (observations)

Consider the following polynomial dynamical system:

$$
\begin{aligned}
f_1(x_1, x_2, x_3) &= \mathrm{AND}(x_1, x_2) = x_1 x_2 \\
f_2(x_1, x_2, x_3) &= \mathrm{AND}(x_1, x_2, x_3) = x_1 x_2 x_3 \\
f_3(x_1, x_2, x_3) &= \mathrm{AND}(x_1, x_2) = x_1 x_2 \,.
\end{aligned}
$$

The state space of the FDS map $F = (f_1, f_2, f_3)$ is the following graph:



Question

What if we only knew part of this state space, e.g.,

$$(1,1,0) \longrightarrow (1,0,1) \longrightarrow (0,0,0) \longrightarrow (0,0,0)\,.$$

Could we recover the individual functions? How many possible models could yield this “fragment”?

Reverse engineering

Broad goal

Find "the best" model $F = (f_1, \ldots, f_n)$ that fits the data:

$$\begin{array}{l} \text{Input states: } \mathbf{s}_1, \ldots, \mathbf{s}_m \in \mathbb{F}^n \\ \text{Output states: } \mathbf{t}_1, \ldots, \mathbf{t}_m \in \mathbb{F}^n \end{array} \qquad \text{with } F(\mathbf{s}_i) = \mathbf{t}_i$$

Note that: $F(\mathbf{s}_i) = (f_1(\mathbf{s}_i), f_2(\mathbf{s}_i), \ldots, f_n(\mathbf{s}_i)) = (t_{i1}, t_{i2}, \ldots, t_{in}) = \mathbf{t}_i$.

Question

What if no models fit the data?

What if many models fit the data? (This is more likely.)

First, we'll find all models that fit the data. This is called the model space:

$$F_1 \times F_2 \times \cdots \times F_n = \{(f_1, \ldots, f_n) \mid f_j(\mathbf{s}_i) = t_{ij} \text{ for all } i \text{ and } j\}.$$

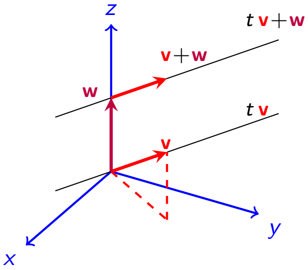
Once we do this, the new problem becomes choosing the "best" one. This is called model selection. We will not discuss this problem.

Similar problems in other areas of mathematics

1. Parametrize a line in $\mathbb{R}^n$.
2. Parametrize a plane in $\mathbb{R}^n$.
3. Solve the underdetermined system $\mathbf{Ax} = \mathbf{b}$.
4. Solve the differential equation $x'' + x = 2$.

Parametrize a line in $\mathbb{R}^n$

Suppose we want to write the equation for a line that contains a vector $\mathbf{v} \in \mathbb{R}^n$:



This line, which *contains the zero vector*, is $t\mathbf{v} = \{t\mathbf{v} : t \in \mathbb{R}\}$.

Now, what if we want to write the equation for a line parallel to $\mathbf{v}$?

This line, which *does not contain the zero vector*, is

$$t\mathbf{v} + \mathbf{w} = \{t\mathbf{v} + \mathbf{w} : t \in \mathbb{R}\}.$$

Note that ANY particular $\mathbf{w}$ on the line will work!!!

Solve an underdetermined system $\mathbf{Ax} = \mathbf{b}$

Suppose we have a system of equations that has "too many variables," so there are infinitely many solutions.

For example:

$$\begin{aligned} 2x + 3y - 6z &= 3 \\ 3x - 4y + 3z &= 1 \end{aligned} \qquad \text{"}\mathbf{Ax} = \mathbf{b}\text{ form"}: \qquad \begin{bmatrix} 2 & 3 & -6 \\ 3 & -4 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

How to solve:

1. Solve the related homogeneous equation $\mathbf{Ax} = \mathbf{0}$ (this is null space, $\mathrm{NS}(A)$);
2. Find any particular solution $\mathbf{x}_p$ to $\mathbf{Ax} = \mathbf{b}$;
3. Add these together to get the general solution: $\mathbf{x} = \mathrm{NS}(A) + \mathbf{x}_p$.

This works because geometrically, the solution space is just a line, plane, etc.

Linear differential equations

Solve the differential equation $x'' + x = 2$.

How to solve:

1. Solve the related homogeneous equation $x'' + x = 0$. The solutions are $x_h(t) = a\cos t + b\sin t$.
2. Find any particular solution $x_p(t)$ to $x'' + x = 2$. By inspection, we see that $x_p(t) = 2$ works.
3. Add these together to get the general solution:

$$x(t) = x_h(t) + x_p(t) = a\cos t + b\sin t + 2.$$

Reverse engineering: Problem statement

Definition

A finite dynamical system (FDS) is a function $F = (f_1, \dots, f_n)\colon X^n \to X^n$ where each $f_i\colon X^n \to X$ is a local function and $|X| < \infty$ (usually $X = \mathbb{F}_2 = \{0, 1\}$).

Key fact

If $X = \mathbb{F}$ is a finite field (e.g., $\mathbb{Z}_2$, $\mathbb{Z}_3$, $\mathbb{Z}_p$, etc.), then every function $f_i\colon \mathbb{F}^n \to \mathbb{F}$ is a polynomial in $x_1, \dots, x_n$.

Goal

Given a set of data:

Input states: $\mathbf{s}_1, \dots, \mathbf{s}_m \in \mathbb{F}^n$

Output states: $\mathbf{t}_1, \dots, \mathbf{t}_m \in \mathbb{F}^n$

with $F(\mathbf{s}_i) = \mathbf{t}_i$

Construct the model space $F_1 \times \cdots \times F_n$ of all models $F = (f_1, \dots, f_n)$ that fit the data:

$$F(\mathbf{s}_i) = (f_1(\mathbf{s}_i), \dots, f_n(\mathbf{s}_i)) = (t_{i1}, \dots, t_{in}) = \mathbf{t}_i.$$

We'll find each $F_1, \dots, F_n$ separately.

Reverse engineering: How to find F_j

We wish to find the set F_j of all local functions (polynomials!) f_j that fit the data:

$$F_j = \{f_j : f_j(\mathbf{s}_1) = t_{1j},\ \dots\ , f_j(\mathbf{s}_m) = t_{mj}\}\,.$$

Define the set I (it is actually an "ideal" of the polynomial ring $\mathbb{F}[x]$)

$$\begin{aligned} I &= \{h : h(\mathbf{s}_i) = 0 \text{ for all } i = 1, \dots, m\} \\ &= \{\text{all polynomials that vanish on the data}\}. \end{aligned}$$

Theorem

The set of polynomials that fit the data at node j is

$$F_j = f_j + I = \{f_j + h : h \in I\}\,,$$

where f_j is any one particular polynomial that fits the data.

Thus, to find F_j, we need to do two things:

1. Find the ideal I; (*all* solutions to $\{f_j(\mathbf{s}_i) = 0\ \forall i\}$)
2. Find *any* polynomial f_j that fits the data. (*one* solution to $\{f_j(\mathbf{s}_i) = t_{ij}\ \forall i\}$)

Reverse engineering: How to find I and f_j

1. *Finding I*: Define $I(\mathbf{s}_i)$ to be the set of polynomials that vanish on $\mathbf{s}_i$:

$$\begin{aligned} I(\mathbf{s}_i) &= \{\text{all polynomials } h_i \text{ such that } h_i(\mathbf{s}_i) = 0\} \\ &= \{(x_1 - s_{i1})g_1(\mathbf{x}) + (x_2 - s_{i2})g_2(\mathbf{x}) + \cdots + (x_n - s_{in})g_n(\mathbf{x})\} \\ &= \langle x_1 - s_{i1}, x_2 - s_{i2}, \dots, x_n - s_{in} \rangle \end{aligned}$$

   Clearly, the set I of polynomials that vanish on all $\mathbf{s}_i$ (for $i = 1, \dots, m$) is

$$I = \bigcap_{i=1}^{m} I(\mathbf{s}_i)\,.$$

2. *Finding f_j*: There are many algorithms. Lagrange interpolation is one of them.

   In this lecture, we will learn another method, and do a hands-on example.

   We'll get started with this now.

Finding f_j (one method)

For each data point $\mathbf{s}_i$ ($i = 1, \dots, m$), we'll construct an r-polynomial that has the following property:

$$r_i(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} = \mathbf{s}_i \\ 0 & \mathbf{x} \neq \mathbf{s}_i \end{cases}$$

Once we have these, the polynomial $f_j(\mathbf{x})$ we seek will be

$$f_j(\mathbf{x}) = t_{1j} r_1(\mathbf{x}) + t_{2j} r_2(\mathbf{x}) + \cdots + t_{mj} r_m(\mathbf{x})\,.$$

One way to construct the r-polynomials:

$$r_i(\mathbf{x}) = \prod_{\substack{k=1 \\ k \neq i}}^{m} b_{ik}(\mathbf{x})\,,$$

where

$$b_{ik}(\mathbf{x}) = (s_{i\ell} - s_{k\ell})^{p-2}(x_\ell - s_{k\ell})$$

and ℓ is the *first* coordinate in which $\mathbf{s}_i$ and $\mathbf{s}_k$ differ.

An example

Consider the following time series in a 3-node system over $\mathbb{Z}_5$:

$$
\begin{array}{c}
\mathbf{s}_1 = (2,0,0) \\
\downarrow \\
\mathbf{s}_2 = (4,3,1) = \mathbf{t}_1 \\
\downarrow \\
\mathbf{s}_3 = (3,1,4) = \mathbf{t}_2 \\
\downarrow \\
(0,4,3) = \mathbf{t}_3
\end{array}
$$

For reference, here are the input vectors $\mathbf{s}_i$ and output vectors $\mathbf{t}_i$:

$$
\begin{aligned}
\mathbf{s}_1 &= (s_{11}, s_{12}, s_{13}) = (2,0,0)\,, \qquad & \mathbf{t}_1 &= (t_{11}, t_{12}, t_{13}) = (4,3,1)\,, \\
\mathbf{s}_2 &= (s_{21}, s_{22}, s_{23}) = (4,3,1)\,, \qquad & \mathbf{t}_2 &= (t_{21}, t_{22}, t_{23}) = (3,1,4)\,, \\
\mathbf{s}_3 &= (s_{31}, s_{32}, s_{33}) = (3,1,4)\,, \qquad & \mathbf{t}_3 &= (t_{31}, t_{32}, t_{33}) = (0,4,3)\,.
\end{aligned}
$$

Note that $\mathbf{s}_1$ differs from $\mathbf{s}_2$ and $\mathbf{s}_3$ in the $\ell = 1$ coodinate, so this ℓ will work for each of r_1, r_2, and r_3.

An example: computing the r-polynomials

Since we are working in $\mathbb{Z}_5$, we are taking the remainder of everything modulo 5.

Particularly useful identities are: $0 = 5$, $-1 = 4$, $-2 = 3$, $-3 = 2$, and $-4 = 1$.

Using our formulas for $b_{ij}(\mathbf{x})$, we compute:

$$b_{12}(\mathbf{x}) = (s_{11} - s_{21})^3(x_1 - s_{21}) = (2-4)^3(x_1 - 4) = -8(x_1 + 1) = 2x_1 + 2$$
$$b_{13}(\mathbf{x}) = (s_{11} - s_{31})^3(x_1 - s_{31}) = (2-3)^3(x_1 - 3) = -x_1 + 3 = 4x_1 + 3\,.$$

Therefore, the first r-polynomial is

$$r_1(\mathbf{x}) = b_{12}(\mathbf{x})b_{13}(\mathbf{x}) = (2x_1 + 2)(4x_1 + 3) = 8x_1^2 + 14x_1 + 6 = 3x_1^2 + 4x_1 + 1\,.$$

In-class Exercise

Compute the other two r-polynomials in this example: $r_2(\mathbf{x})$ and $r_3(\mathbf{x})$.

Solution: $r_2(\mathbf{x}) = 3x_1^2 + 3\,,\qquad r_3(\mathbf{x}) = 4x_1^2 + x_1 + 2.$

An example: computing the ideal I

Recall that the ideal I is the set I of polynomials that vanish on all $\mathbf{s}_i$:

$$I = I(\mathbf{s}_1) \cap I(\mathbf{s}_2) \cap I(\mathbf{s}_3)\,,$$

where

$$\mathbf{s}_1 = (2,0,0)\,, \qquad \mathbf{s}_2 = (4,3,1)\,, \qquad \mathbf{s}_3 = (3,1,4)\,.$$

These are precisely the sets

$$\begin{aligned}
I(\mathbf{s}_1) &= \langle x_1 - 2, x_2, x_3 \rangle = \{(x_1-2)g_1(x) + x_2 g_2(x) + x_3 g_3(x)\} \\
I(\mathbf{s}_2) &= \langle x_1 - 4, x_2 - 3, x_3 - 1 \rangle = \{(x_1-4)g_1(x) + (x_2-3)g_2(x) + (x_3-1)g_3(x)\} \\
I(\mathbf{s}_3) &= \langle x_1 - 3, x_2 - 1, x_3 - 4 \rangle = \{(x_1-3)g_1(x) + (x_2-1)g_2(x) + (x_3-4)g_3(x)\}
\end{aligned}$$

A computer algebra system (e.g., Sage or Macaulay2) can easily compute the intersection of these ideals.

Usually it will return the ideal I by specifying a generating set.

An example: finding the model space

Now that we have all of the pieces, f_1, f_2, and f_3 can be computed easily:

$$f_j(\mathbf{x}) = t_{1j}r_1(\mathbf{x}) + t_{2j}r_2(\mathbf{x}) + \cdots + t_{mj}r_m(\mathbf{x})\,.$$

Our "particular" solution that fits the data is $f = (f_1, f_2, f_3)$, and our "general solution" (the model space) is the set

$$\begin{aligned} F_1 \times \cdots \times F_n &= f + (I \times \cdots \times I) \\ &= (f_1 + I, \ldots, f_n + I)\,. \end{aligned}$$