

Reverse engineering using computational algebra

Matthew Macauley

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

Math 4500, Fall 2017

What is reverse engineering?

Sometimes, complex biological systems can seem a bit like this: ([click here!](#)).

Systems biology is the study of systems of biological components.

A central problem in systems biology is to use experimental data to infer the structure of a system such as a gene regulatory network.

Modeling approaches

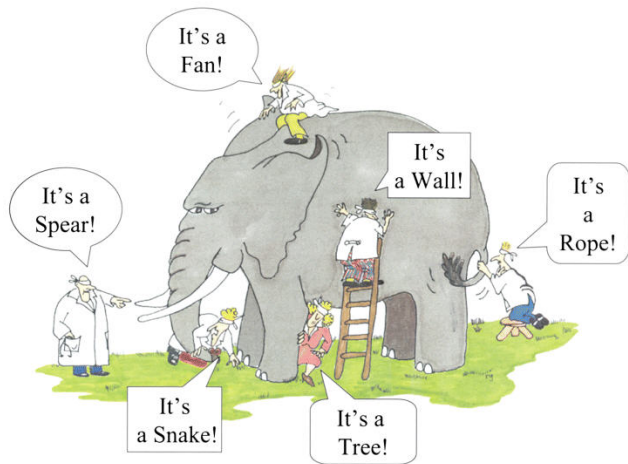
- **Bottom-up:** Build a network from the known local information about every single object.
- **Top-down** (“[Reverse-engineering](#)”): View the system as a black box, then use the available data to make a model.

Previously, we’ve mostly studied the first approach to modeling. In this lecture, we’ll focus on the second approach.

Many problems in statistics (e.g., linear regression) deal with the second approach.

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.

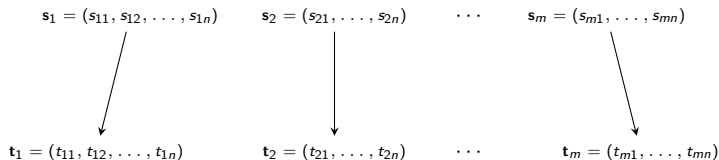


The blind men are trying to **reverse engineer** an elephant from just a few data points.

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

Consider a Boolean network on n nodes, with update function $F: \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$. 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:



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 (elephant) from data (observations)

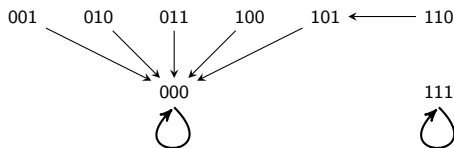
Consider the following Boolean network:

$$f_1(x_1, x_2, x_3) = x_1 \wedge x_2 = x_1 x_2$$

$$f_2(x_1, x_2, x_3) = x_1 \wedge x_2 \wedge x_3 = x_1 x_2 x_3$$

$$f_3(x_1, x_2, x_3) = x_1 \wedge x_2 = x_1 x_2 .$$

The **state space** of $f = (f_1, f_2, f_3)$ 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 the model space

Broad goal

Find “the best” **local model** $f = (f_1, \dots, f_n)$ that fits the 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 $F(\mathbf{s}_i) = \mathbf{t}_i$

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

Question

What if no models fit the data? (This is actually impossible.)

What if many models fit the data?

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

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

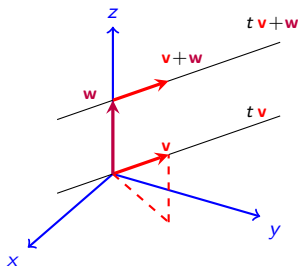
Once we do this, the new problem becomes choosing the “best” one. This is called **model selection**.

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:

$$\begin{aligned} 2x + y + 3z &= 4 \\ 3x - 5y - 2z &= 6 \end{aligned} \quad \text{“}\mathbf{Ax} = \mathbf{b}\text{ form”}: \quad \begin{bmatrix} 2 & 1 & 3 \\ 3 & -5 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}.$$

How to solve:

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

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

Here are two possible ways to write the solution:

$$c \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}, \quad c \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 10 \\ 8 \\ -8 \end{bmatrix}.$$

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**:

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

Note that while the general solution above is unique, its presentation need not be.

For example, we could write it this way:

$$x(t) = x_h(t) + x_p(t) = a(2 \cos t - 3 \sin t) + b \sin t + (2 - \cos t + 8 \sin t).$$

Here, the **particular solution** has (unnecessary) “extra terms” that vanish on the homogeneous part, $x'' + x = 0$.

Reverse engineering: Problem statement

Recall that a **local model over \mathbb{F}** is an n -tuple $f = (f_1, \dots, f_n)$ of functions $f_i: \mathbb{F}^n \rightarrow \mathbb{F}$.

The associated finite dynamical system (FDS) map is

$$f: \mathbb{F}^n \longrightarrow \mathbb{F}^n, \quad f: \mathbf{x} \longmapsto (f_1(\mathbf{x}), \dots, f_n(\mathbf{x})).$$

If $\mathbb{F} = \mathbb{F}_p$ then each $f_i: \mathbb{F}_p^n \rightarrow \mathbb{F}_p$ is a **polynomial** in $\mathbb{F}_p[x_1, \dots, x_n]/\langle x_1^p - x_1, \dots, x_n^p - x_n \rangle$.

Goal

Given a set of data:

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

Construct the **model space** $F_1 \times \dots \times F_n$ of **all local 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 $\mathbb{F}[x_1, \dots, x_n]$)

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

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 which has the **Chinese remainder theorem** lurking behind the scenes.

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. (Actually, any coordinate where they differ will do.)

Remark

When our systems are Boolean (over \mathbb{F}_2), then this reduces to $b_{ik}(\mathbf{x}) = x_\ell - s_{k\ell}$.

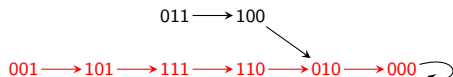
A Boolean example

Consider the following model of the *lac* operon, which implicitly assumes that A degrades slower than M or B .

$$\begin{cases} f_M = x_A \\ f_B = x_M \\ f_A = L \vee (B \wedge L_m) \vee (A \wedge \overline{B}). \end{cases}$$

If lactose levels are low, then $L = L_m = 0$, and this model reduces to the one at left (left) with state space (right):

$$\begin{cases} f_1 = x_3 \\ f_2 = x_1 \\ f_3 = (x_2 + 1)x_3. \end{cases}$$



Exercise

Let's **reverse engineer** the Boolean network from just knowing the 6 red nodes and 5 transitions.

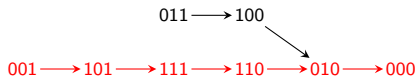
In other words, find **all** triples of polynomials $f = (h_1, h_2, h_3)$ such that:

- $f(0, 0, 1) = (h_1(0, 0, 1), h_2(0, 0, 1), h_3(0, 0, 1)) = (1, 0, 1)$,
- $f(1, 0, 1) = (h_1(1, 0, 1), h_2(1, 0, 1), h_3(1, 0, 1)) = (1, 1, 1)$,
- $f(1, 1, 1) = (h_1(1, 1, 1), h_2(1, 1, 1), h_3(1, 1, 1)) = (1, 1, 0)$,
- $f(1, 1, 0) = (h_1(1, 1, 0), h_2(1, 1, 0), h_3(1, 1, 0)) = (0, 1, 0)$,
- $f(0, 1, 0) = (h_1(0, 1, 0), h_2(0, 1, 0), h_3(0, 1, 0)) = (0, 0, 0)$,

A Boolean example (cont.)

Since we know the original functions *a priori*, we secretly know the answer to this.

$$(f_1, f_2, f_3) = (x_3, x_1, (x_2 + 1)x_3).$$



The ideal of polynomials that vanish on each s_k is:

$$\begin{aligned} I_1 &= \text{ideal}(x_1, x_2, x_3-1); \\ I_2 &= \text{ideal}(x_1-1, x_2, x_3-1); \\ I_3 &= \text{ideal}(x_1-1, x_2-1, x_3-1); \\ I_4 &= \text{ideal}(x_1-1, x_2-1, x_3); \\ I_5 &= \text{ideal}(x_1, x_2-1, x_3); \end{aligned}$$

The ideal of polynomials that vanish on *every* s_k is:

$$I = \text{intersect}\{I_1, I_2, I_3, I_4, I_5\};$$

Thus, the complete **model space** is

$$F_1 \times F_2 \times F_3, \quad F_j = f_j + I.$$

If we hadn't known f_1, f_2, f_3 *a priori*, then we'd have to find our own particular solution that fits the data. We'll do that now.

A Boolean example (cont.)

We're looking for a single solution $f = (f_1, f_2, f_3)$ that fits the data.

We know how to do this. For example:

$$\begin{aligned}f_1(\mathbf{x}) &= t_{11}r_1(\mathbf{x}) + t_{21}r_2(\mathbf{x}) + t_{31}r_3(\mathbf{x}) + t_{41}r_4(\mathbf{x}) + t_{51}r_5(\mathbf{x}) \\ &= 1r_1(\mathbf{x}) + 1r_2(\mathbf{x}) + 1r_3(\mathbf{x}) + 0r_4(\mathbf{x}) + 0r_5(\mathbf{x}) = r_1(\mathbf{x}) + r_2(\mathbf{x}) + r_3(\mathbf{x}),\end{aligned}$$

where

$$r_1(\mathbf{x}) = \prod_{\substack{k=1 \\ k \neq 1}}^5 b_{1k}(\mathbf{x}) = b_{12}(\mathbf{x})b_{13}(\mathbf{x})b_{14}(\mathbf{x})b_{15}(\mathbf{x}).$$

$$\mathbf{s}_1 = (0, 0, 1)$$



$$\mathbf{s}_2 = (\underline{1}, 0, 1) = \mathbf{t}_1$$



$$\mathbf{s}_3 = (\underline{1}, 1, 1) = \mathbf{t}_2$$



$$\mathbf{s}_4 = (\underline{1}, 1, 0) = \mathbf{t}_3$$



$$\mathbf{s}_5 = (0, \underline{1}, 0) = \mathbf{t}_4$$



$$(0, 0, \underline{0}) = \mathbf{t}_5$$

Recall that $b_{1k}(\mathbf{x}) = x_\ell - s_{k\ell}$, where ℓ is the first coordinate that \mathbf{s}_1 differs from \mathbf{s}_k .

skip $k = 1$

$$b_{12}(\mathbf{x}) = (x_1 - s_{21}) = x_1 + 1$$

$$b_{13}(\mathbf{x}) = (x_1 - s_{31}) = x_1 + 1$$

$$b_{14}(\mathbf{x}) = (x_1 - s_{41}) = x_1 + 1$$

$$b_{15}(\mathbf{x}) = (x_2 - s_{52}) = x_2 + 1$$

Now, $r_1(\mathbf{x}) = (x_1 + 1)^3(x_2 + 1) = (x_1 + 1)(x_2 + 1)$.

A Boolean example (cont.)

Recall that $b_{ik}(\mathbf{x}) = x_\ell - s_{k\ell}$, where ℓ is the first coordinate that \mathbf{s}_i differs from \mathbf{s}_k .

$$\mathbf{s}_1 = (0, 0, 1)$$



$$\mathbf{s}_2 = (1, 0, 1) = \mathbf{t}_1$$



$$\mathbf{s}_3 = (1, 1, 1) = \mathbf{t}_2$$



$$\mathbf{s}_4 = (1, 1, 0) = \mathbf{t}_3$$



$$\mathbf{s}_5 = (0, 1, 0) = \mathbf{t}_4$$



$$(0, 0, 0) = \mathbf{t}_5$$

$$b_{21}(\mathbf{x}) = (x_1 - s_{11}) = x_1$$

skip $k = 2$

$$b_{23}(\mathbf{x}) = (x_2 - s_{32}) = x_2 + 1$$

$$b_{24}(\mathbf{x}) = (x_2 - s_{42}) = x_2 + 1$$

$$b_{25}(\mathbf{x}) = (x_1 - s_{51}) = x_1$$

$$b_{31}(\mathbf{x}) = (x_1 - s_{11}) = x_1$$

$$b_{32}(\mathbf{x}) = (x_2 - s_{22}) = x_2$$

skip $k = 3$

$$b_{34}(\mathbf{x}) = (x_3 - s_{43}) = x_3$$

$$b_{35}(\mathbf{x}) = (x_1 - s_{51}) = x_1$$

$$b_{41}(\mathbf{x}) = (x_1 - s_{11}) = x_1$$

$$b_{42}(\mathbf{x}) = (x_2 - s_{22}) = x_2$$

$$b_{43}(\mathbf{x}) = (x_3 - s_{33}) = x_3 + 1$$

skip $k = 4$

$$b_{45}(\mathbf{x}) = (x_1 - s_{51}) = x_1$$

$$b_{51}(\mathbf{x}) = (x_2 - s_{12}) = x_2$$

$$b_{52}(\mathbf{x}) = (x_1 - s_{21}) = x_1 + 1$$

$$b_{53}(\mathbf{x}) = (x_1 - s_{31}) = x_1 + 1$$

$$b_{54}(\mathbf{x}) = (x_1 - s_{42}) = x_1 + 1$$

skip $k = 5$

Recall that $x_i^k = x_i$, and $(x_j + 1)^k = x_j + 1$, so the “ r -polynomials” are

$$r_1(\mathbf{x}) = (x_1 + 1)(x_2 + 1)$$

$$r_2(\mathbf{x}) = x_1(x_2 + 1)$$

$$r_3(\mathbf{x}) = x_1 x_2 x_3$$

$$r_4(\mathbf{x}) = x_1 x_2 (x_3 + 1)$$

$$r_5(\mathbf{x}) = (x_1 + 1)x_2$$

A Boolean example (cont.)

We can now compute our particular solution (f_1, f_2, f_3) that fits the data, using:

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

$s_1 = (0, 0, 1)$	
↓	
$s_2 = (1, 0, 1) = t_1$	$f_1(\mathbf{x}) = t_{11}r_1(\mathbf{x}) + t_{21}r_2(\mathbf{x}) + t_{31}r_3(\mathbf{x}) + t_{41}r_4(\mathbf{x}) + t_{51}r_5(\mathbf{x})$
↓	$= r_1(\mathbf{x}) + r_2(\mathbf{x}) + r_3(\mathbf{x})$
$s_3 = (1, 1, 1) = t_2$	$= 1 + x_2 + x_1x_2x_3$
↓	
$s_4 = (1, 1, 0) = t_3$	$f_2(\mathbf{x}) = t_{12}r_1(\mathbf{x}) + t_{22}r_2(\mathbf{x}) + t_{32}r_3(\mathbf{x}) + t_{42}r_4(\mathbf{x}) + t_{52}r_5(\mathbf{x})$
↓	$= r_2(\mathbf{x}) + r_3(\mathbf{x}) + r_4(\mathbf{x})$
$s_5 = (0, 1, 0) = t_4$	$= x_1$
↓	
$(0, 0, 0) = t_5$	$f_3(\mathbf{x}) = t_{13}r_1(\mathbf{x}) + t_{23}r_2(\mathbf{x}) + t_{33}r_3(\mathbf{x}) + t_{43}r_4(\mathbf{x}) + t_{53}r_5(\mathbf{x})$
	$= r_1(\mathbf{x}) + r_2(\mathbf{x})$
	$= 1 + x_2.$

Our original PDS was $(f_1, f_2, f_3) = (x_3, x_1, x_3 + x_2x_3)$, but our algorithm yielded

$$\begin{aligned}(f_1, f_2, f_3) &= (1 + x_2 + x_1x_2x_3, x_1, 1 + x_2) \\ &= (x_3, x_1, x_3 + x_2x_3) + (1 + x_2 + x_3 + x_1x_2x_3, 0, 1 + x_2 + x_3 + x_2x_3)\end{aligned}$$

Remark

Each polynomial in the 2nd term above is in the *vanishing ideal* I . (Why?)

A Boolean example (cont.)

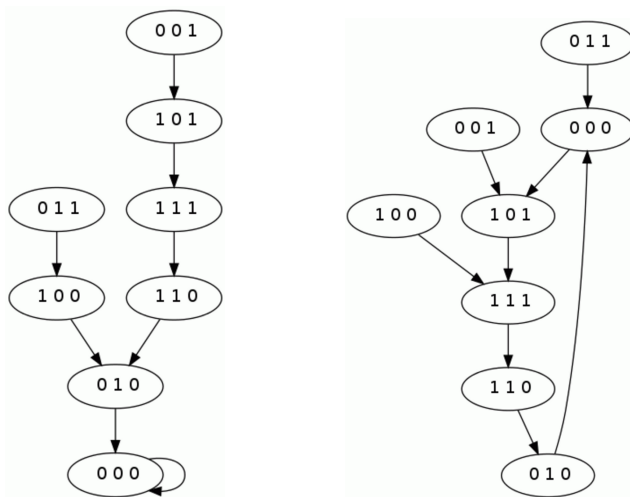


Figure: The original phase space (left), and the reverse-engineered phase space (right).

A Boolean example (cont.)

Now that we found a particular solution $f = (f_1, f_2, f_3)$ that fits the data, we need to (re)compute the **ideal I of polynomials that vanish on the data**.

We'll use Macaulay2 in Sage:

```
%default_mode macaulay2
R=ZZ/2[x1,x2,x3] / ideal(x1^2-x1, x2^2-x2, x3^2-x3);
```

$$s_1 = (0, 0, 1)$$



$$s_2 = (1, 0, 1) = t_1$$



$$s_3 = (1, 1, 1) = t_2$$



$$s_4 = (1, 1, 0) = t_3$$



$$s_5 = (0, 1, 0) = t_4$$



$$(0, 0, 0) = t_5$$

The ideal of polynomials that vanish on each s_k is:

```
I1 = ideal(x1, x2, x3-1);
I2 = ideal(x1-1, x2, x3-1);
I3 = ideal(x1-1, x2-1, x3-1);
I4 = ideal(x1-1, x2-1, x3);
I5 = ideal(x1, x2-1, x3);
```

The ideal of polynomials that vanish on every s_k is:

```
I = intersect{I1,I2,I3,I4,I5}
```

To compute a Gröbner basis:

```
G = gens gb I
```

The output is: | $x_2x_3+x_2+x_3+1$ $x_1x_2+x_1x_3+x_1+x_2+x_3+1$ |

A Boolean example (cont.)

In conclusion, the set of all Boolean networks that fit the data

$$001 \longrightarrow 101 \longrightarrow 111 \longrightarrow 110 \longrightarrow 010 \longrightarrow 000$$

i.e., the **model space**, is the set

$$F_1 \times F_2 \times F_3, \quad F_j = f_j + I$$

where I is the **vanishing ideal**

$$I = \langle g_1, g_2 \rangle = \langle 1 + x_2 + x_3 + x_2x_3, 1 + x_1 + x_2 + x_3 + x_1x_2 + x_1x_3 \rangle.$$

Our reverse-engineered BN is slightly different than the “true model”:

$$\begin{aligned} (f_1, f_2, f_3) &= (1 + x_2 + x_1x_2x_3, x_1, x_1 + x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_1x_2x_3) \\ &= (x_3 + x_1g_1 + g_2, x_1, (x_2 + 1)x_3 + g_1) \end{aligned}$$

Note that g , 0 , and x_1g *must* be in the vanishing ideal I .

Next goal (“model selection”)

We would like to be able to recover functions in $F_j = f_j + I$ that have no “extra terms” in I .

A Boolean example (cont.)

Goal (“model selection”)

We would like to be able to recover functions in $F_j = f_j + I$ that have no “extra terms” in I .

001 \longrightarrow 101 \longrightarrow 111 \longrightarrow 110 \longrightarrow 010 \longrightarrow 000

We can do this with Macaulay2. It's called finding the **remainder of f_j modulo I** , and we use the `%` symbol.

```
f1 = 1+x2+x1x2x3;  
f2 = x1;  
f3 = 1+x2;  
f1%I;  
f2%I;  
f3%I;
```

The output is: x^3 , x^1 , $x^3(x^2+1)$. The original dynamical system!

Question

What would happen if we:

- added the (original) self-loop at 000 to the data?
- removed the data point 010 \longrightarrow 000?

An example over \mathbb{F}_5

Consider the following **time series** in a 3-node local model over \mathbb{F}_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 :

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

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

An example over \mathbb{F}_5 (cont.)

Since we are working in \mathbb{F}_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: $r_2(\mathbf{x}) = 3x_1^2 + 3$, $r_3(\mathbf{x}) = 4x_1^2 + x_1 + 2$.

An example over \mathbb{F}_5 (cont.)

In summary, we computed the r -polynomials to be:

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

$$r_2(\mathbf{x}) = b_{21}(\mathbf{x})b_{23}(\mathbf{x}) = (3x_1 + 4)(x_1 + 2) = 3x_1^2 + 10x_1 + 8 = 3x_1^2 + 3$$

$$r_3(\mathbf{x}) = b_{31}(\mathbf{x})b_{32}(\mathbf{x}) = (x_1 + 3)(4x_1 + 4) = 4x_1^2 + 16x_1 + 12 = 4x_1^2 + x_1 + 2$$

Thus, the following functions fit the data:

$$\begin{aligned}f_1(\mathbf{x}) &= t_{11}r_1(\mathbf{x}) + t_{21}r_2(\mathbf{x}) + t_{31}r_3(\mathbf{x}) \\ &= 4(3x_1^2 + 4x_1 + 1) + 3(3x_1^2 + 3) + 0(4x_1^2 + x_1 + 2) \\ &= x_1^2 + x_1 + 3\end{aligned}$$

$$\begin{aligned}f_2(\mathbf{x}) &= t_{12}r_1(\mathbf{x}) + t_{22}r_2(\mathbf{x}) + t_{32}r_3(\mathbf{x}) \\ &= 3(3x_1^2 + 4x_1 + 1) + 1(3x_1^2 + 3) + 4(4x_1^2 + x_1 + 2) \\ &= 3x_1^2 + x_1 + 4\end{aligned}$$

$$\begin{aligned}f_3(\mathbf{x}) &= t_{13}r_1(\mathbf{x}) + t_{23}r_2(\mathbf{x}) + t_{33}r_3(\mathbf{x}) \\ &= 1(3x_1^2 + 4x_1 + 1) + 4(3x_1^2 + 3) + 3(4x_1^2 + x_1 + 2) \\ &= 2x_1^2 + 2x_1 + 4\end{aligned}$$

Comments on this? [Note that only the variable x_1 appears.]

An example over \mathbb{F}_5 (cont.)

Recall that the ideal I is the set 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

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

These are precisely the sets

$$I(\mathbf{s}_1) = \langle x_1 - 2, x_2, x_3 \rangle = \{(x_1 - 2)g_1(x) + x_2g_2(x) + x_3g_3(x)\}$$

$$I(\mathbf{s}_2) = \langle x_1 - 4, x_2 - 3, x_3 - 1 \rangle = \{(x_1 - 4)g_1(x) + (x_2 - 3)g_2(x) + (x_3 - 1)g_3(x)\}$$

$$I(\mathbf{s}_3) = \langle x_1 - 3, x_2 - 1, x_3 - 4 \rangle = \{(x_1 - 3)g_1(x) + (x_2 - 1)g_2(x) + (x_3 - 4)g_3(x)\}$$

We'll use Macaulay2 in Sage again to compute this. Remember to work over \mathbb{F}_5 :

```
%default_mode macaulay2
R=ZZ/5[x1,x2,x3] / ideal(x1^5-x1, x2^5-x2, x3^5-x3);
```

An example over \mathbb{F}_5 (cont.)

As before, we compute the vanishing ideal I :

```
I1 = ideal(x1-2, x2, x3);  
I2 = ideal(x1-4, x2-3, x3-1);  
I3 = ideal(x1-3, x2-1, x3-4);  
I = intersect{I1,I2,I3};  
f1 = x1^2+x1+3;  
f2 = 3*x1^2+x1+4;  
f3 = 2*x1^2+2*x1+4;
```

We can view the Gröbner basis (w.r.t. the default monomial ordering, *grevlex*):

```
flatten entries gens gb I;
```

The output is $\mathcal{G} = \{x_1 + x_2 + x_3, x_3^2 + x_3, x_2x_3, x_2^2 + x_2\}$.

If we reduce each f_j modulo I : $p_1=f_1\%I$; and $p_2=f_2\%I$; and $p_3=f_3\%I$; we get

$$(p_1, p_2, p_3) = (-x_3 - 1, x_2 - 2, -2x_3 + 1).$$

This is called the **normal form** of f_j with respect to the Gröbner basis \mathcal{G} .

An aside: why monomial order matters!

Let's do the same thing but using *lex* instead of *grevlex*:

```
S=ZZ/5[x1,x2,x3] / ideal(x1^5-x1, x2^5-x2, x3^5-x3);  
p1 = sub(f1,S)%sub(I,S);  
p2 = sub(f2,S)%sub(I,S);  
p3 = sub(f3,S)%sub(I,S);
```

These polynomials are now $(p_1, p_2, p_3) = (-x_3 - 1, 2x_3^2 + x_3 - 2, -2x_3 + 1)$. They are different because the Gröbner basis of I is different with respect to *lex*:

```
flatten entries gens gb sub(I,S);
```

The output is $\mathcal{G} = \{x_3^3 - x_3, x_2 - 2x_3^2 - x_3, x_1 + x_3^2 + 2x_3 - 2\}$.

In summary, the model space is

$$F_1 \times \cdots \times F_n = f + (I \times \cdots \times I) = (f_1 + I, \dots, f_n + I).$$

Here are three choices for the “particular” solution $f = (f_1, f_2, f_3)$ that fits the data:

- our algorithm: $(f_1, f_2, f_3) = (x_1^2 + x_1 + 3, 3x_1^2 + x_1 + 4, 2x_1^2 + 2x_1 + 4)$.
- normal form w.r.t. *grevlex*: $(f_1, f_2, f_3) = (-x_3 - 1, 2x_3^2 + x_3 - 2, -2x_3 + 1)$.
- normal form w.r.t. *lex*: $(f_1, f_2, f_3) = (-x_3 - 1, x_2 - 2, -2x_3 + 1)$.