Math 4500 worksheet: Reverse engineering of polynomial dynamical systems

Goal. Find all models $F = (f_1, \dots, f_n)$: that fit the partial 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 $F(\mathbf{s}_i) = \mathbf{t}_i$.

Here, each $f_i \colon \mathbb{F}_2^n \to \mathbb{F}_2$ can be assumed to be a Boolean *polynomial*, and updating each of these functions synchronously yields the *finite dynamical systems (FDS) map* $F \colon \mathbb{F}_2^n \to \mathbb{F}_2^n$. That is, the equation $F(\mathbf{s}_i) = \mathbf{t}_i$ means

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

The set of all solutions (models) is called the *model space*:

$$F_1 \times \cdots \times F_n = \{(f_1, \dots, f_n) \mid f_j(\mathbf{s}_i) = t_{ij}\} \,.$$

To find all solutions, we find each F_j separately. Note that F_j is the set of all local functions at node j that fit the data:

$$F_j = \{f_j : f_j(\mathbf{s}_1) = t_{ij}, \ \dots \ , f_j(\mathbf{s}_m) = t_{im}\} \,.$$

To find F_j, we use that fact that it can be written as

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

where f_j is *any particular* function in F_j, and I is the set of polynomials that vanish on the data:

$$I = \{h : h(\mathbf{s}_i) = 0 \text{ for all } i = 1, \dots, m\} \,.$$

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

(1) Find the ideal I;
(2) Find *any* polynomial f_j that fits the data.

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 simply

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

2. Finding f_j: This method uses the "Chinese Remainder Theorem" for rings, though this is "hidden" in the background.

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

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

Once we have the r-polynomials, then the polynomial $f_j(\mathbf{x})$ we seek will be

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

So, how do we find these r-polynomials? There are likely many such polynomials that work, but here's a sure-fire way to construct them:

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

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.

This looks horrible! (But it's not too bad.) Let's try it. 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 $\mathbf{t}_i$:

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

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 f_1, f_2, and f_3.

Let's compute the first r-polynomial, which is:

$$r_1(\mathbf{x}) = b_{12}(\mathbf{x}) b_{13}(\mathbf{x})\,.$$

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 $b_{ij}(x)$, we compute:

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

Therefore, the first r-polynomial is

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

Your turn! Compute $r_2(\mathbf{x})$ and $r_3(\mathbf{x})$, and then use these to find $f_1(\mathbf{x})$, $f_2(\mathbf{x})$, and $f_3(\mathbf{x})$. Note that you will need to compute the polynomials $b_{21}(\mathbf{x})$, $b_{23}(\mathbf{x})$, $b_{31}(\mathbf{x})$, and $b_{32}(\mathbf{x})$.

Before proceeding, check to make sure that each of these polynomials fits the data. In other words, for each $j = 1,2,3$, verify (do this now!) that

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

To explore why this works, go back a step further, and verify that each r-polynomial satisfies the equation from (1).

Now that we have found f_1, f_2, and f_3, 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

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

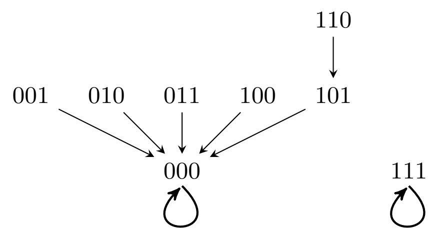
Further exploration. In this project, we will investigate a simple Boolean FDS, explore its phase space, and attempt to reverse engineer it given partial data.

Consider the following polynomial dynamical system:

$$\begin{aligned} f_1(x_1,x_2,x_3) &= \mathrm{AND}(x_1,x_2) = x_1x_2 \\ f_2(x_1,x_2,x_3) &= \mathrm{AND}(x_1,x_2,x_3) = x_1x_2x_3 \\ f_3(x_1,x_2,x_3) &= \mathrm{AND}(x_1,x_2) = x_1x_2\,. \end{aligned}$$

This is called an *AND-network* because the Boolean functions can be written as logical AND functions.

Go to the Analysis of Dynamic Algebraic Models (ADAM) toolbox, at `http://adam.plantsimlab.org/`. Enter the functions above into the "Model Input" box and click the "Analyze" button. The state space should look like this:



This graph literally encodes the entire function $F = (f_1, f_2, f_3)\colon \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^3$.

Let's try to reverse engineer this network given partial data. In particular, let's suppose that all we know is the following "piece" of the state space (elephant):

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

Follow the steps of the above example to find all FDS maps that fit this data. Naturally, you could "cheat" and use the OR functions above for f_1, f_2, and f_3, but try the r-polynomial method. Do you get the same particular solution?