Data-driven Modeling I: Structural and Parametric Models

Abhilash Patel

1 Motivation

In the last lectures on state-space modeling, every model we wrote down came from the same framework: start with a physical law, apply it to the system at hand, arrive at a differential equation, reduce it to a set of first-order differential equations, if required, and write it in state-space representation. This framework works well when the underlying mechanism is known and simple enough to write down. But a large class of systems that control engineers now work with is not straightforward to capture with a set of physical laws, and in some cases, the systems are natural, or we don't understand them well enough to offer a law at all.

Consider the price of a stock. There is no equation of motion for it in the way there is for a mass on a spring or a current in a circuit; no accepted physical law dictates how a stock's price must evolve from one day to the next. What drives it, aggregated buying and selling decisions, macroeconomic news, sentiment, is not reducible to a differential equation anyone can write down with confidence, and a large body of financial theory holds that it should not be predictable from its own past in any simple deterministic sense at all. What is abundantly available, however, is data: daily closing prices, trading volumes, and related indices, going back decades, sampled once a day almost by definition.

This is the situation data-driven modeling addresses in its purest form. Rather than deriving a model from first principles, and with no first principles to derive one from, we ask: given measured signals, can we construct a model directly from the data, one that is accurate enough to be useful even though we do not, and perhaps cannot, know why it works?

2 System Identification/Machine Learning/Data-driven Modeling

Before building any particular method, it helps to state the problem in general terms.

Definition 2.1. *Given a sequence of measurements of a system's states and, if applicable, its inputs, the system identification problem is to find a model, within a chosen class of candidate models, that best reproduces the observed data according to some fitting criterion.*

Three ingredients sit inside this definition, and every data-driven method we discuss is a particular choice of these three.

The data. Trajectories of the state, possibly along with the inputs applied to the system. For the stock example, this is a sequence of measured prices, returns, and volumes across successive trading days.

The model class. The family of candidate models we are willing to search over: linear models, polynomial models, neural networks, and so on. Choosing this class is itself a modeling decision, and it encodes whatever partial knowledge we do have about the system.

The fitting criterion. A way of measuring how well a candidate model explains the data, most commonly a sum of squared prediction errors, which we minimize over the model class.

Definition 2.2. *A white-box model is derived entirely from known physical laws, with no free parameters left to be estimated from data. A black-box model uses a generic, physically uninterpretable structure whose parameters are estimated entirely from data. A grey-box model uses a structure partly justified by physical reasoning, with some parameters estimated from data.*

Definition 2.3. *A model is parametric if identification consists of estimating a finite set of coefficients (or weights) within a functional form chosen in advance. A model is non-structural in this sense if identification instead consists of inferring a representation, an operator, or a distribution over functions, rather than coefficients within a committed form.*

2.1 Choosing an Approach

Which route to take depends on what is available and what is wanted from the model. Parametric models, of the kind developed in this note, are the natural choice when a compact, interpretable structure is wanted, whether that structure is imposed by hand, as in AR and ARX, selected automatically from a large library, as in SINDy, or left almost entirely to be learned, as in a neural network: in every case, identification consists of estimating coefficients or weights within a model class chosen in advance. A second document takes up the complementary case, where identification instead infers a representation of the system directly, an operator acting on a lifted space of observables, or a distribution over functions, without committing to a parametric form at all.

3 Data-driven Parametric Models

We now discuss different structures of parametric models, which can be used for fitting measurements.

3.1 Autoregressive (AR) Models

The simplest parametric model assumes that future output as a linear combination of current and past outputs,

$$y_{k+1} = \sum_{i=1}^{n_a} a_i \, y_{k+1-i}.$$

Unlike the physics-based models of earlier notes, the AR structure is not typically derived from a governing law of the system. It is a model chosen for identification: the coefficients $a_1, \ldots, a_{n_a}$ are not known in advance and must be estimated from measured data. Given an estimate $\hat{a}_1, \ldots, \hat{a}_{n_a}$, the model predicts

$$\hat{y}_{k+1} = \sum_{i=1}^{n_a} \hat{a}_i \, y_{k+1-i},$$

and the error between this prediction and the measured output,

$$e_{k+1} = y_{k+1} - \hat{y}_{k+1},$$

is what identification seeks to minimize.
The AR model admits a companion-form state-space realization,

$$x_{k+1} = A_{\mathrm{AR}} x_k, \qquad y_k = C_{\mathrm{AR}} x_k,$$

where the state vector

$$x_k = \begin{bmatrix} y_k \\ y_{k-1} \\ \vdots \\ y_{k-n_a+1} \end{bmatrix}, \quad A_{\mathrm{AR}} = \begin{bmatrix} a_1 & a_2 & \cdots & a_{n_a-1} & a_{n_a} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}, \quad C_{\mathrm{AR}} = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}.$$

Definition 3.1. *An autoregressive model of order n_a, denoted AR(n_a), is the structure*

$$y_{k+1} = \sum_{i=1}^{n_a} a_i \, y_{k+1-i},$$

with unknown coefficients $a_1, \ldots, a_{n_a}$ to be estimated from measured data.

Remark 3.1. *The AR structure is time-invariant: the coefficients $a_1, \ldots, a_{n_a}$ do not depend on k. The recursion can therefore be shifted in time without changing its form, for instance writing y_k in terms of $y_{k-1}, \ldots, y_{k-n_a}$ instead of y_{k+1} in terms of $y_k, \ldots, y_{k+1-n_a}$. Both are the same model, indexed differently, and we will use whichever indexing is more convenient in a given context.*

Example 3.1. *Let y_k denote the BDMR of the Hall 10 mess on day k. An AR(3) model assumes that today's BDMR can be predicted from its values over the previous three days:*

$$y_{k+1} = a_1 y_k + a_2 y_{k-1} + a_3 y_{k-2}. \tag{1}$$

For example, after identification we may obtain

$$\hat{y}_{k+1} = 0.6 y_k + 0.25 y_{k-1} + 0.10 y_{k-2}. \tag{2}$$

Thus, the model uses only the recent history of BDMR to predict its next value, without explicitly modelling food prices, consumption, or other factors affecting the mess expenditure.

3.2 From AR to ARX to ARMAX

An AR model uses only the past values of the signal being predicted. In many settings an additional measured signal, an input u_k, also affects the output. Extending the AR model to include this input gives the ARX model, X standing for exogenous.

Definition 3.2. *An ARX model of order (n_a, n_b), denoted ARX(n_a, n_b), is the structure*

$$y_{k+1} = \sum_{i=1}^{n_a} a_i \, y_{k+1-i} + \sum_{j=1}^{n_b} b_j \, u_{k+1-j},$$

with unknown coefficients $a_1, \ldots, a_{n_a}$ and $b_1, \ldots, b_{n_b}$ to be estimated from measured data.

As with the AR model, the coefficients are not known in advance; they are estimated from measured pairs of input and output, and the error between the measured output and the value predicted by the estimated model is minimized during identification.

The ARX model can be written in state-space form by augmenting the output history with the required past input values. Let $m = n_b - 1$ and define

$$x_k = \begin{bmatrix} y_k & y_{k-1} & \cdots & y_{k-n_a+1} & u_{k-1} & u_{k-2} & \cdots & u_{k-m} \end{bmatrix}^\top.$$

The ARX model admits the state-space representation

$$x_{k+1} = A_{\mathrm{ARX}} x_k + B_{\mathrm{ARX}} u_k, \qquad y_k = C_{\mathrm{ARX}} x_k,$$

where

$$A_{\text{ARX}} = \begin{bmatrix} A_y & G_u \\ 0 & S_u \end{bmatrix},$$

with

$$A_y = \begin{bmatrix} a_1 & a_2 & \cdots & a_{n_a-1} & a_{n_a} \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}, \qquad G_u = \begin{bmatrix} b_2 & b_3 & \cdots & b_{n_b} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}, \qquad S_u = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 1 & 0 \end{bmatrix}.$$

The input and output matrices are

$$B_{\text{ARX}} = \begin{bmatrix} b_1 \\ 0_{n_a-1} \\ 1 \\ 0_{m-1} \end{bmatrix}, \qquad C_{\text{ARX}} = \begin{bmatrix} 1 & 0_{1\times(n_a+m-1)} \end{bmatrix}.$$

The state x_k collects measured samples, so $C_{\text{ARX}}x_k = y_k$ exactly. Since A_{ARX} and B_{ARX} are built from the estimated coefficients, the first entry of $x_{k+1} = A_{\text{ARX}}x_k + B_{\text{ARX}}u_k$ is the prediction $\hat{y}_{k+1}$, not the measured y_{k+1}.

Remark 3.2. *As with the AR model, the ARX structure is time-invariant, so the recursion can be shifted in time without changing its form.*

An ARMAX model extends the ARX model further, adding a weighted sum of past errors to the structure, MA standing for moving average.

Definition 3.3. *An ARMAX model of order (n_a, n_b, n_c), denoted ARMAX(n_a, n_b, n_c), is the structure*

$$y_{k+1} = \sum_{i=1}^{n_a} a_i\, y_{k+1-i} + \sum_{j=1}^{n_b} b_j\, u_{k+1-j} + \sum_{l=1}^{n_c} c_l\, e_{k+1-l} + e_{k+1}.$$

Example 3.2. *Consider a fitted ARX(1,2) model for the stock's return y_k, driven by the market index return u_k,*

$$\hat{y}_{k+1} = \hat{a}_1 y_k + \hat{b}_1 u_k + \hat{b}_2 u_{k-1}.$$

Here $n_a = 1$ and $n_b = 2$, so $m = n_b - 1 = 1$ and the state vector has two entries,

$$x_k = \begin{bmatrix} y_k \\ u_{k-1} \end{bmatrix}.$$

Instantiating the general ARX state-space matrices at these orders gives

$$\hat{A}_{\text{ARX}} = \begin{bmatrix} \hat{a}_1 & \hat{b}_2 \\ 0 & 0 \end{bmatrix}, \qquad \hat{B}_{\text{ARX}} = \begin{bmatrix} \hat{b}_1 \\ 1 \end{bmatrix}, \qquad \hat{C}_{\text{ARX}} = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

The state update $x_{k+1} = \hat{A}_{\text{ARX}}x_k + \hat{B}_{\text{ARX}}u_k$ has first entry $\hat{a}_1 y_k + \hat{b}_2 u_{k-1} + \hat{b}_1 u_k$, which is the prediction $\hat{y}_{k+1}$, and second entry u_k, the new value stored for the next step.

Example 3.3. *Let us visit the Daily BDMR of Hall 10 Mess Model. Suppose now that the number of residents taking meals is also measured, and let u_k denote the number of meals served on day k. Since the daily expenditure, and hence the BDMR, may also depend on mess participation, an ARX model can include this information. For example,*

$$y_{k+1} = a_1 y_k + a_2 y_{k-1} + a_3 y_{k-2} + b_1 u_k + b_2 u_{k-1}.$$

The model now uses both the past BDMR and a measured external variable to predict the next day's BDMR.

Even after accounting for past BDMR and mess participation, some effects may remain unmeasured, such as fluctuations in vegetable prices, changes in the menu, procurement conditions, or wastage. If these effects have temporal dependence, they can be represented through past prediction errors. An ARMAX model may therefore be written as

$$y_{k+1} = a_1 y_k + a_2 y_{k-1} + a_3 y_{k-2} + b_1 u_k + b_2 u_{k-1} + c_1 e_k + c_2 e_{k-1} + e_{k+1}.$$

3.3 Nonlinear ARMAX (NARMAX) Models

The NARMAX model generalizes ARMAX by replacing the linear combination with a general nonlinear function of past measurements.

Definition 3.4. *A NARMAX model of order (n_a, n_b, n_c), denoted NARMAX(n_a, n_b, n_c), is the structure*

$$y_{k+1} = F\big(y_k, \ldots, y_{k+1-n_a},\, u_k, \ldots, u_{k+1-n_b},\, e_k, \ldots, e_{k+1-n_c}\big),$$

where F is an unknown nonlinear function, to be estimated from measured data along with any parameters used to represent it.

Remark 3.3. *ARMAX is the special case of NARMAX in which F is restricted to be linear in its arguments, $F(\cdot) = \sum_i a_i y_{k+1-i} + \sum_j b_j u_{k+1-j} + \sum_l c_l e_{k+1-l}$. AR and ARX follow in turn by further restricting F to depend only on past outputs, or on past outputs and inputs.*

It is important to understand a class of representation, where a nonlinear function with respect to past measurement/states can be represented as linear function with respect to the parameter.

Definition 3.5. *A model is linear in the parameters (LIP) if it can be written as*

$$\hat{y}_{k+1} = \theta^\top \phi(x_k),$$

where x_k is the regressor, $\phi : \mathbb{R}^{n_x} \to \mathbb{R}^p$ is a fixed, known vector of basis functions, and $\theta \in \mathbb{R}^p$ is the only unknown to be estimated. The basis functions ϕ may be nonlinear in x_k; only the dependence on θ must be linear.

A common choice of ϕ is the set of monomials in the entries of x_k up to a fixed degree d. For a regressor $x_k = (y_k, u_k)$ and $d = 2$, this gives

$$\phi(x_k) = \begin{bmatrix} y_k & u_k & y_k^2 & y_k u_k & u_k^2 \end{bmatrix}^\top,$$

and the resulting model,

$$\hat{y}_{k+1} = a_1 y_k + b_1 u_k + q_1 y_k^2 + q_2 y_k u_k + q_3 u_k^2,$$

is a NARX model in the sense of Definition 3.4, with $F(x_k) = \theta^\top \phi(x_k)$, but is linear in $\theta = (a_1, b_1, q_1, q_2, q_3)$.

Definition 3.6. *A model is nonlinear in the parameters (NLIP) if it can be written as*

$$\hat{y} = f(x, \theta),$$

where x is the regressor, $\theta \in \mathbb{R}^p$ is the unknown parameter vector. The dependence of f on θ cannot be separated from the dependence on x by any choice of basis functions.

A common example is the exponential decay model,

$$\hat{y} = \theta_1 e^{-\theta_2 x},$$

where θ_2 enters inside the exponent, so no fixed transformation of x makes this linear in (θ_1, θ_2) jointly. Another is the saturating model,

$$\hat{y} = \frac{\theta_1 x}{\theta_2 + x},$$

a Michaelis-Menten type relationship common in enzyme kinetics and other saturating biological processes, where θ_2 enters through the denominator. Both are NLIP in the sense of Definition 3.6: in each case, the parameters cannot be moved into a fixed $\phi(x)$, leaving a linear combination behind.

3.4 Sparse Identification of Nonlinear Dynamics (SINDy)

The NARMAX structure of Definition 3.4 leaves F unspecified beyond a chosen basis. Definition 3.5 showed one way to make this concrete: restrict F to be linear in the parameters, $F(x_k) = \theta^\top \phi(x_k)$, with ϕ a fixed set of basis functions such as monomials. SINDy is this same construction, taken further in one specific direction: the basis ϕ is replaced by a large library of candidate functions, and the coefficient vector is expected to be mostly zero, with only the terms that genuinely belong in the dynamics surviving identification.

Definition 3.7. *Given a state $x \in \mathbb{R}^n$ and a library of candidate functions $\Theta(x) = \begin{bmatrix}\theta_1(x) & \theta_2(x) & \cdots & \theta_p(x)\end{bmatrix}$, chosen in advance, SINDy models the dynamics as*

$$x_{k+1} = \Theta(x_k)\,\Xi \quad \textit{(discrete time)}, \qquad \textit{or} \qquad \dot{x} = \Theta(x)\,\Xi \quad \textit{(continuous time)},$$

where $\Xi \in \mathbb{R}^{p \times n}$ is a coefficient matrix, each column of which is expected to have only a small number of nonzero entries: identification consists of estimating Ξ, and enforcing that it be sparse, from measured data.

Remark 3.4. *SINDy is linear in the parameters Ξ, in the sense of Definition 3.5, for any fixed library Θ: the library itself may contain arbitrarily nonlinear functions of x, but the dependence of the model on Ξ is a linear combination. What distinguishes SINDy from the monomial example following Definition 3.5 is only the size of the library and the explicit expectation that most of Ξ is zero, so that the correct terms, not merely a good fit, are recovered.*

The library Θ is typically built from low-order monomials, trigonometric terms, or any other function the modeler has reason to believe might appear in the true dynamics; it plays the same role Definition 2.2 assigns to the grey-box case, since choosing Θ is where whatever partial physical knowledge is available enters the model.

Example 3.4. *Consider again the ARX(1,2) stock-return model of Example 3.2, with true coefficients $a_1 = 0.5$, $b_1 = 0.3$, $b_2 = -0.2$, so that $y_{k+1} = 0.5y_k + 0.3u_k - 0.2u_{k-1}$. Rather than assuming this linear structure, build a SINDy library from the same regressor,*

$$\Theta(x_k) = \begin{bmatrix} y_k & u_k & u_{k-1} & y_k^2 & y_k u_k & u_k^2 & y_k u_{k-1} & u_{k-1}^2 & u_k u_{k-1} \end{bmatrix},$$

nine candidate terms in total, only three of which belong in the true model. Fitting Ξ against sixty simulated samples of y_k, u_k recovers

$$\hat{\Xi} = \begin{bmatrix} 0.500 & 0.300 & -0.200 & 0.000 & -0.000 & -0.000 & -0.000 & 0.000 & 0.000 \end{bmatrix}^\top,$$

exactly the three true coefficients, with every nonlinear entry at numerical zero. This is the same relationship Remark 3.3 noted between NARMAX and ARMAX: ARX is recovered as the special case of SINDy in which the library happens to contain only the terms the true dynamics actually use.

The previous example fit data generated by a linear model, so recovering a sparse, purely linear $\hat{\Xi}$ is expected. SINDy's actual value is in the reverse situation: recovering nonlinear structure from data when the governing equations were not assumed in advance.

Example 3.5. *Consider a predator-prey system, with prey population x and predator population y evolving as*

$$\dot{x} = a\,x - b\,xy, \qquad \dot{y} = -c\,y + d\,xy,$$

the Lotka-Volterra equations, with $a = 1.1$, $b = 0.4$, $c = 0.4$, $d = 0.1$, and initial populations $x_0 = 10$, $y_0 = 5$. No structure is assumed for this example beyond a library of low-order monomials,

$$\Theta(x, y) = \begin{bmatrix} 1 & x & y & x^2 & xy & y^2 \end{bmatrix}.$$

Fitting Ξ against trajectory data, sampled and processed as described in Section 4.6 where the fitting procedure itself is developed, recovers

$$\hat{\dot{x}} = 1.091\,x - 0.395\,xy, \qquad \hat{\dot{y}} = -0.395\,y + 0.099\,xy,$$

matching the true coefficients to within five percent, with every other library entry at numerical zero. Unlike Example 3.4, the correct model structure, which terms belong and which do not, was not known beforehand; SINDy recovered it directly from the trajectory, along with the coefficients.

Remark 3.5. *The library Θ must contain the true governing terms for SINDy to recover them; no amount of data will produce a term absent from the library. This is the price of the flexibility SINDy offers over AR, ARX, ARMAX, and NARMAX with a hand-chosen F: the modeler still commits to a space of candidate structures, only a much larger one, and sparsity, together with sufficient data, decides which of those candidates survive.*

3.5 Neural Network Models, Discrete Time

The NARX model of Definition 3.4 left the function F unspecified. A neural network is a concrete, adjustable form for that function, one whose shape is set by adjustable weights rather than committed in advance. This subsection develops the network as an object for representing an unknown function. It is assembled from a single unit, then a layer of units, then a combination of the layer into a scalar output, and the representational power of the result is stated through the Universal Approximation Theorem.

A Single Neuron

The basic unit takes a vector of inputs, forms a weighted sum, adds a constant offset, and passes the result through a fixed nonlinear function.

Definition 3.8. *Given an input vector $x \in \mathbb{R}^{n_x}$, a neuron with weights $\theta \in \mathbb{R}^{n_x}$, bias $b \in \mathbb{R}$, and activation function $\sigma : \mathbb{R} \to \mathbb{R}$ computes*

$$z = \theta^\top x + b, \qquad h = \sigma(z).$$

The quantity z is called the pre-activation, and h the activation, of the neuron.

An example for $n_x = 3$ is shown in Figure 1: three inputs are each scaled by a weight, summed with the bias, and passed through σ.

A single neuron produces one scalar function of the input, $\sigma(\theta^\top x + b)$, and so can represent only a limited range of shapes. The next step is to apply many neurons, each with its own weights, to the same input, producing many such functions at once.

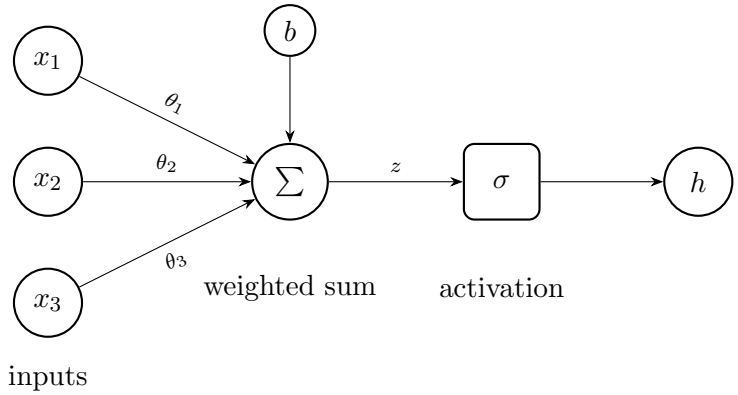

Figure 1: A single neuron. The inputs are scaled by weights $\theta_1, \theta_2, \theta_3$, summed with a bias b to give the pre-activation z, and passed through σ to give the output h.

A Layer of Neurons

Definition 3.9. *A layer of n_h neurons, applied to an input $x \in \mathbb{R}^{n_x}$, computes*

$$z = Vx + c, \qquad \phi(x) = \sigma(z),$$

where $V \in \mathbb{R}^{n_h \times n_x}$ and $c \in \mathbb{R}^{n_h}$ collect the weights and biases of all n_h neurons, one per row, and σ is applied entrywise to give $\phi(x) \in \mathbb{R}^{n_h}$. The vector $\phi(x)$ is called the feature vector, and its entries the feature functions of the layer.

Row i of V and entry i of c are the weights and bias of neuron i in Definition 3.8. A layer is n_h neurons evaluated on the same input, stacked into the vector $\phi(x)$. Each entry $\phi_i(x) = \sigma(V_{i,:}x + c_i)$ is a nonlinear function of the whole input, and the inner weights V, c set the position and steepness of these functions. Figure 2 shows this for $n_x = 3$, $n_h = 3$.

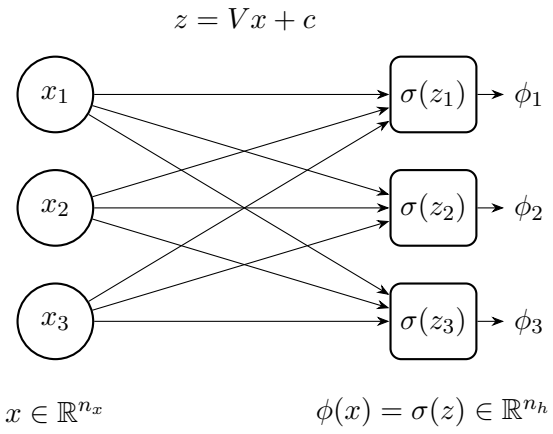

Figure 2: A layer of $n_h = 3$ neurons applied to an input $x \in \mathbb{R}^{n_x}$ with $n_x = 3$. Every input connects to every neuron; row i of V and entry i of c give the pre-activation z_i, and σ applied entrywise gives $\phi_i = \sigma(z_i)$. Stacking the outputs gives the feature vector $\phi(x) = \sigma(Vx + c) \in \mathbb{R}^{n_h}$.

Example 3.6. *Take a regressor $x_k = (y_k, y_{k-1}, u_k) = (0.8, 0.3, -0.4)$ and a layer of $n_h = 2$ neurons with*

$$V = \begin{bmatrix} 0.5 & 0.2 & -0.3 \\ 0.3 & -0.4 & 0.2 \end{bmatrix}, \qquad c = \begin{bmatrix} -0.09 \\ 0.13 \end{bmatrix}, \qquad \sigma(\cdot) = \tanh(\cdot).$$

The pre-activation is $z = Vx_k + c$*, computed row by row:* $z_1 = 0.5(0.8) + 0.2(0.3) - 0.3(-0.4) - 0.09 = 0.49$ *and* $z_2 = 0.3(0.8) - 0.4(0.3) + 0.2(-0.4) + 0.13 = 0.17$*. Applying* σ *entrywise gives*

$$\phi(x_k) = \tanh(z) \approx (0.454,\ 0.168).$$

This feature vector has two entries, one per neuron, each obtained independently from the same input x_k *and its own row of* V *and entry of* c*.*

Combining the Layer into a Network

A layer produces a feature vector $\phi(x) \in \mathbb{R}^{n_h}$, not a scalar prediction. A second weighted sum combines the n_h feature functions into the single output. This second sum introduces a second set of weights, and the two together define the network.

Definition 3.10. *Given a regressor* $x \in \mathbb{R}^{n_x}$*, a single-hidden-layer network with inner weights* $V \in \mathbb{R}^{n_h \times n_x}$*, inner bias* $c \in \mathbb{R}^{n_h}$*, outer weights* $w \in \mathbb{R}^{n_h}$*, and outer bias* $w_0 \in \mathbb{R}$ *computes*

$$\phi(x) = \sigma(Vx + c), \qquad \hat{y} = w^\top \phi(x) + w_0,$$

that is, in one line, $\hat{y} = w^\top \sigma(Vx + c) + w_0$*.*

The network is two weighted sums in sequence, carrying two distinct sets of weights. The inner weights V, c build the feature functions from the regressor. The outer weights w, w_0 combine those functions into the output.

The reason for building F this way is representational. The feature vector $\phi(x) = \sigma(Vx+c)$ is a set of n_h nonlinear functions of the regressor, and the output $w^\top \phi(x) + w_0$ is a weighted combination of them. Changing the inner weights V, c moves and reshapes these feature functions; changing the outer weights w recombines them. A network with enough feature units can therefore take on a wide range of shapes, and identification is the process of adjusting both weight sets so that the shape it takes on matches the function generating the data. This is what distinguishes a network from the fixed-basis models earlier in this note. There the feature functions were committed in advance, as monomials in Definition 3.5 or as a chosen library in Definition 3.7, and only their weights were free. Here the feature functions themselves are learned. Figure 3 illustrates this with a scalar input: two monotone feature functions, shaped by the inner weights, combine under the outer weights into a localized function that neither feature alone can represent.

Remark 3.6. *The activation* σ *must be nonlinear for the two weighted sums to gain anything. If* σ *were the identity, Definition 3.10 would give* $\hat{y} = w^\top(Vx+c)+w_0 = (w^\top V)x+(w^\top c+w_0)$*, a single affine map from* x *to* $\hat{y}$*, of exactly the form already covered by ARX in Definition 3.2. The two weight sets would collapse into one and no new shape would be available. Common nonlinear choices are the hyperbolic tangent,* $\sigma(z) = \tanh(z)$*, the sigmoid,* $\sigma(z) = 1/(1+e^{-z})$*, and the rectified linear unit,* $\sigma(z) = \max(0, z)$*.*

Remark 3.7. *For a fixed feature vector* $\phi(x)$*, the prediction* $\hat{y} = w^\top \phi(x) + w_0$ *is linear in the outer weights* w, w_0*, in the sense of Definition 3.5. Only the inner weights* V, c *enter nonlinearly, through* σ*. Stacking the outer parameters into* $W = (w, w_0)$ *places the network in the form* $\hat{f}(x, W) = W^\top \phi(x)$ *used in Section 3.6 for the continuous-time case. There,* ϕ *is fixed in advance, for instance as a set of radial basis functions, and only* W *is estimated. Here, both* V, c *and* W *are estimated jointly, so the feature functions are themselves learned from data rather than fixed.*

Remark 3.8. *Universal Approximation Theorem. Let* σ *be nonconstant, bounded, and continuous, and let* $\Omega \subset \mathbb{R}^{n_x}$ *be compact. For every continuous* $f : \Omega \to \mathbb{R}$ *and every tolerance* $\delta > 0$*,*

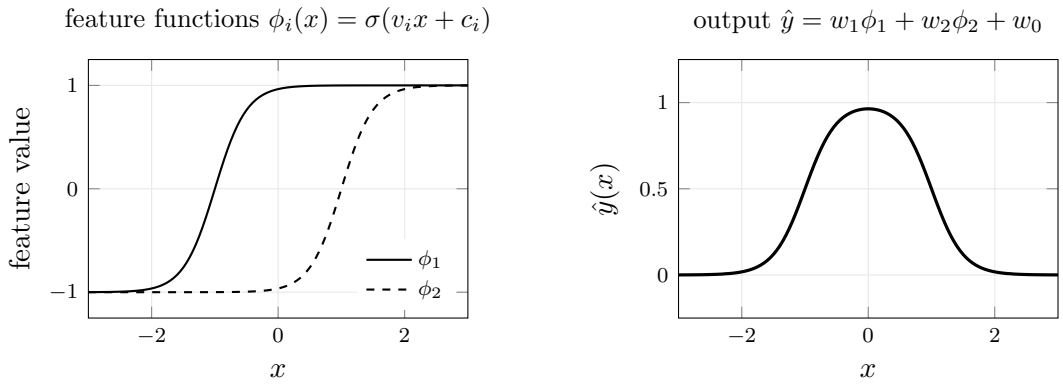

Figure 3: A single-hidden-layer network as a function builder, shown for a scalar input x. Left: two feature functions $\phi_1(x) = \sigma(2x + 2)$ and $\phi_2(x) = \sigma(2x - 2)$, both monotone sigmoids whose positions and steepness are set by the inner weights V, c. Right: the outer weights $w = (0.5, -0.5)$, $w_0 = 0$ combine them into $\hat{y}(x) = 0.5\phi_1(x) - 0.5\phi_2(x)$, a single localized bump reaching about 0.96 at $x = 0$ and near zero at the edges. The output is a shape neither feature function has on its own; the inner weights build the pieces and the outer weights assemble them.

there exist a width n_h and inner weights V, c such that, writing $\phi(x) = \sigma(Vx + c)$, some outer weights w, w_0 satisfy

$$\sup_{x \in \Omega} \left| f(x) - \left(w^\top \phi(x) + w_0 \right) \right| < \delta.$$

Equivalently, $f(x) = W^{*\top}\phi(x) + \varepsilon(x)$ for some weights $W^* = (w^*, w_0^*)$ and a residual ε with $\sup_{x \in \Omega} |\varepsilon(x)| < \delta$. The theorem controls the size of the residual, it does not remove it: for any finite width n_h, ε need not be zero, and increasing n_h can make δ smaller without making it vanish. The result is due to Cybenko (1989) for sigmoidal σ and Hornik, Stinchcombe, and White (1989) for the general case.

Remark 3.9. *The single-hidden-layer network of Definition 3.10 is the smallest case, and is the only one fit in this note. Stacking more such maps in sequence, each feature vector becoming the input of the next,*

$$\phi^{(1)}(x) = \sigma(V^{(1)}x + c^{(1)}), \quad \phi^{(l)} = \sigma\left(V^{(l)}\phi^{(l-1)} + c^{(l)}\right), \quad \hat{y} = w^\top \phi^{(L)} + w_0,$$

gives a network with several layers, the setting of deep learning. The intermediate feature vectors $\phi^{(1)}, \ldots, \phi^{(L)}$ are called hidden layers, hidden because their values are internal to the computation and are never measured, unlike the input x and the fitted output $\hat{y}$. Depth does not change the representational purpose established here; it changes how the feature functions are built. Each layer composes nonlinear functions of the layer below, so a deep network represents a target function as a composition of many simple maps rather than a single wide combination of them, which for many functions of interest reaches a given accuracy with far fewer total units than the single-layer width the Universal Approximation Theorem would require. Universal approximation already holds at the single-layer case of Remark 3.8, so the identification methods of this note do not depend on depth; the deeper networks used in practice are a more efficient route to the same end.

3.6 Neural Network Models, Continuous Time

The neural network of Section 3.5 predicted a discrete-time output $\hat{y}_{k+1}$ directly from a lagged regressor. Many systems, the pendulum among them, are more naturally described in continuous time, by a first-order differential equation $\dot{x} = f(x)$, with f unknown or only partly known.

This section builds the continuous-time counterpart of the discrete-time network already introduced, using the same architecture.

Definition 3.11. *Given a state $x \in \mathbb{R}^n$, a continuous-time neural network model approximates the dynamics as*

$$\hat{\dot{x}} = f(x, W) = W^\top \phi(x), \qquad \phi(x) = \sigma(Vx + c),$$

where ϕ is the hidden layer of Definition 3.9 and W collects the output-layer weights, as in Remark 3.7. The true dynamics are assumed to satisfy $\dot{x} = f(x) = W^{*\top}\phi(x) + \varepsilon(x)$ for some weights W^* and a residual ε, not identically zero, on the compact region of the state space where the model is used.

Remark 3.10. *$\varepsilon(x)$ is not assumed to vanish. A finite hidden layer, however large, need not represent an arbitrary f exactly; the Universal Approximation Theorem guarantees only that ε can be made small, uniformly over a compact region, by choosing ϕ rich enough, not that it can be made zero. Nothing in this note claims otherwise, and any prediction made from $\hat{\dot{x}} = W^\top \phi(x)$ should be read with this residual in mind.*

If $\dot{x}$ were measured directly, identification would proceed exactly as in Section 4.1: build a regressor from $\phi(x_k)$ at each sample, and estimate W against the measured $\dot{x}_k$, linear in W for a fixed ϕ. In practice, $\dot{x}$ is rarely measured; a sensor gives sampled states x_k at discrete times, not their instantaneous derivative. The network's role must instead be understood in terms of what is actually available: pairs of successive sampled states.

Definition 3.12. *Given sampled states x_k at times $t_k = k\Delta t$, and a continuous-time network as in Definition 3.11, the forward-Euler predictor is*

$$\hat{x}_{k+1} = x_k + \Delta t\, W^\top \phi(x_k).$$

Identification consists of choosing W so that $\hat{x}_{k+1}$ matches the measured x_{k+1}, using only the sampled states x_k, x_{k+1} and never requiring a direct measurement of $\dot{x}_k$.

Remark 3.11. *Definition 3.12 introduces a second source of error beyond $\varepsilon(x)$ in Definition 3.11: the forward-Euler step itself is only a first-order approximation of the true continuous-time trajectory over $[t_k, t_{k+1}]$, with error growing with Δt. It is worth stating explicitly here because a poor fit under Definition 3.12 may reflect too large a Δt rather than a poorly chosen W.*

Example 3.7. *Consider a pendulum with state $x = (\theta, \omega)$, θ the angle and $\omega = \dot{\theta}$ the angular velocity. Take a network with $n_h = 2$ hidden units, $\sigma = \tanh$, and weights*

$$V = \begin{bmatrix} 0.4 & -0.2 \\ -0.3 & 0.5 \end{bmatrix}, \quad c = \begin{bmatrix} 0.05 \\ -0.05 \end{bmatrix}, \quad w = \begin{bmatrix} 1.5 \\ -1.2 \end{bmatrix}, \quad w_0 = 0.1,$$

used to approximate $\ddot{\theta}$ alone, so that $\hat{\dot{x}} = (\omega, \hat{\ddot{\theta}})$ and only the second component is computed by the network. At $x_0 = (\theta_0, \omega_0) = (0.6, 0)$,

$$z = Vx_0 + c = \begin{bmatrix} 0.290 \\ -0.230 \end{bmatrix}, \qquad h = \tanh(z) = \begin{bmatrix} 0.282 \\ -0.226 \end{bmatrix}, \qquad \hat{\ddot{\theta}}_0 = w^\top h + w_0 = 0.794.$$

Applying the forward-Euler predictor of Definition 3.12 with $\Delta t = 0.05$,

$$\hat{\omega}_1 = \omega_0 + \Delta t\, \hat{\ddot{\theta}}_0 = 0.0397, \qquad \hat{\theta}_1 = \theta_0 + \Delta t\, \omega_0 = 0.600.$$

These weights were chosen arbitrarily, not fitted, and the resulting $\hat{\ddot{\theta}}_0 = 0.794$ is far from the value a correctly identified model should give; Section 4.7 fits this same network to pendulum data by gradient descent.

4 Estimation Methods

Every model in the previous section left its coefficients unknown. This section introduces some basic methods to estimate them from data, starting with least squares, then extending to the iterative methods needed once the model is no longer linear in its parameters.

4.1 Least Squares (LS)

For a model that is linear in the parameters, in the sense of Definition 3.5, the prediction at step $k+1$ is $\hat{y}_{k+1} = \theta^\top \phi_k$, where $\phi_k \equiv \phi(x_k)$ is the regressor built from measured past outputs and inputs. Given N measured pairs (ϕ_k, y_{k+1}), $k = 0, \dots, N-1$, the least squares problem is to choose θ to minimize the sum of squared prediction errors.

Definition 4.1. *Given regressors $\phi_0, \dots, \phi_{N-1}$ and measured outputs $y_1, \dots, y_N$, the least squares estimate is*

$$\hat{\theta} = \arg\min_{\theta} J(\theta), \qquad J(\theta) = \sum_{k=0}^{N-1} e_{k+1}^2 = \sum_{k=0}^{N-1} \big(y_{k+1} - \theta^\top \phi_k\big)^2.$$

Stack the regressors as rows of a matrix $\Phi \in \mathbb{R}^{N \times p}$ and the outputs into a vector $Y \in \mathbb{R}^N$,

$$\Phi = \begin{bmatrix} \phi_0^\top \\ \vdots \\ \phi_{N-1}^\top \end{bmatrix}, \qquad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}.$$

The cost in Definition 4.1 is then $J(\theta) = (Y - \Phi\theta)^\top (Y - \Phi\theta)$, a quadratic function of θ. Setting its gradient to zero,

$$\nabla_\theta J = -2\Phi^\top (Y - \Phi\theta) = 0 \quad \Longrightarrow \quad \Phi^\top \Phi\, \theta = \Phi^\top Y,$$

gives the normal equations. If $\Phi^\top \Phi$ is invertible, the unique minimizer is

$$\hat{\theta} = (\Phi^\top \Phi)^{-1} \Phi^\top Y.$$

Remark 4.1. *$\Phi^\top \Phi$ is invertible if and only if Φ has full column rank, that is, no column of Φ can be written as a linear combination of the others over the N collected data points. If $\Phi^\top \Phi$ is singular, the data record does not distinguish between some directions in parameter space, and the normal equations admit infinitely many solutions.*

Example 4.1. *Consider an AR(1) model for a stock's daily excess return, $y_{k+1} = a_1 y_k$, with five measured values $r_0 = 2.00$, $r_1 = 1.60$, $r_2 = 1.28$, $r_3 = 1.02$, $r_4 = 0.82$ (in percent). Here the regressor is scalar, $\phi_k = r_k$, and $\theta = a_1$. Stacking the four available pairs,*

$$\Phi = \begin{bmatrix} 2.00 \\ 1.60 \\ 1.28 \\ 1.02 \end{bmatrix}, \qquad Y = \begin{bmatrix} 1.60 \\ 1.28 \\ 1.02 \\ 0.82 \end{bmatrix},$$

gives $\Phi^\top \Phi = 8.7108$ and $\Phi^\top Y = 6.9672$, so

$$\hat{a}_1 = \frac{\Phi^\top Y}{\Phi^\top \Phi} = 0.800.$$

The fitted model reproduces the data closely: $\hat{a}_1 r_0 = 1.600$, $\hat{a}_1 r_1 = 1.280$, $\hat{a}_1 r_2 = 1.024$, $\hat{a}_1 r_3 = 0.816$, against the measured values $1.60, 1.28, 1.02, 0.82$.

4.2 Recursive Least Squares (RLS)

The batch estimate of Section 4.1 requires the full data record Φ, Y and a matrix inversion. In the stock-return setting, new data arrives one trading day at a time, and recomputing $\hat{\theta}$ from scratch after every new sample is wasteful. RLS updates $\hat{\theta}$ directly from its previous value each time a new pair arrives, without re-solving the normal equations.

The starting point is the batch solution itself. From Section 4.1, the least squares estimate on N pairs is

$$\hat{\theta}_N = (\Phi_N^\top \Phi_N)^{-1} \Phi_N^\top Y_N = P_N \, \Phi_N^\top Y_N, \qquad P_N = (\Phi_N^\top \Phi_N)^{-1}.$$

RLS carries exactly these two quantities, $\hat{\theta}_N$ and P_N, from one step to the next. When a new pair (ϕ_N, y_{N+1}) arrives, the goal is to express $\hat{\theta}_{N+1}$ and P_{N+1} using only $\hat{\theta}_N$, P_N, and the new pair. The derivation has three steps: update the two building blocks of the batch formula, invert the first of them, and substitute the result back into the batch formula for $\hat{\theta}_{N+1}$.

The batch formula is built from $\Phi_N^\top \Phi_N$ and $\Phi_N^\top Y_N$. Appending the new row $\phi_N^\top$ to Φ_N updates each of these by a single term,

$$\Phi_{N+1}^\top \Phi_{N+1} = \Phi_N^\top \Phi_N + \phi_N \phi_N^\top, \qquad \Phi_{N+1}^\top Y_{N+1} = \Phi_N^\top Y_N + \phi_N y_{N+1}.$$

Both are rank-one changes to quantities already known. Everything that follows is the result of propagating these two changes through the batch formula.

Step 1: update P_N. The first change is a rank-one update of $\Phi_N^\top \Phi_N$, whose inverse P_N is in hand. Its updated inverse follows from the Sherman-Morrison identity, without inverting a matrix afresh.

Remark 4.2. *For an invertible $A \in \mathbb{R}^{p \times p}$ and a vector $\phi \in \mathbb{R}^p$,*

$$(A + \phi\phi^\top)^{-1} = A^{-1} - \frac{A^{-1}\phi\phi^\top A^{-1}}{1 + \phi^\top A^{-1}\phi},$$

provided $1 + \phi^\top A^{-1}\phi \neq 0$. The right side uses only A^{-1}, already known.

Applying Remark 4.2 with $A = \Phi_N^\top \Phi_N$ and $A^{-1} = P_N$,

$$P_{N+1} = P_N - \frac{P_N \phi_N \phi_N^\top P_N}{1 + \phi_N^\top P_N \phi_N}.$$

The vector $P_N\phi_N$, scaled by the scalar denominator, recurs throughout; name it the gain,

$$K_{N+1} = \frac{P_N \phi_N}{1 + \phi_N^\top P_N \phi_N},$$

so that the covariance update reads compactly as

$$P_{N+1} = (I - K_{N+1}\phi_N^\top) P_N.$$

Step 2: one identity linking the two updates. Before updating the estimate, note a consequence of the gain definition that ties Step 1 to Step 3. Rearranging the definition gives $P_N\phi_N = K_{N+1}(1 + \phi_N^\top P_N \phi_N)$, and therefore

$$P_{N+1}\phi_N = (I - K_{N+1}\phi_N^\top) P_N \phi_N = P_N\phi_N - K_{N+1}\phi_N^\top P_N \phi_N = K_{N+1}.$$

The updated covariance applied to the new regressor returns exactly the gain. This single identity is what collapses the parameter update to its final form.

Step 3: update $\hat{\theta}_N$. Write the new estimate from the batch formula at $N+1$, the anchor applied one step later,

$$\hat{\theta}_{N+1} = P_{N+1}\,\Phi_{N+1}^{\top} Y_{N+1}.$$

Substitute the second building block, $\Phi_{N+1}^{\top} Y_{N+1} = \Phi_N^{\top} Y_N + \phi_N y_{N+1}$, and replace $\Phi_N^{\top} Y_N$ using the anchor at N rearranged, $\Phi_N^{\top} Y_N = P_N^{-1}\hat{\theta}_N$,

$$\hat{\theta}_{N+1} = P_{N+1}\big(P_N^{-1}\hat{\theta}_N + \phi_N y_{N+1}\big) = P_{N+1}P_N^{-1}\hat{\theta}_N + P_{N+1}\phi_N\, y_{N+1}.$$

The first term simplifies by Step 1, $P_{N+1}P_N^{-1} = (I - K_{N+1}\phi_N^{\top})P_N P_N^{-1} = I - K_{N+1}\phi_N^{\top}$; the second simplifies by the Step 2 identity, $P_{N+1}\phi_N = K_{N+1}$. Hence

$$\hat{\theta}_{N+1} = (I - K_{N+1}\phi_N^{\top})\hat{\theta}_N + K_{N+1}y_{N+1} = \hat{\theta}_N + K_{N+1}\big(y_{N+1} - \phi_N^{\top}\hat{\theta}_N\big).$$

The prediction-error form is a consequence of the three steps, not an assumption. The new estimate is the old one, corrected along the gain K_{N+1} by the error the old estimate makes on the new pair.

Definition 4.2. *Given $\hat{\theta}_N$ and $P_N = (\Phi_N^{\top}\Phi_N)^{-1}$, and a new pair (ϕ_N, y_{N+1}), the recursive least squares update is*

$$K_{N+1} = \frac{P_N\phi_N}{1 + \phi_N^{\top}P_N\phi_N}, \qquad \hat{\theta}_{N+1} = \hat{\theta}_N + K_{N+1}\big(y_{N+1} - \phi_N^{\top}\hat{\theta}_N\big), \qquad P_{N+1} = (I - K_{N+1}\phi_N^{\top})P_N.$$

Remark 4.3. *The term $y_{N+1} - \phi_N^{\top}\hat{\theta}_N$ is the prediction error computed with the estimate $\hat{\theta}_N$ available before the update, evaluated on the new pair. RLS corrects the previous estimate in proportion to this error, scaled by the gain K_{N+1}, without revisiting any earlier data pair directly. Since Definition 4.2 was derived from the batch normal equations by exact algebra, with no approximation, the recursive estimate equals the batch estimate of Section 4.1 at every step.*

Example 4.2. *Continue Example 4.1. Using only the first three pairs, $k = 0, 1, 2$,*

$$\Phi_3^{\top}\Phi_3 = 2.00^2 + 1.60^2 + 1.28^2 = 8.1264, \qquad \Phi_3^{\top}Y_3 = 2.00(1.60) + 1.60(1.28) + 1.28(1.02) = 6.4936,$$

giving $\hat{a}_{1,3} = 0.799$ and $P_3 = 1/8.1264 = 0.122$. The fourth pair, $\phi_3 = r_3 = 1.02$ and $y_4 = r_4 = 0.82$, arrives next. The gain is

$$K_4 = \frac{P_3\phi_3}{1 + \phi_3^2 P_3} = \frac{0.122(1.02)}{1 + (1.02)^2(0.122)} = 0.110,$$

and the updated estimate is

$$\hat{a}_{1,4} = \hat{a}_{1,3} + K_4\big(y_4 - \phi_3\hat{a}_{1,3}\big) = 0.799 + 0.110\big(0.82 - 1.02(0.799)\big) = 0.800.$$

This matches the batch estimate $\hat{a}_1 = 0.800$ obtained directly from all four pairs in Example 4.1, as Remark 4.3 guarantees it must.

Remark 4.4. *Whether $\hat{\theta}_N$ approaches the true parameter is read from the anchor formula. Writing the data as $y_{k+1} = \phi_k^{\top}\theta^* + v_{k+1}$ with noise v_{k+1}, the batch estimate is*

$$\hat{\theta}_N = P_N\,\Phi_N^{\top}Y_N = \theta^* + P_N\sum_{k=0}^{N-1}\phi_k v_{k+1},$$

so the estimation error is $\hat{\theta}_N - \theta^* = P_N\sum_k \phi_k v_{k+1}$. If the data is noise-free, $v_{k+1} = 0$, this error is exactly zero once Φ_N has full column rank: RLS reaches θ^* in finitely many steps and stays

there, rather than approaching it asymptotically. With noise, convergence requires persistent excitation, the condition that

$$\alpha I \preceq \sum_{k=N}^{N+m-1} \phi_k \phi_k^\top \preceq \beta I$$

for some $\alpha, \beta > 0$, a window $m \geq p$, and all N. The lower bound forces $\lambda_{\min}(\Phi_N^\top \Phi_N)$ to grow without bound, so $\|P_N\| = O(1/N)$; for zero-mean noise independent of the regressor the sum $\sum_k \phi_k v_{k+1}$ grows only as $O(\sqrt{N})$ in norm, and the error decays as $O(1/\sqrt{N})$. Persistent excitation is the same full-rank requirement as Remark 4.1, now read across a streaming record rather than a fixed one.

Remark 4.5. *Two practical points follow from Remark 4.4. The derivation assumed $P_N = (\Phi_N^\top \Phi_N)^{-1}$ available, which presumes the early samples already give full rank; in practice RLS is started from $P_0 = \delta I$ with large δ and arbitrary $\hat{\theta}_0$, which is ridge regression, Definition 4.7, with $\lambda = 1/\delta$, and the influence of this initialization washes out as excitation accumulates. Separately, $\|P_N\| = O(1/N)$ means the gain K_{N+1} decays to zero, so a converged RLS stops responding to new data. The forgetting factor of Remark 4.15 prevents this by holding P_N at a nonzero steady state proportional to $1 - \lambda_f$, which keeps the gain from vanishing and lets the estimate track a slowly time-varying θ, at the cost of the residual noise sensitivity the non-forgetting version drives to zero.*

4.3 Gradient Descent

The least squares solution of Section 4.1 requires the model to be linear in the parameters. This fails in three cases already encountered in this note: the ARMAX model of Definition 3.3, whose regressor contains past errors e_{k+1-l} that are not directly measured; NLIP models in the sense of Definition 3.6, such as the exponential decay and Michaelis-Menten structures of Section 3; and the neural network of Section 3, whose cost is quadratic in the output weights w, w_0 alone but not in the hidden-layer parameters V, c. In each case $J(\theta)$ is a function of θ for which no closed-form minimizer analogous to the normal equations is available, and θ must instead be found by iteration.

Definition 4.3. *Given a differentiable cost $J(\theta)$ and an initial guess θ_0, the gradient descent update is*

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta J(\theta_n),$$

where $\eta > 0$ is the step size, also called the learning rate, and $\nabla_\theta J(\theta_n)$ is the gradient of J evaluated at the current estimate.

The direction $-\nabla_\theta J(\theta_n)$ is the direction of steepest decrease of J at θ_n; for η small enough, moving in this direction decreases J. If η is too large, the update can overshoot the region where J decreases and increase the cost instead, so η must be chosen with care; Section 4.4 returns to this choice directly.

Remark 4.6. *When $J(\theta)$ is quadratic, as in Section 4.1, $\nabla_\theta J(\theta) = -2\Phi^\top(Y - \Phi\theta)$ is itself linear in θ, and setting it to zero gives the normal equations directly, in one step. Gradient descent applies to this case as well, but reaches the same minimizer only after repeated updates, each moving partway toward it. Gradient descent is used in place of the normal equations only when, as in the three cases above, no closed-form minimizer is available.*

Example 4.3. *Consider the exponential decay model of Section 3, $\hat{y} = \theta_1 e^{-\theta_2 t}$, fit to three measured points (t_k, y_k): $(0, 1.00)$, $(1, 0.82)$, $(2, 0.67)$. The cost is*

$$J(\theta_1, \theta_2) = \sum_k \left(y_k - \theta_1 e^{-\theta_2 t_k}\right)^2,$$

with partial derivatives, writing $e_k = y_k - \theta_1 e^{-\theta_2 t_k}$,

$$\frac{\partial J}{\partial \theta_1} = -2\sum_k e_k\, e^{-\theta_2 t_k}, \qquad \frac{\partial J}{\partial \theta_2} = 2\sum_k e_k\, \theta_1 t_k\, e^{-\theta_2 t_k}.$$

Starting from $\theta_1 = 1.0$, $\theta_2 = 0.1$, *the predictions are* $1.000, 0.905, 0.819$, *giving errors* $e_0 = 0$, $e_1 = -0.085$, $e_2 = -0.149$ *and* $J = 0.0293$. *Evaluating the partial derivatives gives* $\partial J/\partial\theta_1 = 0.397$ *and* $\partial J/\partial\theta_2 = -0.641$. *With step size* $\eta = 0.05$,

$$\theta_1 \leftarrow 1.0 - 0.05(0.397) = 0.980, \qquad \theta_2 \leftarrow 0.1 - 0.05(-0.641) = 0.132.$$

At the updated parameters, $J = 0.0087$*: a single gradient step reduces the cost to under a third of its starting value. Note that the two parameters moved by different amounts for the same step size, a point Section 4.4 returns to.*

4.4 Steepest Descent

Example 4.3 used a fixed step size η, chosen in advance and left unchanged across iterations. Steepest descent is the variant of gradient descent in which η is instead chosen anew at every iteration, by minimizing J exactly along the current gradient direction.

Definition 4.4. *Given a differentiable cost* $J(\theta)$, *steepest descent is the update*

$$\theta_{n+1} = \theta_n - \eta_n\, g_n, \qquad g_n = \nabla_\theta J(\theta_n), \qquad \eta_n = \arg\min_{\eta>0} J(\theta_n - \eta g_n).$$

The step size η_n *is chosen by this one-dimensional line search at every iteration, rather than fixed in advance as in Definition 4.3.*

Remark 4.7. *Gradient descent, Definition 4.3, and steepest descent, Definition 4.4, are often used as synonyms; here they are distinguished precisely. Gradient descent moves along* $-\nabla_\theta J$ *with a step size fixed by the user. Steepest descent moves along the same direction but re-optimizes the step size at every iteration, so it always decreases* J *at least as much, in that iteration, as any other choice of* η *would along that direction.*

For a quadratic cost, the line search in Definition 4.4 has a closed form. Write $J(\theta) = \theta^\top A\theta - 2b^\top\theta + c$ for a symmetric positive definite A; the least squares cost of Definition 4.1 is of this form, with $A = \Phi^\top\Phi$, $b = \Phi^\top Y$, $c = Y^\top Y$, and $g_n = \nabla_\theta J(\theta_n) = 2A\theta_n - 2b$. Substituting $\theta_n - \eta g_n$ into J and differentiating with respect to η,

$$\frac{d}{d\eta} J(\theta_n - \eta g_n) = -2g_n^\top A(\theta_n - \eta g_n) + 2g_n^\top b = -g_n^\top g_n + 2\eta\, g_n^\top A g_n,$$

using $2A\theta_n - 2b = g_n$ to simplify the first two terms. Setting this to zero gives

$$\eta_n = \frac{g_n^\top g_n}{2\, g_n^\top A g_n}.$$

Example 4.4. *Consider the quadratic cost* $J(\theta_1, \theta_2) = \theta_1^2 + 10\theta_2^2$, *of the form above with* $A = \mathrm{diag}(1, 10)$, $b = 0$. *The minimizer is* $\theta = (0, 0)$. *Starting from* $\theta_0 = (5, 1)$, *the gradient is* $g_0 = (2\theta_1, 20\theta_2) = (10, 20)$, *and*

$$\eta_0 = \frac{g_0^\top g_0}{2g_0^\top A g_0} = \frac{500}{2(4100)} = 0.0610,$$

giving $\theta_1 = (4.390, -0.220)$ *and* $J(\theta_1) = 19.76$, *down from* $J(\theta_0) = 35.0$. *Repeating:* $g_1 = (8.780, -4.390)$, $\eta_1 = 0.1786$, *giving* $\theta_2 = (2.822, 0.564)$ *and* $J(\theta_2) = 11.15$.

The second entry of θ changes sign at every iteration, $1 \to -0.220 \to 0.564 \to \ldots$, overshooting zero each time while the first entry decreases steadily. J still decreases at every step, as Definition 4.4 guarantees, but by a roughly constant factor rather than reaching the minimizer quickly: this zig-zagging is characteristic of steepest descent when the cost is far more curved in one direction than another.

Remark 4.8. *The zig-zagging in Example 4.4 is governed by the conditioning of A: with eigenvalues 1 and 10 here, the ratio $10/1 = 10$ is the condition number of A. The larger this ratio, the more elongated the level sets of J, and the more pronounced the zig-zag, since the exact-line-search direction is rarely aligned with the direction toward the minimizer. This is exactly the same conditioning that governs whether $\Phi^\top \Phi$ is well behaved in Section 4.1, and it is the reason fixed small-step gradient descent, momentum-based variants, or the quasi-Newton and conjugate gradient methods used in practice for large parameter sets, such as the neural network weights of Section 3, are generally preferred over steepest descent with exact line search.*

4.5 Gauss-Newton

Example 4.3 needed a full gradient descent step just to bring J down by a factor of three, and would need many more to approach the minimizer closely. Gauss-Newton is a method specialized to costs of the least-squares form $J(\theta) = r(\theta)^\top r(\theta)$, where $r(\theta) \in \mathbb{R}^N$ collects the residuals $r_k(\theta) = y_k - \hat{y}_k(\theta)$, and it typically reaches the minimizer in far fewer iterations by using more information about the shape of J than the gradient alone.

At a current estimate θ_n, linearize the residual in θ,

$$r(\theta_n + \Delta) \approx r(\theta_n) + J_r(\theta_n)\Delta, \qquad J_r(\theta_n) = \left.\frac{\partial r}{\partial \theta}\right|_{\theta_n} \in \mathbb{R}^{N \times p},$$

where J_r is the Jacobian of the residual vector. Substituting this linear approximation into J gives a quadratic problem in Δ,

$$\min_{\Delta} \ \big\| r(\theta_n) + J_r(\theta_n)\Delta \big\|^2,$$

of exactly the form solved in Section 4.1, with $J_r(\theta_n)$ in place of Φ and $-r(\theta_n)$ in place of Y. The normal equations give

$$J_r(\theta_n)^\top J_r(\theta_n)\, \Delta = -J_r(\theta_n)^\top r(\theta_n).$$

Definition 4.5. *Given a residual $r(\theta)$ with Jacobian $J_r(\theta)$, the Gauss-Newton update is*

$$\theta_{n+1} = \theta_n - \big(J_r(\theta_n)^\top J_r(\theta_n)\big)^{-1} J_r(\theta_n)^\top r(\theta_n),$$

provided $J_r(\theta_n)^\top J_r(\theta_n)$ is invertible.

Example 4.5. *Return to the exponential decay model of Example 4.3, $r_k(\theta) = y_k - \theta_1 e^{-\theta_2 t_k}$, at the same starting point $\theta_0 = (1.0, 0.1)$. The Jacobian entries are*

$$\frac{\partial r_k}{\partial \theta_1} = -e^{-\theta_2 t_k}, \qquad \frac{\partial r_k}{\partial \theta_2} = \theta_1 t_k e^{-\theta_2 t_k},$$

giving, at θ_0,

$$J_r(\theta_0) = \begin{bmatrix} -1.000 & 0.000 \\ -0.905 & 0.905 \\ -0.819 & 1.637 \end{bmatrix}, \qquad r(\theta_0) = \begin{bmatrix} 0.000 \\ -0.085 \\ -0.149 \end{bmatrix}.$$

Solving the normal equations of Definition 4.5 gives $\Delta = (-0.001, 0.091)$, so

$$\theta_1 = (1.0, 0.1) + (-0.001, 0.091) = (0.999, 0.191),$$

at which $J(\theta_1) = 0.00017$. A single Gauss-Newton step reduces the cost fifty-fold below the value $J = 0.0293$ at θ_0, well past the reduction to $J = 0.0087$ that a single gradient descent step achieved in Example 4.3. A second Gauss-Newton step gives $\theta_2 = (1.000, 0.200)$ with $J(\theta_2) = 1.4 \times 10^{-6}$, matching the true decay parameters used to generate the data essentially exactly.

Remark 4.9. *Gauss-Newton is a structured special case of Newton's method. The exact Hessian of $J(\theta) = r(\theta)^\top r(\theta)$ is $\nabla^2_\theta J = 2\big(J_r^\top J_r + \sum_k r_k \nabla^2_\theta r_k\big)$, and Newton's method uses this exact Hessian, $\theta_{n+1} = \theta_n - (\nabla^2_\theta J)^{-1}\nabla_\theta J$. Gauss-Newton drops the second term, keeping only $J_r^\top J_r$, which is exact when the residuals are small or the model is close to linear in θ near θ_n, and is the reason Definition 4.5 converges quickly once θ_n is already close to the minimizer, as in Example 4.5.*

Remark 4.10. *If $J_r(\theta_n)^\top J_r(\theta_n)$ is singular or poorly conditioned, in the sense of Remark 4.1 and Remark 4.8, the Gauss-Newton step of Definition 4.5 is undefined or unreliable. Levenberg-Marquardt addresses this by damping the update,*

$$\theta_{n+1} = \theta_n - \big(J_r(\theta_n)^\top J_r(\theta_n) + \lambda I\big)^{-1} J_r(\theta_n)^\top r(\theta_n),$$

with $\lambda > 0$ adjusted at each iteration: increased when a step fails to decrease J, which makes the update behave more like gradient descent, and decreased when a step succeeds, which recovers the fast convergence of Gauss-Newton.

4.6 Sequential Thresholded Least Squares (STLSQ) for SINDy

Section 3.4 presented SINDy models with the sparse coefficient matrix Ξ already fitted, deferring the fitting procedure itself to this section. Plain least squares, Section 4.1, does not by itself produce a sparse $\hat{\Xi}$: with a large library, it distributes small nonzero weight across nearly every candidate term, rather than setting the wrong ones to exactly zero. STLSQ enforces sparsity by alternating a least squares fit with a thresholding step that removes small coefficients from the model.

Definition 4.6. *Given a library Θ, a target derivative $\dot{y}$ (or, in discrete time, a target y_{k+1}), and a threshold $\gamma > 0$, the STLSQ update is*

$$\theta^{(0)} = \arg\min_\theta \|\dot{y} - \Theta\theta\|^2,$$

and, for $n = 0, 1, 2, \ldots$ until the set of nonzero entries stops changing,

$$S_n = \{i : |\theta_i^{(n)}| < \gamma\}, \qquad \theta_i^{(n+1)} = 0 \text{ for } i \in S_n, \qquad \theta^{(n+1)}_{S_n^c} = \arg\min_\theta \big\|\dot{y} - \Theta_{S_n^c}\theta\big\|^2,$$

where $\Theta_{S_n^c}$ denotes the columns of Θ not in S_n. Each refit step is an ordinary least squares problem, Definition 4.1, restricted to the surviving columns.

Example 4.6. *Continue Example 3.5. With library $\Theta(x, y) = [1,\ x,\ y,\ x^2,\ xy,\ y^2]$ and threshold $\gamma = 0.05$, the plain least squares fit for $\dot{x}$ is*

$$\theta^{(0)} = (-0.110,\ 1.100,\ 0.101,\ -0.002,\ -0.389,\ -0.020),$$

with every entry nonzero. Thresholding removes the constant, y, x^2, and y^2 terms (all below γ in magnitude), and refitting on the remaining columns $\{x, xy\}$ gives

$$\theta^{(1)}_{\{x,xy\}} = (1.077,\ -0.389), \qquad \theta^{(1)}_{rest} = 0.$$

A second threshold-and-refit removes the residual constant term picked up in the intermediate step, and a third iteration leaves the support unchanged at $\{x, xy\}$, converging to

$$\hat{\theta}_{\dot{x}} = (0,\ 1.091,\ 0,\ 0,\ -0.395,\ 0),$$

matching $\hat{\dot{x}} = 1.091\,x - 0.395\,xy$ from Example 3.5. The same procedure applied to $\dot{y}$ converges to $\hat{\theta}_{\dot{y}} = (0, 0, -0.395, 0, 0.099, 0)$, matching $\hat{\dot{y}} = -0.395\,y + 0.099\,xy$.

Remark 4.11. *STLSQ and LASSO, Remark 4.14, both produce sparse estimates, by different mechanisms: LASSO penalizes $\|\theta\|_1$ directly inside a single optimization problem, while STLSQ alternates ordinary least squares with a hard threshold. STLSQ is the estimator generally used for SINDy in the literature (Brunton, Proctor, Kutz, 2016), in part because the threshold γ has a direct, interpretable meaning, the smallest coefficient magnitude considered physically meaningful, whereas the LASSO penalty λ does not map onto a library coefficient's scale as directly.*

4.7 Gradient Descent for the Pendulum Network

The continuous-time network of Section 3.6 was evaluated in Example 3.7 at arbitrary, unfitted weights. This section takes one gradient descent step toward fitting the output weights w, w_0 to the single measured pair used there, following the same construction as Example 4.3.

Example 4.7. *Continue Example 3.7. The forward-Euler predictor gives $\hat{\theta}_1 = \theta_0 + \Delta t\,\omega_0$, independent of the network, and $\hat{\omega}_1 = \omega_0 + \Delta t\,(w^\top h + w_0)$, so only the second term of $J_{\text{data}} = (\theta_1 - \hat{\theta}_1)^2 + (\omega_1 - \hat{\omega}_1)^2$ depends on w, w_0. With the true next state $(\theta_1, \omega_1) = (0.600, -0.277)$, the prediction error is $e_\omega = \omega_1 - \hat{\omega}_1 = -0.277 - 0.040 = -0.316$, and $J_{\text{data}} = 0.100$. The gradients, writing $h = \tanh(Vx_0 + c)$ as in Example 3.7,*

$$\frac{\partial J_{\text{data}}}{\partial w} = -2e_\omega \Delta t\, h = (0.0089,\ -0.0072), \qquad \frac{\partial J_{\text{data}}}{\partial w_0} = -2e_\omega \Delta t = 0.0316.$$

With step size $\eta = 0.5$,

$$w \leftarrow (1.500, -1.200) - 0.5(0.0089, -0.0072) = (1.496, -1.196), \qquad w_0 \leftarrow 0.100 - 0.5(0.0316) = 0.084.$$

At the updated weights, $\hat{\hat{\theta}}_0 = 0.777$ and $J_{\text{data}} = 0.0995$, a small decrease from 0.100. A single data pair, through the forward-Euler predictor, supplies only a weak correction to the output weights.

Remark 4.12. *This example updates only the output weights w, w_0, which enter J_{data} linearly for fixed h. Updating the hidden-layer weights V, c requires propagating e_ω backward through* tanh *and through w, since h itself depends on V, c; this is backpropagation, and is not derived here.*

4.8 Regularization and Robustness

Remark 4.1 noted that $\Phi^\top\Phi$ can be singular, leaving the least squares normal equations without a unique solution; Remark 4.8 noted that even when invertible, $\Phi^\top\Phi$ can be poorly conditioned. This subsection addresses both issues, and two related extensions of least squares that arise from other properties of the data.

Definition 4.7. *Given regressors Φ and outputs Y as in Definition 4.1, the ridge regression estimate is*

$$\hat{\theta} = \arg\min_{\theta} J(\theta), \qquad J(\theta) = \|Y - \Phi\theta\|^2 + \lambda\|\theta\|^2,$$

for a fixed $\lambda > 0$, called the regularization parameter.

Setting $\nabla_\theta J = 0$ as in Section 4.1 gives the normal equations $(\Phi^\top\Phi + \lambda I)\theta = \Phi^\top Y$, so

$$\hat{\theta} = (\Phi^\top\Phi + \lambda I)^{-1}\Phi^\top Y.$$

Remark 4.13. *$\Phi^\top\Phi + \lambda I$ is invertible for any $\lambda > 0$, regardless of whether $\Phi^\top\Phi$ itself is singular: adding λI shifts every eigenvalue of $\Phi^\top\Phi$, which is symmetric positive semi-definite, away from zero. Ridge regression is therefore well defined exactly in the case, Remark 4.1, where plain least squares is not.*

Example 4.8. *Suppose the ARX regressor includes both a signal u_k and a scaled copy $2u_k$, so $\phi_k = (u_k, 2u_k)$, with true model $y_{k+1} = 1.0\,u_k$ and measured values $u_k = 0.5, 1.0, 1.5, 2.0$. Then*

$$\Phi^\top\Phi = \begin{bmatrix} 7.5 & 15 \\ 15 & 30 \end{bmatrix},$$

which is singular, since its second column is twice its first: plain least squares has no unique solution here. With $\lambda = 0.01$, ridge regression gives $\hat{\theta} = (0.200, 0.400)$, and the prediction error against the true model is smaller than 10^{-3} at every data point. Note $\hat{\theta}_1 + 2\hat{\theta}_2 = 1.000$, consistent with the true model $y_{k+1} = 1.0\,u_k = 1.0\,\phi_{k,1} + 0.5\,\phi_{k,2}$ written in terms of the two collinear regressors; ridge regression selects, among the infinitely many θ satisfying this relation, the one of smallest norm.

Remark 4.14. *Replacing the penalty $\lambda\|\theta\|^2$ in Definition 4.7 with $\lambda\|\theta\|_1 = \lambda\sum_i |\theta_i|$ gives LASSO regression. Unlike ridge, LASSO has no closed form, since $\|\theta\|_1$ is not differentiable at $\theta_i = 0$, but it tends to drive many entries of $\hat{\theta}$ to exactly zero rather than merely small, which ridge regression does not do. This sparsity-promoting property is the basis of the STLSQ estimator used for SINDy in Section 4.6.*

Remark 4.15. *Weighting the terms of the least squares cost unequally, $J(\theta) = \sum_k \rho_k e_{k+1}^2$ for weights $\rho_k > 0$, gives weighted least squares, with closed form $\hat{\theta} = (\Phi^\top W\Phi)^{-1}\Phi^\top WY$ for $W = \mathrm{diag}(\rho_0, \ldots, \rho_{N-1})$. The RLS update of Section 4.2 can be extended with a forgetting factor $\lambda_f \in (0, 1)$, giving each new data pair full weight and past pairs weight $\rho_k = \lambda_f^{N-k}$, decaying geometrically with age. This is weighted least squares with a specific, exponentially decaying choice of ρ_k, and lets RLS track a slowly time-varying θ by discounting old data rather than weighting all past data equally.*

Remark 4.16. *Least squares, Definition 4.1, treats Φ as known exactly and attributes all error to Y. In the AR and ARX models of Section 3, the regressor Φ is built from lagged output values, which are themselves noisy measurements, not known exactly. Total least squares accounts for this by allowing both Φ and Y to be perturbed, and finding the smallest joint perturbation, in a combined matrix $[\Phi\ Y]$, that makes the system exactly consistent; the solution is obtained from the singular value decomposition of $[\Phi\ Y]$ rather than from the normal equations. We do not develop this further here.*

Remark 4.17. *Least squares gives a biased estimate when the regressor is correlated with the error, which is exactly the situation flagged for ARMAX in Section 4.3: past errors appear in the ARMAX regressor itself. Instrumental variables address this by replacing $\Phi^\top\Phi$ in the normal equations with $Z^\top\Phi$, for a chosen instrument Z correlated with Φ but not with the error, giving $\hat{\theta} = (Z^\top\Phi)^{-1}Z^\top Y$. Choosing a suitable Z for a given model is itself nontrivial, and a full treatment is beyond the scope of this note.*

5 Model Validation

A data-driven model, whichever route produced it, an AR/ARX/ARMAX fit, a SINDy library, or a fitted network, is only as trustworthy as the data used to build it, and this matters more,

not less, when there is no physical law to check the fit against, as is the case throughout this note.

Train-test split. For time-series data such as returns, this split must respect time: the model is fit on an earlier window and tested on a strictly later one, never on data shuffled out of order, since a model that has effectively seen the future will appear far more accurate than it is. This is often called an out-of-time or walk-forward test in finance specifically to emphasize the point.

Residual analysis. Even on the training data, the pattern of leftover error, $e_{k+1} = y_{k+1} - \hat{y}_{k+1}$ for the AR/ARX/ARMAX/NARMAX family, is informative. Residuals that look like unstructured noise suggest the model has captured the dominant dynamics; residuals with clear structure suggest that something systematic has been left unmodeled.

Sparsity recovery for SINDy. Beyond the residual's size, a SINDy fit should be checked against its actual object of inference: which library terms survived thresholding, not only how small J is. A small residual obtained by keeping many small, spurious coefficients is not evidence that the correct structure was found; the check in Example 4.6, that the recovered support matched the terms known to generate the data, is available whenever a portion of the true structure is independently known, and is not replaced by residual size alone.

Remark 5.1. *An AR or NARMAX model trained on a calm trading period may show excellent residuals when tested on more data from the same calm period, and yet fail badly the moment markets enter a period of crisis-level volatility that the training window never sampled. Good validation numbers on data drawn from a narrow historical regime are not evidence that a model may be extrapolated to a different regime; the same caution applies to a SINDy library that fit one regime's data well, and to a network fit on that regime alone.*

6 Summary

Every model in this note was built without deriving anything from a physical law alone, only from data and a choice of model class. We began with a small, interpretable structure, one signal predicted from its own past, and grew it in stages: adding an exogenous input, then a moving-average term, then dropping a fixed linear form altogether in favor of a large candidate library narrowed by sparsity, and finally a structure whose functional form is itself learned from data. This progression traces the white-box to black-box spectrum of Definition 2.2 directly: AR and ARX sit close to the black-box end, structure imposed only by the choice of lagged regressors; SINDy narrows a large black-box library down to a sparse, interpretable result, the point at which partial knowledge of the candidate terms enters; and the neural network is fully black-box, its feature functions learned entirely from data.

Fitting any of these models came down to two situations. When a model is linear in its parameters, the coefficients are found in closed form, by least squares, and updated one sample at a time by its recursive form. When it is not, whether because the regressor includes unmeasured errors, because sparsity must be enforced explicitly, or because the parameters enter through an activation function, an iterative search is required instead. Gradient descent provides the general method, steepest descent refines its step size by an exact line search, Gauss-Newton exploits the specific structure of a sum-of-squares cost to converge faster than either, and STLSQ adapts least squares itself to enforce sparsity by alternating fitting and thresholding. Regularization addressed what happens when the data itself does not pin down a unique answer.

None of this comes with a physical law to check it against, which is why validation carried more weight here than in earlier notes: how a model performs on data it was not trained on, and, for SINDy specifically, whether the model recovered the right structure and not merely a small residual, is the evidence that it can be trusted at all. The next note takes up the case where inference targets a representation of the system directly, an operator on a lifted space of observables, or a distribution over functions, rather than coefficients within an assumed form.