Using Fourier-Neural Recurrent Networks to Fit Sequential Input/Output Data

Renée Koplon
Dept. of Mathematics and Statistics
Wright State University
Dayton, Ohio 45435

Eduardo D. Sontag
Dept. of Mathematics
Rutgers University
New Brunswick, New Jersey 08903

March 1, 1995
Revised: September 15, 1996

Abstract

This paper suggests the use of Fourier-type activation functions in fully recurrent neural networks. The main theoretical advantage is that, in principle, the problem of recovering internal coefficients from input/output data is solvable in closed form.

Keywords: recurrent neural networks, identification, nonlinear dynamics

1 Introduction

Neural networks provide a useful approach to parallel computation. The subclass of *recurrent architectures* is characterized by the inclusion of feedback loops in the information flow among processing units. With feedback, one may exploit context-sensitivity and memory, characteristics essential in sequence processing as well as in the modeling and control of processes involving dynamical elements. Recent theoretical results about neural networks have established their universality as models for systems approximation as well as analog computing devices (see e.g. [16, 13]).

The use of recurrent networks has been proposed in areas as varied as the design of control laws for robotic manipulators, in speech recognition, speaker identification, formal language inference, and sequence extrapolation for time series prediction. In spite of their attractive features, recurrent networks have not yet attained as much popularity as one might expect, compared to the feedforward nets so ubiquitous in other applications. One important reason for this is that training ("learning") algorithms for recurrent nets suffer from serious potential limitations. The learning problem is that of finding parameters

that "fit" a general form to training (experimental) data, with the goal of obtaining a model which can subsequently be used for pattern recognition and classification, or for extrapolation of numerical values.

Various learning methodologies for recurrent networks have been proposed in the literature, and have been used in applications. All algorithms attempt to achieve the optimization of a penalty criterion by means of steepest descent, but due to memory and speed constraints, they usually only involve an estimate of the gradient. They differ on the approximation used, and thus on their memory requirements and convergence behavior. Among these are "recurrent backpropagation," "backpropagation through time," and the "real time recurrent learning" algorithm. In complex applications involving large models, it is difficult to simultaneously achieve good convergence speeds and reasonable accuracy.

It has been pointed out by many authors that the field of control theory, with its emphasis on dynamics and optimization, is naturally related to recurrent nets. It was shown that at least some algebraic issues such as testing for identifiability and observability of system parameters can be handled analogously to the case of linear systems (see [1, 2, 3]). In a related context, recurrent net models have been proposed, and used by many authors, for adaptive control and system identification applications (see e.g. [10, 12]). In this paper, we propose a control theory-based technique for the initial estimation of weights. Most implementations of recurrent network learning algorithms assume a random initialization of weights. We suggest instead the use of nontrivial techniques from nonlinear synthesis, based on mathematical developments that have taken place during the past few years, for the initial choice of such weights.

Comparison With Linear Systems

Historically linear systems have been used to approximate various nonlinear systems since the parameters can be identified easily. Linear system identification is a well-established and widely used tool; see for instance [9]. In broad ranges of operation, linearity is a reasonable simplification, as the success of the linear theory attests to. However, linear systems are limited. Most real world processes are inherently nonlinear. In many areas, linear simplifications are not valid, which means that nonlinear approaches must be tried.

In searching for a canonical form for approximating all nonlinear systems, we would like a class of nonlinear dynamical systems that is representationally rich, like recurrent neural nets, but at the same time parameter-identifiable, like linear systems. At the very least we want a class for which realization (noise-free identification) algorithms are simple, and an algebraic structure theory similar to that available for linear systems is possible.

Motivated by this need to broaden the scope of the linear theory, and to utilize the advantages of neural nets, this paper aims to describe the development of techniques for parameter reconstruction in a class of nonlinear systems that we call *Fourier-neural recurrent networks*. The main departure from current practice in neural networks is that we

suggest the use of sine and cosine activations (or, more precisely, because it makes the theoretical development far simpler, a complex-exponential) rather than the "sigmoids" that are used in most applications. This allows the design of parameter identification methods in closed-form. Within the resulting mathematical class of system models, parameter reconstruction can be accomplished in a theoretically justified and methodical fashion, at least in the noise-free case and if the responses to appropriate regularly spaced inputs can be obtained.

This work is a first step towards an eventual theory of identification from noisy data for such systems. Viewing a system outside our class as a "noisy version" of a recurrent net, just as is routinely done when using linear techniques for nonlinear plants, may allow this approach to be broadly applicable. The approximation theorems in [15] and [17] justify such potential applications. In those papers, sigmoidal activation functions are used, but the theory can be adapted to the complex exponential activation functions used here.

The Approach Here

The approach taken here is inspired by the major developments in [1] (for continuous-time) and [2] (for discrete-time) that internal coefficients or "weights" of a recurrent net can be determined from external dynamical behavior in an essentially unique manner. Even though intrinsically nonlinear, the actual implementation of the algorithm involves a number of *subtasks* that are routinely carried out in *linear* identification theory, an area of control engineering that has had major practical successes, as remarked above.

We show that the parameters of an exp-system can be identified using a technique that avoids all the pitfalls of gradient descent, at least assuming low noise and a judicious choice of test inputs ("learning with queries"). Section 7 describes a preliminary computer implementation of the algorithm described in Section 5 utilizing averaging in order to handle a small amount of noise. Section 7 also includes simulation results.

2 Main Results

We first define general dynamic systems, since many concepts can be defined at that level of generality, and then specialize to recurrent networks.

A discrete-time initialized *input/output dynamical system* consists of a state space $\mathcal{X}$ with an initial state x_0, an input value space $\mathcal{U}$, an output value space $\mathcal{Y}$, and a transition function $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ that computes the state at time $t+1$ from the state and input values at time t. There is also an output function $h : \mathcal{X} \to \mathcal{Y}$ that assigns an output value to the current state. The equations are written as

$$x(t+1) = f(x(t), u(t)), \qquad y(t) = h(x(t)).$$

The dimension of the state space $\mathcal{X}$ is called the dimension of the system.

The system model that we deal with here, a recurrent neural network with one input and one output, can be seen as a nonlinear input/output system with a state $x \in \mathbb{C}^n$, a scalar input, and a scalar output. Let σ be a fixed nonlinear function, which we call an *activation function*. Let $A \in \mathbb{C}^{n\times n}, B \in \mathbb{C}^{n\times 1}, C \in \mathbb{C}^{1\times n}$. Recurrent neural networks are described by equations

$$\begin{array}{rcl} x(t+1) & = & \vec{\sigma}(Ax(t)+Bu(t)) \\ y(t) & = & Cx(t) \\ x(0) & = & x_0, \end{array} \tag{1}$$

where $\vec{\sigma}(x) = (\sigma(x_1), \ldots \sigma(x_n))'$. We use $\sigma(x) = \exp(ix)$ as the activation function and call a system as in (1) with this activation function an *exp-system* or *Fourier-neural recurrent network*, denoted $\Sigma = (A, B, C)_\sigma$. We use the subscript x_0 to denote the system with initial state x_0, when necessary, Σ_{x_0}.

This choice of σ as an activation function is very natural, since we obtain in this manner a dynamic extension of Fourier analysis. Moreover, the choice of this σ is critical to our results. Indeed, our method depends in an essential way on being able to transform sums into products, a property which uniquely (among continuous functions) forces us to consider this activation. (A small variation would consist of using real as opposed to complex exponentials, but this seems less interesting as it leads to highly unstable dynamics.) The idea of using Fourier-type activations has already appeared before in the neural network literature; see for instance [5] for an application in the context of feedforward nets and function approximation.

Before stating our main result, dealing with identification of the parameters, we need to briefly address the question of observability, that is, the possibility of determining the internal state from input/output data, for this class of systems. Notice that if the initial state x_0 is unknown, it too is a parameter that should be identified. A control system is said to be *observable* if given any two distinct initial states, x_0 and z_0, there is a finite sequence of inputs that causes the two systems, Σ_{x_0} and Σ_{z_0}, to have a different output. We will look at this standard notion of observability rather than the perhaps more desirable notion of actually determining the unknown initial state. If our goal is to be able to simulate the input/output behavior of the system, we do not need to identify the unknown initial state. It is enough to measure the inital output and to be able to determine the state at time 1, as well as the parameters A, B, and C. Nevertheless, we characterize observability for this class of systems.

Lemma 2.1 Let $\Sigma = (A, B, C)_\sigma$ be an exp-system such that

a. the components $b_1, \ldots, b_n$ of B are nonzero and $\sigma(b_1), \ldots, \sigma(b_n)$ are distinct, and

b. the entries of C are all nonzero.

Then Σ is observable if and only if

$$\ker C \cap A^{-1}(\mathbb{Z}^n) = \{0\}. \tag{2}$$

This is proved in Section 3. Condition (2) is easily seen to be generic. Indeed, for any fixed nonzero $k \in \mathbb{C}^n$, the $(n+1) \times (n+1)$ matrix

$$\begin{pmatrix} A & k \\ C & 0 \end{pmatrix}$$

has full rank for all A and C except those that are described by the zero set of an analytic function. Thus all nonzero $k \in \mathbb{Z}^n$ are avoided for (A, C) in the complement of a countable union of such sets. The two assumptions in the theorem are important for proving the sufficiency part of our result, because they allow certain matrices to be inverted.

The following technical condition will be used in the identification theorem. Let $B = (b_1, \ldots, b_n)' \in \mathbb{C}^n$ and assume again that $\sigma(b_1), \ldots, \sigma(b_n)$ are distinct. Let

$$B_\sigma = \begin{pmatrix} \sigma(b_1) - 1 & \cdots & \sigma(b_n) - 1 \\ \vdots & & \vdots \\ \sigma\left((n+1)b_1\right) - 1 & \cdots & \sigma\left((n+1)b_n\right) - 1 \end{pmatrix},$$

and let R(B) be the condition that the column space of B_σ does not contain any nonzero integer vectors. That is,

$$col\, B_\sigma \cap \mathbb{Z}^{n+1} = \{0\}. \tag{R(B)}$$

Although it can be proved to be generic, there are exceptions to Condition R(B). For example, if any of $\sigma(b_1), \ldots, \sigma(b_n)$ are integers, the condition fails. This condition is used in the proof to guarantee that a certain equation has a unique solution. In that situation, any two solutions would have to differ by an integer vector and R(B) then implies that the two solutions are not, in fact, distinct. It should be possible to eliminate the need for this condition in the theorem below by modifying the part of the procedure described in the proof where we find C, to use varying inputs, but the algorithm that we give is simpler if we assume R(B).

Next is the identification result. It states that the parameters of a Fourier network can be recovered using a finite set of input/output data all starting from the same unknown initial state. More than one set of input/output data is needed, but the training program does not need the complete input/output behavior as a function.

Theorem 1 *Let Σ_{x_0} be a single input, single output system as in (1), with $\sigma(x) = \exp(ix)$. Assume that*

1. *the components of B are nonzero, strictly increasing in modulus, and $b_j - b_k \notin 2\pi\mathbb{Z}$ for $j \neq k$,*

2. *the entries of* C *are all nonzero, and*

3. B *satisfies* R(B).

If the dimension of the system is known, then the input/output data is sufficient to uniquely determine the parameters A, B, C.

If in addition the system is observable, then the initial state x_0 *is also uniquely characterized.*

The components of B are uniquely identifiable only up to a change in order. Thus, we assume that the components of B are initially in increasing order and as we identify them, we put them in the correct order. The assumptions that $b_j - b_k \notin 2\pi\mathbb{Z}$ and the entries of C are nonzero are made, as in the observability theorem, to allow us to invert certain matrices. Regarding the assumption that the dimension of the system is known, this is a standard assumption in neural nets and more generally in function approximation, where problems are solved by means of search procedures over given parameter spaces. In linear systems theory, analogously, one assumes a fixed dimension, in most identification and adaptive control problems. Methods for estimating dimensions are a matter for future research, but this is an area that is not well-understood even for function approximation (viz. work on regularization in numerical analysis, structural risk in learning theory, etc).

The theorem is proved by means of an explicit procedure for reconstructing A, B, and C, which is described in Section 4.

3 Proof of Observability Lemma

We now prove Lemma 2.1 First assume that Σ is observable. Suppose there exists a nonzero $v \in \mathbb{C}^n$ such that

$$Cv = 0 \quad \text{and} \quad Av \in \mathbb{Z}^n.$$

Then the initial states 0 and $x_0 = 2\pi v$ are indistinguishable, contradicting observability. To see that 0 and x_0 are indistinguishable, note that

$$C0 = Cx_0,$$

so the outputs at time zero are the same. Next, the states at time one are equal for the two initial states, after applying *any* input. Indeed, fix any input value u. For the initial state x_0,

$$x(1) = \vec{\sigma}(Ax_0 + Bu) = \begin{pmatrix} \sigma(A_1 x_0)\sigma(b_1 u) \\ \vdots \\ \sigma(A_n x_0)\sigma(b_n u) \end{pmatrix},$$

where A_j is the jth row of A. Since $Ax_0 \in 2\pi\mathbb{Z}^n$, $\sigma(A_j x_0) = 1 = \sigma(A_j 0)$ for all j. Thus $x(1) = \vec{\sigma}(A0 + Bu)$ which is equal to the state at time one for initial state 0. Hence the outputs from time one and on are equal.

For the converse, suppose there exists a pair $x_0, \hat{x}_0$ that is indistinguishable. Then

$$\begin{aligned} Cx_0 &= C\hat{x}_0, \quad and \\ C\vec{\sigma}(Ax_0 + Bu) &= C\vec{\sigma}(A\hat{x}_0 + Bu), \quad \forall u. \end{aligned}$$

We can apply the above equality for the n different input values $0, 1, \ldots, n-1$. Since $\sigma(kb) = \sigma(b)^k$,

$$\begin{pmatrix} c_1\sigma(A_1x_0) & \cdots & c_n\sigma(A_nx_0) \end{pmatrix} \begin{pmatrix} \sigma(b_1)^0 & \cdots & \sigma(b_1)^{n-1} \\ \vdots & & \vdots \\ \sigma(b_n)^0 & \cdots & \sigma(b_n)^{n-1} \end{pmatrix} =$$

$$\begin{pmatrix} c_1\sigma(A_1\hat{x}_0) & \cdots & c_n\sigma(A_n\hat{x}_0) \end{pmatrix} \begin{pmatrix} \sigma(b_1)^0 & \cdots & \sigma(b_1)^{n-1} \\ \vdots & & \vdots \\ \sigma(b_n)^0 & \cdots & \sigma(b_n)^{n-1} \end{pmatrix}.$$

The matrix appearing on both sides of the above equation is a Vandermonde matrix. That is, it is a matrix of the form

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & & & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{pmatrix}.$$

with $x_i \neq x_j$ for all $i \neq j$. Thus it is invertible, yielding

$$\begin{pmatrix} c_1\sigma(A_1x_0) & \cdots & c_n\sigma(A_nx_0) \end{pmatrix} = \begin{pmatrix} c_1\sigma(A_1\hat{x}_0) & \cdots & c_n\sigma(A_n\hat{x}_0) \end{pmatrix}.$$

Dividing out the components of C (which are nonzero by assumption), we obtain

$$\vec{\sigma}(Ax_0) = \vec{\sigma}(A\hat{x}_0),$$

so $A(x_0 - \hat{x}_0) = 2\pi k$ for some integer vector k. Let $v = (x_0 - \hat{x}_0)/2\pi \neq 0$. Then $Cv = 0$, and $Av = k$ contradicting condition (2) in the lemma. ∎

The condition in the Lemma does not always hold, as we can see from the simple example

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 1 \end{pmatrix}.$$

The two initial states

$$x_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \hat{x}_0 = \begin{pmatrix} 2\pi \\ -2\pi \end{pmatrix}$$

are indistinguishable.

4 The Basic Algorithm

Suppose Σ is a system of known dimension n that is modeled by equations such as (1), but with unknown values of A, B, C, x_0. Assuming we may reset the system to the unknown initial state and apply any input to obtain the corresponding output, we can identify the parameters A, B, C using the following procedure. In the procedure, we input a finite set of controls and then view the list of outputs as the output sequence of a different system. This other system is linear, allowing the use of linear realization techniques.

Finding B

To compute the values of B, first apply integer-valued inputs of length one equal to $0, 1, \ldots, 2n-1$. The resulting $2n$ output values are

$$C\vec{\sigma}(Ax_0),\ C\vec{\sigma}(Ax_0+B), \ldots, C\vec{\sigma}(Ax_0+(2n-1)B). \tag{3}$$

Let $\tilde{C}$ be the row vector

$$\tilde{C} = \begin{pmatrix} c_1\sigma(A_1x_0) & c_2\sigma(A_2x_0) & \cdots & c_n\sigma(A_nx_0) \end{pmatrix},$$

where c_j is the jth component of the row vector C and A_j is the jth row of A. Then the sequence (3) can be written as

$$\tilde{C}\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \quad \tilde{C}\begin{pmatrix} \sigma(b_1) & & \\ & \ddots & \\ & & \sigma(b_n) \end{pmatrix}\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \ldots,$$
$$\tilde{C}\begin{pmatrix} \sigma(b_1)^{2n-1} & & \\ & \ddots & \\ & & \sigma(b_n)^{2n-1} \end{pmatrix}\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}.$$

These terms can be viewed as the first $2n$ Markov parameters (or impulse response parameters) for an n-dimensional linear system whose "A" matrix has eigenvalues exactly equal to $\sigma(b_1), \ldots, \sigma(b_n)$. These eigenvalues can be found by applying linear realization techniques to the sequence. This step of the algorithm is based on Hankel-matrix techniques that are classical in the context of linear recurrences and their multivariable extensions developed in control theory (see a detailed discussion in Chapter 5 of [14]). The method appears in many other areas; for instance in coding theory for decoding BCH codes, and in learning theory for sparse polynomial interpolation (see section 3 in the paper [4]). Specifically, if we denote the sequence in (3) by $h_1, h_2, \ldots h_{2n}$, we form the two Hankel matrices

$$H_1 = \begin{pmatrix} h_1 & \cdots & h_n \\ & \ddots & \\ h_n & \cdots & h_{2n-1} \end{pmatrix}, \quad H_2 = \begin{pmatrix} h_2 & \cdots & h_{n+1} \\ & \ddots & \\ h_{n+1} & \cdots & h_{2n} \end{pmatrix}. \tag{4}$$

The matrix A_{obs} satisfying

$$A_{\text{obs}} H_1 = H_2$$

is the observability form of the "A" matrix for the triple

$$\tilde{C}, \quad \begin{pmatrix} \sigma(b_1) & & \\ & \ddots & \\ & & \sigma(b_n) \end{pmatrix}, \quad \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}.$$

The eigenvalues of A_{obs} are equal to $\sigma(b_1), \ldots, \sigma(b_n)$.

In order to compute $b_1, \ldots, b_n$ themselves, one would need to apply the "inverse" of σ. This is slightly more complicated than it seems, since the complex exponential is not one-to-one. To recover the information lost by taking the complex exponential, we repeat the procedure described above, but instead of applying the first $2n$ nonnegative integer inputs, we apply inputs equal to $0, \lambda, 2\lambda, \ldots, (2n-1)\lambda$, where λ is theoretically irrational (or "close" to it for a computer implementation). By comparing the two sets of resulting eigenvalues we can determine the correct log.

The procedure is as follows. First, from the integer inputs we obtain a vector B_1, by taking any determination of the logarithm. Observe that this necessarily differs from the true B by an integer multiple of 2π

$$B_1 = B + 2\pi r, \quad r \text{ an integer vector.} \tag{5}$$

If we could determine r, we would know B, which is our goal. Now from the second set of inputs we obtain the values $\sigma(\lambda b_j)$, and from these values we obtain some possibly different estimate, B_2, again using any log. Here we know that

$$B_2 = B + \frac{2\pi}{\lambda} s, \quad s \text{ an integer vector.} \tag{6}$$

From (5) and (6) we deduce

$$\frac{B_1 - B_2}{2\pi} + \frac{s}{\lambda} = r. \tag{7}$$

The integer vectors r and s are unique. To see this, suppose there were two possible solutions to (7), (r_1, s_1) and (r_2, s_2). Then

$$r_1 - \frac{s_1}{\lambda} = r_2 - \frac{s_2}{\lambda},$$

or

$$\frac{r_1 - r_2}{s_1 - s_2} = \frac{1}{\lambda}.$$

The last equality is impossible since the left hand side is rational while the right side is irrational. Hence, there cannot be two possible solutions. Theoretically, then, from the input/output data one has the vector

$$v = \frac{B_1 - B_2}{2\pi}.$$

There is a unique solution (r, s) to the equation

$$r - \frac{s}{\lambda} = v.$$

Thus the true B is recovered as $B = B_1 - 2\pi r$. Of course, this does not yet provide an explicit procedure.

For the computer implemention, of course, λ cannot actually be irrational. But if we assume r and s are reasonably small (which corresponds to the entries of B being reasonably small, since the determinations of log used by most computer systems have arguments in $[-\pi, \pi]$ or $[0, 2\pi]$), then a λ that is not close to being the quotient of two small integers will work.

Using (7) we can, in principle, find the unique integer vectors r and s and recover the true B as follows. The simplest idea for this step is to exhaustively search through integer values for s until the left hand side of (7) is equal to a vector of integers. A preliminary problem is that we are assuming the components of B are distinct, and increasing in modulus, but the components of B_1 and B_2 may not be in the right order. So we need to match up each component of B_1 with the corresponding component of B_2.

Start by letting $s = 0$ and compare the first component of B_1 with the first component of B_2. If

$$\frac{B_1(1) - B_2(1)}{2\pi}$$

is not an integer, then compare $B_1(1)$ to $B_2(2), B_2(3), \ldots$ and so on. That is, test if

$$\frac{B_1(1) - B_2(k)}{2\pi}$$

has an integer value for some $k = 1, \ldots, n$, and rearrange B_2 such that $B_2(k)$ is now the first component of B_2, then proceed to comparing $B_1(2)$ with $B_2(2)$. Otherwise, try the next s,

$$s := -s + \begin{cases} 1 & \text{if} \quad s \leq 0 \\ 0 & \text{if} \quad s > 0, \end{cases}$$

and repeat the evaluation of

$$\frac{B_1(1) - B_2(j)}{2\pi} + \frac{s}{\lambda},$$

for j ranging from 1 to n. Continuing in this way, obtain a rearranged vector B_2 and two integer vectors r and s. When integer match-ups are found for all entries, we are done. The true B can be then computed either from the pair B_1, r by sorting the components of

$$B = B_1 - 2\pi r,$$

or from the pair B_2, s by sorting the vector

$$B = B_2 - \frac{2\pi}{\lambda} s.$$

Both yield the same answer, so in the actual implementation it is not necessary to store both r and s.

Of course, such an exhaustive search may not be practical for systems of large dimension. But for moderate sizes and for vectors B that are a priori subject to bounds (such that the entries of r and s are small integers), our experience is that this step represents only a minor part of the time needed to execute our procedure.

Finding C

The next step in the identification procedure is to determine the vector C. For this, apply inputs of length two with integer values ranging from 0 to $n+1$ for the first input and 0 to $n-1$ for the second. We introduce the $(n+2) \times n$ dimensional matrix Y composed of the corresponding output values

$$Y = \begin{pmatrix} y_{0,0} & y_{0,1} & \cdots & y_{0,n-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,n-1} \\ \vdots & & & \vdots \\ y_{n+1,0} & y_{n+1,1} & \cdots & y_{n+1,n-1} \end{pmatrix}.$$

The main observation at this point is that Y may be factored as $Y = \tilde{C}_u V$, where V is the Vandermonde matrix

$$V = \begin{pmatrix} 1 & \sigma(b_1) & \cdots & \sigma(b_1)^{n-1} \\ 1 & \sigma(b_2) & \cdots & \sigma(b_2)^{n-1} \\ \vdots & & & \vdots \\ 1 & \sigma(b_n) & \cdots & \sigma(b_n)^{n-1} \end{pmatrix}.$$

To keep the notation clear, from now on we display the special case $n = 2$, but the same procedure works in general. We can write, as just explained,

$$Y = \tilde{C}_u \begin{pmatrix} \sigma(0) & \sigma(b_1) \\ \sigma(0) & \sigma(b_2) \end{pmatrix} \tag{8}$$

where

$$\tilde{C}_u = \begin{pmatrix} c_1\sigma(A_1\vec{\sigma}(Ax_0)) & c_2\sigma(A_2\vec{\sigma}(Ax_0)) \\ c_1\sigma(A_1\vec{\sigma}(Ax_0+B)) & c_2\sigma(A_2\vec{\sigma}(Ax_0+B)) \\ c_1\sigma(A_1\vec{\sigma}(Ax_0+2B)) & c_2\sigma(A_2\vec{\sigma}(Ax_0+2B)) \\ c_1\sigma(A_1\vec{\sigma}(Ax_0+3B)) & c_2\sigma(A_2\vec{\sigma}(Ax_0+3B)) \end{pmatrix},$$

c_j is the jth component of C, b_j is the jth component of B, and A_j is the jth row of A. Observe that Y is known (from the data) and the last matrix on the right in (8) is a Vandermonde matrix, also already known. Thus, we can solve the linear equation and obtain $\tilde{C}_u$.

Next observe that for each column of $\tilde{C}_u$,

$$\tilde{C}_{u,j} = \vec{\sigma}\left[\begin{pmatrix} 1 & \sigma(b_1)^0 & \sigma(b_2)^0 \\ 1 & \sigma(b_1)^1 & \sigma(b_2)^1 \\ 1 & \sigma(b_1)^2 & \sigma(b_2)^2 \\ 1 & \sigma(b_1)^3 & \sigma(b_2)^3 \end{pmatrix}\begin{pmatrix} v_j \\ A_{j1}\sigma(A_1x_0) \\ A_{j2}\sigma(A_2x_0) \end{pmatrix}\right], \tag{9}$$

where $\sigma(v_j) = c_j$. Let V_e be the extended Vandermonde type matrix on the right hand side of (9). The goal here is to solve for the last vector on the right, call it h_j. Then for each j we can use the first component of h_j to find c_j, and the rest of the vector will be used to recover A.

Remark 4.1 The $(n+1)\times(n+1)$ dimensional Vandermonde matrix

$$\begin{pmatrix} 1 & \sigma(b_1) & \cdots & \sigma(b_n) \\ \vdots & & & \vdots \\ 1 & \sigma\left((n+1)b_1\right) & \cdots & \sigma\left((n+1)b_n\right) \end{pmatrix}$$

has rank $n+1$. After performing an elementary column operation, we see the matrix

$$\begin{pmatrix} 1 & \sigma(b_1)-1 & \cdots & \sigma(b_n)-1 \\ \vdots & & & \vdots \\ 1 & \sigma\left((n+1)b_1\right)-1 & \cdots & \sigma\left((n+1)b_n\right)-1 \end{pmatrix}$$

also has rank $n+1$. Thus, the matrix B_σ has full column rank so if R(B) holds, $B_\sigma v \in \mathbb{Z}^{n+1}$ $\Rightarrow B_\sigma v = 0$ which in turn implies that $v = 0$. □

Lemma 4.2 Assume B satisfies Condition R(B). If $\sigma(V_e\xi) = \sigma(V_e\hat{\xi})$ for $\xi, \hat{\xi} \in \mathbb{C}^{n+1}$, then $\xi_1 - \hat{\xi}_1 \in 2\pi\mathbb{Z}$ and $\xi_j = \hat{\xi}_j$ for $j = 2, \ldots, n+1$.

Proof. Let $\omega = \xi - \hat{\xi}$. We need to show that if $V_e\omega \in 2\pi\mathbb{Z}^{n+2}$, then $\omega = (2\pi k, 0, \ldots, 0)'$, or equivalently,

$$V_e\omega \in \mathbb{Z}^{n+2} \quad \Rightarrow \quad \omega = (k, 0, \ldots, 0)'.$$

Assume $V_e\omega \in \mathbb{Z}^{n+2}$. Then

$$\begin{aligned} \omega_0 + \omega_1 + \cdots + \omega_n &= k_0 \\ \omega_0 + \sigma(b_1)\omega_1 + \cdots + \sigma(b_n)\omega_n &= k_1 \\ &\vdots \\ \omega_0 + \sigma(b_1)^{n+1}\omega_1 + \cdots + \sigma(b_n)^{n+1}\omega_n &= k_{n+1}. \end{aligned}$$

Subtracting the first equation from all of the others gives

$$B_\sigma \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_n \end{pmatrix} = \begin{pmatrix} k_1 - k_0 \\ \vdots \\ k_{n+1} - k_0 \end{pmatrix}$$

which implies that $\omega_1 = \cdots = \omega_n = 0$. Thus $\omega_0 = k_0$, and so $\omega = (k_0, 0, \ldots, 0)'$. ∎

We just showed that for each j, there is a unique solution for h_j, up to the addition of an integer multiple of 2π in the first coordinate. This does not contradict the uniqueness of C since for each $j = 1, \ldots, n$,

$$c_j = \sigma(v_j) = \sigma(v_j + 2\pi k), \quad \forall k \in \mathbb{Z}.$$

The procedure that we implemented for finding h_j is as follows. Pick any componentwise log of $\tilde{C}_{u,j}$. Then we can write

$$-i \log(\tilde{C}_{u,j}) + 2\pi K = V_e h_j, \tag{10}$$

where K is an unknown $(n+2) \times n$ integer matrix that depends on the chosen determination of log. Now we search through the integers in order to find an integer vector K and a solution h_j for (10) that minimize the error

$$|V_e h_j - (i \log \tilde{C}_{u,j} + 2\pi K)|.$$

(We already proved that such an h_j is unique except for integer multiples of 2π in the first row.) After finding h_j for each j, let H be the matrix $\begin{pmatrix} h_1 & \cdots & h_n \end{pmatrix}$. Apply σ to the components of the first row of the matrix H to obtain C.

We recognize that an exhaustive search through the integers is a very inefficient way to solve for the matrix H. Intuitively, we are looking for a point in a hyperplane with small integer coordinates, or more precisely a point with small integer coordinates closest to the hyperplane spanned by the columns of an $(n+1) \times n$ matrix. A more efficient way to do this might be using ellipsoid methods or interior point methods ([11]) and linear programming for finding lattice points. These ideas will be developed in the future.

Finding A

Once we have C and H we can immediately obtain A. Drop the first row of H and transpose the result,

$$\begin{pmatrix} A_{11}\sigma(A_1x_0) & A_{12}\sigma(A_2x_0) \\ A_{21}\sigma(A_1x_0) & A_{22}\sigma(A_2x_0) \end{pmatrix}. \tag{11}$$

Next use the first n outputs that were obtained earlier by applying the integer inputs,

$$a = (C\vec{\sigma}(Ax_0),\ C\vec{\sigma}(Ax_0 + B)).$$

This row vector can be written as the product

$$a = (c_1\sigma(A_1x_0),\ c_2\sigma(A_2x_0)) \begin{pmatrix} \sigma(0) & \sigma(b_1) \\ \sigma(0) & \sigma(b_2) \end{pmatrix}.$$

Solve the linear equation to obtain the row vector $(c_1\sigma(A_1x_0),\ c_2\sigma(A_2x_0))$. By now we know the components of C so we may divide these out of each component leaving the vector

$$A_x^\sigma = (\sigma(A_1x_0), \sigma(A_2x_0)).$$

Finally, divide each row of (11) term by term by the components of the vector A_x^σ to obtain A.

This completes the proof of the theorem. *The input/output behavior can now be simulated even without precise knowledge of* x_0. The last two pieces of information that are needed to have full knowledge of the input/output behavior are Cx_0, which is simply the time zero output, and

$$\vec{\sigma}(Ax_0) = A_x^{\sigma,T},$$

which provides the state at time one for any input. Both are already known.

5 Procedure for Noisy Output

In this section we discuss various purely heuristic considerations. The following are intuitive ideas for extending the procedure in the previous section to accomodate more practical situations. One obvious next step involves modifying the basic algorithm to make use of longer output sequences in order to find more uniform approximations. The basic algorithm uses information obtained only in the first two time steps. This may be sufficient for identification of systems that are of the desired form, but it is not enough for approximation of other systems. Using only the information obtained in the first two time steps will produce a desired network approximation for a system that will closely approximate the input/output behavior of the original system for the first two time steps only. We cannot control the accuracy of the approximation beyond that. Another major problem with the

basic approach is the lack of robustness with respect to noise in the output. We have improved this by simultaneously conducting the procedure described above on sequential time intervals of length two and "averaging" the information. In Section 7, we define the noise used and give results of testing this variation of the algorithm on exp-systems with noisy output.

Averaging to find B

The basic algorithm applies $2n$ different length one integer inputs to obtain a scalar sequence that can be seen as the Markov sequence for the triple $(\tilde{A}, \tilde{B}, \tilde{C})$. (This procedure was then repeated with "irrational" inputs so the proper logs could be recovered.)

Now instead of applying one set of length one inputs, apply p sets of inputs increasing in length from one to p. That is, apply the sets of inputs

$$\begin{aligned} length\ 1: &\quad 0, 1, 2, \ldots, 2n-1 \\ length\ 2: &\quad 00, 01, 02, \ldots, 0(2n-1) \\ &\vdots \\ length\ p: &\quad 0\cdots00,\ 0\cdots01,\ 0\cdots02,\ \ldots,\ 0\cdots0(2n-1). \end{aligned}$$

Let y_u denote the output that is measured after applying the input u. The number p can be as high as $2n$ provided the corresponding outputs do not get too small or too large. If y_u is very large or very small for some u, then just let $p = \text{length}(u) - 1$. If we arrange the outputs in an array as follows

$$\begin{array}{lllll} y_0 & y_1 & y_2 & \cdots & y_{2n-1} \\ y_{00} & y_{01} & y_{02} & \cdots & y_{0(2n-1)} \\ \vdots & & & & \\ y_{0\cdots0} & & & \cdots & y_{0\cdots2n-1}, \end{array} \tag{12}$$

we can view the columns of (12) as $2n$ terms of a $p \times 1$ Markov sequence. This sequence of vectors can be realized by the triple

$$\tilde{A} = \begin{pmatrix} \sigma(b_1) & & \\ & \ddots & \\ & & \sigma(b_n) \end{pmatrix}, \qquad \tilde{B} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \tag{13}$$

$$\tilde{C} = \begin{pmatrix} c_1\sigma(A_1x_0) & \cdots & c_n\sigma(A_nx_0) \\ c_1\sigma(A_1\vec{\sigma}(Ax_0)) & \cdots & c_n\sigma(A_n\vec{\sigma}(Ax_0)) \\ \vdots & & \\ c_1\sigma(A_1\vec{\sigma}(A\vec{\sigma}(\cdots(Ax_0)\cdots))) & \cdots & c_n\sigma(A_n\vec{\sigma}(A\vec{\sigma}(\cdots(Ax_0)\cdots))) \end{pmatrix}.$$

The rest of the procedure for recovering $b_1, \ldots b_n$ is almost the same as in the basic algorithm. Here we denote

$$h_j = \begin{pmatrix} y_j \\ y_{0j} \\ \vdots \\ y_{0\cdots 0j} \end{pmatrix},$$

and form the two $pn \times n$ Hankel matrices as in (4). Whereas we could have used either the controllability or the observability form in the first case, here we have to use the controllability form since H_1 and H_2 are no longer square. We solve

$$H_1 A_{contr} = H_2,$$

to obtain the controllability form of $\tilde{A}$. We are interested in the eigenvalues of $\tilde{A}$, which are the same as the eigenvalues of A_{contr}. As before, we "invert" σ to obtain an estimate for B, repeat the whole procedure with inputs that are "less" rational and resolve the ambiguity in B that was created by taking the complex exponential.

Averaging to find C

In the basic strategy we formed the $(n+2) \times n$ dimensional matrix $Y = [y_{i-1,j-1}]$, and performed a few factorizations (and special steps to get the right log) to isolate the matrix in 9 and recover C by applying σ to the components of the first row. We used information obtained only during the first two time steps.

A similar method can be used with any matrix Y, where the elements of Y are outputs corresponding to inputs of the form

$$u = \underbrace{0 \cdots 0}_{any\ \#\ of\ zeros} \quad (i-1)(j-1).$$

We then obtain a matrix with

$$\begin{pmatrix} v_1 & \cdots & v_n \end{pmatrix}$$

as the first row, where $c_j = \sigma(v_j), j = 1, \ldots, n$. In this version of the algorithm we obtain numerous estimates for the vector C by repeating the procedure with different Y's and averaging these estimates. The number of estimates that are included in the average is at most $2n$. We use fewer if the output becomes either too large or too small.

Specifically, we get the first estimate for C using the basic algorithm. Next we apply inputs $u_{ij} = 0(i-1)(j-1)$ to obtain the matrix

$$Y = \begin{pmatrix} y_{0,0,0} & \cdots & y_{0,0,n-1} \\ y_{0,1,0} & \cdots & y_{0,1,n-1} \\ \vdots & & \vdots \\ y_{0,n+1,0} & \cdots & y_{0,n+1,n-1} \end{pmatrix}.$$

Just as before, we can write Y as the product (assuming $n = 2$ for notational clarity)

$$Y = \tilde{C}_u \begin{pmatrix} \sigma(0) & \sigma(b_1) \\ \sigma(0) & \sigma(b_2) \end{pmatrix}$$

where

$$\tilde{C}_u = \begin{pmatrix} c_1\sigma(A_1\vec{\sigma}(A\vec{\sigma}(Ax_0))) & c_2\sigma(A_2\sigma(A\vec{\sigma}(Ax_0))) \\ c_1\sigma(A_1\vec{\sigma}(A\vec{\sigma}(Ax_0)+B)) & c_2\sigma(A_2\vec{\sigma}(A\vec{\sigma}(Ax_0)+B)) \\ c_1\sigma(A_1\vec{\sigma}(A\vec{\sigma}(Ax_0)+2B)) & c_2\sigma(A_2\vec{\sigma}(A\vec{\sigma}(Ax_0)+2B)) \\ c_1\sigma(A_1\vec{\sigma}(A\vec{\sigma}(Ax_0)+3B)) & c_2\sigma(A_2\vec{\sigma}(A\vec{\sigma}(Ax_0)+3B)) \end{pmatrix}.$$

Solve the linear equation for $\tilde{C}_u$, and note that for a chosen determination of the (componentwise) log,

$$-i\log(\tilde{C}_u) + 2\pi K = \begin{pmatrix} 1 & \sigma(0) & \sigma(0) \\ 1 & \sigma(b_1) & \sigma(b_2) \\ 1 & \sigma(b_1)^2 & \sigma(b_2)^2 \\ 1 & \sigma(b_1)^3 & \sigma(b_2)^3 \end{pmatrix} \begin{pmatrix} v_1 & v_2 \\ A_{11}\sigma(A_1\vec{\sigma}(Ax_0)) & A_{21}\sigma(A_1\vec{\sigma}(Ax_0)) \\ A_{12}\sigma(A_2\vec{\sigma}(Ax_0)) & A_{22}\sigma(A_2\vec{\sigma}(Ax_0)) \end{pmatrix}.$$

As in the basic approach, we must simultaneously solve for the rightmost matrix above and the integer matrix K. Then apply σ to the entries of the first row of the resulting matrix to obtain the second estimate for C.

Repeat the procedure starting with a new Y obtained by applying inputs $u_{ij} = 00(i-1)(j-1)$. At the end we will have

$$\begin{pmatrix} v_1 & v_2 \\ A_{11}\sigma(A_1\vec{\sigma}(A\vec{\sigma}(Ax_0))) & A_{21}\sigma(A_1\vec{\sigma}(A\vec{\sigma}(Ax_0))) \\ A_{12}\sigma(A_2\vec{\sigma}(A\vec{\sigma}(Ax_0))) & A_{22}\sigma(A_2\vec{\sigma}(A\vec{\sigma}(Ax_0))) \end{pmatrix} \tag{14}$$

and a third estimate for C can be obtained from the first row of this matrix.

Repeating this with an increasing number of zeros in the beginning of the input sequences provides many estimates for the vector C. The last step is to average all of the estimates.

Averaging to find A

Here we use $\tilde{C}$ from (13) and all of the matrices such as (14) used in finding C. (Now we can discard the first row of each of these matrices.)

The first task is to find $\tilde{C}$. By now we know $\tilde{A}$ and $\tilde{B}$. We also know the controllability form for the triple: $(A_{contr}, B_{contr}, C_{contr})$, since all we really need to know for these matrices is the characteristic polynomial for $\tilde{A}$ and the first n terms of the vector Markov sequence which are exactly the columns in (12). The two triples are related through an invertible

matrix T as follows

$$\begin{aligned} T^{-1}A_{contr}T &= \tilde{A} \\ T^{-1}B_{contr} &= \tilde{B} \\ C_{contr}T &= \tilde{C}. \end{aligned}$$

We retrieve T by solving

$$\begin{pmatrix} B_{contr} & A_{contr}B_{contr} & \cdots & A_{contr}^{n-1}B_{contr} \end{pmatrix} = T \begin{pmatrix} \tilde{B} & \tilde{A}\tilde{B} & \cdots & \tilde{A}^{n-1}\tilde{B} \end{pmatrix}.$$

(The reachability matrices are invertible. Remember, these are not the A, B, C matrices of the underlying exp-system. The triple $(\tilde{A}, \tilde{B}, \tilde{C})$ is a minimal realization of the sequence found in (12).)

Multiply C_{contr} by T to get $\tilde{C}$. Next average the rows of $\tilde{C}$. The resulting row vector is $\tilde{C}_{ave}$, where

$$\tilde{C}_{ave}(j) = c_j \, ave \left\{ \sigma(A_j x_0), \sigma(A_j \vec{\sigma}(Ax_0)), \ldots, \sigma(A_j \vec{\sigma}(A\vec{\sigma}(\cdots \vec{\sigma}(Ax_0)\cdots))) \right\}.$$

Since we already have an estimate for C, we can divide out c_j from each component and we call the outcome $A^{\sigma}_{x,ave}$.

The second matrix we need in this method for finding A comes from averaging the matrices obtained in the procedure for finding C. We drop the first row of each matrix of the form

$$\begin{pmatrix} v_1 & v_2 \\ A_{11}\sigma(A_1\vec{\sigma}(\cdots\vec{\sigma}(Ax_0))) & A_{21}\sigma(A_1\vec{\sigma}(\cdots\vec{\sigma}(Ax_0))) \\ A_{12}\sigma(A_2\vec{\sigma}(\cdots\vec{\sigma}(Ax_0))) & A_{22}\sigma(A_2\vec{\sigma}(\cdots\vec{\sigma}(Ax_0))) \end{pmatrix}.$$

Next let H_{ave} be the matrix formed by averaging (in a term by term way) all of these matrices.

The last step is identical to the basic method. Transpose H_{ave} to obtain

$$\begin{pmatrix} A_{11} ave\{\sigma(A_1x_0), \ldots\} & A_{21} ave\{\sigma(A_2x_0), \ldots\} \\ A_{12} ave\{\sigma(A_1x_0), \ldots\} & A_{22} ave\{\sigma(A_2x_0), \ldots\} \end{pmatrix}.$$

Then divide each row, element by element, by the row vector $A^{\sigma}_{x,ave}$.

One technicality is that the number of rows used in the average $A^{\sigma}_{x,ave}$, and the number of matrices used in the average H_{ave} must be equal. So if one of these is less than $2n$ because of potential numerical problems (numbers becoming either too large or too small), then only the smaller number is used for both computations.

6 Topics for Further Study

The algorithms described above only scratch the surface of the body of work that could and should be done in this area. Section 7 describes some of the shortcomings in the current numerical adaptation of the algorithm. In addition to the points mentioned there, one would also like to be able to identify the parameters without being restricted to testing with the very special (integer and $\lambda\times$integer) inputs used above. This should be possible. In the first step, for example, finding B, applying random inputs corresponds to measuring the output of a continuous-time system at irregularly spaced intervals. This relates to identification of continuous-time linear systems from irregularly spaced samples.

The program should be tested on systems that are not exp-systems, and developed further, if necessary, to handle approximating a broad range of systems. Comparative studies should be made with other identification and approximation techniques.

There are variations on the basic algorithm that should be explored further. One such variation considers the fact that engineering processes are often *hybrid* models with local accurate linear control together with nonlinear components, so a better model for approximation might be a parallel connection of a linear system and an exp-system

$$\begin{aligned} x(t+1) &= A_L x(t) + B_L u(t) \\ z(t+1) &= \vec{\sigma}(A_E z(t) + B_E u(t)) \\ y(t) &= C_L x(t) + C_E z(t). \end{aligned} \tag{15}$$

This model is still a recurrent neural network with

$$A = \begin{pmatrix} A_L & 0 \\ 0 & A_E \end{pmatrix}, \quad B = \begin{pmatrix} B_L \\ B_E \end{pmatrix}, \quad C = \begin{pmatrix} C_L & C_E \end{pmatrix},$$

and $\vec{\sigma}$ acting as the identity on the first few components. We can expand the algorithm to include this model. The parallel system may serve as a better approximator for general systems with linear components. In particular, it trivially includes the class of linear systems. The algorithm for the parallel system closely resembles the basic algorithm. This procedure will be developed in the future and incorporated into the computer implementation.

7 Numerical Techniques

We experimented with the algorithm in Section 5 using MATLAB. The program employs a naive, but effective, approach to carrying out the given techniques for parameter reconstruction. Despite the fact that numerous steps lack creative and even robust numerical solutions, the program and the simulations that follow provide a "proof" that the concepts described above are practical, and indeed have the potential to be executed in an efficient manner. Some of the steps in the computer formulation require further study. The linear

realization that is performed when finding B is done using a simple Hankel matrix technique. This step should be replaced with more sophisticated techniques like Hankel norm approximation and robust identification. The next two steps, finding C and A, use a tedious integer search to find the correct log at some point. This step should also be improved.

The program behaved relatively well in practice. We tested the program on systems that were truly exp-systems, but with small random noise (uniformly distributed on $(-10^{-3}, 10^{-3})$) added to the output. The test systems that we used have dimension between two and four and the entries of the parameters all have magnitude less than 10.

The program was sensitive to the number used for λ - the "irrational" number, and the thresholds used when comparing numbers to integers. This lack of robustness could be eliminated by implementing more refined numerical techniques.

In the simulations below, we use stable systems so that we can make comparisons between the input/output behaviors of the true and estimated systems over a longer time period than the one used for the approximation.

Example 7.1 In this example, we reconstructed the parameters for a three dimensional system. The true system parameters are

$$A = \begin{pmatrix} 0.5000 & -0.6000 & 0.1000 \\ 1.3000 & 0.5000 & 1.0000 \\ 0.4000 & 1.0000 & 0.2000 \end{pmatrix}, \qquad B = \begin{pmatrix} 1.0300 \\ 2.7000 \\ 3.2000 \end{pmatrix},$$

$$C = \begin{pmatrix} 0.1000 & 0.5000 & 0.8000 \end{pmatrix}, \qquad x_0 = \begin{pmatrix} -1.0000 \\ 0.3000 \\ 2.2000 \end{pmatrix}.$$

In the absence of noise in the output, the program estimated the parameters exactly:

$$\hat{A} = \begin{pmatrix} 0.5000 - 0.0000i & -0.6000 - 0.0000i & 0.1000 + 0.0000i \\ 1.3000 - 0.0000i & 0.5000 + 0.0000i & 1.0000 - 0.0000i \\ 0.4000 - 0.0000i & 1.0000 + 0.0000i & 0.2000 - 0.0000i \end{pmatrix},$$

$$\hat{B} = \begin{pmatrix} 1.0300 + 0.0000i \\ 2.7000 + 0.0000i \\ 3.2000 - 0.0000i \end{pmatrix},$$

$$\hat{C} = \begin{pmatrix} 0.1000 - 0.0000i & 0.5000 + 0.0000i & 0.8000 - 0.0000i \end{pmatrix}.$$

In the presence of noise, the following A, B, C were estimated:

$$\hat{A} = \begin{pmatrix} 0.5073 - 0.0019i & -0.5872 - 0.0049i & 0.0864 - 0.0169i \\ 1.2989 + 0.0083i & 0.5039 - 0.0245i & 0.9988 + 0.0136i \\ 0.4065 + 0.0009i & 1.0006 - 0.0035i & 0.2057 + 0.0124i \end{pmatrix},$$

$$\hat{B} = \begin{pmatrix} 1.0290 + 0.0023i \\ 2.6951 - 0.0061i \\ 3.2002 - 0.0031i \end{pmatrix},$$

$$\hat{C} = \begin{pmatrix} 0.1001 - 0.0002i & 0.4964 - 0.0075i & 0.8034 + 0.0058i \end{pmatrix}.$$

The pair of figures (1 and 2) compare the real parts of the outputs for the true system and the one reconstructed when noise was present in the output during the estimation procedure. The input sequences used have length 15; the first is a sequence of 15 ones, the second is a random input sequence.

Example 7.2 This is an example of the parameter reconstruction for a four dimensional system. The true system has parameters

$$A = \begin{pmatrix} 0.5000 & -0.1000 & -0.6000 & -0.3000 \\ 0 & 0.4000 & 0.5000 & 0 \\ 0.7000 & 1.0000 & 0.2000 & 0.3000 \\ -1.0000 & 0.8000 & 0.6000 & 0.1000 \end{pmatrix}, \quad B = \begin{pmatrix} 0.2000 \\ 1.0000 \\ 1.9000 \\ 2.3000 \end{pmatrix},$$

$$C = \begin{pmatrix} 0.4000 & 1.1000 & 6.2000 & 5.0000 \end{pmatrix}, \quad x_0 = \begin{pmatrix} -1.0000 \\ 0.3000 \\ 2.2000 \\ 0.9000 \end{pmatrix}.$$

Without noise in the output, the program correctly estimated the parameters:

$$\hat{A} = \begin{pmatrix} 0.5000 - 0.0000i & -0.1000 + 0.0000i & -0.6000 + 0.0000i & -0.3000 - 0.0000i \\ 0.0000 - 0.0000i & 0.4000 + 0.0000i & 0.5000 + 0.0000i & 0.0000 - 0.0000i \\ 0.7000 - 0.0000i & 1.0000 + 0.0000i & 0.2000 - 0.0000i & 0.3000 + 0.0000i \\ -1.0000 + 0.0000i & 0.8000 - 0.0000i & 0.6000 - 0.0000i & 0.1000 - 0.0000i \end{pmatrix},$$

$$\hat{B} = \begin{pmatrix} 0.2000 - 0.0000i \\ 1.0000 - 0.0000i \\ 1.9000 + 0.0000i \\ 2.3000 - 0.0000i \end{pmatrix},$$

$$\hat{C} = \begin{pmatrix} 0.4000 - 0.0000i & 1.1000 + 0.0000i & 6.2000 - 0.0000i & 5.0000 + 0.0000i \end{pmatrix}.$$

When there was noise in the output, the following A, B, C were estimated:

$$\hat{A} = \begin{pmatrix} 0.4962 + 0.0308i & -0.1166 - 0.0020i & -0.6130 - 0.0135i & -0.3069 + 0.0098i \\ -0.0036 - 0.0023i & 0.4000 + 0.0000i & 0.4954 - 0.0008i & 0.0032 + 0.0003i \\ 0.7112 + 0.0036i & 1.0082 + 0.0092i & 0.1958 + 0.0083i & 0.3036 - 0.0024i \\ -1.0118 - 0.0238i & 0.8008 + 0.0021i & 0.5938 - 0.0008i & 0.1031 + 0.0004i \end{pmatrix},$$

$$\hat{B} = \begin{pmatrix} 0.1999 - 0.0006i \\ 0.9987 + 0.0019i \\ 1.8999 + 0.0003i \\ 2.2999 + 0.0001i \end{pmatrix},$$

$$\hat{C} = \begin{pmatrix} 0.4039 + 0.0069i & 1.1000 + 0.0052i & 6.1728 - 0.0272i & 4.9831 - 0.0053i \end{pmatrix}.$$

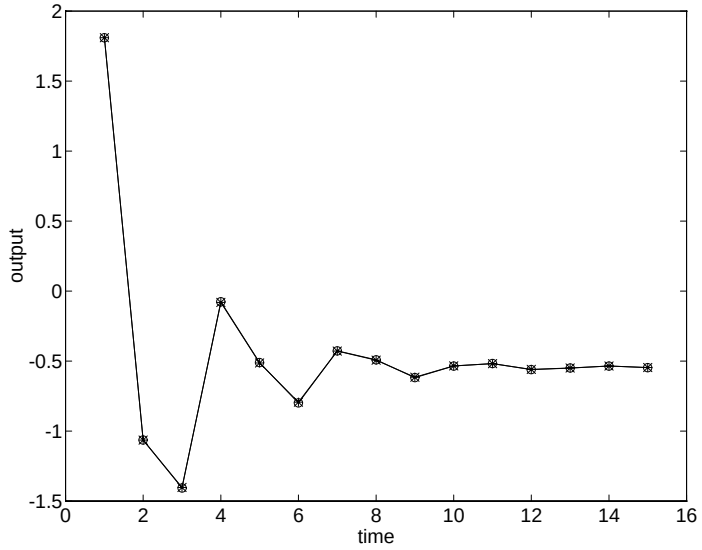

Figure 1: Three dimensional example. True (∘) and approximating (*) outputs for an input sequence of 15 ones. Noisy case.

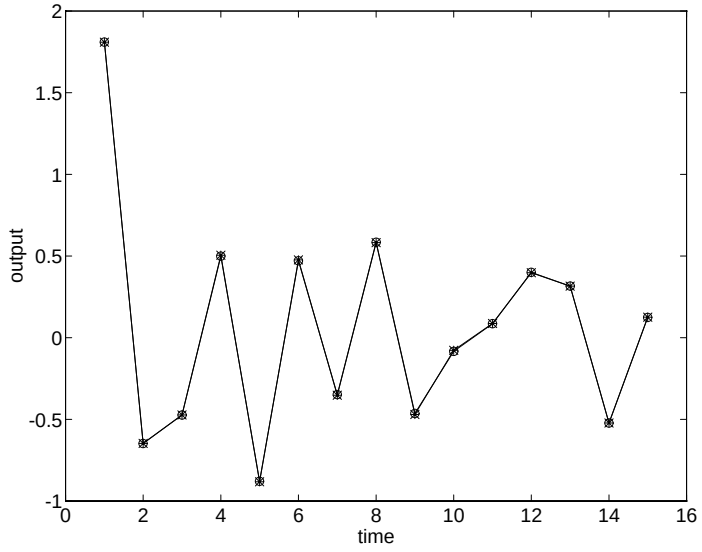

Figure 2: Three dimensional example. True (∘) and approximating (*) outputs for a random input sequence of length 15. Noisy case.

The final two figures compare the real part of the output of the true system versus that of the estimated system. As in the previous examples, the inputs used were a sequence of ones (Figure 3) and a random sequence (Figure 4).

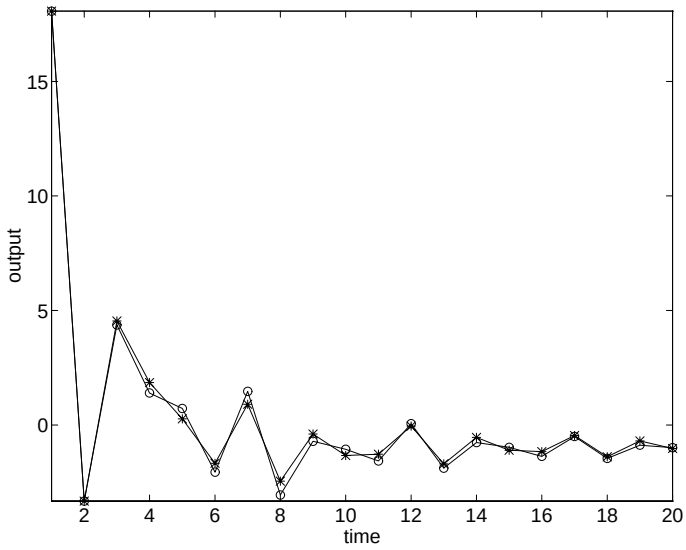

Figure 3: Four dimensional example. True (∘) and approximating (*) outputs for an input sequence of 20 ones. Noisy case.

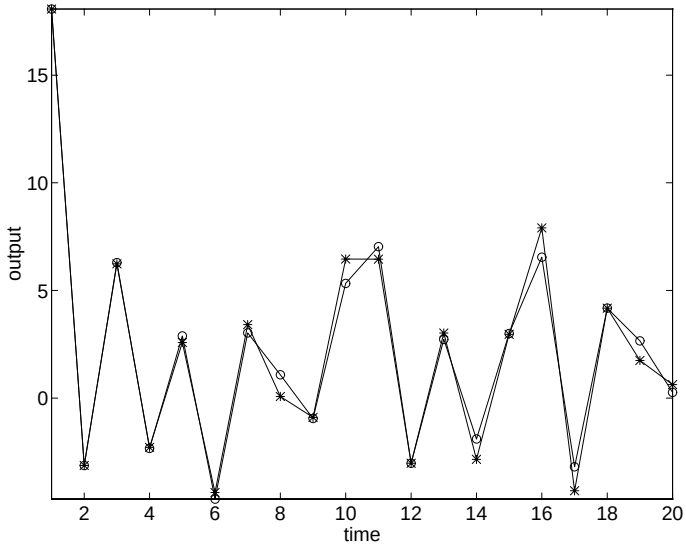

Figure 4: Four dimensional example. True (∘) and approximating (*) outputs for a random input sequence of length 20. Noisy case.

References

[1] Albertini, Francesca, and Eduardo D. Sontag, "For neural networks, function determines form," *Neural Networks* 6 (1993), pp. 975–990. Summarized version in *Proc. IEEE Conf. Decision and Control, Tucson, Dec. 1992*, IEEE Publications, 1992, pp. 26–31.

[2] Albertini, Francesca, and Eduardo D. Sontag, "Identifiability of discrete-time neural networks," *Proc. European Control Conference*, Groningen, June 1993, pp. 460–465.

[3] Albertini, Francesca, and Eduardo D. Sontag, "State observability in recurrent neural networks," *Proc. IEEE Conf. Decision and Control, San Antonio, Dec. 1993*, IEEE Publications, 1993, pp. 3706–3707.

[4] Ben-Or, M., and P. Tiwari, "A deterministic algorithm for sparse multivariate polynomial interpolation," in *Proc. 29th IEEE Symp. Foundations of Comp. Sci.*, 1988, pp. 301–309.

[5] Gallant, A.R., and H. White, "There exists a neural network that does not make avoidable mistakes," *Proc. IEEE International Conference on Neural Networks*, San Diego, 1988, pp. I657–I664.

[6] Koplon, Renée, *Linear Systems with Constrained Outputs and Transitions*, Ph.D. Dissertation, Rutgers University, 1994.

[7] Koplon, Renée, and Eduardo D. Sontag, "Techniques for Parameter Reconstruction in Fourier-Neural Recurrent Networks," *Proc. IEEE Conf. Decision and Control, Orlando, Dec. 1994*, IEEE Publications, 1994, pp. 213–218.

[8] Kuhn, Gary M., and Norman P. Herzberg, "Some Variations on Training of Recurrent Networks," *Neural Networks: Theory and Applications*, (Richard J. Mammone and Yehoshua Zeevi, eds.) pp. 233–244.

[9] Ljung, L., *System Identification: Theory for the User*, Prentice Hall, 1987.

[10] Narendra, K.S. and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Nets*, 1 (1990), pp. 4–27.

[11] Nesterov, Y. and A. Nemirovskii, *Interior-point Polynomial Algorithms in Convex Programming*, SIAM, Philadelphia, 1994.

[12] Polycarpou, M.M., and P.A. Ioannou, "Neural networks and on-line approximators for adaptive control," in *Proc. Seventh Yale Workshop on Adaptive and Learning Systems*, pp. 93-798, Yale University, 1992.

[13] Siegelmann, Hava T., and Eduardo D. Sontag, "Some results on computing with 'neural nets'," *Proc. IEEE Conf. Decision and Control, Tucson, Dec. 1992*, IEEE Publications, 1992, pp. 1476–1481.

[14] Sontag, Eduardo D., *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, Springer, New York, 1990.

[15] Sontag, Eduardo D., "Neural nets as system models and controllers," in *Proc. Seventh Yale Workshop on Adaptive and Learning Systems*, pp. 73–79, Yale University, 1992.

[16] Sontag, Eduardo D., "Neural networks for control," in *Essays on Control: Perspectives in the Theory and its Applications* (H.L. Trentelman and J.C. Willems, eds.), Birkhauser, Boston, 1993, pp. 339–380.

[17] Żbikowski, R., K. J. Hunt, A. Dzieliński, R. Murray-Smith, and P.J. Gawthrop, "A review of advances in neural adaptive control systems," *Technical Report of the ESPRIT NACT Project TP-1*, Glasgow University and Daimler-Benz Research, 1994.

Biographies of the authors

Renée Koplon recieved the A.B. degree in Mathematics from Barnard College in 1987 and the Ph.D., also in Mathematics, from Rutgers, the State University of New Jersey, in 1994.

In 1994, Dr. Koplon joined the Department of Mathematics and Statistics at Wright State University, Dayton, Ohio, where she is currently an Assistant Professor. Her research interests are in observability and identification of nonlinear control systems and neural networks. Dr. Koplon's professional memberships include SIAM, IEEE and AWM.

Eduardo Sontag received the Licenciado degree (Mathematics) from the University of Buenos Aires in 1972, and the Ph.D. (Mathematics) under Rudolf E. Kalman at the Center for Mathematical Systems Theory, University of Florida (1976).

Since 1977, Dr. Sontag has been with the Department of Mathematics at Rutgers, The State University of New Jersey, where he is currently Professor II of Mathematics as well as a Member of the Graduate Faculties of the Department of Computer Science and of the Department of Electrical and Computer Engineering. He is also the director of SYCON, the Rutgers Center for Systems and Control. His major current research interests lie in several areas of control theory and foundations of learning and neural networks.

Dr. Sontag has authored over two hundred journal and conference papers and book chapters in the above areas, as well as the books Topics in Artificial Intelligence (in Spanish, Buenos Aires: Prolam, 1972), Polynomial Response Maps (Berlin: Springer, 1979), and Mathematical Control Theory: Deterministic Finite Dimensional Systems (Texts in Applied Mathematics, Volume 6, New York: Springer, 1990). His industrial experience includes consulting for Bell Laboratories and Siemens. He is or has been an Associate Editor for various journals, including: Systems and Control Letters, IEEE Transactions in Automatic Control, Control-Theory and Advanced Technology, SMAI-COCOV, Journal of Computer and Systems Sciences, Dynamics & Control, Neurocomputing, and Neural Networks. In addition, he is a co-founder and co-Managing Editor of the Springer journal MCSS (Mathematics of Control, Signals, and Systems).

Dr. Sontag is an IEEE Fellow, and has been Program Director and Vice-Chair of the Activity Group in Control and Systems Theory of SIAM. He has been a member of several committees at SIAM and the AMS, and is a former Chair of the Committee on Human Rights of Mathematicians of the latter.