# Automating the Simulation of Complex Discrete–time Control Systems: A Mathematical Framework, Algorithms and a Software Package*

R. Douglas Ellis, R. Ravikanth and P. R. Kumar†
Dept. of Electrical and Computer Engineering, and
Coordinated Science Laboratory
University of Illinois
1308 West Main Street
Urbana, IL 61801
USA

**Abstract**

With the rapid proliferation of complex control systems, such as adaptive control systems, there has arisen a need to automate as much as possible of the time consuming preparatory work needed to simulate such systems. This burden currently falls on the user. The smaller the user time required, and the more effortless the process, the greater, we believe, will be the eventual widespread acceptability of these control systems.

This paper is aimed at bridging this gap between the theoretical specification of a complex discrete–time control system as described in a typical research paper, and the obtaining of a first simulation. It provides a graph theoretical formulation of the issues involved, discrete algorithms to execute these tasks, and finally describes ISIM, an interpretive simulator which incorporates these features. These algorithms automate the simulation process, as seen by the user. The net result is a package which essentially requires only that the user type in a control system more or less verbatim from a technical paper, in order to obtain a simulation output.

*Please address all correspondence to the third author.

†The research reported here has been supported in part by the U.S. Army Research Office under Contract No. DAAL–03–91–G–0182, the National Science Foundation under Grant No. NSF–ECS–92–16487, and the Joint Services Electronics Program under Contract No. N00014–90–J–1270.

# 1 Introduction

In recent years we have witnessed the advent and proliferation of rather complex identification and adaptive control laws. When confronted with such a control law or identification procedure, the typical user would like to quickly simulate it to determine its behavior. The theoretical revolution has therefore given rise to the need for a simulation package which eliminates as many as possible of the barriers and time consuming work which lie between the theoretical specification of a control law or dynamic system, as described, for example, in a typical research paper, and its translation into a first simulation run. The smaller the time that the user is required to invest in obtaining a first simulation run, and the more effortless this process is, we believe the greater will be the eventual widespread acceptability of these theoretical developments and complex control laws. Moreover, the greater the amount of bookkeeping and preparatory work needed on the part of the user, the greater is the likelihood of errors, and the greater is the subsequent debugging effort required.

This paper is addressed precisely at bridging this gap between theoretical specification and simulation output. We describe the barriers and time consuming preparatory work which current simulation packages require of the user before she can write a program to simulate a complex feedback control system. Then we provide a discrete–mathematical and graph theoretical formulation of the issues involved, determine algorithms to automate this process, and prove their validity.

Finally, we briefly describe some software engineering aspects of a simulation package, called ISIM, which has been built around these algorithms. It is an interpretive simulator with an unique interface requiring little user training, and little preparatory work required of the user to simulate a particular system. The net result is an automation of the process of simulation, to the point where essentially all that is required is that the user type in, more or less verbatim, the equations as written in a typical research paper. We hope that this package will lead to greater attention being paid to the issues which lie between theoretical

specification and simulation output.

To quickly motivate the reader, let us consider the well known Extended–Least–Squares–Based Adaptive Tracker, as described in the recent paper of Guo and Chen [1]. It is specified by the following equations.

$$\begin{aligned} y(t) &= -a_1 y(t-1) - \cdots - a_p y(t-p) + b_1 u(t-1) + \cdots + b_q u(t-q) \\ &\quad + w(t) + c_1 w(t-1) + \cdots + c_r w(t-r) \end{aligned} \tag{1}$$

$$u(t) = b_1^{-1}(y^*(t+1) - \theta^T(t)\phi(t)) \tag{2}$$

$$\theta(t+1) = \theta(t) + a(t)P(t)\phi(t)(y(t+1) - b_1 u(t) - \theta^T(t)\phi(t)) \tag{3}$$

$$\begin{aligned} P(t+1) &= P(t) - a(t)P(t)\phi(t)\phi^T(t)P(t) \\ a(t) &= (1 + \phi^T(t)P(t)\phi(t))^{-1} \end{aligned}$$

$$\phi(t) = (y(t), \ldots, y(t-p+1), u(t-1), \ldots, u(t-q+1), \widehat{w}(t), \ldots, \widehat{w}(t-r+1))^T \tag{4}$$

$$\widehat{w}(t) = y(t) - b_1 u(t-1) - \theta^T(t)\phi(t-1) \tag{5}$$

$$w(t) = \text{ white noise}$$

$$y^*(t) = \sin(t). \tag{6}$$

An inspection of this set of equations shows that considerable *reordering* and *time–shifting* is necessary to recursively update them. For example, $y(t+1)$ should be computed before $\theta(t+1)$, as seen from (3). This suggest two possibilities. One option is to time–shift (3) and obtain an equation for $\theta(t)$, or one could time–shift (1) to obtain an equation $y(t+1)$. Investigating the first option, we could time–shift (3) and re–write it as,

$$\theta(t) = \theta(t-1) + a(t-1)P(t-1)\phi(t-1)(y(t) - b_1 u(t-1) - \theta^T(t-1)\phi(t-1)). \tag{1'}$$

However, the equation (2) for $u(t)$ can then be computed only after $\theta(t)$ has been determined, and so the equation (2) will need to follow (1′). However one also has to time–shift (6) to,

$$y^*(t+1) = \sin(t+1)$$

and compute it before (2). In addition, (4) will have to precede (2), and (5) will have to precede (4), etc. Thus, it is apparent that there is considerable user preparation needed to even write the equations down in a manner in which they can then be updated.

Next, the user has to confront the issue of specifying adequate initial conditions, which will enable all the recursions to start. This requires additional user time. Finally, since one is employing vectors and matrices such as $\theta(t)$ and $P(t)$, one may even have to re–write the equations for each *component* of each vector or matrix.

The rest of this paper is organized as follows. In Section 2, we examine when a system of equations purportedly describing a dynamic system is well–posed so that one is guaranteed that, after appropriate re–ordering and time–shifting, it can be recursively computed in forward time. Then in Section 3, we provide a graph theoretical characterization, and specify algorithms. In Section 4, we examine the role of the initial conditions, and determine when they are adequate. Finally, in Section 5, we describe some of the software engineering features of ISIM, a simulation package which incorporates these algorithms.

# 2 Well–Posedness and Forward Recursive Computation

Consider the following system of equations:

$$\begin{aligned} x_1(t) &= f_1(x_1(t-a_{11}), x_2(t-a_{12}), \ldots, x_N(t-a_{1N})) \\ &\vdots \\ x_i(t) &= f_i(x_1(t-a_{i1}), x_2(t-a_{i2}), \ldots, x_N(t-a_{iN})) \\ &\vdots \\ x_N(t) &= f_N(x_1(t-a_{N1}), x_2(t-a_{N2}), \ldots, x_N(t-a_{NN})). \end{aligned} \tag{7}$$

Let $\{k_1, k_2, \ldots, k_N\}$ be a permutation of $\{1, \ldots, N\}$, and define

$$y_i(t) := x_{k_i}(t).$$

The corresponding system of equations for $\{y_1(t), \ldots, y_N(t)\}$ is a *re–ordered* version of the equations (7) for $\{x_1(t), \ldots, x_N(t)\}$. It is given by

$$\begin{aligned} y_1(t) &= g_1(y_1(t-b_{11}), \ldots\ldots, y_N(t-b_{NN})) \\ &\vdots \\ y_i(t) &= g_i(y_1(t-b_{i1}), \ldots\ldots, y_N(t-b_{iN})) \\ &\vdots \\ y_N(t) &= g_N(y_1(t-b_{N1}), \ldots\ldots, y_N(t-b_{NN})), \end{aligned}$$

where

$$b_{ij} := a_{k_i k_j}$$

and

$$g_i(y_1(t-b_{i1}), \ldots, y_N(t-b_{iN})) := f_{k_i}(x_1(t-a_{k_i 1}), \ldots, x_N(t-a_{k_i N})).$$

Let $\{d_1, \ldots, d_N\}$ be a set of time–shifts, where each $d_i$ is an integer (positive or non–positive). Define

$$z_i(t) := y_i(t-d_i).$$

The corresponding system of equations for $\{z_1(t), \ldots, z_N(t)\}$ is a *time–shifted* version of the equations for $\{y_1(t), \ldots, y_N(t)\}$. It is given by:

$$\begin{aligned} z_1(t) &= g_1(z_1(t-c_{11}), \ldots\ldots, z_N(t-c_{1N})) \\ &\vdots \\ z_i(t) &= g_i(z_1(t-c_{i1}), \ldots\ldots, z_N(t-c_{iN})) \\ &\vdots \\ z_N(t) &= g_N(z_1(t-c_{N1}), \ldots\ldots, z_N(t-c_{NN})) \end{aligned} \tag{8}$$

where

$$c_{ij} := b_{ij} + d_i - d_j.$$

We shall say that the set of equation (8) is a *re–ordered and time–shifted version* of (7) with re–ordering $\{k_1, \ldots, k_N\}$ and time–shifts $\{d_1, \ldots, d_N\}$.

Suppose now that $\{z_1(t-m), \ldots, z_n(t-m)\}$ have been already computed for all $m \geq 1$ (or provided as initial conditions). Let us determine whether $z_1(t), \ldots, z_N(t)$ can be updated *in that order*. Clearly, from (8), in order to update $z_1(t)$, we need $z_1(t-c_{11}), \ldots, z_N(t-c_{1N})$. Thus $z_1(t)$ can be computed if and only if,

$$c_{1i} \geq 1 \quad \text{for} \quad 1 \leq i \leq N.$$

Now let us consider whether $z_2(t)$ can be computed. Clearly, one needs $z_1(t-c_{21}), \ldots, z_N(t-c_{2N})$. Hence $z_2(t)$ can be computed (given that $z_1(t)$ has already been), if and only if

$$c_{21} \geq 0, \quad \text{and} \quad c_{2i} \geq 1 \quad \text{for} \quad 2 \leq i \leq N.$$

Continuing in this way, we see that $z_j(t)$ can be computed if and only if

$$c_{ji} \geq 0, \quad \text{for} \quad 1 \leq i \leq j-1 \quad \text{and} \quad c_{ji} \geq 1 \quad \text{for } j \leq i \leq N.$$

This motivates the following two definitions.

**Definition: Forward Recursive Computability of a System of Equations.** *A system of equations (8) is said to be* forward recursively computable *if*

$$\begin{aligned} c_{ji} &\geq 0 \quad \textit{for} \quad 1 \leq i \leq j-1 \\ &\geq 1 \quad \textit{for} \quad j \leq i \leq N. \end{aligned}$$

**Definition: Well–posedness.** *A system of equations (7) is said to be* well–posed *if there exists a re–ordering $\{k_1, \ldots, k_N\}$ and a set of time shifts $\{d_1, \ldots, d_N\}$ such that the resulting re–ordered and time shifted version (8) is forward recursively computable.*

# 3 Characterization of Well–Posedness and Determination of a Forward Recursively Computable Version

Let us suppose that our starting point for a simulation project is a system of equations (7) purportedly defining a dynamic system. Our first goal is to determine whether the system of equations is indeed well–posed, and, if so, to obtain a re–ordering and time–shifts which yield a recursively computable version.

Given the system of equations (7), or equivalently the integers $\{a_{ij}\}$, we will associate a weighted digraph.[1] First, we associate a node with each variable $x_i$. Then we connect the nodes $x_i$ and $x_j$ by a directed arc of weight $a_{ij}$ from $x_j$ to $x_i$. We shall call this the *delay graph* of the system (7).

**Remark 1.** We should note that, in general, a given system of equations may differ from (7) in two respects. Consider, as an example, the system,

$$\begin{aligned} x_1(t) &= f_1(x_1(t-1), x_4(t+1)) \\ x_2(t) &= f_2(x_1(t), x_3(t-1), x_3(t-4)) \\ x_3(t) &= f_3(x_1(t+1), x_2(t-1), x_4(t)) \\ x_4(t) &= f_4(x_3(t-3)). \end{aligned} \tag{9}$$

First note that not all variables occur in each equation. For example, the variables $x_2$ and $x_3$ are missing in the equation for $x_1(t)$. In this case, there are simply no arcs from $x_2$ or $x_3$ to $x_1$. Second, a given variable may occur more than once in an equation. For example, the

[1] We recall the following definitions from graph theory; see [2]. A *loop* is an arc whose initial and final nodes are the same. A *path* is a sequence of nodes $(x_1, x_2, \ldots, x_k)$ such that each of the pairs $(x_1, x_2), (x_2, x_3), \ldots, (x_{k-1}, x_k)$ is an arc of the graph. The nodes $x_1$ and $x_k$ are referred to as the starting and ending nodes, respectively. A *circuit* is a path with the same starting and ending node, i.e., $x_1 = x_k$. A *simple circuit* is a circuit in which no node, except the starting node, is visited more than once, while the starting node is visited twice.

variable $x_3$ appears twice, as $x_3(t-1)$ and $x_3(t-4)$, in the equation for $x_2(t)$. In this case, we simply draw two arcs from $x_3$ to $x_2$. The resulting weighted digraph is given in Figure 1. For the sake of notational brevity, we shall mainly continue to focus on the set of equations (7). □

Figure 1: The weighted digraph for the system (9).

The first theorem characterizes when a system of equation (7) is well–posed. We should note that such a characterization has already been obtained earlier by Karp, Miller and Winograd [3]; see Theorem 1 there. Our proof, from [6], is constructive, and provides an algorithm for determining a re–ordering $\{k_1, \ldots, k_N\}$ and a set of time shifts $\{d_1, \ldots, d_N\}$ which yield a forward recursively computable version, whenever the system is well–posed. A constructive proof with a different construction can also be found in Liu, Ho and Sheu [5].

**Theorem 1: Well–Posedness.** *The system of equations (7) is well–posed if and only if all the simple circuits in the delay graph, including self-loops, have a strictly positive weight.*

*Here, the weight of a simple circuit is defined as the sum of its arc weights.*

**Proof** The necessity is obvious, since if there is a simple circuit with weight $-w < 0$, and if $x_i$ is a node along the simple circuit, then one needs knowledge of $x_i(t+w)$ to determine $x_i(t)$. Hence one can never compute $x_i(t)$ without having its future values.

The proof of sufficiency is constructive; we will determine a forward recursively computable version with appropriate re–ordering and time shifts. That is, if all simple circuits have strictly positive weight, we will determine a reordering $\{k_1, \ldots, k_N\}$, and time–shifts $\{d_1, \ldots, d_N\}$, so that

$$\begin{aligned} a_{k_j k_i} + d_{k_j} - d_{k_i} &\geq 0 \text{ for } 1 \leq i \leq j-1 \\ &\geq 1 \text{ for } j \leq i \leq N. \end{aligned} \tag{10}$$

Let $d_{ij}$ be the weight of the shortest path from $x_j$ to $x_i$. (If there is no path from $x_j$ to $x_i$, set $d_{ij} = +\infty$). Moreover $d_{ij}$ cannot be $-\infty$ for any $i$, $j$, since that would imply the existence of a simple circuit with strictly negative weight. Consider the associated weighted digraph with nodes $x_1, \ldots, x_N$ and having arcs with weights $d_{ij}$ from node $x_j$ to node $x_i$ whenever $d_{ij} < +\infty$. We shall call this the *shortest path graph.*

The shortest path graph has four properties.

(i) $d_{ij} \leq a_{ij}$

(ii) $d_{ik} \leq d_{ij} + d_{jk}$

(iii) All simple circuits in the shortest path graph also have strictly positive weights.

(iv) $d_{ij} + d_{ji} \geq 1$.

The first two properties are a consequence of the shortness of the paths, while the third is inherited from the original delay graph. The fourth property follows from (iii).

Clearly, in view of (i), to show (10) it suffices to show that,

$$\begin{aligned} d_{k_j k_i} + d_{k_j} - d_{k_i} &\geq 0 \text{ for } 1 \leq i \leq j-1 \\ &\geq 1 \text{ for } j \leq i \leq N, \end{aligned} \tag{11}$$

for some permutation $\{k_1, \ldots, k_N\}$, and integers $\{d_1, \ldots, d_N\}$.

Our proof of (11) uses the following independent lemma.

**Lemma 1.** *Suppose*

*(i)* $d_{ij} + d_{ji} \geq 1$ *for all* $i$, $j$

*(ii)* $d_{ij} + d_{jk} \geq d_{ik}$ *for all* $i$, $j$, $k$,

*where* $\{d_{ij} : 1 \leq i, j \leq N\}$ *are integers (positive or non-positive). Then there exists integers* $d_1, \ldots, d_N$ *such that*

$$d_{ij} + d_i - d_j \geq 0 \text{ for } 1 \leq i, j \leq N.$$

**Proof.** The proof is constructive. Let $D^{(0)}$ be the matrix $[d_{ij}]$. For $m = 1, \ldots, N$, let

$d_m$ := –(smallest element in the $m$-th row of $D^{(m-1)}$).

$D^{(m)}$ := matrix obtained by adding $d_m$ to the first row of $D^{(m-1)}$, and subtracting it from the $m$-th column (thus leaving the $(m,m)$-th element unchanged).

Clearly $D^{(N)} = [d_{ij} + d_i - d_j]$.

We claim that all elements of $D^{(n)}$ are non-negative, which would prove the lemma. The proof is by induction on the following hypothesis.

**Hypothesis $\mathbf{H^{(m)}}$.** *Let* $d_{ij}^{(m)}$ *denote the* $ij$*-th element of* $D^{(m)}$. *Then,*

a) *All elements in rows* $1, 2, \ldots m$ *of* $D^{(m)}$ *are nonnegative.*

*b)* $d_{ij}^{(m)} + d_{ji}^{(m)} \geq 1$

*c)* $d_{ij}^{(m)} + d_{jk}^{(m)} \geq d_{ik}^{(m)}$.

For $H^{(0)}$, the claim (a) is trivially true, while (b) and (c) are simply properties (iv) and (ii).

Suppose $H^{(m-1)}$ is true. The matrices $D^{(m-1)}$ and $D^{(m)}$ differ only in their $m$-th row and $m$-th column. Hence all elements of $D^{(m)}$ in the first $(m-1)$ rows, except possibly for those in the $m$-th column are nonnegative. Thus, to show (a) we need only show that $d_{im}^{(m)} \geq 0$ for $i = 1, 2, \ldots, m-1$, $d_{mj}^{(m)} \geq 0$ for $j \neq m$, and $d_{mm}^{(m)} \geq 0$.

Let $d_{mk}^{(m-1)}$ be the smallest element in the $m$-th row of $D^{(m-1)}$. Then, by definition, $d_m = -d_{mk}^{(m-1)}$. For $i = 1, 2, \ldots, m-1$,

$$\begin{aligned} d_{im}^{(m)} &:= d_{im}^{(m-1)} - d_m \\ &= d_{im}^{(m-1)} + d_{mk}^{(m-1)} \\ &\geq d_{ik}^{(m-1)} \qquad \text{(by (c) of } H^{(m-1)}\text{)} \\ &\geq 0 \qquad \text{(by (a) of } H^{(m-1)}\text{)}. \end{aligned}$$

Also, for $j \neq m$,

$$\begin{aligned} d_{mj}^{(m)} &:= d_{mj}^{(m-1)} + d_m \\ &= d_{mj}^{(m-1)} - d_{mk}^{(m-1)} \\ &= d_{mj}^{(m-1)} - \min_{\ell} d_{m\ell}^{(m-1)} \\ &\geq 0. \end{aligned}$$

Finally $d_{mm}^{(m)} := d_{mm}^{(m-1)} \geq 1$ by (b) of $H^{(m-1)}$ using the integer nature of all the matrix elements. Thus we have proved (a) of $H^{(m)}$.

For (b) of $H^{(m)}$, we simply note that $d_{ij}^{(m-1)} + d_{ji}^{(m-1)}$ is invariant when we add and subtract the same quantity to and from the $m$-th row and $m$-th column, respectively. Similarly,

$d_{ij}^{(m-1)} + d_{jk}^{(m-1)} - d_{ik}^{(m-1)}$ is also an invariant, proving (c) of $H^{(m)}$. □

Now we complete the proof of Theorem 1. By property (iii), all circuits in the graph of $D := [d_{ij}]$ have strictly positive weights. Since $D^{(N)} := [d_{ij} + d_i - d_j]$, all circuits in the graph of $D^{(N)}$ have the same weights as in the graph of $D$, and so $D^{(N)}$ inherits strictly positive weights for all its circuits. Moreover all elements of $D^{(N)}$ are nonnegative, from Lemma 1.

Now we claim that there is at least one row of $D^{(N)}$ which has only strictly positive elements. If not, every row of $D^{(N)}$ has at least one zero element, and so each node has an incoming arc with weight zero. This however implies that there is a simple circuit with weight zero, a contradiction.

Let $k_1$ be the number of that row of $D^{(N)}$ which has only strictly positive elements, i.e.,

$$d_{k_1 j}^{(N)} := d_{k_1 j} + d_{k_1} - d_j \geq 1 \text{ for all } j.$$

Now eliminate the $k_1$-th row and $k_1$-th column from $D^{(N)}$, and consider the resulting $(N-1) \times (N-1)$ matrix. Again, not all rows of this reduced matrix can contain a zero. Hence there exists $k_2$ such that

$$d_{k_2 j}^{(N)} := d_{k_2 j} + d_{k_2} - d_j \geq 1 \text{ for all } j \neq k_1.$$

Repeating this procedure, we determine $k_1, k_2, \ldots, k_N$ satisfying (11). □

Above, we have not only characterized well–posedness, but we have also obtained an *algorithm* for determining a forward recursively computable version, whenever a system of equations is well–posed.

In order to check well–posedness, one thus needs to determine whether all simple circuits in the delay (or shortest path) graph have strictly positive weights. This can be done by the following algorithm, which at each step reduces the size of the graph by one node, while preserving the lengths of short cycles.

Step 1: Let $G$ be the delay graph (or shortest path graph).

Step 2: If any self-loop (i.e., an arc of the form $(i,i)$) has non-positive weight, then stop since the system is not well–posed.

Step 3: If the graph consists of only one node, then stop and conclude that the system is well–posed.

Step 4: Eliminate all self-loops.

Step 5: If there are multiple arcs between any two nodes, then eliminate all except the arc with the smallest weight.

Step 6: Take any node from the graph, say $i$, and eliminate it. Replace every pair of arcs $(j,i)$ and $(i,k)$ with weights $w_{ji}$ and $w_{ik}$, by a single arc $(j,k)$ with weight $w_{ji} + w_{ik}$.

Step 7: Return to Step 2.

Figure 2 illustrates this algorithm for the system specified by (9).

# 4 Determining Adequacy of Initial Conditions and Proper Initialization

In this section, we characterize when a given set of initial conditions specified by the user is adequate.

We shall presume that the initial conditions are specified in the following form.

i) There is a set $S \subsetneq \{1, \ldots, N\}$.

ii) For $i \notin S$, no initial values are provided for the variable $x_i$.

iii) For $i \in S$, the values of $x_i(t)$ are specified for $\underline{T}_i \leq t \leq \overline{T}_i$ for some integers $\underline{T}_i \leq \overline{T}_i$.

Figure 2: Example of graph reduction to determine if all circuit weights are strictly positive. (a) The original graph has a strictly positive loop which may be eliminated. (b) The parallel arcs are replaced by one with least weight. (c) Node $x_1$ is eliminated, and all combinations of the incoming and outgoing arcs are replaced by two arcs on the reduced graph. (d) No loops were created from this reduction, but the parallel arc with the longer weight is eliminated. (e) Node $x_2$ is eliminated. (f) The incoming/outgoing arcs result in one loop and parallel arcs in the reduced graph, and the self loop can be eliminated. (g) The larger of the parallel arcs is eliminated. (h) The graph is reduced for the last time by eliminating node $x_3$. (i) This last loop has a strictly positive weight; thus the system is well–posed.

Let us suppose that our goals are the following.

iv) For $i \in S$, we wish to determine $x_i(t)$ for all $t \geq \overline{T}_i + 1$.

v) For $i \notin S$, we wish to determine the smallest $\Gamma_i$, such that all values of $x_i(t)$ for $t \geq \Gamma_i$ can be determined.

vi) Moreover all these should be computed without recourse to $x_i(t)$, $t < \underline{T}_i$ for $i \in S$.

If these goals are indeed feasible, we wish to determine $\Gamma_i$ for all $i \notin S$, and then to compute the solution. We shall then say that the set of initial conditions is *adequate*. If the set is not adequate, we wish to know that too.

In this formulation, the only mild biases we have imposed on the problem statement are that (i) when initial values are provided for a variable, they are provided over a *contiguous* range of times, and (ii) one wants to determine all future values of a variable after the initial conditions, without recourse to values prior to the initial conditions. The following example clarifies these issues.

## Example 1

Consider the system:

$$\begin{aligned} x_1(t) &= x_2(t-1) \\ x_2(t) &= x_1(t-1) \end{aligned}$$

with the given initial conditions,

$$\begin{aligned} x_1(0) &= 1 \\ x_2(4) &= 2. \end{aligned}$$

Then it is clear that one can compute $x_1(t)$ as well as $x_2(t)$ for all $t \geq 4$, but *not* all values of $x_1$ between $t = 0$ and 4. Thus, by our criteria, these initial conditions are *not* adequate.

However, the initial conditions

$$\begin{aligned} x_1(0) &= 1 \\ x_2(0) &= 2 \end{aligned}$$

are indeed adequate. □

As alluded to in Remark 1, let us allow for the possibility that in the equation for $x_i(t)$, the right hand side depends on values $x_j(t-\bar{a}_{ij})$, $x_j(t-\underline{a}_{ij})$ with $\underline{a}_{ij} \leq \bar{a}_{ij}$ and possibly some other values in between, or does not depend on $x_j$ at all.

**Theorem 2.** *Consider a well–posed system of equations. Define* $\Gamma_j := \underline{T}_j$ *for* $j \in S$. *Set* $S^{(0)} := S$ *and recursively define,*

$$\begin{aligned} S^{(m+1)} &:= S^{(m)} \cup \{i \notin S^{(m)} \mid f_i \textit{ depends only on } x_j \textit{ for } j \in S^{(m)}\}. \\ \Gamma_i &:= \max_{\{j | f_i \textit{depends on } x_j\}} (\Gamma_j + \bar{a}_{ij}) \quad \textit{for } i \in S^{(m+1)}. \end{aligned} \tag{12}$$

*Then the set of initial conditions is adequate if and only if,*

*(i)* $S^{(m)} = S^{(m+1)} = \{1, 2, \ldots, N\}$ *for some* $m \leq |S^c| :=$ *cardinality of* $S^c$.

*(ii) The values of* $x_i(\overline{T}_i + 1)$ *for* $i \in S$ *can be determined.*

*(iii) The values of* $x_i(\Gamma_i)$ *for* $i \notin S$ *can be determined.*

**Proof:** First we prove necessity. We assume the non-trivial case $S \subsetneq \{1, 2, \ldots, N\}$. Suppose the set of initial conditions is adequate. Then there has to be some $i \notin S$ for which $x_i$ depends only on $x_j$ for $j \in S$, for otherwise no value of any $x_i$ in $S^c$ for $i \notin S$ can be determined. Hence $S^{(1)} \neq S^{(0)}$. For such an $i \in S^{(1)}$, $x_i(\Gamma_i)$ can be determined for some $\Gamma_i$, which we shall suppose to be the earliest instant possible.

If $f_i$ depends on $x_j$, then the computation of $x_i(\Gamma_i)$ requires knowledge of $x_j(\Gamma_i - \bar{a}_{ij})$. Since only the values of $x_j(t)$ for $t \geq \underline{T}_j$ should be used, we should have $\Gamma_i - \bar{a}_{ij} \geq \underline{T}_j$. Thus

$$\Gamma_i \geq \max_{\{j|f_i \ depends \ on \ x_j\}} (\underline{T}_j + \bar{a}_{ij}).$$

Since $x_i(\Gamma_i)$ is the *earliest* value of $x_i$ which can be computed, $x_i(\Gamma_i - 1)$ cannot be determined, and so we have equality above, i.e.,

$$\Gamma_i = \max_{\{j|f_i \ depends \ on \ x_j\}} (\Gamma_j + \bar{a}_{ij})$$

where we have also used the fact that $\Gamma_j := \underline{T}_j$ for $j \in S$. A repetition of this argument shows that unless $S^{(1)} = \{1, 2, \ldots, N\}$, one must have $S^{(2)} \supsetneq S^{(1)}$, and that $\Gamma_i$ as defined in (12) is the earliest time at which $x_i$ can be determined for $i \in S^{(2)}$. An induction completes the proof of necessity.

For sufficiency, it is enough to observe that if $\{x_i(\Gamma_i); i \notin S\}$ can be determined from knowledge of $\{x_i(t) \mid \underline{T}_i \leq t \leq \overline{T}_i, i \in S\}$, then $\{x_i(\Gamma_i + 1); i \notin S\}$ can be determined from knowledge of $\{x_i(t) \mid \overline{T}_i \leq t \leq \overline{T}_i + 1\}$. Since the latter set can be determined, by assumption, induction shows that $\{x_i(t) \mid t \geq \overline{T}_i + 1 \text{ for } i \in S, \ t \geq \Gamma_i \text{ for } i \notin S\}$ can be determined. Moreover, as in the proof of necessity, $x_i(\Gamma_i)$ for $\Gamma_i$ given by (12) is the earliest value which can be computed for $i \notin S$, thus completing the proof. □

Thus, given a set of initial conditions, and after having determined $\Gamma_i$ for all $i \notin S$, we need to check whether $x_i(\overline{T}_i + 1)$ for $i \in S$ and $x_i(\Gamma_i)$ for $i \notin S$ can be determined. The following theorem shows how to check this.

**Theorem 3.** *Suppose the system is well posed. Let $M_j := \overline{T}_j$ if $j \in S$, and $M_j := \Gamma_j - 1$ if $j \notin S$, when $\Gamma_i$ is specified by (12). Define a "cycle" of computation as one where a single pass is made through each of the equations for $x_1, \ldots, x_N$, in that order, attempting to determine the next unknown value of the variable $x_i(t)$ for $t \geq M_i + 1$. Then the set of*

*initial conditions is adequate if and only if all the values of* $x_i(M_i+1)$ *for* $1 \le i \le N$ *can be determined in* $\sum_{i=1}^{N} \tau_i$ *such cycles, where*

$$\tau_i := \max\{0, \max_{1 \le j \le N}(M_j + 1 - d_{ji} - M_i)\}.$$

**Proof** Sufficiency is obvious by Theorem 2. For necessity, consider $x_j(M_j+1)$. The most recent value of $x_i$ that is needed to compute $x_j(M_j+1)$ is $x_i(M_j+1-d_{ji})$, by virtue of the shortest distance nature of $d_{ji}$ (or else $x_i$ is necessary if $d_{ji} = +\infty$). Thus the values $\{x_i(M_i+1), \ldots, x_i(M_i+\tau_i)\}$ are necessary to determine $\{x_j(M_j+1); 1 \le j \le N\}$. Moreover, the values $\{x_i(M_i+t) \mid t \ge \tau_i + 1\}$ are unnecessary to compute $\{x_j(M_j+1) \mid 1 \le j \le N\}$. Thus, the $\sum_{i=1}^{N} \tau_i$ *new* values in $\mathcal{F} := \{x_i(t) \mid M_i + 1 \le t \le M_i + \tau_i, i \in \{1, 2, \ldots, N\}\}$ are necessary and sufficient to determine $\{x_j(M_j+1) \mid 1 \le j \le N\}$.

Now let $(\ell+1)$ be the first cycle of computation in which no new value of a variable in $\mathcal{F}$ is determined. Then, on cycle $(\ell+2)$ also, no new value in $\mathcal{F}$ can be determined, because any value of any variable determined on the $(\ell+1)$-th cycle is not useful for a value in $\mathcal{F}$. Since, by assumption, the set of initial conditions is adequate, it follows that in each of the cycles $1, 2, \ldots, \ell$, at least one new value in $\mathcal{F}$ is determined. Hence $\ell \le \sum_{i=1}^{N} \tau_i$, and the proof of necessity is complete. □

From the proof of this theorem it can be seen that one also has an algorithm for ordering the equations to compute the values of $\{x_i(t) \mid t \ge M_i + 1\}$ for $1 \le i \le N$, as is made explicit in the following corollary.

**Corollary.**

*(i) In the scheme above, once a variable* $x_i(M_i+\tau_i)$ *is computed, it can be dropped from future cycles, until all variables* $\{x_j(M_j+\tau_j) \mid 1 \le j \le N\}$ *have been determined.*

*(ii) After* $\{x_i(M_i + \tau_i) \mid 1 \leq i \leq N\}$ *have been determined, re–order the equations for* $\{x_i(M_i + \tau_i) \mid 1 \leq i \leq N\}$ *in the order that the* $x_i(M_i + \tau_i)$*'s were determined. This gives a reordering and time–shifts, where in every future cycle, all the variables are updated once in every cycle.*

Thus we have an "efficient" method for updating variables $\{x_i(t) \mid t \geq M_i + \tau_i + 1\}$, in contrast to the brute force method used for determining the $x_i(M_i + 1)$'s.

# 5 ISIM

A package called ISIM has been built around these algorithms. It is designed as a simple to use, free–form computer interface to simulate discrete–time systems. More details may be found in [4] and [6].

ISIM is an "interpretive simulator." It consists of an interpreter for a system of equations. It is composed of three parts, an interpreter, a command director, and a command library. The system to be simulated is defined in an ASCII input file, and passed to ISIM as a command line parameter. A simplified block diagram of the ISIM program is shown in Figure 3.

The interpreter, whose mathematical basis has been described in Sections 2–4, is the feature which sets it apart from other simulators such as SIMNON; see Astrom [7] and Elmqvist [8]. It has the ability to comprehend a system of difference equations and associated parameters which are mostly written in standard mathematical form. Items such as matrix dimensions, a proper ordering of the equations, or any other information that is apparent from the statement of the system need not be provided to the interpreter.

Once a system has been interpreted and transformed into a proper format, the *command director* is used for user interaction. It accepts a variety of commands which change system parameters, change initial conditions, change the type or form of output information required,

Figure 3: Block diagram of ISIM

perform simulation, etc. The *command library* contains the set of functions which performs these tasks. Simulation is the primary function; other functions provide user interaction, full matrix operations including eigenvalue support, support of many common mathematical functions, limited logic support, etc.

The following program for the ELS-based tracker of Section 1 is illustrative.

```
y(t)=-1*y(t-1)+2*u(t-1)+2*u(t-2)+w(t)+0.5*w(t-1)
u(t)= (ystar(t+1)-TRANS(theta(t))*phi(t))/b1
theta(t+1)=theta(t)+P(t)*phi(t)*(y(t+1)-b1*u(t)-TRANS(theta(t))*phi(t))*a(t)
P(t+1)=P(t)-P(t)*phi(t)*a(t)*TRANS(phi(t))*P(t)
a(t)=1/(1+TRANS(phi(t))*P(t)*phi(t))
phi(t)=[1][0][0]*y(t)+[0][1][0]*u(t-1)+[0][0][1]*what(t)
```

```
what(t)=y(t)-b1*u(t-1)-TRANS(theta(t))*phi(t-1)
w(t)=noise
ystar(t)=sin(t)
b1=2
u(0)=0
u(-1)=0
y(0)=0
phi(0)=[0][0][0]
P(0)=[1,0,0][0,1,0][0,0,1]
time 1 to 10
theta(0)=[0][0][0]
print(0,y,w)
```

# 6 Concluding Remarks

In this paper we have developed algorithms that automate the simulation of discrete time systems. These algorithms have been incorporated in the software package ISIM.

The problem tackled in this paper involve systems where the "time" index is a scalar. The more general problem where the "time" index is a vector is useful in program parallelization for scientific computation. It is addressed in [9].

We hope that this work stimulates greater research into the interface between mathematical description of systems and their simulation. It is an interesting open question whether similar progress is possible for continuous–time systems.

# References

[1] L. Guo and H. Chen, "The Åström-Wittenmark self-tuning regulator revisited and ELS-based adaptive trackers," *Transaction on Automatic Control*, vol. 36, pp. 802–812, July

1991.

[2] N. Deo, *Graph Theory.* New Delhi, India: Prentice Hall of India, 1986.

[3] R. M. Karp, R. E. Miller, and S. Winograd, "The organization of computations for uniform recurrence equations," *Journal of the Association for Computing Machinery*, vol. 14, pp. 563–590, July 1967.

[4] R. D. Ellis, "ISIM: An interpretation and simulation program for nonlinear discrete-time systems," Master's thesis, University of Illinois, Urbana, IL, 1988.

[5] L. Liu, C. Ho, and J. Sheu, "On the parallelism of nested for – loops using index shift method," in *Proceedings of the 1990 International Conference on Parallel Processing*, (University Park and London), pp. 119–123, The Pennsylvania State University, The Pennsylvania State University Press, August 13–17 1990.

[6] R. Ravikanth, "Simulation of discrete time systems using ISIM," Technical Report UILU-ENG-91-2215, University of Illinois, Urbana, IL, March 1991. Also, CSL Rept. DC-129.

[7] K. J. Åström, "A SIMNON tutorial." Department of Automatic Control, Lund Institute of Technology, Report TFRT-3168, 1985.

[8] H. Elmqvist, *SIMNON, An Interactive Simulation Program for Nonlinear Systems, User's Manual.* Department of Automatic Control, Lund Institute of Technology, April 1975. Report 7502.

[9] W. J. Aldrich and P. R. Kumar, "Computation of uniform recurrence equations over finite domains," technical report, C. S. L., University of Illinois, Urbana, IL, 1993.