Structure-oriented Behavior Tests in Model Validation[1]

Yaman Barlas and Korhan Kanar
Bogaziçi University
Department of Industrial Engineering
Bebek, Istanbul, Turkey
Email: ybarlas@boun.edu.tr

Structure *validation means establishing that the relationships used in a model are an adequate representation of the real relationships and it can be done in two ways:* ***direct*** *structure testing and* ***indirect*** *structure (or* ***structure-oriented behavior****) testing.* ***Direct*** *structure tests assess the validity of the model structure, by direct comparison with knowledge about real system structure. This involves evaluating each relationship in the model against available knowledge about real system. These tests are qualitative in nature; no simulation is involved.* ***Structure-oriented*** *behavior tests on the other hand assess the validity of the structure indirectly, by applying certain behavior tests on model-generated behavior patterns. For example,* ***extreme-condition*** *(indirect) test involves assigning extreme values to selected parameters and comparing the model-generated behavior to the "anticipated" (or observed) behavior of the real system under the same extreme condition. These are "strong" behavior tests that can provide (indirect) information on potential structural flaws. In a typical* ***structure-oriented*** *behavior test, the modeler makes a claim of the form: "if the system operated under condition C, then the behavior B would result." The model is then run under condition C and it is said to "pass" this structure-oriented behavior test, if the resulting behavior is similar to the anticipated behavior. This article presents a computerized algorithm that automates this comparison/testing process. The modeler would hypothesize a dynamic pattern from the template of all basic patterns (such as "exponential growth", "S-shaped growth", "oscillations", "exponential decay"...) and then run the model under condition C. The algorithm would take the dynamic behavior generated by the model, "recognize" it and test if it belongs to the class hypothesized by the modeler. The algorithm, a* ***Hidden Markov*** *model-based pattern classifier, has been tested with various typical test patterns and proven to be quite effective and reliable.*

1. Introduction

Model validity and validation have long been recognized as one of the main issues in system dynamics field. (Forrester 1968; Forrester et al.1974; Forrester and Senge 1980; Sterman 1984; Barlas 1989a, Barlas & Carpenter 1990). Richardson (1996) identifies "confidence and validation" as one of the eight key problems for the future of system dynamics discipline. Yet, there has been little active research devoted to the development of concrete methods and tools suitable for system dynamics validation. Barlas (1996) states that only three of all the articles published in System Dynamics Review (between 1985 - 1995) deal with model validity/validation. Furthermore, there is no clear evidence of consistent and widespread use of even the established validity tools. (See Peterson and Eberlein 1994 and Scholl 1995). Barlas et al (1989a and 1997) provides a set of tools ("BTS") for

[1] This research is supported by Bogaziçi University Research Fund No. 97HA304

testing the *behavior* validity of a model. This current article presents a method that addresses the *structural* aspect of model validity.

Validity of a causal-descriptive (theory-like, “white-box”) model is critically different than that of a merely correlational (purely data-driven, “black-box”) model. (Barlas 1990 and 1996). In purely correlational (black-box) modeling, since there is no claim of causality in structure, the model is assessed valid, if its output behavior matches the “real output” within some specified range of accuracy, without any questioning of the validity of the relationships that constitute the model. Models that are built primarily for forecasting purpose (such as time-series or regression models) belong to this category. On the other hand, causal-descriptive (white-box) models are statements as to how real systems actually operate in certain aspects. In this case, generating an “accurate” output behavior is not sufficient for model validity; what is crucial is the validity of the internal structure of the model. A white-box model, being a “theory” about the real system, must not only reproduce/predict its behavior, but also explain how the behavior is generated, and possibly suggest ways of changing the existing behavior. System dynamics models - and all models that are design-oriented in general - fall in this category. In short, it is often said that a system dynamics model must generate the “right output behavior for the right reasons.”

Validation of a system dynamics model thus consists of two broad components: *structure* validation and *behavior* validation. *Structure* validation means establishing that the relationships used in the model are an adequate representation of the real relationships, with respect to the purpose of the study. *Behavior* validation consists of demonstrating that the behavior of the model is “close enough” to the observed real behavior. In system dynamics validation, there is no point in testing the behavior validity, until the model demonstrates some acceptable level of structure validity. The model would be refuted if it is shown that a relationship in the model conflicts with a known/established “real relationship”, even if the output behavior of the model matches the observed system behavior. For such models, validity ultimately means validity of the internal structure of the model. (See Barlas 1996 for more discussion).

Although structure validity is crucial, a big majority of technical research in model validation literature deals only with what we call behavior validation. There may be two main reasons why structure validity has been ignored so long in modeling literature. The first one stems from a lack of recognition of the philosophical importance of structure validity in white-box modeling (as opposed to black-box modeling). The second reason has to do with the technical difficulty of designing formal/statistical tools that address structure validity. In an attempt to initiate research in structure validation, Barlas (1989b and 1996) distinguishes between two types of structural testing: 1- *direct* structure testing, 2- *indirect* structure (or structure-oriented behavior) testing. *Direct* structure tests assess the validity of the model structure, by direct comparison with knowledge about real system structure. This involves taking each relationship (mathematical equation or any form of relationship) individually and comparing it with available knowledge about real system. There is no simulation involved and these tests are as such highly qualitative in nature. (See Forrester and Senge 1980 for example tests such as structure confirmation and extreme-conditions). *Indirect* structure (or *structure-oriented* behavior) tests, on the other hand assess the validity of the structure indirectly, by applying certain behavior tests on model-generated behavior patterns. (See Barlas 1989b; Forrester and Senge 1980). These tests involve simulation, and can be applied to the entire model, as well as to isolated sub-models of it. For example, *extreme-condition* (indirect) test involves assigning extreme values to selected parameters and comparing the model-generated behavior to the “anticipated” (or observed) behavior of the real system under the same extreme condition. (See Barlas 1989b for illustrations).

Structure-oriented behavior tests are strong behavior tests that can provide information on potential structural flaws. Their main advantage over direct structure tests is that they are much more suitable to formalize and quantify. Thus, Barlas (1996) submits that *structure-oriented* behavior testing is the most promising direction for research on model validation. Earlier examples of such tests include the “Qualitative Features Analysis” by Carson & Flood (1990) and the “Reality Check” feature of VENSIM simulation software. (See Peterson and Eberlein 1994). In this article, we offer a method, a software, developed for *structure-oriented behavior testing*.

2. Automated Dynamic Pattern Recognition/Testing

In a typical structure-oriented behavior test, the modeler makes a claim of the form: “if the system operated under condition C, then the behavior B would result.” The model is then run under condition C and it is said to “pass” this structure-oriented behavior test, if the resulting behavior is similar to the anticipated behavior. The test condition “C” is typically (but not always) an “extreme condition” since we are much better at anticipating the behavior of the real system under extreme conditions. For example, the modeler may claim that the food consumption (and household waste, etc) would gradually decline to zero, if birth rate is set to zero. The model is run under this extreme condition and if it yields a different dynamics (such as growing food consumption) we suspect that there are some structural flaws in the formulations. Note that in this general version of the test, the modeler hypothesizes a certain “dynamic pattern” (like “gradual decline to zero”) and then tests if the model yields the expected dynamic pattern. The comparison of the anticipated and model-generated patterns is done visually by the modeler, which makes the test subjective and time consuming (considering hundreds of such comparisons). An automated dynamic pattern comparison/testing software could be a big contribution. (Reality Check feature of VENSIM incorporates a limited version of this type of extreme-condition testing, where the “anticipated behavior” is simply a numeric value, not a dynamic pattern). The purpose of this research is to design a computerized algorithm that can take two dynamic patterns, compare them and decide if they “belong to the same class.”

The first problem in the design of a dynamic pattern recognition/validation tool is the determination of the basic patterns to be included in the algorithm. The theory and practice of system dynamics define some forms of basic behavior patterns. A survey of simple analytical models reveals certain types of patterns, including constant, linear, positive and negative exponential trends, s-shaped growth, growth-and-decline and oscillations. These basic patterns, although derived from simple structures, are also frequently encountered in many large-scale and complex models.

The shape of the basic patterns mentioned above can be characterized by various combinations of constant, growth, decline and oscillatory components. A second characterization could be the nature of the rate of growth or decline (negative linear, positive linear, negative exponential, positive exponential or zero) in successive time segments. Using this approach, the set of basic patterns can be enriched further to include patterns such as decline-and-exponential growth or boom-then-bust. Figure

2.1 shows the complete template of basic behavior patterns used in the method. Observe that there are six classes of basic patterns (constant, growth, decline, growth-then-decline, decline-then-growth and oscillatory). Each pattern class may in turn have several pattern variants. For instance, the “growth” class consists of four different growth patterns: (a) linear growth, (b) exponential growth, (c) negative exponential growth and (d) s-shaped growth. Similarly, the “growth-then-decline” class consists of (a) growth-then-exponential decay to zero, (b) growth-then-exponential decay to non-zero and (c) growth-then-crash. (When the steady-state behavior approaches an equilibrium value, whether the equilibrium value is zero or non-zero may be important in evaluating the validity of the extreme-behavior, although this difference seems mathematically trivial. A zero equilibrium is an indication of the total extinction of the variable, which, for example in a population model, has a very different real-life meaning that a nonzero equilibrium).

Oscillation is the last basic pattern on the template of Figure 2.1. Oscillation could be further classified as neutral, damped, expanding, or can even be regarded as an additional pattern component riding on top of any of the patterns listed above. In this research however, we consider oscillation more broadly and treat it in three subclasses: around a constant mean, around a growing trend and around a declining trend. Once the fundamental method proves its effectiveness with the basic patterns of Figure 2.1, extending the template to include additional patterns should not be too difficult.

With the proposed algorithm, the modeler can automatically test a claim like: “if the system operated under condition C, then an exponential crash would result.” S/he would choose the basic pattern “Decline(b)” from the template and then run the model under condition C. The algorithm would take the dynamic behavior generated by the model, “recognize” it and test if it belongs to the hypothesized class (Decline(b)). A thorough structure-oriented behavior test would consist of making numerous validity claims, then let the computerized algorithm test them one by one automatically and report the fraction of passes.

3. Selection And Extraction Of Features From Data

Automating the structure-oriented behavior testing is in part a “pattern recognition/classification” problem. The pattern recognition literature is quite rich with different approaches and algorithms. But due to the “dynamic” nature of our pattern recognition problem, the classical pattern recognition algorithms are inadequate for this task. Therefore, a dynamic pattern recognition algorithm based on “Hidden Markov Models” is to be developed. The major difference between classical and Hidden Markov Model (HMM) based pattern recognition lies in the feature extraction process. In HMM-based pattern recognition, one-dimensional data is divided into segments and a sequence of feature vectors is extracted, whereas in classical approach a single feature set is extracted from the whole data. Patterns to be recognized in dynamic patterns which are the subject of this study are inherently one-dimensional and suitable for HMM implementation.

In our dynamic behavior recognition problem, a dynamic signal can be denoted by a sequence $y(k)$, $k = 1, 2, \dots , K$, where K is the number of data points. As depicted in Figure 3.1, such a signal would be a somewhat distorted (or "noisy") version of one of the patterns given in the template of Figure 2.1. The procedure starts with dividing the sequence $y(k)$ into T number of segments of equal length L. Each segment is denoted

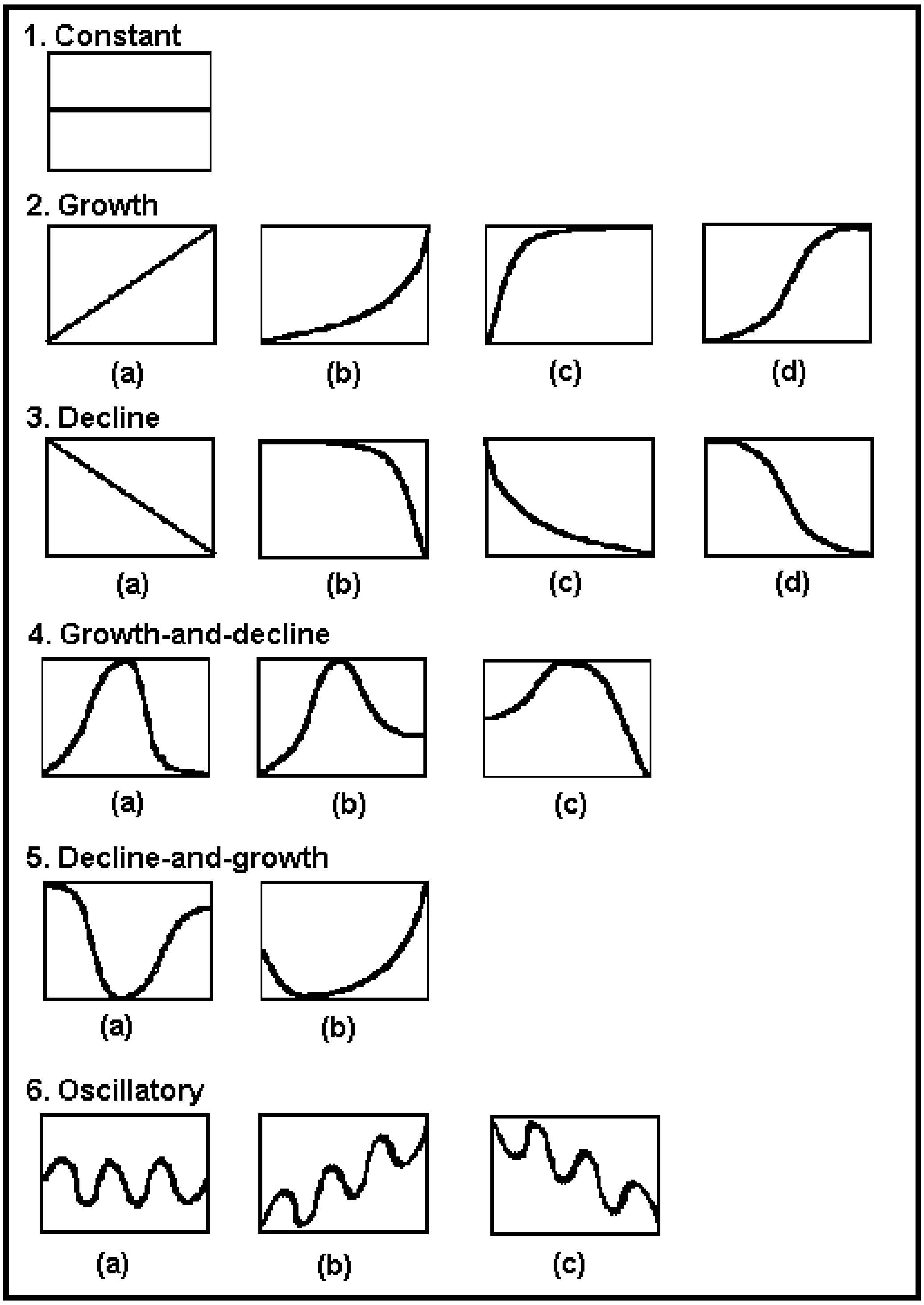

Figure 2.1. Template of Dynamic Patterns

by the sequence $y_t(l)$, t = 1, 2, …, T.

$$y_t(l) = y\,[(t-1)L+l],\quad l = 1, 2, \dots , L \qquad (3.1)$$

Here, the choice of value T is one of the decisions to be made in the design of the recognition system.

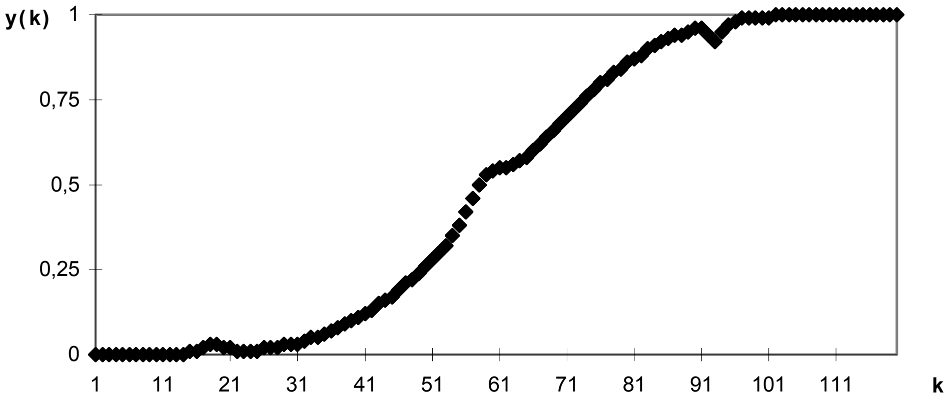

Figure 3.1. A Dynamic Signal Example

The next step is to extract features from each data segment. Basic dynamic patterns are characterized by successive time segments of growth or decline and their trends (as growing or declining rates). Therefore, it is reasonable to form our feature vector using the slope and 2nd derivative ("curvature") information of the data in each segment. The features can be obtained by fitting polynomials to each segment data. The slope of the first order polynomial provides trend information which is either growth, decline or constant. The second order polynomial can be used to obtain the second derivative (which will yield the curvature information).

In our sample data illustrated in Figure 1, we have K=120 data points and assuming that we have taken number of segments as T = 6, the segment size becomes L = 120/6 = 20. The 5th segment shown in Figure 3.2 includes data points from 81 to 100.

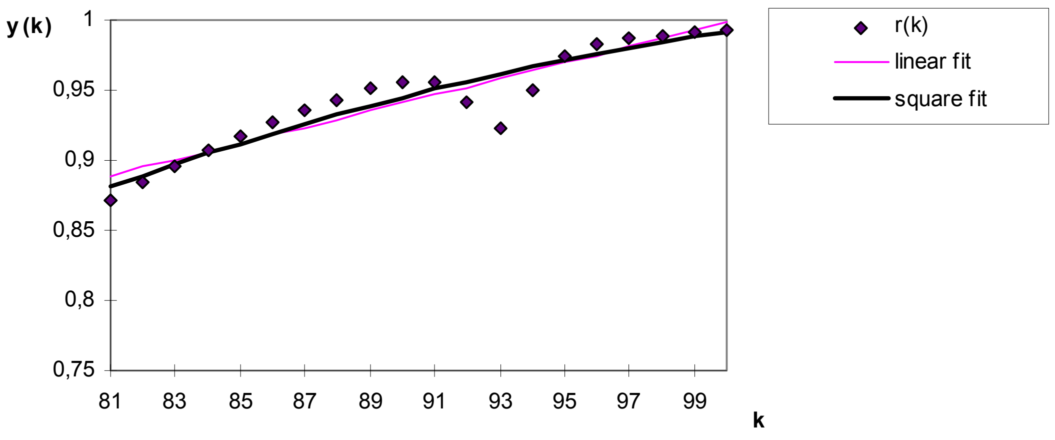

Figure 3.2. Segment 5 of the Signal given in Figure 3.1

The first order polynomials fitted on the segment data points are in the form of,

$$f_t(x) = \phi_{t1}\, x + \phi_{t0}\,, \qquad t = 1, 2, \ldots T \tag{3.2}$$

where, x is a normalized continuous independent variable such that k=1 and k=K correspond to x=0 and x=1 respectively. Here, the value of ϕ_{t1} is our first feature that provides the slope information.

The second order polynomials are in the form of,

$$g_t(x) = \gamma_{t2}\, x^2 + \gamma_{t1}\, x + \gamma_{t0}, \; t = 1, 2, \ldots T \tag{3.3}$$

This can be used to drive the second derivative, however, instead of using simply d^2g_t/dx^2 , we use "curvature" $c_t(x)$, given by,

$$c_t(x) = \frac{d^2 g_t / d^2 x}{\left(1 + \left(dg_t / dx\right)^2\right)^{3/2}} \qquad t = 1, 2, \ldots T \tag{3.4}$$

which is more relevant for our purpose, owing to the “shape” information it yields. Curvature value at any point on the fitted polynomial can be easily calculated from (3.4). We take the midpoint of the segment, denoted by κ_t, and evaluate the curvature $c_t(\kappa_t)$ at the midpoint, as our second feature.

In addition to the slope and the curvature, the level of the state variable also provides useful information. Thus, the segment mean becomes our third feature.

$$\mu_t = \frac{\sum r_t(l)}{L} \tag{3.5}$$

To sum up, our feature vectors are M = 3 dimensional and are given by three components slope, curvature, and mean,

$$o_t = \begin{bmatrix} \phi_{t1} \\ c_t(\kappa_t) \\ \mu_t \end{bmatrix} \qquad for\ t = 1,2,\ldots,T \tag{3.6}$$

Therefore, the result of the segmentation and feature extraction process for a pattern sample using T number of segments is a sequence $\{o_1, o_2, \ldots, o_T\}$. For the example signal in Figure 3.1 - and specifically for its 5th segment in Figure 3.2- feature extraction yields,

$$O = \left\{ o_1, o_2, o_3, o_4, \begin{bmatrix} 0.6711 \\ -2.5308 \\ 0.9440 \end{bmatrix}, o_6 \right\}$$

4. Continuous Density HMM

In a Markov process, a new state $s_t \in \{1, 2, \ldots, N\}$ is entered at each step t = 1, 2, ..., T, depending on an initial probability vector Π and a state transition matrix A, where

$$\Pi = \{\pi_i\},\ \pi_i = \Pr(s_1 = i), \qquad i = 1, 2, \ldots, N \tag{4.1}$$

$$A = \{a_{ij}\},\ a_{ij} = \Pr(s_{t+1} = j \mid s_t = i) \quad i, j = 1, 2, \ldots N \tag{4.2}$$

T = length of the state sequence

N = number of states.

The resulting state sequence is denoted by $S = \{s_t, t = 1, 2, ... ,T\}$ and its realization probability is given by,

$$\Pr(S \mid A, \Pi) = \pi_{s_1} \prod_{t=2}^{T} a_{s_{t-1}s_t} \tag{4.3}$$

In a continuous density HMM the states of the process are not observed directly. The outcome of the process is a sequence of vectors $O = \{o_1, o_2, ..., o_T\}$ where $o_t \in \Re^M$, M-dimensional Euclidean space. The output vector is produced according to a probability distribution, depending on the current state. There are N number of observation-vector probability distributions, denoted by a vector B, where

$B = \{b_j(o_t)\}$, $b_j(o_t)$ = a posteriori density of observation vector o_t in state j (4.4)

Here we make the assumption that the observation densities are Gaussian. In this case, a conditional mean vector μ_j and a conditional covariance matrix V_j determine the density corresponding to state j . We denote this density as $N(\mu_j, V_j)$.

A continuous density HMM is specified by parameters A, Π, and B. The parameters can be compactly represented as a set denoted by $\lambda = (A,\Pi,B)$.

With this model, a process is described such that transition probabilities from a step t to t+1 is only dependent on the state at step t. This may not always provide satisfactory results when dynamic behavior patterns are considered. In order to increase the realism of the model, a nonstationary component can be introduced, i.e. by introducing time-dependence in transition probabilities (He and Kundu 1990). In this case, rather than having a single state transition matrix, A, we have T-1 number of matrices, A_t, t=1,2,..,T-1.

The states of an HMM only reflect clustering properties of the features and should not be regarded as having a physical meaning (He and Kundu 1990).

5. Application Of HMM To Dynamic Behavior Recognition

In our problem, each observation vector, o_t, is the feature vector extracted from the t^th segment of the signal as explained in Section 3. Each pattern class is characterized by a HMM, i.e. $\lambda = (A,\Pi,B)$, which is built using the training pattern set.

Optimization Criterion

Suppose we are given a model λ and an observation sequence $O = \{o_1, o_2, ..., o_T\}$. A choice of optimization criterion in estimation (training) and classification processes is to maximize the state-optimized likelihood function (Juang and Rabiner 1990) defined by,

$$p(O, S^* \mid \lambda) = \max_S p(O, S \mid \lambda)$$
$$= \max_S \pi_{s_1} b_{s_1}(o_1) \prod_{t=2}^{T} a_{s_{t-1}s_t} b(o_t) \tag{5.1}$$

where, $S^* = \{s_1^*, s_2^*, \ldots, s_T^*\}$ is the state sequence associated with the state-optimized likelihood function.

Equation (5.1) is the density of the optimal or the most likely state sequence path among all possible state sequences.

5.1 Training of the HMM

The training procedure is the process in which the model parameter set λ = (A,Π,B) for a class is adjusted so that the state optimized likelihood function defined in (5.1) is maximized for the "training set" of that class. (A training set for a given class is a collection of dynamic patterns, all of which are "noisy" versions of the basic dynamic pattern that define that classs). At the end of the training phase, the algorithm essentially "learns" the basic dynamic pattern of a given class. For Gaussian density functions, there is a "segmental K-means algorithm" that converges to the state-optimized likelihood function (Juang and Rabiner 1990). Segmental K-means algorithm can be outlined briefly as below.

Given an initial model $\hat{\lambda}^0$, calculate

$$\hat{\lambda}^{k+1} = \arg\max_{\lambda^k} \left\{ \max_S \ p(O, S^* \mid \hat{\lambda}^k) \right\} \qquad (5.2)$$

iteratively until $\hat{\lambda}^{k+1} = \hat{\lambda}^k$, where k is the iteration number. Maximization of the state-optimized likelihood p(O,S*|λ) in (5.2) for each training observation sequence is achieved using the Viterbi algorithm (given in Appendix A). Given a model λ = (A,Π,B) and an observation sequence O ={o_1, o_2, ..., o_T}, the Viterbi algorithm finds the state-optimized likelihood function and the optimal state sequence. At each iteration, the optimal state sequences are assigned to the observation vectors of each training sample. The new states are again used to estimate new model parameters. The iteration proceeds until none of the state assignments change at the end of the maximization. (Appendix A). A more detailed description of the process is given by the numerical example in the next section.

5.2 Classification

As a result of the training procedure, we obtain one HMM for each class. We denote the P models by λ_p , p = 1, 2, …, P. When a signal O of unknown class is given, we calculate p(O,S*|λ_p) for each class p = 1,2,..,P using the Viterbi algorithm. The goal is to clasify the given signal in one of the known pattern classes. The classification is based on the state-optimized likelihood function which is a measure of how well the input signal is representative of a given class. However, the likelihood values for different classes by themselves may not be suitable for direct comparison, depending on the degree of dispersion of the training feature vectors within their clusters. He and Kundu (1990) normalize the state-optimized likelihood function for a class by dividing it to the mean of the likelihoods of the training set used for that class. We move one step further and take into account the variation of the likelihood function values of the training samples within classes. Our criterion is the state-optimized likelihood value of the input signal normalized by the mean, m_p, and the standard deviation of the likelihood function of the training set, $\hat{\sigma}_p$. We denote the normalized likelihood function by h(O,S|λ_p) such that,

$$h(O, S^* \mid \lambda_p) = \frac{p(O, S^* \mid \lambda_p) - m_p}{\hat{\sigma}_p} \qquad (5.3)$$

Here,

$$m_p = \frac{1}{n_p}\sum_{i=1}^{n_p} p(O_i^p, S^* | \lambda_p) \qquad p = 1,2,\ldots,P \qquad (5.4)$$

and,

$$\hat{\sigma}_p = \left\{ \frac{1}{n_p - 1}\sum_{i=1}^{n_p} (p(O_i^p, S^* | \lambda_p) - m_p)^2 \right\}^{1/2} \qquad p = 1,2,\ldots,P \qquad (5.5)$$

where, n_p is the number of training samples used for class p and O_i^p is the observation (feature) vector sequence extracted from the ith training sample of class p.

When an input does not belong to any of the classes, it should not be erroneously classified into one of the classes. The criterion in (5.3) also allows us to define a common lower rejection region. Since the likelihood values of the training set are approximately normal, (5.5) is an estimator for the population standard deviation. We take the ad hoc value of 3 times the standard deviation of the likelihood functions within a class as our rejection region. Thus, if the outcome of the classifier (5.3) is less than 3.0 for all P classes, we classify the input signal into the "none" class. Therefore our classification rule becomes:

$$choose \begin{cases} \arg\max_p \left[h(O, S^* | \lambda_p) \right] & if \ \max_p h(O, S^* | \lambda_p) > -3.0 \\ none & otherwise \end{cases} \qquad (5.6)$$

6. Training Of A Pattern Class - A Simplified Example

In this simplified example we a have training set of 8 sample patterns each belonging to the "negative-exponential growth" class. Each pattern consists of 120 data points. The training samples are presented in Figures 6.1 (a) and (b).

We take number of segments T =10 and number of states, N=3 as model parameters. For simplicity of illustration, a stationary model will be used.

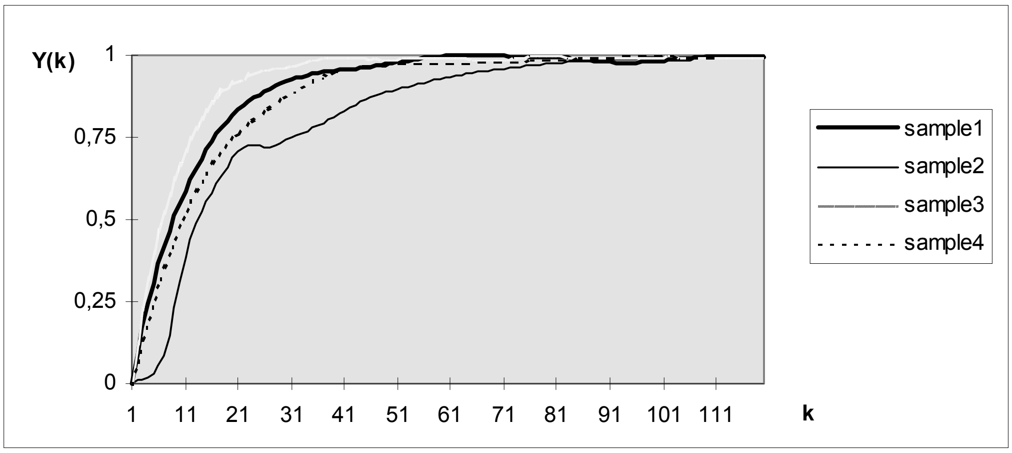

Figure 6.1 (a) Training samples for negative-exponential growth class (Samples 1 - 4)

Figure 6.1 (b) Training samples for negative-exponential growth class (Samples 5 - 8)

The first step in the training procedure is to segment each pattern and extract features from the segments as outlined in Section 3. The resulting observation sequence for sample 1, denoted by O^1, is a set of 10 vectors of size 3x1:

$$O^1 = \left\{ \begin{bmatrix} 6.6096 \\ -0.2507 \\ 0.3602 \end{bmatrix}, \begin{bmatrix} 2.3209 \\ -1.6724 \\ 0.7802 \end{bmatrix}, \begin{bmatrix} 0.6991 \\ -4.2347 \\ 0.9186 \end{bmatrix}, \begin{bmatrix} 0.2213 \\ 0.2285 \\ 0.9594 \end{bmatrix}, \begin{bmatrix} 0.2607 \\ -2.2482 \\ 0.9866 \end{bmatrix}, \begin{bmatrix} -0.0273 \\ -2.3789 \\ 0.9985 \end{bmatrix}, \begin{bmatrix} -0.0553 \\ -1.8583 \\ 0.9938 \end{bmatrix}, \begin{bmatrix} -0.1139 \\ 3.2653 \\ 0.9812 \end{bmatrix}, \begin{bmatrix} 0.1383 \\ -1.0925 \\ 0.9860 \end{bmatrix}, \begin{bmatrix} -0.0002 \\ -0.0305 \\ 0.9919 \end{bmatrix} \right\}$$

Here, the first feature corresponds to the slope in the segment, the second designates curvature, where the negative and positive values indicate concave down or up respectively. The third feature gives level information. We have similar vector sequences for the training samples 2 to 8, which make altogether 80 feature vectors.

Initial Model.

First, all 80 feature vectors are clustered in 3-dimensional space. We use the same minimum distance algorithm used by He and Kundu (1991). The result is an assignment of each feature vector, $S_j = \{s_{jt}, t = 1,2,\ldots,6\}$ for $j = 1,2,\ldots,8$, to one of the 3 states:

$S_1 = \{1,2,3,3,3,2\}$
$S_2 = \{1,3,2,2,3,3\}$
$S_3 = \{1,2,3,3,3,3\}$
$S_4 = \{1,2,2,3,3,3\}$
$S_5 = \{1,2,2,3,2,3\}$
$S_6 = \{1,2,2,3,3,3\}$
$S_7 = \{1,2,3,3,3,2\}$
$S_8 = \{1,2,3,3,3,3\}$

Clustering of the vectors into 3 states in 3-dimensional space is visually depicted in Figure 6.2.

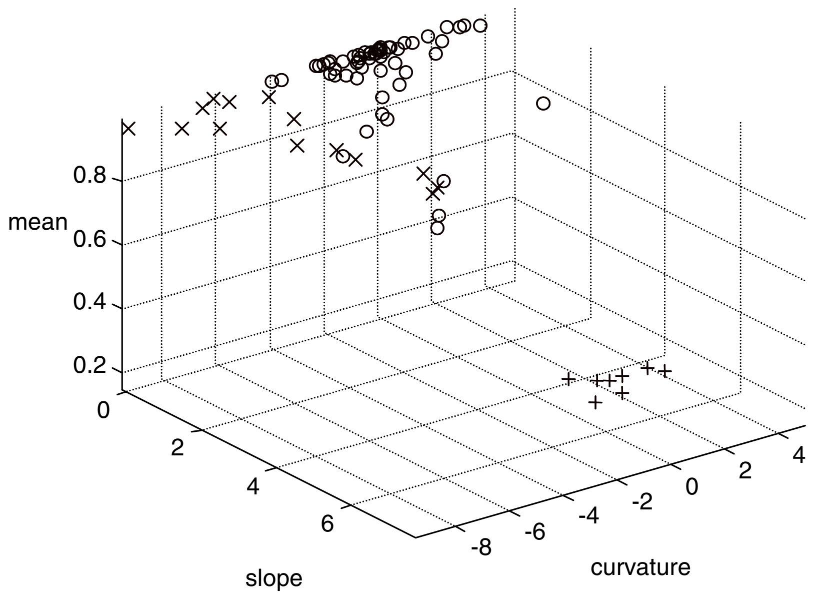

Figure 6.2. Initial cluster of the feature vectors. (+: State-1, o: State2, x: State3)

From these state assignments, initial model parameters are estimated. Initial probability vector, Π, and the state transition matrix, A, can be calculated using (He and Kundu 1990):

For $1 \le i \le N$

$$\hat{\pi}_i = \frac{\text{number of occurances } \{s_1 = 1\}}{\text{number of training samples}} \tag{6.1}$$

For $1 \le i \le N$ and $1 \le j \le N$

$$\hat{a}_{ij} = \frac{\text{Number of occurances } \{s_t = i \text{ and } s_{t+1} = j\} \text{ for all } t}{\text{Number of occurances } \{s_t = i\} \text{ for all } t} \tag{6.2}$$

For the current state assignment, calculation yields,

$$\Pi = \{\hat{\pi}_i\} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad A = \{\hat{a}_{ij}\} = \begin{bmatrix} 0 & 0.8750 & 0.1250 \\ 0 & 0.3077 & 0.6923 \\ 0 & 0.2105 & 0.7895 \end{bmatrix}$$

The mean vectors and the covariance matrices for states 1, 2, and 3 are estimated using (He and Kundu 1990):

For $1 \le i \le N$

$$\hat{\mu}_i = \frac{1}{N_i} \sum_{o_t \in i} o_t, \tag{6.3}$$

$$\hat{V}_i = \frac{1}{N_i} \sum_{o_t \in i} (o_t - \hat{\mu}_i)^t (o_t - \hat{\mu}_i) \tag{6.4}$$

Calculation of the mean vector and covariance matrix for this state assignment gives the following:

$$\mu_1 = \{\hat{\mu}_1\} = \begin{bmatrix} 4.8072 \\ -0.3731 \\ 0.4756 \end{bmatrix}, \qquad \mu_2 = \{\hat{\mu}_2\} = \begin{bmatrix} 0.5322 \\ -2.2591 \\ 0.9297 \end{bmatrix}, \qquad \mu_3 = \{\hat{\mu}_3\} = \begin{bmatrix} 0.0733 \\ 0.5042 \\ 0.9807 \end{bmatrix}$$

and,

$$V_1 = \{\hat{V}_1\} = \begin{bmatrix} 0.1312 & 0.0124 & 0.0158 \\ 0.0124 & 0.0278 & -0.0115 \\ 0.0158 & -0.0115 & 0.0089 \end{bmatrix}, \qquad V_2 = \{\hat{V}_2\} = \begin{bmatrix} 0.2033 & -0.1488 & -0.0245 \\ -0.1488 & 0.7496 & 0.0189 \\ -0.0245 & 0.0189 & 0.0032 \end{bmatrix},$$

$$V_3 = \{\hat{V}_3\} = \begin{bmatrix} 0.0235 & 0.0921 & -0.0066 \\ 0.0921 & 1.6103 & -0.0373 \\ -0.0066 & -0.0373 & 0.0023 \end{bmatrix}$$

The values μ_i and V_i determine the observation probability densities, B, such that,

$$\hat{b}_j(o_t) = \frac{1}{(2\pi)^{M/2}\left|\hat{V}_j\right|^{1/2}} \exp\left[-\frac{1}{2}(o_t - \hat{\mu}_j)\hat{V}_j^{-1}(o_t - \hat{\mu}_j)^t\right] \qquad (6.5)$$

Iteration 1.

For the model, λ=(A, Π, B), at hand, optimal state sequences for the 8 observation sequences are traced using the Viterbi algorithm (Appendix A). The resulting state sequences, S_i, i=1,2,..8, and their corresponding state-optimized likelihood function values are calculated to be:

S_1 ={1,2,3,3,3,3}	(-13.2520)
S_2 ={1,3,2,2,3,3}	(-22.6685)
S_3 ={1,2,3,3,3,3}	(-7.4058)
S_4 ={1,2,2,3,3,3}	(-6.4410)
S_5 ={1,2,2,3,3,3}	(-10.1784)
S_6 ={1,2,2,3,3,3}	(-6.7685)
S_7 ={1,2,3,3,3,2}	(-7.6222)
S_8 ={1,2,3,3,3,3}	(-4.5015)

(note that the optimum "likelihood functions" shown in parentheses are negative, bacause logarithms of the likelihoods are taken in computations in order to prevent numerical errors)

The assignment of the observation vectors to different states, new model parameters, λ = (A,Π,B), are reestimated using (6.1-5). The cycle of model parameter estimation and tracing the optimal state sequences is repeated until no new assignments are made. At the end of the 5th iteration, the model converges to:

$$\Pi = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad A = \begin{bmatrix} 0 & 0.8750 & 0.1250 \\ 0 & 0.1111 & 0.8889 \\ 0 & 0.0435 & 0.9565 \end{bmatrix}$$

$$\mu_1 = \begin{bmatrix} 4.8072 \\ -0.3731 \\ 0.4756 \end{bmatrix}, \qquad \mu_2 = \begin{bmatrix} 0.8140 \\ -2.7135 \\ 0.8968 \end{bmatrix}, \qquad \mu_3 = \begin{bmatrix} 0.0803 \\ 0.1013 \\ 0.9804 \end{bmatrix}$$

$$V_1 = \begin{bmatrix} 0.1312 & 0.0124 & 0.0158 \\ 0.0124 & 0.0278 & -0.0115 \\ 0.0158 & -0.0115 & 0.0089 \end{bmatrix}, \quad V_2 = \begin{bmatrix} 0.1235 & 0.0919 & -0.0157 \\ 0.0919 & 0.6679 & -0.0080 \\ -0.0157 & -0.0080 & 0.0024 \end{bmatrix}, \quad V_3 = \begin{bmatrix} 0.0214 & 0.0637 & -0.0056 \\ 0.0637 & 2.0040 & -0.0295 \\ -0.0056 & -0.0295 & 0.0019 \end{bmatrix}$$

The above HMM model characterizes the "negative-exponential-growth" class. The final cluster of the feature vectors is depicted in Figure 6.3.

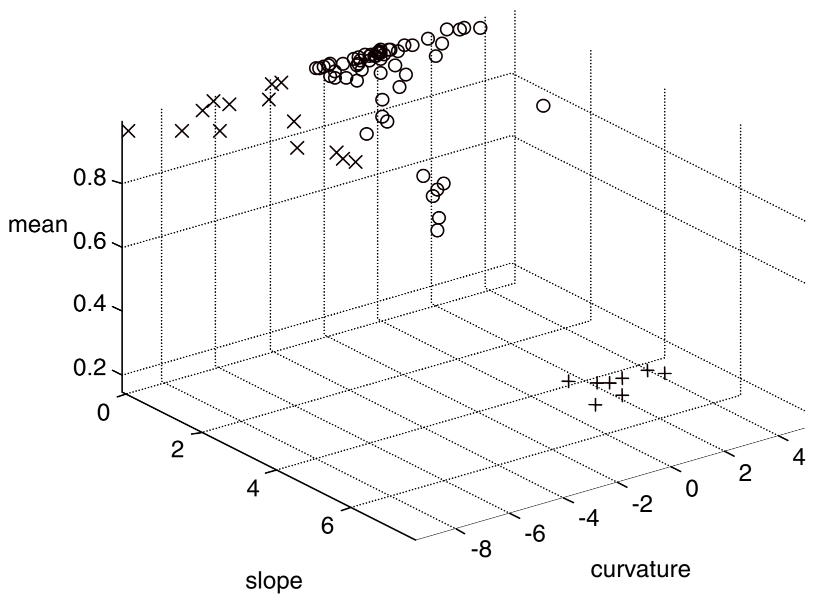

Figure 6.3. Final cluster of the feature vectors. (+: State-1, o: State2, x: State3)

7. Classification Examples

This section is intended to give a few classification examples. The HMMs for 5 classes are trained using training sets with sizes ranging from 29 to 119 samples. Referring to Figure 1.1 these classes are : i) negative-exponential growth (2-c), ii), positive-exponential growth (2-b), iii) S-shaped-growth (2-d), iv) Growth-and-decline-to-zero (4-a), v) Growth-and-decline-to-nonzero (4-b). In the feature extraction process, data is divided into 12 segments. The nonstationary HMMs are based on 6 states.

The test samples, representative of each of the 5 classes, have been used for illustration (Figures 7.1 to 7.5). Here, the test signal in Figure 7.1 is generated from class 2-b, Figure 7.2 from class 2-c, Figure 7.3 from class 2-d, Figure 7.4 from class 4-a, and, Figure 7.5 from class 4-b. In the generation of the test samples, "pure" patterns of their representative classes are first constructed, and then autocorrelated noise is added. The noise level used in the test samples is comparable to the average noise level in the training samples. Finally, the test sample-6 shown in Figure 7.6 is a signal that is generated from a pattern that does not belong to any of the 5 pattern classes.

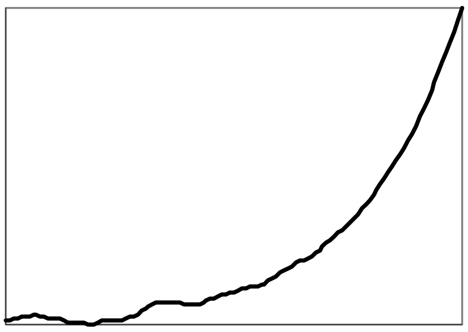
Figure 7.1. Test Sample-1

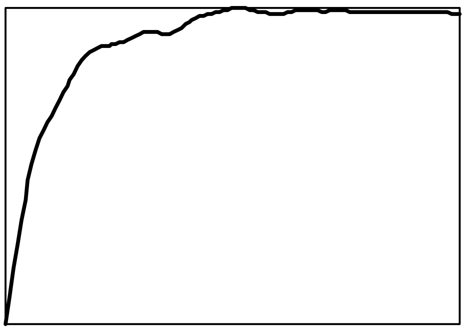
Figure 7.2. Test Sample-2

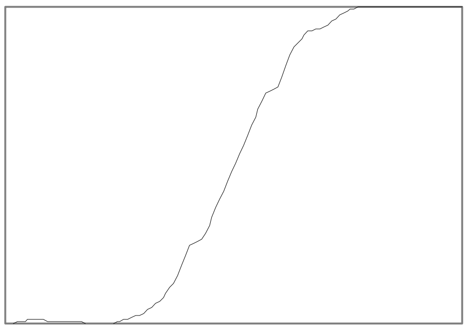
Figure 7.3. Test Sample-3

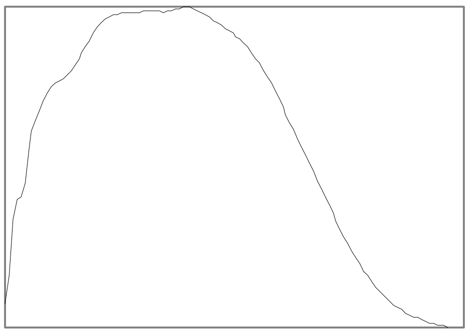
Figure 7.4. Test Sample-4

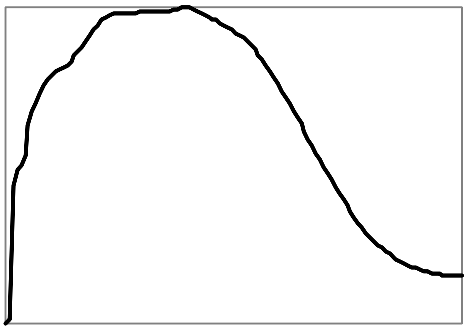
Figure 7.5. Test Sample-5

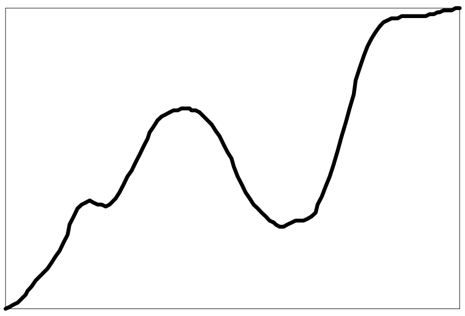
Figure 7.6. Test Sample-6

Test Sample-1

For the first test sample, the calculation results of the classification criterion, which is the normalized optimum likelihood function (5.3), is given in Table 7.1. According to our classification rule (5.6), the maximum value is attained for Class 2-b. Thus, Test Sample-1 is corerctly classified as belonging to class 2-b, i.e. positive exponential growth.

Class 2-b	Class 2-c	Class 2-d	Class 4-a	Class 4-b
-1.958	-13.086	-3.749	-11.923	-12.862

Table 7.1. Normalized optimum likelihood values of the Test Sample-1 for each of the 5 classes. (Negative values result from logarithms used in computations in order to prevent numeric errors)

Test Sample -2

Similarly, the normalized state-optimized likelihoods for each HMMs of Test Sample-2 are presented in Table 7.2. The greatest likelihood is observed for Class 2-c, which is the negative exponential growth - correct class.

Class 2-b	Class 2-c	Class 2-d	Class 4-a	Class 4-b
-25.173	-0.925	-3.3775	-7.110	-4.602

Table 7.2. Normalized optimum likelihoods of the Test Sample-2 for each of the 5 classes.

Test Sample -3

By similar reasoning, Test Sample-3 is correctly classified into Class 2-d, i.e. s-shaped growth class.

Class 2-b	Class 2-c	Class 2-d	Class 4-a	Class 4-b
-12.409	-8.054	0.257	-16.405	-14.917

Table 7.3. Normalized optimum likelihoods of the Test Sample-3 for each of the 5 classes.

Test Sample-4

As a result of the classification rule, Test Sample-4 is correctly classified into Class 4-a, which is the growth-and-decline-to-zero pattern.

Class 2-b	Class 2-c	Class 2-d	Class 4-a	Class 4-b
-27.459	-9.906	-8.336	-1.526	-1.783

Table 7.4. Normalized optimum likelihoods of the Test Sample-4 for each of the 5 classes.

Test Sample-5

Test Sample-5 is classified correctly into Class 4-b, i.e. growth-and-decline-to-nonzero pattern.

Class 2-b	Class 2-c	Class 2-d	Class 4-a	Class 4-b
-27.401	-9.747	-8.220	-1.816	-0.647

Table 7.5. Normalized optimum likelihoods of the Test Sample-5 for each of the 5 classes.

Test Sample-6

Table 7.6 shows the normalized state optimized likelihoods of Test Sample-6 under the five HMMs corresponding to each class. The largest likelihood seems to occur with Class 2-d, which is the s-shaped growth. However, this value is less than -3.0, and, according to classification rule (5.6) the input sample is correctly classified as belonging to none of the classes.

Class 2-b	Class 2-c	Class 2-d	Class 4-a	Class 4-b
-23.424	-9.495	-6.843	-14.976	-15.079

Table 7.6. Normalized state-optimized likelihoods of the Test Sample-6 for each of the 5 classes.

The above test examples demonstrate that the algorithm can indeed classsify a given dynamic pattern into the proper pattern class. We are now in the process of testing the performance and reliability of the algorithm with extensive input test patterns. Results obtained so far are very promising.

8. Conclusion

This article presents an automated, computerized method for *Structure-oriented* behavior testing. In a typical structure-oriented behavior test, the modeler makes a claim of the form: “if the system operated under condition C, then the behavior B would result.” The model is then run under condition C and it is said to “pass” this structure-oriented behavior test, if the resulting behavior is similar to the anticipated behavior. This paper presents an algorithm that automates this comparison/testing process. The modeler would hypothesize a dynamic pattern from the template of all basic patterns (such as "exponential growth", "S-shaped growth", "oscillations", "exponential decay"…) and then run the model under condition C. The algorithm takes the dynamic behavior generated by the model, “recognizes” it and tests if it belongs to the class hypothesized by the modeler. The algorithm, a Hidden Markov model-based pattern classifier, has been tested with various typical test patterns and proven to be very effective and reliable.

The tool, being a “dynamic behavior pattern recognizer/classifier”, can potentially contribute in other steps of system dynamics methodology, most notably in data analysis, reference behavior identification, behavior validation and finally in automating policy improvement (pattern-oriented) simulation experiments.

Appendix A - Viterbi Algorithm (He and Kundu 1990)

Step 1. Initialization
For $1 \le i \le N$,

$$\delta_1(i) = \pi_i b_i(o_i)$$
$$\psi_1(i) = 0$$

Step 2. Recursive computation
For $2 \le t \le T$ for $1 \le j \le N$

$$\delta_t(j) = \max_{1\le i\le N}\left[\delta_{t-1}(i)a_{ij}\right]b_j(o_t)$$
$$\psi_t(j) = \arg\max_{1\le i\le N}\left[\delta_{t-1}(i)a_{ij}\right]$$

Step 3. Termination

$$P^* = \max_{1\le i\le N}\left[\delta_T(i)\right]$$
$$s^*_T = \arg\max_{1\le i\le N}\left[\delta_T(i)\right]$$

Step 4. Tracing back the optimal sequence.
For t = T-1, T-2, …, 1

$$s^*_T = \psi_{t+1}(s^*_{t+1})$$

P* is the state-optimized likelihood function, and
$S^* = \{s_1^*, s_2^*, \ldots, s_T^*\}$ is the optimal state sequence.

Note 1: When calculating the state-optimized likelihood function using the Viterbi algorithm, successive multiplications in the evaluation of $\delta_t(j)$ may lead to underflow errors. To avoid this, logarithms of probability and density values are taken and multiplications are replaced by additions. Logarithms for zero values can be assigned a very small negative number.

Note 2: The procedure above is given for a stationary HMM. In nonstationary models, a_{ij}'s in the algorithm are replaced by $a_{ij}(t)$'s.

References

Barlas, Y. (1996). "Formal Aspects of Model Validity and Validation in System Dynamics" *System Dynamics Review* 12(3):183-210.

Barlas, Y. and S. Carpenter. (1990). "Philosophical Roots of Model Validation: Two Paradigms." *System Dynamics Review* 6(2):148-166.

Barlas, Y. (1989a). "Multiple Tests for Validation of System Dynamics Type of Simulation Models." *European Journal of Operational Research* 42(1):59-87.

Barlas, Y. (1989b). "Tests of Model Behavior That Can Detect Structural Flaws: Demonstration With Simulation Experiments." In *Computer-Based Management of Complex Systems*: International System Dynamics Conference. P.M.Milling and E.O.K.Zahn, eds. Berlin: Springer-Verlag.

Carson, E.R.and R.L.Flood. (1990). "Model Validation: Philosophy, Methodology and Examples." *Trans Inst MC*.12(4): 178-185.

Eberlein, R.L and D.W. Peterson. (1994). "Understanding Models with VENSIM." In *Modeling For Learning Organizations*. Morecroft, J.D.W and J.D. Sterman, eds. Portland, OR: Productivity Press

Forrester J.W. and P.M.Senge. (1980). "Tests For Building Confidence in System Dynamics Models." In *System Dynamics*. Legasto, A.A., J.W. Forrester and J.M. Lyneis, eds. Amsterdam: North-Holland

Forrester, J.W., G.W. Low and N.J. Mass. (1974). "The Debate on World Dynamics: A Response to Nordhaus." *Policy Sciences* 5: 169-190.

Forrester, J.W. (1968). "A Response to Ansoff and Slevin." *Management Science* 14: 601-618.

He, Y. and Kundu, A. (1991). 2-D Shape Classification Using Hidden Markov Model. *IEEE Trans. Patt. Anal. Machine Intell. vol. PAMI-13, no. 11*, 1172-1184.

Juang, B. and Rabiner, L. R. (1990). The Segmental K-Means Algorithm for Estimating Parameters of Hidden Markov Models. *IEEE Trans. Acoust. Speech Signal Processing vol. ASSP-38, no. 9*, 1639-1641

Peterson, D.W. and R.L. Eberlein. (1994). “Reality Check: A Bridge Between Systems Thinking and System Dynamics.” *System Dynamics Review* 10(2-3): 159-174.

Richardson, G.P. (1996). “Problems for the future of system dynamics.” *System Dynamics Review* 12(2): 141-157.

Scholl, G.J. (1995). “Benchmarking the System Dynamics Community: Research Results.” *System Dynamics Review* 11(2): 139-155.

Sterman, J. D. (1984). Appropriate Summary Statistics for Evaluating the Historical Fit of System Dynamics Models. *Dynamica*. 10(2):51-66.