13

Inductive Reasoning

Preview

In the last chapter, we discussed a crude approach to analyzing the behavior of a system by means of a coarse, qualitative, structural description of the real physical system. The claim was that this models the way humans reason about processes, and since humans are very apt at making correct decisions on the basis of incomplete knowledge, this approach may enable algorithms to duplicate such aptitude. It turned out that the results were not as promising as some researchers would like us to believe. Strong indicators can also be found that humans mostly assess the behavior of a system not on the basis of qualitative physical considerations, but on the basis of analogies with similar processes, the operation of which they have previously observed, i.e., that they use *pattern recognition* to analyze system behavior. In this chapter, we shall discuss one of several pattern recognition techniques that may be able to mimic how humans apply pattern recognition to reason about system behavior.

13.1 Introduction

In Chapter 12, I mentioned the following example: If I hold a glass with water in my hand, and if I open my fingers, I *know* that the glass will fall to the ground, break into a thousand pieces, and spill the water over my carpet. I don't need to solve a set of differential equations to come up with this assessment. I argued that it is sufficient to know that a positive force exists that pulls the glass down. The amount of that force is not important to the conclusion, except if I wish to know *when* exactly the glass will hit the floor. But is

this really how I came to the correct conclusion? Or is it maybe because I remembered that my father, when I was 10 years old, let a two liter bottle of Chianti wine slip through his fingers, and our dog came, and licked the entire contents of the bottle from the carpet, and got himself completely drunk? But if this were true, how come that my brain correlated the water glass event with a seemingly unrelated event that happened more than three decades ago, taking into consideration that I observe and store an unbelievable manifold of different *episodes* every day of my life? How come a physician whom I had visited only once four years earlier introduced himself to me (since his office had misplaced my previous patient card), but frowned after the third word and exclaimed: "But you *have* been here before, haven't you?" although the guy must be seeing 50 patients a day? In this chapter, and also in the next, I shall try to bring us a little closer to unraveling these mysteries.

Let me start by explaining why I believe that pattern recognition is a much more frequently used and much more powerful tool in assessing qualitatively the behavior of a system. I still own a dog who loves to play ball. I kick the ball with the side of my foot (I usually wear sandals, and a straight kick hurts my toes), and my dog runs after the ball as fast as he can. I was able to observe the following phenomenon: If I place my foot to the left of the ball, my dog will turn to the right to be able to run after the ball as soon as I hit it. He somehow *knows* that the ball will be kicked to the right. If I now change my strategy, and place my foot to the right of the ball, my dog immediately swings around to be ready to run to the left. He obviously has some primitive understanding of the mechanics involved in ball kicking. However, I assure you that I never let my dog near my physics texts, and thus, he had no opportunity to study Newton's laws — not even in their naïve form.

A number of strong arguments can be mentioned in favor of the pattern recognition approach, but a number of strong arguments can also be brought forward that advise against it. What makes pattern recognition attractive?

(1) In Chapter 10, I demonstrated that *structure characterization* is a very tough problem. In the pattern recognition approach, we bypass this problem entirely by going directly for the behavior itself. It is a much simpler task to identify a pattern than to characterize a structure.

(2) Pattern recognition is qualitative by nature. We don't need to artificially modify our problem to make it qualitative.

(3) Pattern recognition algorithms lend themselves naturally to implementation on a parallel processor architecture. Structure characterization algorithms don't share this property. Thus, pattern recognizers can be very fast.

What makes pattern recognizers unattractive for the purpose of reasoning?

(1) As I mentioned earlier, the behavioral complexity of a system is much larger than its structural complexity. Thus, we need to store *much* more information about a system if we represent it directly in a behavioral form. Consequently, pattern recognizers will usually *not* be able to characterize a system in general, but only for a limited set of input stimuli.

(2) In a reasoning system, we like to know why a system behaves in one way and not in another. Pattern recognition does not usually provide much insight into the *why*, only into the *how*. Consequently, it is very difficult to use knowledge extracted from a pattern recognizer in the process of *knowledge generalization*. However, this is also a weakness of us humans. Only few people are truly capable of effectively participating in the process of knowledge generalization.

Problem #1 above can be overcome if we are able to make the pattern recognizer so fast that we can learn the behavior of our system *on-line*. In that case, we can identify the system behavior in the vicinity of the current operating point, make a decision, implement it, observe the effects of that implementation, determine a new operating point, and re-identify the process for this new operating point. Problem #2 above is tough luck, but remember that the Naïve Physics approach hasn't helped us much with this problem either. This is simply an awfully difficult problem to tackle.

The question remains: How did my physician recognize me after all these years? Somehow, we must be able to store *generic properties* of a system away, properties which are so generic that we can easily and quickly access them, and compare them with the generic properties of a new system to find similar or equal patterns. In this chapter, I shall introduce one methodology that allows us to mimic this capability to a certain extent.

13.2 The Process of Recoding

In the last chapter, I discussed how continuous trajectories are discretized into episodes for the purpose of Naïve Physics modeling. Inductive reasoning, like all other qualitative modeling techniques, also requires a discretization of continuous phenomena.

This time, we shall allow more values than just −, 0, and +, and we shall concentrate on the regions themselves rather than on the landmarks that separate these regions from each other. The values that represent such regions can be symbolic (for example, *tiny*, *small*, *average*, and *big*, denoting four distinct regions), or they can be integer numbers (for example, '1', '2', '3', and '4', denoting the same four regions as above). In an inductive reasoning system which is coded in LISP, symbolic names are probably preferred, whereas in an inductive reasoner coded in a predominantly numeric software such as MATLAB or CTRL–C, integers will be the representation of choice. From a practical point of view, it really doesn't matter which of the two representations is being used since one can easily be mapped into the other. The symbolic representation will make the code more readable though.

In inductive reasoning, the regions are called *levels*. Notice the difference with System Dynamics. In System Dynamics, a "level" denotes a continuous state variable, whereas in inductive reasoning, it denotes one value of a discrete state variable. The process of discretizing continuous trajectories into discrete episodes is called *recoding*. Finally, a combination of legal levels of all state variables of a model is called a *state*. Thus, a model with n state variables, each of which is recoded into k levels has k^n legal states. An *episodical behavior* is a time history of legal states. The "episodical behavior" is the qualitative counterpart of the quantitative "trajectory behavior".

Inductive reasoning is a technique which was invented in the seventies by George Klir [13.5]. A first software system implementing Klir's ideas was SAPS [13.11]. Unfortunately, the original SAPS system was not sufficiently flexible to be of much practical use. We have therefore developed a new implementation, called SAPS–II, which is available as either a CTRL–C library or a MATLAB toolbox [13.4]. The CTRL–C version of SAPS–II can be accessed using the command:

```
[>  DO saps : saps
```

MATLAB recognizes all its toolboxes automatically.

In SAPS–II, levels are represented through positive integers. While QualSim is a very simple program package, SAPS–II is quite intricate, and I won't be able to discuss all the details of how SAPS has been implemented. However, this chapter can serve as a somewhat simplified user's guide.

The first question that we must ask ourselves is: *How* do we recode? How many levels should we select for each of our state variables? Where do we draw the borderline (i.e., where do we select the landmark) that separates two neighboring regions from each other?

Inductive reasoning is a completely inductive approach to modeling. It operates on a set of measured data points, and identifies a "model" from previously made observations.

From statistical considerations, we know that, in any class analysis, we would like to record each possible discrete state at least five times [13.7]. Thus, a relation exists between the possible number of legal states, and the number of data points that we require to base our modeling effort upon:

$$n_{rec} \geq 5 \cdot n_{leg} = 5 \cdot \prod_{\forall i} k_i \tag{13.1}$$

where n_{rec} denotes the total number of recordings, i.e., the total number of observed states, n_{leg} denotes the total number of different legal states, i is an index that loops over all variables, and k is an index that loops over all levels. If we postulate that each variables assumes the same number of levels, eq(13.1) can be simplified to:

$$n_{rec} \geq 5 \cdot (n_{lev})^{n_{var}} \tag{13.2}$$

where n_{var} denotes the number of variables, and n_{lev} denotes the chosen number of levels for each variable. The number of variables is usually given, and the number of recordings is frequently predetermined. In such a case, we can find the optimum number of levels from eq(13.3):

$$n_{lev} = ROUND(\sqrt[n_{var}]{\frac{n_{rec}}{5}}) \tag{13.3}$$

For reasons of symmetry, we often prefer an odd number of levels over an even number of levels. For example, the five levels *much too low*, *too low*, *normal*, *too high*, and *much too high* might denote states of the heart beat of a patient undergoing surgery. By choosing an

odd number of levels, we can group anomalous levels symmetrically around the normal state.

If the number of recordings is not predetermined, we might consider consulting with a human expert (in the above example, the surgeon) to determine a meaningful number of levels for a given variable.

The number of levels of our variables determines the expressiveness and the predictiveness of our qualitative model. The *expressiveness* of a qualitative model is a measure of the information content that the model provides. Later in this chapter, I shall present formulae describing the information content of a qualitative model. The *predictiveness* of a qualitative model is a measure of its forecasting power, i.e., it determines the length of time over which the model can be used to forecast the future behavior of the underlying system [13.8].

If all variables are recoded into exactly one level, the qualitative model exhibits only one legal state. It is called a "null model". It will be able to predict the future behavior of the underlying system perfectly over an infinite time span (within the framework of its model resolution). Yet, the prediction does not tell us anything useful. Thus, the null model is characterized by an infinitely high predictiveness, and by a zero expressiveness.

On the other hand, if we recode every variable into 1000 levels, the system exhibits myriads of legal states. The expressiveness (i.e., resolution) of such a model will be excellent. Each state contains a large amount of valuable information about the real system. Yet, the predictiveness of this model will be lousy, unless we possess an extremely large base of observed data. In all likelihood, this model cannot be used to predict the behavior of the real system for even a single time-step into the future.

Consequently, we must compromise. For most practical applications, we found that either three or five levels were about optimal [13.3,13.12].

Once we decided upon the number of levels for each variable, we must choose the landmarks that separate neighboring regions. Often, this is best done by consulting with a human expert. For example, we may ask a surgeon what s/he considers a *normal* heart beat during surgery, and when s/he would believe that the heart beat is definitely *too low* or *too high*, and when s/he would consider it to be *critically too low* or *critically too high*. If we are then able to predict the future behavior of the patient in terms of these qualitative variables,

we may be able to construct a heart monitor which will warn the surgeon ahead of time about a predictable problem. This clearly sounds like a worthwhile research topic.

However, if the amount of observed data is limited, it may be preferable to maximize the expressiveness of the qualitative model. This demand leads to a clearly defined optimal landmark selection algorithm [13.8]. The expressiveness of the model will be maximized if each level is observed equally often. Thus, one way to find an optimal set of landmarks, is to sort the observed trajectory values into ascending order, cut the sorted vector into n_{lev} segments of equal length, and choose the landmarks anywhere between the extreme values of neighboring segments. Let me demonstrate this process by means of an example. Fig.13.1 shows an observed trajectory of a continuous variable:

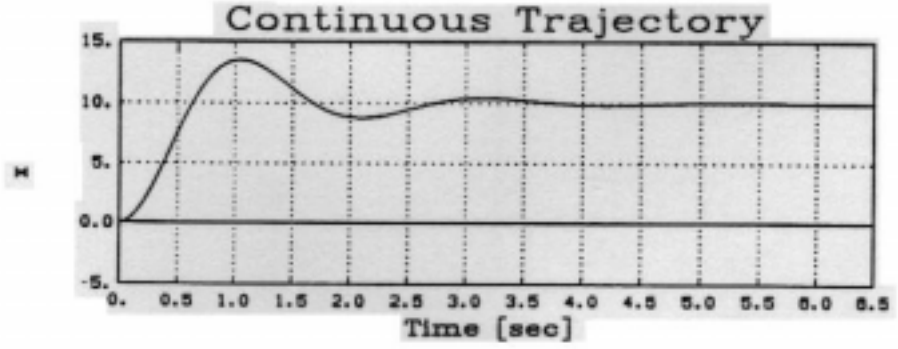

Figure 13.1. Trajectory behavior of a continuous variable

We first discretize the time axis (how this is done in an optimal manner, will be explained in due course). Let us say, that this process leads to a trajectory vector of length 131. The observed values range from 0.0 to 13.5. Let us assume that we wish to recode this trajectory into the three distinct levels '1', '2', and '3'. If we would simply cut the domain into equal intervals of length 4.5, i.e.:

$$\text{'1'} \longleftrightarrow [0.0, 4.5]$$
$$\text{'2'} \longleftrightarrow (4.5, 9.0]$$
$$\text{'3'} \longleftrightarrow (9.0, 13.5]$$

the levels '1' and '2' would only occur very briefly, and they would only occur during the initial phase of the episode. Thereafter, we

would constantly observe a level of '3'. Fig.13.2 shows the recoded episode.

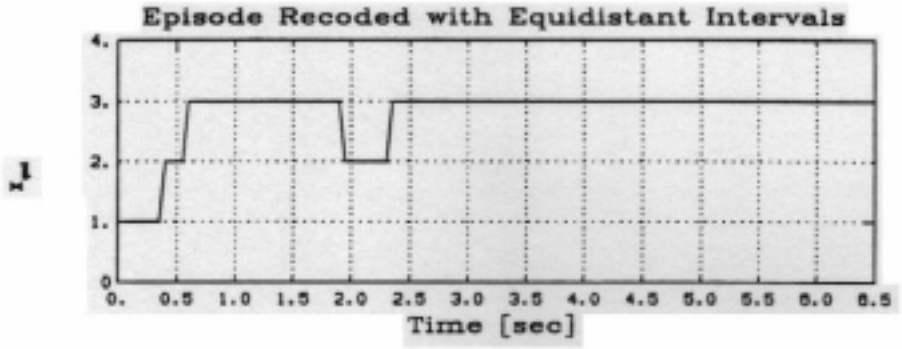

Figure 13.2. Episodical behavior of a recoded variable

In the process of recoding, we throw away a lot of information. In our example, we lost most of the information regarding the oscillation frequency. However, if we use the above described optimal algorithm, we sort the trajectory values in ascending order such that the first value is 0.0, and the last value is 13.5. We then cut the resulting vector of length 131 into three vectors of approximately equal length. The first vector contains the elements 1 to 43, the second vector contains the elements 44 to 86, and the third vector contains the elements 87 to 131. For the given example, the following values were found:

$$\begin{aligned} x_{sorted}(43) &= 9.8898 \\ x_{sorted}(44) &= 9.8969 \\ x_{sorted}(86) &= 10.0500 \\ x_{sorted}(87) &= 10.0501 \end{aligned}$$

and by using the arithmetic mean values of neighboring observed data points in different segments as our landmarks LM_i, we find:

$$\begin{aligned} LM_1 &= \frac{x_{sorted}(43) + x_{sorted}(44)}{2.0} = 9.8934 \\ LM_2 &= \frac{x_{sorted}(86) + x_{sorted}(87)}{2.0} = 10.0500 \end{aligned}$$

Using these landmarks, we obtain the following three regions:

'1' ⟷ [0.0, 9.8934]
'2' ⟷ (9.8934, 10.05]
'3' ⟷ (10.05, 13.5]

Fig.13.3 shows the same continuous trajectory as Fig.13.1 with the two new landmarks superimposed:

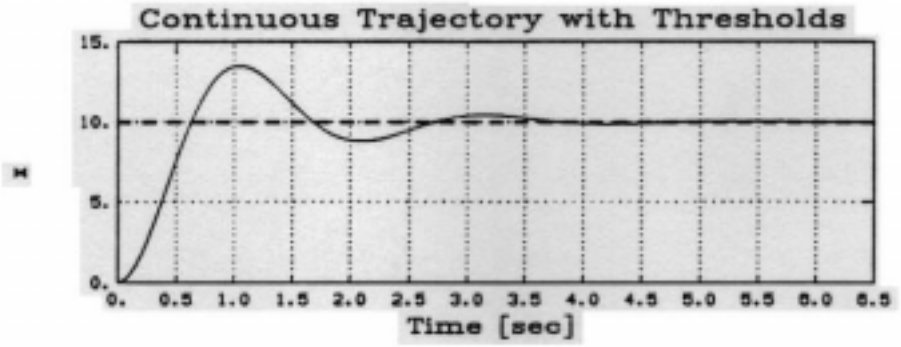

Figure 13.3. Continuous trajectory behavior with landmarks superimposed

The band width of level '2' is very narrow indeed. Fig.13.4 shows the recoded episodical behavior:

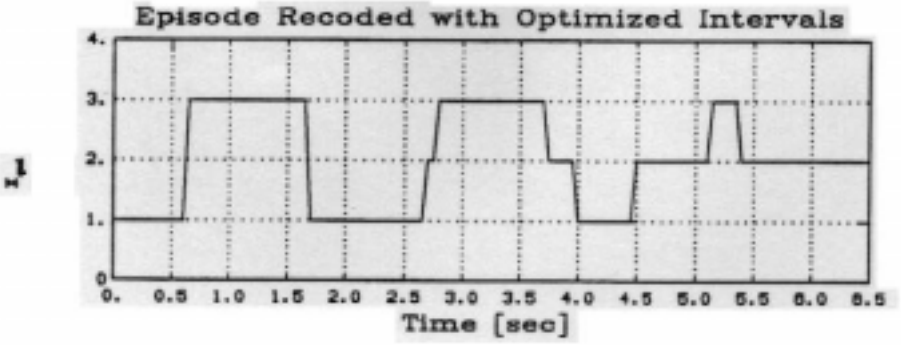

Figure 13.4. Episodical behavior of a recoded variable

Clearly, the recoding of Fig.13.4 has preserved more information about the real system than the recoding of Fig.13.2.

Which technique will work best depends heavily on the application area. For the case of the heart surgeon, the "optimized" recoding would be meaningless. His or her goal is to receive an early warning when the heart beat is expected to become critical, and not to observe each level equally often. Obviously, s/he wishes to observe level '3' (out of five levels) only, i.e., s/he wishes to keep her or his patient constantly within the *normal* range.

The following example, which was taken from Hugo Uyttenhove's Ph.D. dissertation [13.11], may serve as an illustration for the process of recoding. A heart monitor observes six different variables about a patient undergoing surgery. Each of these variables is being recoded into five different levels using the qualitative states:

'1' ⟷ *much too low*
'2' ⟷ *too low*
'3' ⟷ *normal*
'4' ⟷ *too high*
'5' ⟷ *much too high*

Table 13.1 lists the six variables together with their five ranges.

Table 13.1 Recoding of heart variables

variable	'1'	'2'	'3'	'4'	'5'
Systolic Blood Pr.	< 75	[75, 100)	[100, 160)	[160, 180]	> 180
Mean Blood Pr.	< 50	[50, 65)	[65, 100)	[100, 110]	> 110
Central Venous Pr.		< 4	[4, 20)	> 20	
Cardiac Output	< 2	[2, 3]	[3, 7]	> 7	
Heart Rate	< 50	[50, 60)	[60, 100)	[100, 110]	> 110
Left Atrial Pr.	< 1	[1, 4)	[4, 20)	> 20	

Let us assume that the trajectory behavior of this six variable system has been recorded in the form of a trajectory behavior matrix *meas* with six columns denoting the six different variables, and 1001 rows denoting different measurement instants (different time values). The following SAPS–II program segment can be used to recode these values:

```
[> DO saps:saps

[> from = [ 0.0   75.0  100.0  150.0  180.0
            75.0  100.0 150.0  180.0  999.9];
[> to = 1 : 5;
[> raw = RECODE(meas(:,1),'domain', from, to);

[> from = [ 0.0   50.0   65.0  100.0  110.0
            50.0  65.0  100.0  110.0  999.9];
[> r = RECODE(meas(:,2),'domain', from, to);
[> raw = [raw, r];

[> from = [ 0.0   4.0   20.0
            4.0   20.0  999.9];
[> to = 2 : 4;
[> r = RECODE(meas(:,3),'domain', from, to);
[> raw = [raw, r];

[> from = [ 0.0   2.0   3.0   7.0
            2.0   3.0   7.0   999.9];
[> to = 1 : 4;
[> r = RECODE(meas(:,4),'domain', from, to);
[> raw = [raw, r];

[> from = [ 0.0   50.0  60.0  100.0  110.0
            50.0  60.0  100.0 110.0  999.9];
[> to = 1 : 5;
[> r = RECODE(meas(:,5),'domain', from, to);
[> raw = [raw, r];

[> from = [ 0.0   1.0   4.0   20.0
            1.0   4.0   20.0  999.9];
[> to = 1 : 4;
[> r = RECODE(meas(:,6),'domain', from, to);
[> raw = [raw, r];
```

RECODE is one of the SAPS–II functions. As it is used in this example, it maps the regions (domains) specified in columns of the *from* matrix to the levels that are specified in the *to* vector. Thus, the *from* matrix and the *to* vector must have the same number of columns. In this example, each variable (column) is being recoded separately. The resulting episode *r* is then concatenated from the right to the previously found episodes which are stored in *raw*. The code should be fairly self–explanatory otherwise.

One problem remains to be discussed: How big is big? Obviously, qualitative terms are somewhat subjective. In comparison with an adult, a 10 year old has usually quite a different opinion about what

an "old person" is. The concept of landmarks is a treacherous one. Is it really true that a systolic blood pressure of 100.1 is "normal", whereas a systolic blood pressure of 99.9 is "too low"? Different physicians may have a different opinion altogether. My wife has usually a systolic blood pressure of about 90. Yet, her physician always smiles when he takes her blood pressure and predicts that she will have a long life. 90 is too low for what? What the surgeon probably meant when s/he declared 100 the borderline between "normal" and "low" was the following: If a "normal" patient, i.e. a patient with an average "normal" systolic blood pressure of 125 experiences a sudden drop of the blood pressure from 125 to 90 during surgery, then something is probably wrong. Does this mean that we should look at the time derivative of the systolic blood pressure in addition to the blood pressure itself? We probably should, but the surgeon couldn't tell us, because this is not the way s/he thinks. Most medical doctors aren't trained to think in terms of gradients and dynamic systems.

We have discovered two different problems.

(1) Asking an "expert" about which variables to look at can be a dubious undertaking. Experts in heart surgery aren't necessarily experts in expert system design. The surgeon won't understand the purpose of our question. Thus, we must use our own scrutiny and intelligence to interpret the answers of the so–called "expert" in an adequate way. The automation of the process of variable selection may be the toughest problem of all. We shall address this problem later in this chapter to some extent.

(2) Even if the question to the expert has been both properly formulated and properly understood, the determination of landmarks contains usually an element of subjectiveness. The crispness of a landmark may be deceiving. While a systolic blood pressure of 125 is clearly and undoubtedly a good and normal value, the matter becomes more confusing as we approach one of the neighboring "landmarks".

Lotfi Zadeh tackled the latter problem [13.13,13.14,13.15]. He introduced *fuzzy measures* as a technique to deal with the uncertainty of landmarks. Instead of saying that the systolic blood pressure is "normal" for values above 100, and "low" for values below 100, a fuzzy measure allows us to specify that, as we pass the value 100 in negative direction, the answer "normal" becomes less and less likely, while the answer "low" becomes more and more likely.

In a graphical form, we can depict the fuzzy measure as follows:

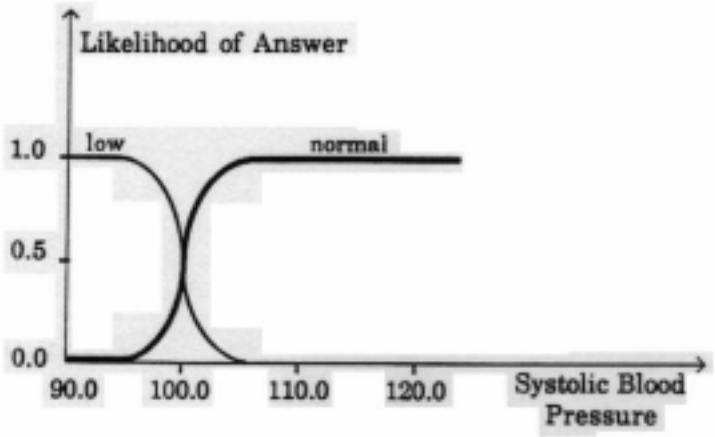

Figure 13.5. Fuzzy qualitative variable

The sigmoidal curves are called *membership functions*. How precisely these membership functions are shaped is up to the user. In SAPS–II, we have implemented only one type of membership function: a normal distribution which is 1.0 at the arithmetic mean value μ_i of any two neighboring "landmarks", and which is 0.5 at the landmarks themselves [13.8]. This membership function can be easily calculated using the equation:

$$Memb_i = \exp(-k_i \cdot (x - \mu_i)^2) \tag{13.4}$$

where x is the continuous variable which needs to be recoded, say the systolic blood pressure, and k_i is determined such that the membership function $Memb_i$ degrades to a value of 0.5 at the neighboring landmarks. Fig.13.6 shows the membership functions for the systolic blood pressure:

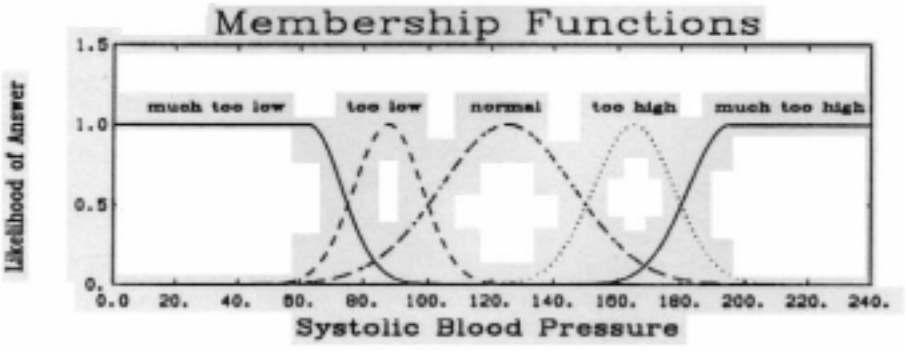

Figure 13.6. Membership functions of the systolic blood pressure

The first and the last membership functions are treated a little differently. Their shape (k_i value) is the same as for their immediate neighbors, and they are semi–open.

If we wish to compute the membership functions for the heart monitor example, we need to modify the previously shown program segment in the following way:

```
[> DO saps:saps

[> from = [ 0.0   75.0  100.0  150.0  180.0
            75.0  100.0 150.0  180.0  999.9];
[> to = 1 : 5;
[> [raw, Memb, side] = RECODE(meas(:,1),' fuzzy', from, to);

[> from = [ 0.0   50.0   65.0  100.0  110.0
            50.0  65.0  100.0  110.0  999.9];
[> [r, m, s] = RECODE(meas(:,2),' fuzzy', from, to);
[> raw = [raw, r];   Memb = [Memb, m];   side = [side, s];

[> from = [ 0.0   4.0   20.0
            4.0   20.0  999.9];
[> to = 2 : 4;
[> [r, m, s] = RECODE(meas(:,3),' fuzzy', from, to);
[> raw = [raw, r];   Memb = [Memb, m];   side = [side, s];

[> from = [ 0.0   2.0   3.0   7.0
            2.0   3.0   7.0   999.9];
[> to = 1 : 4;
[> [r, m, s] = RECODE(meas(:,4),' fuzzy', from, to);
[> raw = [raw, r];   Memb = [Memb, m];   side = [side, s];

[> from = [ 0.0   50.0   60.0  100.0  110.0
            50.0  60.0  100.0  110.0  999.9];
[> to = 1 : 5;
[> [r, m, s] = RECODE(meas(:,5),' fuzzy', from, to);
[> raw = [raw, r];   Memb = [Memb, m];   side = [side, s];

[> from = [ 0.0   1.0   4.0   20.0
            1.0   4.0   20.0  999.9];
[> to = 1 : 4;
[> [r, m, s] = RECODE(meas(:,6),' fuzzy', from, to);
[> raw = [raw, r];   Memb = [Memb, m];   side = [side, s];
```

The *raw* matrix will be exactly the same as before (since *RECODE* will always pick the most likely answer), but in addition, we obtain the fuzzy memberships of all our qualitative variables which are stored in the *Memb* matrix. The third matrix *side* contains a value of

0 whenever the measured data point coincides with the mean value of the neighboring landmarks, it assumes a value of -1 if the measured variable is smaller than the mean between the landmarks, and it is $+1$ if the measured variable is larger than the mean between the landmarks.

In the process of recoding, a large amount of valuable information about our real system is discarded. The fuzzy membership retains some of this information which will prove useful in due course. In fact, up to this point, no information has been lost at all. The original continuous signal can be *regenerated* accurately using the SAPS function:

```
[>  meas = REGENERATE(raw, Memb, side, from, to)
```

where the meaning of the *from* and *to* parameters is opposite from that before [13.8].

13.3 Input/Output Behavior and Masking

By now, we have recoded our trajectory behavior into a discrete episodical behavior. In SAPS–II, the episodical behavior is stored in a *raw data matrix*. Each column of the raw data matrix represents one of the observed variables, and each row of the raw data matrix represents one time point, i.e., one recording of all variables, i.e., one recorded state. The values of the raw data matrix are in the set of legal levels that the variables can assume, i.e., they are all positive integers, usually in the range from '1' to '5'.

How does the episodical behavior help us identify a model of our system for the purpose of forecasting the future behavior of the system for any given input stream? Any model describes relationships between variables. That is its purpose. For example, in a state–space model, we describe the relationships between the state variables x_i and their time derivatives $\dot{x}_i$:

$$\dot{x}_i = f_i(x_1, x_2, \ldots, x_n) \tag{13.5}$$

If the state variables x_i have been recoded into the qualitative variables v_i, and their time derivatives have been recoded into the qualitative variables w_i, we can write:

$$w_i = \bar{f}_i(v_1, v_2, \ldots, v_n) \tag{13.6}$$

While $\tilde{f}_i$ will be a different function than f_i, the fact that (deterministic) relationships exist between the x_i and the $\dot{x}_i$ variables, can be partially preserved in the process of recoding. The $\tilde{f}_i$ functions are (possibly deterministic) relationships between the v_i and the w_i variables.

The beauty of this transformation becomes evident when we try to identify these functional relationships. While the identification (characterization) of the f_i functions is a difficult task, the identification of the $\tilde{f}_i$ functions is straightforward. Since each of the v_i variables can assume only a finite set of values, we can characterize the $\tilde{f}_i$ functions through enumeration. Let me provide an example:

$$y = \sin(x) \tag{13.7}$$

is a quantitative relationship between two quantitative variables x and y. Let us recode the variable x into a qualitative variable v with four states, such that:

$$\begin{matrix} & x & v \\ \left(\begin{matrix} 1^{st} \\ 2^{nd} \\ 3^{rd} \\ 4^{th} \end{matrix} \right. & \begin{matrix} quadrant \\ quadrant \\ quadrant \\ quadrant \end{matrix} & \left. \begin{matrix} \text{'1'} \\ \text{'2'} \\ \text{'3'} \\ \text{'4'} \end{matrix} \right) \end{matrix} \tag{13.8}$$

If the angle x is anywhere between $0°$ and $90°$ plus or minus a multiple of $360°$, the qualitative variable v assumes a value of '1', etc. v simply denotes the quadrant of x. Let us recode the variable y into a qualitative variable w with two states, such that:

$$\begin{matrix} & y & w \\ \left(\begin{matrix} negative \\ positive \end{matrix} \right. & & \left. \begin{matrix} \text{'1'} \\ \text{'2'} \end{matrix} \right) \end{matrix} \tag{13.9}$$

Thus, w simply denotes the sign of y. This allows us to characterize the functional relationship between the two qualitative variables v and w as follows:

$$\begin{matrix} & v & w \\ \left(\begin{matrix} \\ \\ \\ \end{matrix} \right. & \begin{matrix} \text{'1'} \\ \text{'2'} \\ \text{'3'} \\ \text{'4'} \end{matrix} & \left. \begin{matrix} \text{'1'} \\ \text{'1'} \\ \text{'2'} \\ \text{'2'} \end{matrix} \right) \end{matrix} \tag{13.10}$$

Eq(13.10) is the qualitative counterpart of eq(13.7). Qualitative functions are *finite automata* which relate the qualitative variables to each other.

However, we have to be careful that we recode all variables in a consistent fashion. For example, if x had been recoded differently:

$$\begin{array}{cc} x & v \\ \begin{pmatrix} -45^\circ .. +45^\circ & \text{'1'} \\ +45^\circ .. +135^\circ & \text{'2'} \\ +135^\circ .. +225^\circ & \text{'3'} \\ +225^\circ .. +315^\circ & \text{'4'} \end{pmatrix} \end{array} \tag{13.11}$$

with the same recoding for y, we would have obtained a non-deterministic relationship between v and w:

$$\begin{array}{ccc} v & w & prob \\ \begin{pmatrix} \text{'1'} & \text{'1'} & 50\% \\ \text{'1'} & \text{'2'} & 50\% \\ \text{'2'} & \text{'1'} & 100\% \\ \text{'3'} & \text{'1'} & 50\% \\ \text{'3'} & \text{'2'} & 50\% \\ \text{'4'} & \text{'2'} & 100\% \end{pmatrix} \end{array} \tag{13.12}$$

The third column denotes the *relative frequency of observation* which can be interpreted as the *conditional probability* of the output w to assume a certain value, given that the input v has already assumed an observed value.

Eq(13.12) can be rewritten in a slightly different form:

$$\begin{array}{cc} v \backslash w & \begin{matrix} \text{'1'} & \text{'2'} \end{matrix} \\ \begin{matrix} \text{'1'} \\ \text{'2'} \\ \text{'3'} \\ \text{'4'} \end{matrix} & \begin{pmatrix} 0.5 & 0.5 \\ 1.0 & 0.0 \\ 0.5 & 0.5 \\ 0.0 & 1.0 \end{pmatrix} \end{array} \tag{13.13}$$

which is called a *state transition matrix* relating v to w. The values stored in the state transition matrix are the transition probabilities between a given level of v and and certain level of w, i.e., they are the conditional probabilities of w given v:

$$ST_{i,j} = p\{w = \text{'}j\text{'} | v = \text{'}i\text{'}\} \tag{13.14}$$

The element $< i, j >$ of the state transition matrix is the conditional probability of the variable w to become 'j', assuming that v has a value of 'i'.

We don't want to fall into the same trap as in Chapter 12 where we assumed that we already knew the state-space model of our system. The technique which is advocated in Chapter 13 is *totally inductive*. Consequently, we are not going to assume that we know anything

about our system with the exception of the observed data streams, i.e., the measured trajectory behavior. Obviously, this says that we cannot know *a priori* what it means to recode our variables "consistently". All we know is the following: For any system which can be described by a deterministic (yet unknown) arbitrarily non–linear state–space model of arbitrary order, if we are lucky enough to pick all state variables and all state derivatives as our output variables, i.e., if all these variables are included in our trajectory behavior, and if we are fortunate enough to recode all these variables in a consistent fashion, then deterministic and static relationships will exist between the qualitative state variables and the qualitative state derivatives.

In the process of modeling, we wish to find finite automata relations between our recoded variables which are *as deterministic as possible*. If we find such a relationship for every output variable, we can forecast the behavior of our system by iterating through the state transition matrices. The more deterministic the state transition matrices are, the better the certainty that we will predict the future behavior correctly.

Let us now look at the development of our system over time. For the moment, I shall assume that our observed trajectory behavior was produced by quantitatively simulating a state–space model over time. Let us assume that the state–space model was coded in ACSL, that we integrate it using a fixed step forward Euler algorithm, and that we log every time step in our trajectory behavior. In this case, we can write:

$$x_i(k+1) = x_i(k) + \Delta t \cdot \dot{x}_i(k) \tag{13.15}$$

The qualitative version of eq(13.15) is:

$$v_i(k+1) = \tilde{g}(v_i(k), w_i(k)) \tag{13.16}$$

Eq(13.15) is obviously a deterministic relationship between $x_i(k)$, $\dot{x}_i(k)$, and $x_i(k+1)$. Is $\tilde{g}$ a deterministic function? Let me assume that we choose our step size Δt very small in order to integrate accurately. In this case, the state variables will change very little from one step to the next. This means that, after the recoding, $v_i(k+1)$ is almost always equal to $v_i(k)$. Only when x_i passes through a landmark will $v_i(k+1)$ be different from $v_i(k)$. Consequently, our qualitative model will have a tough time predicting when the landmark crossing will take place. Thus, it is not a good idea to include every tiny time step in our trajectory behavior. The time distance between two logged entries of our trajectory behavior δt

should be chosen such that the two terms in eq(13.15) are of the same order of magnitude, i.e.:

$$\|x_i\| \approx \delta t \cdot \|\dot{x}_i\| \tag{13.17}$$

Notice that δt is the *communication interval*, whereas Δt denotes the *integration step size*. These two variables have no direct relationship with each other. The data stream could as well be the output of a digital oscilloscope observing a real physical system. In such a case, Δt has no meaning, but δt still exists, and must be chosen carefully.

Let us now forget about state–space models. All we know is that we have a recorded continuous trajectory behavior available for modeling. We want to assume furthermore that we know which are the inputs into the real system, and which are the outputs that we measure. Our trajectory behavior can thus be separated into a set of input trajectories u_i concatenated from the right with a set of output trajectories y_i, for example:

$$\begin{array}{l|ccccc} \textit{time} & u_1 & u_2 & y_1 & y_2 & y_3 \\ 0.0 & \cdots & \cdots & \cdots & \cdots & \cdots \\ \delta t & \cdots & \cdots & \cdots & \cdots & \cdots \\ 2\cdot\delta t & \cdots & \cdots & \cdots & \cdots & \cdots \\ 3\cdot\delta t & \cdots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (n_{res}-1)\cdot\delta t & \cdots & \cdots & \cdots & \cdots & \cdots \end{array} \tag{13.18}$$

The trajectory behavior is recoded into an episodical behavior using the techniques described in the last section. Our modeling effort now consists in finding finite automata relations between the recoded variables which make the resulting state transition matrices as deterministic as possible. Such a relation could look like:

$$y_1(t) = \tilde{f}(y_3(t-2\delta t), u_2(t-\delta t), y_1(t-\delta t), u_1(t)) \tag{13.19}$$

Eq(13.19) can be represented as follows:

$$\begin{array}{c|ccccc} i\backslash^{o} & u_1 & u_2 & y_1 & y_2 & y_3 \\ t-2\delta t & 0 & 0 & 0 & 0 & -1 \\ t-\delta t & 0 & -2 & -3 & 0 & 0 \\ t & -4 & 0 & +1 & 0 & 0 \end{array} \tag{13.20}$$

The negative elements in the above matrix denote inputs of our qualitative functional relationship. The above example has four inputs. The sequence in which they are enumerated is immaterial. I usually enumerate them from the left to the right, and from the top to

the bottom. The positive value is the output. Thus, eq(13.20) is a matrix representation of eq(13.19). In inductive reasoning, such a representation is called a *mask*. A mask denotes a dynamic relationship between qualitative variables. In SAPS–II, masks are written as either MATLAB or CTRL–C matrices. A mask has the same number of columns as the episodical behavior to which it should be applied, and it has a certain number of rows. The number of rows of the mask matrix is called the *depth* of the mask. The mask can be used to flatten a dynamic relationship out into a static relationship. We can shift the mask over the episodical behavior, pick out the selected inputs and outputs, and write them together in one row. Fig.13.7 illustrates this process.

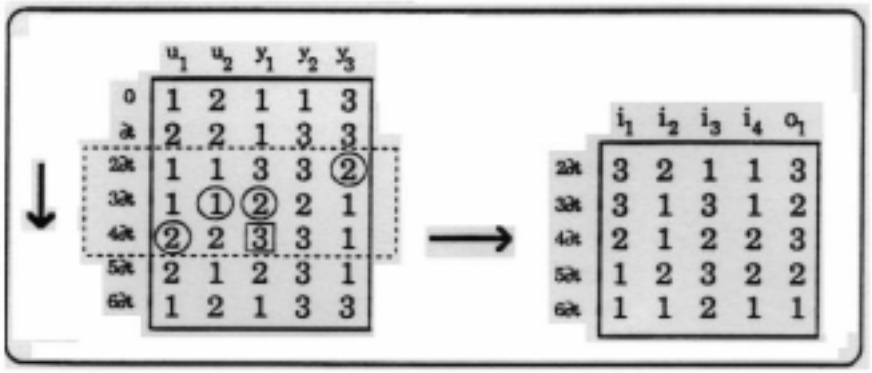

Figure 13.7. Flattening dynamic relationships through masking

After the mask has been applied to the raw data, the formerly dynamic episodical behavior has become static, i.e., the relationships are now contained within single rows. in SAPS–II, this operation can be performed using the *IOMODEL* function:

```
[>  io = IOMODEL(raw, mask)
```

IOMODEL will translate the raw data matrix on the left side of Fig.13.7 into the flattened data matrix on the right side of Fig.13.7.

We still haven't discussed how δt is picked in practice. Experience has shown that the equality of eq(13.17) can be translated into the following general rule: The mask should cover the largest time constant that we wish to capture in our model. If the trajectory behavior stems from measurement data, we should measure a Bode diagram of the system that we wish to model. This allows us to

determine the band width ω_{3dB} of the system. The largest time constant (i.e., the settling time) of the system can be computed from eq(13.21):

$$t_s \approx \frac{2\pi}{\omega_{3dB}} \tag{13.21}$$

If our chosen mask depth is 3, the mask spans a time interval of $2\delta t$, thus:

$$\delta t = \frac{t_s}{2} \tag{13.22}$$

The mask depth should be chosen as the ratio between the largest and the smallest time constant that we wish to capture in our model, but this ratio should not be larger than 3 or 4. Otherwise, the inductive reasoner won't work very well since the computing effort grows exponentially with the size of the mask.

13.4 Inductive Modeling and Optimal Masks

An inductive reasoning model is simply a set of masks that relate the input variables and previous values of the output variables to the current values of the outputs. We shall usually forbid relations between various outputs at the current time, since if:

$$y_1(t) = \tilde{f}_1(y_2(t)) \tag{13.23a}$$
$$y_2(t) = \tilde{f}_2(y_1(t)) \tag{13.23b}$$

we have an *algebraic loop.*

The question remains: How do we find the appropriate masks. The answer to this question was already given. We need to find the masks that, within the framework of the allowable masks, present us with the most deterministic state transition matrix since this matrix will optimize the predictiveness of our model. In SAPS–II, we have introduced the concept of a *mask candidate matrix.* A mask candidate matrix is an ensemble of all possible masks from which we choose the best one by a mechanism of *exhaustive search.* The mask candidate matrix contains -1 elements where the mask has a *potential input*, it contains a $+1$ element, where the mask has its output, and it contains 0 elements to denote forbidden connections. Thus,

the mask candidate matrix for our previous five variable example will be:

$$\begin{array}{c|ccccc} t\backslash^{\sigma} & u_1 & u_2 & y_1 & y_2 & y_3 \\ t-2\delta t & -1 & -1 & -1 & -1 & -1 \\ t-\delta t & -1 & -1 & -1 & -1 & -1 \\ t & -1 & -1 & +1 & 0 & 0 \end{array} \qquad (13.24)$$

The SAPS–II program segment:

```
[> mcan = -ONES(3,5);
[> mcan(3,3:5) = [1,0,0];
[> maxcompl = 5;
[> mask = OPTMASK(raw,mcan,maxcompl)
```

determines the optimal mask from the set of candidate masks. *raw* is the raw data matrix, and *mcan* is the mask candidate matrix. *OPTMASK* will go through all possible masks of complexity two, i.e., all masks with one input, and find the best. It will then proceed and try all masks of complexity three, i.e., all masks with two inputs, and find the best of those, etc. The third parameter *maxcompl* enables us to limit the maximum complexity, i.e., the largest number of non–zero elements that the mask may contain. This is a useful feature. In all practical examples, the quality of the masks will first grow with increasing complexity, then reach a maximum, and then decay rapidly. Thus, by setting *maxcompl*, we can reduce the time that the optimization takes. A good value for *maxcompl* is usually five. In order to disable this feature, *maxcompl* can be set to zero.

How do we determine the quality of a mask? Let me explain the process by means of a simple example. Let us assume that we have found the following raw data matrix (episodical trajectory) of a three variable system:

```
raw = [ 2 1 1
        1 2 3
        1 1 4
        1 1 3
        1 1 2
        2 2 2
        2 1 2
        2 2 3
        1 2 1
        1 2 1
        2 1 4 ]
```

Each mask will lead to a different state transition matrix. For example, the mask:

$$mask = \begin{bmatrix} -1 & -2 & 0 \\ 0 & 0 & +1 \end{bmatrix}$$

will lead to the following input/output model:

$$io = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 4 \\ 1 & 1 & 3 \\ 1 & 1 & 2 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \\ 2 & 2 & 1 \\ 1 & 2 & 1 \\ 1 & 2 & 4 \end{bmatrix}$$

The *basic behavior* of this input/output matrix is a lexical listing of all observed states together with their observation frequencies:

$$b = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 3 \\ 1 & 2 & 1 \\ 1 & 2 & 4 \\ 2 & 1 & 3 \\ 2 & 2 & 1 \\ 2 & 2 & 2 \end{bmatrix} \qquad p = \begin{bmatrix} 0.2 \\ 0.1 \\ 0.1 \\ 0.2 \\ 0.2 \\ 0.1 \\ 0.1 \end{bmatrix}$$

In SAPS–II, the basic behavior can be computed using the statement:

```
[>   [b,p] = BEHAVIOR(io)
```

This gives rise to the following state transition matrix:

$in \backslash out$	'1'	'2'	'3'	'4'
'11'	0.000	0.667	0.333	0.000
'12'	0.333	0.000	0.000	0.667
'21'	0.000	0.000	1.000	0.000
'22'	0.500	0.500	0.000	0.000

which shows on the left side the combined values of the two inputs, on the top row the values of the output, and in the table itself, the conditional probabilities. In out example, the two inputs are binary variables, whereas the single output has four levels. In addition, we need the absolute probabilities (observation frequencies) of the input states. For our example, the following values are found:

$$\begin{array}{cc} input & prob \\ \text{'11'} & \left(\begin{array}{c} 0.3 \\ 0.3 \\ 0.2 \\ 0.2 \end{array} \right) \\ \text{'12'} & \\ \text{'21'} & \\ \text{'22'} & \end{array}$$

In SAPS–II, the state transition matrix and the input probability vector can be computed using the statement:

```
[>  [st, ip] = STMATRIX(io, 2)
```

where the second input argument denotes the number of input variables of the input/output matrix. In our example, the input/output matrix contains two inputs and one output.

Now, we can compute the Shannon entropy [13.10] of the state transition matrix which is a measure of the information content of the state transition matrix. The Shannon entropy is computed with the following formula:

$$HM = -\sum_{\forall i}(p\{inp = \text{'}i\text{'}\} \cdot \sum_{\forall j}(p\{out = \text{'}j\text{'}|inp = \text{'}i\text{'}\} \cdot \log_2(p\{out = \text{'}j\text{'}|inp = \text{'}i\text{'}\}))) \tag{13.25}$$

In our example:

$$\begin{aligned} -HM &= 0.3 \cdot [0.667 \cdot \log_2(0.667) + 0.333 \cdot \log_2(0.333)] \\ &+ 0.3 \cdot [0.667 \cdot \log_2(0.667) + 0.333 \cdot \log_2(0.333)] \\ &+ 0.2 \cdot [1.0 \cdot \log_2(1.0)] \\ &+ 0.2 \cdot [0.5 \cdot \log_2(0.5) + 0.5 \cdot \log_2(0.5)] \\ &= -0.275 - 0.275 + 0.0 - 0.2 \\ &= -0.75 \end{aligned}$$

and thus:

$$HM = 0.75$$

The state transition matrix is completely deterministic if it contains one +1 element in every row, while all other elements are 0. In that case, the Shannon entropy is:

$$HM_{min} = 0.0$$

The worst case occurs if all outcomes are equally probably, i.e., if the state transition matrix contains only elements of the same magnitude, in our case: 0.25 (since the output has four levels). For this case, we find the following Shannon entropy:

$$HM_{max} = 2.0$$

The maximum entropy depends on the number of rows and columns of the state transition matrix. We can introduce an *uncertainty reduction measure* which is defined as follows:

$$HR = 1.0 - \frac{HM}{HM_{max}} \tag{13.26}$$

In SAPS–II, the Shannon entropy and the uncertainty reduction measure of a state transition matrix can be determined using the statement:

```
[> [HM,HR] = ENTROPY(st,ip)
```

HR can be used as a *quality measure*. In the worst case, *HR* is equal to 0.0, while in the best case, *HR* is equal to 1.0.

However, a problem remains with this approach. If we increase the complexity of the mask, we find that the state transition matrix becomes more and more deterministic. With growing mask complexity, more and more possible input states (combinations of levels of the various input variables) exist. Since the total number of observations n_{rec} remains constant, the observation frequencies of the observed states will become smaller and smaller. Very soon, we shall be confronted with the situation where every state that has ever been observed has been observed precisely once. This leads obviously to a completely deterministic state transition matrix. Yet, the predictiveness of the model may still be very poor, since already the next predicted state has probably never before been observed, and that will be the end of our forecasting. Therefore, we must include this consideration in our quality measure.

I had mentioned earlier that, from a statistical point of view, we would like to make sure that every state is observed at least five times [13.7]. Therefore, we introduce an *observation ratio* [13.8]:

$$OR = \frac{5 \cdot n_{5\times} + 4 \cdot n_{4\times} + 3 \cdot n_{3\times} + 2 \cdot n_{2\times} + n_{1\times}}{5 \cdot n_{leg}} \tag{13.27}$$

where:

n_{leg}	= number of legal input states
$n_{1\times}$	= number of input states observed only once
$n_{2\times}$	= number of input states observed twice
$n_{3\times}$	= number of input states observed thrice
$n_{4\times}$	= number of input states observed four times
$n_{5\times}$	= number of input states observed five times or more

If every legal input state has been observed at least five times, OR is equal to 1.0. If no input state has been observed at all (no data), OR is equal to 0.0. Thus, also OR qualifies for a quality measure.

We define the *quality of a mask* as the product of its uncertainty reduction measure and its observation ratio:

$$Q = HR \cdot OR \tag{13.28}$$

The *optimal mask* is the mask with the largest Q value.

The *OPTMASK* function can be used to compute all these quantities. The full syntax of this function is as follows:

```
[> [mask,HM,HR,Q,mhis]=OPTMASK(raw,mean,maxcompl)
```

mask is the optimal mask found in the optimization. *HM* is a row vector that contains the Shannon entropies of the best masks for every considered complexity. *HR* is a row vector that contains the corresponding uncertainty reduction measures. Q is a row vector which contains the corresponding quality measures, and *mhis* is the mask history matrix. The mask history matrix contains, concatenated to each other from the right, the best masks at each of the considered complexities. One of these masks is the optimal mask which, for reasons of convenience, is also returned separately.

Until now, we haven't used our fuzzy membership functions yet. Remember that the fuzzy membership associated with the value of a qualitative variable is a *measure of confidence*. It specifies how confident we are that the assigned value is correct. If we compute the input/output matrix, we can assign a *confidence* to each row. The confidence of a row of the input/output matrix is the *joint membership* of all the variables which are associated with that row [13.8].

Let me demonstrate the concept by means of our simple three variable example. Assume that the following fuzzy membership matrix accompanies our raw data matrix:

$$raw = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 4 \\ 1 & 1 & 3 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 3 \\ 1 & 2 & 1 \\ 1 & 2 & 1 \\ 2 & 1 & 4 \end{bmatrix} \qquad Memb = \begin{bmatrix} 0.61 & 1.00 & 0.83 \\ 0.73 & 0.77 & 0.95 \\ 0.73 & 0.88 & 1.00 \\ 0.51 & 0.91 & 1.00 \\ 0.55 & 0.92 & 0.92 \\ 0.71 & 0.77 & 0.78 \\ 0.63 & 0.91 & 0.69 \\ 0.86 & 0.83 & 0.83 \\ 0.77 & 0.97 & 0.70 \\ 0.78 & 0.93 & 0.75 \\ 0.89 & 0.81 & 1.00 \end{bmatrix}$$

The joint membership of i membership functions is defined as the smallest individual membership:

$$Memb_{joint} = \bigcap_{\forall i} Memb_i = \inf_{\forall i}(Memb_i) \stackrel{\text{def}}{=} \min_{\forall i}(Memb_i) \tag{13.29}$$

SAPS-II's *FIOMODEL* function computes the input/output matrix together with the confidence vector:

$$[io, conf] = FIOMODEL(raw, mask)$$

Applied to our raw data matrix and using the same mask as before:

$$mask = \begin{bmatrix} -1 & -2 & 0 \\ 0 & 0 & +1 \end{bmatrix}$$

we find:

$$io = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 4 \\ 1 & 1 & 3 \\ 1 & 1 & 2 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \\ 2 & 2 & 1 \\ 1 & 2 & 1 \\ 1 & 2 & 4 \end{bmatrix} \qquad conf = \begin{bmatrix} 0.61 \\ 0.73 \\ 0.73 \\ 0.51 \\ 0.55 \\ 0.69 \\ 0.63 \\ 0.70 \\ 0.75 \\ 0.78 \end{bmatrix}$$

The *conf* vector indicates how much confidence we have in the individual rows of our input/output matrix. We can now compute the basic behavior of the input/output model. Rather than counting the observation frequencies, we shall accumulate the confidences. If a state has been observed more than once, we gain more and more

confidence in it. Thus, we sum up the individual confidences. In SAPS–II, this can be achieved using the statement:

```
[> [b,c] = FBEHAVIOR(io,conf)
```

Applied to our simple example, we find:

$$b = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 3 \\ 1 & 2 & 1 \\ 1 & 2 & 4 \\ 2 & 1 & 3 \\ 2 & 2 & 1 \\ 2 & 2 & 2 \end{bmatrix} \qquad c = \begin{bmatrix} 1.06 \\ 0.73 \\ 0.75 \\ 1.51 \\ 1.24 \\ 0.70 \\ 0.69 \end{bmatrix}$$

Notice that the c vector is no longer a probability. The c_i elements no longer add up to 1.0.

This leads now to a modified state transition matrix. The SAPS–II statement:

```
[> [st,ic] = FSTMATRIX(io,conf,2)
```

produces the following results:

$in \backslash out$	'1'	'2'	'3'	'4'
'11'	0.00	1.06	0.73	0.00
'12'	0.75	0.00	0.00	1.51
'21'	0.00	0.00	1.24	0.00
'22'	0.70	0.69	0.00	0.00

which shows on the left side the combined values of the two inputs, in the top row the values of the output, and in the table itself, the confidence values. The *total input confidence* vector is:

input	*conf*
'11'	1.79
'12'	2.26
'21'	1.24
'22'	1.39

The total input confidences are computed by summing up the individual confidences of all occurrences of the same input in the basic behavior. Notice that in all these computations, the actual qualitative variables are exactly the same as before, only their assessment has changed. The previously used *probability measure* has been replaced by a *fuzzy measure.*

The optimal mask analysis can use the fuzzy measure as well. The statement:

```
[> [mask, HM, HR, Q, mhis] = FOPTMASK(raw, Memb, mean, maxcompl)
```

uses the fuzzy measure to evaluate the optimal masks. In order to be able to still use the Shannon entropy, we normalize the row sums of the state transition matrices to 1.0. It can happen that *FOPTMASK* picks another mask as its optimal mask than the previously used *OPTMASK* routine. Since we use more information about the real system, we shall obtain a higher mask quality in most cases. Notice that the concept of applying the Shannon entropy to a confidence measure is somewhat dubious on theoretical grounds since the Shannon entropy was derived in the context of probabilistic measures only. For this reason, some scientists prefer to replace the Shannon entropy by other types of performance indices [13.6,13.9] which have been derived in the context of the particular measure chosen. However, from a practical point of view, numerous simulation experiments performed by my students and me have shown the Shannon entropy to also work satisfactorily in this context.

13.5 Forecasting Behavior

Once we have determined the optimal mask, we can compute the input/output model resulting from applying the optimal mask to the raw data, and we can compute the corresponding state transition matrix. Now, we are ready to forecast the future behavior of our system, i.e., we are ready to perform a *qualitative simulation.*

Forecasting is a straightforward procedure. We simply loop over input states in our input/output model, and forecast new output states by reading out from the state transition matrix the most probable output given the current input. Let me explain this procedure by means of the previously used example. Given the raw data matrix:

```
raw = [  2 1 1
         1 2 3
         1 1 4
         1 1 3
         1 1 2
         2 2 2
         2 1 2
         2 2 3
         1 2 1
         1 2 1
         2 1 4 ]
```

which is assumed to consist of two input variables and one output variable. The future inputs over the next four steps are:

```
inp = [  1 1
         1 1
         2 2
         2 1 ]
```

and we wish to forecast the output vector over the same four steps. Let me assume furthermore that the optimal mask is the one used earlier:

```
mask = [ -1  -2   0
          0   0  +1 ]
```

For this case, we have already computed the input/output model and the state transition matrix. After the input/output model has been computed, the mask covers the final two rows of the raw data matrix. In order to predict the next output, we simply shift the mask one row further down. The next input set is thus: '2 1'. From the state transition matrix, we find that this input leads in all cases to the output '3'. Thus, we copy '3' into the data matrix at the place of the next output. We then shift the mask one row further down. At this time, the mask reads the input set '1 1'. From the state transition matrix, we find that the most probable output is '2', but its probability is only 66.7%. We continue in the same manner. The next input set is again '1 1'. Since this input set is assumed to be *statistically independent* of the previous one (an unreasonable but commonly made assumption), the joint probability is the product of the previous cumulative probability with the newly found probability, thus $p = \frac{2}{3} \cdot \frac{2}{3} = \frac{4}{9} = 44.4\%$. The next input set is '22'. For this case, we find that the outcomes '1' and '2' are equally likely (50%). Thus, we pick arbitrarily one of those. The cumulative output probability has meanwhile decreased to 22.2%.

This is exactly how, in SAPS–II, the FORECAST routine predicts future states of a recoded system.

$$[> \quad [f2, p] = FORECAST(f1, mask, nrec, minprob)$$

forecasts the future behavior of a given system $f1$ where $f1$ contains the raw data model concatenated from below with the future inputs filled from the right with arbitrary zero values, thus:

$$[> \quad f1 = [raw; inp, ZROW(nstp, nout)]$$

where *nstp* denotes the number of steps to be forecast, and *nout* denotes the number of output variables in the raw data model. *mask* is the optimal mask to be used in the forecasting, *nrec* denotes the number of recorded past data values, i.e., *nrec* tells the forecasting routine how many of the rows of $f1$ belong to the past, and how many belong to the future, and *minprob* instructs SAPS to terminate the forecasting process if the cumulative probability decreases below a given value. This feature can be disabled by setting *minprob* to zero.

Upon return, $f2$ contains the same information as $f1$ but augmented by the forecast outputs, i.e., some or all of the ZROW values have been replaced by forecasts. p is a column vector containing the cumulative probabilities. Of course, up to row *nrec*, the probabilities are all 1.0 since these rows contain past, i.e. factual, information.

If, during the forecasting process, an input state is encountered which has never before been recorded, the forecasting process comes to a halt. It is then the user's responsibility to either collect more data, reduce the number of levels, or pick an arbitrary output and continue with the forecasting.

How can we make use of the fuzzy memberships in the forecasting process? The procedure is very similar to our previous approach. However, in this case, we don't pick the output with the highest confidence. Instead, we compare the membership and side functions of the new input with the membership and side functions of all previous recordings of the same input, and pick as the output the one that belongs to the previously recorded input with the most similar membership [13.8]. For this purpose, we compute a cheap approximation of the regenerated continuous signal:

$$d = 1 + side * (1 - Memb) \tag{13.30}$$

for every input variable of the new input set, and store the regenerated d_i values in a vector. We then repeat this reconstruction for all

previous recordings of the same input set. We finally compute the L_2–norms of the difference between the d vector of the new input and the d vectors of all previous recordings of the same input, and pick the one with the smallest L_2–norm. Its *output* and *side* values are then used as forecasts for the *output* and *side* values of the current state. We proceed a little differently with the membership values. Here, we take the five previous recordings with the smallest L_2–norms, and compute a distance weighted average as the forecast for the fuzzy membership values of the current state.

In SAPS–II, this is accomplished by use of the function:

```
[> [f2, Memb2, side2] = FFORECAST(f1, Memb1, side1, mask, nrec)
```

The fuzzy forecasting function will usually give us a more accurate forecast than the probabilistic forecasting function. Also, if we use fuzzy forecasting, we can retrieve pseudo–continuous output signals with a relatively high quality using the REGENERATE function.

13.6 A Linear System – An Example

Let us once more analyze the same example that we discussed in Chapter 12:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\cdot\mathbf{x}+\mathbf{b}\cdot u \\ &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -3 & -4 \end{pmatrix}\cdot\mathbf{x}+\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\cdot u \end{aligned} \qquad (13.31a)$$

$$\begin{aligned} y &= \mathbf{C}\cdot\mathbf{x}+\mathbf{d}\cdot u \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\cdot\mathbf{x}+\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}\cdot u \end{aligned} \qquad (13.31b)$$

This time, we shall use fixed values for the parameters, and we shall use the entire state vector as output. Fig.13.8 shows the three Bode diagrams of this multivariable system superposed onto a single graph.

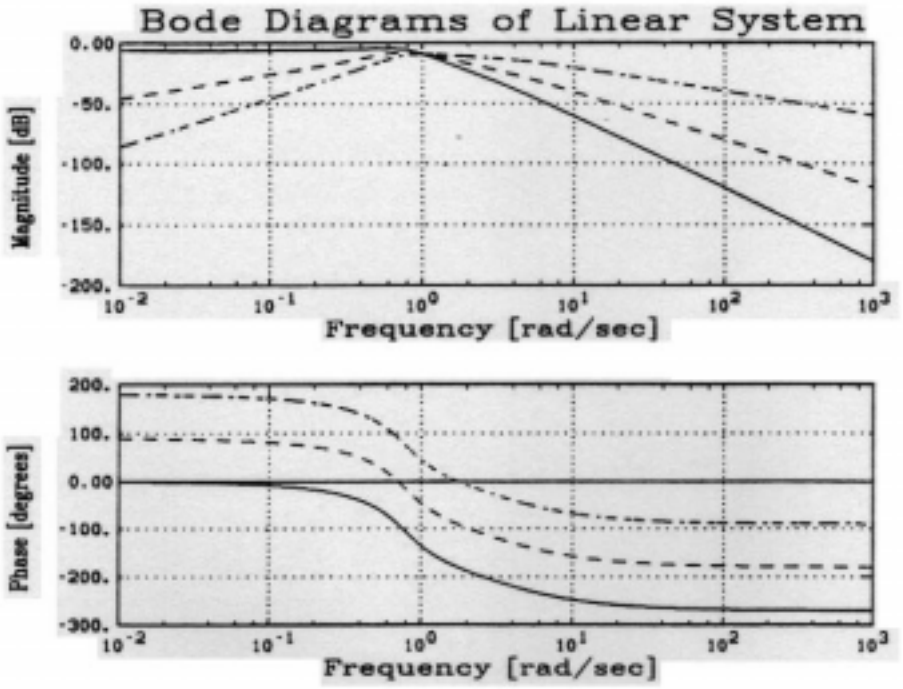

Figure 13.8. Bode diagrams of linear system

In this example, I computed the Bode diagrams directly in CTRL–C. However, for any stable physical system, we can measure the Bode diagram, thus, this does not compromise the generality of the approach. We see that the band width of this system is $\omega_{3dB} \approx 1\ sec^{-1}$. Therefore, the settling time is $t_s \approx 6\ sec$. We wish to use a mask with a depth of three, and therefore, the communication interval should be $\delta t \approx 3\ sec$.

In order to exert all frequencies of this system in an optimal manner, we shall simulate this system (directly in CTRL–C or MATLAB) applying a *binary random sequence* as the input signal [13.1]. We decided to recode each of the output states into three levels (the input is already binary), and therefore, the number of legal states can be computed as:

$$n_{leg} = \prod_{\forall i} k_i = 2 \cdot 3 \cdot 3 \cdot 3 = 54 \tag{13.32}$$

and therefore, the required number of recordings is:

$$n_{rec} = 5 \cdot n_{leg} = 270 \tag{13.33}$$

Let us simulate the system over 300 communication intervals. This is accomplished as follows:

```
[> t = 0 : 3 : 900;
[> u = ROUND(RAND(t));
[> x0 = ZROW(3,1);
[> SIMU('ic', x0);
[> y = SIMU(a, b, c, d, u, t);
```

Fig.13.9 shows the results of the continuous–time simulation.

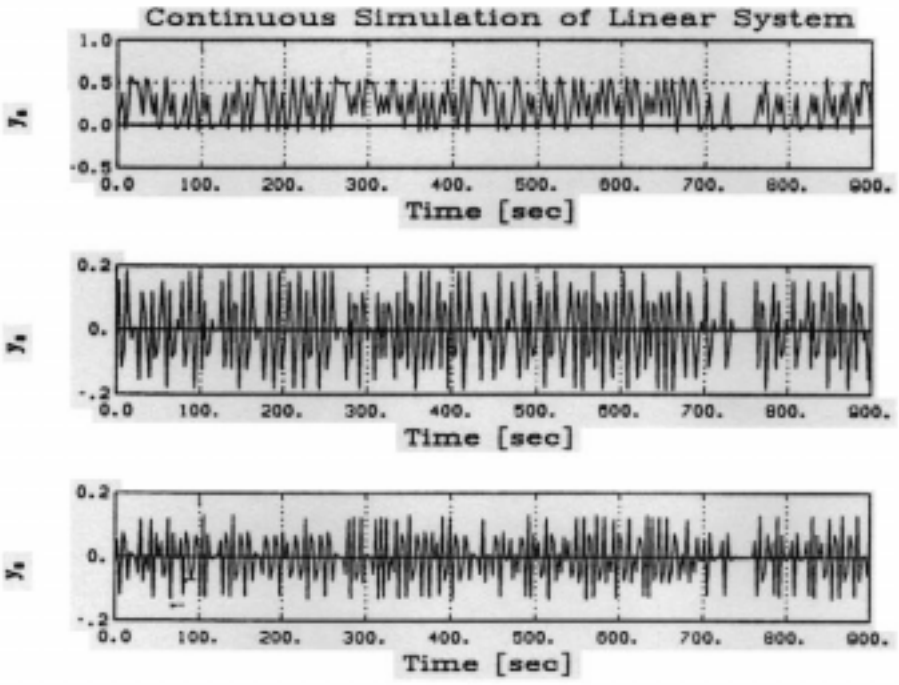

Figure 13.9. Continuous–time simulation of linear system

Notice that we use the continuous–time simulation here as we could have used a digital oscilloscope. We shall not make any use of the fact that we know the structure of our system.

We shall use our optimal recoding algorithm to discretize our three output variables:

```
[>  DO saps:saps
[>  meas = ['u', 'y'];
[>  m = meas;
[>  FOR i = 2 : 4, ...
      [indx, mi] = SORT(meas(:, i)); ...
      m(:, i) = mi; ...
    END
[>  LM = m(1, :);
[>  LM = [LM; 0.5 * (m(100, :) + m(101, :))];
[>  LM = [LM; 0.5 * (m(200, :) + m(201, :))];
[>  LM = [LM; m(300, :);
[>  raw = meas;
[>  to = 1 : 3;
[>  FOR i = 2 : 4, ...
      from = [LM(1 : 3, i), LM(2 : 4, i)]'; ...
      r = RECODE(meas(:, i), 'domain', from, to); ...
      raw(:, i) = r; ...
    END
```

The above code segment sorts each trajectory (column) vector separately, then subdivides the sorted vector into three segments of equal size to determine the optimal landmark values (LM). Thereafter, the measurement data are recoded separately for each trajectory. At the end of the code segment, *raw* contains the recoded raw data matrix.

We are now ready to search for the optimal masks. We operate on three separate mask candidate matrices, one for each of the three outputs. We shall compute the quality vector and the mask history matrix since it turns out that we shall put also the suboptimal masks to good use. We shall keep the three best masks and sort them in order of decreasing quality. The following code segment shows the optimal mask analysis for the first output. The other two masks are computed accordingly. Notice that we shall use the first 270 rows of the raw data matrix only.

```
[>  rraw = raw(1 : 270, :);
[>  mcan = -ONES(3, 4);
[>  mcan(3, 2 : 4) = [1, 0, 0];
[>  [mask, hm, hr, q1, mhis1] = OPTMASK(rraw, mcan, 5);
[>  indx = SORT(q1);
[>  m1a = mhis1(:, 4 * (indx(1) - 1) + 1 : 4 * indx(1));
[>  m1b = mhis1(:, 4 * (indx(2) - 1) + 1 : 4 * indx(2));
[>  m1c = mhis1(:, 4 * (indx(3) - 1) + 1 : 4 * indx(3));
[>  m1 = [m1a, m1b, m1c];
```

At the end of this code segment, $m1$ contains the three best masks concatenated to each other from the right.

We are now ready to forecast. During the forecasting process, it will happen from time to time that an input state is encountered which has never before been recorded. In this case, the forecasting routine will come to a halt and leave it up to the user what to do next. We decided to try the following strategy: since the input state depends on the masks, we simply repeat the forecasting step with the next best mask hoping that the problem goes away. If this doesn't help, we try the third mask. This is the reason why I saved the suboptimal masks.

I coded the forecasting in a separate routine called FRC which is called from the main procedure as follows:

```
[> DEFF frc
[> inpt = raw(271 : 300, 1);
[> pred = FRC(rraw, inpt, m1, m2, m3);
```

At this point, it should have become clear why I simulated over 300 steps although I needed only 270 recordings. The final 30 steps of the continuous–time simulation will be used to validate the forecast.

The FRC routine operates in the following way: We loop over the 30 steps of the forecast. In each step, we call the SAPS–II routine FORECAST three times, once with each of the three optimal masks to forecast one value only. At the end of the step, we concatenate the new row (forecast) to the raw data from below, and repeat. If FORECAST isn't able to predict a value since the input state has never been-seen before, it returns the raw data unchanged, i.e., the number of rows upon output is the same as upon input. In that case, we repeat the FORECAST with the next best mask. If none of the three best masks is able to predict the next step, we pick a value at random. The following code segment shows how FRC works:

```
//[frcst] = FRC(raw, inpt, m1, m2, m3);
   m1a = m1(:, 1 : 4);  m1b = m1(:, 5 : 8);  m1c = m1(:, 9 : 12);
   m2a = m2(:, 1 : 4);  m2b = m2(:, 5 : 8);  m2c = m2(:, 9 : 12);
   m3a = m3(:, 1 : 4);  m3b = m3(:, 5 : 8);  m3c = m3(:, 9 : 12);
   r = raw;
   [row, col] = SIZE(raw);
   [n, m] = SIZE(inpt);
   FOR i = 1 : n, ...
      in = inpt(i); ...
      fc = [in, ZROW(1, 3)]; ...
      fcc = [r; fc]; ...
      ff1 = FORECAST(fcc, m1a, row + i - 1, 0); ...
      [rf, cf] = SIZE(ff1); ...
      IF rf <> row + i, ...
         ff1 = FORECAST(fcc, m1b, row + i - 1, 0); ...
         [rf, cf] = SIZE(ff1); ...
         IF rf <> row + i, ...
            ff1 = FORECAST(fcc, m1c, row + i - 1, 0); ...
            [rf, cf] = SIZE(ff1); ...
            IF rf <> row + i, ...
               ff1 = [ff1; ROUND(RAND(1, 4))]; ...
            END, ...
         END, ...
      END, ...
      ... //
      ... // Same code for ff2 and ff3
      ... //
      ff = [in, ff1(row + i, 2), ff2(row + i, 3), ff3(row + i, 4)]; ...
      r = [r; ff]; ...
   END
   frcst = r;
RETURN
```

I then compared the data from the simulation with the forecast data:

```
[>  simdat = raw(271 : 300, :);
[>  frcdat = pred(271 : 300, :);
[>  error = simdat - frcdat;
```

and these are the data that I found:

$$simdat = \begin{bmatrix}
0 & 3 & 1 & 1 \\
0 & 1 & 1 & 3 \\
0 & 1 & 2 & 2 \\
0 & 1 & 2 & 2 \\
0 & 1 & 2 & 2 \\
1 & 2 & 3 & 3 \\
0 & 3 & 1 & 1 \\
1 & 2 & 3 & 3 \\
0 & 2 & 1 & 1 \\
0 & 1 & 1 & 3 \\
0 & 1 & 2 & 2 \\
1 & 2 & 3 & 3 \\
1 & 3 & 2 & 1 \\
0 & 2 & 1 & 2 \\
1 & 1 & 3 & 3 \\
0 & 3 & 1 & 1 \\
0 & 1 & 1 & 3 \\
1 & 2 & 3 & 3 \\
0 & 3 & 1 & 1 \\
1 & 2 & 3 & 3 \\
1 & 3 & 3 & 1 \\
0 & 2 & 1 & 1 \\
0 & 1 & 2 & 3 \\
1 & 2 & 3 & 2 \\
0 & 3 & 1 & 1 \\
1 & 2 & 3 & 3 \\
1 & 3 & 3 & 1 \\
1 & 3 & 2 & 2 \\
1 & 3 & 2 & 2 \\
0 & 2 & 1 & 1
\end{bmatrix}
\qquad
frcdat = \begin{bmatrix}
0 & 3 & 1 & 1 \\
0 & 1 & 1 & 3 \\
0 & 1 & 2 & 2 \\
0 & 1 & 2 & 2 \\
0 & 1 & 2 & 2 \\
1 & 2 & 3 & 3 \\
0 & 3 & 1 & 1 \\
1 & 2 & 3 & 3 \\
0 & 2 & 1 & 1 \\
0 & 1 & 1 & 3 \\
0 & 1 & 2 & 2 \\
1 & 2 & 3 & 3 \\
1 & 3 & 2 & 1 \\
0 & 2 & 1 & 2 \\
1 & 1 & 3 & 3 \\
0 & 3 & 1 & 1 \\
0 & 1 & 1 & 3 \\
1 & 2 & 3 & 3 \\
0 & 3 & 1 & 1 \\
1 & 2 & 3 & 3 \\
1 & 3 & 3 & 1 \\
0 & 2 & 1 & 1 \\
0 & 1 & 2 & 3 \\
1 & 2 & 3 & 2 \\
0 & 3 & 1 & 1 \\
1 & 2 & 3 & 3 \\
1 & 3 & 3 & 1 \\
1 & 3 & 2 & 2 \\
1 & 3 & 2 & 2 \\
0 & 2 & 1 & 1
\end{bmatrix}$$

It can be seen that the thirty data rows do not contain even a single error. The forecasting procedure worked beautifully.

It is very easy to replace the RECODE, OPTMASK, and FORECAST functions by their fuzzy counterparts. Also in this case, the forecasting works without a single error. However, now we can use routine REGENERATE to obtain a forecast also of the continuous-time signals. Fig.13.10 displays the true signals with the regenerated ones superimposed.

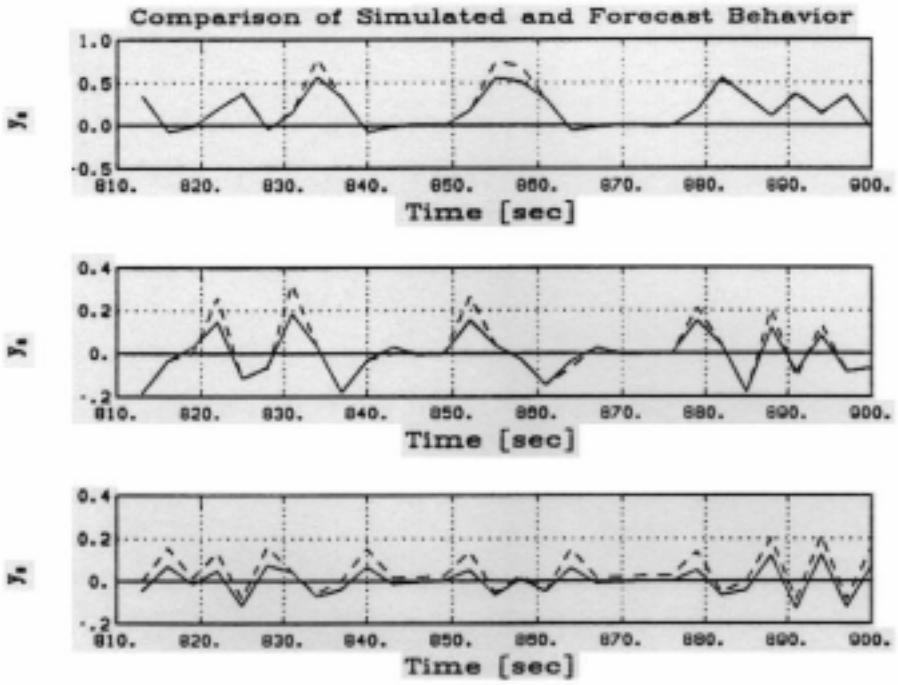

Figure 13.10. True and regenerated continuous–time signals

The solid lines represent the results from the continuous simulation. They look discontinuous because of the large communication interval of 3 *sec* used in the simulation. CTRL–C's (MATLAB's) plot routine uses linear interpolation between communication points. The dashed lines are the pseudo–continuous signals that were regenerated from the forecast using the fuzzy membership functions. If you are interested in how Fig.13.10 was produced, solve hw(13.1).

Since this example gave us sensational results, let us check whether we can capitalize on this idea.

13.7 Gambling the Stock Market

Fig.13.11 shows the end of day prices of a stock of a particular company recorded over a period of 775 days.

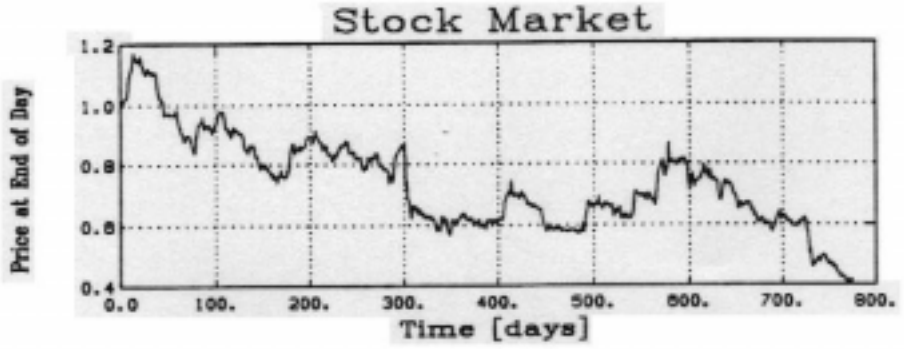

Figure 13.11. End of day prices of a company

These are real data. The name of the company is unimportant, especially since the company obviously didn't fare too well over these two years.

We notice immediately a first problem with these data. Even if the data would follow a straight decreasing line, SAPS would have problems predicting anything since each new value is "new" (has never occurred before). Before we can apply SAPS to these data, it is important that we manipulate the data in such a way that they become *stationary*, yet such that the original data can be reconstructed at any time. This process is called *detrending* of the data.

Many algorithms exist for the detrending of data. In some cases, we may decide to compute an *informal derivative*:

$$y_k = x_k - x_{k-1} \tag{13.32}$$

which, in SAPS–II, can be computed using the function:

```
[>   y = DIFF(x,1)
```

In the world of finances, it is more common to compute the *daily return* which is defined as:

$$ret_k = \frac{pr_k - pr_{k-1}}{pr_{k-1}} \tag{13.33}$$

The return on day k is defined as the end–of–day price at day k minus the end–of–day price at day $k-1$ divided by the end–of–day price at day $k-1$. SAPS–II doesn't provide for a function to compute the daily return directly, but it is a trivial task to define such a function:

```
// [y] = RETRN(x)
  [n, m] = SIZE(x)
  FOR i = 1 : m, ...
     y(:, i) = DIFF(x, 1) ./ x(1 : n − 1, i)
  END
RETURN
```

Fig.13.12 shows the daily return for the same company:

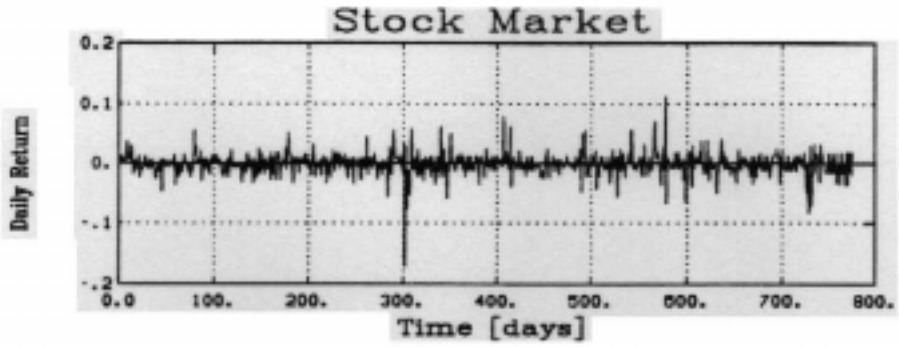

Figure 13.12. Daily return of a company

The data is now clearly detrended. It looks pretty much like a noise signal. Since we have nothing better to go by, let us try to predict future values of return from earlier values of return, i.e., we use a sort of *recursive filter* for prediction.

All we are interested in is whether the value of the stock will increase or decrease. Thus, we use our optimal recoding algorithm to recode the daily return data into the three levels 'up', 'stationary', and 'down', such that each of these levels is recorded equally often. In SAPS–II, we shall represent 'down' as '1', 'stationary' as '2', and 'up' as '3'. A natural gambling strategy would then be to sell stock when the indicator is 'down', to buy stock when it is 'up', and to do nothing when it is 'stationary'.

I decided on a mask depth of 10, I performed an optimal mask analysis using the first 500 data points (just like in the last section), and I tried to forecast the next 200 values. Then I computed the error by subtracting the forecast values from the true values, and I computed a bar chart of the errors which is shown in Fig.13.13.

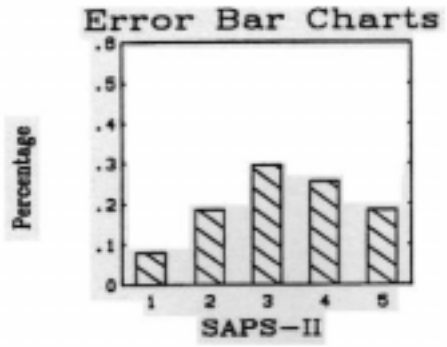

Figure 13.13. Bar chart of the SAPS-II forecasting error

CTRL-C's bar chart routine doesn't place correct labels on the x-axis. The values shown can be interpreted as:

$$x\_value = true\_value - forecast\_value + 3$$

Thus, a value of '3' indicates that the forecast was correct, a value of '1' indicates that the true value had been 'down' while the predicted value had been 'up', etc. Obviously, SAPS predicts the correct value in roughly 33% of all cases — which is not overly impressive taking into account that we operate with three levels.

To prove my point, I performed the following experiment. Instead of using an optimal mask analysis for forecasting, I simply picked, at random, a number between '1' and '3', and called this my forecast. The error bar chart for this case is shown in Fig.13.14.

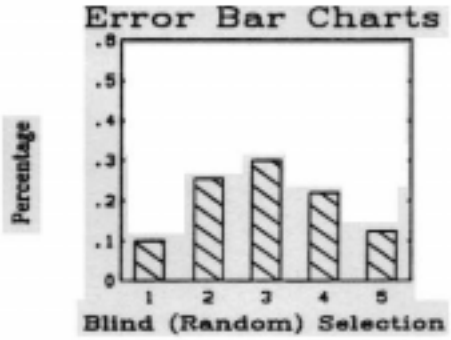

Figure 13.14. Bar chart of the blind forecasting error

Clearly, the forecasting power of SAPS was nil in this case. SAPS performed about as well as the blind algorithm, i.e., we simply coded a very expensive random number generator. Since the sample size chosen was fairly small (200 samples), a large variability remains in the data, and the theoretical values of 33% correct answers, 22% answers off by one in either direction, and 11% answers of by two in either direction are not reflected accurately by our limited statistical experiment.

Can anything be done at all? Let us first look at the most obvious forecasting strategy: the return tomorrow will be the same as today. Fig.13.15 shows the error bar chart for this case.

Figure 13.15. Bar chart of the persistent forecasting error

Again, the results are similar, that is, little positive correlation exists within the data. Let us therefore try a more refined technique: *linear regression*. I postulate that the return at day k can be predicted from a weighted sum of the returns of the previous 10 days, i.e.:

$$x_k = \sum_{i=1}^{10} a_i \cdot x_{k-i} \tag{13.34}$$

If we apply eq(13.34) to a number of days, e.g. one month, and write all the resulting equations below each other, we obtain:

$$\begin{pmatrix} x_{k-1} \\ x_{k-2} \\ \vdots \\ x_{k-30} \end{pmatrix} = \begin{pmatrix} x_{k-2} & x_{k-3} & \cdots & x_{k-11} \\ x_{k-3} & x_{k-4} & \cdots & x_{k-12} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k-31} & x_{k-32} & \cdots & x_{k-40} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{10} \end{pmatrix} \tag{13.35}$$

That is:

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{10} \end{pmatrix} = \begin{pmatrix} x_{k-2} & x_{k-3} & \cdots & x_{k-11} \\ x_{k-3} & x_{k-4} & \cdots & x_{k-12} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k-31} & x_{k-32} & \cdots & x_{k-40} \end{pmatrix}^{-1} \cdot \begin{pmatrix} x_{k-1} \\ x_{k-2} \\ \vdots \\ x_{k-30} \end{pmatrix} \tag{13.36}$$

The matrix to be "inverted" is a so–called Hankel matrix (a matrix which is constant along its anti–diagonals), and the inverse is, in fact, a pseudo–inverse since the matrix is rectangular. By using the pseudo–inverse of the Hankel matrix in eq(13.36), we solve the overdetermined linear equation system in a least square's sense. This is called the linear regression problem. CTRL–C's (or MATLAB's) "\" operator can be used to formulate the linear regression problem. The following CTRL–C code forecasts the return of days 501 to 700 on the basis of the available data. We recompute a new regression vector a for every new day on the basis of the previous month of measured data. The regression vector is then used to predict the actual (continuous) return for the next day only. The following code segment implements this algorithm:

```
[>  pdata = data
[>  FOR i = 501:700,...
      r = data(i - 2 : -1 : i - 40);...
      m = [r(1:30), r(2:31), r(3:32), r(4:33), r(5:34),...
           r(6:35), r(7:36), r(8:37), r(9:38), r(10:39)];...
      z = data(i - 1 : -1 : i - 30);...
      a = m\z;...
      pdata(i) = data(i - 1 : -1 : i - 10)' * a;...
    END
```

I then used my optimal recoding algorithm to recode the predicted data into three levels, and compared the recoded prediction with the true data. Fig.13.16 shows the resulting error bar chart for this case:

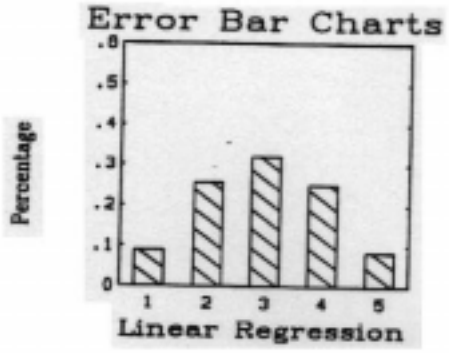

Figure 13.16. Bar chart of the regression forecasting error

Obviously, the regression algorithm didn't fare much better. However, we have a number of parameters that can be varied: (i) the length of the regression vector, in our case 10, (ii) the number of columns of the Hankel matrix, in our case 30, and (iii) the frequency of recomputing the regression vector, in our case 1 *day*. The performance of the regression algorithm depends somewhat on the selection of these three parameters. I optimized these parameters and soon discovered that the best results are obtained when a square Hankel matrix is used, and when the regression vector is recomputed every day. The results for a 2×2 Hankel matrix are shown in Fig.13.17.

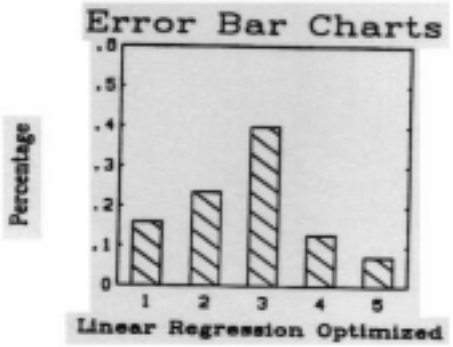

Figure 13.17. Bar chart of optimized regression forecasting error

This time, we had some success. We now predict roughly 40% of the values correctly in comparison with 33% from before. Somewhat promising results were obtained for Hankel matrices of sizes 2×2 up

to 7×7. For larger Hankel matrices, the predictive power is again reduced to noise level. The best results were found for the 2×2 case. This proves that the system exhibits at least some eigendynamics which can be captured and exploited, but the results are not very impressive.

Why did SAPS perform so poorly in this case? Two major reasons are to be mentioned. The first problem is the following: Every system's behavior is dictated by two components: (i) its eigendynamics, and (ii) its inputs. This is true also for the stock market. Unfortunately, we have not captured the effects of the inputs at all. All our models were simple data filters that tried to predict future values of return from previous values of return. Obviously, the influence of external events on the stock market is formidable. However, we weren't able to capture those since we have no clear indicator as to what these inputs are. Since very little positive correlation exists between neighboring data points, it is insufficient to estimate these inputs indirectly by measuring their effects on the stock market. We are badly in need of direct measurements of these inputs which, at this point in time, we even don't know how to characterize. The second problem is the following: SAPS requires lots of data points to come up with a decent optimal mask. Since we didn't capture the full eigendynamics of the system but only a very limited excerpt of those, the system seems to change constantly. We cannot rely on 500 past data points (i.e. 1.5 *years* worth of data) to generate a model since the behavior of the system, as experienced through the daily return data, changes so quickly. This is what really broke our neck in the case of SAPS. The more successful regression model operated on a few days of data only, whereas SAPS needed 1.5 *years* worth of data in order to come up with a model. The regression model will not work either if we extend the size of the Hankel matrix, or if we don't recompute the regression vector in regular intervals.

I am quite convinced that SAPS could work. The dominant time constant of the stock market is in the order of one day, thus the sampling rate is quite appropriate. We don't really need to rely on interday data in order to predict the stock market. Secondly, I am convinced that the behavior of the stock market is not as time–varying as it appears to be. This is only the case because we haven't determined yet which state variables to use. With the proper choice of inputs and state variables, SAPS could clearly outperform the regression model. The need to rely on so much past data shouldn't be a problem if we were able to reduce the apparent time dependency of the system. This is clearly a worthwhile research topic.

The true (and unsolvable) problem is the following: We shall never be able to predict the stock market far into the future since the influence of external inputs is at least half of the game. In order to retain the observed (mild) success with the regression model, we were able to predict only one day ahead. The probabilities of correctness of any longer predictions rapidly decay to the noise level of 33%. Within a short time span, however, the market price of an individual stock will not grow very much. Due to the commission that we have to pay in order to buy stock, we can't really exploit the system unless we are able to operate on a very large scale (since the commission to be paid depends on the trade volume). Thus, even with the best model in the world, small savers like you and me will have to rely on professional investors if our "gambling" should turn into an investment and not merely be a gamble.

13.8 Structure Characterization

We have seen that the mechanism of optimal mask analysis can provide us with effective means to describe the behavior of a system. Yet, since the methodology relies on an exhaustive search of possible masks from a set of mask candidates, this search will become prohibitively expensive if we make the set of mask candidates too large. Experience has shown that the number of variables among which the optimal mask is to be found should not be much larger than five, and that the mask depth should usually be limited to three or four. This can pose a severe limitation of the technique since we are often confronted with systems with large numbers of candidate variables among which we should choose an appropriate subset for the optimal mask analysis. The stock market illustrates this problem very well. What are adequate inputs, and which are appropriate state variables? The choices are manifold.

In this section, we shall study the problem of selecting subsets of variables for optimal mask analyses. We call this the *structure characterization* problem. Contrary to Chapter 11 where structure characterization had been defined as the process of identifying a particular function that relates one or several inputs to one or several outputs, we shall now define the structure characterization as the process of determining which variables are related to which other variables in a multivariable system.

This problem is more closely related to what we called *causality* in Chapter 11, and the classical statistical technique to decide this question is *correlation analysis*. If a strong (positive or negative) correlation exists between two variables, they belong to the same subsystem, i.e., they are causally related to each other. However, a problem remains with this concept. Let us assume that we measure two variables a and b which are related to each other through the equation:

$$a = b$$

Obviously, these variables have the largest positive correlation possible. Yet, we should not select both a and b for our optimal mask analysis since once we selected one of these variables the other does not add any additional information to our analysis. It is redundant.

In this section, we shall introduce a new technique called *reconstruction analysis* which can be used as an alternative to the previously introduced correlation analysis. However, while correlation analysis works well on continuous signals, reconstruction analysis has been devised to operate on recoded, i.e. discrete, signals.

SAPS–II distinguishes between three different types of structure representation. A *causal structure* lists the variables that form the subsystems as a row vector whereby subsequent subsystems are separated by a zero. Below this row vector, masks are coded that show the causal relation between the variables of the subsystems. For example, the structure shown in Fig.13.18:

Figure 13.18. A causal structure

can be coded as:

$$istc = \begin{bmatrix} 1 & 4 & 6 & 0 & 2 & 3 & 4 & 5 & 0 & 5 & 6 & 7 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & -1 & 0 & -2 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & -2 & 0 & 1 \end{bmatrix}$$

As indicated in the first row, one subsystem consists of the variables 1, 4, and 6. The mask written underneath this structure shows that variable 1 is an input (negative entry in the mask), while variables 4 and 6 are two outputs (positive entries in the mask).

By eliminating the masks from the causal structure, we obtain a *composite structure*. Obviously, the composite structure contains less information than the causal structure. The direction information is lost, and also the timing information is gone. In SAPS–II, the composite structure is represented by the first row of the causal structure:

$$istr = [\ 1\ 4\ 6\ 0\ 2\ 3\ 4\ 5\ 0\ 5\ 6\ 7\ ]$$

which can stand for the structure shown in Fig.13.19a:

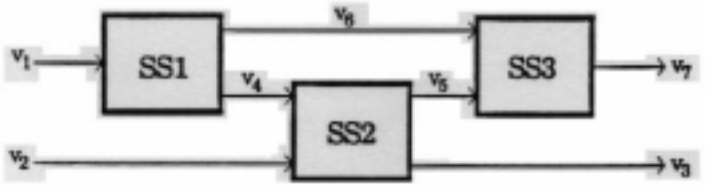

Figure 13.19a. Possible representation of a composite structure

but which can also represent the quite different structure shown in Fig.13.19b:

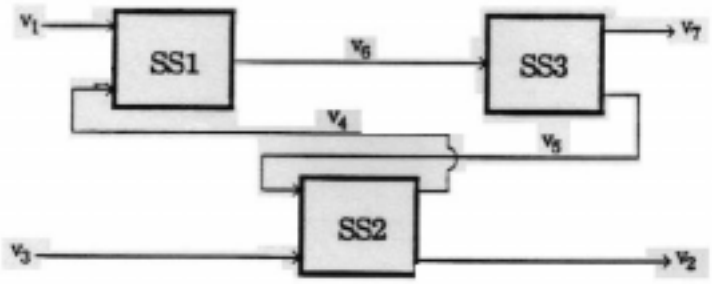

Figure 13.19b. Another representation of the same composite structure

It can be easily verified that both systems contain the same three subsystems, and are thus identified by the same composite structure.

The composite structure no longer contains information about which variables are inputs and which are outputs.

The third structure representation available in SAPS–II is the *binary structure*. A binary structure is an ordered list of all binary relations between variables belonging to the same subsystem. The SAPS–II function:

```
[>  istb = BINARY(istr)
```

generates the binary structure out of the composite structure. For our example, the resulting binary structure is:

```
istb = [ 1 4
         1 6
         2 3
         2 4
         2 5
         3 4
         3 5
         4 5
         4 6
         5 6
         5 7
         6 7 ]
```

The binary structure can be easily obtained from the composite structure by drawing imaginary lines between any two variables of each subsystem, writing down the variable names (numbers) at the two ends of each such line as a pair, sorting all these pairs alphabetically, and eliminating all the redundant pairs.

Again, the binary structure contains less information than the composite structure it represents. For example, the system shown in Fig.13.20:

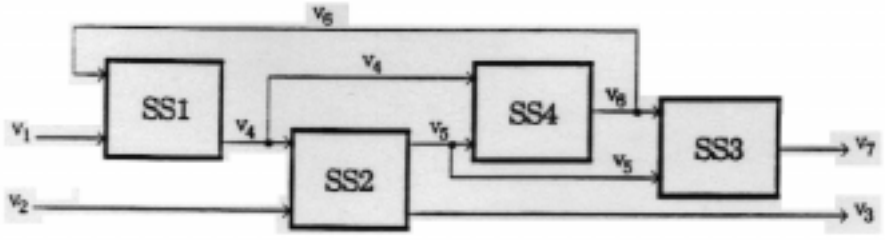

Figure 13.20. Another realization of the same binary structure

with the composite structure:

$$istr = [\ 1\ 4\ 6\ 0\ 2\ 3\ 4\ 5\ 0\ 5\ 6\ 7\ 0\ 4\ 5\ 6\ ]$$

possesses the same binary structure as our previous system, although it contains an additional subsystem (SS4), and thus a different composite structure.

In SAPS–II, the command:

$$[>\quad istr = COMPOSE(istb)$$

produces the *minimal* composite structure among all possible composite structures representable by the same binary structure, i.e., it produces the composite structure with the smallest number of subsystems.

Similar to the optimal mask analysis, we wish to perform an *optimal structure analysis* which we shall present with a set of candidate structures out of which the qualitatively best structure is selected. This means that we need to come up with a quality measure for structures. This measure is based on the composite structure.

Given the *behavior model* of a set of raw (recoded) data:

$$[>\quad [b, p] = BEHAVIOR(raw)$$

Notice that the behavior model does not imply knowledge of which variables are inputs and which are outputs. In the past, we always applied the BEHAVIOR function to input/output models, but this is not necessary. If we *extract* a particular subsystem from the behavior model, we simply throw those columns away which represent variables that are not included in the subsystem, we then merge rows which have become undistinguishable, add up their probabilities (or confidences), and sort the resulting behavior model again in alphabetical order. In SAPS–II, this can be accomplished using the function:

$$[>\quad [b2, p2] = EXTRACT(b1, p1, istr)$$

where *istr* is a *primitive composite structure*, i.e., a composite structure which contains one subsystem only.

Given the behavior model:

$$b = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix} \qquad p = \begin{bmatrix} 0.2 \\ 0.1 \\ 0.1 \\ 0.2 \\ 0.1 \\ 0.3 \end{bmatrix}$$

We can extract the first two variables using the command:

```
[> [b12,p12] = EXTRACT(b,p,[1,2])
```

which results in the behavior model:

$$b12 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad p12 = \begin{bmatrix} 0.3 \\ 0.1 \\ 0.2 \\ 0.4 \end{bmatrix}$$

We can alternatively extract the second and third variable using the command:

```
[> [b23,p23] = EXTRACT(b,p,[2,3])
```

which results in the behavior model:

$$b23 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad p23 = \begin{bmatrix} 0.4 \\ 0.1 \\ 0.1 \\ 0.4 \end{bmatrix}$$

We can then *combine* the two substructures using the command:

```
[> [bb,pp] = COMBINE(b12,p12,b23,p23,[1,2,0,2,3])
```

where the fifth input parameter is again a composite structure which describes the nature of the recombination of substructures. The probability of a recombined state is computed as the probability of the particular state of the first subsystem multiplied with the conditional probability of the particular state of the second subsystem. For example, the probability of the state $[0,0,0]$ is 0.24 since the probability of the state $[0,0]$ of the first subsystem is 0.3, and the conditional probability of the state $[0,0]$ of the second subsystem is 0.8 (if the variable '2' has assumed a value of 0, variable '3' assumes a value of 0 in 80% of the cases and a value of 1 in 20% of the cases). The resulting behavior model is:

$$bb = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix} \qquad pp = \begin{bmatrix} 0.24 \\ 0.06 \\ 0.02 \\ 0.08 \\ 0.16 \\ 0.04 \\ 0.08 \\ 0.32 \end{bmatrix}$$

When we compare the recombined system with the original system, we find that two additional states have been introduced, but both of these have low probabilities of occurrence. We can compute the differences between the probabilities of the original behavior model and the recombined behavior model, and compute the L_2–norm of the difference vector. This is defined as the *reconstruction error* of the particular composite structure.

In SAPS–II, the reconstruction error of a particular composite structure can be computed directly using the function:

$$[> \quad err = STRUCTURE(b, p, istr)$$

The optimal structure analysis is a *minimax* problem. Obviously, the totally connected structure has always a reconstruction error of 0.0 since nothing is extracted and nothing is recombined. Thus, we cannot simply minimize the reconstruction error. Instead, we try to maximize the number of subsystems (i.e., enhance the complexity of the internal structure of the system) while keeping the reconstruction error below a user specified maximum allowed error. For any given complexity, we choose the structure which minimizes the reconstruction error.

How do we enumerate possible structures, and how do we define the complexity of a structure? The structure search algorithm used in the optimal structure analysis bases on the binary structure representation of the system. The "complexity" of a structure is simply defined as the number of rows of the binary structure. For example, the binary structure shown earlier in this section has a complexity of 12. Notice that this definition of complexity is not congruent with the previously used term "complexity of the internal structure". The most "complex" system is the totally connected system which has no internal structure at all.

Three different structure optimization algorithms have currently been implemented. The *structure refinement algorithm* starts with a totally interlinked composite structure in which all possible binary

relations are present. This structure, of course, shows no reconstruction error at all as there is nothing to be reconstructed. Binary relations are cancelled one at a time, and the structure is selected which exhibits the smallest reconstruction error. The iteration goes on by taking the selected binary structure and again cancelling one binary relation at a time and picking the structure with the smallest reconstruction error. The search proceeds from the highest complexity to lower and lower complexities. At each level, the resulting reconstruction error will be either larger than before or will be the same. The search continues until the reconstruction error becomes too large.

The *structure aggregation algorithm* starts with a system in which each variable forms a substructure of its own that is not linked to any other variable. No binary relations are thus initially present, and the reconstruction error of this structure is very large. Binary relations are added one at a time, and at each level the structure is selected which shows the largest reduction of the reconstruction error. The iteration goes on until the reconstruction error has become sufficiently small.

The *single refinement algorithm* starts similar to the structure refinement algorithm. However, instead of cancelling one binary relation only, all binary relations that exhibit a sufficiently small reconstruction error are cancelled at once, and only one step of refinement is performed.

All three algorithms are *suboptimal* algorithms, since neither of them investigates all possible structures. Therefore in a sufficiently complex system, the three algorithms may well suggest three different structures, and it often pays off to try them all. The single refinement algorithm is much cheaper than the other two algorithms, and yet it performs amazingly well. Thus, in a real time (on–line) structure identification, this will probably be the algorithm of choice.

In SAPS–II, an optimal structure analysis is performed using the function:

```
[>  istr = OPTSTRUC(b,p,errmax,group,algor)
```

where $[b, p]$ is the behavior relation of the system to be analyzed, and *errmax* is the largest reconstruction error tolerated. The *group* parameter allows to aid the optimal structure algorithm by providing *a priori* knowledge about the structure to be analyzed. For example, the grouping information:

```
group = [ 1 2 3 1 0 4 ]
```

tells the optimization algorithm that, in a six variable system, the first and the fourth variable appear in any subsystem either together or not at all, whereas the fifth variable is certainly disassociated, i.e., it does appear only in one subsystem in which no other variable is represented. Thus, the six variable system is effectively reduced to a four variable system. If no *a priori* knowledge about the structure exists, the grouping information should be coded as:

```
group = 1 : 6
```

The *algor* parameter finally tells the analysis which of the three algorithms to use. Possible values are:

```
algor = 'REFINE'
algor = 'AGGREGATE'
algor = 'SINGLEREF'
```

istr is the resulting composite structure of the system.

One additional feature has been built into SAPS–II. After an optimization has taken place, the resulting structure can be *postoptimized* by applying the single refinement algorithm once more to each of its substructures. This algorithm is executed using the command:

```
[> istp = SINGLEREF(istr, b, p, errmax)
```

As an example, let us once more consider the open heart surgery problem discussed earlier in this chapter. This problem consists of a six variable model recoded as shown in Table 13.1. Expert knowledge was used to determine the landmarks between the five different levels for each variable. The recoded raw data model was stored away on a file using the CTRL–C command:

```
[> SAVE raw > saps:heart.dat
```

The following CTRL–C macro performs an optimal structure analysis on the previously saved raw data model:

```
[>  DO saps:saps
[>  repo = 1;
[>  LOAD < demo:heart.dat
[>  [b,p] = BEHAVIOR(raw);
[>  igr = 1 : 6
[>  ermax = 0.016
[>  is1 = OPTSTRUC(b,p,ermax,igr,'refine')
[>  isr1 = SINGLEREF(is1,b,p,ermax)
[>  is2 = OPTSTRUC(b,p,ermax,igr,'aggregate')
[>  isr2 = SINGLEREF(is2,b,p,ermax)
[>  is3 = OPTSTRUC(b,p,ermax,igr,'singleref')
[>  isr3 = SINGLEREF(is3,b,p,ermax)
```

The analysis starts by applying the structure refinement algorithm to the behavior relation of the system. No *a priori* knowledge about the structure is yet available. The resulting composite structure is:

$$is1 = (1,2,4)(1,3,4)(2,4,5)(2,4,6)$$

A postoptimization of this structure leads to:

$$isr1 = (1,2,4)(1,3,4)(2,4,6)(5)$$

Obviously, the fifth variable (heart rate) is only weakly related to the other variables in the system.

Next, the structure aggregation algorithm is applied. This algorithm suggests the composite structure:

$$is2 = (2,4,6)(1,2,3,4)(5)$$

and postoptimization leads to:

$$isr2 = (3,4)(1,2,4)(2,4,6)(5)$$

Finally, I tried the single refinement algorithm. This time, the following composite structure is found:

$$is3 = (3,4)(4,6)(1,2,4)(5)$$

which is not modified by postoptimization.

Analyzing the different suggested structures, it becomes obvious that the heart rate does not need to be considered at all. It also shows that a very strong link exists among variables one and four. Thus, those two variables can easily be grouped together. The analysis is now repeated applying the appropriate grouping information:

```
[> igr=[ 1 2 3 1 0 4 ]
```

This time, all three algorithms suggest the same composite structure:

$$tstr = (1,2,4)(1,3,4)(1,4,6)(5)$$

which seems to be a good working hypothesis for a continuation of the system analysis.

13.9 Causality

While the concept of a "causal structure" was introduced in the last section, we haven't made use of this concept yet. While the optimal structure analysis was able to reveal causal relations among variables, it has not helped us to determine causality relations between those variables.

We could employ the correlation techniques that were suggested in Chapter 11 for this purpose, i.e., after we have determined which variables belong together, we could compute cross–correlations pairwise between them and check whether one of the variables of any such pair lags significantly behind the other. Yet, correlation analysis works better on continuous variables, and it might be more appropriate to tackle the recoded data directly.

Two separate routes are basically open to solve this problem:

(1) We can apply an optimal mask analysis repetitively in a loop, i.e., we compute an optimal mask separately for every true output at time t (as we have done in the past), and then interpret some of the inputs to these optimal masks as outputs from another optimal mask shifted further back in time. If an input to an optimal mask is a true system input, no further mask needs to be evaluated to explain it. If it is a true output (evaluated at an earlier point in time), we already know the optimal mask – it is the same mask as for the current time simply shifted back a number of steps. However, if the input to the optimal mask is an auxiliary variable which is neither a true input nor a true output, we make it an output, and compute an optimal mask for it.

(2) We realize that the optimal structure analysis is flat. The time dependency is not preserved. Remember that the structure analysis operates on the behavior model rather than on the raw data model. The behavior model does not retain the time flow information. In order to remedy this problem, we can duplicate the raw data model, shift it by one row, and write the shifted data to the side of the original data as additional variables. We can repeat this process for a number of times. In SAPS–II, this can be achieved by applying the IOMODEL function to the raw data with a mask in which all elements are "inputs" except for the last which we shall call an "output". If the mask has n_{depth} rows and n_{var} columns, we obtain an input/output model with $n_{var} \times n_{depth}$ columns. We can then apply an optimal structure analysis to the enhanced input/output model and reinterpret the resulting structure in terms of the original time dependent variables.

Both techniques may be used to obtain a causal structure of our system. However, the resulting structures are not necessarily the same, and it is not obvious at this point what is the precise relationship between the resulting structures, and when which of the two techniques might be more more appropriate to use than the other. Moreover, many "correct" structures may exist that faithfully represent a given data set. How can we discriminate between them? How can we gain additional evidence which will help us increase our faith in one of the possible structures and discard others? Many questions haven't been answered yet in this context. This is therefore an excellent area for research.

13.10 Summary

In this chapter, we have introduced a number of pattern recognition techniques that can be used for qualitative simulation or forecasting of system behavior. Optimal mask analysis allows us to determine qualitative causality relations among a set of causally related variables. Optimal masks can be viewed as a sort of feature extractor. Similar patterns will lead to the same optimal mask, and therefore, the optimal mask can help us recognize similarities between patterns. These patterns can be either temporal patterns (such as time signals) or static patterns (such as images). Optimal mask analysis is still a far cry from automating the capability of my physician to remember

faces after a long time, but it may be a first step towards mimicking this capability in a computer program. Optimal structure analysis allows us to reveal causal relations among a set of time–related variables. This technique can be useful as a means to identify important factors that are related to a particular event.

SAPS–II was introduced as a tool to qualitatively analyze systems using these techniques. SAPS–II is available as a CTRL–C function library and also as a Pro–Matlab (PC–Matlab) toolbox. In this text, I reproduced the CTRL–C solutions to the presented algorithms. The MATLAB solutions look very similar.

The research described in this chapter can be attributed largely to one single scientist and colleague, George Klir, a great thinker. I wish to acknowledge my gratitude for his constructive comments and frequent encouragements.

References

[13.1] François E. Cellier (1987), "Qualitative Simulation of Technical Systems Using the General System Problem Solving Framework", *International J. of General Systems*, **13**(4), pp. 333–344.

[13.2] François E. Cellier (1987), "Prisoner's Dilemma Revisited — A New Strategy Based on the General System Problem Solving Framework", *International J. of General Systems*, **13**(4), pp. 323–332.

[13.3] François E. Cellier (1990), "General System Problem Solving Paradigm for Qualitative Modeling", in: *Qualitative Simulation, Modeling, and Analysis*, (P.A. Fishwick and P.A.Luker, eds.), Springer Verlag.

[13.4] François E. Cellier, and David W. Yandell (1987), "SAPS–II: A New Implementation of the Systems Approach Problem Solver", *International J. of General Systems*, **13**(4), pp. 307–322.

[13.5] George J. Klir (1985), *Architecture of Systems Problem Solving*, Plenum Press, New York.

[13.6] George J. Klir (1989), "Inductive Systems Modelling: An Overview", *Modelling and Simulation Methodology: Knowledge Systems' Paradigms*, (M.S. Elzas, T.I. Ören, and B.P. Zeigler, eds.), Elsevier Science Publishers B.V. (North–Holland), Amsterdam, The Netherlands.

[13.7] Averill M. Law, and W. David Kelton (1990), *Simulation Modeling and Analysis*, Second Edition, McGraw–Hill, New York.

[13.8] DongHui Li, and François E. Cellier (1990) "Fuzzy Measures in Inductive Reasoning", *Proceedings 1990 Winter Simulation Conference*, New Orleans, LA.

[13.9] Glenn Shafer (1976), *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, N.J.

[13.10] Claude E. Shannon, and Warren Weaver (1964), *The Mathematical Theory of Communication*, University of Illinois Press, Urbana, IL.

[13.11] Hugo J. Uyttenhove (1979), *SAPS — System Approach Problem Solver*, Ph.D. Dissertation, (G.J. Klir, adv.), SUNY Binghampton.

[13.12] Pentti J. Vesanterä, and François E. Cellier (1989), "Building Intelligence Into an Autopilot - Using Qualitative Simulation to Support Global Decision Making", *Simulation*, **52**(3), pp. 111–121.

[13.13] Lotfi A. Zadeh (1985), "Syllogistic Reasoning in Fuzzy Logic and its Application to Usuality and Reasoning with Dispositions", *IEEE Trans. Systems, Man, and Cybernetics*, **SMC–15**(6), pp. 754–763.

[13.14] Lotfi A. Zadeh (1986), "A Simple View of the Dempster–Shafer Theory of Evidence and its Implication for the Rule of Combination", *The AI Magazine*, Summer Issue, pp. 85–90.

[13.15] Lotfi A. Zadeh (1987), "A Computational Theory of Dispositions", *International J. of Intelligent Systems*, **2**, pp. 39–63.

Bibliography

[B13.1] Russell L. Ackoff (1978), *The Art of Problem Solving*, Wiley–Interscience, New York.

[B13.2] Ludwig von Bertalanffy (1969), *General System Theory: Foundations, Development, Applications*, G. Braziller Publishing, New York.

[B13.3] G. Broekstra (1978), "On the Representation and Identification of Structure Systems", *International J. of Systems Science*, **9**(11), pp. 1271–1293.

[B13.4] Brian R. Gaines (1979), "General Systems Research: Quo Vadis", *General Systems Yearbook*, **24**, pp. 1–9.

[B13.5] George J. Klir, ed. (1978), *Applied General Systems Research*, Plenum Press, New York.

[B13.6] Allan Newell, and Herbert A. Simon (1972), *Human Problem Solving*, Prentice–Hall, Englewood Cliffs, N.J.

[B13.7] John L. Pollock (1990), *Monic Probability and the Foundations of Induction*, Oxford University Press, New York.

[B13.8] Joseph E. Robertshaw, S. J. Mecca, and M. N. Rerick (1979), *Problem Solving: A Systems Approach*, McGraw–Hill, New York.

[B13.9] Herbert A. Simon (1972), "Complexity and the Representation of Patterned Sequences of Symbols", *Psychological Reviews*, **79**, pp. 369–382.

[B13.10] Herbert A. Simon (1977), *Models of Discovery and Other Topics in the Methods of Science*, Reidel Publishing, Boston, MA.

[B13.11] G. Towner (1980), *The Architecture of Knowledge*, University Press of America, Washington, D.C.

Homework Problems

[H13.1] Fuzzy Forecasting

Implement the algorithms as presented in Section 13.6 and reproduce the probabilistic forecasting results shown in that section.

Implement a fuzzy forecasting routine:

```
[>  [frcst,Mf,sf] = FFRC([raw,Mr,sr],[inpt,Mi,si],m1,m2,m3)
```

It is necessary to concatenate the membership and side matrices from the right to the raw data matrix and the input matrix since CTRL–C limits the number of formal arguments of any function to no more than ten. FFRC is very similar to FRC, except that the calls to FORECAST are replaced by corresponding calls to FFORECAST.

Modify the main routine by replacing the 'domain' parameter by the 'fuzzy' parameter in RECODE, and by replacing the OPTMASK calls by corresponding FOPTMASK calls.

Add at the end a call to REGENERATE to retrieve the continuous-time signals, and plot the true signals (from the CTRL–C or MATLAB simulation) together with the retrieved ones as shown in Fig.13.10.

[H13.2] Political Modeling

We wish to investigate the relationship between the political situation in this country and the long range stock market trends. The political situation is captured in terms of three variables: P_P denotes the party affiliation of the President, C_H denotes the House control, and C_S denotes the Senate control. Each of these three variables is of the enumerated type with values R for Republican, and D for Democrat. The stock market index is

represented through the variable I_{SM} which is also enumerated with the values u for up, and d for down.

Table H13.2 lists the values of these four variables between 1897 and 1989.

Table H13.2 Political/marketing variables

period (years)	P_P	C_H	C_S	I_{SM}
1897 – 1901	R	R	R	u
1901 – 1905	R	R	R	u
1905 – 1909	R	R	R	u
1909 – 1913	R	R	R	u
1913 – 1917	D	D	D	d
1917 – 1921	D	D	D	d
1921 – 1925	R	R	R	d
1925 – 1929	R	R	R	u
1929 – 1933	D	R	R	u
1933 – 1937	D	D	D	u
1937 – 1941	D	D	D	d
1941 – 1945	D	D	D	d
1945 – 1949	D	D	D	u
1949 – 1953	D	D	D	d
1953 – 1957	R	R	R	u
1957 – 1961	R	R	D	u
1961 – 1965	D	D	D	u
1965 – 1969	D	D	D	u
1969 – 1973	R	D	D	d
1973 – 1977	R	D	D	d
1977 – 1981	D	D	D	d
1981 – 1985	R	D	R	u
1985 – 1989	R	D	D	u

It is assumed that the party affiliation of the President is an *exogenous variable*, i.e., it depends only on quantities that have not been included in the model (in particular, the personality of the candidate). However, the other three variables are *endogenous variables*, i.e., they are determined internally to the model. In terms of our normal nomenclature, the party affiliation of the President is an *input* variable, while the other three variables are *output* variables.

We wish to use optimal mask analysis to make predictions about the future of this political system. Start by computing three optimal masks for the three output variables. House and Senate control may depend upon past values of all four variables one and two steps back only. The stock market index may depend on these same variables plus the current value of the President's party affiliation. Forecast the future of the system over 12 *years* using these optimal masks. Repeat your forecast for all eight possible combinations of the input variable.

[H13.6] Logic Circuit With Memory

Given the logical circuit with memory shown in Fig.H13.6.

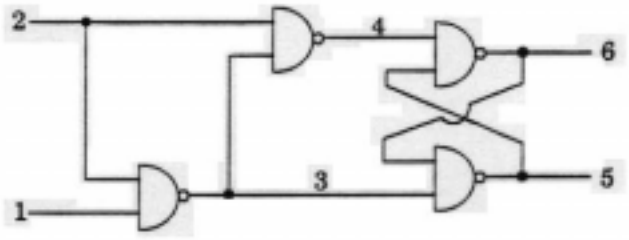

Figure H13.6. Circuit diagram of a logical circuit with memory

This circuit consists of four *NAND* gates. However, contrary to the previous examples, this circuit is not memoryless.

This time, the approach used above won't work, since not all input/output relations are single–valued. Instead, start with the signals '1', '2', and '5' in their *false* (i.e., 0) state. Compute the resulting values for the signals '3', '4', and '6'.

Then apply pseudo–random binary signals to the two inputs, i.e., at each "clock", each of the two inputs can either toggle or stay the same. For each clock, compute the resulting values of the two outputs '5' and '6', and of the two auxiliary variables '3' and '4'. Repeat for 200 steps.

At this point, we have produced our raw data matrix. Notice that while the overall circuit is able to memorize, each of the four subcircuits is still memoryless. The memory is achieved by means of the feedback loops. Apply the same techniques as in the last two homework problems to check whether you can re–learn the structure of this system.

Now, we wish to try yet another approach. This time, let me assume that we know for a fact that the circuit to be identified was built in *NAND* logic. Use the EXTRACT function of SAPS–II to associate each of the outputs with any two other signals, extract the three variable subsystem, and check whether the resulting behavior model corresponds to the truth table of a *NAND* gate. If one or the other of the identified gate inputs is an auxiliary variable, repeat the above analysis with this variable as the new output.

Since quite a bit of testing needs to be performed, we better write a general–purpose CTRL–C (MATLAB) function that can perform this analysis for an arbitrary behavior model.

we cannot tolerate any reconstruction error at all. Set the SAPS system variable *repo* (internal output) to 1, so that you can see how the software reasons about your data. Interpret the results.

Let us now try to solve the same problem using optimal mask analysis. This time, we want to assume that we know that variables 1 to 5 are true system inputs, while variables 9 and 10 are true system outputs which furthermore do not depend directly on each other. Since we know that all input/output relations are immediate, we choose for each output a mask candidate matrix of depth one in which the five inputs and the three auxiliary variables are possible inputs. If we find that the optimal masks depend on any of the auxiliary variables, we repeat the analysis by making this auxiliary variable our new output. It can depend on the five inputs and on all other auxiliary variables. We repeat the analysis until all optimal mask inputs have been reduced to true system inputs. Interpret the results.

The second technique provides us immediately with a *causal structure*. Manually derive the resulting composite structure. Compute the behavior model of the overall system, and extract one after the other each of the subsystems. Compare the behavior models of the subsystems with the logical truth tables of the hardware components.

[H13.5] Boolean Logic

Given the memoryless logical circuit shown in Fig.H13.5.

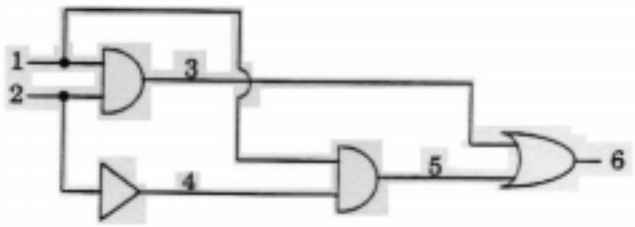

Figure H13.5. Circuit diagram of a logical circuit

Repeat the analyses proposed in hw(H13.4) to this new model. This time, the raw data model has six columns (variables) but only four rows (legal states). What do you conclude?

[H13.3] Model Validation

We wish to validate the model that was derived for the political system of hw(H13.2). For this purpose, we recompute optimal masks using only the first 75% of the data. "Forecast" the behavior of the system to the current date using the optimal masks found above, and compare the outcome with the real data. Repeat the forecast with some of the suboptimal masks in the mask history that have a similar quality, and check which of these suboptimal masks produces the smallest forecasting error.

[H13.4] Boolean Logic

Given the memoryless logical circuit shown in Fig.H13.4.

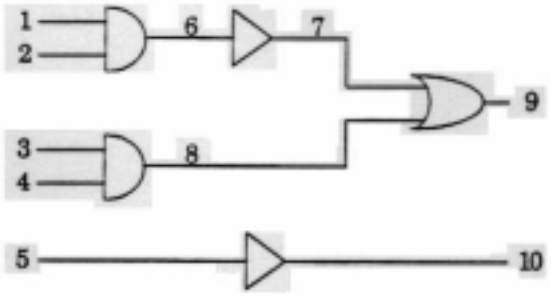

Figure H13.4. Circuit diagram of a logical circuit

The circuit contains two logical *AND* gates, two logical inverters, and one logical *OR* gate. The fact that the circuit is not completely connected is of no concern.

Write a logical table of all possible states of this circuit. Enumerate all possible combinations of inputs, and compute the set of resulting outputs. The enumeration results in a table containing 32 rows (possible states) and 10 columns (variables). Represent the logical *false* state as '0', and the logical *true* state as '1'. This table can be interpreted as a raw data model.

We now forget where the raw data came from. We wish to re-learn the structure of our circuit from the given raw data model using optimal structure analysis and optimal mask analysis.

Let us start with optimal structure analysis. We know that this circuit has no memory. Thus, every input/output relation is immediate. Therefore, we can apply the optimal structure analysis directly to the behavior model of the given raw data matrix. Try all three optimization algorithms. No grouping information is to be used. Set the maximum error *errmax* to the machine resolution *eps*. This is reasonable since we know that the data has been produced by a completely deterministic logical circuit. Thus,

[H13.7]* Logic Circuit With Faults

We want to repeat hw(H13.6). This time, we wish to study the influence of faults in the raw data matrix.

We generate the raw data matrix in exactly the same manner as in hw(H13.6). Then, we disturb the raw data matrix by introducing a certain percentage of *erroneous elements*. For instance, 1%, 2%, 5%, or 10% of the elements in the raw data matrix are being assigned the wrong value.

Repeat the same analyses as in hw(H13.6) using the disturbed raw data matrix for the four degrees of fault severity suggested above. When using optimal structure analysis, it may now be necessary to raise the *errmax* parameter beyond the machine resolution *eps* in order to accommodate for the digital noise. When using the EXTRACT function, you need to allow for a somewhat inaccurate truth table. In each step, try all combinations of gate input variables, and pick the one that gives you the smallest least square error when compared to the truth table of the *NAND* gate.

Critically evaluate the capability of the three techniques to function properly under the influence of digital noise.

Projects

[P13.1] TicTacToe

Design a CTRL–C (MATLAB) routine that can play TicTacToe.

$$[>\quad new = TICTACTOE(old)$$

plays one step of TicTacToe. *old* refers to the bord before the step. *old* is represented through a 3×3 matrix consisting of integer elements with the values 0, 1, and 2. A value of 0 indicates that the particular field is unoccupied, 1 indicates that the field is occupied by the program, and 2 indicates that the field is occupied by the human player. *new* refers to the bord after the step has been executed. If the game starts with a zero matrix, the program will make the first move, if it starts with one field already occupied by a 2 element, the human player begins.

Begin by designing a set of logical rules that describe both the (syntactic) rules of the game (such as: 'only those fields can be occupied which are currently unoccupied', or: 'only one field can be altered within one step'), as well as the (semantic) strategies (such as: 'if the opponent has currently occupied two fields in a line, and the third field in that line is currently unoccupied, then occupy that field immediately'). It is possible to describe the entire TicTacToe game by a set of seven logical rules.

Design a CTRL–C (MATLAB) routine which can interrogate the set of rules (the so–called *rule base*), and therefore, can play TicTacToe. This program is a so–called *expert system.*

Design a data collection routine which can observe the game, and which stores the data (the bord) away in a raw data matrix. Design another MATLAB (CTRL–C) routine which, using SAPS–II functions, can analyze the raw data matrix by either observing how the program plays against you, or by observing how you play against the program. It can thereby re–learn a set of logical rules (a behavior model) that describe both the syntactic rules and the semantic strategies of the game. In all likelihood, this automatically generated rule base will be considerably longer than the one that you created manually. When I tried it, the new rule base contained 124 rules.

Notice that you need to operate on an *input/output model.* If the bord before the TicTacToe program plays is stored as input, and the bord after the TicTacToe program has played is stored as output, SAPS–II will learn the strategy of the TicTacToe program. On the other hand, if the bord after the TicTacToe program has played is stored as input, and the bord before the TicTacToe program plays again is stored as output, SAPS–II will learn the strategy of the human player.

Let the expert system now play using the rule base that was automatically generated by SAPS–II rather than the manually generated rule base. If all goes well, the new program should be as good at playing TicTacToe as the original program was.

Notice the significance of this result. Instead of generating a rule base manually, the data collection routine could also have been used to observe two human players playing one against the other. Thus, we have found a methodology to automatically synthesize knowledge bases from observed data.

Replace TicTacToe by another game, such as the Prisoner's Dilemma problem [13.2]. The data collection routine must be modified to interface correctly with the new problem. However, the rule base synthesizer (the SAPS–II routine) and the expert system shell (the play routine) should be able to learn and play the new game without a single modification.

[P13.2] Intelligent Autopilot

Apply the techniques presented in Section 13.6 to the continuous simulation of a Boeing 747 jetliner in high altitude horizontal flight, i.e., to the simulation program designed in pr(P4.1). The purpose of this study is to check whether we can apply the proposed techniques as reliably to a highly non–linear system as to a linear system.

Enhance the ACSL program designed in pr(P4.1) by a set of faulty operational modes such as heavy ice on the wings or loss of one of the four engines.

For each of the fault modes, identify a set of optimal masks that characterize the fault.

Modify the ACSL program once more. This time, one of the faults should be chosen arbitrarily at a randomly selected point in time during the simulation.

Use the forecasting routine to identify *when* the accident has happened, and discriminate the correct fault by comparing the recoded continuous data after the accident occurred and after the transients resulting from the accident have died out with forecasts obtained using all of the stored fault masks. The forecast with the smallest deviation identifies (most likely) the type of fault that has occurred.

Research

[R13.1] Rule Base Minimization

Start with the two automatically generated rule bases of pr(P13.1). The purpose of this research is to come up with an automated procedure that can reduce the synthesized rule base back to a minimal rule base. For this purpose, you might have to design a statistical extension to the Morgan rules of logic, or alternatively, a statistical extension to the Karnaugh diagram. If applied to the synthesized rule base of the TicTacToe problem, the result should be a rule base that looks similar to the rule base that had been manually designed to start with.

The solution of this problem is important for the understanding of *automated knowledge generalization* mechanisms.

[R13.2] Heart Monitor

In collaboration with a heart surgeon, establish a set of variables to be monitored during open heart surgery. For a number of months, collect data during actual surgery, and record what actions the surgeon took when and why on the basis of the data that s/he observed. Use the collected physical data as inputs, and the actions of the surgeon as outputs. Use inductive reasoning to come up with a model that behaves similarly to the surgeon.

Design an apparatus which can thereafter observe data during actual surgery, reason about this data on-line in a similar fashion as the surgeon did before, and come up with consulting advise for the same or another surgeon. If asked why the particular advice was given, the apparatus should

be able to relate the decision back to the reasoning performed by the original surgeon. It is important that the model has some power of prediction, i.e., that it can forecast potential problems *before* they actually occur, and provide, together with the actual advice, a verbal prediction of the physical state on which that advice was based.

[R13.3] Anesthesiology

A major problem in anesthesiology is the assessment of the *depth* of the anesthesia. If too little drug is applied, the anesthesia is not sufficiently deep, the patient feels pain, and the risk of a postoperative shock syndrome is enhanced. However, the drug itself is highly toxic, and it is therefore important to apply the minimum amount required for safe surgery.

In collaboration with an anesthesiologist, decide on a number of secondary factors that s/he uses to indirectly assess the depth of anesthesia. Similarly to the heart monitor problem, come up with a consultation system that, on the basis of physical measurements suggests an appropriate dosage of the drug and optimal timing for its administration.

[R13.4] Stock Market

Decide on a number of input parameters (such as the Dow–Jones index) and state variables (such as daily returns of companies) to be monitored. Choose the variables such that the time dependence of the resulting optimal masks is minimized (i.e., for any subset of data over time, the resulting optimal masks should be the same or at least very similar).

Design a SAPS forecasting model with optimal prediction power, and compare the results with an optimized linear regression model.

[R13.5] Causal Structure Characterization

Design a general–purpose routine which is able to derive a causal structure using several different techniques, which can then compare the resulting causal structures with each other, and decide which of the resulting structures is the most likely candidate.