# FURTHER STUDIES ON DISTRIBUTED ADAPTATION IN NEUROMIME NETWORKS

*V. V. GRIFFITH*
*G. H. BOLEN*
*W. E. WUERTHELE*

*GOODYEAR AEROSPACE CORPORATION*

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

# FOREWORD

This study was initiated by the Biomedical Laboratory of the Aerospace Medical Research Laboratories, Aerospace Medical Division, Wright-Patterson Air Force Base, Ohio. The research was conducted by the Avionics and Electronics Research and Development Division of the Goodyear Aerospace Corporation (GAC), Akron, Ohio, under contract AF33(615)-1891. Mr. V.V. Griffith directed the research for the Goodyear Aerospace Corporation. Lieutenant Colonel Jack E. Steele of the Mathematics and Analysis Branch, Biodynamics and Bionics Division, was the contract monitor for the Aerospace Medical Research Laboratories. The work was performed in support of project 7232, "Researchon the Logical Structures and Function of the Nervous System," and task 723203, "Principles of Neural Organization." The research covered herein was performed between June, 1964 and May, 1967.

The authors thank Mr. N.D. Diamantides of Goodyear Aerospace Corporation for his contribution to the Kolmogorov prediction theory section of the report.

This report is filed by Goodyear Aerospace Corporation as report GER-13243.

This technical report has been reviewed and is approved.

J. W. HEIM, Ph. D.
Technical Director
Biomedical Laboratory
Aerospace Medical Research Laboratories

# ABSTRACT

This report describes investigations of networks with adaptive ability distributed through them. It is thought that large-scale adaptive systems can be constructed of adaptive building blocks. These adaptive systems would be flexible in function, reliable and would resist severe damage characteristics of living creatures. Neuron models were tested by interconnecting them into various networks to perform simple control tasks. The test results were evaluated and the evaluation used to improve the theory and the neuron model. The distributed adaptation concept was analyzed from an abstract algebraic approach, using optimal control theory. The combined approach, when studied in depth, contributed to the understanding of the problem. Although the conclusions of this report are at best tentative, one conclusion seems reasonably valid: any required adaptive controller can be built using iterative elements provided only that all terminal segments of optimal trajectories of the process are themselves optimal trajectories, and that the process is controllable and observable.

# TABLE OF CONTENTS

Section — Page

# LIST OF ILLUSTRATIONS

Figure Page

# SECTION I

## INTRODUCTION

This program is an effort to discover whether adaptive systems can be built that are modeled on the known characteristics of neurons, with adaptive ability distributed among the elements. Each neuron receives input signals from other neurons. On the basis of these inputs and outputs, it is postulated that each neuron uses some internal criteria to decide whether or not it should produce an output. The adaptation in a neural net is not centralized, but is distributed throughout the net, with each neuron contributing to the whole.

An aim of the research has been to develop models and techniques that are consistent with known facts from both physiology and behavior theory. The writers know of no physiological evidence that neuron synaptic weights or connections on a lower level are directed by signals from higher centers.

The basic premise of the work presented here is that large-scale adaptive systems with great flexibility can be constructed of adaptive building blocks. Each building block adjusts its own behavior according to some relatively simple rule. Reference 1 points out that systems based on this concept should tend to display the wide range of plasticity, reliability, and ability to operate effectively after severe damage that is displayed by living creatures.

The basic problem has been attacked from four points of view.

(1) An overall system approach using abstract algebra
(2) A simulation approach using optimal control theory for linear systems
(3) Analysis of threshold logic networks
(4) Review of behavior literature and one experiment with animals (company sponsored)

A coherent picture is emerging. Although it is incomplete in many details, the following salient points appear:

(1) There are ways to design a system so that many single, independent decisions made according to definite rules at local points within the system will cause a single complex global (overall) criterion to be met.

(2) A linear system can be controlled optimally by an adaptive controller made of many autonomous identical elements.

(3) Logical decisions of any desired complexity can be made adaptively by a "hierarchy of decisions" (described later).

(4) Observed animal behavior indicates that the decisions made and the methods used for making them resemble the mathematical models that are being developed.

# SECTION II

## GLOBAL PERFORMANCE CRITERIA AND THEIR LOCAL IMPLEMENTATIONS

### A. GENERAL

Early ideas as to how a neuron acts locally to optimize overall system performance are reviewed in this section of the report. The flaws in these early schemes are pointed out, and the progression to more complex ideas of a performance functional is outlined. Present concepts are based on the theory of sequential machines, dynamic programming, and concepts from behavior theory.

Although implementations of the various schemes are mentioned, they are not discussed in detail in this section. Detailed implementations have not yet been developed for the latest concepts. Some implementation requirements that have been considered will be discussed in Section III.

### B. EARLY RESEARCH

At the beginning of the program it was believed that each neuromime could adjust itself by considering solely the time histories of its own inputs and outputs. The adjustment rule that was used required the neuron to attempt to minimize its inputs and output while providing the necessary control of its environment. More formally, a "system power" function, $\mathscr{P}$, was defined by

$$\mathscr{P} = R_o{}^2 + \sum_{1}^{n} R_i{}^2 \tag{1}$$

where $R_o$ and $R_i$, $i = 1, 2, \ldots, n$, are respectively the neuron's output rate and various input rates. It is readily shown that if the appropriate partial derivatives exist and are well-defined, a minimum of $\mathscr{P}$ is obtained when

$$R_o = - \sum_{1}^{n} \frac{\partial R_i}{\partial R_o} R_i \tag{2}$$

For adaptation, the system had to "test" its environment, discover with sufficient accuracy the values of the partial derivatives of Equation 2, and adjust its response accordingly.

The result of the experimentation was to point out a need for a more sophisticated view of the nature of the function to be minimized. Failure of the empirical model under some conditions to operate as initially hoped served to direct the succeeding research, and it is thus worthy of brief discussion.

The elements of the system were electronic units that attempted to mimic certain characteristics of biological neurons. The following characteristics were selected for modeling.

(1) Any particular neuron (excluding sensory neurons) receives inputs from many sources. These inputs are pulse trains of variable repetition rate. The output is a pulse train similar to those appearing as inputs.

(2) Some inputs tend to excite the neuron to emission of output pulses, some tend to inhibit the production of output pulses. In the model, we associated a parameter

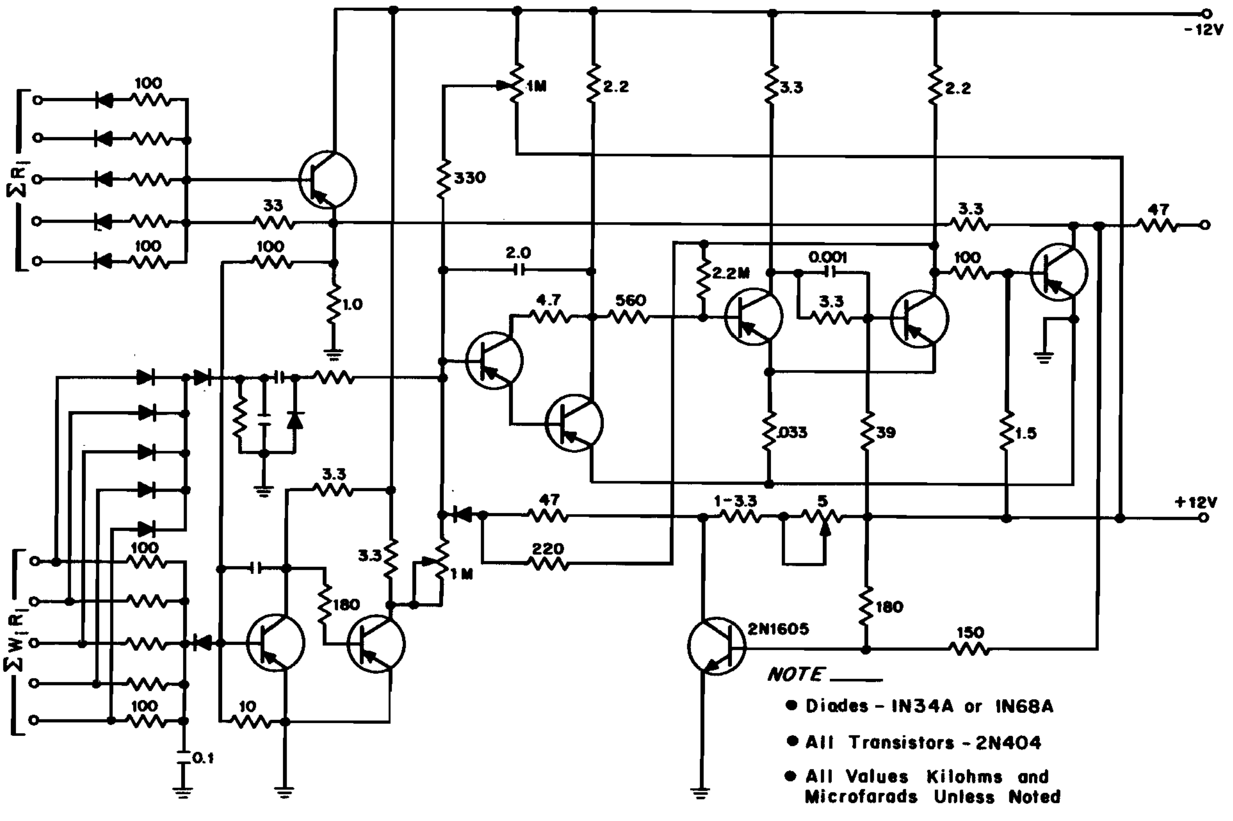
$w_i$, $-1 \leq w_i \leq 1$ with each input to the hyperpolarizing or depolarizing effect of the separate endbulbs. If $w_i > 0$, the ith input is excitatory, and if $w_i < 0$, the ith input is inhibitory. (If $w_i = 0$, the ith input has no effect on the output of the neuron.)

(3) A neuron takes the spatial and temporal sum of its inputs and compares this weighted sum against a threshold. The threshold can vary with time.

(4) Neurons usually produce outputs only when they are receiving inputs whose net effect is excitatory. They occasionally produce spontaneous activity (pacemakers).

For the experiment, elements were constructed resembling Harmon's "neuromimes" (see Figure 1 and Reference 2). The elements displayed the characteristics listed above. In addition, the weights $w_i$ were adjustable, and various empirical rules for adjusting them were investigated.

The scheme seemed to operate more or less satisfactorily in simple networks, as judged by the fact that the signs of the computed synaptic weights became such that control signals of the proper polarity were produced for simple "environments." For more complicated environments, or for more complicated networks of elements, the empirical system very quickly began to show defects. It was determined qualitatively that these defects resulted from two causes. The first was an inability of any individual element to determine whether changes in its input were the result of its own activity or the activity of other elements. The second was the inability of the network elements to consider the long-term effects of their activity.



NOTE

- Diodes - IN34A or IN68A
- All Transistors - 2N404
- All Values Kilohms and Microfarads Unless Noted

Figure 1. Modified Harmon Neuromime Transistor Circuit Used in the Experiments

At this point, a twofold analytical attack on the problem was initiated, in which both discrete and continuous models were investigated. For the discrete models, it was assumed that the environment was a sequential machine or Markov process; for the continuous models it was a linear process. Hand-computed results of the discrete case and analog simulations of the continuous case are discussed later.

The concepts that have been developed since that early research allow one to exhibit the limitations of the earliest scheme in terms of mappings between sets. The assumption was that stimuli map into responses, i. e., there is some function F such that

$$F: S \longrightarrow R,$$

or that this rule holds at least at the level of the individual neuron. Variability of behavior was assumed to result only from changes in the internal parameters of the mapping. These parameters are the "learning" of the neuron. It will be shown later that for various reasons additional "storage" is needed.

Despite the flaws, the moderate successes of the simulation for simple control tasks implied that a more sophisticated rule for adaptation might increase the range of environments the simulation could successfully control. The rules that were investigated next were attempts to retain the basic concept that each neuron adjusts itself, while making the overall system criterion more complicated. The major attempt was directed at control of linear systems. The linearity assumption is an extremely restrictive one. However, optimal control of linear systems has been extensively studied. The investigation of linear systems has given a great deal of insight into the problems the research has attempted to solve, so that the value of the study apparently has not been adversely affected by this restrictiveness.

## C. APPROACH USING OPTIMAL CONTROL OF LINEAR SYSTEMS

Following the early work reported above, a system was developed which was made up of many autonomous elements and which could adapt so as to minimize a global cost functional defined on the system by local computation of an adjustment rule.

Consider Figure 2. The "environment" is assumed to be a linear differential system, described by

$$\dot{Y} = AY + BM + D \tag{3}$$

where Y is an n-vector, M is an m-vector, D is a disturbance vector, A is an n x n matrix of constants, and B is an n x m matrix of constants. Let the "system power" be defined by

$$\mathcal{P} = \frac{1}{2} \int_0^\infty (Y^tY + M^tM)dt,$$

and define the optimum control vector $\check{M}$ to be that control which produces

$$\check{v} = \min_{M \epsilon \Omega_M} \int_0^\infty (Y^tY + M^tM)dt$$

where $\Omega_M$ is the space of all possible control vectors M.

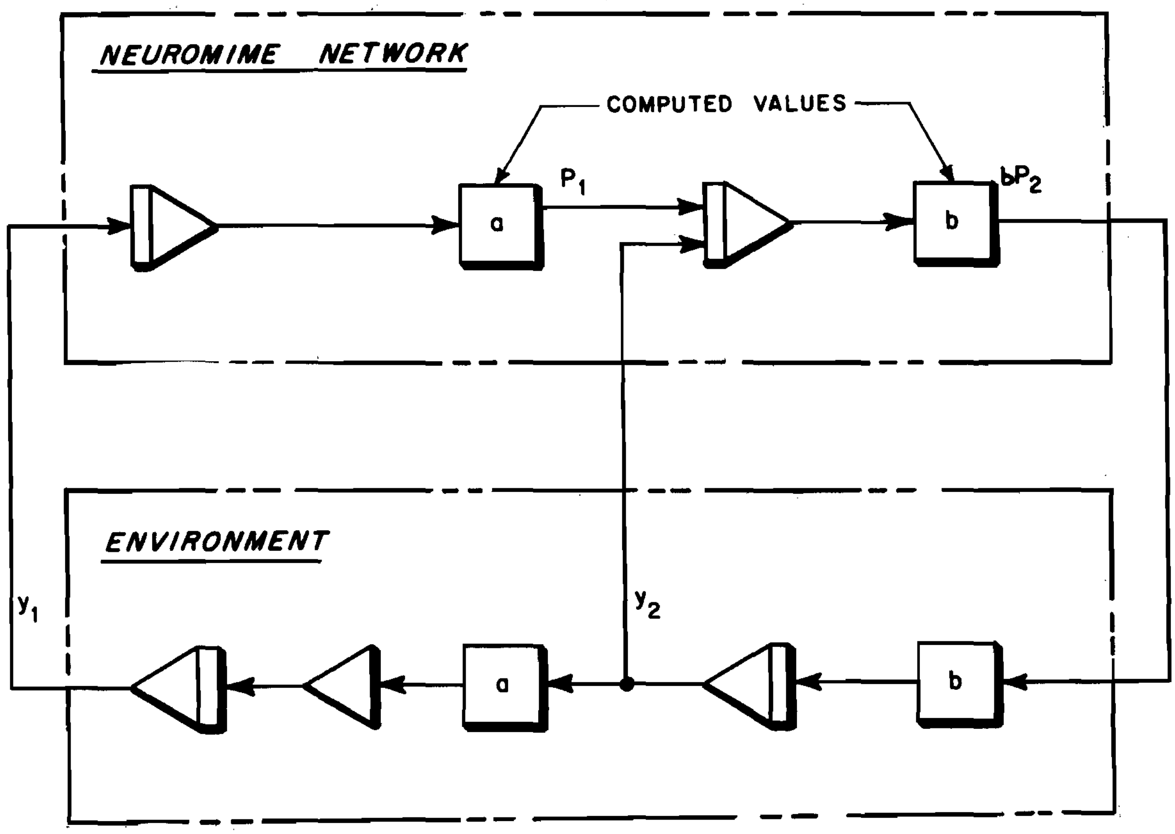


Figure 2. Neuromime Network in a Linear System

Expressed concisely, we wish to find a $\check{v}$ such that

$$\check{v} = v(\check{M})$$

$$= \min_{M \in \Omega_M} \frac{1}{2} \int_0^{\infty} (Y^tY + M^tM)\,dt \qquad (4)$$

subject to the constraint that

$$\dot{Y} = AY + BM.$$

Using the methods of calculus of variations we can write that the Hamiltonian, H, of the above is

$$H = Y^tY + M^tM + p^t (AY + BM)$$

where P = col ($p_1$, $p_2$, . . . , $p_n$) is a vector function to be determined. It is well-known that the functional is minimized for

$$-\dot{P} = \frac{\partial H}{\partial Y}$$

$$\check{M} = M: \frac{\partial H}{\partial M} \text{ is minimum}$$

and we obtain for the specific problem at hand the following three simultaneous equations

$$\dot{P} = -A^tP - Y \tag{5}$$

$$M = -B^tP \tag{6}$$

$$\dot{Y} = AY + BM \tag{7}$$

with boundary conditions

$$Y(0) = Y_0$$

$$\lim_{T \to \infty} P(T) = \text{col}(0, 0, \ldots, 0)$$

$$= 0$$

The result is the classical two-point boundary value problem, which has been studied extensively (Reference 3). Routine manipulation between Equations 3, 5, and 6, assuming D = 0, yields that P = KY, where K is a positive definite, symmetric matrix which satisfies

$$\dot{K} = -I - A^tK + KA + KBB^tK,$$

$$K(T) = 0,$$

a matrix Riccati equation.

Solution of this equation in non-trivial cases is somewhat refractory, but yields to solution by quadratures, or to numerical integration by computer. In the case where the time of integration of the cost functional goes to infinity, K goes to the asymptote given by setting $\dot{K} = 0$ above. The resulting matrix quadratic can be solved by machine methods. A digital computer program was developed to solve a matrix Riccati equation. This program is discussed in detail in Appendix VIII.

The elements of the autonomous-element system that was developed used the components of the equations derived above to adjust their individual parameters. Experimental tests were successful for environments up to second order. A fifth order system was simulated, but tests were not completed because of hardware design problems. The evidence indicated that the system would have operated properly had hardware problems been solved.

The use of a quadratic global cost functional, with the resulting derivation of the Lagrange multipliers, P, leads to consideration of some concepts from behavior theory, particularly the drive-reduction hypothesis. (A discussion of this concept is presented in Appendix III.) It appears that the variables P are, in a sense, predictions of things to come, in the same sense that the state vector Y is a summary of events past.

Note, for example, the relation between Lagrange multipliers, P, and the expected change in cost-to-go to the end of the trajectory. Each of the components of P represents the negative of the derivative of cost-to-go with respect to the corresponding Y. One can maximize some function of the vector P as equivalent to minimizing a function of Y, a saddle-value property of such minimization problems which is well-known. It is also well-known that for stable systems the equation in P is stable in backwards time. This property, coupled with final conditions on P, exhibits its role as a summary of events to come on the optimal trajectory.

The drive-reduction hypothesis leads to similar ideas. One wishes at each instant of time to take that action which will serve to maximize the reduction of drive level. Drives can be acquired, i. e., they can be synthetically produced variables like the components of P. These considerations lead the writers to question whether optimal trajectories could be defined on sequential machines, and whether synthetic variables could be introduced that resemble in some way the variable P. The answer to both questions is affirmative.

## D. OPTIMAL CONTROL OF SEQUENTIAL MACHINES

### 1. Definition of Cost Functional

The following paragraphs show that a cost functional can be defined on a sequential machine in a way that is analogous to the linear system presented earlier, and that such systems can be related to organism-environment interactions.

Before discussing creature-environment interrelations, it is instructive to consider the problem of optimal control of a sequential machine, without regard to the form of the controller.

Consider a sequential machine with state set Z, initial state set $Z_i$, output function $\psi$, and state transition function, t. (See Appendix I.) Assume that some particular state, $z_0$, has been selected as a desired state and that there is a mapping C(z, n) which we will call a cost functional. The system has the following properties:

(1) There exist pairs $(z_i, n_i)$ such that

$$(\forall z_i)\ (\exists n_i)\quad [(z_i \in Z_i) \longrightarrow (n_i \in N) \wedge t(z_i, n_i)] = z_0$$

(2) $C(z_i, n_i)$ exists for all pairs $(z_i, n_i)$, such that (1) holds and is undefined otherwise.

(3) $C(z_i, n_i) \in R^+$ (the non-negative reals) when it exists.

Let $n_i \in N$ be written as

$$n_1, n_2, n_3, \ldots, n_m.$$

We define an optimal trajectory from $z_i$ to be the sequence of pairs

$$\big(z_i, n_1\big),\ \big(t(z_i, n_1),\ n_2\big),\ \big(t(t(z_i, n_1),\ n_2),\ n_3\big),\ \ldots,$$
$$\big(t(t(\ \ldots\ t(z_i, n_1)\ \ldots\ (n_{m-2}),\ n_{m-1}),\ n_m)\big)$$

such that

$$C(z_i, n_i) = \min\ \big(C(z_i, n)\big).$$

Consider the case where at some time, t, the machine occupies state (t). In general, the optimal input to the machine at time t, i. e., that input n(t) for which t (z(t), n(t)) is the t + 1th member of the optimal sequence, cannot be determined from the knowledge of z(t) alone. There is nothing in the definition that prevents the existence of two sequences which are defined to be optimum as follows:

$$\big(z_i, n_1\big),\ \big(t(z_i, n_1),\ n_2\big)\ \ldots,\ \big(t(t(\ \ldots\ t(z_i, n_1)\ \ldots\ ),\ n_n)\big)$$
$$\big(z_j, n_a\big),\ \big(t(z_j, n),\ n\big)\ \ldots,\ \big(t(t(\ \ldots\ t(z_j, n_\alpha)\ \ldots\ n_\omega)\big)$$

for which the first component of the ith pair of the first sequence is equal to the component of the jth pair of the second sequence, while the corresponding second components are not equal.

A large class of interesting systems, however, display the following properties:

(1) Every terminal segment of an optimal trajectory is an optimal trajectory.

(2) A non-negative real number (transition cost) can be assigned to each pair $(z_i, a_j)$, where $a_j \epsilon A$, the generators of N, in such a way that the trajectory functional defined as the sum of the transition costs for each step of a trajectory to $z_0$ is minimized for the optimal trajectories.

(3) 1 $\longleftrightarrow$ 2. To show this we introduce concepts from dynamic programming.

## 2. Dynamic Programming Considerations

Suppose a sequential machine $\Sigma \subseteq N \times V$ is given, together with a set of allowed initial states, $Z_i \subseteq Z$ for the machine. Suppose further that one particular state, $z_0 \epsilon Z$, which is accessible from all initial states, has been selected as a desired state, and that there is a cost associated with each transition in the machine, i.e., there is a mapping $C: Z \times N \longrightarrow R^+$ ($R^+$ is the set of non-negative reals).

We develop in the following paragraphs an algorithm for discovering optimal trajectories, defined as those input sequences which take the machine from any given state to the desired state with least cumulative cost.

The objective is to show that another machine (a controller) exists, which can generate the input strings leading to these optimal trajectories, and to examine the properties of such controllers. We first exhibit an algorithm for assigning to each state a pair (b, c) where c is the minimum cost for reaching the desired state and b is the input symbol which will transfer the machine from the given state to a state which is the succeeding state along the optimal trajectory to the desired state.

(1) The Algorithm - Initial Step. Assign to the desired state, $z_0$, the trajectory pair $(b_0, c_0) = (\lambda, 0)$. $\lambda$ is the empty string.

(2) Iterative Step. Consider each member of $z_p$ of the set of states $Z_p$ that have at least one transition to a state which has an assigned trajectory pair. Let the set of those states that have assigned trajectory pairs be called $Z_a$. Any given $z_{pi} \epsilon Z_p$ either has an assigned trajectory pair or it does not. Order these states by any convenient scheme. Taking them in order, assign to each $z_{pi}$ the pair $(b_{pi}, c_{pi})$ where

$$c_{pi} = \min_{\forall n} \left[ C(z_{pi}, n)^{+} \; c_{ai} \right]$$

where $c_{ai}$ is the trajectory cost associated with the successor state $t(z_{pi}, n)$. The input that yields minimum cost is $n_{pi}$; $c_{pi}$ is undefined if $c_{ai}$ is not defined.

(3) Repeat Step 2 until no further assignments or replacements can be made.

It can be proved by induction that the algorithm yields optimal trajectories and terminates for a finite set of states. We assert that there is a minimal length optimal trajectory from any given state which does not occupy any state twice. The absolute length trajectory can traverse at most every state in the machine.

By inspection, the algorithm finds all optimal trajectories of length one on the first iteration. Assume that at the end of the nth iteration, all least-cost trajectories of length n or less have been found, and further that the assignments and replacements yielded exactly those least-cost trajectories that are of the length n. From the algorithm, a pair will be assigned or replaced for a given state during a particular iteration only if one of its immediate successors has had a pair assigned or replaced in the previous iteration. The new trajectory from any node that receives an assignment or replacement on the n + first iteration is therefore of length n + 1. No other path of length n + 1 or less from that node is superior, or it would have been selected by the algorithm. Since there is a maximal length of optimal path, the process must terminate.

Since much of the experimental work described later is devoted to the continuous or discrete cases where the number of states is not finite, it should be pointed out that to the writers' knowledge no algorithm exists for finding optimal trajectories in the general case. Halkin (Reference 4) and others have found algorithms for cases satisfying certain convexity conditions on the constraints, however.

From the termination proof we also see that each terminal segment of an optimal trajectory is an optimal trajectory. We now show that if every terminal segment of every optimal trajectory is an optimal trajectory, then appropriate transition costs can be defined.

Assign a transition cost of zero to every transition in every optimal trajectory. Assign a transition cost of one to every other transition. Clearly the transition takes on a non-zero value for a non-optimal trajectory.

### 3. The Optimal Controller

Given a sequential machine (or process) with a set of initial states, a desired state, and a set of optimal trajectories, we wish to find a controller which drives the process along the optimal trajectories. The following paragraphs show that the controller is (as would be expected) closely related to the process. There are a number of possibilities: (1) the case of the generalized optimal trajectory versus the case where each terminal segment of every optimal trajectory is an optimal trajectory; (2) the case where the initial state of the process is known versus the case where it is not; and (3) the case where outputs from the process are accessible versus the case where they are not.

Our attention will be primarily confined to the case where terminal segments are optimal and outputs from the process are accessible (but may not be one-to-one with the process states). The initial states may or may not be known. In the more general case, where terminal segments of optimal trajectories may be non-optimal, a moment's reflection shows that the controller must have as many initial states as there are initial states of the process. The total number of states can be as large as the sum of the lengths of all optimal trajectories. The writers were unable to discover any interesting properties of this case. In contrast, consider the case where all terminal segments of all optimal trajectories are optimal trajectories. As was pointed out earlier, this requirement can result in assignment of a pair to each state; one element of the pair is the optimal transition out of that state, the other is the cumulative cost to the desired state.

We show first that there exists a controller that is a Moore machine and that has exactly as many states as the process. Formally, we have the process, $\Sigma_p \subseteq N \times V$, and

$$t_p: Z_p \times N \longrightarrow Z_p,$$

$$\psi_p: Z_p \times N \longrightarrow V$$

Assume a controller $\Sigma_c \subseteq V \times N$ such that

$$t_c: Z_c \times V \longrightarrow Z_c$$

and

$$\psi_p: Z_c \longrightarrow N$$

with the initial state sets $Z_{pi}$, $Z_{ci}$, and a desired state of the process $z_{po}$ together with a set of pairs, U(n, c) defining optimal trajectories. We need define $t_c$ only for those pairs that are necessary to produce optimal trajectories. Consider first an "open loop" controller. Let $t_c$ be independent of V, i.e.,

$$t_c: Z_c \rightarrow Z_c \quad \text{(a generator)}$$

Let the states $z_c \in Z_c$ be paired one-to-one in any desired way with the states $z_p \in Z_p$, i.e., there exists a mapping

$$f: Z_p \rightarrow Z_c$$

and $f^{-1}$ exists. Define $\psi_c$ such that

$$\psi_c(f(z_{pj})) = n_j,$$

where $n_j$ is the first component of the pair $(n_j, c_j)$ assigned to $z_{pi}$. This is clearly always possible. Let $t_c$ be chosen in such a way that

$$t_c(f(z_{pj})) = f(t_p(z_{pj}, n_j)),$$

which is always clearly possible.

From the construction, $\Sigma_c$ is an optimal controller, in the sense that given an initial state of the process $z_{pi}$, there is a corresponding initial state of the controller $z_{ci} = f(z_{pi})$, such that the output sequence of the controller starting in state $z_{ci}$ is the sequence that will drive the process from state $z_{pi}$ to the state $z_0$ via the optimal trajectory.

We now show that it is possible to find an optimal open loop controller with fewer states than the process it controls. It is obvious that such would be the case if the process were not reduced and connected, i.e., minimal. We therefore assume a minimal process. Also, we assume that there is an optimal trajectory terminal segment from each state in the machine, so that no states can be immediately omitted from the controller on that account. We show that under these conditions the controller can still have fewer states than the process. The result follows immediately from the conventional reduction process for sequential machines. Consider first the process. For each input-state pair there is associated a next state and an output. Two states can be merged if for every input their outputs are identical and their next states lie in the same equivalence class. Nelson describes the algorithm thoroughly (Reference 5). Assume, therefore, that the process contains two states whose outputs are not identical. Assume that the optimal trajectory from each of these states goes to the same successor state, and that the same input symbol produces the optimal transition for each of the states.

Consider now the two states of the controller corresponding to those two states. Clearly they produce the same output symbol and have the same successor state. They can therefore be merged.

We have so far confined ourselves to Moore machines. The following paragraphs show that a Mealy machine exists which, under certain conditions, is an optimal closed-loop controller. Further, such a controller may have fewer states than a minimal Moore controller.

We consider the simple case where the outputs from the process are such that

$$\psi_n(z_{pj}) = v_k$$

and has an inverse, i.e.,

$$\psi_n^{-1}(v_b) = z_{pj}$$

exists and is unique. Each output from the process can be identified with only one process state. The process $\Sigma_p$ is described by

$$\psi_p: Z_p \times N \rightarrow V$$

$$t_p: Z_p \times N \rightarrow Z_p$$

and the controller $\Sigma_c$, is described by

$$\psi_c: Z_c \times V \rightarrow N$$

$$t_c: Z_c \times V \rightarrow Z_c$$

where $t_p$, $\psi_c$ can be constructed as follows. Let f: $Z_p \rightarrow Z_c$ be one-to-one. Let $\psi_c$ be such that

$$\psi_c(f(z_{pj}), \psi_n(z_{pj})) = n_j$$

where $n_j$ produces the optimal transition out of process state $z_{pj}$. Let

$$t_c(f(z_{pj}), \psi_n(z_{pj})) = f\left[t_p(z_{pj}, n_j)\right]$$

Note that $t_c$ is constrained for only some of the pairs in $Z_c \times V$. The others can be selected arbitrarily.

Clearly, $\Sigma_c$ is an optimal controller. To show that an optimal controller with fewer states may exist, we consider the following case.

Consider first the process. Let it contain exactly two states, $z_{p1}$ and $z_{p2}$, for which the optimal trajectories occupy the same successor state, $z_{p3}$, and for which the optimal trajectories require respectively the input symbols $n_1$ and $n_2$.

Consider now the controller. Let the states corresponding to $z_{p1}$ and $z_{p2}$ be, respectively, $z_{c1}$ and $z_{c2}$, with their common successor denoted by $z_{c3}$. We have that

$$t(z_{c1}, \psi_p(z_{p1})) = z_{c3}$$

$$t(z_{c2}, \psi_p(z_{p2})) = z_{c3}$$

It was originally assumed that

$$\psi_p(z_{p1}) \neq \psi_p(z_{p2}).$$

These states can be merged, while with a Moore controller they cannot.

The question of the complete description of the mapping $t_c$: $Z_c \times V \rightarrow Z_c$ has so far been deferred. Only those transitions required for the optimal trajectories have been discussed. The remaining transitions can sometimes be assigned in such a way as to identify an unknown initial state. The question of process state identification is discussed later.

Before continuing with the relation of the results to iterated element theory, it is perhaps useful to consider some of the assumptions implicit in the construction above. From a control theory point of view, we have the following results for sequential machine processes and controllers.

(1) If a process is controllable to a given final state, then a controller exists to drive it there.

(2) If optimal trajectories exist, then a controller exists that will drive the process along them.

(3) The controller may have fewer states than the process it controls.

(4) The existence at each process step of a number which is monotone decreasing only along optimal trajectories implies that all terminal segments of all optimal trajectories are optimal trajectories, and vice versa.

We have assumed that the process is noiseless, i. e., its transitions are determined uniquely by the initial state and the signals from the controller. We have also assumed that the initial state of the process is known and the corresponding initial state for the controller is selected.

If the cost-to-the-desired state is equated formally to the concept of drive level in behavior theory, it follows that the drive-reduction hypothesis requires that each terminal segment of an optimal response chain be an optimal response chain. To the writers' knowledge,. the best evidence that this is so is the phenomenon of chaining itself, in that creature "trajectories" are often acquired backwards. Several facets of the system theoretical problem that are pertinent to creature behavior have not been considered, however.

The assertion is that a creature-environment relation is modeled by a system which devolves in state (Appendix I) in such a way that after a long time only optimal (or "near-optimal") trajectories are traversed. The possible methods for it to devolve in state have not been discussed; we have only considered terminal behavior.

## 4. Comparison of Algebraic and Linear Systems

Consider now the analogy between the abstract algebra formulation and the linear system which was discussed earlier. We have

| | General | Specific |
|---|---|---|
| (1) | $\psi_p: R \times Z_p \longrightarrow S$ | $\dot{Y} = AY + BM$ |
| (2) | $t_p: R \times Z_p \longrightarrow Z_p$ , | $\dot{Y} = AY + BM$<br>(state and output are identical in this case) |
| (3) | $t_c: S \times Z_c \longrightarrow Z_c$<br>(we can identify $Z_c$ and D) | $\dot{P} = -A^tP - Y$ |
| (4) | M: $Z_c \longrightarrow R$ | $-B^tP = M$ |

The parallelism of the construction shows that optimization of trajectories, in general, leads to the generation of "intervening variables" - a term borrowed from behavior theory. The next section shows how these intervening variables can be of use in the local computations at the level of individual neurons or neuron-like elements.

# SECTION III

# LOCAL RULES AND THEIR IMPLEMENTATION

This section discusses methods for generating the local rules for neuron adjustment. The adjustment rules for each scheme that has been investigated are discussed. Detailed implementations of the most recent concepts have not been completed. However, results of some aspects of the implementation that have been examined are presented.

## A. EARLY WORK

The earliest scheme that was investigated established only the algebraic sign of a particular synaptic weight. Each neuron adjusted each endbulb according to

$$\dot{w}_i = -k R_0 \operatorname{sgn} f(R_i),$$

where k was a positive constant and f was a function of $R_i$ that strongly reflected the influence of the first derivative.

Two concepts were considered that incorporated the adjustment rule. In Concept 1, the neuron model adjusted the synaptic weights of its inputs. If a neuron model had 1000 inputs, it would compute 1000 synaptic weights. In Concept 2, the neuron model adjusted the synaptic weight associated with its own output. If the neuron model synapsed on 1000 other neuron models or had 1000 inputs, it would compute only one synaptic weight. Previous research sponsored by GAC used Concept 1. It was believed, however, that Concept 2 was more plausible physiologically. It seemed reasonable that a neuron could more readily adjust the effects of its own endbulbs than the endbulbs of other neurons. Also, research by Dale (Reference 6) has indicated that all of the endbulbs of any particular neuron are either excitatory or inhibitory. The most recent work uses a combination of both concepts, together with considerably more complex rules for adjusting the weight. A more complete discussion of the physiological considerations is given in Appendix II.

## B. THE LINEAR SYSTEM

In the previous section it was pointed out that optimal control of a linear process with quadratic cost functional defined over all time led to the following set of equations:

$$\dot{Y} = AY + BM \tag{8}$$

$$\dot{P} = -A^tP - Y \tag{9}$$

$$P = KY \tag{10}$$

$$M = -BB^tP \tag{11}$$

$$Y(0) = Y_0 \tag{12}$$

$$P(\infty) = 0 \tag{13}$$

Equations 8, 10, and 11 can be used to define a feedback optimal controller for the system of Equation 8, assuming K and B are known. It is a linear system as is shown in Figure 3. However, the system has no adaptive ability, which can be introduced as follows:

Let the $k_{ij}$ be initially unknown. Instead, parameters $\kappa_{ij}$, which are the best estimates available, are inserted into the network. For simplicity, assume temporarily that A is 0. We have that

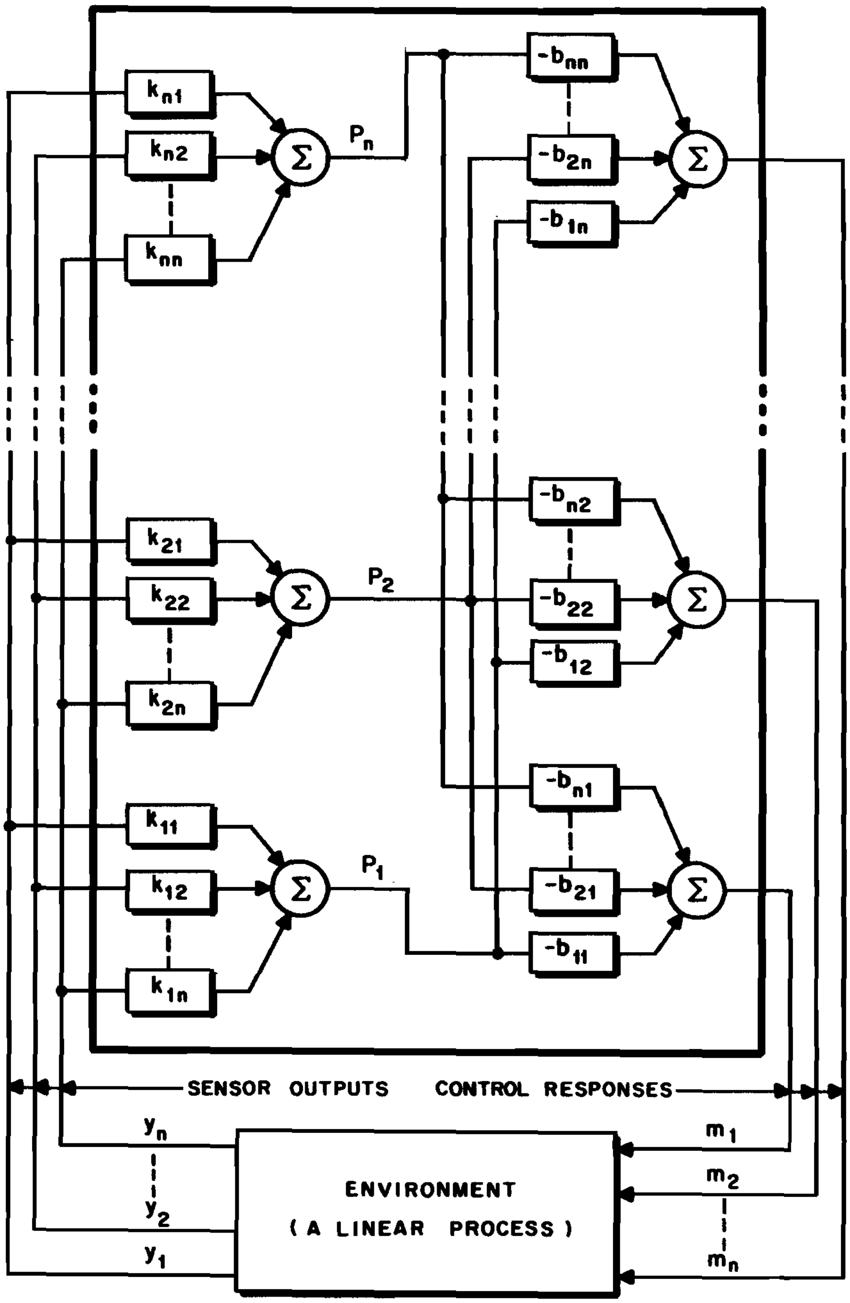


Figure 3. Feedback Controller for a Linear Process

$$P = \mathcal{K} Y; \qquad \mathcal{K} = [\kappa_{ij}] . \tag{14}$$

One obtains from Equation 14 for A = 0 that

$$\mathcal{K}\dot{Y} + Y = \epsilon_1,$$

where $\epsilon_1$ is an error signal. The error signal can be used to revise the values of the $\kappa_{ij}$. The rule used in the experimentation

$$\dot{\kappa}_{ij} = - \frac{\partial \epsilon_1}{\partial \kappa_{ij}} \operatorname{sgn}[\epsilon_1]_i = y_j \operatorname{sgn}[\epsilon_1]_i .$$

The rule is discussed in Appendix VII. Figure 4 illustrates the method. Note that each element uses only information found in its own inputs to adjust its input weights.

It was assumed above that B was known. Assume now that, like K, B is only approximately known, and that estimates, $\beta_{ij}$, are originally inserted. The equation

$$\dot{Y} - BM = D$$

becomes

$$\dot{Y} - \beta M = \epsilon_2; \qquad \beta = [\beta_{ij}]$$

when the disturbance is zero.

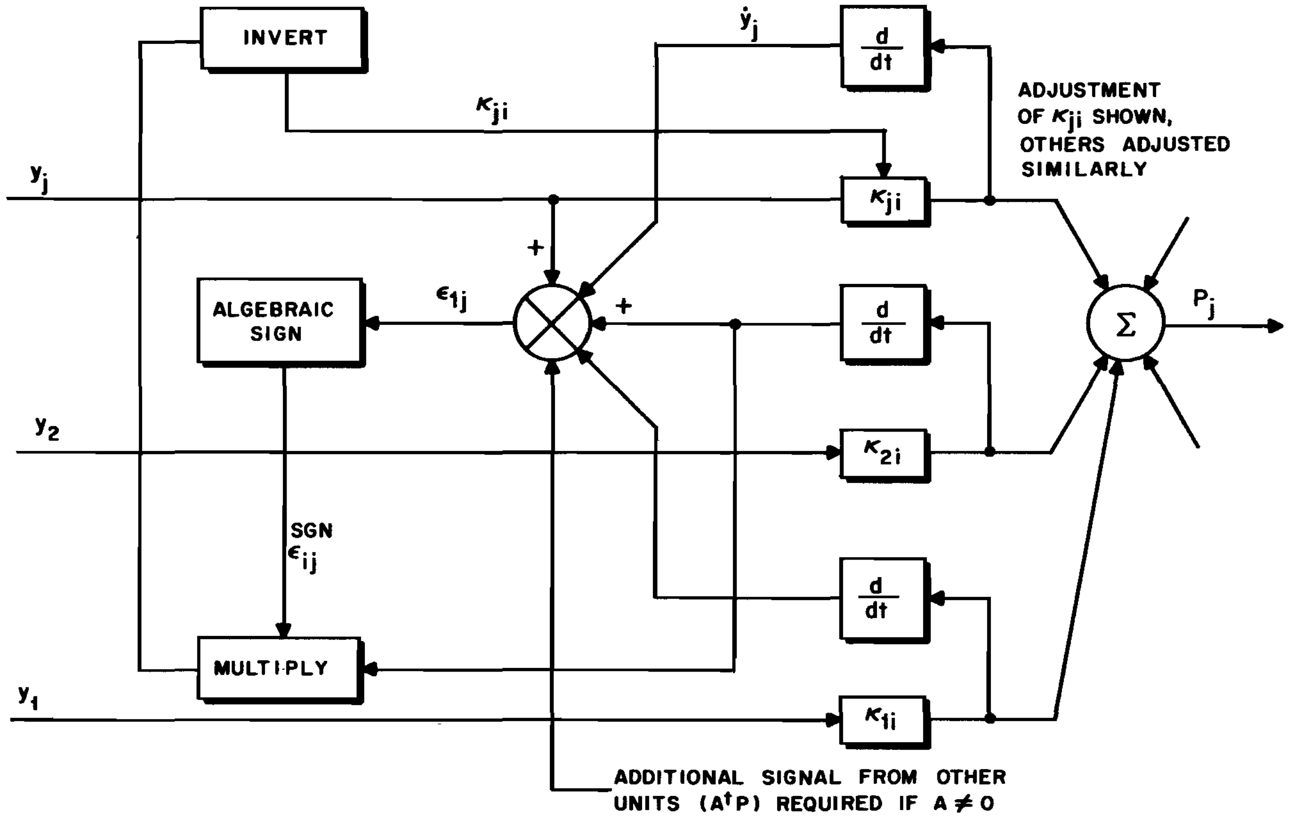


Figure 4. Method of Adjusting the K Optimal Control Parameter

The values $\beta_{ij}$ can be adjusted by a method similar to that used for adjusting $\kappa$. Figure 5 shows the method. The output signals $m_i$ are multiplied by the same coefficients $\beta_{ij}$ that have been computed as weights on the output of the element. The $\beta_{ij}$ are altered by equations of the form

$$\dot{\beta}_{ij} = -m_i \operatorname{sgn} \left[\epsilon_2\right]_j \quad .$$

Again, each element uses for computation only signals which are directly accessible to it.

From the description of the simplified system it is now possible to see how an element to handle a more general case would be constructed. A block diagram is shown in Figure 6. The parameters $\alpha_{ij}$ are adjusted in a method analogous to that already presented for the $\beta_{ij}$, using the equations of the system in their complete form ($A \neq 0$).

The above has described only one method for adjusting the parameters. Others, perhaps perturbations or correlation computations, might be suitable for some sort of iterative computation, depending on the exact system to be controlled and the statistics of the disturbances expected. The system that was instrumented for experimentation assumed that disturbances occurred rarely. Convergence conditions for the rule are discussed in Appendix IX.

The important point that has emerged from the investigation is that for this linear system there are two separate error equations, both of which must be satisfied by the individual elements. It seems intuitively likely that at least two analogous equations must be instrumented in the general case.

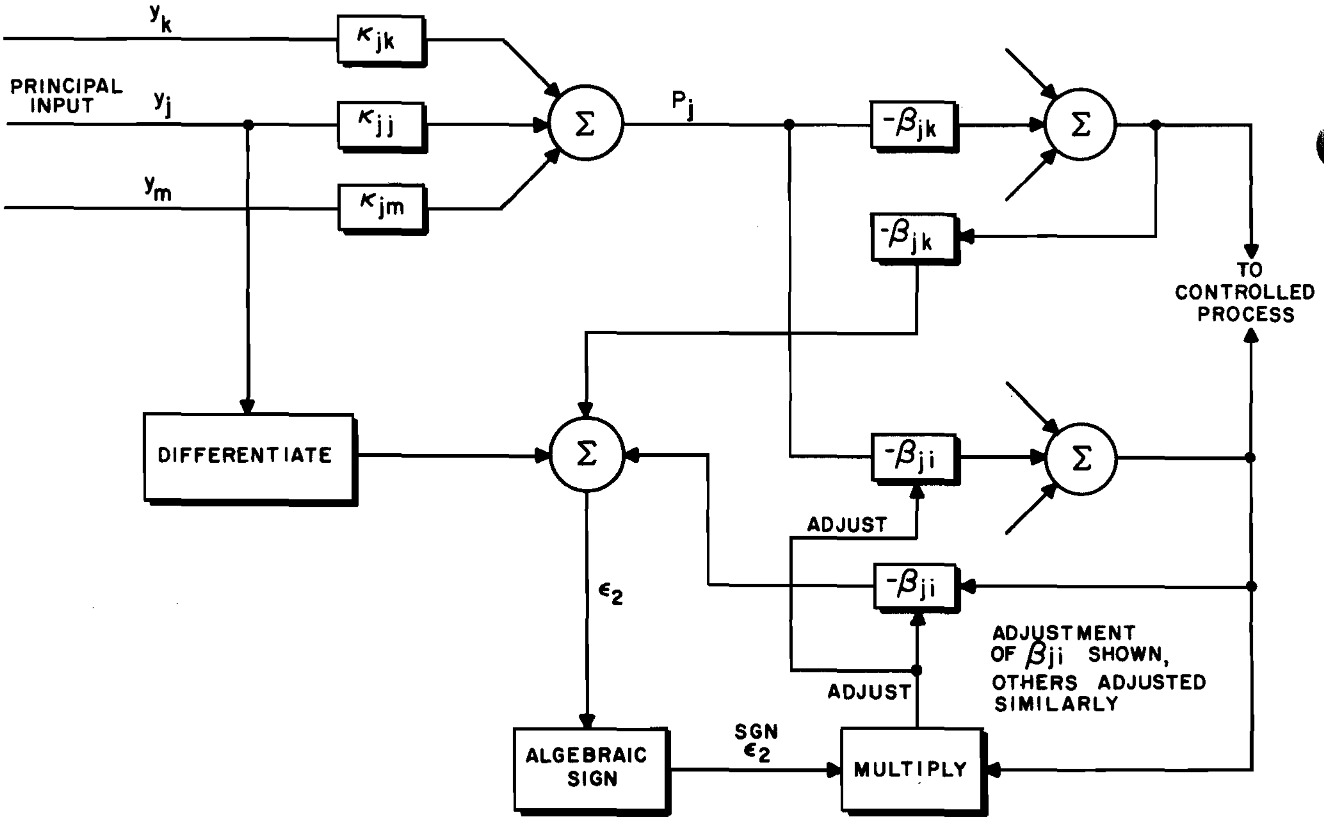


Figure 5. Method of Adjusting the $\beta$ Parameter

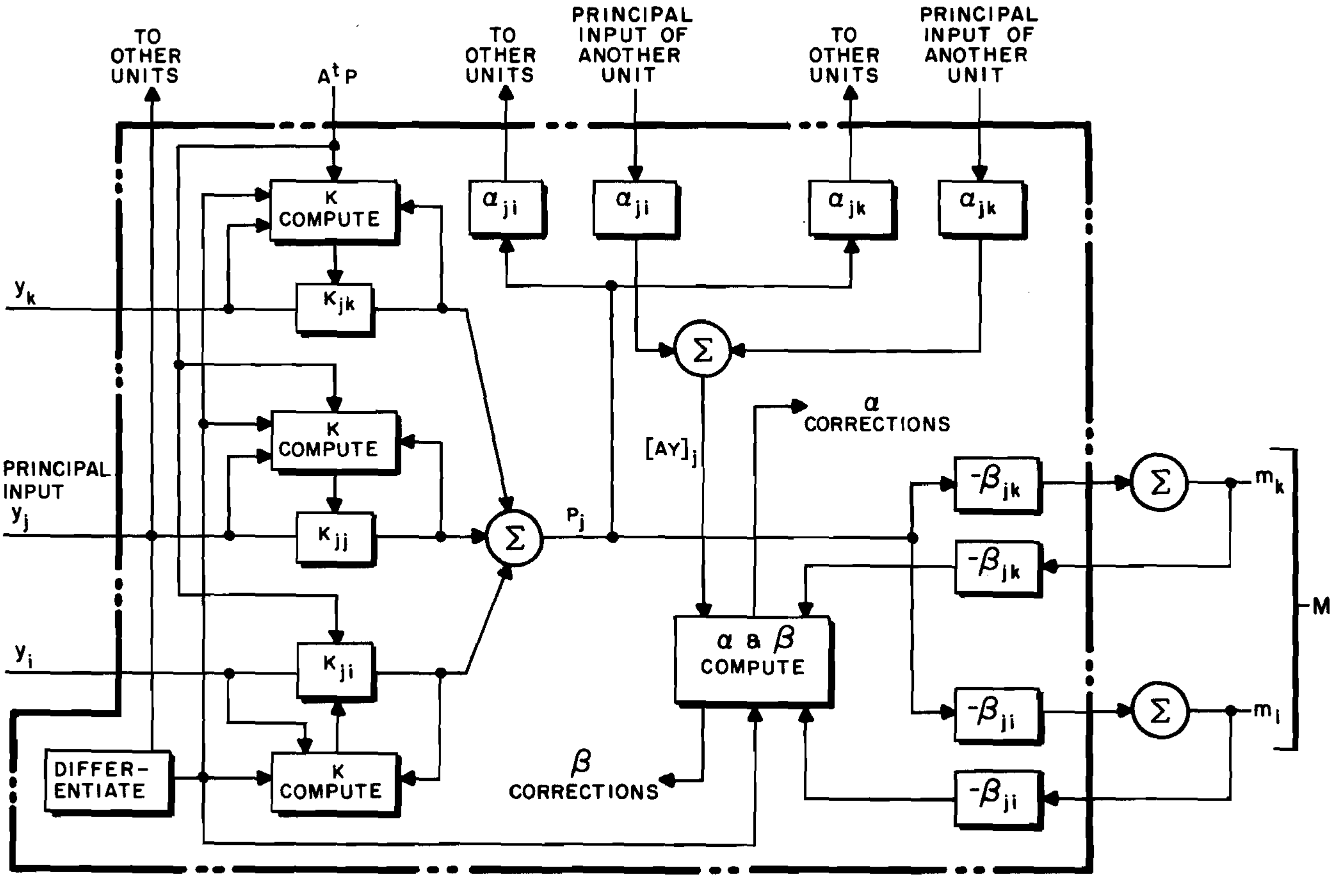


Figure 6. Block Diagram of Adjustment Mechanism of a Single Neuron Model

The equations have a physical interpretation. The first equation ($\epsilon_1$) detects errors in the transformation of environment states, Y, to equivalent controller states, P. The second, $\epsilon_2$, detects errors in the controller's internal model of the environment.

As is discussed in the next section, more generalized linear systems than the one that has been investigated lead to additional requirements on the $\epsilon_1$, $\epsilon_2$ equations. We defer this question for the moment, however, and consider a method that has been developed for implementation of appropriate mappings in the sequential machine case.

## C. HIERARCHY OF THRESHOLD ELEMENTS

In the linear case, one obtains that

$$P = KY,$$

$$M = -B^tP$$

two linear transformations. In the more general case, one would have

$$P = K(Y) \text{ and } M = -B(P),$$

i. e., the matrices would go over to generalized functions. A method has been developed for generating components of P and M. This method, which assumes the problem variables are

Boolean variables, appears to be amenable to adjustment by an error equation. The implementation needs only to be presented with information as to whether its output is right or wrong, and it will approach the correct behavior. Convergence in all cases has not yet been proved, but seems intuitively likely.

Consider an element which receives Boolean inputs from n sources. Define a vector $A = \text{col}(a_1, a_2, \ldots, a_n)$, where $a_i$ takes on the values 1 or 0 according to whether the corresponding input is energized or deenergized, respectively. Associated with the element are n weights, $w_1, w_2, \ldots, w_n$, and a threshold, T. The element produces an output iff

$$\langle W, A \rangle \geq T$$

where W is the vector $\text{col}(w_1, w_2, \ldots, w_n)$, and the notation $\langle W, A \rangle$ represents the inner product of W and A. The equation above with the equality holding and continuous A defines a plane in an n-dimensional (Euclidean) space. It is immediate that all points A which satisfy it must lie in the closed half-space "beyond" the separating plane. The set I = A of admissible vectors is such that the vectors A have components equal to either 1 or 0, and form the vertices of an n-cube in the space, with one vertex at the origin. It follows that only those functions can be generated by a threshold logic element in which the minterms of the function, considered as vertices of an n-cube, can be separated from the minterms of the function complement by a single plane passed through the cube. We define a linearly separable logic function as one which meets this criterion.

In the following, we restrict our attention to logic functions that are zero for the zero input vector. It is easily shown that this restriction introduces no loss in generality, since it can always be met by replacing one or more input variables with their complements. The restriction is equivalent to restricting the threshold to positive values.

A slightly different geometrical interpretation of the problem lends some insight into the algorithm to be presented. Consider the space $\Omega_W$ of all possible vectors W. Each admissible vector A, (except the zero vector) together with the threshold T, defines a plane in the solution space, $\Omega_W$, i.e., given any

$$A_0 \in \{A\}, \quad A_0 \neq 0,$$

the set of

$$\{W \mid \langle W, A_0 \rangle = T\}$$

is a plane in $\Omega_W$. Each possible non-zero input vector, A, defines a plane in the solution space which is perpendicular to the vector from the origin to the point $A_i$. The distance from the origin to the plane is determined by the threshold. No two of the planes are parallel. Selection of a particular logic function for the instrumentation corresponds to selecting a set $R = B_i \subset I$, $i = 1, 2, \ldots, m$, of vertices of the n-cube for which the threshold is to be reached or exceeded. The selection defines a set of inequalities

$$\begin{array}{l} \langle W, B_1 \rangle \geq T \\ \langle W, B_2 \rangle \geq T \\ \vdots \\ \langle W, B_m \rangle \geq T \end{array} \qquad (15)$$

and also the set of opposite inequalities

$$\begin{array}{l} \langle W, C_1 \rangle < T \\ \langle W, C_2 \rangle < T \\ \vdots \\ \langle W, C_{n-m-1} \rangle < T \end{array} \qquad (16)$$

where C is I-R. These $2^n-1$ inequalities can be interpreted geometrically as dictating that W must lie in a particular one of the two half-spaces defined by each of the $2^n-1$ non-zero input vectors. A logic function which is not linearly separable arises when the $2^n-1$ constraints are not consistent. As an example, consider the function

$$f = (a v b) \wedge (a \wedge b'),$$

the "exclusive-or" function.

The three corresponding inequalities, defining three planes in the solution space (Figure 7) are (letting T = 1)

$$\langle W, (1,0)\rangle \geq 1$$
$$\langle W, (0,1)\rangle \geq 1$$
$$\langle W, (1,1)\rangle < 1$$

The inconsistency can be seen from the figure, and also from the fact that the left-hand member of the third inequality is the sum of the left-hand members of the first two.

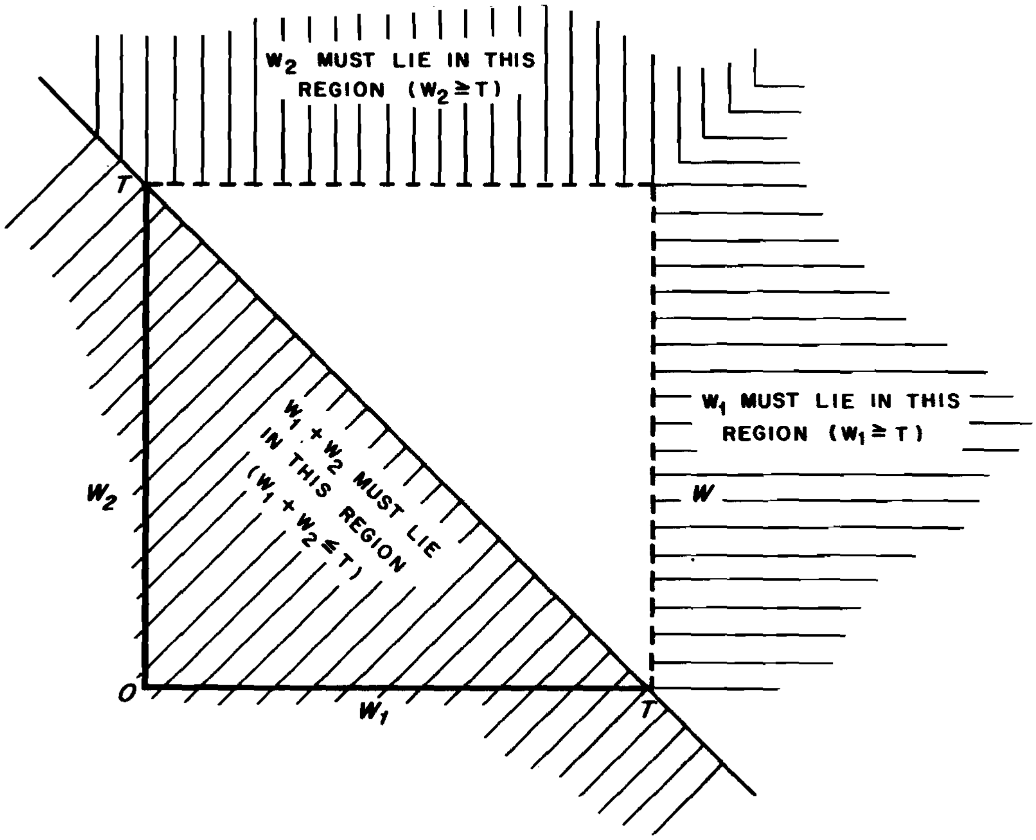

$W_2$ MUST LIE IN THIS REGION ($W_2 \geq T$)
T
$W_1$ MUST LIE IN THIS REGION ($W_1 \geq T$)
$W$
$W_1 + W_2$ MUST LIE IN THIS REGION ($W_1 + W_2 \leq T$)
$W_2$
O
$W_1$
T

Figure 7. Three Planes in the Solution Space

## D. HIERARCHICAL NETWORKS

The intuitive notion which is instrumented by a hierarchy of threshold elements (Figure 8) is as follows. Weights for the lowest element (labeled No. 1 in the figure) are first chosen so that the lowest element, if it were operating by itself, would fire on all inputs that are supposed to produce an output, even though it may also fire on some inputs that are not supposed to produce outputs. The next higher element in the hierarchy is then adjusted to correct all the lowest element's mistakes, in that it prevents the lowest one from firing when no output is supposed to be produced. Its input to the lowest element is inhibitory, i.e., the weight, $w_{21}$, associated with it is negative. The second level element may also make mistakes, in that if it were operating without inputs from the next higher element it might prevent the lowest element from firing when the network should produce an output. The third-level element is adjusted to correct all the second-level element's mistakes, and so on. The following paragraphs prove that a hierarchy with a finite number of levels can always be found to produce any given logic function of n variables.

Given a logic function, $\mathcal{L}$, defining two sets, R and I-R, where R does not contain the zero vector, we consider the inequalities of Equations 15 and 16 above. Select a vector $W_1$ which satisfies all the inequalities of Equation 15. We note in passing that if all inequalities of Equation 16 are also satisfied, the function would be linearly separable.

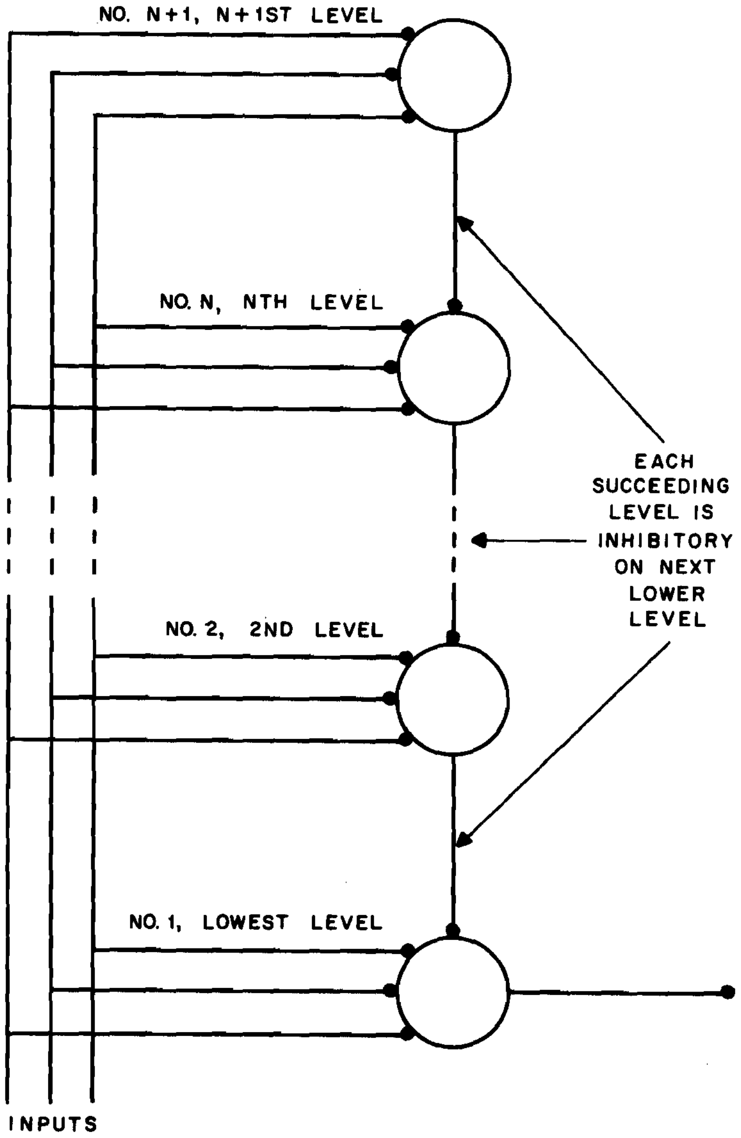


Figure 8. Hierarchy of Threshold Elements

The vector W clearly can be chosen so that equality holds in at least one of the Equation 16 set. Let an input vector for which equality holds be labeled $B_a$. Assume some collection of Equation 16 are not satisfied, in that vectors $E_a$, $E_b$, . . . , $E_p$, which are supposed not to produce outputs, cause the threshold to be reached or exceeded when their inner product is taken with $W_1$. It will be shown that a vector $W_2$ can be chosen which causes the second level threshold to be exceeded for all vectors $E_a$, $E_b$, . . . , $E_p$, and for which at least the vector $B_a$ will be below the threshold. We consider two cases.

Case I - The inner products of the vectors $E_a$, $E_b$ . . . , $E_p$, with $W_1$ all exceed the threshold by at least an amount $\delta$. Select $W_2$ to be

$$W_1\text{-col}(\delta/n+1),\ \delta/n+1,\ \delta/n+1,\ \ldots,\ \delta/n+1).$$

Since the components of any input are 0 or 1, it is seen that for any non-zero $A_i$

$$\frac{\delta}{n+1} \leq \left\langle A_i,\ \text{col}\left(\frac{\delta}{n+1},\frac{\delta}{n+1},\frac{\delta}{n+1},\ldots,\frac{\delta}{n+1}\right)\right\rangle \geq \frac{n\delta}{n+1}$$

and the necessary conditions are satisfied.

Case II - The inner products of some subset of the vectors $E_a$, $E_b$, . . . , $E_p$, are exactly equal to the threshold. Since no two input vectors are parallel, all of the vectors, $E_a$, $E_b$, . . . , $E_p$, have components orthogonal to $B_a$. Form $W_2$ by first subtracting some

$$\delta_i B_a,\ \delta > 0,$$

from $W_1$ and then adding some

$$\delta_2 \cdot P_1,\quad \delta_3 \cdot P_2,$$

etc, where $P_1$, $P_2$, . . . are orthogonal to $B_a$. Clearly, $\delta_1$, $\delta_2$, . . . , $\delta_{p+1}$ can be chosen to satisfy the requirement.

By an identical argument, the third-level vector can be chosen to allow the second-level element to fire on at least one of the inputs $E_a$, $E_b$, . . . , $E_p$. The fourth-level can allow the third-level to fire on $B_b$, etc.

At each level the threshold element corrects all the mistakes of the previous level while allowing the previous level to fire on at least one input for which it operates correctly. We have

$$f_1 = (B_a \vee B_b \vee \ldots \vee B_m \vee E_a \vee \ldots \vee E_p) \quad \text{(lowest level)}$$

$$f_2 = (E_a \vee E_b \vee \ldots \vee E_p \vee P_1)\bar{B}_a \quad \text{(second level)}$$

where $P_1$ is some proper subset of the $B_i$

$$f_3 = (P_1 \vee Q_1)\bar{E}_a \quad \text{(third level)}$$

where $Q_1$ is some proper subset of the $E_i$

$$f_4 = (Q_1 \vee P_2)\bar{B}_b \quad \text{(fourth level)}$$

where $P_2$ is some proper subset of $P_1$ and $B_b \in P_1$, and so on.

The output of the network is given by

$$f = f_1 \sim (f_2 \sim (f_3 \sim \ldots (f_{2n}) \ldots )),$$

which simplifies to

$$f_1\bar{f}_2 \vee f_1f_3\bar{f}_4 \vee f_1f_3f_5\bar{f}_6 \vee \ldots \vee f_1f_3f_5 \ldots \bar{f}_{2n} \ .$$

Substitution yields

$$f_1\bar{f}_2 = B_a \vee (R - P_1)$$

$$f_1f_3\bar{f}_4 = (B_a \vee B_b \vee \ldots \vee B_n \vee E_a \vee \ldots \vee E_p)$$

$$(P_1 \vee Q_1)\bar{E}_a \cdot (\bar{P}_2\bar{Q}_1 \vee B_b) \ .$$

Since $P_1 \subseteq R$ and $Q_1 \subseteq E_a \vee \ldots \vee E_p$,

$$f_1f_3\bar{f}_4 = (P_1 \vee Q_1)\bar{E}_a \ (\bar{P}_2 \ \bar{Q}_1 \vee B_b).$$

We note that $B_b \in P_1$, which yields

$$f_1f_3\bar{f}_4 = (P_1 - P_2) \ \bar{Q}_2\bar{E}_a \ ,$$

and that $P_1$ and $Q_1$ are disjunct, yielding finally

$$f_1f_3\bar{f}_4 = (P_1 - P_2) \ .$$

Similarly

$$f_1f_3f_5\bar{f}_6 = (P_2 - P_3) \ ,$$

and so on, yielding that

$$f = (R - P_1) \vee (P_1 - P_2) \vee (P_2 - P_3) \ldots \quad .$$

Since R is finite, and $P_{i+1}$ is a proper subset of $P_i$, the construction must terminate in a finite number of steps.

## E. PROCEDURE FOR ITERATIVE WEIGHT ADJUSTMENT

To aid in the understanding of the weight adjustment procedure for hierarchies, we first consider a geometric interpretation of weight adjustment for a single element to generate a linearly separable logic function. As was argued previously, any one of the inequalities listed in Equations 15 and 16 corresponds to requiring that the point W in $\Omega_W$ lie on one or the other particular side of a plane in $\Omega_W$. Consider the following procedure. All possible input vectors are presented repeatedly in some convenient order to the element. Each time the element "makes a mistake," the vector W is to be adjusted. Let the set of values of W which satisfy the logic function to be represented by $\Omega_0$, $\Omega_0 \in \Omega_W$. Testing whether the logic function is satisfied for a particular input corresponds to testing whether the present value of W lies on the same side of the plane defined by the input as does the solution space. An error is detected when W and $\Omega_0$ lie on the opposite side of the plane in question (Figure 9). The shortest distance to a point in the solution space from the present value of W can be expressed in two components: a component that is perpendicular to the plane being tested and a component that is parallel. Each time an error is detected, let the point W be moved exactly to the plane being tested along the path perpendicular to the plane. This procedure clearly reduces at each step the distance from the point W to the solution space. Since at each iteration this distance is reduced, the process converges, although not necessarily in a finite number of steps, as is illustrated schematically in Figure 10a.

Convergence in a finite number of steps can be produced by the following additional construction. Note that $\Omega_0$ is bounded by a set of planes. Define $\Omega_0'$ to be a subset of $\Omega_0$ that lies a distance $\delta$ from each of the boundary planes of $\Omega_0$. $\delta$ is chosen sufficiently small so that $\Omega_0'$ is not empty.

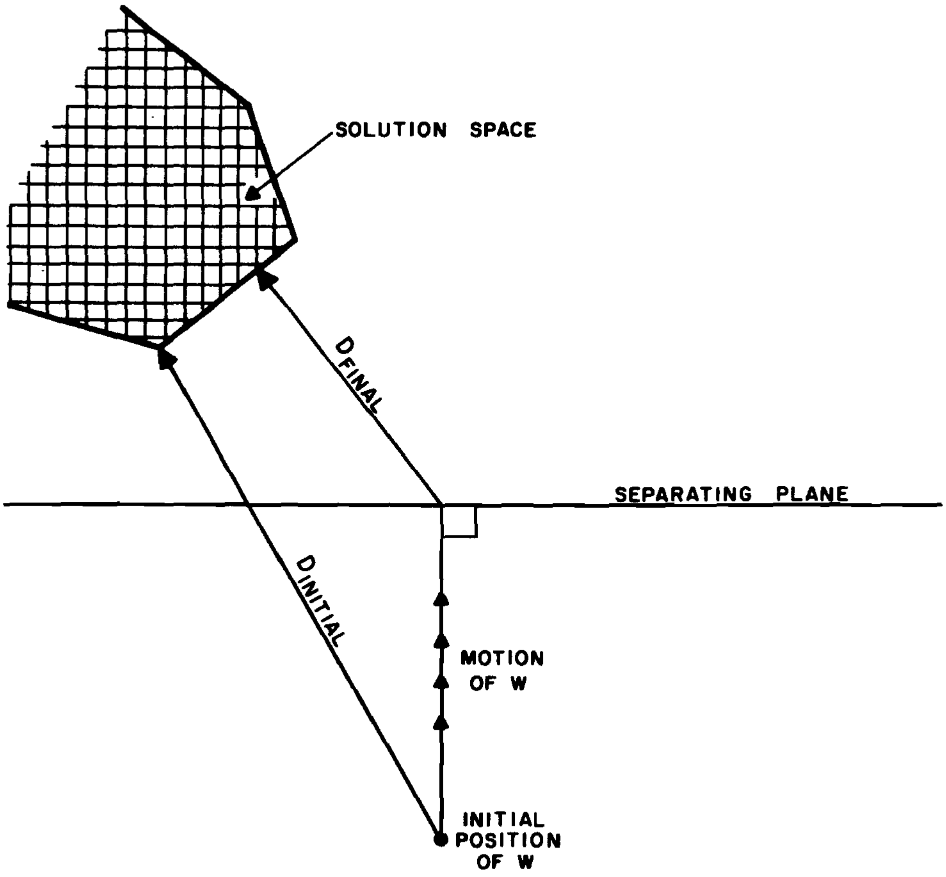


Figure 9. Convergence of W along Perpendicular to a Separating Plane in Solution Space

At each iteration the point W is moved perpendicular to the plane being tested to a position a distance from the plane on the same side as the solution space (Figure 10b). By the identical argument presented previously, the point W converges to $\Omega_0'$. From elementary theory of infinite series, there exists a finite M for which the distance to $\Omega_0'$ is less than or equal to $\delta$.

To produce the motion of W along the perpendicular to the plane being tested, we note that each input vector is exactly the perpendicular to the plane it defines. At each step where an error is detected one can write that the successor of W, W', is given by

$$W' = W + kA$$

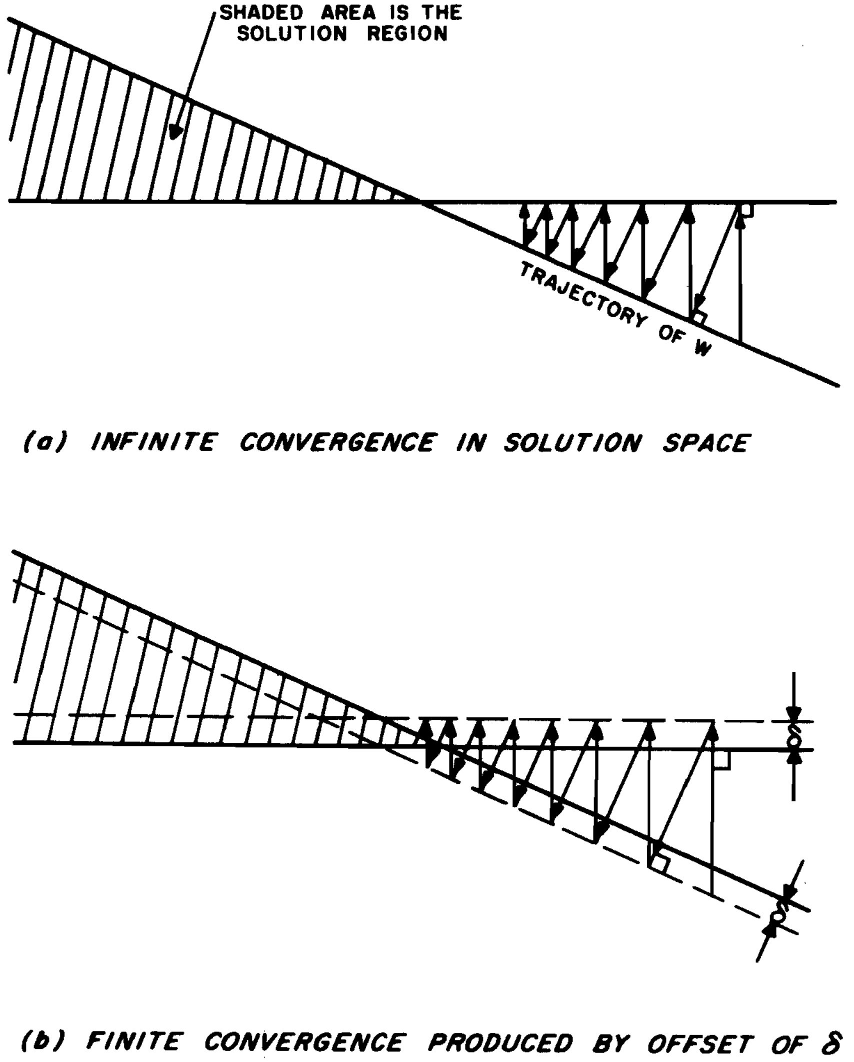
where k is chosen to make

$$\langle W', A \rangle = T \pm \delta .$$

The plus sign is chosen when $\langle W', A \rangle$ is supposed to be greater than or equal to T, the minus sign when $\langle W', A \rangle$ is supposed to be less than T.

It should be noted here that the usual form of the algorithm to select W is to increment W according to the rule

$$W' = W \pm \frac{\delta}{n+1} A, \quad \delta \text{ fixed,}$$



*(a) INFINITE CONVERGENCE IN SOLUTION SPACE*

*(b) FINITE CONVERGENCE PRODUCED BY OFFSET OF δ*

Figure 10. Convergence of W in Solution Space

with the positive or negative sign chosen by the rule given above. A moment's reflection shows that this rule is analogous to the one given, but in general might converge somewhat more slowly.

For adjusting the weights in a hierarchy, the procedure presented below will sometimes move the weights for a particular element in the "wrong" direction, i. e., away from its solution space. It will be shown, however, that at each step of the algorithm, a number tends to decrease which is the sum of the projections of the distances to the separate solution spaces onto the input vector for each of the elements in the hierarchy. Since these distances are non-negative, they tend individually to go to zero.

The first case to be considered is the case in which exactly two levels are needed for synthesis of the function. Let each test of the network be characterized by (1) the value of the function, $\chi$ , for A; and (2) the output of the network for A.

From these data a signal can be derived which informs the network when it makes a mistake. The rule that the elements will follow in their changing weights will be one in which an element's weights change when it could be "at fault" for the error. If either element could be culpable, they will both change by equal increments (although, it so happens, with opposite signs). The rules of adjustment are as follows:

Case I - The network fails to produce an output when it should.

(1) Case Ia - The second level is inhibiting the first level.
Rule: Reduce weights of second level along the direction of input vector until second level sum is reduced to $T - \delta$ .

(2) Case Ib - The second level is not inhibiting the first level.
Rule: Increase weights of first level until sum is $T + \delta$ .

Case II - The network produces an output level when it should not.

(1) Case IIa - The second level is inhibiting the lowest level.
Rule: The second level weights are unchanged. Weights of first level - including inhibitory connection from the second level - are decreased along the vector input to the first level until sum is $T - \delta$ .

(2) Case IIb - The second level is not inhibitory to the first level.
Rule: The lowest level weights are decreased and second level weights are increased simultaneously in equal amounts until either the first level reaches $T - \delta$ or the second level reaches $T + \delta$ .

Note that Case Ia can go over to Case Ib if first level weights are too low. Similarly, Case IIb can go over to Case IIa if the second level is not sufficiently inhibitory on the first level to inhibit outputs from the first level.

To examine whether these rules converge to the solution space, one can list 18 possible conditions that are the result of taking all possible combinations of:

(1) Projections of first level weights on input vector are too high, within solution range, too low

(2) Projections of second level weight on input vector are too high, within solution range, too low

(3) Network is firing when it should not, and not firing when it should

These possible conditions (Table I) together with the rules above, define the motions of a point in a $P_1$, $P_2$ plane, where $P_1$ and $P_2$ are the projections on the input vector of the respective distance of the weight vectors $W_1$ and $W_2$ from their solution spaces. Note that $W_1$ has one more component than $W_2$, since it includes the inhibitory weight of the second level element on the lowest level element. Figure 11 illustrates graphically that for every possible weight condition, and for $f = 1$ and $f = 0$, the sum of $P_1$ and $P_2$ either reduces or remains constant, as can be seen from the direction of motion of the vector in the $P_1$, $P_2$ plane for every possible condition.

Consider now the case where three levels are needed for synthesis. Partition the network into upper and lower sections, the lower section containing two elements, the upper section containing one. Let $P_1$ be the sum of the projections onto the input vector of the distances from their respective solution spaces of the lower two elements, and let $P_2$ be the projection onto the input vector of the distance for the upper weights. The same argument intuitively applies as was used previously. There is, however, at least one factor omitted from the informal discussion above. For many logic functions, several distinct syntheses exist, depending on exactly which minterms are chosen for each level to control. There are not unique solution spaces corresponding to $P_1 - P_2 = 0$, but perhaps numerous disjunct regions in the $W_1 \times W_2$ space that are allowable solutions. Intuitively it seems likely that convergence would continue

Table I. Eighteen Possible Conditions of the Weight Vectors of a Two-Level Hierarchy

| Conditions[a] | | | Change | | Remarks |
|---|---|---|---|---|---|
| f | $f_1$ | $f_2$ | $w_{f1}$ | $w_{f2}$ | |
| 0 | -- | -- | -- | -- | No possibility of error |
| 0 | -- | 0 | -- | -- | |
| 0 | -- | + | -- | -- | |
| 0 | 0 | -- | ↓, - | ↑, - | Changes if $f_2$ is supposed to inhibit $f_1$ |
| 0 | 0 | 0 | -- | -- | No possibility of error |
| 0 | 0 | + | -- | -- | |
| 0 | + | -- | ↓ | ↑ | |
| 0 | + | 0 | ↓ | ↑ | |
| 0 | + | + | ↓ | -- | |
| 1 | -- | -- | ↑ | -- | Steady if $f_2 = 0$, decreases if $f_2 = 1$ |
| 1 | -- | 0 | ↑ | -, ↓ | |
| 1 | -- | + | -- | ↓ | |
| 1 | 0 | -- | -- | -- | No possibility of error |
| 1 | 0 | 0 | -- | -- | |
| 1 | 0 | + | -- | ↓ | |
| 1 | + | -- | -- | -- | |
| 1 | + | 0 | -- | -- | |
| 1 | + | + | -- | ↓ | |

[a] In columns $f_1$ and $f_2$, + indicates that weights are too high, - indicates they are too low, and 0 indicates they are acceptable.

when one of these solution regions is approached, but it is not certain at this time. In view of the potential applicability of such methods to adaptive control problems, the question should undoubtedly be pursued.

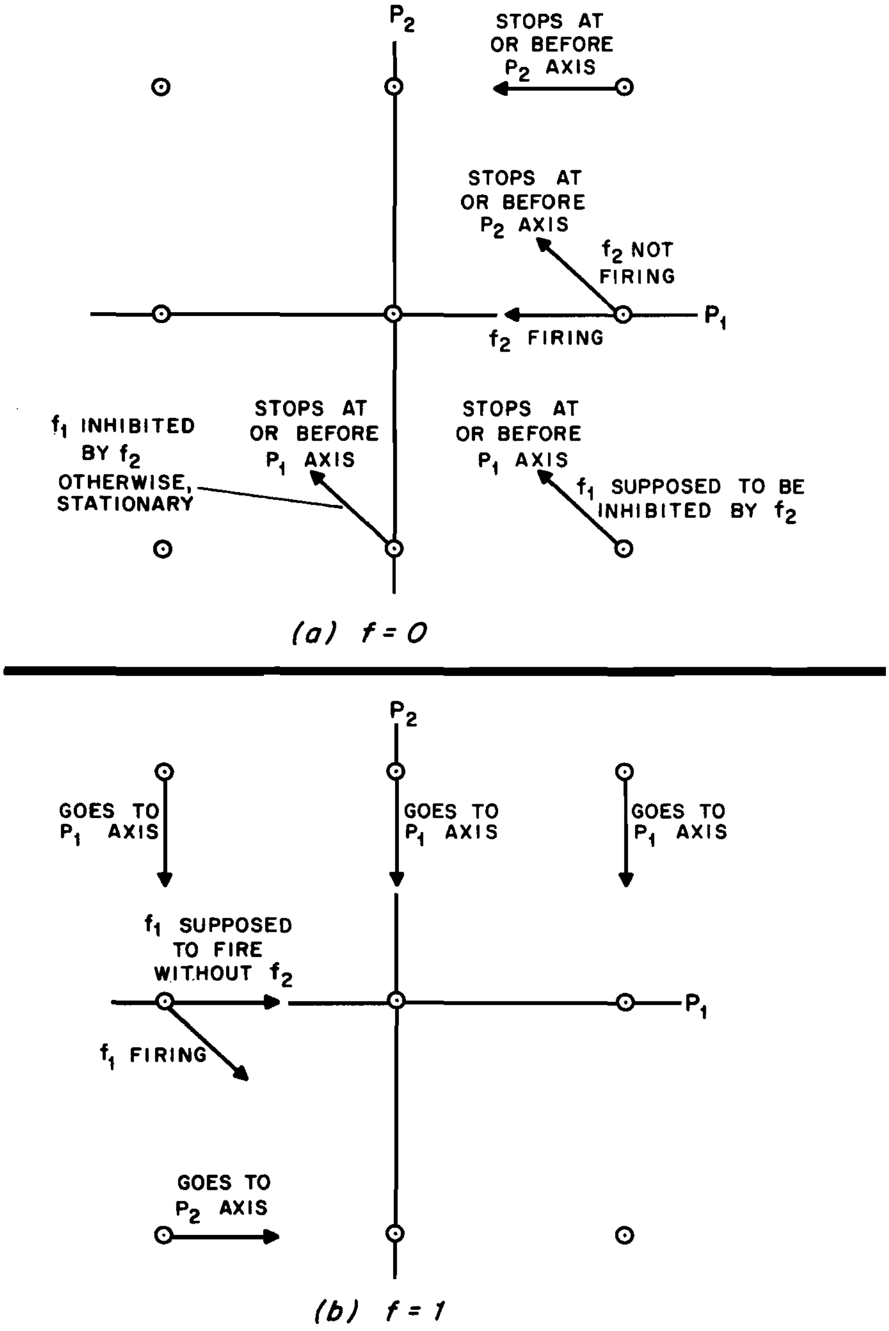

P2
STOPS AT OR BEFORE P2 AXIS
STOPS AT OR BEFORE P2 AXIS
f2 NOT FIRING
P1
f2 FIRING
f1 INHIBITED BY f2
OTHERWISE, STATIONARY
STOPS AT OR BEFORE P1 AXIS
STOPS AT OR BEFORE P1 AXIS
f1 SUPPOSED TO BE INHIBITED BY f2
(a) f = 0
P2
GOES TO P1 AXIS
GOES TO P1 AXIS
GOES TO P1 AXIS
f1 SUPPOSED TO FIRE WITHOUT f2
P1
f1 FIRING
GOES TO P2 AXIS
(b) f = 1

Figure 11. Projections onto Input Vector of Distances from $W_1$ and $W_2$ to Their Solution Spaces

## SECTION IV

## THE STATE IDENTIFICATION PROBLEM AND AN ALGEBRAIC STRUCTURE OF LEARNING

One factor that has been ignored in the work presented to this point is that the output from the environment at any instant may not define its internal state. As an example, consider a linear process which satisfies

$$\dot{X} = AX + BM$$

$$Y = CX$$

and in which only variable Y can be observed by the controller. Such problems are standard in the literature. Such systems are said to be observable if the state vector Y can be determined by observation of the vector X over finite time. It is readily shown that a linear system is observable if, and only if, the columns of

$$F = [C^t, \ A^tC^t, \ (A^t)^2 \ C^t, \ . \ . \ . \ . \ , \ (A^t)^{n-1} \ C^t]$$

span the state space.

Similar remarks apply to controllability. A system is controllable if it can be transferred from the initial state to the zero state in finite time. Again, for the linear case, the requirement yields a relationship on the system variables; i.e.,

$$G = [B, \ AB, \ A^2B, \ . \ . \ . \ A^{n-1}B]$$

must span the state space.

From the standpoint of networks of iterated elements, the implications are that any component of the intervening variables P cannot be computed using only the present values of the components of Y, but must also consider the past history of Y.

For adaptive control by iterated elements, a further complication is introduced. One is led to question whether a process can indeed be identified by an adaptive controller. The problem has been investigated for sequential machines, with affirmative results. Assume first that a given finite state process with finite input set has two properties:

(1) Every state is accessible from every other state (strongly connected)

(2) The process is a Moore machine with the property that the output mapping $\psi$: Z ⟶ V is one-to-one.

A state-transition function for the machine can obviously be discovered in finite time by exhaustive testing. By the same argument, a state-transition function for the equivalent Mealy machine can also be discovered.

Consider now a process, $\Sigma$, like that above except that $\psi$: Z ⟶ V is not one-to-one. We define such a process to be observable if, knowing the state transition function and the function, it is always possible to determine the present state of the machine from a finite number, r, of immediate past observations. The state-transition and output function of such a machine can also be discovered in finite time by the following proof.

Define a new machine, $\Sigma'$, such that its input alphabet is all sequences of length r or less of symbols from n, its set of initial states is the set of states of $\Sigma$, and its output alpha is all sequences of symbols of length r or less from v. For each point in time, define the input to $\Sigma'$ to be that element from its alphabet that corresponds to the last r inputs to $\Sigma$, and its output symbol to be the one corresponding to the last r outputs from $\Sigma$. Clearly the mapping $\Sigma'$ is one-to-one for this new machine, and the preceding argument applies.

Thus it is possible to discover the state transition and output functions of any finite state sequential machine that is observable and strongly connected. Note, however, that the determination, in general, can only be made by an exhaustive process. If outputs from the machine are from the set S, we have a mapping

$$I_p: S^r \longrightarrow Z_p .$$

Consider, however, the fact that the optimal controller may have fewer states than the process. The controller states then represent a set of equivalence classes of process states suitable for optimal control. For control purposes it is adequate to identify only the equivalence class of the process state. We have then the mapping

$$I: S^r \longrightarrow Z_c ,$$

where $Z_c = \{Z_p\}$ and $\{Z_p\}$ is the set of equivalence classes of $Z_p$.

It has been previously established that for each process state there is an optimal input symbol the controller should emit, i.e., there is an optimal control function C:

$$C: Z_c \longrightarrow N .$$

This can be combined with the state identification map to yield a "feedback law,"

$$F: S^r \longrightarrow N ,$$

so that it is possible to control a process optimally with a controller consisting of a mapping, of stimuli into responses, if the cost of state equivalence class identification is ignored, and if identification is possible in a finite (and bounded) number of steps.

Let us consider now the various mappings that have been ascribed to the controller, or to the computation of optimal trajectories, at various points in the discussion above. First there are the transitions of the controller itself:

$$t_c: Z_c \times S \longrightarrow Z_c$$

$$\psi_c: Z_c \times S \longrightarrow R$$

which yields the pair (b, c), where c is cost to the desired state along the optimal trajectory and b is the response that should be emitted to follow the optimal trajectory. Combining $t_c$ and C can yield a prediction of the next cost to desired state, which can be denoted by the second mapping

$$C': S \times Z_c \longrightarrow D'$$

where D' is defined as the change in cost-to-go which should be observed. In addition,

$M: D \longrightarrow R$ gives the output of the controller,

$I: S^r \longrightarrow Z_c$ identifies the process state,

$R: Z_p \longrightarrow Z_c$ establishes equivalence classes of process states, and

$F: S^r \longrightarrow R$ is the overall feedback law obtained by combining I and $\psi_c$ .

We are now in a position to point out that intuitive ideas of "learning" include two algorithms: (1) a procedure for process identification, which reconciles the results of the functions I and $t_c$, and (2) a procedure for establishing optimal trajectories, which reconciles C and C' .

If the steps taken in arriving at this point are indeed justified, we are led to an interesting conclusion. An iterative element synthesis of any required adaptive controller is possible provided only that all terminal segments of optimal trajectories of the process are themselves optimal trajectories, and that the process is controllable and observable.

## SECTION V

## SUMMARY AND CONCLUSIONS

The major algebraic results presented in this report are as follows:

(1) A model of creature behavior and of neuron function is presented in which both the creature and its environment are assumed to be sequential machines, and the neurons implement the transitions of the "creature" machine.

(2) It is shown that if costs are assigned to the transitions of the environment and one environment state is selected as a desired state, one can define and find the optimal trajectories of the environment from each possible initial state.

(3) The number of states in an optimal controller for the environment above is less than or equal to the number of states in the environment.

(4) If the environment is observable and connected with respect to the desired state, the optimal trajectories can be discovered by an orderly testing procedure, which establishes the trajectories "backward," in a pattern analogous both to dynamic programming and to chaining.

(5) An auxiliary variable can be defined in such a way that its value is monotone, decreasing along any optimal trajectory, and can be identified with "drive level."

(6) Six mappings for the controller can be defined, two of which are "predictions" and two "confirmations," which can be implemented with neuron-like elements.

(7) In the linear case and in at least some selected discrete cases, algorithms are exhibited for comparing the predictions with the confirmations and correcting the operation of the controller so as to cause it to traverse optimal trajectories, using neuron-like elements.

These results show that insofar as the formal assumptions hold for creature behavior experiments, a drive-reduction hypothesis for creature behavior is a purely theoretical consequence of the assumptions, whether it has any physiological validity or not.

Although the result most easily exhibited is the validity of the drive-reduction hypothesis, other consequences of the analysis pertinent to behavior theory have been exhibited.

(1) The analysis tends to support the view that stimuli do not become indicators of responses directly, but indicators of environment state equivalence classes, which in turn become connected to responses. That is, the mapping

$$S \times Z_c \rightarrow R$$

is the result of two mappings

$$S \times Z_c \rightarrow D'$$

$$D' \rightarrow R \ .$$

Although these two mappings can be combined formally into a single mapping, the introduction of the intervening variable offers computational advantages and also offers an explanation of some creature behavior and of neuron interconnections. The construction resembles Tolman's sign learning.

(2) The various mappings exhibited form categories for classifying behavior and physiological experiments and can define meaningful experiments for future research. One example animal experiment was actually performed.

The research makes some minor contributions to optimal control theory and to algebraic systems theory, in that

(1) An algorithm is presented for discovering optimal trajectories on a sequential machine.

(2) An adaptive system for minimizing a quadratic cost functional on a linear system is exhibited.

(3) A method is given for generating logic functions that are not linearly separable by use of threshold elements.

Any conclusions advanced as coming out of the research described in this report are of necessity tentative. The neuron has not been modeled in sufficient detail and with sufficient proved accuracy for one to say with any degree of assurance that the conclusions are firm. Nevertheless, the research results point strongly to the view that neurons could operate independently, forming their own connections to other neurons and adjusting their own synapses. By these actions they could produce overall organism behavior that would optimize some global performance criterion. No evidence has yet been discovered to refute the view. The concepts of synthetic intervening variables, hierarchies of decision making, equivalence classes of stimuli, and optimal trajectories seem to have cogent relations to the results of both physiological and behavioral experimentation, and to the introspective views of cognition and thought.

Whether these relations will continue to hold must be revealed by future research - the problems are not close to being solved.

The concepts advanced so far do appear to have applications in the field of adaptive optimal control. If the present promise is fulfilled, complex adaptive systems could be built entirely of iterative building blocks like those described in preceding sections.

It is the view of the authors that the research shows at least as much promise for the future as it did when the task was undertaken, and that the interim results could, with additional study, become "spin-offs" of value in the control field.

# APPENDIX I

## ALGEBRAIC SYSTEMS, SEQUENTIAL MACHINES, DEVOLVING SYSTEMS

### A. ALGEBRAIC SYSTEMS

The following discussion presents a formal algebraic construction that relates sequential machine theory to systems that show more variability of behavior initially than they do later in time. A number of properties of sequential machines are reviewed.

We define a system, $\Sigma$, to be a relation, $\Sigma \subseteq N \times V$. For reasons that will be apparent later, we will call N the disturbance space and V the parameter space. N and V are arbitrary sets.

We use the conventional definitions of such terms as function, relation, linearly ordered, semigroup, etc. See, for example, Goffman (Reference 7) or Nelson (Reference 5).

We will say that $\Sigma$ has a state representation if there exists a set $Z_0$ and a mapping $t\colon Z_0 \times N \rightarrow V$ such that

$$\Sigma = \{ (n,v) \mid (z,n) \in (Z_0 \times N) \rightarrow t(z,n) = v \}$$

Windeknicht (Reference 8) has shown that every system has a state representation. We will call $Z_0$ a set of states for the system.

Let us now let $\Sigma$ be a partial groupoid. We will call the system $\Sigma$ a normal system if there exists a set Z and a mapping $\psi$ and t such that

$$\psi\colon Z \times N \rightarrow V$$

$$t\colon Z \times N \rightarrow Z$$

where

$$(\forall z)(\forall n)\ [n \Sigma \psi(z,n)]$$

$$(\forall z)(\forall n)(\forall n')\ [(\psi(z,nn') = \psi(z,n)\psi(t(z,n),n')) \wedge (t(z,nn') = t(t(z,n),n'))]$$

and where nn' represents the groupoid product of n and n' if it exists. We say that

Z is a set of states of the system,
$\psi$ is a state representation of the system,
t is a state transition function of the system.

As has been pointed out by Windeknicht, the set $Z_0$ and the set Z must satisfy

$$Z_0 \subseteq Z.$$

We must therefore distinguish between a set of states for a normal system and a set of initial states for that system. We call the set of pairs $[(n,v) \mid (n,v) \in \Sigma]$ the "disturbance-response" pairs of the system $\Sigma$ .

## B. CONCEPTS FROM THEORY OF SEQUENTIAL MACHINES

Let N be a semigroup on a set of generators, A, and let the semigroup be a product of the system concatenation. Such a system can be identified with a "sequential machine." A conventional definition of a sequential machine, or a semi-Thue system, has been defined by Nelson (Reference 5) to be a quadruple $F = \langle A, B, \mathcal{A}, \mathcal{P} \rangle$ where A is an alphabet, B is the set of words on A, $\mathcal{A}$ is an axiom or axiom scheme ($\mathcal{A} \in B$), and $\mathcal{P}$ is a (finite) set of rules of inference (productions) of the form:

$$PgQ \rightarrow Pg'Q$$

where g, $g' \in B$ and P, Q are syntaptic variables on B.

A finite transducer (or sequential machine) is defined by Nelson to be a semi-Thue system, $T = \langle A, B, \mathcal{A}, \mathcal{P} \rangle$ where A consists of two not necessarily disjunct subalphabets S and R and a subset of auxiliaries Z disjunct from S and R. The axiom is of the form $Z_ix\#$, where $z_i \in Z$, $x \in B$.

$\mathcal{P}$ is a set of rules of the following type:

(1) $P_is_jQ \rightarrow Pr_kq_\ell Q$

$Pz_i\# \rightarrow P$ (#, $\# \in A$, is a special symbol to denote the end of a word in B).

(2) There is exactly one production of type a for each pair $(z_i, s_j) \subset Z \times S$ and exactly one of type b for each $z_i \in Z$.

(3) Nothing is a production unless its being so follows from 1 and 2 above.

In the formal system theoretical context above, let N be a free semigroup on a set of generators S, and V be a free semigroup on generators R. We can make the following identification with Nelson's definition.

The alphabet of Nelson's system is identified with $S \cup R \cup Z$ where Z is the state set.

Let $z_i$ be in Z, and let $s_jQ$ be a word on B. P is empty, Q may or may not be. The production

$$Pz_is_jQ \rightarrow Pr_kz_\ell Q$$

is defined by

$$z_\ell = t(z_i, s_j)$$

$$r_k = \psi(z_i, s_j)$$

Clearly, since $\psi$, t are functions, there is exactly one such production for each pair $(z_i, s_j)$. We can further require that

$$(\forall z \in Z) \; [\psi(z, \lambda) = t(z, \lambda) = \lambda]$$

where $\lambda$ is the empty string. This satisfies the requirement that for all z in Z, there exists a production of type b, where the special symbol # stands for the empty string.

The transition diagram of a sequential machine with finite number of states, Z, is a directed linear graph in which there is a node for each state, z, and for each elemental mapping of $t(z, s) = z'$ there is a directed branch from the node z to the node z'. Each such branch is labeled either with the symbol s or with a pair $[s, \psi(z, s)]$, depending on the form of $\psi$, as is discussed below.

Two states, $z_1$ and $z_2$, are equivalent (written $z_1 \equiv z_2$) if

$$(\forall x)\ [t_{z1}\,(x) \ = \ t_{z2}\,(x)]$$

where x is any string of symbols from N.

A sequential machine is reduced if

$$(\forall z_i)\ (\forall z_j)\ [(z_i \ = \ z_j) \longleftrightarrow (z_i \ = \ z_j)];$$

otherwise it is redundant.

Two machines, $\Sigma_A$ and $\Sigma_B$, with common input and output alphabets S and R, are homomorphic if there is a function, f, on $Z_a$ into $Z_b$ such that

$$f(t_a(z, s)) \ = \ t_b(f(z), s)$$

and

$$\psi_a(z, s) \ = \ \psi_b(f(z), s)\ .$$

The machines are isomorphic if $f^{-1}$ exists.

The following differs slightly from the usual definition. A sequential machine is "connected with respect to initial state $z_0$" ($z_0 \in Z_0$) if for each $z \in Z$ there is an $n \in N$ such that $t(z_0, n) \ = \ z$. If the machine is connected with respect to each state in Z, it is "strongly connected." A strongly connected system does not devolve in state, a concept which is discussed later.

A sequential machine is minimal if it is reduced and connected. A submachine, or subsystem of a machine $\Sigma = \langle S, R, Z, \psi, t, Z_0\rangle$ is a machine $\langle S', R', Z', \psi', t', Z_0'\rangle$ such that $Z' \subseteq Z$, $S = S'$, $R' \subseteq R$, $Z_0' \subset Z_0$, and for which t' and $\psi'$ are the restrictions of t and $\psi$ to $Z' \times S'$.

Two machines are indistinguishable if for every $z_a \in Z_a$ and $s = s_1 s_2 \ldots s_n \in S$ there is a $z_b \in Z_b$ such that the sequence of outputs $\psi_a\left(z_a, s_1\right)$, $\psi_a\left(t(z_a, s_1),\ s_2\right), \ldots,$ $\psi_a\left(t(\ldots(t(z_a, s_1)\ldots),\ s_n\right)$ is equal to $\psi_b\left(z_b, s_1\right)$, $\psi_b\left((t\,(z_b, s_1),\ s_2\right), \ldots,$ $\psi_b\left(t(\ldots(t(z_b, s)\ldots),\ s_n\right)$ and vice versa.

Two forms of the function $\psi$ have been studied in the literature. The first is such that

$$(\exists s_1)\ (\exists s_2)\ (\exists z)\ [\psi(z, s_1) \ \neq \ \psi(z, s_2)]$$

and the other is

$$(\forall s_1)\ (\forall s_2)\ (\forall z)\ [\psi(z, s_1) \ = \ (z, s_2)]$$

The first formulation has been called Mealey's machine. The second is equivalent to the formulation called Moore's machine in which

$$\psi :\ Z \longrightarrow R$$

is the form of the output function.

That these are equivalent is shown, for example, in Nelson (Reference 5). The assumption of one or the other form does lead, however, to differences in the instrumentation of equivalent machines. The implications of this fact to the present discussion is examined later.

In presenting a formal model of a creature-environment system, a slight change is made in semantic identifications. Consider three free semigroups, S, R, and N, on sets of generators A, B, and C, with concatenation as the semigroup operation. We will call them, respectively, the stimulus, response, and disturbance semigroups.

Elements of the various semigroups and sets will be denoted by corresponding lower case letters. Let the system $\Sigma$ be such that

$$\Sigma \subseteq N \times R \times S$$

We now require that the system display the following partitioning properties.

There exist two sets, $Z_c$ and $Z_e$, and mappings such that

$$\psi_c\colon S \times Z_c \longrightarrow R$$

$$\psi_e\colon (R \times N) \times Z_e \longrightarrow S$$

$$t_c\colon S \times Z_c \longrightarrow Z_c$$

$$t_e\colon (R \times N) \times Z_e \longrightarrow Z_e$$

and further that the transition conditions on $\psi$ and t given in the definition of a normal system are satisfied for each of the subsystems.

The parentheses are inserted above to clarify the notion that the "input" to the environment is the Cartesian product of creature responses and disturbances (perhaps random effects or perhaps effects introduced by the experimenter).

This partitioning is a special case of system interaction, studied by Birta (Reference 9). It is readily shown that $\Sigma$ is a normal system by eliminating S between the four defining mappings, yielding

$$\psi_c\colon R \times N \times Z_e \times Z_c \longrightarrow R$$

$$t_c\colon R \times N \times Z_e \times Z_c \longrightarrow Z_c$$

$$t_e\colon R \times N \times Z_e \longrightarrow Z_e$$

Let the state set of the overall system be $Z \subseteq R \times Z_e \times Z_c$, and let its output be R. Rearranging the above yields

$$\psi\colon N \times Z \longrightarrow R$$

$$t\colon N \times Z \longrightarrow Z$$

Note that Z may be a proper subset of $R \times Z_e \times Z_c$. The point is examined further below; not all elements of $R \times Z_e \times Z_c$ are reachable.

To define reachable states, consider a normal system on free semigroups N and V (as defined earlier) with set of initial states $Z_0$. Form a sequence of sets $Z_0$, $Z_1$, . . . , as follows:

$Z_0 = Z_0$ (The set of initial states)

$Z_1 = t(Z_0 \times A)$ (The set of all possible successor states to $Z_0$. A is the set of generators of N)

$Z_2 = t(t(Z_0 \times A) \times A)$ (Possible successors of $Z_1$)

$Z_n = t(t(\ldots t(Z_0 \times A)\ldots) \times A)$ ($Z_n$ is the set of states reachable after n steps)

## C. EXTENDED AND DEVOLVING SYSTEMS

The following discussion defines two terms, an extended system and a devolving system, and formalizes the intuitive concept of a system that shows more variability of behavior initially than it does later. The motivation is to formalize the experimental situation in which living creatures are observed to alter their behavior from time to time, tending toward stereotyping or habit formation.

We construct formally the following new system, $\Sigma^{\infty} \subseteq N \times V$, which we will call the extension of the system $\Sigma \subseteq N \times V$.

Let the state set $W_0$ of $\Sigma^{\infty}$ be $\bigcup\limits_{i=0}^{\infty} Z_i$ (the set of potential states of $\Sigma$ at $t = 0$).

Let $t_W = t_{\Sigma}$ and $\psi_W = \psi_{\Sigma}$

It is immediate that

$$W_j = \bigcup_{i=j}^{\infty} Z_i \ .$$

Physically, this construction corresponds to the union of all possible right translations of the time origin of the original system; i.e., any point on the trajectories of the original system can be considered as occurring at $t = 0$.

The definition for a system that devolves in state is for all i, $0 \leq i$, for which trajectories are defined,

(1) $W_{i-1} \supseteq W_i$

(2) There exists at least one i for which $W_{i-1} \supset W_i$ .

We consider a normal system $\Sigma \subseteq N \times V$, where N is a free semigroup with set of generators A. We require that the set of initial states $Z_0$ be equal to the initial set of potential states, $W_0$. We note that a free semigroup with identity is closed under left cancellation. (We include the empty string as a member of the semigroup.) Consider any arbitrary pair $(n, v) \in \Sigma$, where $n = n_1 n_2$. The following theorem examines the relation of the extended system to the original system.

Given a system $\Sigma$ and its extension $\Sigma^{\infty}$ as defined above,

$$(\forall(n,v))\ (\forall v_2)\Big[\big((n,v) \in \Sigma\big) \wedge (n = n_1 n_2) \longrightarrow (v = v_1 v_2) \wedge \big((n_2, v_2) \in \Sigma^{\infty}\big)\Big]$$

A proof of the theorem can be supplied by contradiction. Suppose there exists an $(n, v) \in \Sigma$ such that no appropriate $(n_2, v_2)$ can be found. From the definitions above, the initial segment of $(n, v)$ (we write $(n_1, v_1)$) is in $\Sigma$. We can write that

$$\big((n,v) \in \Sigma\big) \longrightarrow \neg(\exists z)\ \big[(z \in Z_0) \wedge (\psi(z, n_2) = v_2)\big]$$

From the definition of normality, however

$$\bigl((n_1n_2,\ v_1v_2) \in \Sigma\bigr) \rightarrow (\exists z_0)\ [\psi(z_0,\ n_1n_2)\ =\ \psi(z_0, n_1)\psi(t(z_0, n_1)n_2) \wedge$$

$$(t(z_0, n_1n_2)\ =\ t(t(z_0, n_1), n_2)]$$

or

$$z \in Z_0,$$

where

$$z\ =\ t(z_0, n_1)\ .$$

From the definition of t

$$t(z_0, n_1) \in Z,$$

which implies, from the definition of $W_0$, that

$$t(z_0, n_1) \in W_0,$$

and our assumption that $Z_0 = W_0$ is violated.

The above demonstrates that any terminal segment $(n_t, v_t)$ of any pair $(n, v) \in \Sigma$ is also in $\Sigma$. It does not follow, however, that the converse is true, i. e., $(n, v) \in \Sigma$ does not imply the existence of an $(n', v') \in \Sigma$ such that $(n, v)$ is a terminal segment of $(n', v')$. In fact, we have chosen in the main text to examine the case in which there exists at least one pair $(n, v)$ which is not a terminal segment of any $(n', v') \in \Sigma$. Such a system devolves in state, since the initial state $z_0$ such that $v = \psi(z_0, n)$ never recurs.

The above has formalized the concept of initial variability of behavior which disappears with experience, as typified by a system that devolves in state. Although the concept of a devolving system allows for variability, it does not account for the fact that creature behavior is goal-directed. To include this idea, we earlier considered certain concepts of optimal control of a sequential machine.

# APPENDIX II

## PHYSIOLOGICAL CONSIDERATIONS

Physiologists have apparently spent little effort on the theory of learning as related to known characteristics of the nervous system. Hebb has postulated that if a neuron A repeatedly or persistently aids in the firing of neuron B, A will become excitatory on B (Reference 10). Milner extended this postulate to include inhibitory connections (Reference 11). The rule bears at least some resemblence to Thorndike's more or less discredited "law of exercise" (Reference 12). It has been repeatedly shown in behavior experimentation that practice without reinforcement leads to little or no strengthening of response tendency. One is tempted to question Hebb's statement, in view of modern behavior theory. In justice to Hebb and Milner, it must be pointed out that their statements were tentative, and not central to their principal studies of the nervous system.

However, since the origin of behavior is the nervous system, it is helpful to review some of the known facts of neural behavior. The brain is made mostly of two kinds of cells, neurons and glial cells. The neurons generate electrical activity and are the primary source of behavior. The glia (from the Greek word meaning "slimy") were up until recently considered only as something to fill up the spaces between neurons. Some recent research shows that they play a more complicated role. There are about $10^{10}$ neurons in the human brain, and perhaps 100 to 1000 times as many glial cells.

A neuron cell body is a few microns in diameter and its input and output extensions, which are part of it, are of the order of one to three microns in diameter. The extensions (processes) go surprising distances, often several inches or a foot. Glial cells are of the same order of magnitude in size as neurons and also take on all sorts of odd shapes.

Most of a neuron's processes are inputs (dendrites). A neuron usually has only one output (axon), which branches profusely. Each branch ordinarily is terminated with an endbulb, which usually lies very close to some part of another neuron. The place where one neuron affects another, i.e., the place where an endbulb touches or almost touches another neuron, is called a synapse.

Each neuron (except sensors) receives pulse train inputs from many other neurons. Estimates on the number of inputs to a neuron range from hundreds to hundreds of thousands. A neuron emits pulse trains that may affect several hundred other neurons. When a neuron generates a signal, an electrical pulse is transmitted down its output lead to its endbulbs. The endbulbs emit a chemical agent under the electrical stimulation. This transmitter agent affects the following neuron on which the endbulb terminates. Some chemical agents tend to cause the following neurons to emit a pulse, some tend to inhibit pulse emission. Whether a neuron emits a pulse at any instant is determined by the consensus of its inputs and by the time histories of its inputs and output over the immediate past. As far as the gross anatomy is concerned, the nervous system both inside and outside the brain is highly organized. There are thousands of interrelated regulatory and control loops.

There are about $10^6$ inputs to the brain from the eyes, and perhaps $10^4$ from the rest of the body. There are perhaps $10^4$ output leads going ultimately to muscles. Presumably, any number of these inputs and outputs can be activated at the same time. There is evidence that the inputs to the brain are not raw data, but are extensively preprocessed by peripheral neuron assemblies before reaching the brain, and that outputs are postprocessed before reaching muscles. Some quite sophisticated reflexes operate directly on the spinal or medullar level, requiring no intervention from the higher centers at all.

To the writers' knowledge, no direct evidence exists to bolster the view that synaptic connections change in any way as the result of experience. Neither, however, is there any direct evidence to refute the view. As is discussed later, various writers have presented models based on neuron assemblies in which "learning" is the behavior change due to changed physical connections. The view dates back in essence to Thorndike (Reference 12), who used the then known properties of neurons in his theory, although not in any way so as to make the properties essential to his theory.

# APPENDIX III

## CONCEPTS FROM BEHAVIOR THEORY

### GENERAL

The presentation below follows generally the format of Hilgard (Reference 13), with some interpolations from Dollard and Miller (Reference 14) and from Spence (Reference 15). Paraphrases and direct quotations from these sources are frequent.

Most behavioral data is derived from conditioning experiments in which the subjects are often lower animals. Such experiments are usually divided into two categories, classical conditioning and instrumental (or operant) conditioning. Although the two types of conditioning differ in external form, one can argue that each contains at least some elements of the other. In all such experiments the subjects are presented with stimuli, some of which are controlled by the experimenter, and the responses are observed, usually over numerous trials.

Some problems of definition immediately arise in describing such experiments. By the word "stimuli" one can mean either the attributes of the environment the subject is sensing at a particular instant (often called effective stimuli), or all the attributes of the environment that could be sensed (potential stimuli); or the meaning can be expanded to include proprioceptive stimuli, which are not usually amenable to observation by the experimenter. Similar difficulties attend the definition of "response."

Despite these semantic and observational difficulties, one can confirm some intuitive notions by such experimentation, and can examine whether certain widely held beliefs about behavior that appear to be "common sense" notions are really valid. Some of the notions of this type, which are accepted by most experimentalists without serious objection, are listed below.

(1) Creature behavior changes from time to time, as a result of prior experience.

(2) Under more or less identical circumstances, the behavior of two or more subjects will often be significantly different. Further, the actual observed behavior of a single subject will vary from time to time, regardless of the effort expended to duplicate the previous experimental situation.

(3) Each time a particular behavior pattern is reinforced in a given experimental situation, the probability that the same or a similar pattern will recur in similar future situations is increased (Thorndike's law of effect).

(4) A stimulus and a causally unrelated response that occur contiguously can become associated, in that the stimulus tends to elicit the response (classical conditioning).

(5) The presentation or withholding of certain stimuli, contingent on the subject's prior behavior, serves to reinforce or extinguish response patterns (operant conditioning).

(6) Complex acquired behavior patterns are often acquired by concatenating simple activities backwards in time (chaining).

To explain these phenomena, certain constructs, which may or may not have a physiological basis, have been introduced.

(1) Intervening variables are those constructs that are introduced formally with no direct physiological correlation that determine which particular one of the possible responses to a stimulus will occur in a particular experimental trial (Reference 13).

(2) Drives are those factors that motivate behavior. Drives must be carefully distinguished from stimuli, although they have stimulus properties. According to Dollard and Miller, all drives are stimuli but not all stimuli are drives. Drives are often categorized as basic or acquired. Early writers often wrote lists of basic drives, with such categories as hunger, thirst, sex, and self-preservation. The modern tendency is to avoid such lists.

(3) The drive-reduction hypothesis can be summarized by saying that in a given stimulus situation those responses are strengthened that lead to reduction of the intensity of the drives contiguously with the occurrence of the response. For any stimulus-drive situation the subject tends to make that response that in the past has led to the greatest net reduction in drives. An element of prediction can be included; drives may not be reduced as an immediate consequence of the response, but may be reduced ultimately by response activity of the present.

## B. LEARNING THEORY

Freud did not present an explicit theory of learning. He did state a pleasure principle, i.e., that people and lower creatures seek "pleasure" and avoid "pain." In this he was voicing an idea that dates at least to Aristotle (Reference 16). He also introduced the idea of reduction of tension as a motivation,anticipating Hull's more formal drive reduction. The earliest formal theories that gained wide acceptance are those of Thorndike, Ebbinghaus, Bryan and Harting, and Pavlov, circa 1900. Since that time, some major divisions have sprung up among theorists.

Some theories still current are Guthrie's contiguity theory, Hull's systematic theory, Skinner's operant conditioning theory, the Gestalt theory (typified by Lashley), Lewin's field theory, and Tolman's sign learning. The functionalism of Dewey and others, originally separate schools, has been more or less absorbed into current reinforcement theory. The various theories differ in the importance they place on cognition, the role of reinforcements in conditioning, and the importance of responses themselves in the conditioning process. As is pointed out by Hilgard, "all the theorists accept all the facts . . . the differences between two theorists are primarily differences in interpretation" (Reference 13). Expressed in another way, the theories differ in the constructs they use to explain observed facts, rather than differing in the facts they explain.

On the roles of ideation and cognition in learning, the theories can be divided roughly into (1) connectionists, who ascribe changes in behavior to formation of abstract (or physiological) connections between stimuli and responses (or between successive responses), and (2) cognitivists, who ascribe changes in behavior to ideation and the formation of cognitive structures. Most of the work presented in this report is of a strictly connectionist point of view.

The connectionist philosophy is more easily modeled by the methods of this report, and for that matter has been more often modeled by others than has the cognitive approach. It is therefore proper to mention briefly some of the observed phenomena with which the strict connectivist view can be challenged. Three such phenomena are place learning, latent learning, and reward expectancy.

(1) Place Learning - Experiments can easily be designed to show that the learner does not move from starting box to goal box by a fixed system of movements, but varies his behavior with changed conditions to reach the goal.

(2) Latent Learning - An animal can learn by exploring a maze, without food being presented, as evidenced by the fact that it performs better than a naive animal when food is later placed in the maze.

(3) Reward Expectancy - The pertinent behavior was first observed by Tinklepaugh (Reference 17). A monkey was allowed to observe a banana being concealed under a cup. The monkey was then removed from the location and a lettuce leaf was

substituted for the banana. Later, the monkey showed skill in choosing the correct cup, but rejected the lettuce leaf and searched for the banana. Similar experiments are numerous.

The above phenomena lend credence to, for example, Tolman's sign-learning theory, in which stimuli are signs and the animal learns relations between signs and their significance. On the other hand, numerous results can be cited which bolster the view that movements are learned. Guthrie observed that cats escaping from a puzzle box frequently exhibited exactly the same behavior as was successful in their first escape from the box. This stereotyping argues against the ideationists.

## C. CONNECTIONISM

The basic tenet of the connectionist view is that learning results from sense impressions becoming associated with impulses to action. The view as to how these "bonds," "associations," or "connections" are formed is the basic differentiating factor among the theorists. The contiguity theorists (Guthrie was probably the leading proponent of this view) hold that "a combination of stimuli which has accompanied a movement will on its recurrence tend to be followed by that movement" (Reference 18). The bond is assumed to reach full strength on its first occurrence. Variability of behavior is explained by variations in the stimulus patterns present from occasion to occasion. The "principle of postremity" and associated concepts have been put in postulational form by Voeks (Reference 19).

Reinforcement theorists hold to some form of Thorndike's law of effect. Quoting Hilgard, (Reference 13) "when a modifiable connection is made and is accompanied by or followed by a satisfying state of affairs, the strength of the connection is increased; if the connection is made and followed by an annoying state of affairs, its strength is decreased."

Hull and Skinner are two major reinforcement theorists. Hull's view differs from that of Guthrie in one essential way. For Guthrie, stimuli and responses occurring contiguously are always strengthened. For Hull, an association is strengthened only if the response occurs in company with positive reinforcement. In Hull's view, reinforcement is brought about by a decrease in the stimuli produced by a drive, or by a decrease of stimuli associated with the anticipation (i. e., prediction) of a decrease in these stimuli.

A similar point of view is taken by Skinner. However, Skinner emphasizes the predictive nature of conditioned behavior by his distinction between respondent and operant behavior. Classical theory considers only responses elicited by stimuli. Skinner contends that some activities are emitted that are not necessarily correlated with stimuli, but simply tend to occur as a result of having been reinforced on prior occasions. Stimuli may serve as descriminants, so that one or another response may be emitted depending on the stimuli present, but they do not elicit the response. Responses occur because they are reinforced.

Skinner and his followers have displayed quite striking success in animal training through use of his methods. He has, however, studiously avoided postulating any intermediary concepts to explain observed phenomena without themselves being observable. He uses the concept of drive as a mediator of conditioning but states (Reference 20) that a drive is not a stimulus, nor a physiological state, nor a phychic state, nor simply a state of strength. For his purposes, a drive results from certain operations he may perform (such as food deprivation) and affects the outcome of his experiments in a different way from these things that are reinforcements.

# APPENDIX IV

## BEHAVIOR MODELING

Numerous investigators have attempted to model creature behavior mathematically. The earliest such attempt reported by Hilgard (Reference 13) is due to Ebbinghaus, who fitted an equation of the form

$$b = \frac{100k}{(\log t)^c + k}$$

to an experimentally obtained retention curve. In this equation b is percent of retained learning, t is elapsed time and c and k are arbitrary constants. As Hilgard points out, in contrasting such empirical curve fitting with rational curve fitting, ". . . in empirical curve fitting . . . we select the curve family solely on the basis of fit, and not on the basis of any theory. . . . the word "rational" implies that the family of curves is chosen according to some theory or theoretical deduction. . . ."

Empirical curve fits can be useful to the experimenter, but yield little insight into the underlying mechanisms. Rational curve fitting is the approach that has been used in most succeeding models.

Mathematical theories of learning are classified by Hilgard as based on the following:

(1) Information theory

(2) Theory of feedback mechanisms

(3) Game theory

(4) Differential calculus (a misleading name - the models are based on theory of differential equations with constant coefficients)

(5) Stochastic models

Each of these fields of study promises to relate intuitively to creature behavior. It is not surprising that various investigators have studied their applicability, with greater or lesser success.

The feedback theory approach was investigated extensively by Wiener (Reference 21), but thus far has been little used in learning theory. The Game theory of von Neuman and Morgenstern (Reference 22) has been particularly useful in economic theory. Its applicability to learning theory was tested by Flood (Reference 23). Since Game theory is one possible basis for an optimal control theory, it also enters into the research presented here. Theoretical aspects of behavior have been studied by Hovland (Reference 24).

Models based on probability theory and differential equations of probabilities are numerous. Examples are the work of Estes and of Bush and Mosteller, reported in Hilgard (Reference 13).

The models of Hull (Reference 25) and Spence (Reference 15) do not fit readily into any of Hilgard's classifications. To the writers they appear to be algebraic theories, based on the algebraic properties of the real number system. We present a simple example from Hull's formulation.

We define ${}_SH_R$ to be the "associative strength" of a particular stimulus onto a particular response. ${}_SH_R$ is variable with time, and its value depends on reinforcement history. We define ${}_S E_R$ to be the observed tendency to respond to the stimulus with the particular response in question. Hull postulates that

$${}_S E_R = V \cdot D \cdot K \cdot {}_S H_R \ ,$$

where V is stimulus intensity, D is drive level, and K is "incentive motivation," i. e., a measure of how "desirable" the results of activity will be. These various factors are said to be multiplicative, since if any one is zero, ${}_S E_R$ is zero. Other factors in Hull's complete theory become additive or subtractive. He thus uses the algebraic properties of the real number system to model behavior. In at least one case he defines a novel binary operation in terms of elementary arithmetic operations. If ${}_S H_R$ is the habit strength due to reinforced practice, and ${}_S \bar{\bar{H}}_R$ is habit strength due to transfer of other learning, Hull states that the combined strength, denoted by ${}_S H_R \dotplus {}_S \bar{\bar{H}}_R$, is given by

$${}_S H_R \dotplus {}_S \bar{\bar{H}}_R = {}_S H_R + {}_S \bar{\bar{H}}_R - {}_S H_R \cdot {}_S \bar{\bar{H}}_R$$

This last equation is of a form which occurs in adding probabilities. It can be interpreted in terms of the disjunction of two independent measurable sets.

Spence's approach is in some ways similar. He uses the algebraic properties of multiplication, addition, etc, on the real line, and also defines what are effectively mappings from one linearly ordered set into another.

Sudden impetus was given to biological modeling by Pitts and McCulloch, who pointed out in 1943 (Reference 26) that the behavior of a neuron can be partly described by appropriate Boolean algebra. This concept was instrumented by Rosenblatt in his "Perceptron" (Reference 27), which presumably modeled contingent reinforcement conditioning.

More recently, a number of researchers have investigated electromechanical systems that instrument some of these concepts. Examples are the work of Widrow (Reference 28) and Lee and Gilstrap (Reference 29). The investigators make some case for the notion that learning in living creatures is in some way directed by signals from the environment which change the strength of neural connections and which "reward" the creature for doing the "right" things and punish it for doingthe "wrong" things. The idea is clearly borrowed from contingency reinforcement experiments, and there is no doubt that such an electromechanical system resembles, at least superficially, a reinforcement conditioning experiment. Louis Fein (Reference 30) has pointed out that to his knowledge (and to the writers') none of the research into these models has resulted in "a particular experimentally verifiable piece of knowledge of how the brain works." But from a purely theoretical point of view, networks of threshold logic elements pose some interesting problems in analysis. Investigation of such networks and other bionic research have certainly contributed to technology in a number of ways, both by posing problems which had not previously been considered in technology and by suggesting novel "lifelike" solutions to long-standing problems from other fields.

Most behavior theorists today subscribe to some form of drive-reduction hypothesis. The research presented above has attempted to incorporate such a concept into the framework of present theory, thus making the theory more "lifelike," and thus by definition more <u>bionic</u>.

From a drive-reduction point of view, any stimulus becomes a positive or negative reinforcement only if it serves to reduce or increase the drive state of the subject at the time of reinforcement. The following paragraphs describe the Pitts-McCulloch model and point out how this model and experiments based on it fail to include certain known characteristics of both individual neuron behavior and those characteristics of gross creature behavior that lead to the drive-reduction concepts. Some of these lacks have been remedied by the research presented above. The attempt is certainly not to denigrate the significant contributions of Pitts, McCulloch, and others, but rather to point out that, like all first steps into a new field, their contributions must be refined, expanded, and ultimately integrated into the main body of scientific knowledge.

Although Pitts and McCulloch showed that certain aspects of neuron behavior could be modeled with Boolean algebra, their model ignored the observed time dependence of neuron behavior, usually referred to as temporal integration, and took little note of the fact that the rate of production of action potentials in sensory neurons is a function of the intensity of the stimulus applied.

Their model assumes that a neuron receives inputs that are either present or absent at any instant, and can be represented by a 1 when present and 0 when absent. Some endbulbs are excitatory and receive a weight of +1 when a signal is present; others are inhibitory and receive a weight of -1 when a signal is present. The neuron produces an output whenever the weighted sum of the instantaneous inputs exceeds its threshold. If $\theta$ represents production of an output, T is the threshold, $a_1, \ldots, a_e$ are variables which are 1 respectively when the separate excitatory endbulbs are energized, and $b_1, \ldots, b_i$ take similar roles for inhibitory inputs, one can write

$$\theta \longleftrightarrow \left( \sum_{j=1}^{e} aj - \sum_{j=1}^{i} bj \geq T \right)$$

as a description of the model. Since one afferent can have numerous endbulbs synapsing on one internuncial, the above can be modified to

$$\theta \longleftrightarrow \sum_{i=1}^{n} w_i a_i \geq T$$

where $w_i$ are weights associated with the separate afferents and $a_i$ now represent afferents, rather than endbulbs. In the Pitts-McCulloch model the $w_i$'s take on integral values. Later workers have allowed the $w_i$ to be continuous. The difference is only of minor interest, since the analytical results are so far the same.

Note that the equation

$$\sum_{i=1}^{n} w_i a_i = T$$

is the equation of a plane in an n-dimensional Euclidean space and that

$$\sum_{i=1}^{n} w_i a_i > T$$

is the equation of the half-space lying above the plane. Since each possible input configuration (minterm) can be represented by a vector of 1's and 0's, representing the $a_i$ in order, the model is a mapping of all such vectors into the space $E^1$. Further, the sets of vectors mapped to 0 and to 1 must be such that they can be separated from each other in n-space by a plane passed through the n-cube of which they are the vertices.

Needless to say, many logic functions do not meet this linear separability requirement. In fact, a majority of them do not. A simple logic function that cannot be instrumented by the model is to distinguish, for any fixed number of inputs greater than 2, between an odd number of inputs excited versus an even number.

Rosenblatt was the first to attempt to instrument the Pitts-McCulloch model physically and to introduce a notion of plasticity of neural connections. His "Perceptron" was modeled generally after a contingent reinforcement experiment. There are, however, several differences between the operation of his machine and contingent reinforcement conditioning.

The following describes generally the organization of most recent "learning" machines, such as Widrow's "Madeline" and Rosenblatt's "Perceptron." Consider a collection of threshold elements like those described above. Figure 12 shows three elements, which are enough for descriptive purposes. Each input to each threshold element is equipped with an adjustable weight, which can range continuously from, say, +1 to -1. The thresholds are fixed at some arbitrary value, say 0. 5. It is established a priori by the experimenter that he desires that the presentation of one subset of the possible input patterns should produce output No. 1, another subset should produce output No. 2, and a third should produce output No. 3. Further, no input patterns other than those selected should produce the outputs. Weights are set at any arbitrary initial values and then adjusted from time to time by the following orderly scheme:

(1) A pattern is presented on the inputs.

(2) If the output is what has previously been selected as "correct," no adjustments are made.

(3) If an output appears erroneously, the weights to the threshold element producing the erroneous output and associated with those inputs that are presently excited are reduced in value by some small increment. This has often been referred to in the literature as "punishing" the output.

(4) If the correct output fails to appear, those weights to the appropriate threshold element corresponding to inputs presently energized are increased ("rewarded") by a small increment.

(5) The procedure is repeated until the outputs are "correct" for all possible inputs or the experiment terminates.

On the surface, such experimentation appears to contain the elements of contingent reinforcement conditioning. There are, however, several criticisms that can be leveled at the model. Consider steps 1 and 4 of the adjustment procedure. The machine receives no "reinforcement" when it does the right thing (step 1) and receives a positive "reinforcement" when it does the wrong thing (step 4). To the writer, the scheme resembles classical conditioning more than it does instrumental conditioning, since the procedure consists of presenting a stimulus and contiguously forcing (or inhibiting) a response until the stimulus and response become associated. It is an axiom of instrumental conditioning that the first requirement is a motivated subject. The "motivation" of the Perceptron is nonexistent.

A second point of the model that can be criticized is its failure to handle "decisions" that are not linearly separable. One can readily show that a two-layer cascade arrangement of threshold elements can produce any desired logic function. To the writers' knowledge, however,

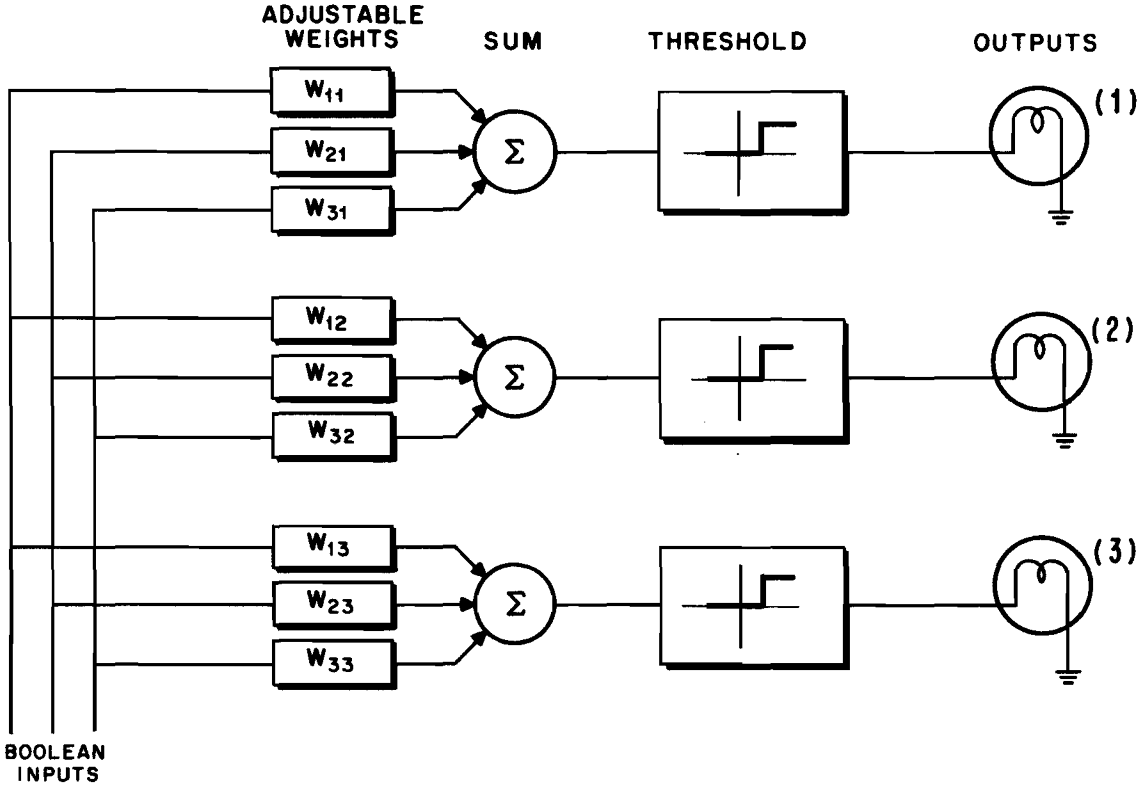


Figure 12. A Three-Element Threshold Logic Network

no one has yet exhibited an effective algorithm for adjusting the weights in such a configuration without access to the responses of the first layer. Appendix X presents the results of a conditioning experiment on rats involving a decision which is not a linearly separable logic function. The learning curves show the interference phenomenon that was predicted by the model of this report.

# APPENDIX V

## PREDICTION THEORY

The usual approach to the problem of prediction of stationary time series is the Wiener approach. One assumes a weakly stationary random process, x(t) such that the set of possible inputs to the predictor are elements of the process. One then tries to find a weighting function w(t) such that

$$x^*(t) = \int_{\infty}^{t} w(\tau, \sigma)x(t - \tau)d\tau$$

is the best least squares prediction of $x(t + \sigma)$; i. e.,

$$E\ [\ x^*(t) - x\ (t + \sigma)]^{\ 2}\ \text{is minimized.}$$

An alternate approach to the prediction problem was formulated by Kolmogorov almost simultaneously with Wiener's work. The Kolmogorov approach, in contrast to Wiener's, does not assume that the entire past history of the signal is available. When expressed in sampled form, the Komogorov formulation seems more suitable for application to the neuromime networks under study. Following is a discussion of this application of the Komogorov theory.

Suppose a function f(x), where both x and f(x) are real variables, is given in terms of equally spaced samples,

$$f(-n),\ .\ .\ .\ f(-2),\ f(-1),\ F(0), \tag{17}$$

and its next value f(1) is to be predicted. Further, suppose f(x) is weakly stationary, and therefore

(1) It has a finite second moment,

$$E\quad \{[f(x)]\ \} < \infty \quad , \tag{18}$$

where E stands for "expected value."

(2) The joint probability of any two of its values is a function of their separation only,

$$E\quad [f(x)\ f(x-\tau)] = \phi_{ff}(\tau). \tag{19}$$

(3) It has a continuous covariance function, $\phi_{ff}(\tau)$.

Assuming the prediction should be based only on the known samples of Equation 17, the simplest process of finding f(1) will be to consider it a linear combination of these samples:

$$f(1) = a_0f(0) + a_1f(-1) + .\ .\ . + a_nf(-n). \tag{20}$$

The problem is to find the appropriate values of the weights $a_i$.

Multiplying Equation 20 successively by the right-hand members of Equation 17, we get the following set of equations:

$$\left.\begin{aligned}
f(0)f(1) &= a_0f(0)f(0) + a_1f(0)f(-1) + \dots + a_nf(0)f(-n) \\
f(-1)f(1) &= a_0f(-1)f(0) + a_1f(-1)f(-1) + \dots + a_nf(-1)f(-n) \\
&\vdots \\
f(-1)f(1) &= a_0f(-n)f(0) + a_1f(-n)f(-1) + \dots + a_nf(-n)f(-n)
\end{aligned}\right\} \qquad (21)$$

The expected values of the left-hand members of each equation in Equation Set 21 will be the following set:

$$\left.\begin{aligned}
E\,[f(0)f(1)] &= a_0E\,[f(0)f(0)] + a_1E\,[f(0)f(-1)] + \dots + a_nE\,[f(0)f(-n)] \\
E\,[f(-1)f(1)] &= a_0E\,[f(-1)f(0)] + a_1E\,[f(-1)f(-1)] + \dots + a_nE\,[f(-1)f(-n)] \\
&\vdots \\
E\,[f(-n)f(1)] &= a_0E\,[f(-n)f(0)] + a_1E\,[f(-n)f(-1)] + \dots + a_nE\,[f(-n)f(-n)]
\end{aligned}\right\} \qquad (22)$$

However, due to the weak stationarity conditions given by Equations 18 and 19, the expected value $E\,[f(i)f(i \pm \xi)]$, exists and has the value

$$E\,[f(i)f(i \pm \xi)] = E\,[f(x)f(x \pm \xi)] \qquad (23)$$

Then, by making i successively equal to 0, -1, . . ., -n, and $\xi$ equal to 1, 0, -1, . . ., -n, we get from Equation Set 22

$$\left.\begin{aligned}
E\,[f(x)f(x+1)] &= a_0E\,[f(x)f(x)] + a_1E\,[f(x)f(x-1)] + \dots \\
&\qquad + a_nE\,[f(x)f(x-n)] \\
E\,[f(x)f(x+2)] &= a_0E\,[f(x)f(x+1)] + a_1E\,[f(x)f(x)] + \dots \\
&\qquad + a_nE\,\{f[x]\,f[x-(n-1)]\} \\
&\vdots \\
E\,\{f[x]\,f[x+(n+1)]\} &= a_0E\,[f(x)f(x+n)] + a_1E\,\{f[x]\,f[x+(n-1)]\} \\
&\qquad + \dots + a_nE\,[f(x)f(x)]
\end{aligned}\right\} \qquad (24)$$

By the condition of Equation 19,

$$E\,[f(x)f(x \pm \xi)] = \phi_{ff}(\pm \xi)$$

where $\phi_{ff}$ is the autocorrelation function of f(x). Consequently, (25)

$$\left.\begin{aligned}
\phi_{ff}(-1) &= a_0\phi_{ff}(0) + a_1\phi_{ff}(1) + \ldots + a_n\phi_{ff}(n) \\
\phi_{ff}(-2) &= a_0\phi_{ff}(-1) + a_1\phi_{ff}(0) + \ldots + a_n\phi_{ff}(n-1) \\
&\cdot \\
&\cdot \\
&\cdot \\
\phi_{ff}\,[-(n+1)] &= a_0\phi_{ff}(-n) + a_1\phi_{ff}\,[-(n-1)] + \ldots + a_n\phi_{ff}(0)
\end{aligned}\right\} \qquad (26)$$

Since f(x) was assumed to be a real variable, its autocorrelation is an even function, i.e.

$$\phi_{ff}(-\xi) = \phi_{ff}(\xi). \qquad (27)$$

Therefore,

$$\left.\begin{aligned}
\phi_{ff}(1) &= a_0\phi_{ff}(0) + a_1\phi_{ff}(1) + \ldots + a_n\phi_{ff}(n) \\
\phi_{ff}(2) &= a_0\phi_{ff}(1) + a_1\phi_{ff}(0) + \ldots + a_n\phi_{ff}(n-1) \\
&\cdot \\
&\cdot \\
&\cdot \\
\phi_{ff}(n+1) &= a_0\phi_{ff}(n) + a_1\phi_{ff}(n-1) + \ldots + a_n\phi_{ff}(0).
\end{aligned}\right\} \qquad (28)$$

This is a system of (n + 1) equations in the (n + 1) unknowns $a_0$, $a_1$, . . . $a_n$, and can be put in a matrix form suitable for a computer solution:

$$\begin{bmatrix}
\phi_{ff}(0) & \phi_{ff}(1) & \phi_{ff}(2) & \ldots & \phi_{ff}(n) \\
\phi_{ff}(1) & \phi_{ff}(0) & \phi_{ff}(1) & \ldots & \phi_{ff}(n-1) \\
\cdot & & & & \\
\cdot & & & & \\
\cdot & & & & \\
\phi_{ff}(n) & \phi_{ff}(n-1) & \phi_{ff}(n-2) & \ldots & \phi_{ff}(0)
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_n \end{bmatrix}
=
\begin{bmatrix} \phi_{ff}(1) \\ \phi_{ff}(2) \\ \cdot \\ \cdot \\ \cdot \\ \phi_{ff}(n+1) \end{bmatrix} \qquad (29)$$

It can be shown that the values of the weights $a_i$ found from Equation Set 29 minimize the square error between the left- and right-hand members of each equation in Equation Set 22.

If instead of the sample f(1) a more distant sample f(k) is desired, then replace f(1) by f(k) in Equation 20. It is easy to see that in this case Equation Set 29 becomes

$$\begin{bmatrix}
\phi_{ff}(0) & \phi_{ff}(1) & \phi_{ff}(2) & \ldots & \phi_{ff}(n) \\
\phi_{ff}(1) & \phi_{ff}(0) & \phi_{ff}(1) & \ldots & \phi_{ff}(n-1) \\
\cdot & & & & \\
\cdot & & & & \\
\cdot & & & & \\
\phi_{ff}(n) & \phi_{ff}(n-1) & \phi_{ff}(n-2) & \ldots & \phi_{ff}(0)
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_n \end{bmatrix}
=
\begin{bmatrix} \phi_{ff}(k) \\ \phi_{ff}(k+1) \\ \cdot \\ \cdot \\ \cdot \\ \phi_{ff}(k+n) \end{bmatrix}$$

$$k \geqq 1. \qquad (30)$$

In a similar fashion, a missing sample f(0) from a set of equally spaced samples of the function f(x),

$$f(-n),\ f\left[-(n-1)\right],\ \ldots\ f(-1),\ f(0),\ f(1),\ \ldots,\ f(m-1),\ f(m), \qquad (31)$$

can be reconstituted.

Similar to the previous analysis, we express f(0) as

$$f(0) = a_{-n}f(-n) + \ldots + a_{-1}f(-1) + a_1f(1) + \ldots + a_mf(m). \qquad (32)$$

After multiplying Equation 32 successively by the members of Equation 31, the expected values will be

$$\left.\begin{aligned}
E\left[f(-n)f(0)\right] &= a_{-n}E\left[f(-n)f(-n)\right] + \ldots + a_{-1}E\left[f(-n)f(-1)\right] \\
&\quad + a_1E\left[f(-n)f(1)\right] + \ldots + a_mE\left[f(-n)f(m)\right] \\
&\vdots \\
E\left[f(-1)f(0)\right] &= a_{-n}E\left[f(-1)f(-n)\right] + \ldots + a_{-1}E\left[f(-1)f(-1)\right] \\
&\quad + a_1E\left[f(-1)f(1)\right] + \ldots + a_mE\left[f(-1)f(m)\right] \\
E\left[f(1)f(0)\right] &= a_{-n}E\left[f(1)f(-n)\right] + \ldots + a_{-1}E\left[f(1)f(-1)\right] \\
&\quad + a_1E\left[f(1)f(1)\right] + \ldots + a_mE\left[f(1)f(m)\right] \\
&\vdots \\
E\left[f(m)f(0)\right] &= a_{-n}E\left[f(m)f(-n)\right] + \ldots + a_{-1}E\left[f(m)f(-1)\right] \\
&\quad + a_1E\left[f(m)f(1)\right] + \ldots + a_mE\left[f(m)f(m)\right]
\end{aligned}\right\} \qquad (33)$$

Using the conditions of Equations 19, 23, and 27 as before, Equation Set 33 becomes the set

$$\left.\begin{aligned}
\phi_{ff}(n) &= a_{-n}\phi_{ff}(0) + a_{-(n-1)}\phi_{ff}(1) + \ldots + a_{-1}\phi_{ff}(n-1) \\
&\quad + a_1\phi_{ff}(n+1) + \ldots + a_{m-1}\phi_{ff}(n+m-1) + a_m\phi_{ff}(n+m) \\
&\vdots \\
\phi_{ff}(1) &= a_{-n}\phi_{ff}(n-1) + a_{-(n-1)}\phi_{ff}(n-2) + \ldots + a_{-1}\phi_{ff}(0) \\
&\quad + a_1\phi_{ff}(2) + \ldots + a_{m-1}\phi_{ff}(m) + a_m\phi_{ff}(m+1) \\
\phi_{ff}(1) &= a_{-n}\phi_{ff}(n+1) + a_{-(n-1)}\phi_{ff}(n) + \ldots + a_{-1}\phi_{ff}(2) \\
&\quad + a_1\phi_{ff}(0) + \ldots + a_{m-1}\phi_{ff}(m-2) + a_m\phi_{ff}(m-1) \\
&\vdots \\
\phi_{ff}(m) &= a_{-n}\phi_{ff}(n+m) + a_{-(n-1)}\phi_{ff}(n+m-1) + \ldots + a_{-1}\phi_{ff}(m+1) \\
&\quad + a_1\phi_{ff}(m-1) + \ldots + a_{m-1}\phi_{ff}(1) + a_m\phi_{ff}(0)
\end{aligned}\right\} \qquad (34)$$

In matrix form, Equation Set 34 is

$$[A] = \begin{bmatrix} \phi_{ff}(0) & \phi_{ff}(1) \, . \, . \, . \, . \, . \, \phi_{ff}(n-1) & \phi_{ff}(n+1) \, . \, . \, . \, \phi_{ff}(m+n-1) & \phi_{ff}(m+n) \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ \phi_{ff}(n-1) & \phi_{ff}(n-2) \, . \, . \, . \, . \, \phi_{ff}(0) & \phi_{ff}(2) \, . \, . \, . \, . \, \phi_{ff}(m) & \phi_{ff}(m+1) \\ \phi_{ff}(n+1) & \phi_{ff}(n) \, . \, . \, . \, . \, . \, \phi_{ff}(2) & \phi_{ff}(0) \, . \, . \, . \, . \, \phi_{ff}(m-2) & \phi_{ff}(m-1) \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ \phi_{ff}(n+m) & \phi_{ff}(n+m-1) \, . \, . \, \phi_{ff}(m+1) & \phi_{ff}(m-1) \, . \, . \, \phi_{ff}(1) & \phi_{ff}(0) \end{bmatrix}$$

therefore

$$[A] = \begin{bmatrix} a_{-n} \\ \cdot \\ \cdot \\ \cdot \\ a_{-1} \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_m \end{bmatrix} = \begin{bmatrix} \phi_{ff}(n) \\ \cdot \\ \cdot \\ \cdot \\ \phi_{ff}(1) \\ \phi_{ff}(1) \\ \cdot \\ \cdot \\ \cdot \\ \phi_{ff}(m) \end{bmatrix} \tag{35}$$

Finally, suppose a second function g(x) is known to be the result of the function f(x) being modified by a linear operator whose response is to be determined. This is the case of plant identification. In the general case, the present state of the response depends on all past values of the input function f(x). We may write

$$g(0) = a_0 f(0) + a_1 f(-1) + \ldots + a_n f(-n) \tag{36}$$

Successive multiplication of the terms of Equation 36 by the terms of Equation 17 yields

$$\left.\begin{aligned} E\,[f(0)g(0)] &= a_0 E\,[f(0)f(0)] + a_1 E\,[f(0)f(-1)] + \ldots \\ &\qquad + a_n E\,[f(0)f(-n)] \\ E\,[f(-1)g(0)] &= a_0 E\,[f(-1)f(0)] + a_1 E\,[f(-1)f(-1)] + \ldots \\ &\qquad + a_n E\,[f(-1)f(-n)] \\ &\cdot \\ &\cdot \\ &\cdot \\ E\,[f(-n)g(0)] &= a_0 E\,[f(-n)f(0)] + a_1 E\,[f(-n)f(-1)] + \ldots \\ &\qquad + a_n E\,[f(-n)f(-n)] \,. \end{aligned}\right\} \tag{37}$$

Therefore,

$$\left.\begin{aligned}
\phi_{fg}(0) &= a_0\phi_{ff}(0) + a_1\phi_{ff}(1) + \ldots + a_n\phi_{ff}(n) \\
\phi_{fg}(-1) &= a_0\phi_{ff}(-1) + a_1\phi_{ff}(0) + \ldots + a_n\phi_{ff}(n-1) \\
&\;\;\vdots \\
\phi_{fg}(-n) &= a_0\phi_{ff}(-n) + a_1\phi_{ff}\ [-(n-1)]\ + \ldots + a_n\phi_{ff}(0),
\end{aligned}\right\} \tag{38}$$

where

$$\left.\begin{aligned}
\phi_{fg}(\xi) &= E\ [f(x)g(x-\xi)] \\
\phi_{ff}(\xi) &= E\ [f(x)f(x-\xi)]
\end{aligned}\right\} \tag{39}$$

Here again, because of Equation 27,

$$\left.\begin{aligned}
\phi_{fg}(0) &= a_0\phi_{ff}(0) + a_1\phi_{ff}(1) + \ldots + a_n\phi_{ff}(n) \\
\phi_{fg}(-1) &= a_0\phi_{ff}(1) + a_1\phi_{ff}(0) + \ldots + a_n\phi_{ff}(n-1) \\
&\;\;\vdots \\
\phi_{fg}(-n) &= a_0\phi_{ff}(n) + a_1\phi_{ff}(n-1) + \ldots + a_n\phi_{ff}(0);
\end{aligned}\right\} \tag{40}$$

and in matrix form,

$$\begin{bmatrix}
\phi_{ff}(0) & \phi_{ff}(1) & \phi_{ff}(2) \ldots\ldots \phi_{ff}(n) \\
\phi_{ff}(1) & \phi_{ff}(0) & \phi_{ff}(1) \ldots\ldots \phi_{ff}(n-1) \\
\cdot & & \\
\cdot & & \\
\cdot & & \\
\phi_{ff}(n) & \phi_{ff}(n-1) & \phi_{ff}(n-2) \ldots \phi_{ff}(0)
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_n \end{bmatrix}
=
\begin{bmatrix} \phi_{fg}(0) \\ \phi_{fg}(-1) \\ \cdot \\ \cdot \\ \cdot \\ \phi_{fg}(-n) \end{bmatrix} \tag{41}$$

The negative sign in the argument of the cross-correlation function $\phi_{fg}$ means that the physical significance of the weights $a_i$ may be deduced from the output g(x) of a linear device being caused by an input f(x);

$$g(x) = \int_0^x f(x-\tau)w(\tau)d\tau \tag{42}$$

where $w(\tau)$ is the unit input response or weighting function of the device.

For the case of a discrete, instead of continuous, system where f(x), and therefore g(x), is given in terms of equally spaced samples, Equation 42 has the counterpart

$$g(x) = \Sigma w(\tau)f(x - \tau) \qquad (43)$$

$$= 0,\ 1,\ 2,\ \ldots\quad .$$

hence

$$g(x) = w(0)f(x) + w(1)f(x - 1) + \ldots + w(n)f(x - n). \qquad (44)$$

Since this equation is true for any value of $x \geq 0$, it will hold for $x = 0$, i.e.,

$$g(0) = w(0)f(0) + w(1)f(-1) + \ldots + w(n)f(-n). \qquad (45)$$

It can be seen from Equations 36 and 45 that

$$a(i) = w(i) \qquad (46)$$

$$i = 0,\ 1,\ 2,\ \ldots\quad .$$

Therefore, $a_i$ is the value of the system input response at the point $x = i$.

# APPENDIX VI

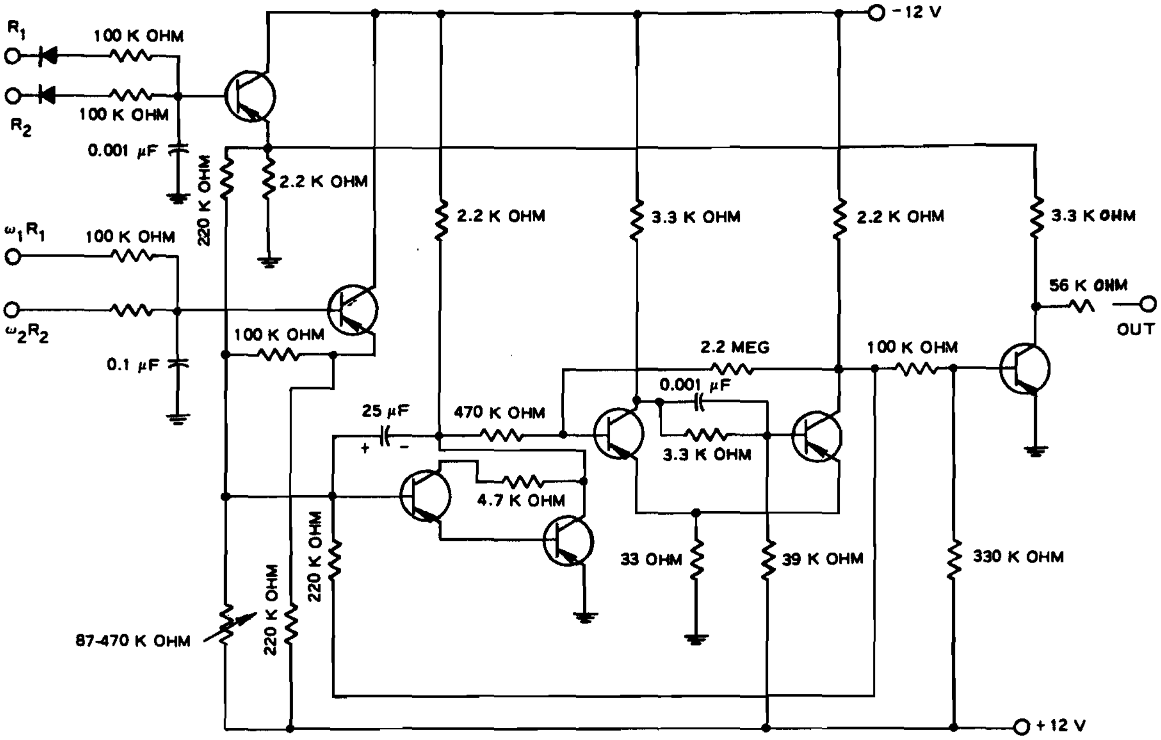
## EQUIPMENT DEVELOPMENT

### A. TRANSISTOR CIRCUIT DEVELOPMENT

Figure 13 shows the transistor threshold circuit of the modified neuromime at the beginning of the program. Tests run on neuromime networks during the contract period indicated certain circuit changes and additions. The threshold circuit shown in Figure 1 of this report evolved from these changes and additions.

Figure 14 is a block diagram showing the threshold circuit operation. The threshold circuit has two inputs and one output. The two inputs are the sums of weighted and unweighted neuromime pulses. The neuromime's output is a burst of pulses whose repetition rate is roughly a concensus of the neuromime's input activity. A separate circuit accepts the first negative-weighted input in a series of inputs and injects a short pulse into the summation point of the threshold integrator. The neuromime output is fed back with a reversed sign and integrated with the two input summations and the short injected pulse. This integral is then summed with the rest threshold potential and the neuromime's output. The threshold equation is then

$$T = T_0 - KR_0 - \int_0^t \left[ \Sigma R_i + g(t)\ \Sigma W_i R_i - h(t)\ W_i e^{-nt} - AR_0 \right] dt.$$



NOTE:
ALL TRANSISTORS ARE 2N404 (RCA)

Figure 13. Original Neuromime Transistor Circuit

where

$T$ = threshold

$T_o$ = rest threshold

$R_i$ = ith input to neuromime

$W_i$ = ith weight associated with ith input

$R_o$ = neuromime output

$g(t)$ = 0 when $W_iR_i < 0$
= 1 when $W_iR_i > 0$

$h(t)$ = 0 when all $W_i < 0$
= 1 when at least one $W_i$, say $W_k$, $< 0$

$K, A, n$ = constants

$t$ = time variable.

When the neuromime firing threshold was exceeded, the trigger circuit shown in the diagram triggered the gate, and the neuromime fired a burst of pulses. The threshold circuit determined the length of the burst of pulses, and also how long the neuromime waited before producing another burst of pulses. Without inputs, the threshold circuit returned to its quiescent state.

Tests were run on the threshold circuit to determine its response to inputs, weighted and unweighted. The response curves are shown in Figures 15, 16, 17, and 18.

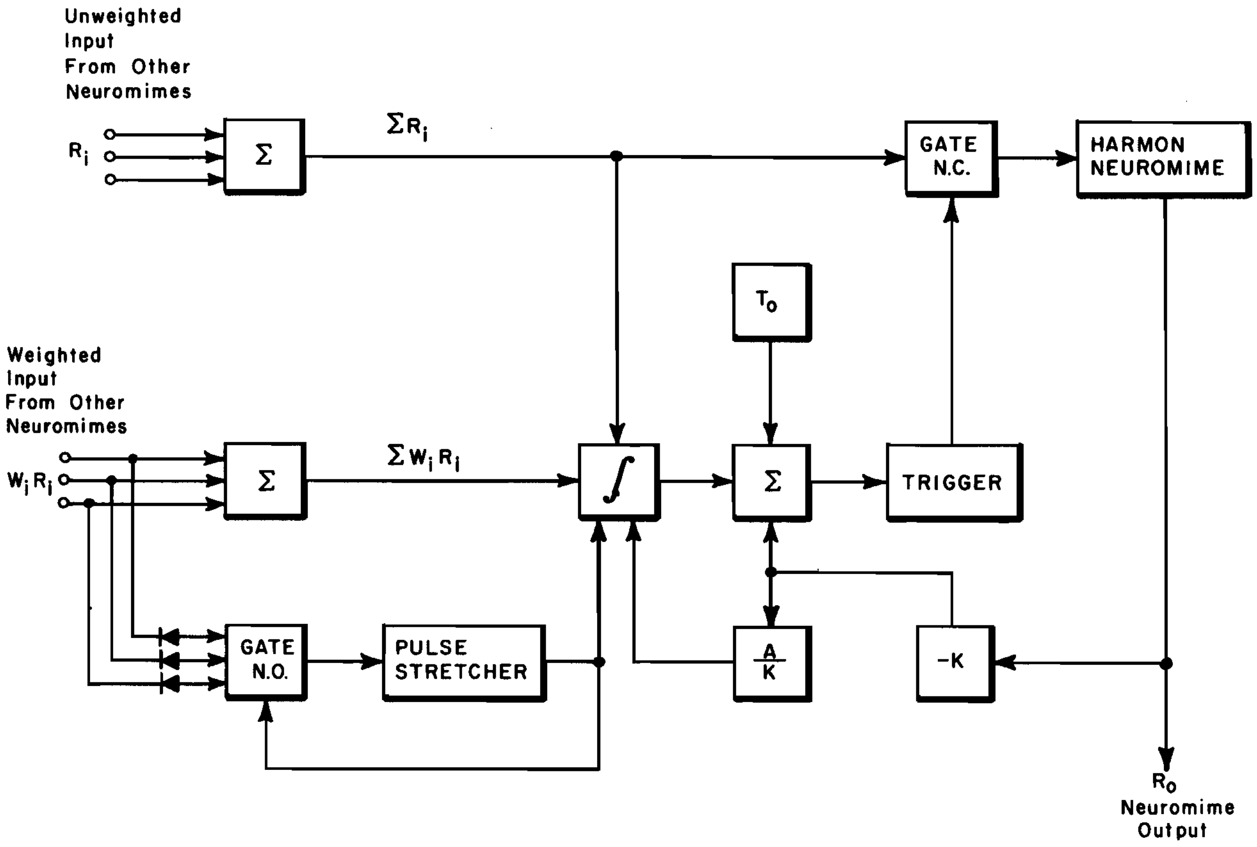


Figure 14. Block Diagram of Threshold Circuit

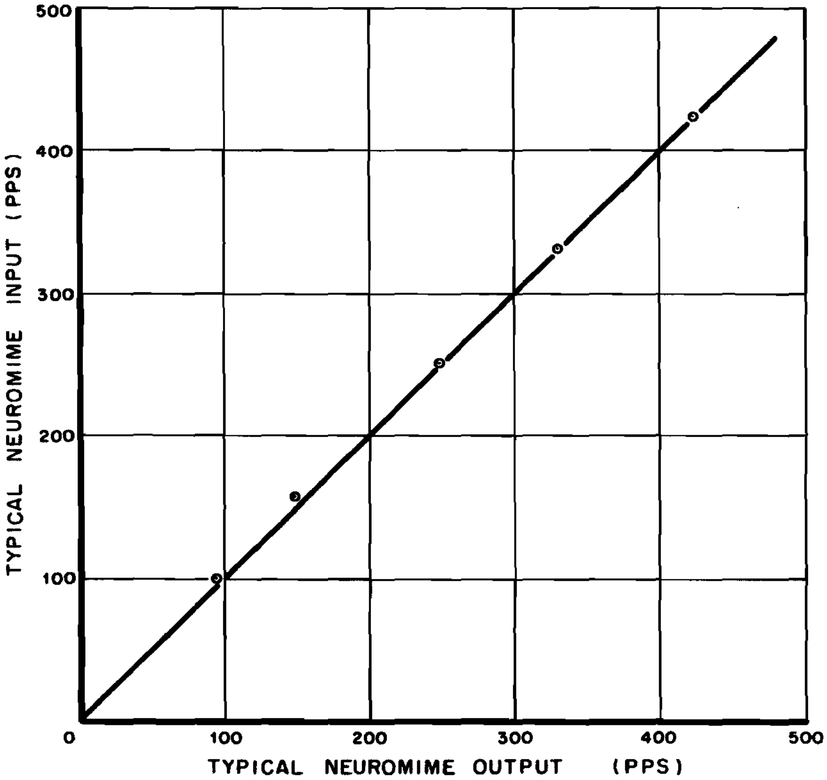


Figure 15. Neuromime Response to Pulses

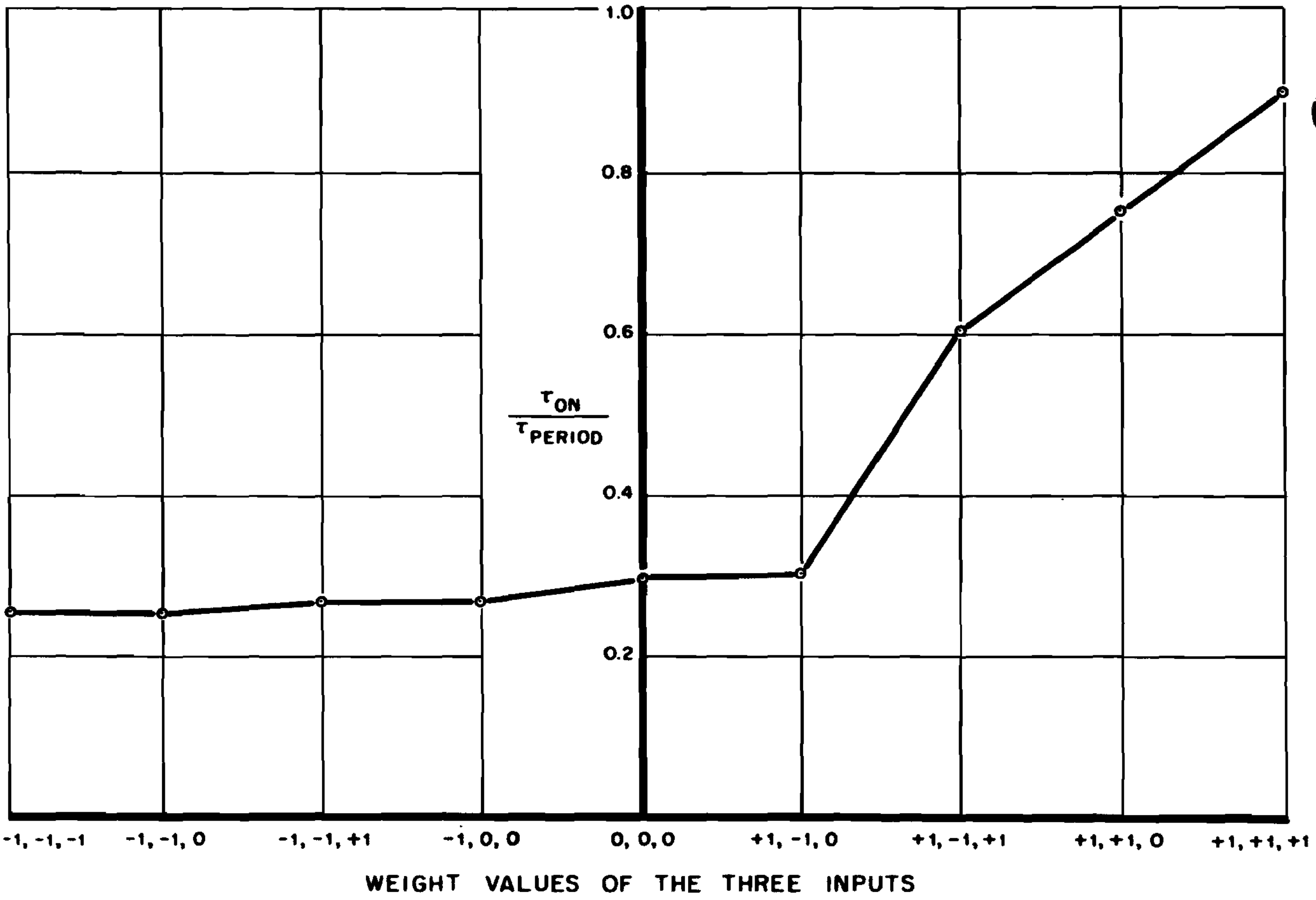


Figure 16. Weighted Inputs

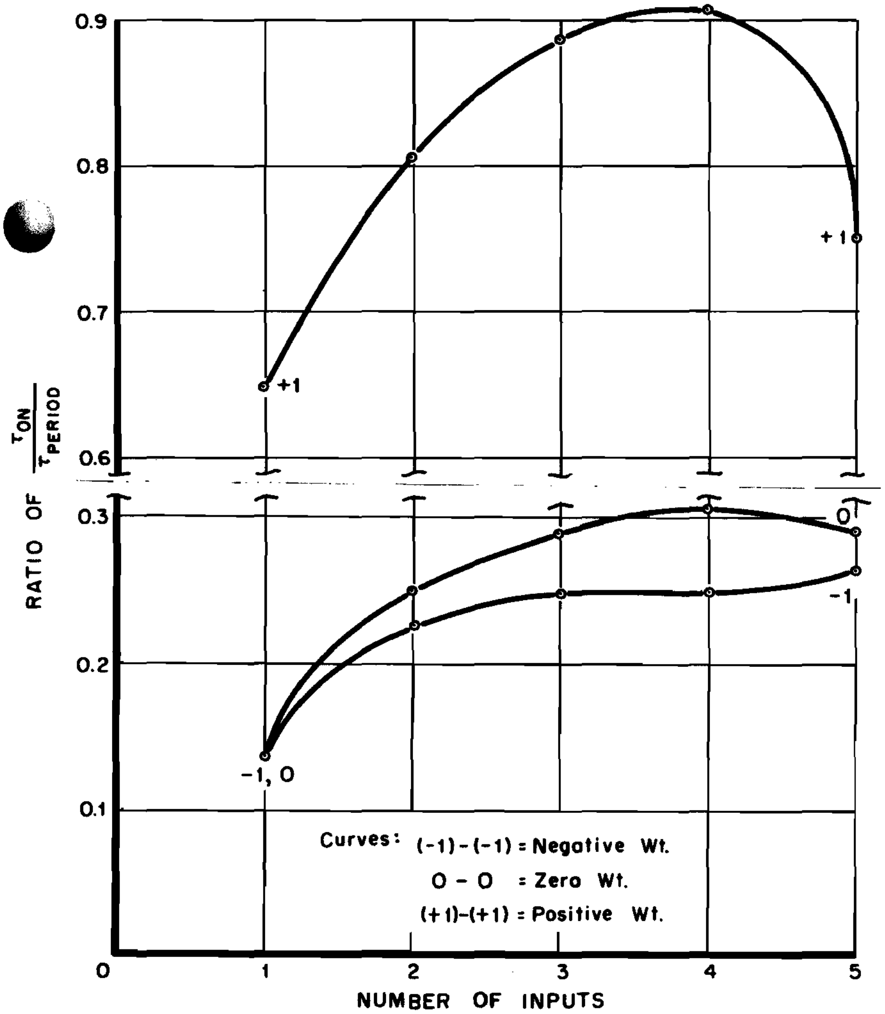


Figure 17. Neuromime Response to Several Inputs

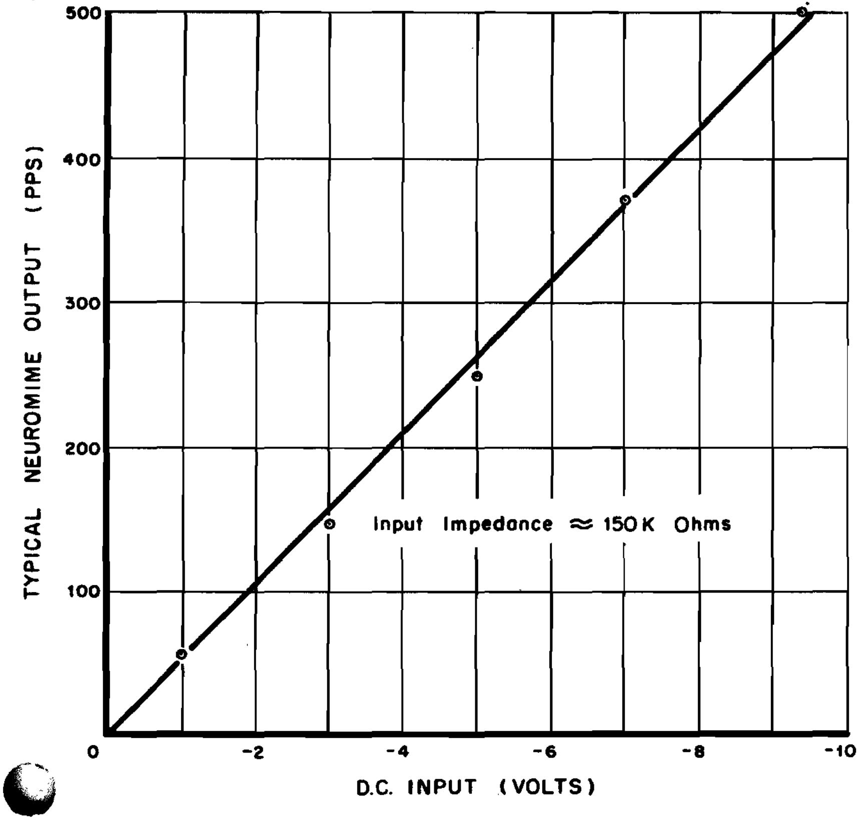


Figure 18. Neuromime Response to Direct Current

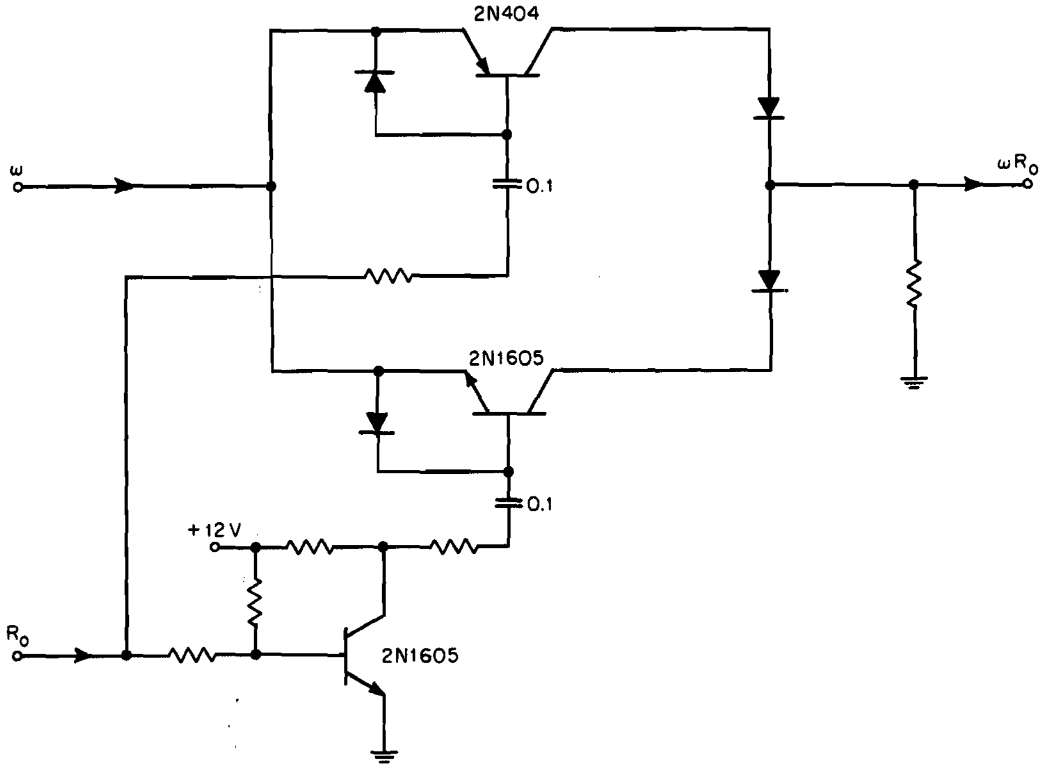


Figure 19. Transistor Gating Circuit

In addition to the circuitry developed for the neuromime model, a transistor circuit was developed as support for the neuromime hardware. This circuit was designed and built to combine a neuromime's output pulse rate and synaptic weight. The circuit turned the synaptic weight (d-c voltage) on and off at the same rate as that of the neuromime. The result was a voltage porportional to the synaptic weight, with a pulse rate equal to that of the neuromime. The circuit is shown in Figure 19.

## B. ANALOG CIRCUIT DEVELOPMENT

The analog computer has played two roles in the development of the neuron model. The first role was that of the synaptic weight computer used as support for the transistor hardware. The second role was that of the total neuron model simulation.

Figure 20 shows the analog computer simulation of the synaptic weight computer at the beginning of the program. Tests on the neuromime networks led to several revisions of the analog weight computer circuit. Figure 21 shows the resultant analog computer circuit.

The block diagram of the synaptic weight computer simulation is shown in Figure 22. The weight computer has three inputs and one output. The summation of the neuromime's inputs is the first of the three inputs to the weight computer. It goes through a $\delta(S)$ circuit, which yields, roughly, the time changes of the summation. The changes enter a gate circuit, which is controlled by a gating function, g(S). The gate circuit passes the summation changes unimpeded if it is "open." When closed, no changes pass through the gate circuit. The two remaining inputs to the weight computer, the neuromime's own firing rate and a function, $h_i$, are used to trigger the gating function.

The gating function, g(S) in the block diagram, controlled the inputs to the weight integrator by operating a gate. As long as the gating function received an input, the gate was

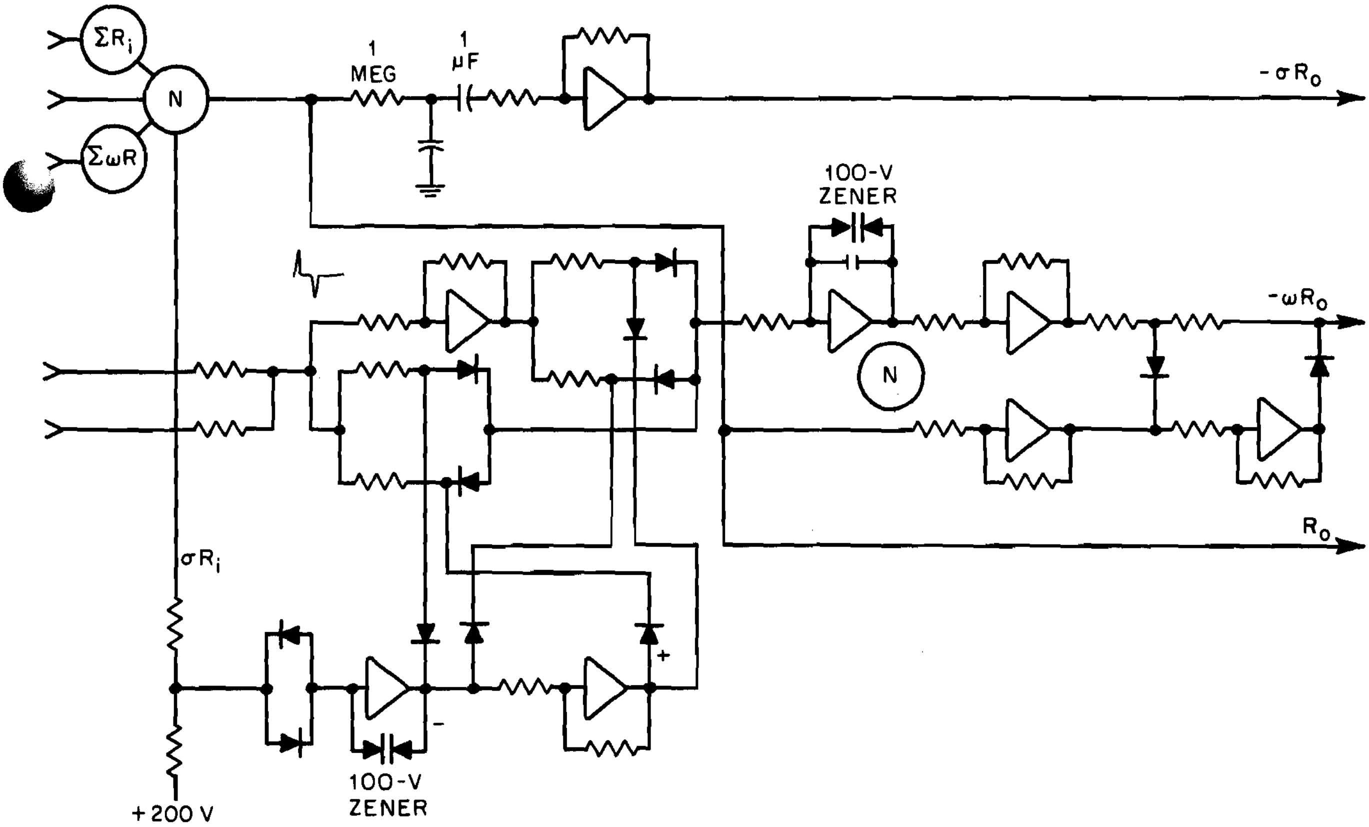


Figure 20. Original Analog Computer Circuit for Weight Computer

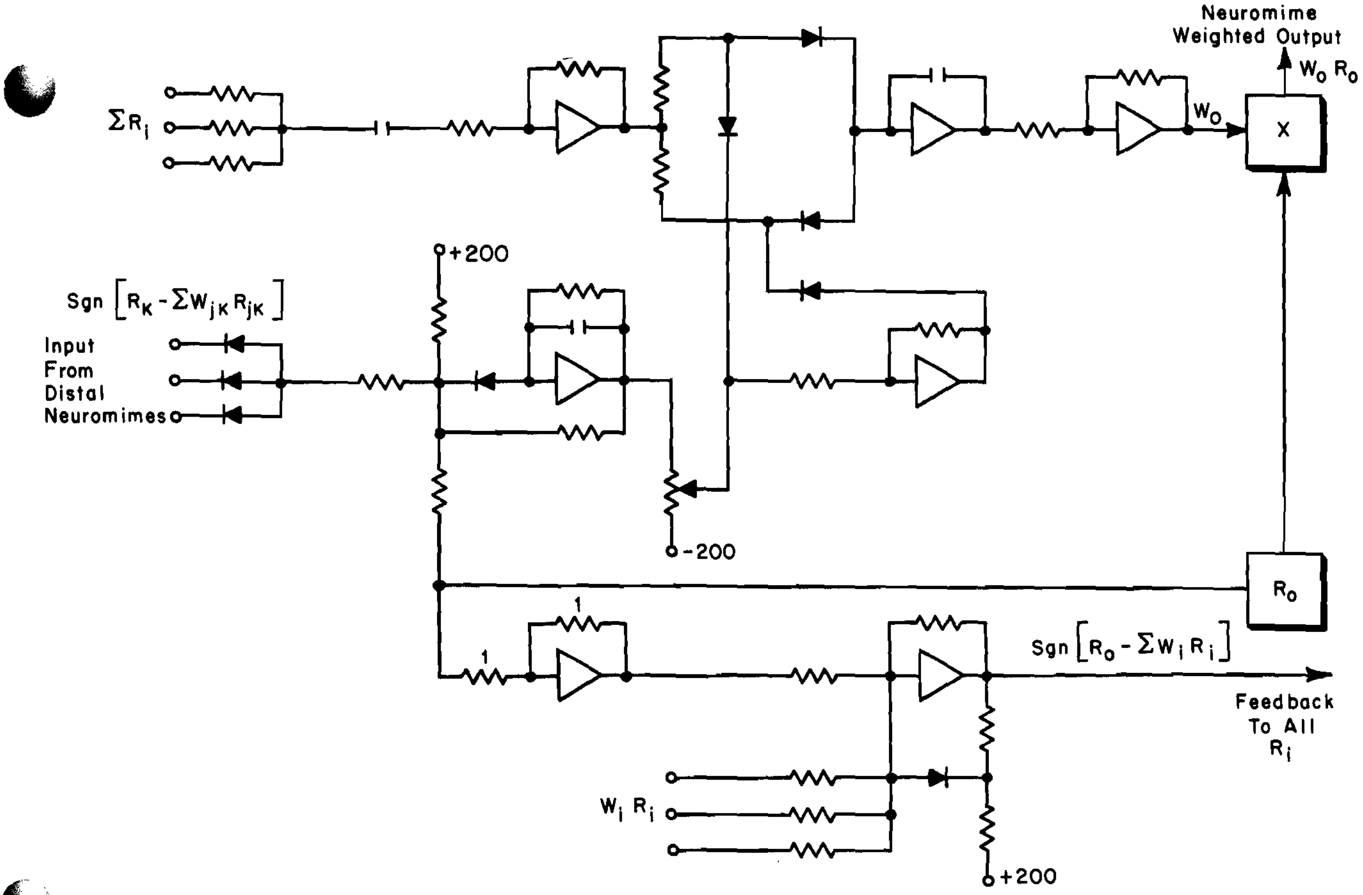
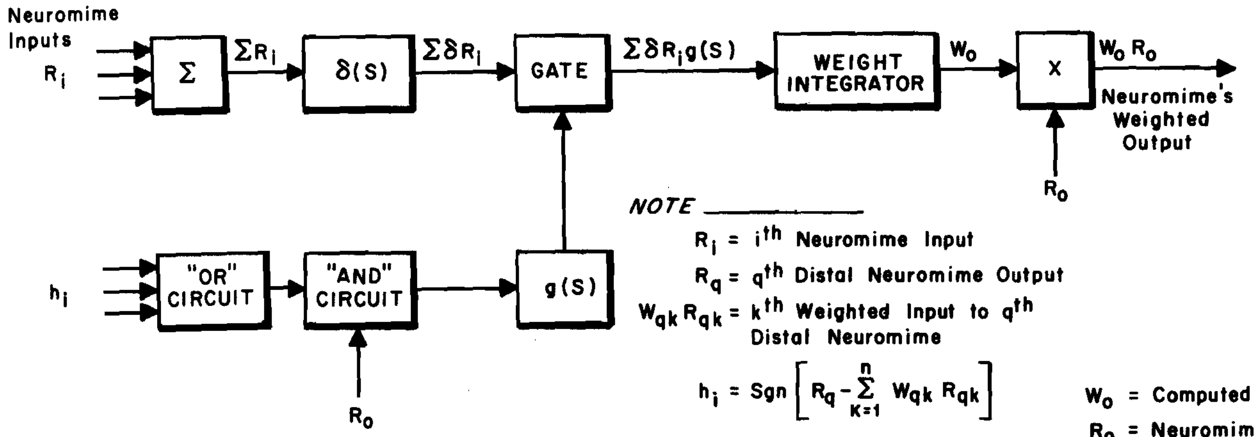


Figure 21. Resultant Synaptic Weight Computer Circuit



*NOTE*

$R_i$ = $i^{th}$ Neuromime Input

$R_q$ = $q^{th}$ Distal Neuromime Output

$W_{qk} R_{qk}$ = $k^{th}$ Weighted Input to $q^{th}$ Distal Neuromime

$$h_i = \mathrm{Sgn}\left[R_q - \sum_{K=1}^{n} W_{qk} R_{qk}\right]$$

$$\delta(S) = \frac{\tau S}{\tau S + 1}$$

$W_o$ = Computed Weight

$R_o$ = Neuromime Output

$g(S)$ = Gating Function

Figure 22. Block Diagram of Synaptic Weight Computer

"open" and the weight integrator received inputs. If the input to the gating function disappeared, the gating function gradually "closed" the gate so that the weight integrator received smaller and smaller inputs until the gate "closed."

The gating function was the most critical circuit of the weight computer simulation, and subject to more changes than any other weight computer circuit. It controlled the times that the weight integrator could compute. Figure 23A shows the weight integrator gate circuit at the beginning of the contract. The neuromime's gating circuit let the weight integrator integrate the inputs during the neuromime's firing period.

When the distal neuromimes stopped firing, the gating circuit changed the algebraic signs of the weight integrator inputs and let the synaptic weight integration continue. The gating circuit was changed as shown on Figure 23B to let the weight integrator work only during the distal neuromimes' firing cycle.

Further neural net tests showed that the gating circuit should allow some weight computation after a neuromime stops firing. The gating circuit was changed to let the weight integrator integrate during a neuromime's firing time and shortly after (see Figure 23C).

Further tests pointed out another fault with the synaptic weight computer. The gating function circuit allowed some neuromimes to compute synaptic weights without ever firing. Figure 23D shows a gating function circuit that corrected this fault. In this circuit, no synaptic weight could be computed unless the neuromime was firing.

Other tests showed that the gating function was adequate when a neuromime had a single input, but was inadequate when the neuromime had more than one input. A neuromime was not getting enough information from its distal neuromimes to discover whether or not it was causing the distal neuromimes to fire. The gating circuit shown in Figure 24 allows a neuromime to compute a synaptic weight only if it is firing, or has recently fired, and at least one of the distal neuromimes is firing against the orders of its weighted inputs.

The second role played by the analog computer was that of the complete simulation of the neuron model. This neuron model development was inspired by the application of optimal control theory to neuromime networks. Two basic models were tested, each using a different method to compute its synaptic weights.

Figure 25 shows the first neuron model using the optimal control concept. In this model, the synaptic weight computer computed the weights by comparing a "predicted" trajectory with the actual trajectory of the input signal. Any difference between the "predicted" trajectory and the actual trajectory was used to correct the synaptic weights.

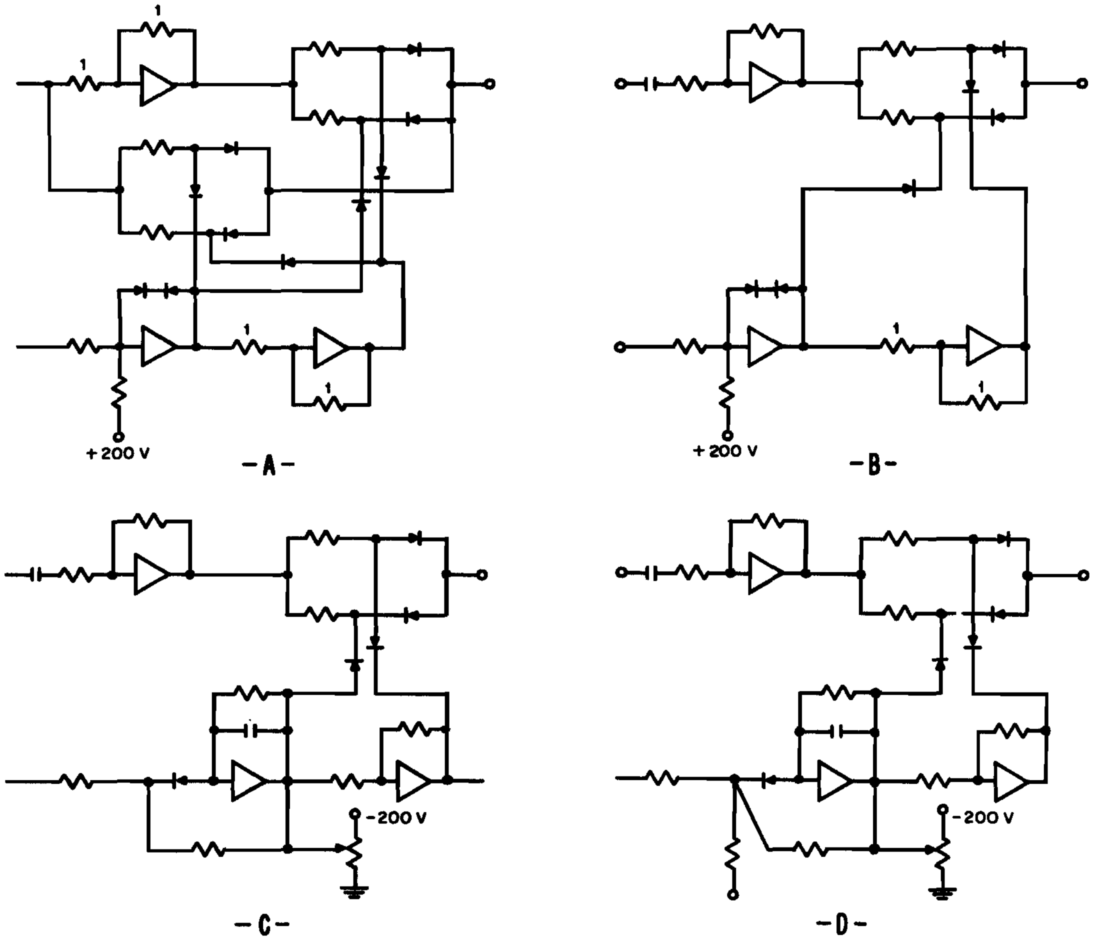


Figure 23. Gating Circuit Development

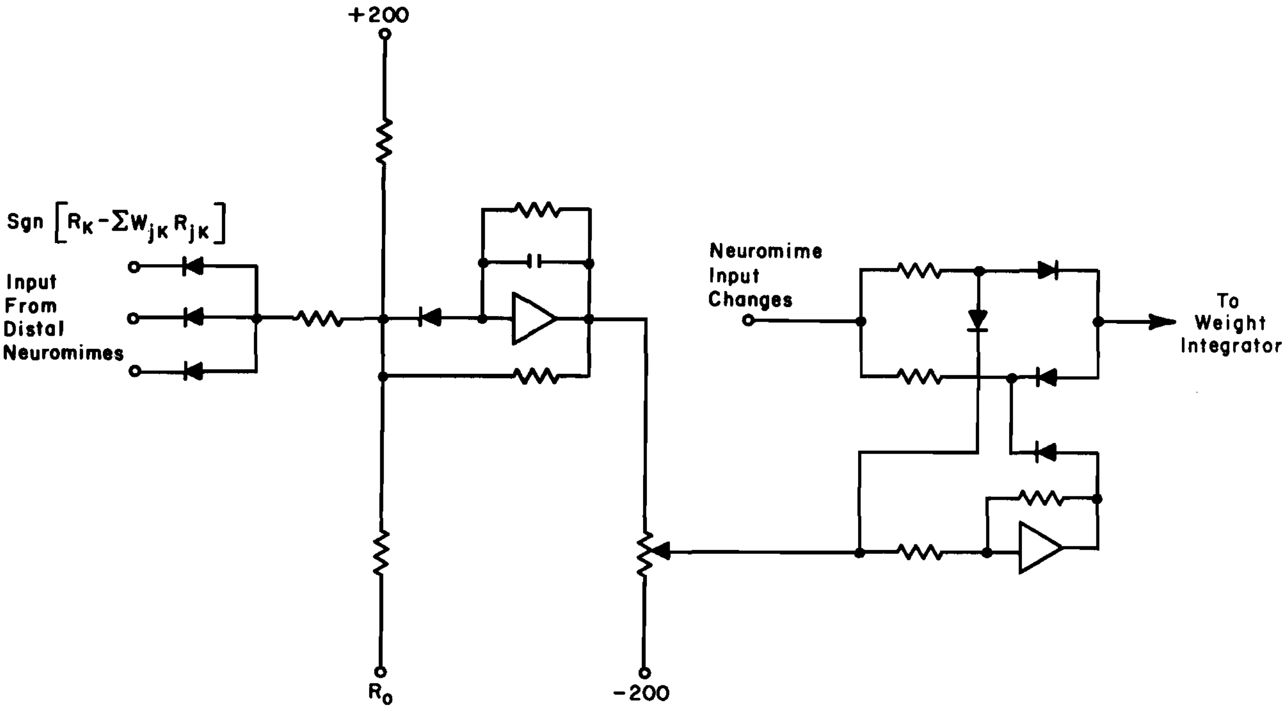


Figure 24. Analog Gating Function Circuit

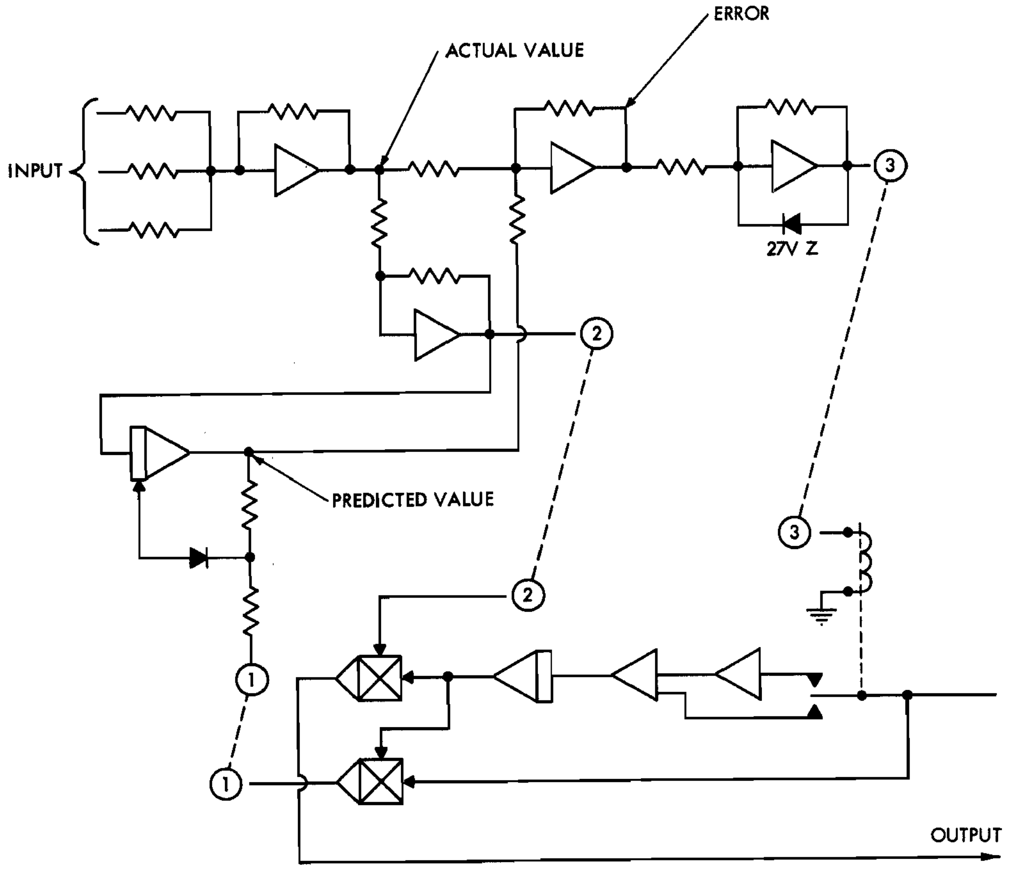


Figure 25. First Neuron Model Using Optimal Control Concept

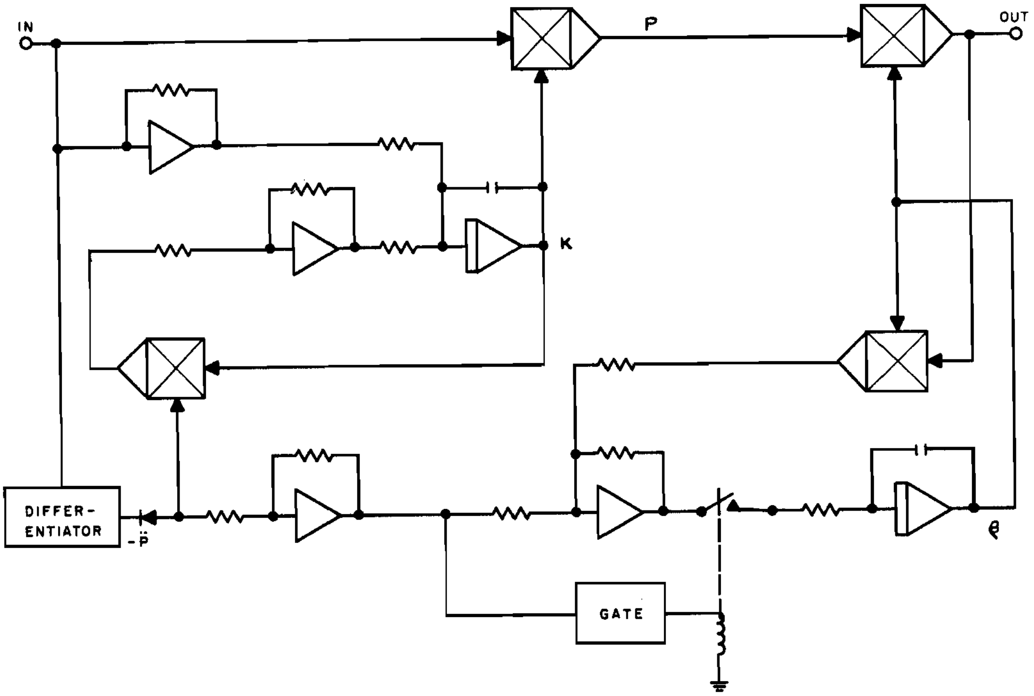


Figure 26. Second Neuron Model Using Optimal Control Concept

The method used to compute the synaptic weights was based on the optimal control equations given in the text. However, preliminary tests showed that this computing method was too difficult to instrument for neuron models having several inputs.

Figure 26 shows the second basic neuron model using the optimal control concept. The synaptic weight computing method for this model was based on a modification of the optimal control equations. The modification is an assumption. To compute the synaptic weights, the neuron model assumes that the optimal control output is a linear combination of the environment outputs. The neuron model computes the entries for a transformation matrix. This matrix maps the environment output vector into the optimal control vector.

The development of the second basic neuron model was pursued in lieu of the first. The second neuron model showed the ability to compute optimum synaptic weights as well as the ability to accept large numbers of inputs.

Figure 27 shows the results of the neuron model development. Comparison with the neuron model in Figure 26 shows the various changes. Briefly, the significant changes are the addition of a sign-taking circuit and a circuit that indicates when an input variable and its time derivative have opposite signs. The sign-taking circuit enables the neuron model to use a single error signal to compute many parameters. The technique for computing the synaptic weights using the sign-taking circuit is a form of hill-climbing, and is also known as the gradient method of steepest descent. The second additional circuit evolved from certain test results which revealed that the optimal control equations developed were not valid when an environmental disturbance was present. To obviate this invalidity, it was postulated that a disturbance was present when an input variable and its time derivative had the same sign. (This postulate is not true for all cases, but the approximation is good for transient conditions.) This circuit, then, was designed to stop synaptic weight computations during disturbances. Tests on the neuron model using these additional circuits have proved their usefulness.

Two additional circuits, shown in Figures 28 and 29, were developed for use in the simulation of an adaptive autopilot using neuron model networks to control a two-channel aircraft. These transistor circuits were designed for three purposes:

(1) The circuits supported the extension of the previous simulation to the more complex two-channel aircraft adaptive autopilot.

(2) The circuits provided a more reliable switching response than could be obtained from the analog switching simulation.

(3) These transistor circuits represented the beginning of the reduction of the analog computer simulation to hardware.

The first transistor circuit yields the algebraic sign of its input signal as its output. The equations relating the circuit's input and output are

$$\text{if:} \quad y > 0 \qquad x = +1$$

$$y < 0 \qquad x = -1$$

$$y = 0 \qquad x = 0$$

where y is the input signal and x is the output. This circuit replaced two operational amplifiers in the analog computer simulation of the neuron model.

The second circuit was a gate circuit which was triggered by the output of the first transistor circuit discussed above. The specific task of this circuit was to accept an analog voltage and, depending on the sign of the trigger signal, yield either an inverted or non-inverted signal proportional to the input voltage. If the trigger signal is zero, this circuit has no output voltage. The two circuits, the aircraft simulation unit,and a unit of five neuromimes are shown in Figure 30.

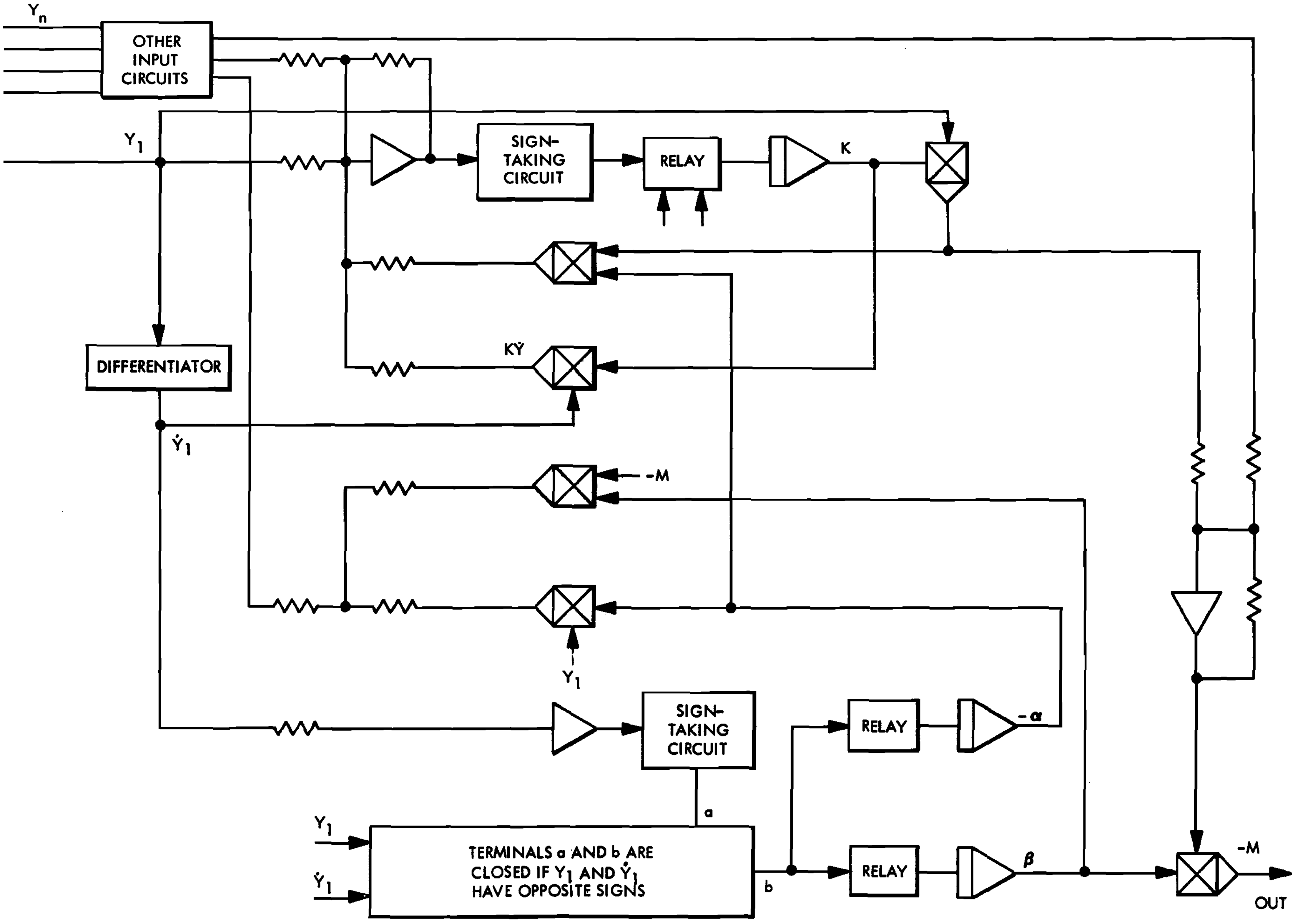


Figure 27. Present Neuron Model

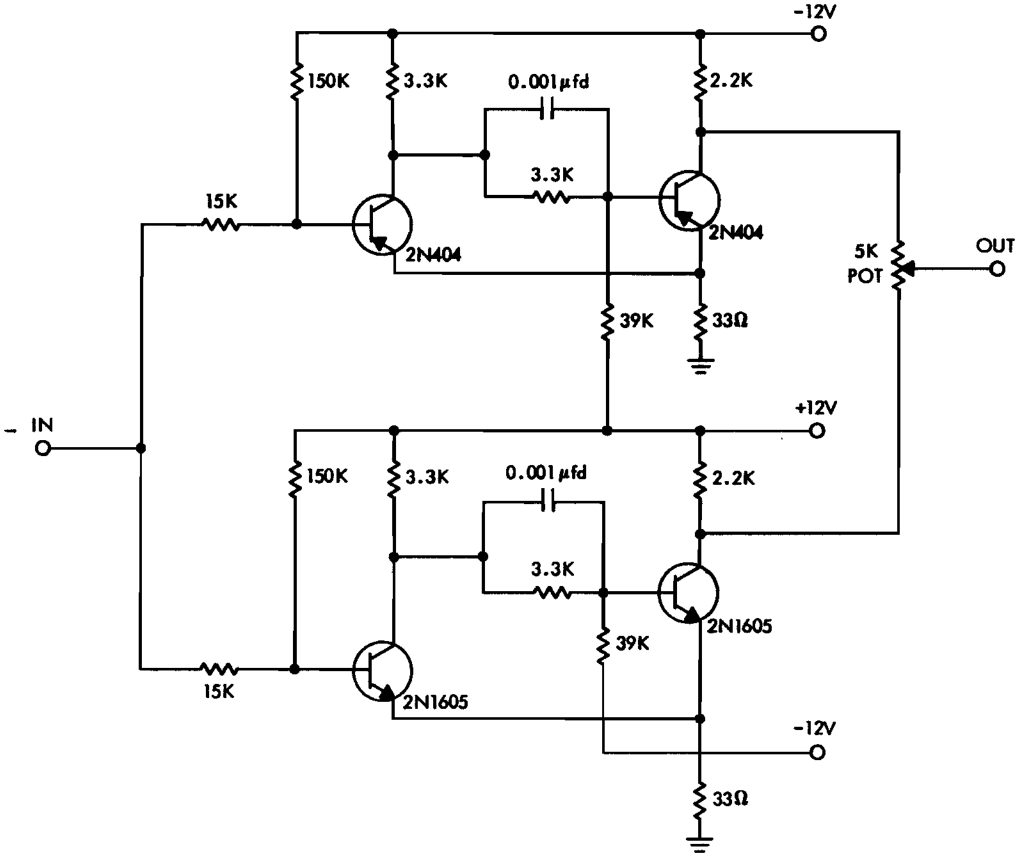


Figure 28. Signum Circuit

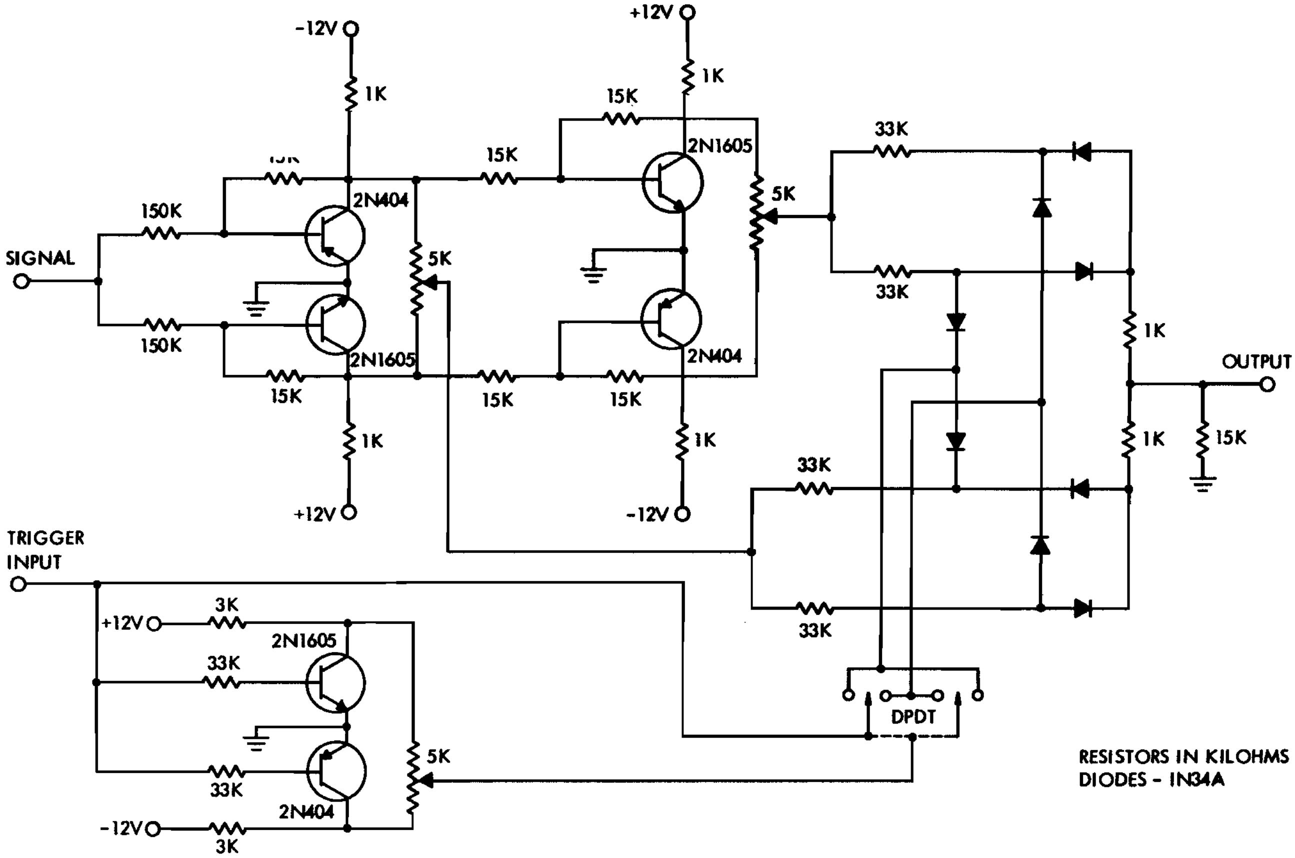
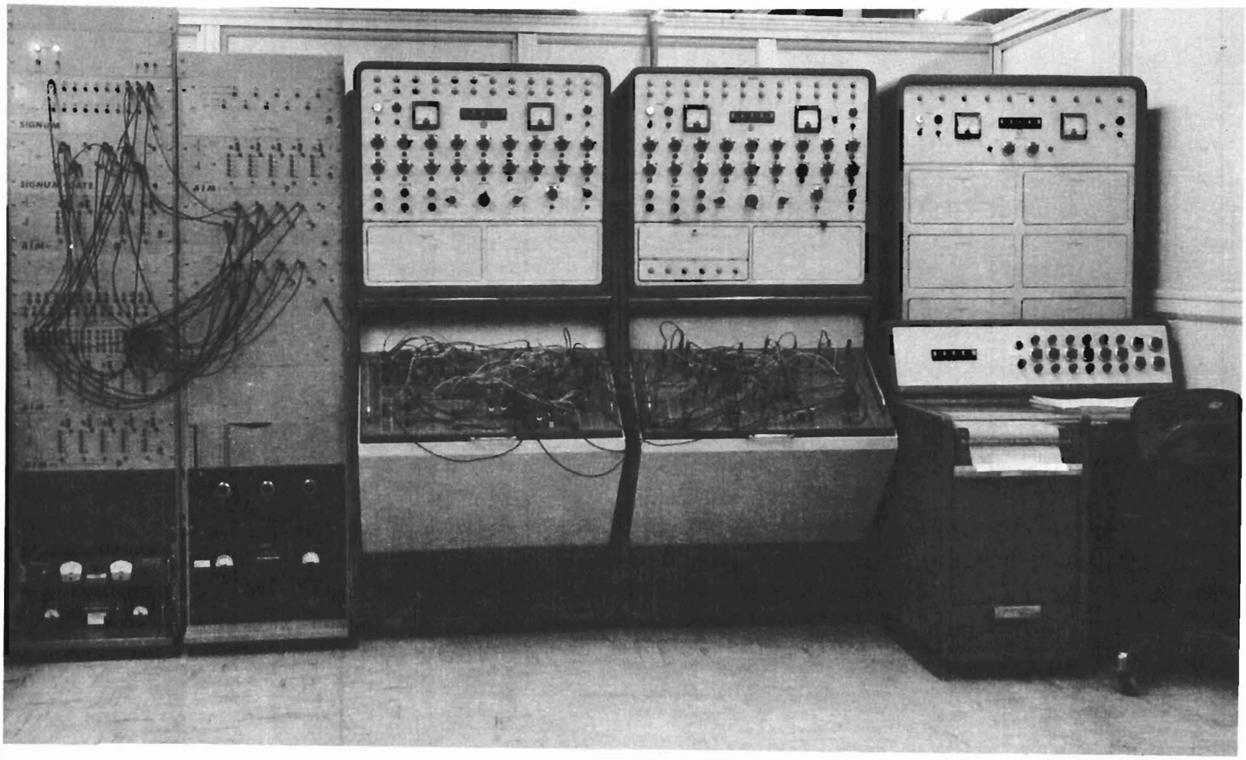
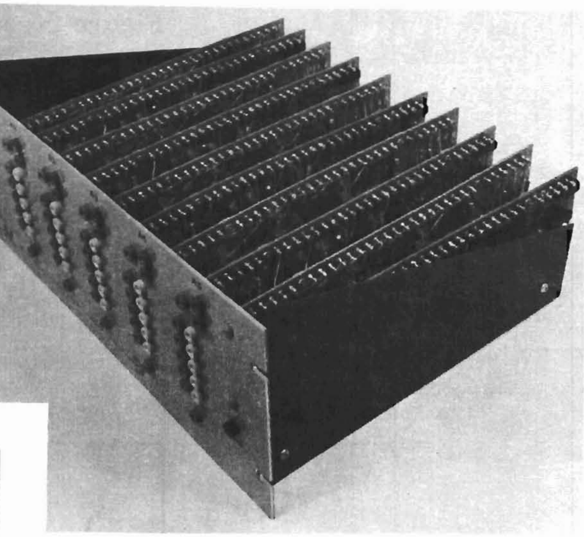


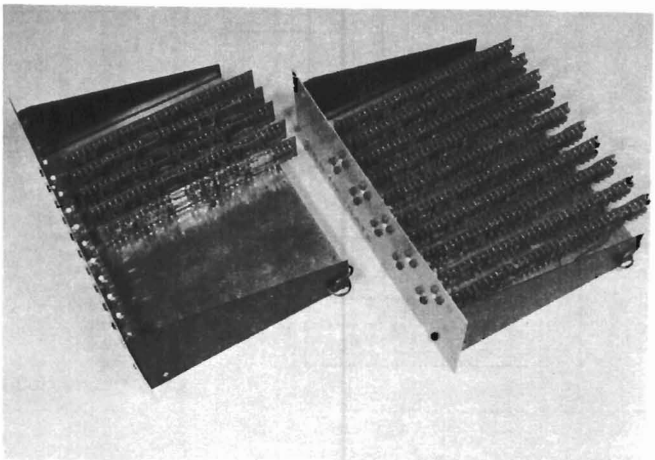
Figure 29. Signum-Gate Circuit



NEUROMIME NET SIMULATION

UNIT OF FIVE NEUROMIMES





SIGNUM AND SIGNUM-GATE
LOGIC CIRCUITS

Figure 30. Neuromime Equipment

# APPENDIX VII

# NETWORK EXPERIMENTS

## A. GENERAL

The tests described in this appendix represent the form and the results of the experimental work on the contract to date. Following is an outline of the experimental work:

(1) Tests using the transistor neuromime hardware

(a) Networks with contralateral connections

(b) Networks with internal feedback

(c) Network with paralleled neuromime output

(d) Approximation to an aircraft pitch channel

(2) Tests using the new optimal control concept

(a) Preliminary test circuits

(b) Tests on more complex systems

(c) Tests on the convergence of the neuron model's parameters to optimum values

(d) Tests on redundant networks

(e) Tests on a roll-yaw coupled aircraft

Although most of the tests were designed to parallel the analytical work, some tests were performed to aid the instrumentation development. The former tests are emphasized, because their results hold more significance for the contract work. Diagrams of the test circuits, accompanied by brief explanations of the tests, results, and evaluations, are presented in this appendix.

## B. TESTS USING THE TRANSISTOR NEUROMIME HARDWARE

### 1. Networks with Contralateral Connections

Two environment transfer functions were tested with the net - a gain and an integrator (see Figure 31). The purpose of the test was to determine the net's capability to reduce an error signal to zero. The weight computation rates observed were judged to be slower than expected. The time taken for a weight computation generally exceeded five munutes, compared to an expected value of one or possibly two minutes. However, the weights computed for the various synapses usually had the correct algebraic signs. In several cases the wrong sign was computed for a synaptic weight. The reason for the wrong synaptic weight signs was discovered to be a faulty computing method. The synaptic weight was computed continuously. The synaptic weight integrator integrated the input changes occurring during the firing cycle and subtracted the integral of the input changes occurring after firing stopped. The computed synaptic weight was correct only at the end of a firing cycle. If the synaptic weight computation was started or stopped at some time within the cycle, the computed synaptic weight had a wrong sign.

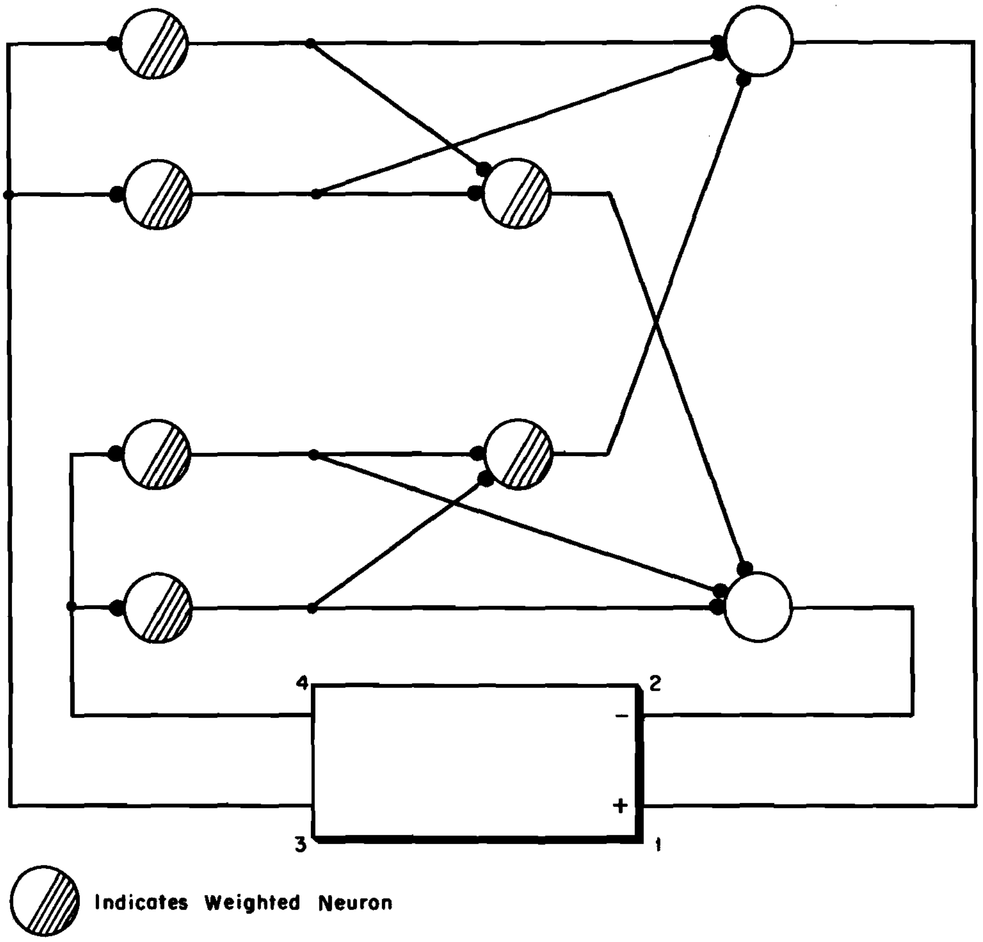


Figure 31. Network with Two Simple Environment Transfer Functions

A second order environment was used to test the network (see Figure 32). The net was tested to determine its ability to reduce the error signal to zero and to note its response to a second order environment. The environment used has the following transfer function:

$$G(S) = \frac{\omega^2}{S^2 + 2\omega \xi S + \omega^2}$$

where

$G(S)$ = environment transfer function

$\omega^2$ = environment gain

$\xi$ = damping variable

S = Laplace transform variable

The value for $\omega$ was varied from 0.3 to 10 and the value for $\xi$ varied from 0.03 to 0.8. The results of the test are plotted as a stability curve in Figure 33. The network would not compute synaptic weights with $\omega$'s over 5, which limited the test results somewhat. The network made small synaptic weight computations when $1/\omega$ was below the network response time of one second. Some difficulty was experienced when testing the system for the stability points plotted in Figure 33.

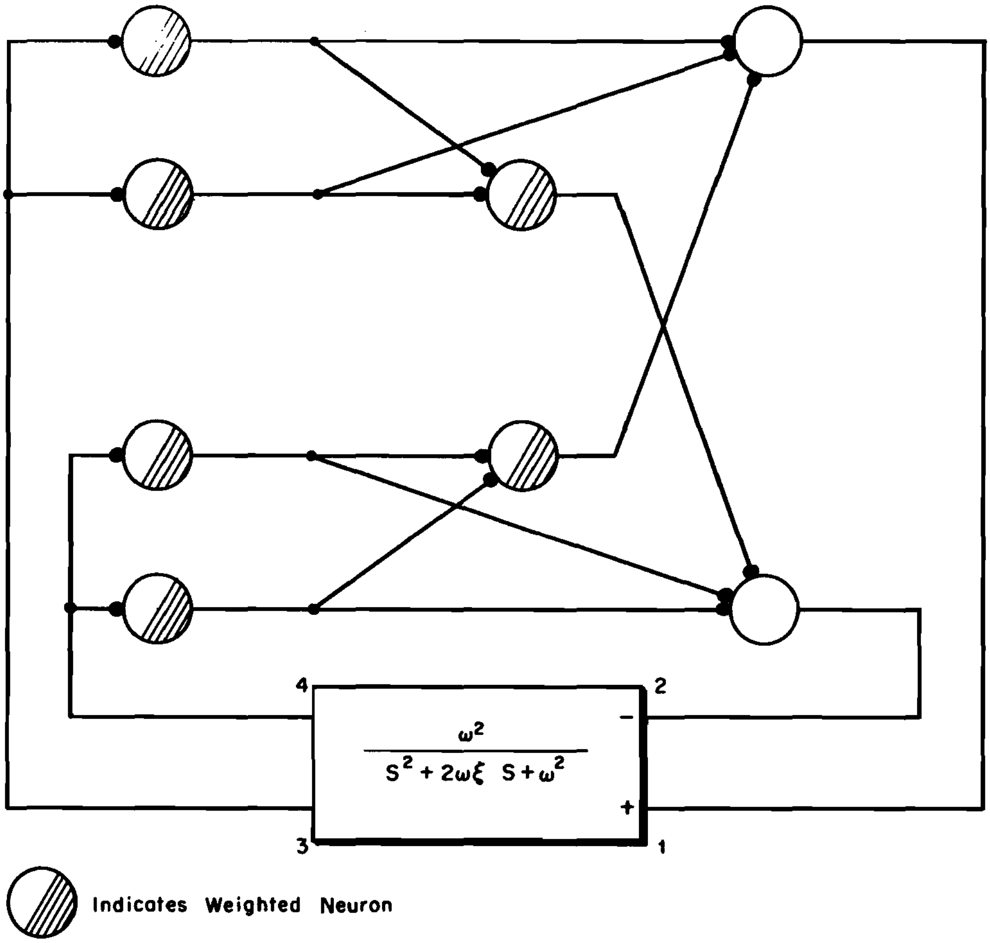
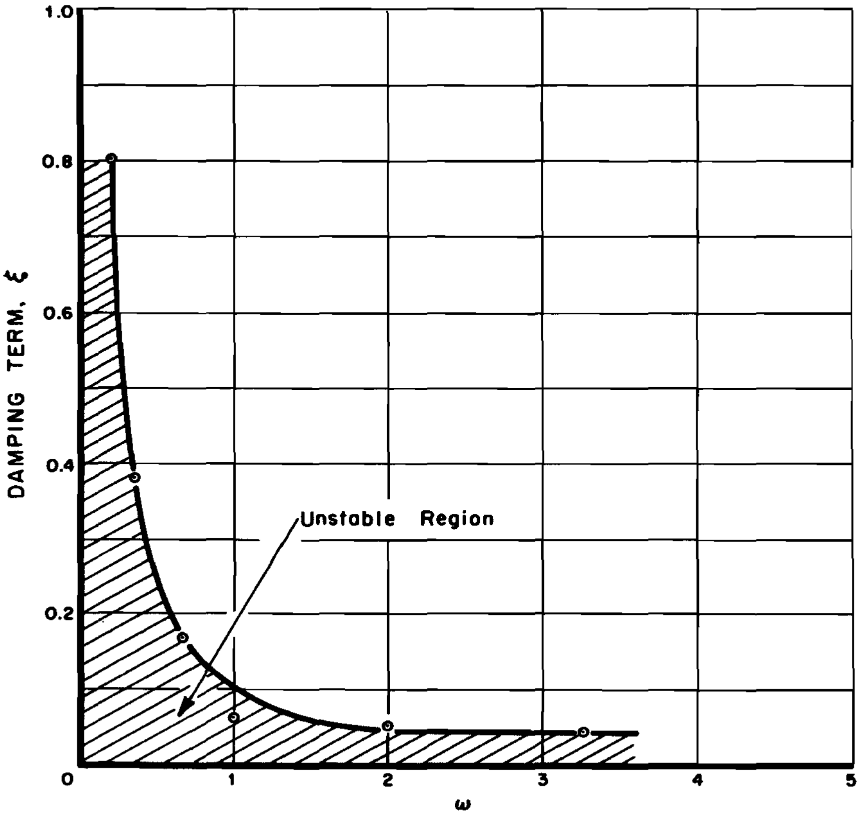

4
2
3
1
−
+
$\frac{\omega^2}{S^2 + 2\omega\xi\ S + \omega^2}$
Indicates Weighted Neuron

Figure 32. Test Network with Second Order Environment

Figure 33. Stability Curve of Second Order Environment Test



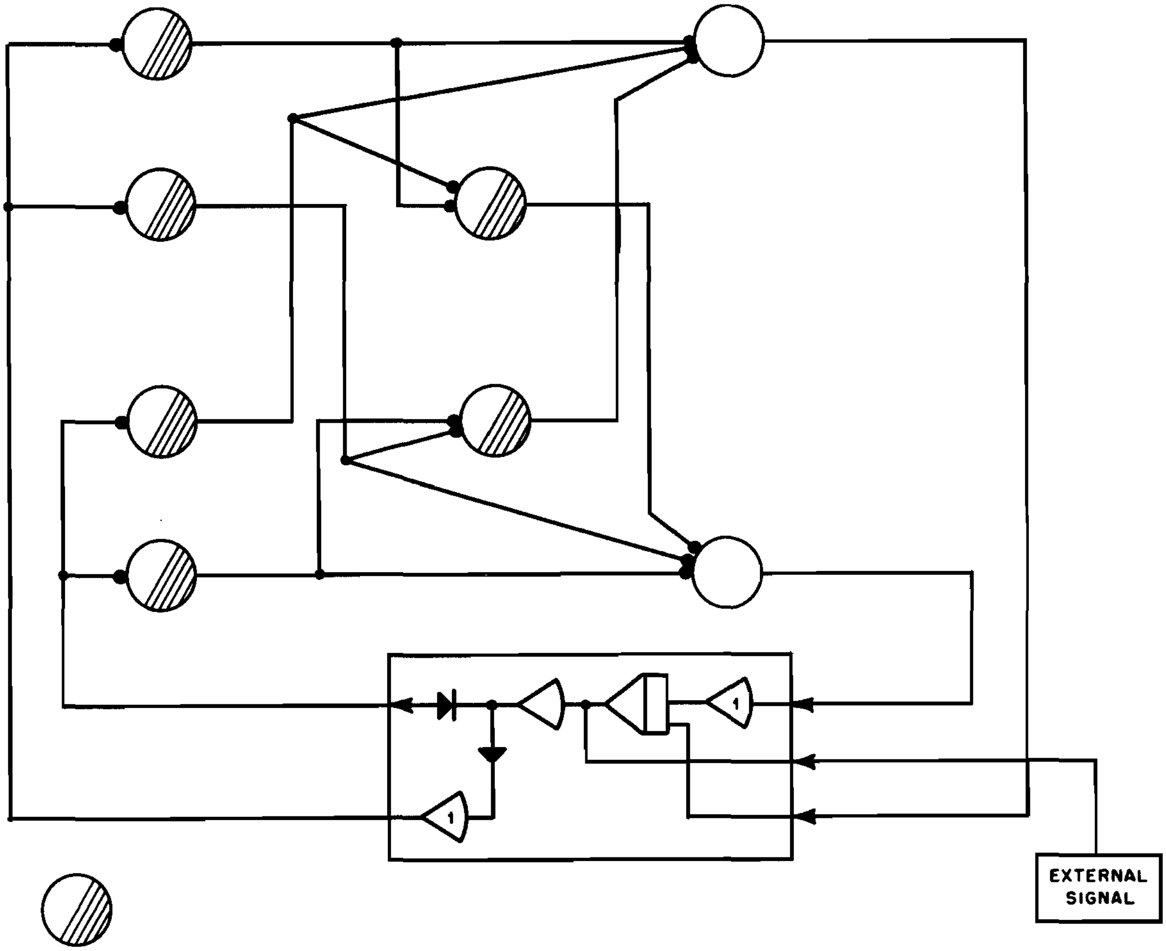


Figure 34. Test Network Connected to a Gain with a Control Reversal

For the next test, the network was connected to a gain, or "environment" transfer function (see Figure 34) and tested to determine its ability to stabilize the system after an"environment" input control reversal. The computation of weights following the input control reversal was sluggish. Preliminary tests showed that several changes should be made on the neuromime model. These changes were made and the new network was tested as outlined later.

2. Networks with Internal Feedback

Environments of a gain and an integrator were used to test four networks (see Figures 35, 36, 37, and 38). These four networks were connected to study the effects of internal feedback. The network was tested with and without internal feedback connections. Further, the network's control connections to the environment were severed to test the ability of internal feedback to assist the network to revise the synaptic weight computations when the network's output no longer affected the environment.

In the network shown in Figure 35, neuromime G2 has an environment change input and a feedback change input. The feedback change input has an opposite effect on neuromime G2 from the environment change input; i.e., neuromime G2 computes a negative synaptic weight with only feedback changes, and a positive synaptic weight with only environment changes. The ratio of the magnitudes of the environment changes to the feedback changes was defined to be A. Three values of A were used: $A < 1$; $A = 1$; and $A > 1$.

The network without internal feedback computed positive synaptic weights and stabilized the system, i.e., reduced the injected error signal to zero. Then, when the network outputs

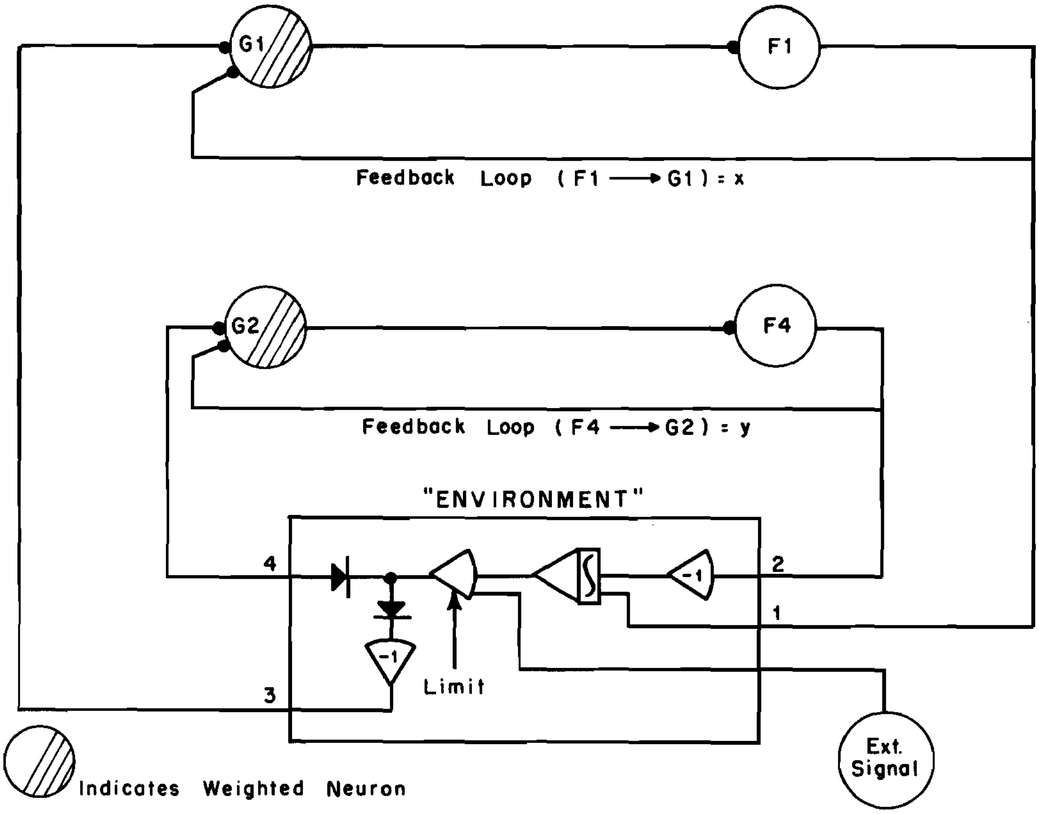

G1
F1
Feedback Loop ( F1 ⟶ G1 ) = x
G2
F4
Feedback Loop ( F4 ⟶ G2 ) = y
"ENVIRONMENT"
4
2
1
3
-1
-1
Limit
Ext. Signal
Indicates Weighted Neuron

Figure 35. Test Network with Environment Input and Direct Feedback Input

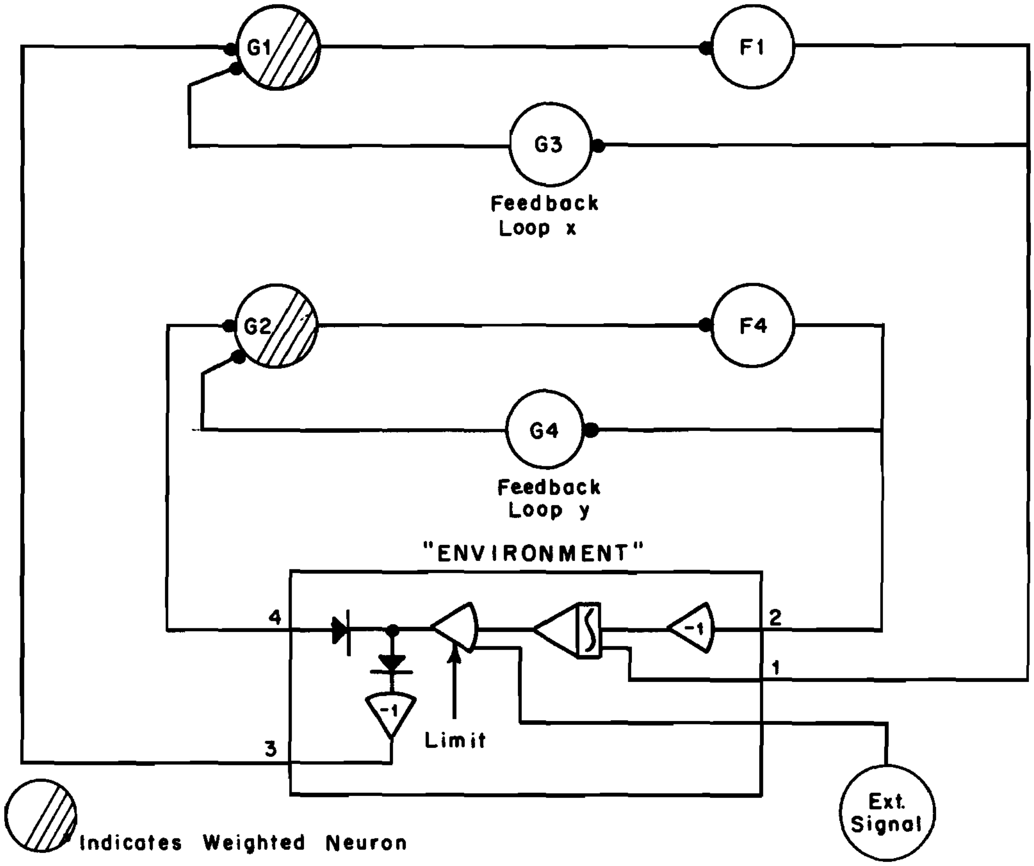


Figure 36. Test Network with Environment Input and Indirect Feedback Input

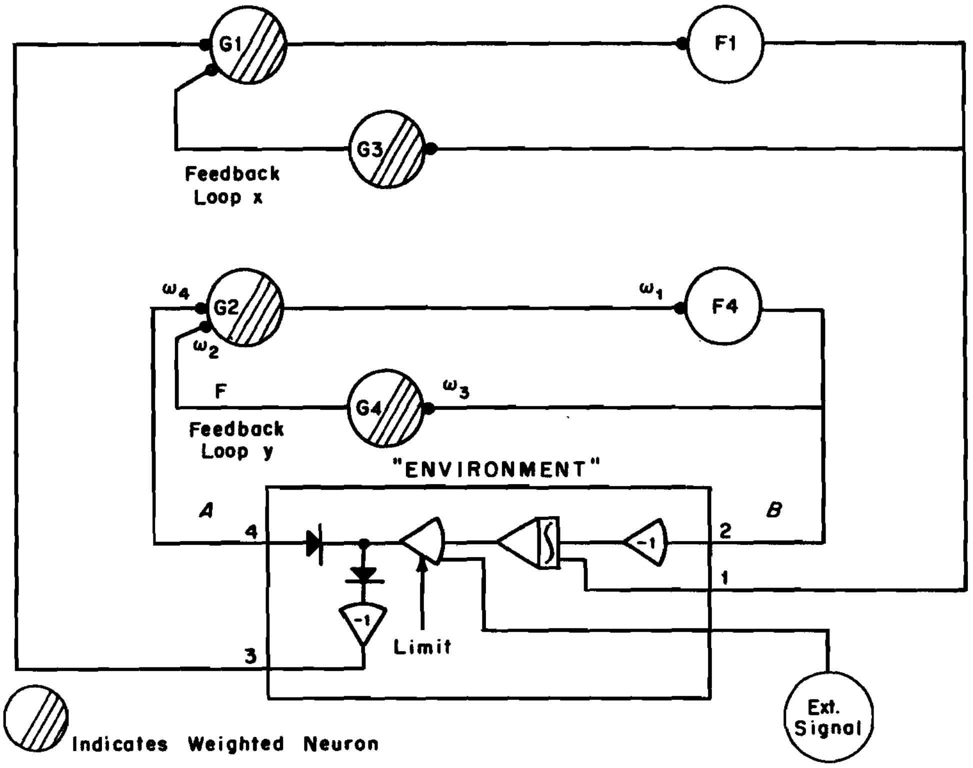


Figure 37. Test Network with Feedback Neuromime Having Computed Synaptic Weight

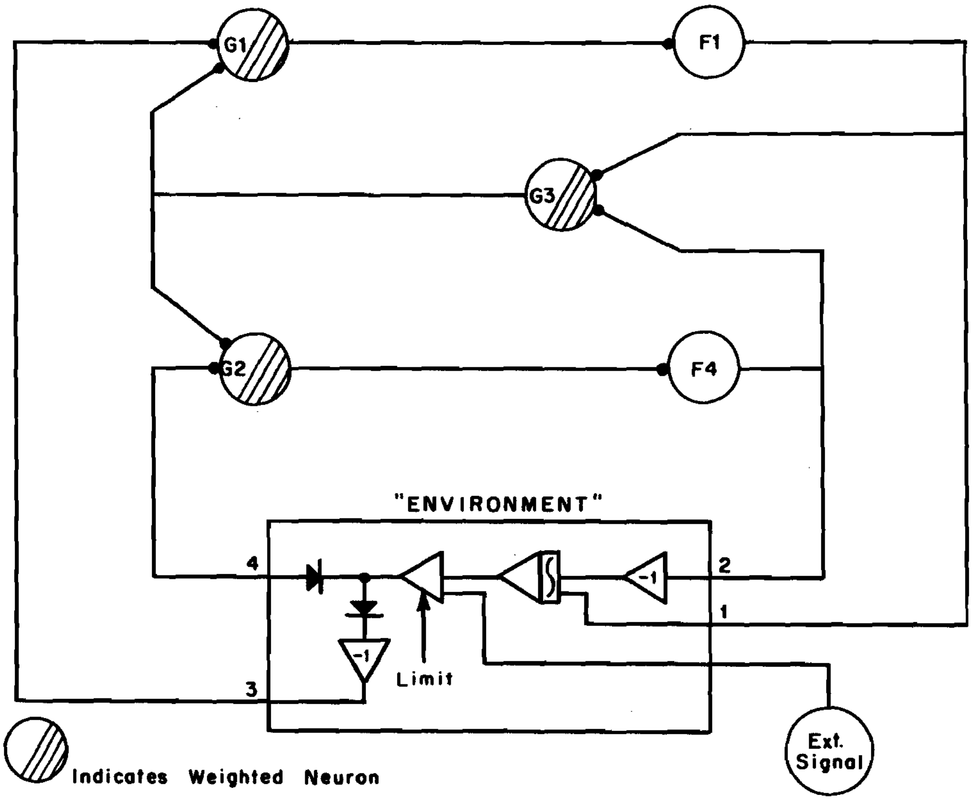


Figure 38. Test Network with Common Feedback Loop

were disconnected from the environment input, the synaptic weight values stayed constant. The network could not and did not revise its computation. The network with internal feedback, when connected to the environment, also computed positive synaptic weights and stabilized the system. When the network outputs were disconnected from the environment input, the synaptic weight computations slowly computed negative values. If neuromime G2's weighted rate and unweighted rate output were equal but opposite in sign, G2's synaptic weight computation stopped. Neuromime F4's input is the sum of the weighted and unweighted rate outputs of neuromime G2. If the weighted and unweighted rate outputs are equal but opposite in sign, their sum is zero. Neuromime F4 can never fire with a zero input. If F4 can never fire, the feedback loop to G2 is broken and all feedback changes seen by G2 stop.

Since neuromime G2 needs input changes to compute a synaptic weight, the broken feedback loop stops G2's input changes and synaptic weight computation.

With A < 1, neuromime G2 computed a negative synaptic weight. With A = 1, neuromime G2 computed a zero synaptic weight. With A >1, neuromime G2 computed a positive synaptic weight.

In the network shown in Figure 36, neuromime G4 replaced the direct feedback connection from the output of F4 to the input of G2 (as shown in Figure 35). The extra delay time of neuromime G4 was the only difference between that network and the previous one. The test results were the same as the previous one.

In the next network (Figure 37), the feedback neuromime G4 had a computed synaptic weight, in contrast to the previous network. Neuromime G4's synaptic weight became negative in all tests. The other test results were much the same as those reported for the previous network.

The networks shown in Figures 35, 36, and 37 had two separate feedback loops. In the next network, shown in Figure 38, neuromime G3 replaced the two feedback loops. The test results were the same as those reported for the previous networks. Although some interplay between the two separate network channels due to the common feedback loop was expected, none was seen.

In the next network to be tested, a gain and an integrator were used as environments (see Figure 39). The environment control connections could be reversed. The two aims for

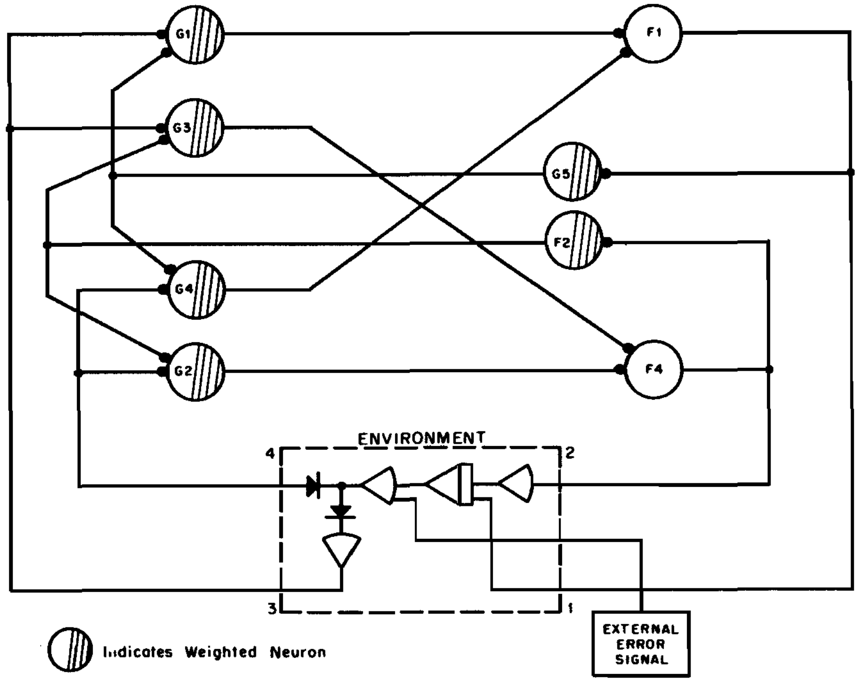


Figure 39. Test Network with Two Environment Transfer Functions with Control Reversal

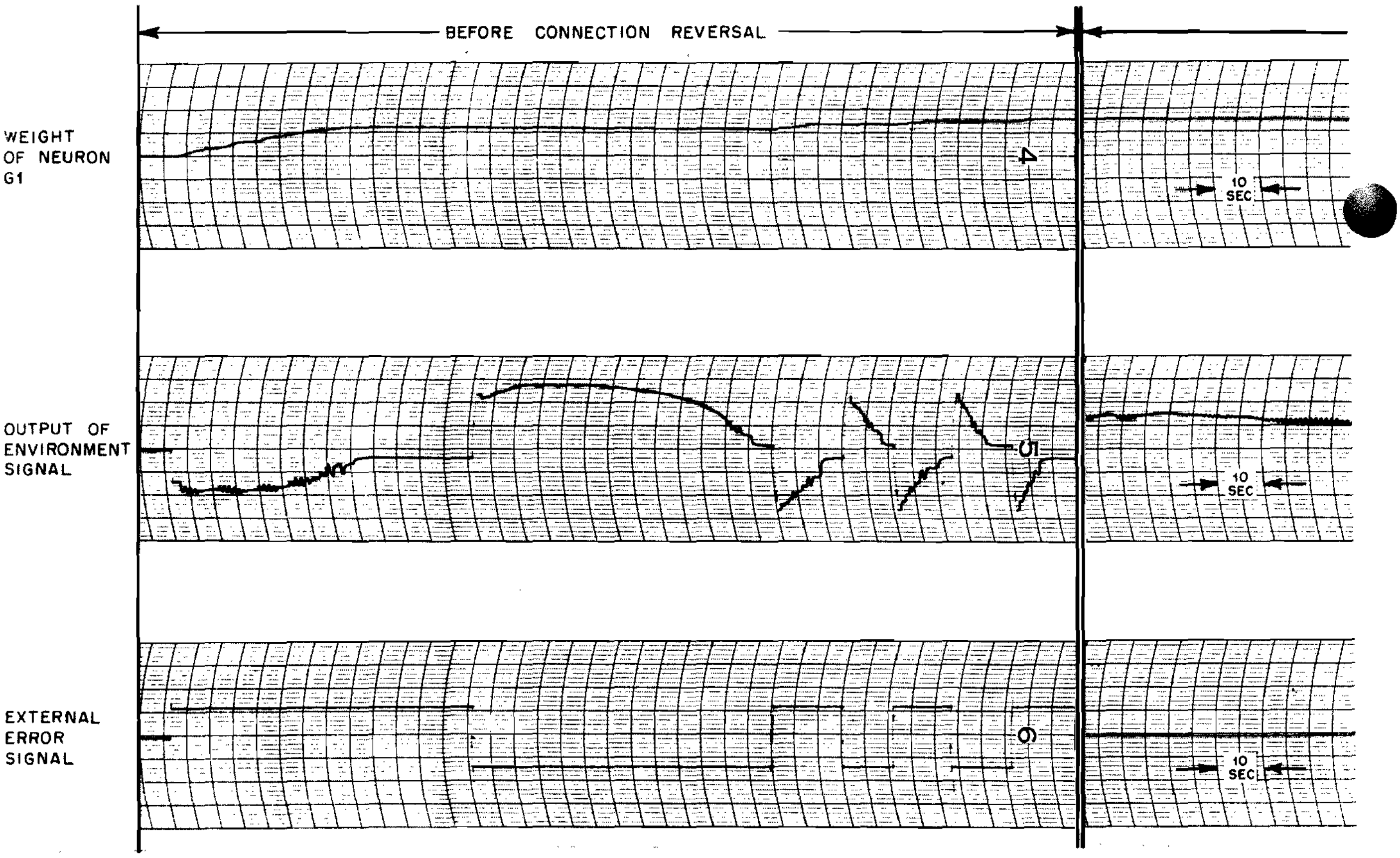


Figure 40. Analog Computer Test Run of Network without Internal Feedback (Sheet 1 of 2)

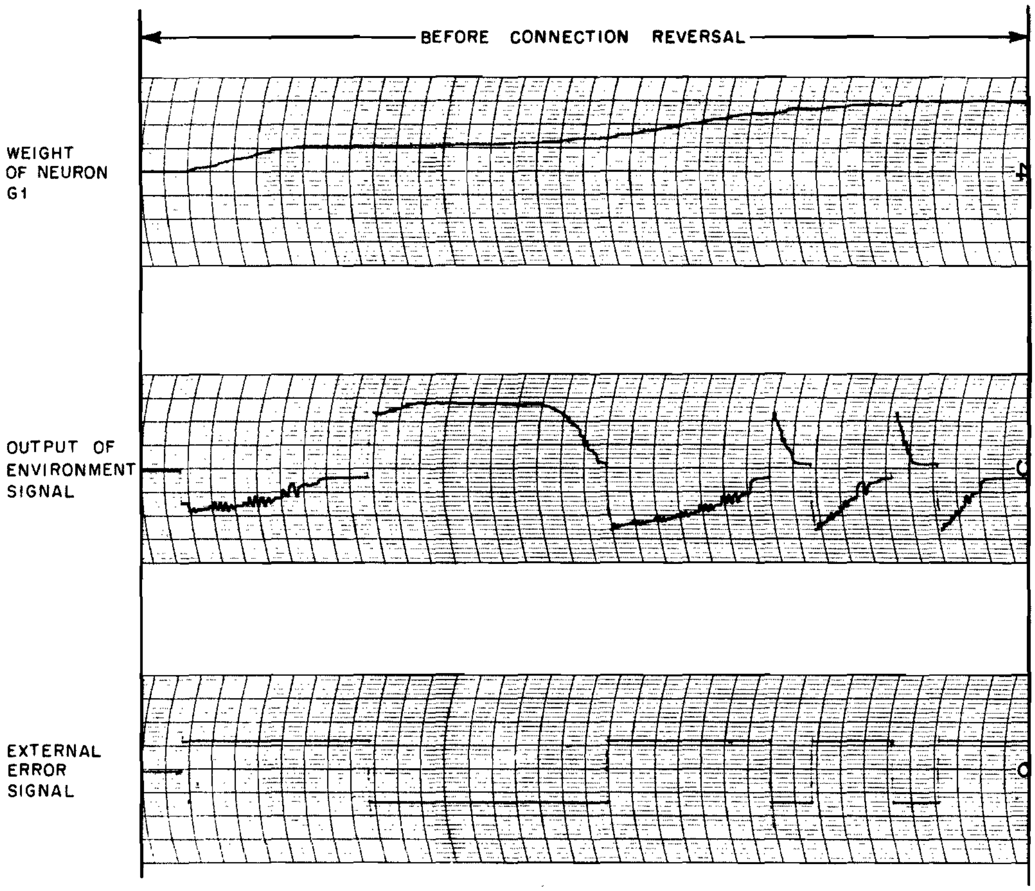


Figure 41. Analog Computer Test Run of Network with Internal Feedback (Sheet 1 of 2)

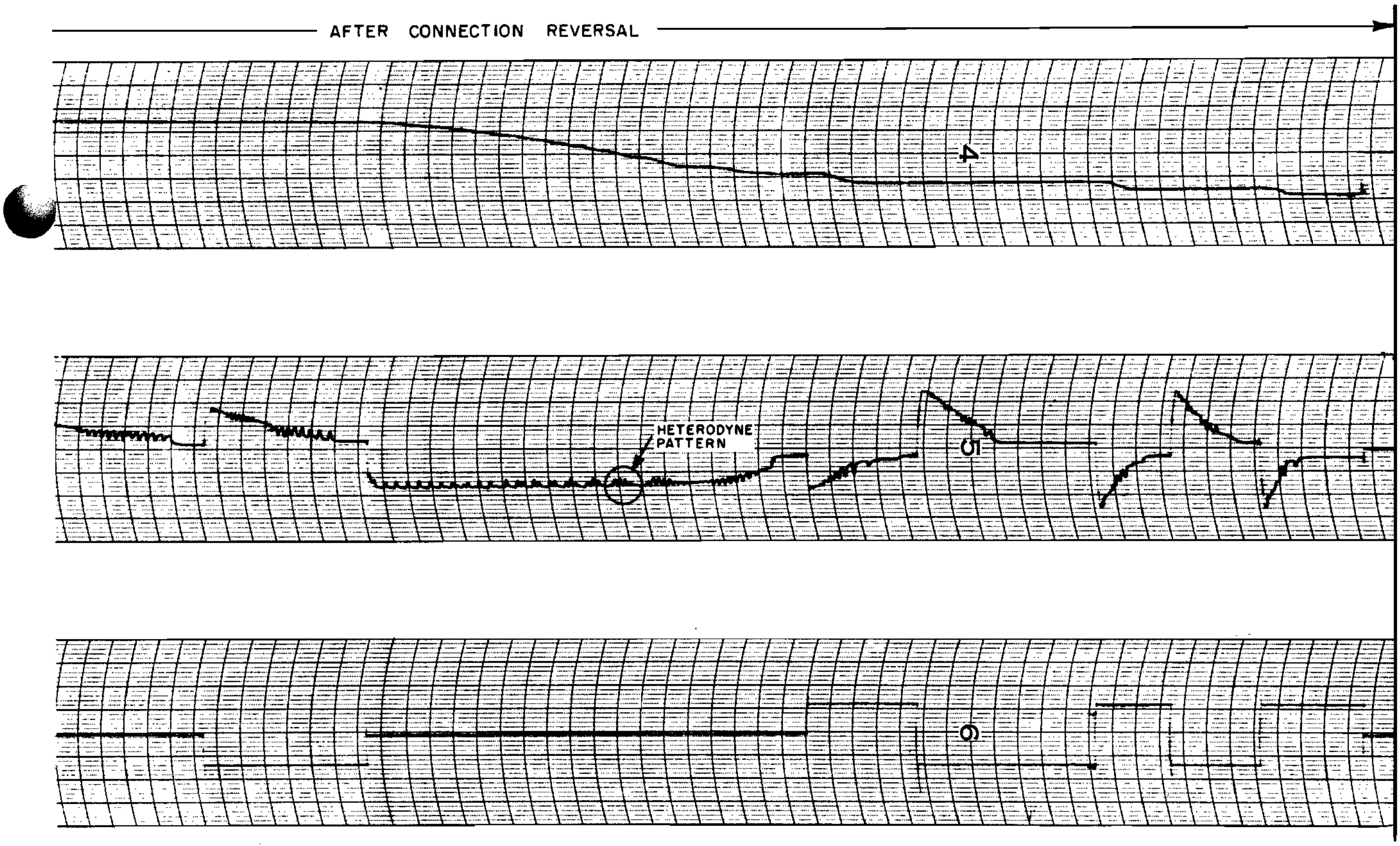


Figure 40. Analog Computer Test Run of Network without Internal Feedback (Sheet 2 of 2)

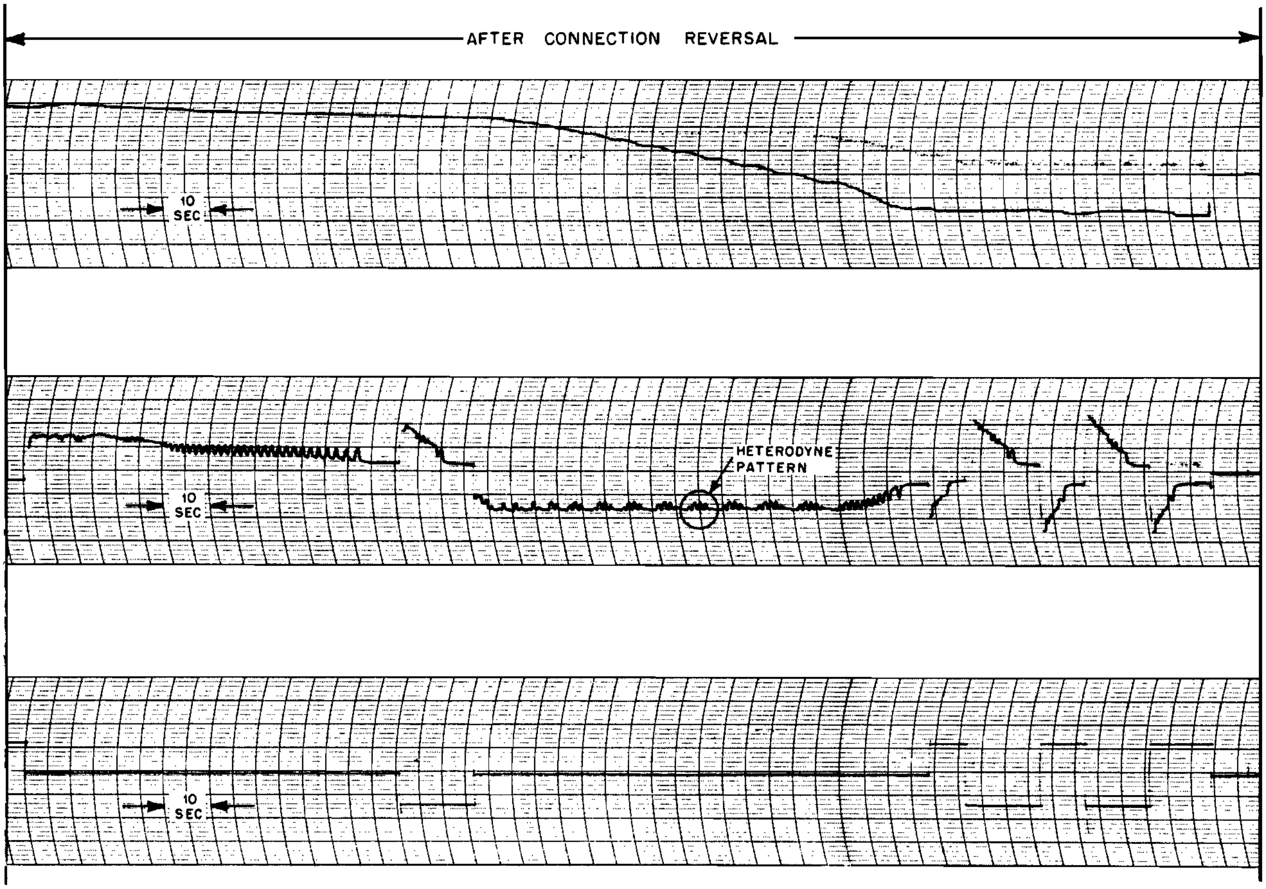


Figure 41. Analog Computer Test Run of Network with Internal Feedback (Sheet 2 of 2)

the series of tests were to determine the ability of a network to adjust to an "environment" input control reversal and to determine the difference in network action made by internal feedback. The test results shown in Figures 40 and 41 are typical analog computer test runs. Since this network used internal feedback, the testing procedure was similar to that of the previous four networks using internal feedback. The ability of the network to adjust to an environment input control reversal is shown on these analog computer test runs. The network did adjust to the control reversal. A careful study of the two test runs, one for the network with feedback and the other for the network without feedback, showed no significant difference. Notice that the neuromime revised its synaptic weight computation to adjust to the new situation.

The analog computer runs show a heterodyne pattern in the environment output error signal caused by the network input to the environment. The interference of one network output with the other produces this heterodyne pattern.

3. Networks with Paralleled Neuromime Output

A gain and an integrator were the environments used in the next network tests (see Figure 42). The purpose of the tests was to study the effects that many neuromimes firing at once had on the environment. Two to four neuromimes were connected to the same stimuli

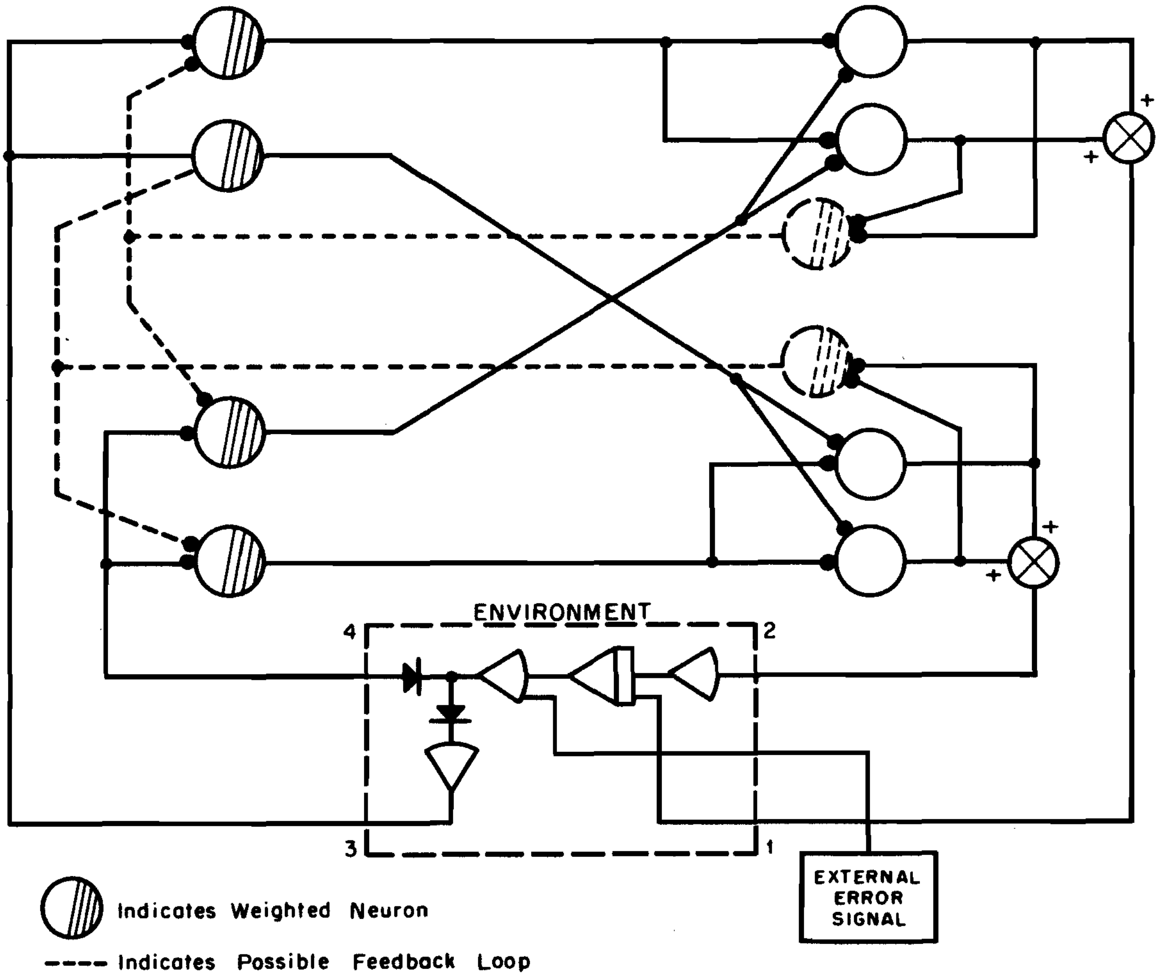

+
+
+
+
ENVIRONMENT
4
2
3
1
EXTERNAL ERROR SIGNAL
Indicates Weighted Neuron
---- Indicates Possible Feedback Loop

Figure 42. Test Network with an Output of Two Paralleled Neuromimes

and their outputs added together. This sum was the network output and produced a heterodyne pattern like that observed in the previous network.

The network reduced an error signal to zero, but took a longer time to do this task than a similar network with one output neuromime. The synaptic weights were computed correctly, but slowly. When three and four neuromimes were paralleled, the network took longer to perform the control task than the network with two neuromimes paralleled. Of course, the synaptic weight computations for the networks with three and four neuromimes paralleled were also slower than those of the network with only two neuromimes paralleled. The tests showed that the approximation to many neuromimes firing by paralleling two to four neuromimes was too crude.

An attempt was made to test the network with a second-order environment. The environment was designed to allow a control reversal. The purpose of the test was to study the network interconnections necessary to control a second-order environment after a control reversal. However, preliminary tests on the system showed a lack of knowledge of the network interconnections that would be required to control a second-order environment without a control reversal. The network was revised to study the control of a second- and third-order environment.

4. Approximation to an Aircraft Pitch Channel

A third-order environment was used to simulate, rather crudely, an aircraft pitch channel (see Figure 43). The parameters used in the simulation were not designed for any

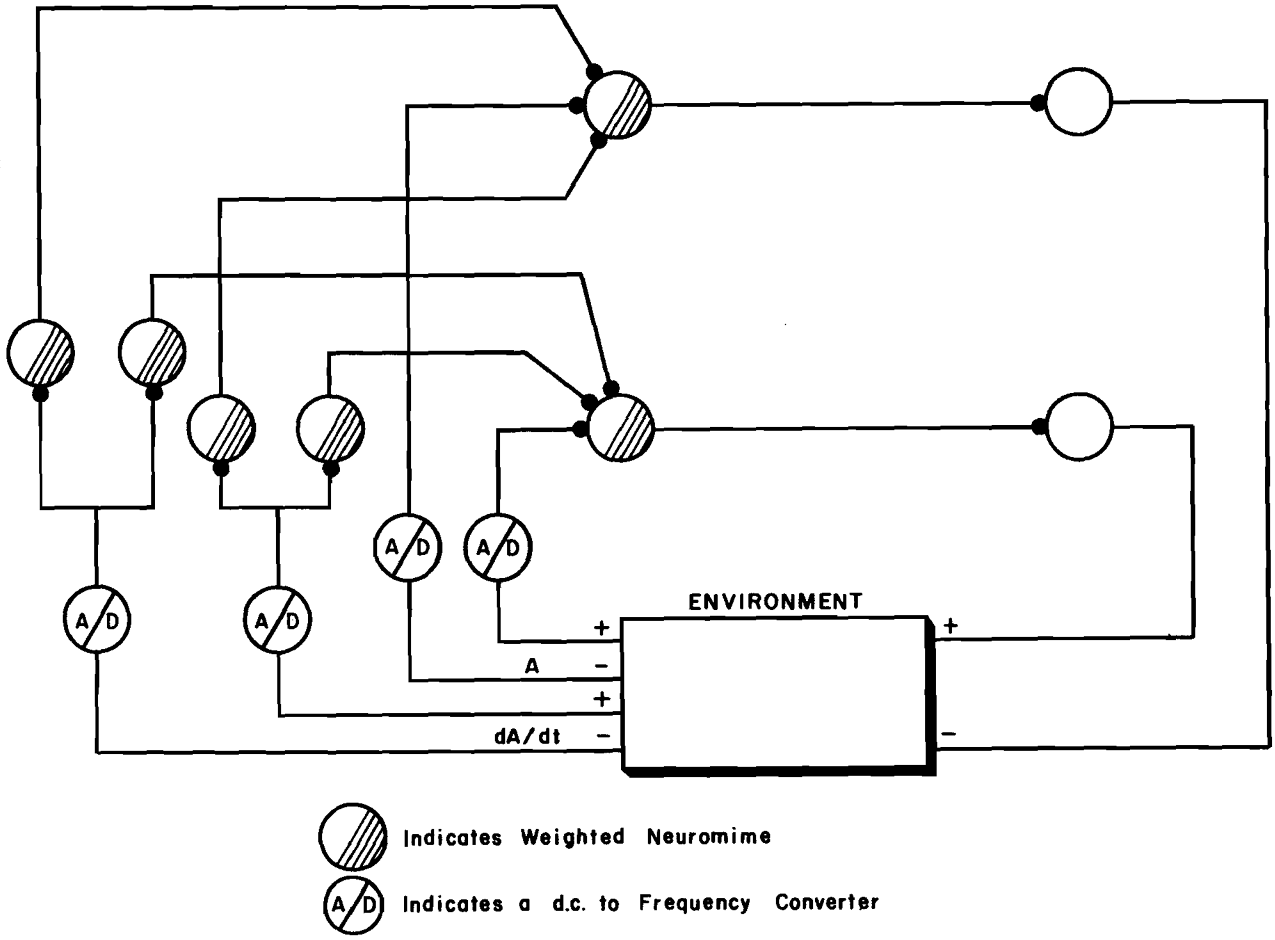

A/D
A/D
A/D
A/D
ENVIRONMENT
+
A
−
+
dA/dt
−
+
−
Indicates Weighted Neuromime
A/D
Indicates a d.c. to Frequency Converter

Figure 43. Test Network with Rate Feedback

specific aircraft. The basic study was that of controlling a third-order system. The premise for the tests was that a neuromime network that controls a third-order system could control a third-order pitch channel with specific aircraft pitch parameters. The network was tested both with and without pitch-rate feedback. The results of the pitch-rate feedback tests were evaluated to aid the instrumentation development.

The analog computer recordings (Figures 44 and 45) show that the network reduced pitch error to zero regardless of the presence of rate feedback. Further, all synaptic weights had the proper algebraic sign. However, the synaptic weight computations did not perform as expected. Some synaptic weights were small and had little effect on the control. The overall system response was improved when pitch-rate information was supplied to the neuromime network.

The network without pitch-rate feedback did not reduce the transient "wiggling" response of the pitch-error signal to a step input. Further, this network did not reduce the overshoot which occurs as the pitch-error signal swings through zero. The network with pitch-rate feedback improved the system response in both these conditions.

## C. TESTS USING THE NEW OPTIMAL CONTROL CONCEPT

Networks were designed to study the application of optimal control theory to neuron model networks. The networks tested for this phase of the neuron model development used no external hardware. All simulations were done on the analog computer. The equations instrumented for the tests are discussed more fully in the text of this report. In this appendix, the networks tested are described and significant results are summarized to illustrate the development of the neuron model through the application of optimal control theory. In general, the equations used to describe the tests in this section are as follows:

$$\dot{Y} = AY + BM + D \tag{47}$$

$$M = -B^tP \tag{48}$$

$$-\dot{P} = A^tP + Y \tag{49}$$

where Y is the environment output matrix, M is the neuromime network output matrix, D is the environment disturbance matrix, and A and B are environment coefficient matrices. The matrix P entered into the equations from Pontryagin's maximum principle.

The optimal criterion was to minimize a system "cost" function, which closely resembles a minimum energy criterion. The equation is

$$C = \frac{1}{2} \int_0^\infty (Y^tY + M^tM)\, dt.$$

### 1. Preliminary Test Circuits

The preliminary tests presented here were designed with two immediate goals in mind: first, to develop instrumentation methods for use with the optimal control theory; and second, to illustrate certain parts of the theory.

Figure 46 shows the first circuit using the new optimal control concept. To control the system optimally, initial conditions on the integrators within the neuromime network were set at optimum values. These initial conditions for the parameters of the test network were computed by hand to yield an optimum path. These parameter values were then set on the analog computer and tests were run on the network. The network converged to zero following the computed optimum path. If one of the network parameters was varied away from the computed optimum value, the system diverged.

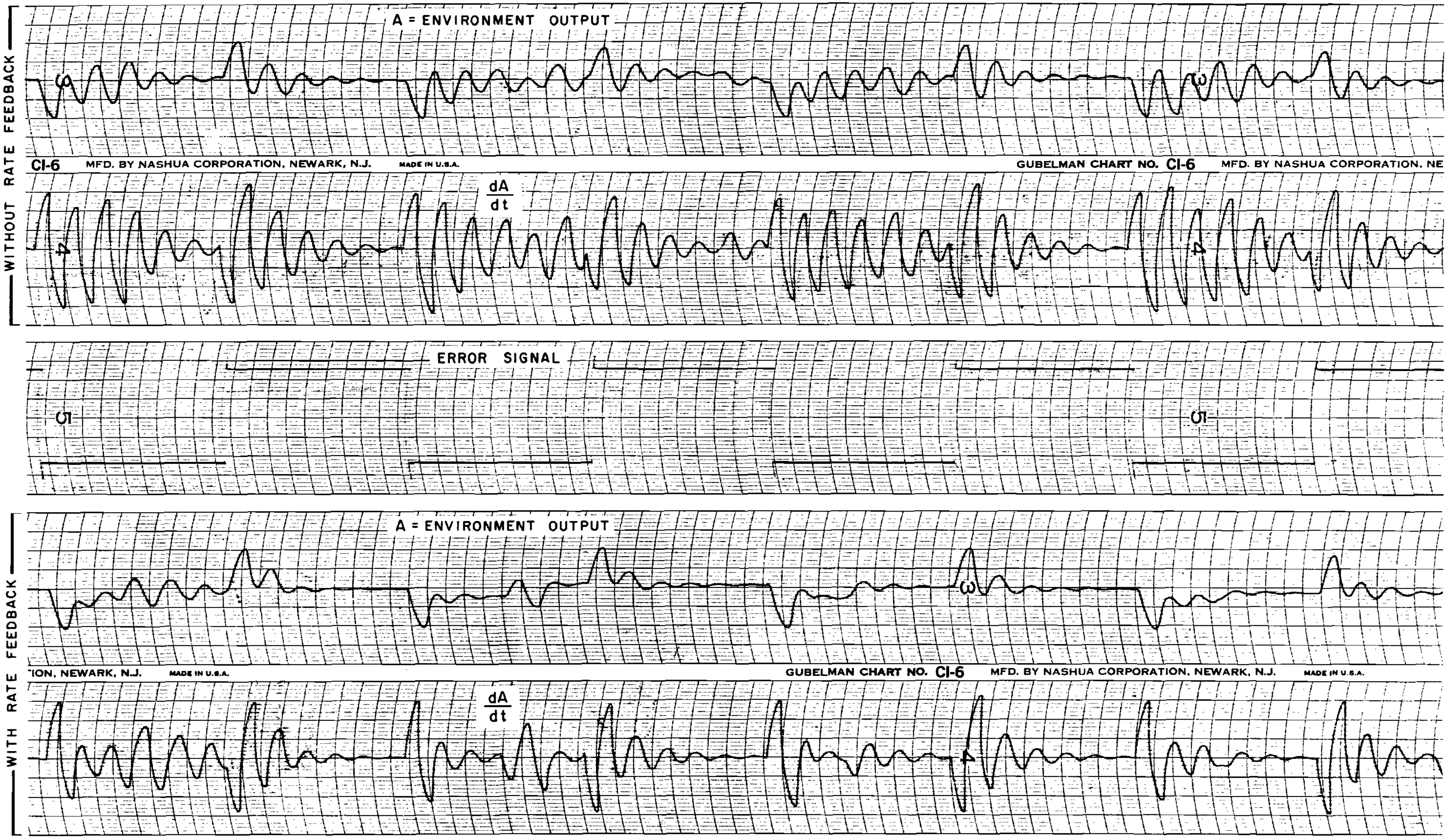


**Figure 44. Analog Computer Test Run of Network Using Rate Feedback**

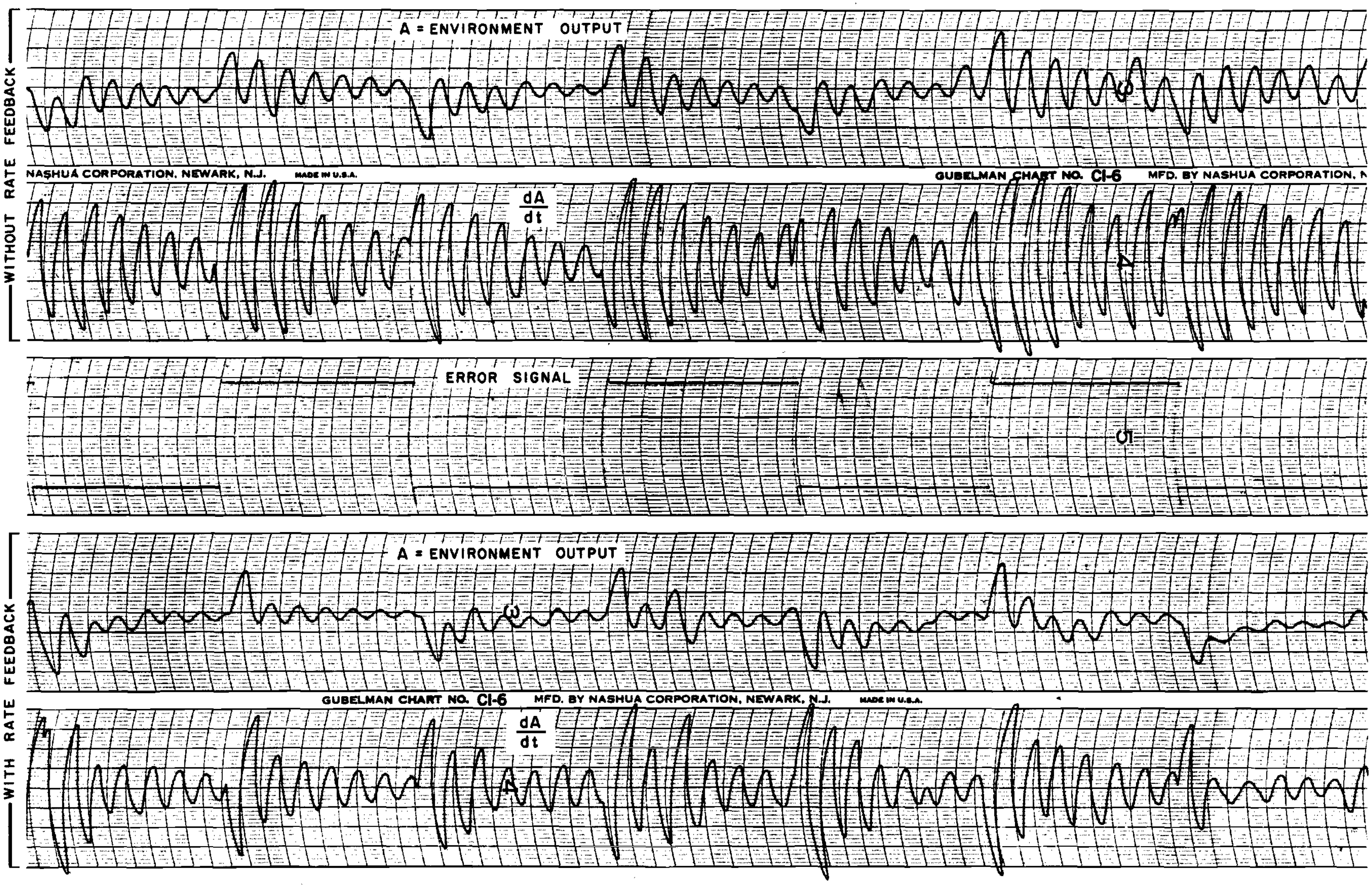


Figure 45. Analog Computer Run of Network Using Rate Feedback - Rerun at Smaller Damping Constant

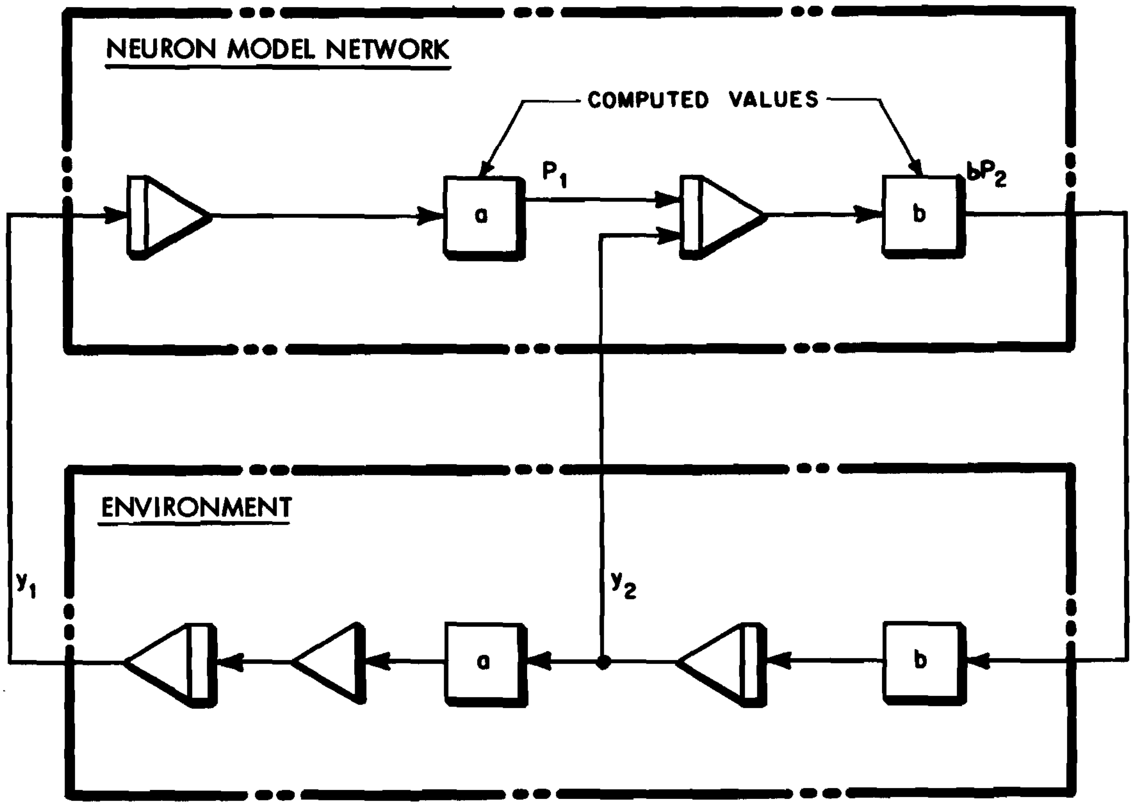


Figure 46. First Test Network for New Concept

The neuron model network used for this test was then extended, and an adjustment rule postulated so that the neuron model computed the optimum parameters automatically. In this adjustment rule a neuron model considered the reaction of its inputs to its output and compared the actual input reaction to the predicted input reaction. Any deviation of the predicted reaction from the actual reaction was integrated, and the deviation integral was used to adjust the predicted reaction. When adjusting the predicted reaction, the neuron model subsequently adjusted its synaptic weight. The result of the adjustments was a predicted input reaction that matched the actual input reaction. At the same time, the optimum synaptic weight was computed. The major problem with the simulation was the difficulty encountered when the size of the system was increased to three or more neurons. The instrumentation problems arising from this attempt were too restrictive.

To further the instrumentation development, six networks were set up and tested to determine the effect of feedback on the computation of synaptic weights. Figure 47 shows block diagrams of five of the networks tested (the sixth will be shown in Figure 49). The networks used either one or two neuron models and two environments, an integrator and an exponential decay. To test the networks, the optimum system response and the corresponding synaptic weights were computed. The networks were then tested and the synaptic weights and system responses recorded. These recorded values were compared with the ideal computed values.

Figure 48 shows a set of typical responses. The responses shown in Figure 48A and 48B were obtained from the test networks shown in Figure 47A and 47B, respectively. Note that in Figure 48A, the predicted value of the input is updated until it matches the actual value. The updating is a result of the synaptic weight adjustment. The neuron models at first computed

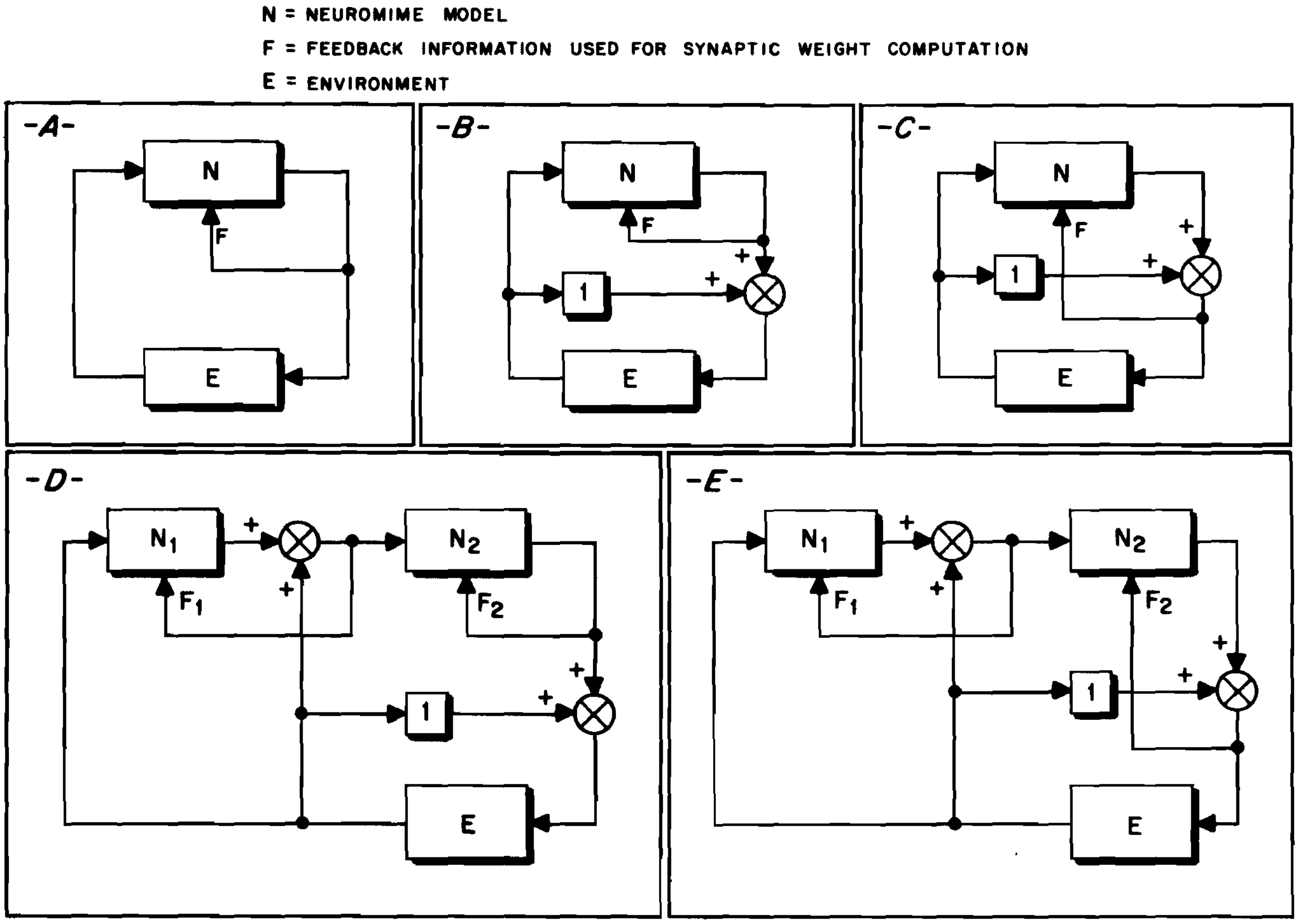


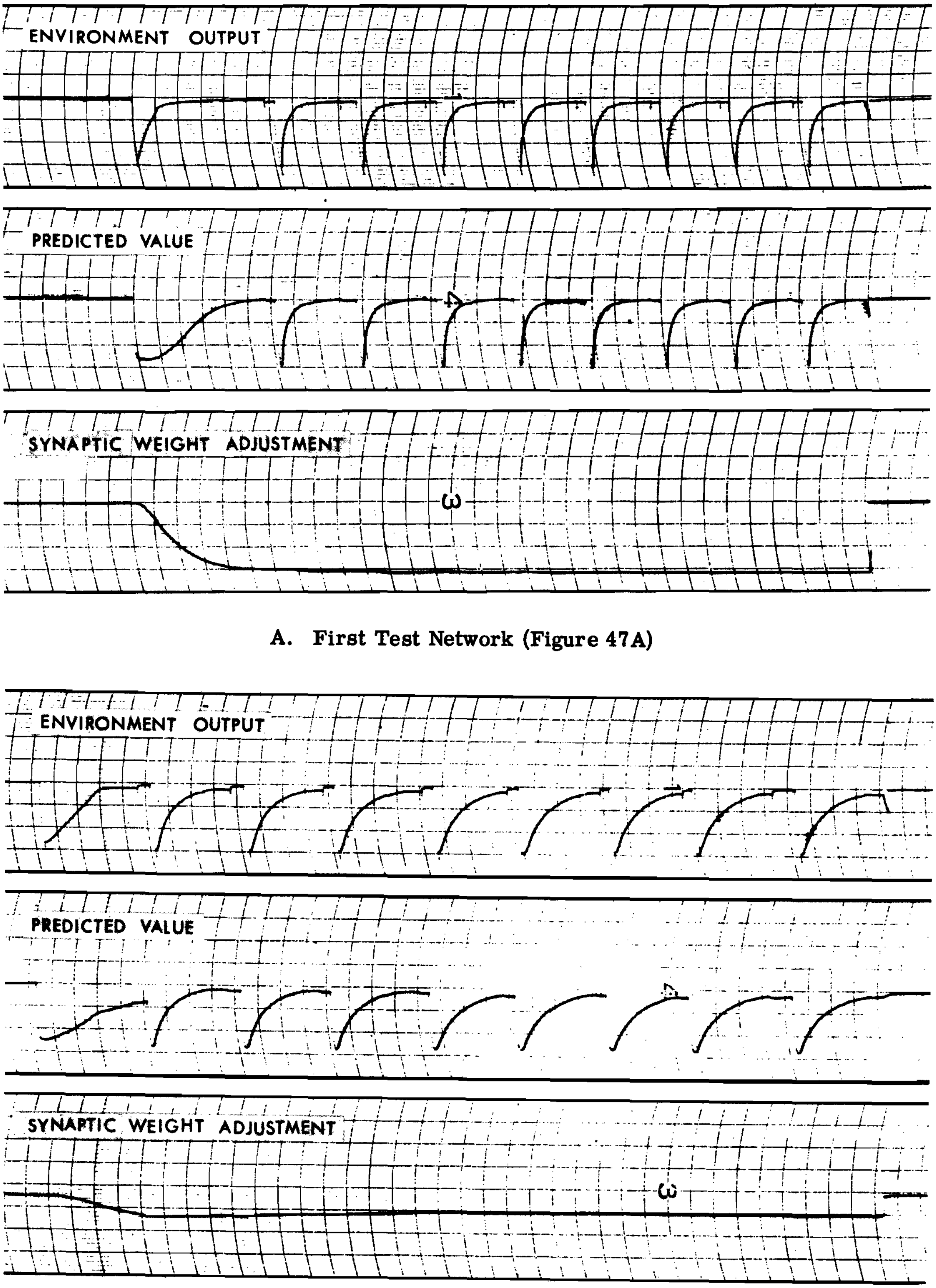
Figure 47. Five Preliminary Test Networks

synaptic weights that overshot the optimum synaptic weight, and then approached the optimum value asymptotically. The environment used for this test was an integrator. One control input, that of the neuron model, was used. In this case, the information fed back to the neuron model was the neuron model's own output.

When another control input was added, as in the system shown in Figure 47B, the neuron model computed a non-optimum synaptic weight. The extra control input was viewed as another neuron model or, possibly, many other neuron models. In this case also, the feedback information was the neuron model's own output. Figure 48B shows a typical response to disturbances injected periodically into the system. Notice that the predicted path and the actual path of the input trajectory are different. This difference is caused by the non-optimum synaptic weight computation. The synaptic weight computed was such that the integral of the error between the predicted path and the actual path of the input trajectory was zero.

One general conclusion resulting from analysis of this test series is that the neuron models seem to require feedback information about the activity beyond the synaptic junction, rather than activity at the junction.

A block diagram of the last test in the preliminary test series is shown in Figure 49. This test examined one particular adjustment rule for computing synaptic weights. The environment used for the tests was an integrator. The equations derived for the system in Figure



A. First Test Network (Figure 47A)

B. Second Test Network (Figure 47B)

Figure 48. Typical Response of Preliminary Test Networks

49, using Pontryagin's maximum principle, were

$$\dot{y} = bx \qquad (50)$$

$$-P = y \qquad (51)$$

$$m = -\beta P \qquad (52)$$

where y was the environment output, m the network output, $\beta$ and P parameters controlled by the neuron model, and b the fixed environment parameter. The adjustment rule used for this system was

$$y = K\dot{y} \qquad (53)$$

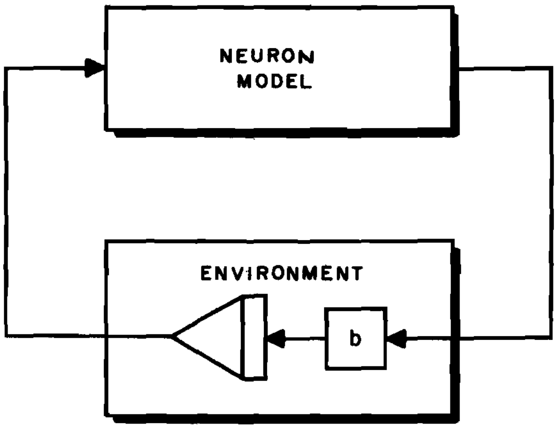


Figure 49. Last Preliminary Network Tested

where K is a parameter controlled by the neuron model. By comparing y with its time derivative $\dot{y}$, the neuron model computed the value of K. Proper manipulation of Equations 50, 51, 52, and 53 yielded the equation

$$P = -Ky, \qquad (54)$$

which gave the value for P when the value for K is given. Combining Equations 50 and 52 yielded the equation

$$\dot{y} = -\beta^2 P, \qquad (55)$$

which the neuron model used to compute $\beta$ given the value for P. Notice that Equations 53, 54, and 55 are interdependent. The three equations reacted in a "bootstrapping" fashion to compute simultaneously all three parameters, K, P, and $\beta$. The analog computer circuit of the neuron model used for this test is shown in Appendix VI, Section B. There were two parts to the test. The first part showed that the hand-calculated optimum parameters were optimum. The second part showed that the neuron models actually computed these optimum parameters.

For the first part of the test, the system equations and the criterion used by the neuron model were used to calculate optimum values of K, P, and $\beta$ for various values of the environment's parameter, b. The calculated optimum values for K, P, and $\beta$ were tested to be certain they were indeed optimum. The criterion used to determine optimality was that used in the development of the optimum system (Equations 50, 51, and 52). This criterion is

$$C = \int_0^T (y^2 + m^2)\, dt$$

where C (cost) is to be minimized, and y and m are the neuron model's input and output, respectively. To test the calculated optimum values, the values were set up on the analog computer and the values of C were measured. The values of C for corresponding values of K, P, and $\beta$ around the optimum values were also measured. The tests were conclusive, proving that the calculated optimum values were indeed optimum. A typical result is plotted on the graph in Figure 50.

For the second part of the test, the system was tested with five values of the environment parameter b. The environment parameter b had a range of 0.5 to 2.0. A step voltage was used as an output from the environment. The neuron model computed the optimum values for K, P, and $\beta$ as verified by the previous test on optimum values. A set of ideal values for the parameters K and b was plotted on the graph in Figure 51 and compared to the set of values computed by the neuron model.

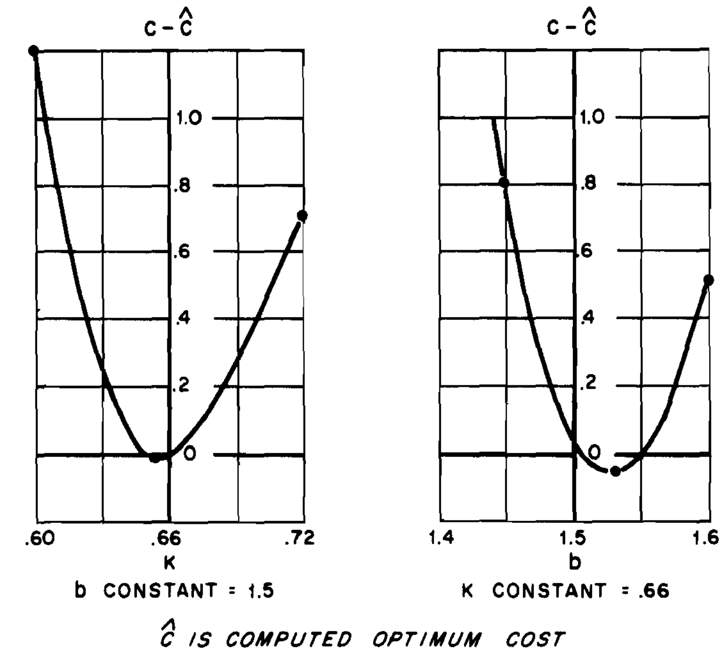


Figure 50. Test Results Showing Cost

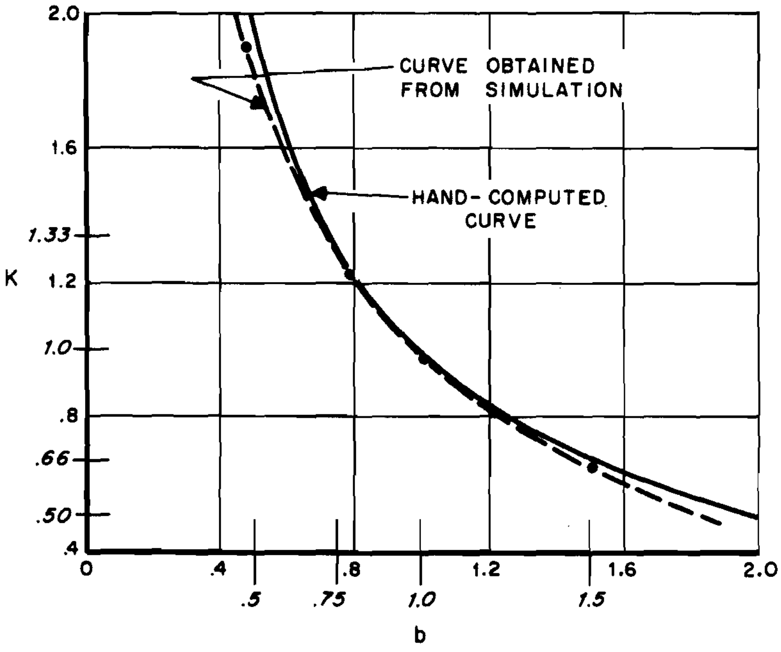


Figure 51. Comparison of Ideal K versus b Curve with Computer K versus $\beta$ Curve

## 2. Tests on More Complex Systems

Two tests were performed in this test series. In both tests the environment was second order.

a. First System Tests. The environment for the first test system, shown in Figure 52, was quite simple, having only two parameters. An additional equation was used for the simulation. The equation was

$$P = KY \tag{56}$$

The specific parameter and state variable matrices for this test were as follows:

$$Y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & a \\ 0 & 0 \end{bmatrix}, \tag{57}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & b \end{bmatrix}, \quad P = \begin{bmatrix} P_1 \\ P_2 \end{bmatrix}, \tag{58}$$

$$M = \begin{bmatrix} 0 \\ m \end{bmatrix}, \quad K = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}. \tag{59}$$

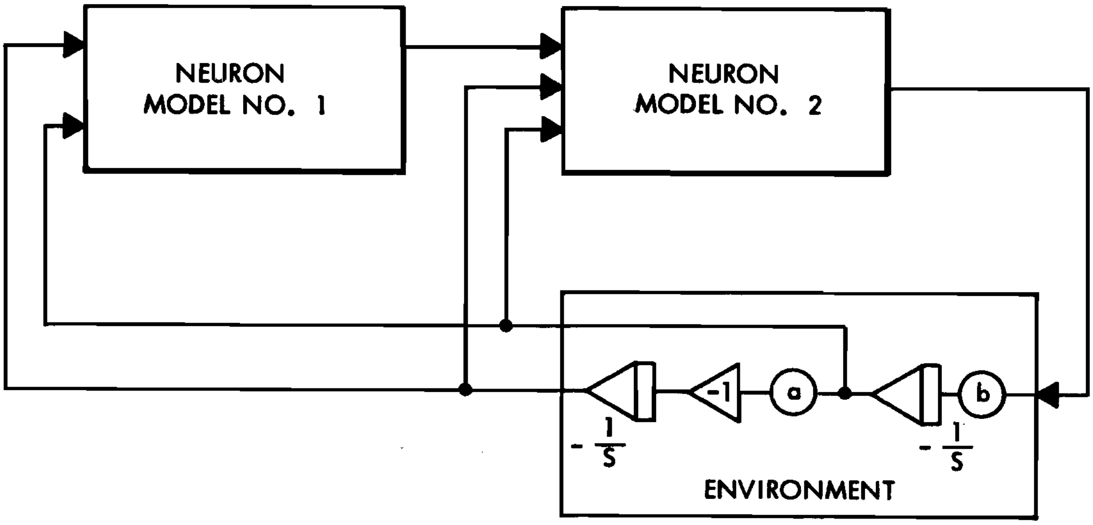


Figure 52. First Complex Test System

Further, the matrix K had the following form in terms of the parameters a and b:

$$K = \begin{bmatrix} \dfrac{\pm\sqrt{1 + \dfrac{2a}{b}}}{a} & \dfrac{1}{b} \\ \dfrac{1}{b} & \dfrac{\pm\sqrt{1 + \dfrac{2a}{b}}}{b} \end{bmatrix} , \qquad (60)$$

where Y is the environment output matrix, M is the neuromime network output matrix, and A and B are environment coefficient matrices. The matrix P entered into the equations from Pontryagin's maximum principle. The matrix K enters into the equations from an assumption described in detail in the main text of this report. The criterion used to evaluate the tests was the same as that described at the beginning of this appendix.

There were three parts to the test. The first part was performed to show that the optimal controller did yield a minimum value for the system cost function. Various parameter value sets were tested by holding all parameters in a set constant except one. This one parameter was varied and the cost function measured. Each parameter value in a set was tested in like manner. During these tests, neuron models made no parameter computations.

The second part of the test was performed to show that the neuron models did compute the optimum parameters. The neuron models tested used a hill-climbing technique to compute their parameters. Various parameter value sets were programmed into the environment and initial conditions set for the neuron model parameters. Then environment disturbances were injected and the neuron models were allowed to compute.

The objective of the third part of the test was related to the second. Tests were performed to observe the reaction of the neuron models to two simultaneous disturbances of the same (and opposite) polarity. The two simultaneous disturbances, which formed the complete disturbance vector, took the form

$$D = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} .$$

Note that in the second part of the test, either $d_1$ or $d_2$ of the disturbance vector was zero.

In general, all of the test objectives were achieved. Figure 53 is a graph of normalized cost versus $\alpha$. The parameter $\alpha$ corresponds to the entry "a" in the environment coefficient matrix A in Equation 40, and was computed by the neuron model. The zero point on the cost axis is the theoretical point at which the cost is a minimum. The graph shows a 2 percent deviation from the theoretical point. Figure 54 is a comparison of various theoretical values of $\alpha$ and the actual values computed by the neuron models. The straight line is the locus of all points where the theoretical and actual values of $\alpha$ were equal. The graph shows minor deviations of the points tested. No point has more than a 5 percent deviation.

The curves in Figures 55 and 56 are similar to Figures 53 and 54, respectively, but represent typical results of the neuron model computations of the K matrix parameters in Equations 56, 59, and 60. The graphs show accuracies similar to those obtained with the $\alpha$ parameter computations.

Figure 57 is an analog computer recording. It shows the system reaction to a disturbance in the environment. Starting at the top of the chart, the first two channels make up the

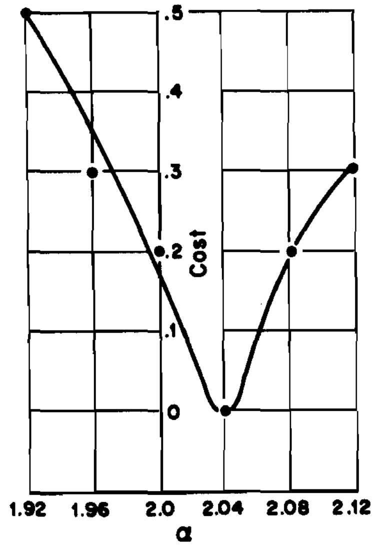


Figure 53. Normalized Cost versus *α*

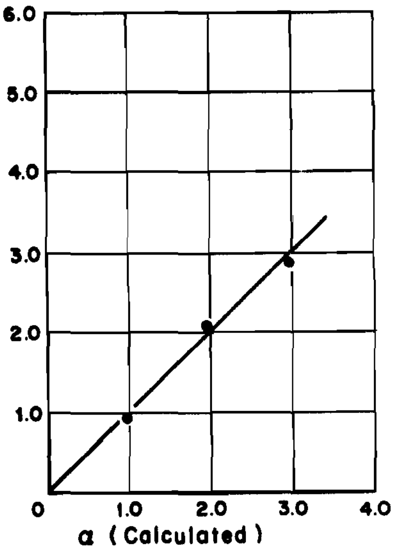


Figure 54. Theoretical versus Actual Values of *α*

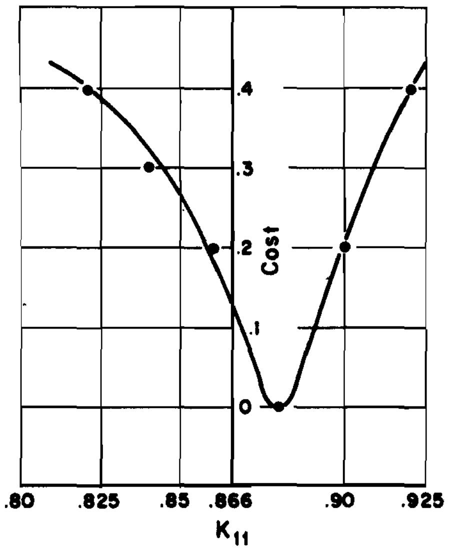


Figure 55. Normalized Cost versus K

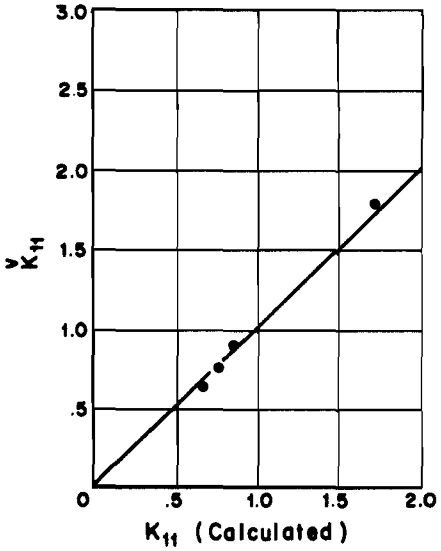
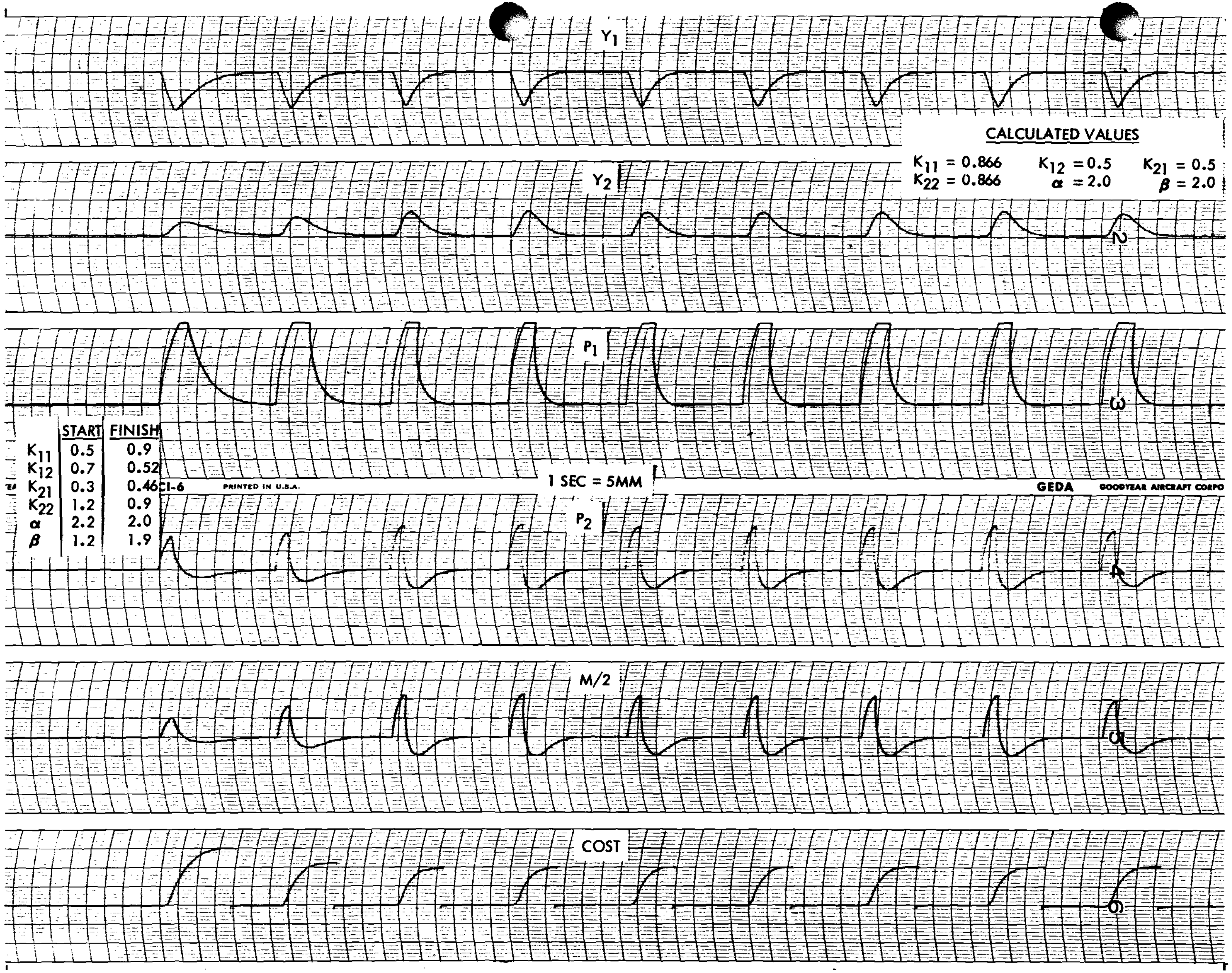


Figure 56. Theoretical versus Actual Values of K



| | START | FINISH |
|---|---|---|
| $K_{11}$ | 0.5 | 0.9 |
| $K_{12}$ | 0.7 | 0.52 |
| $K_{21}$ | 0.3 | 0.46 |
| $K_{22}$ | 1.2 | 0.9 |
| $\alpha$ | 2.2 | 2.0 |
| $\beta$ | 1.2 | 1.9 |

CALCULATED VALUES

$K_{11} = 0.866$ $K_{12} = 0.5$ $K_{21} = 0.5$
$K_{22} = 0.866$ $\alpha = 2.0$ $\beta = 2.0$

Figure 57. Analog Computer Recording of System Response (First Test Series)

environment output matrix, Y, as in Equation 57. The next two channels, labeled $P_1$ and $P_2$, make up the matrix P, as in Equation 58. The last two channels are the neuron model output, M, and the cost function. The system was given ample time to settle down after a disturbance before another disturbance appeared. Note how the value of the cost function decreased as the system adjusted to the optimum parameters. After the third disturbance, the cost function remained fairly constant, indicating completion of the optimum parameter computation.

When one of the neuron model's parameters was held fixed, the neuron model computed parameter values that were not optimum for the total system. However, the parameter values computed still caused the cost function to be reduced. This indicated that the neuron models did the best they could under other than optimum conditions. When part or whole of a neuron model was "damaged," the remaining neuron models readjusted their parameters to allow for the "damage." The result of the readjustment did not yield the optimum system response, but the response was near optimum. Of course, any damage that might disconnect the neuromime network from the environment would stop system control.

In the third part of the test, the correlation between the two disturbances seemed to confuse the neuron model computations. Improvements were made on some of the analog circuits to lessen the confusion. Although the parameters computed by the neuron models were not optimum, the value of the cost function was decreased.

In all tests the system was stable. An exception to this statement occurred when the neuromime network's control output had a reversed sign. In this case the system diverged.

b. Second System Test. The environment used for the second test was more complex than that used for the first test. The second-order environment was changed to admit all entries in the environment coefficient matrix, A, as follows:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} . \tag{61}$$

A comparison of Equation 61 with Equation 57 illustrates the increased complexity.

The test on this system was composed of three parts. The first part showed whether or not the system parameters, computed by hand, minimized total system cost. Each parameter affecting the system cost was varied separately, while other system parameters were held constant at the optimum values computed by hand. The result of the test series was a set of cost versus parameter variation curves. The second part showed whether or not the neuron model computed the optimum parameter values. The test was made by simply allowing the neuron models to react to environment disturbances. The result of the tests was a set of analog computer recordings showing a time history of the total system cost as affected by the neuron model parameter adjustments. The third part demonstrated the ability of the neuron model to adapt to external damage. The external damage was an environment parameter change.

Figure 58 shows the results of the first part of the test. A glance at the shape and location of the curves is enough to see that the objective was accomplished. The method used to compute the optimum parameters by hand was shown to be correct. Further, the method can easily be set up for digital computers.

Figure 59 shows the results of the second part. Note the change in the system response as the parameter values approach optimum. The response went from a damped oscillation to an exponential decay, which was the expected result. Note also the reduction in total system cost as the neuron models adjusted their parameters.

Once the optimum parameters were reached, the total system cost tended to oscillate slightly about a minimum value. This slight oscillation was caused by small variations in th

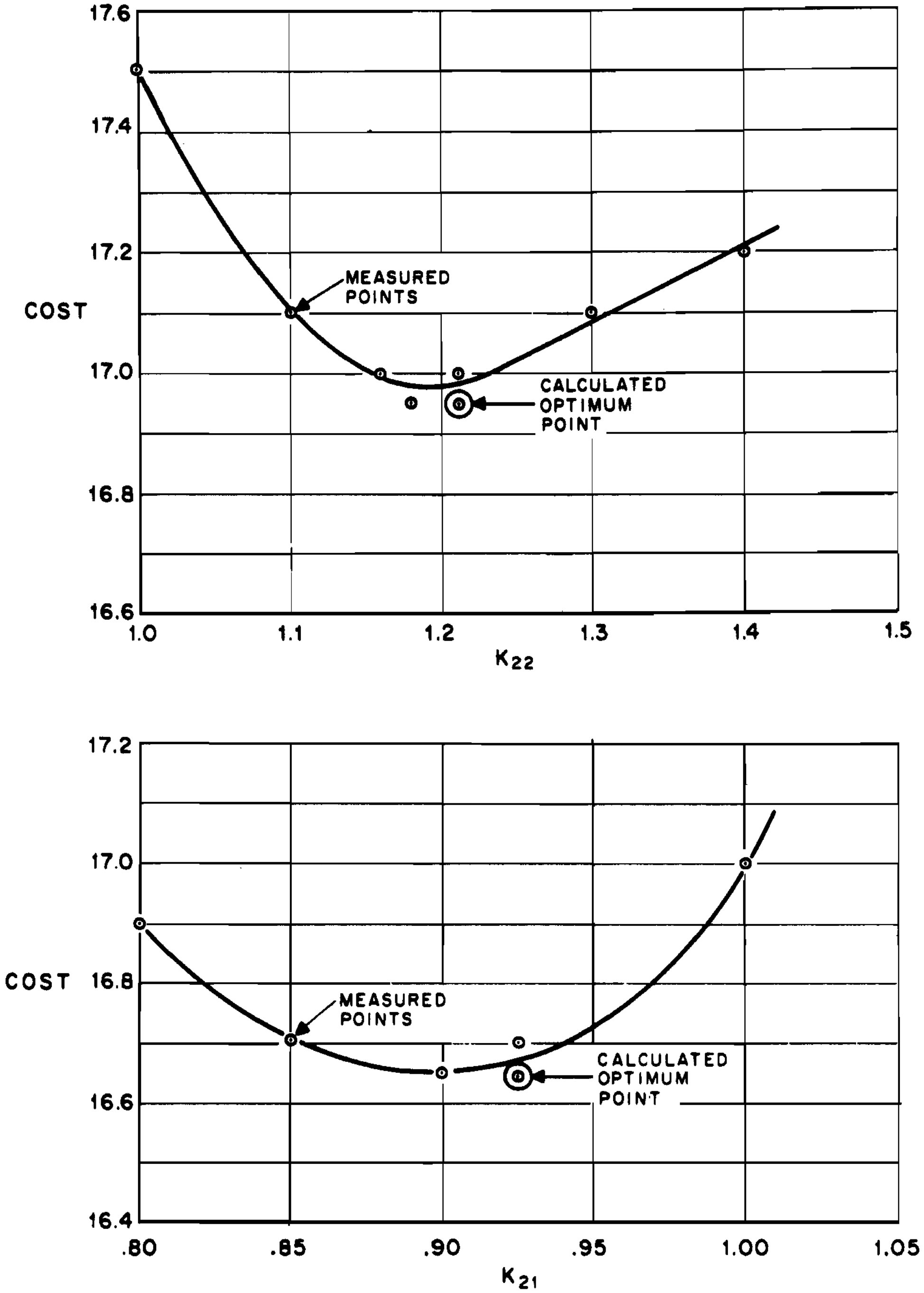


Figure 58. Cost versus Parameter Variation (Second Test Series)

parameters about their optimum values. The total system cost curve was flat around the minimum cost; thus cost variations about the minimum cost were expected to be small.

The results of the third part of the test were as expected. The neuron models adjusted their parameters within their adjustment range to accompany changes in the environment's parameters. Actually, adjusting to a new set of parameters after a parameter change was no different to the neuron models than initially adjusting to the original parameters. The worst case for an environment parameter change was a change to some value outside the neuron model's adjustment range. The neuron models in this case did not compute the optimum

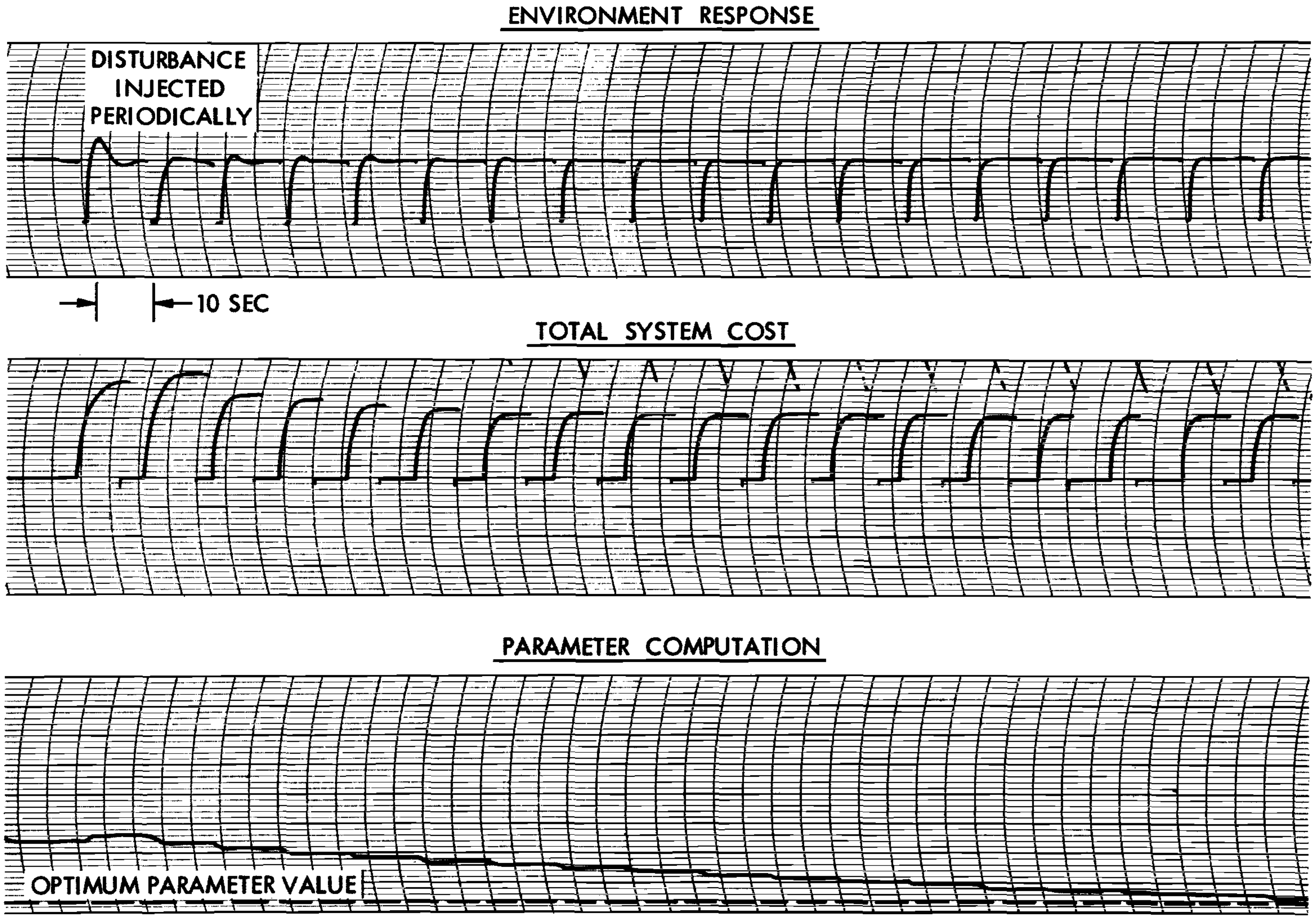


Figure 59. Optimum Parameter Computation (Second Test Series)

parameters. Rather, the maximum (or minimum) values for the neuron model's parameters were computed. Even though in this case total system cost was not minimum, it was reduced.

3. Tests on the Convergence of the Neuron Model Parameters to Optimum Values

Tests were designed and set up to examine the convergence of the optimum control parameters of a simple, first-order system. Basically, the tests performed were quite simple. A first order system was set up on the analog computer. At the beginning of the tests the controller was not the optimal controller. The task of the controller (a neuron model network) was to adjust its parameters and become the optimal controller of the plant.

The analytical basis for the convergence tests comes from the system equations listed at the beginning of this appendix. A formal derivation is given in Appendix IX. One obtains that

$$(\alpha - A) - (\beta - B)\,\beta^t K = \phi.$$

If A and B are given, a complete zero-error trajectory in the $\alpha$, $\beta$ plane is defined for each value of K. It was postulated that perturbations in the value of K allowed the $\alpha$ and $\beta$ parameters to converge to the optimal parameters. In the tests two values of K were switched back and forth to demonstrate the theory. The two values were chosen far enough apart to exaggerate the convergence trajectory. Figure 60 shoes the zero-error trajectories in the $\alpha$, $\beta$ plane for two values of K.

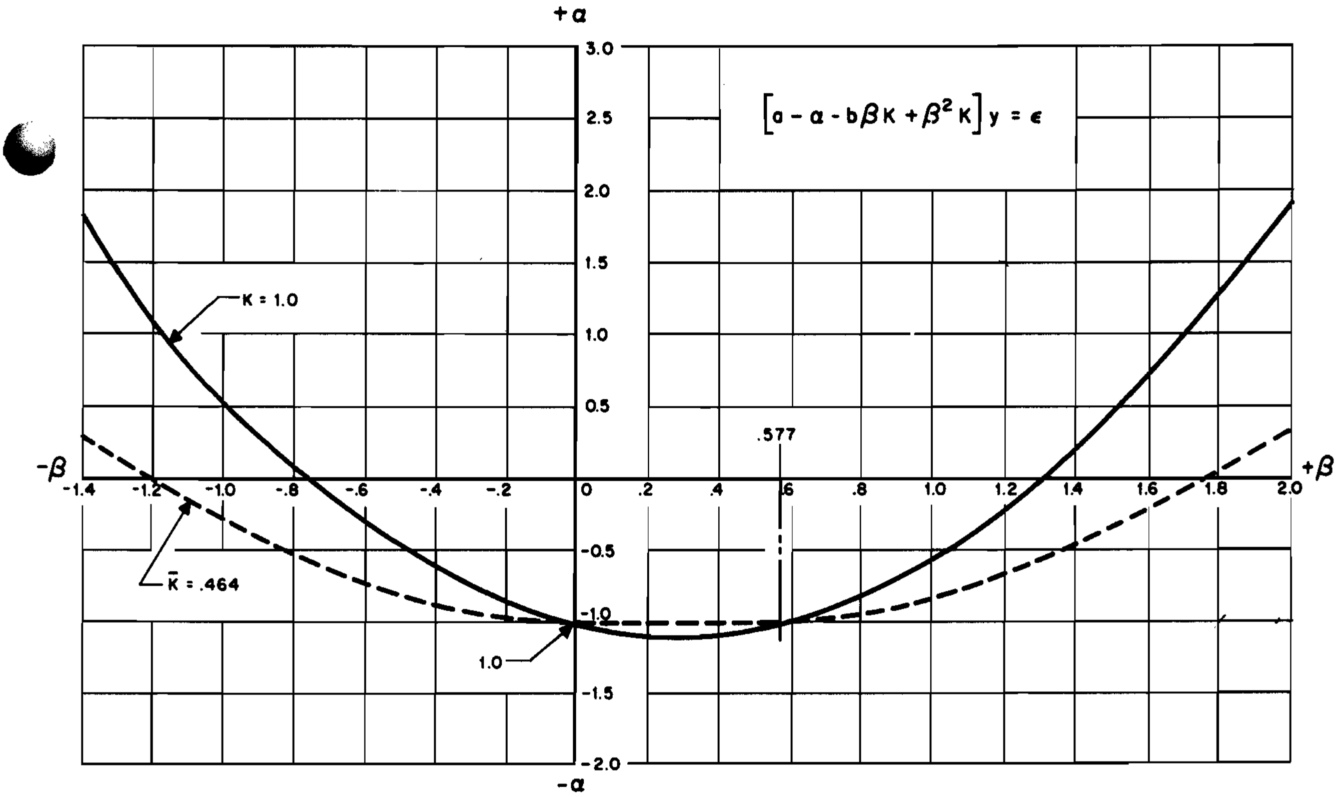


Figure 60. Zero-Error Trajectories for Two K Values

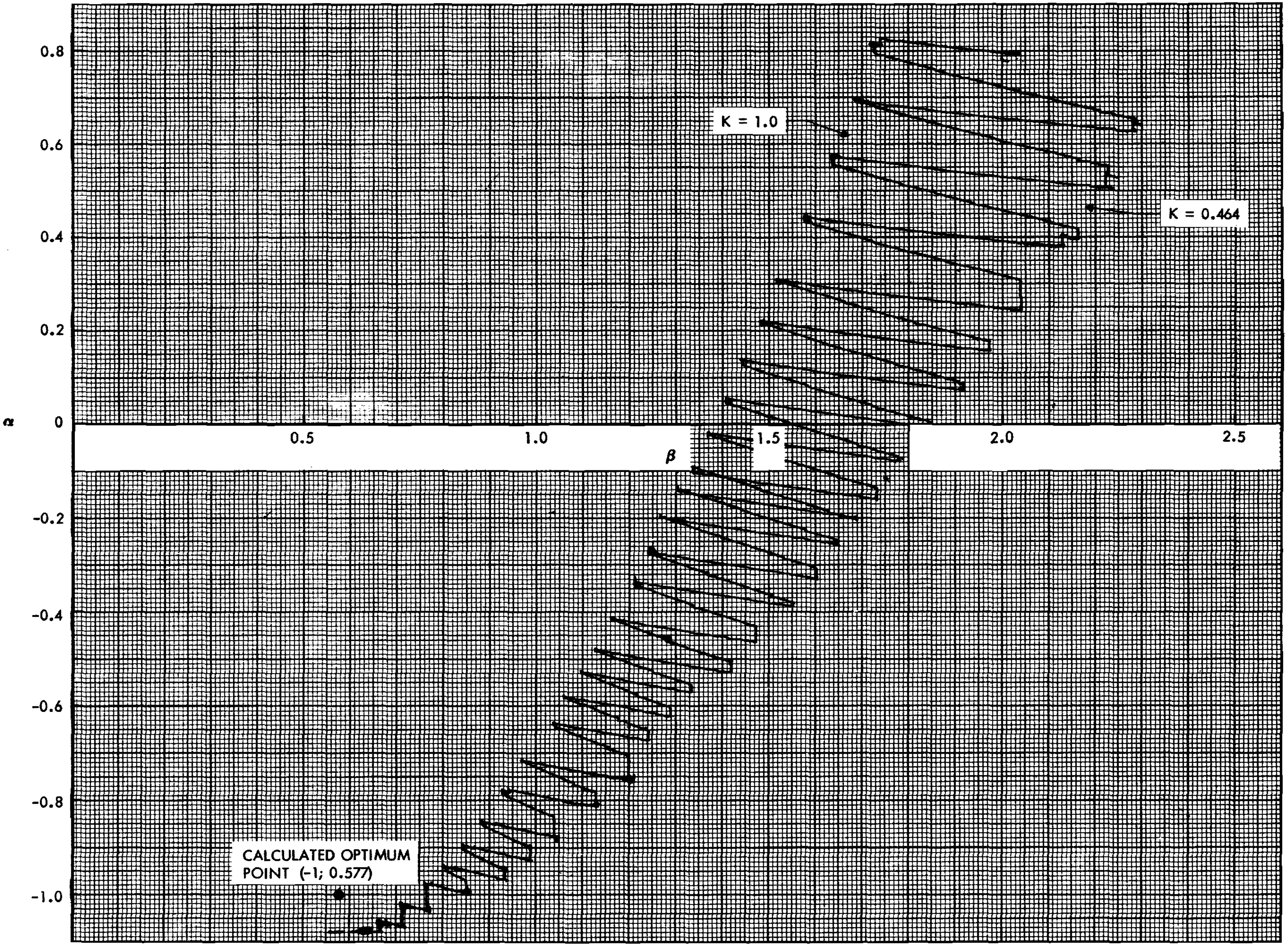


Figure 61. Typical Convergence Path for Initial Starting Point in Far Right Half of Plane

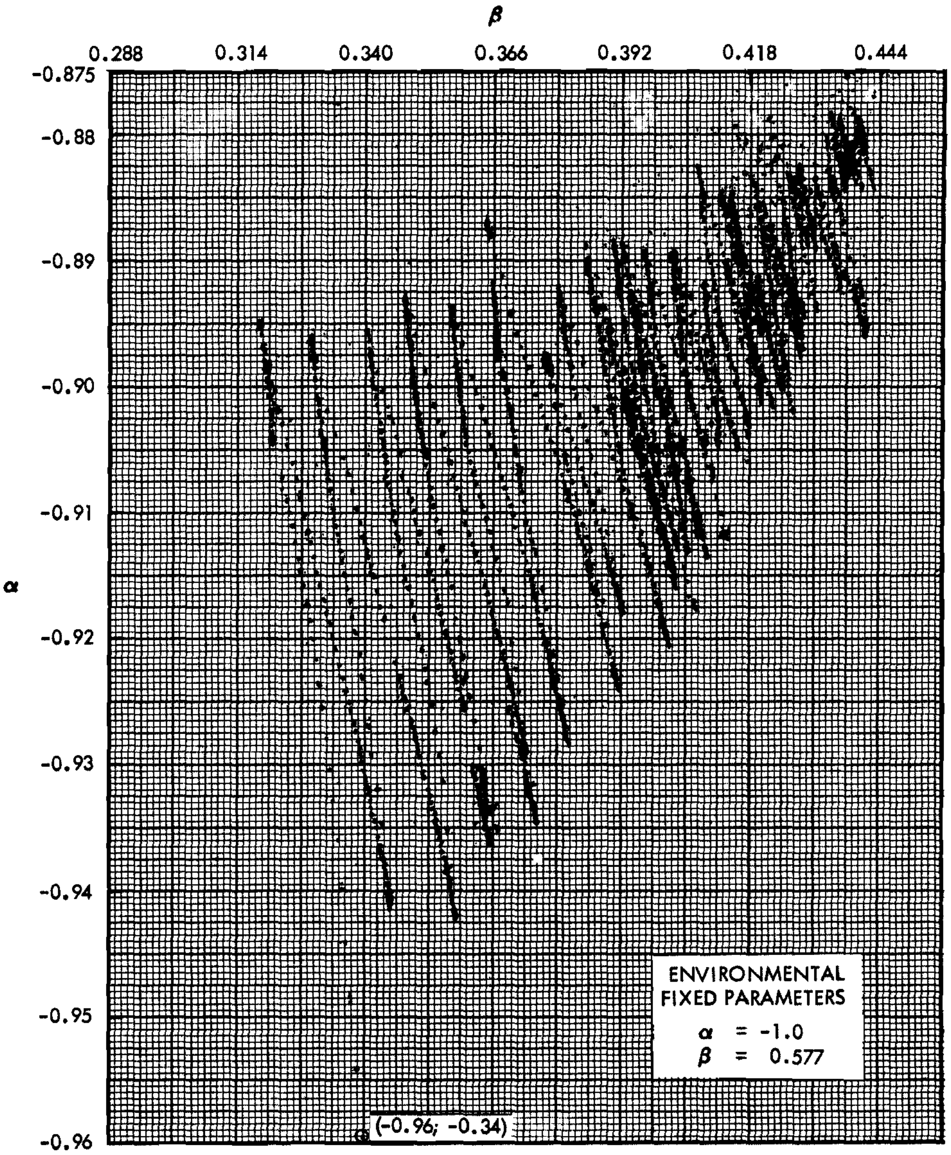


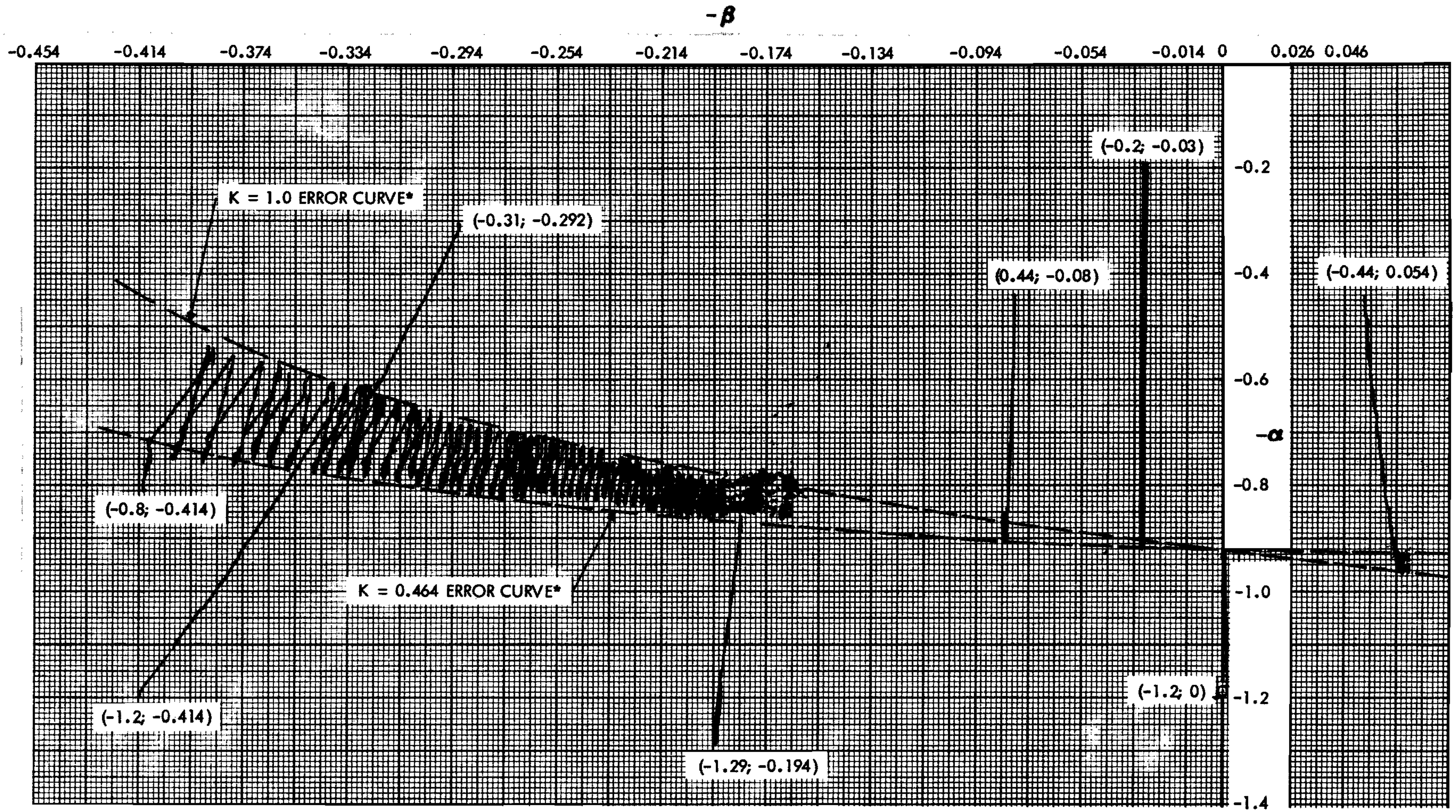
Figure 62. Typical Convergence Path for Initial Starting Point in Region between $\beta = 0$ and $\beta = B$

Figures 61, 62, and 63 show the results of the tests. Each figure is a section of the curve shown in Figure 60. Note the zigzag convergence path in all the figures. This path explains, at least in part, why the parameters in other experiments have taken so long to converge to the optimum values. The error equation and also the curve of Figure 60 show that with $\alpha = A$, $\beta$ can have two values $\beta = B$ and $\beta = 0$, regardless of the value of K. Whenever the starting point ($\alpha$, $\beta$) was on an error trajectory between those two points, the parameters moved to the right as shown in Figure 62 until they arrived at the $\beta = B$, $\alpha = A$ point. Figure 63 also shows the change in direction of movement for the initial starting point ($\alpha$, $\beta$) as this initial starting point was moved from the left half of the plane to the right half of the plane.

Notice in Figure 63 that when the initial starting point was in the left half of the plane, the parameters did not converge to the optimum values. There were two cases:

(1) As the parameters, which were zigzagging between the two zero-error curves, approached the $\beta = 0$ axis, progress toward the axis stopped.

(2) When the initial starting point for the neuron model's parameters were on the $\beta = 0$ axis, the parameters went straight to the ($\alpha = A$, $\beta = 0$) point and stopped.



*ERROR CURVES AS SEEN BY COMPUTER

Figure 63. Typical Convergence Path for Initial Starting Point in Left Half of Plane

In the first case, the noise level interfered with the parameter computation and stopped the progress toward the $\beta = 0$ axis. In the second case, the $\beta$ computation was dormant because $\beta = 0$ initially. The adjustment drive for the neuron model's $\beta$ parameter was proportional to the initial $\beta$, and in this case was zero. With no signal to adjust its $\beta$ parameter, the neuron model made no $\beta$ adjustments.

Generalizing the above result, the $\beta = 0$ point must be excluded as a possible value of $\beta$ to ensure that the neuron model's $\beta$ parameter will converge to the optimum value.

4. Tests on Redundant Networks

Three tests were performed on redundant networks using the optimal control concept. One of the tests used the transistor hardware previouslydeveloped for the neuron model. The other two tests used the analog computer exclusively for the simulation of redundant networks. So far, small-scale networks have been tested. The lessons learned from these small-scale networks wi ll be used as guidelines for the larger networks.

a. Redundant Network Test using Transistor Hardware. The first test used the Harmon neuromime (Reference 2) and other transistor hardware developed for the neuron models. This test was broken into two parts. The first part tested the reaction of the Harmon neuromime to weighted inputs. The second part tested the adjustment properties of a redundant network of neuromime elements.

The test for the first part was straightforward. A Harmon neuromime was set up having a weighted input. The weighted input had a constant pulse rate. The object of the test was to discover the weight required to force the Harmon neuromime to exactly follow the input pulse rate. Three different input pulse rates were used. Figure 64 shows the result of the test. Note that as the input pulse rate increases, the weight required to force the Harmon neuromime to follow also increases. Also, notice the plateau effect of the Harmon neuromime's output. The plateau occurs at a submultiple of the input pulse rate.

The test for the second part was set up as shown in Figure 65. The Harmon neuromimes were not directly in parallel. The inputs to all of the neuromimes were the same. All outputs were weighted and fed to another Harmon neuromime. This neuromime produced the network's output. Synaptic weight computers were set up on the analog computer. The task set for the redundant network was to maintain a constant input-output network relation, regardless of internal failure.

Figure 66 shows the synaptic weight values computed by the neuromimes as failures occurred. The failures were in the form of total neuromime function loss. Note that the synaptic weights computed by each of the surviving neuromimes increase to absorb the failures.

b. Redundant Network Tests using Analog Computer. The second test, shown in Figure 67, used the analog computer exclusively for the simulation. The environment used for this test was second order. For this test two neuron models were simulated in parallel, each contributing its share to the total system control. One of the neuron models had its parameters fixed at optimum values; the remaining neuron model had to compute its own optimum values. The first two columns of Figure 68 show the results of the computations.

After the two neuron models were controlling the system optimally, one of the neuron models was "destroyed." The last two columns of Figure 68 show the results of the one neuron model adapting to the severe internal damage. Even with only two neuron models in parallel, the loss of one did not affect the total system cost too drastically. Neither was the amount of adjustment required of the surviving neuron model drastic.

Thisresultcan be extended to networks with 10 or more neuron models. The effect of losing one, two, or three neuron models simultaneously in a network would not appreciably

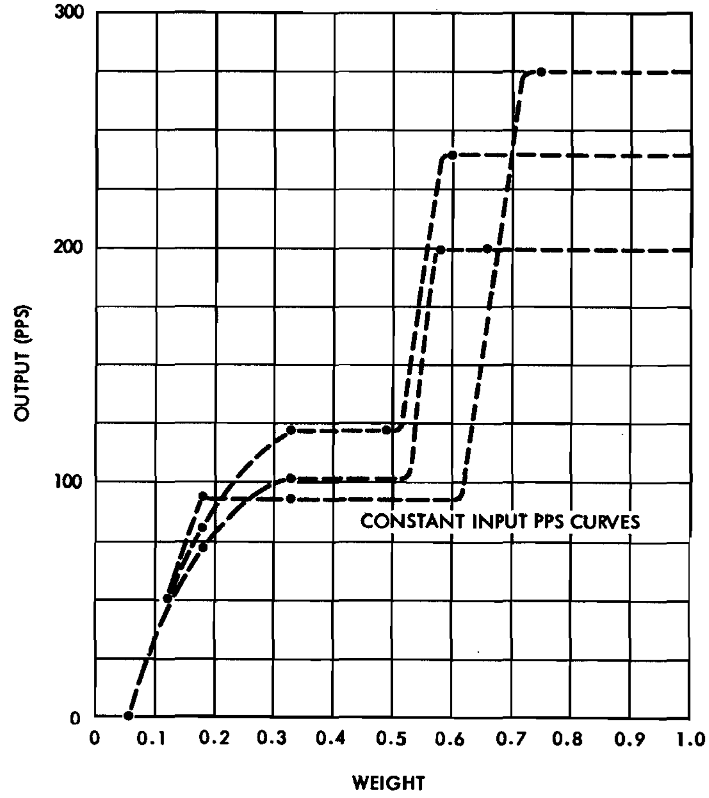


**Figure 64. Test Results on Redundant Networks Using Transistor Hardware**

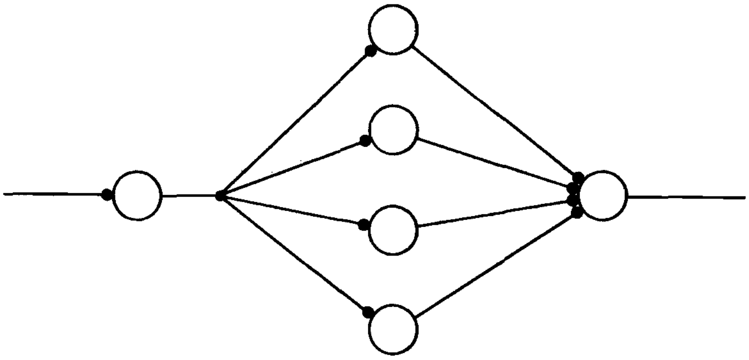


**Figure 65. Redundant Network of Neuromime Elements**

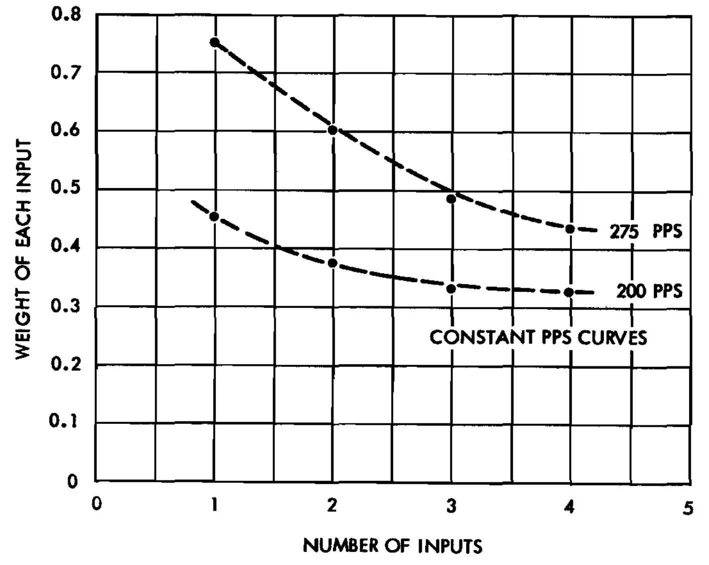


**Figure 66. Synaptic Weight Computations of Redundant Network**

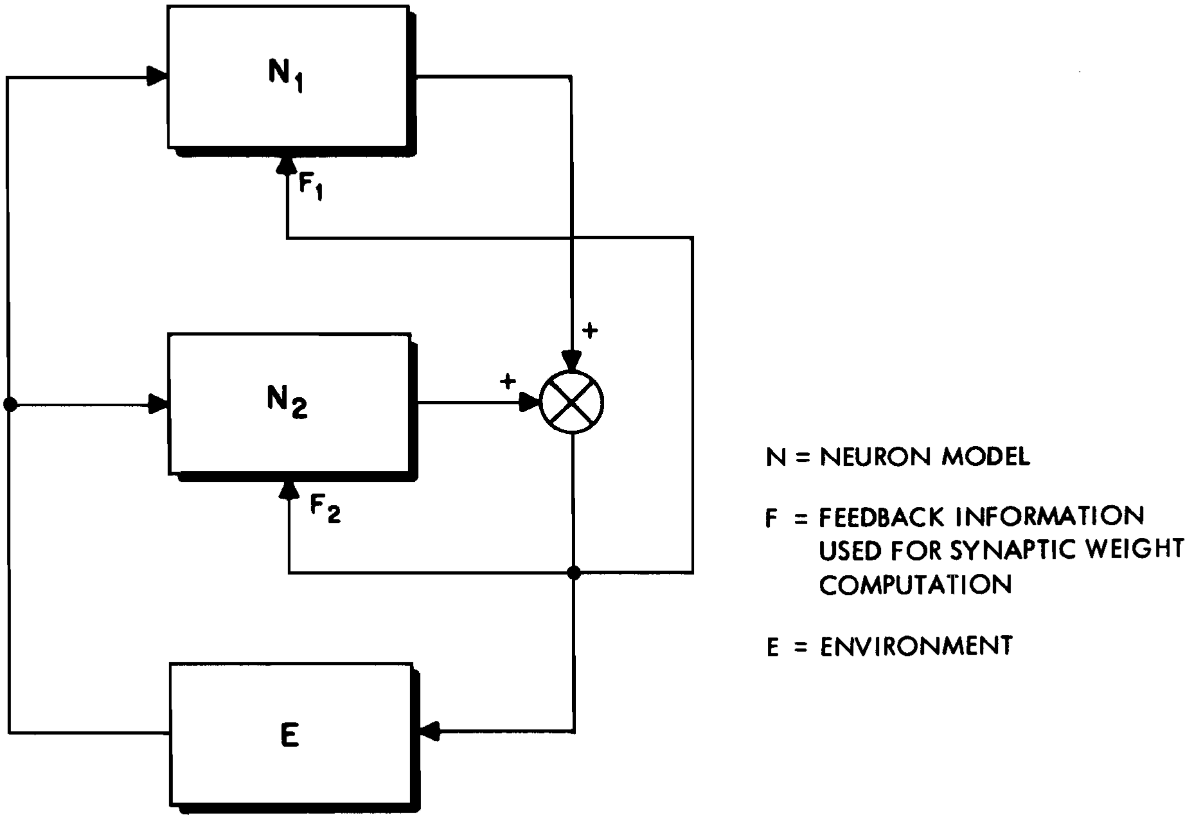

N1
F1
+
N2
+
F2
E
N = NEURON MODEL
F = FEEDBACK INFORMATION USED FOR SYNAPTIC WEIGHT COMPUTATION
E = ENVIRONMENT

Figure 67. Redundant Network with Two Neuron Models in Parallel

affect the total network response. Further, the larger the number of neuron models in a network, the smaller the dynamic range of an individual neuron model need be.

The third test, similar to the one in Figure 67, also used the analog computer exclusively for the neuron model simulation. Two neuron models were simulated in parallel. Each model contributed to the total system control. This test differed from the second test in that both neuron models were free to compute their optimum parameters. To simplify the instrumentation, a first-order environment was used for this test. Interest was focused on the ability of the two neuron models to begin at some initial state and adjust to the optimum state, with both sharing the control load equally. The optimum state for the two parallel neuron models occurred when the parameters of both were equal, and when these parameters were the optimum values.

Figure 69 shows a plot of parameter differences. The optimum point on the graph is at the origin. The graph shows that the parameters of the two parallel neuron models converged to the optimum values simultaneously. The graph shows only that the parameters of both neuron models adjusted until the neuron models shared equally the control load; it does not show that the final adjusted parameters were optimum. However, the parameters checked well with the optimum parameters when they were checked at the end of each run.

Note the zigzag convergence pattern in Figure 69. This pattern resembles the convergence patterns obtained for the convergence test of the previous section of this appendix. The convergence pattern observed in this test was also due to perturbations in the K parameter. The pattern was obtained by holding the K parameter of one neuron model fixed at a value different from that of the other neuron model.

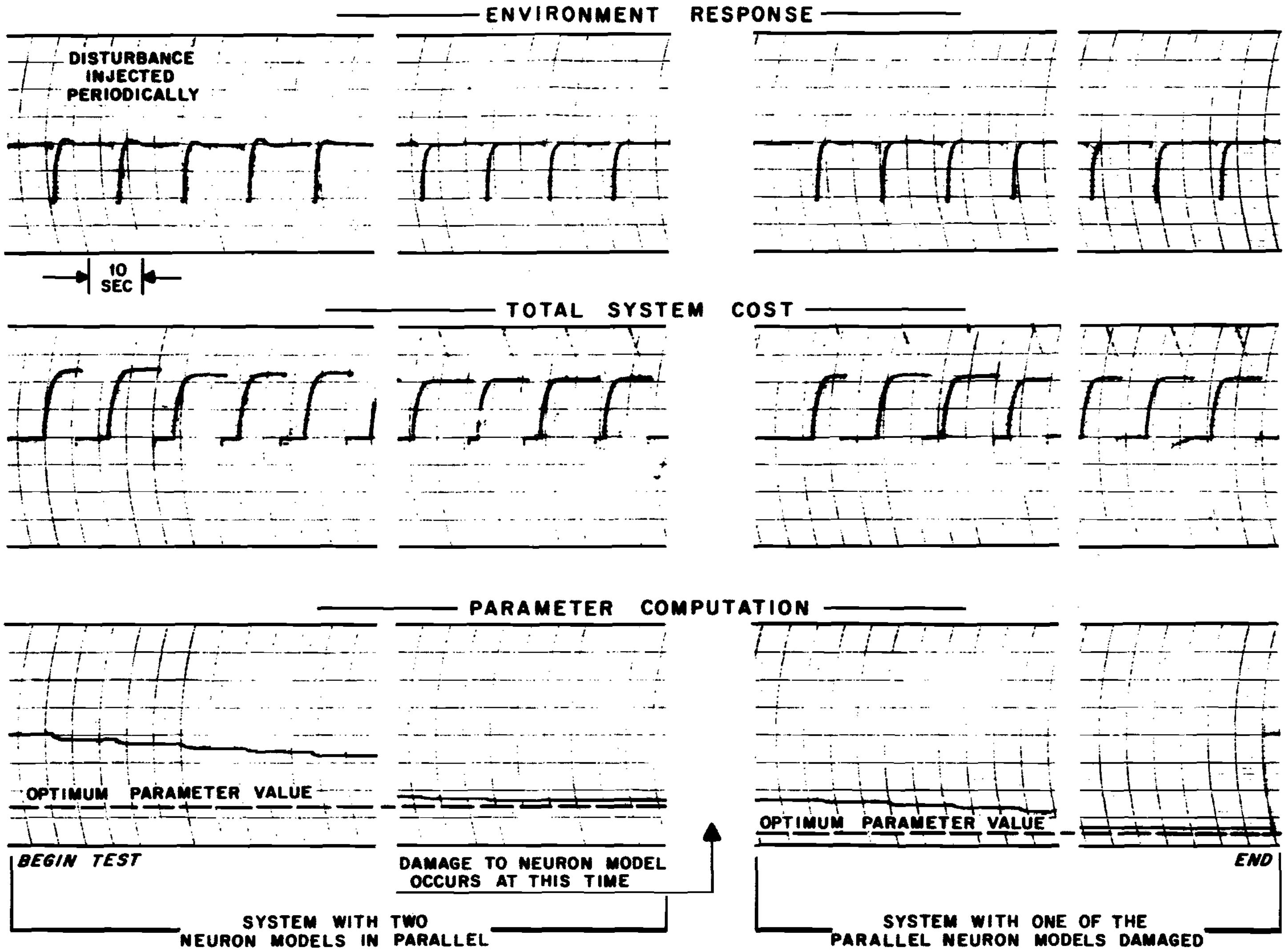


Figure 68. Adaptation after Internal Damage

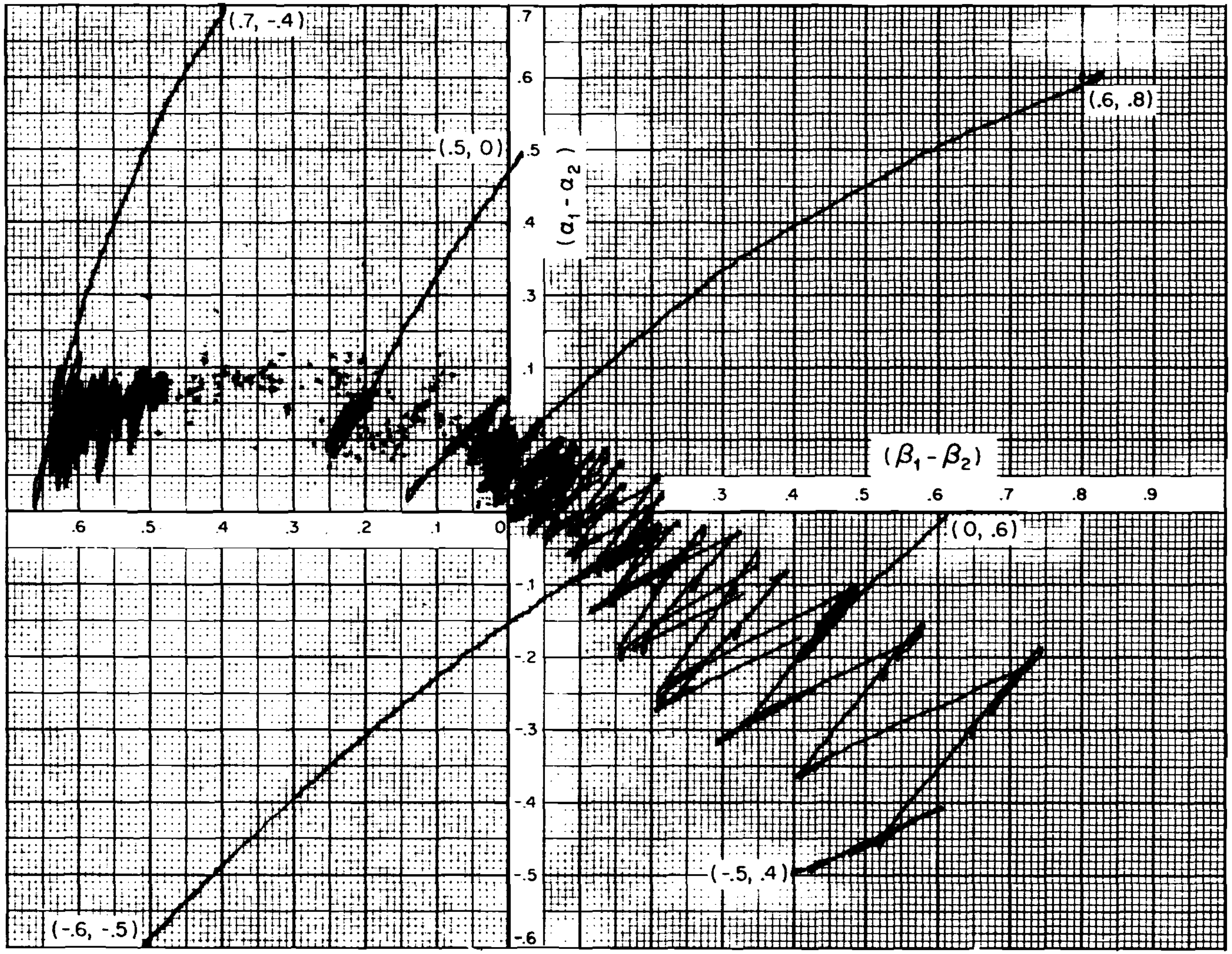


Figure 69. Plot of Parameter Differences - Two Neuron Models in Parallel

## 5. Tests on a Roll-Yaw Coupled Aircraft

A fifth-order environment was constructed to simulate an aircraft obeying the following aerodynamic equations:

$$\left.\begin{aligned}
&\dot{\beta} + \dot{\psi} - \phi \frac{g}{u0} - \psi\theta_0 \frac{g}{u0} = \beta \frac{y_\beta}{u0} + \dot{\phi}\frac{y_p}{u0} + \dot{\psi}\frac{y_r}{u0} \\
&\ddot{\phi} - \ddot{\psi} i_{zx} = \ell_\beta \beta + \ell_{\dot{\psi}}\dot{\psi} + \ell_{\dot{\phi}}\dot{\phi} + \ell_{\delta a}\delta_a \\
&\dot{\psi} - \ddot{\phi} k_{zx} = \eta_\beta \beta + \eta_{\dot{\beta}}\dot{\beta} + \eta_{\dot{\phi}}\dot{\phi} + \eta_{\delta r}\delta r
\end{aligned}\right\} \qquad (62)$$

where $\beta$ is the sideslip angle, $\psi$ the yaw angle, $\phi$ the roll angle, $\theta_0$ is a fixed pitch angle, $\delta_a$ the aileron defection, $\delta r$ the rudder deflection, and all other parameters are constants determined experimentally. All angles are measured with respect to the body axes of the aircraft.

Using standard state variable techniques, routine manipulation yields:

$$\left.\begin{aligned}
&\dot{\psi}_2 = Q_1\phi_2 + Q_2\psi_2 + Q_3\beta_1 + Q_4\psi_1 + Q_5\phi_1 - Q_6\delta_r - Q_7\delta_a \\
&\dot{\phi}_2 = R_1\phi_2 + R_2\psi_2 + R_3\beta_1 + R_4\psi_1 + R_5\phi_1 - R_6\delta_r - R_7\delta_a \\
&\dot{\beta}_1 = G_1\phi_2 - G_2\psi_2 + G_5\beta_1 + G_4\psi_1 + G_5\phi_5
\end{aligned}\right\} \qquad (63)$$

where $\psi 1 = \psi$, $\psi_2 = \dot{\psi}_1 = \dot{\psi}$, $\phi_1 = \phi$, $\phi_2 = \dot{\phi}_1 = \dot{\phi}$, and $\beta_1 = \beta$.

Converting to matrix form yields:

$$\begin{bmatrix} \dot{\psi}_1 \\ \dot{\psi}_2 \\ \dot{\phi}_1 \\ \dot{\phi}_2 \\ \dot{\beta}_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ Q_4 & Q_2 & Q_5 & Q_1 & Q_3 \\ 0 & 0 & 0 & 1 & 0 \\ R_4 & R_2 & R_5 & R_1 & R_3 \\ G_4 & -G_2 & G_5 & G_1 & G_3 \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \phi_1 \\ \phi_2 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -Q_6 & -Q_1 \\ Q & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -R_6 & -R_7 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \delta_r \\ \delta_a \end{bmatrix}$$

The above form corresponds to the linear matrix equation discussed in the text of the report.

$$\dot{Y} = AY + BM \qquad (64)$$

where Y is the environment output matrix, M is the control matrix, and A and B are environment coefficient matrices.

The aircraft equations (in state variable form) were then programmed on an analog computer as shown in Figure 70. The aircraft simulation was first tested without a controller to ensure that its response was equivalent to that of the aircraft being simulated. A digital computer program was used to determine the optimal control parameter matrix, K, necessary to control the aircraft simulation. A more thorough discussion of the digital program is presented in Appendix VIII. A nonadaptive optimal controller was then programmed on an analog computer and connected to the aircraft simulation.

Figure 71 shows the optimal controller simulation. Cost curves were run on the optimal control parameters in much the same manner as in previous experiments on optimal control systems. Two of the cost curves obtained are shown in Figure 72. Their quadratic shapes

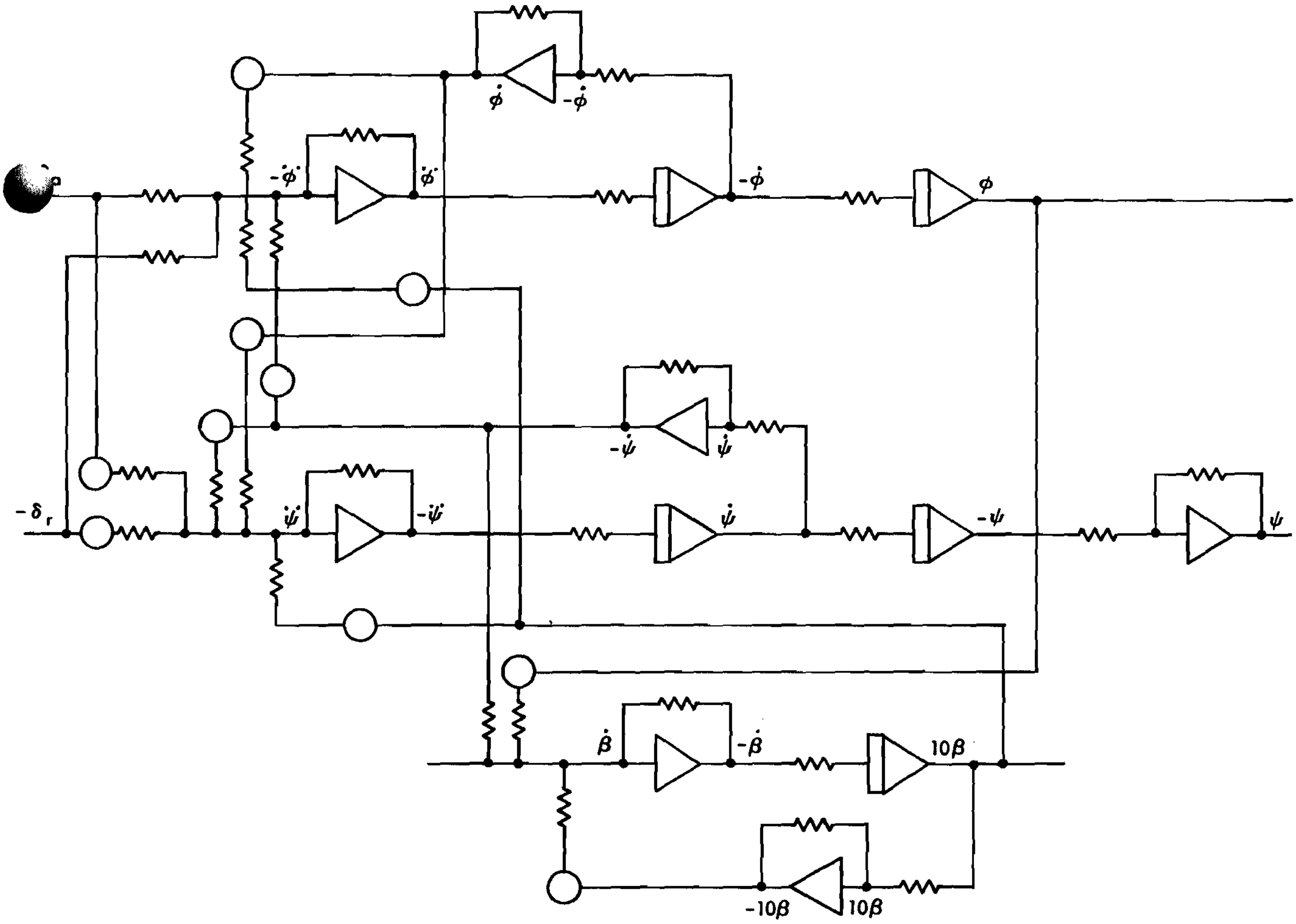


Figure 70. Aircraft Simulation

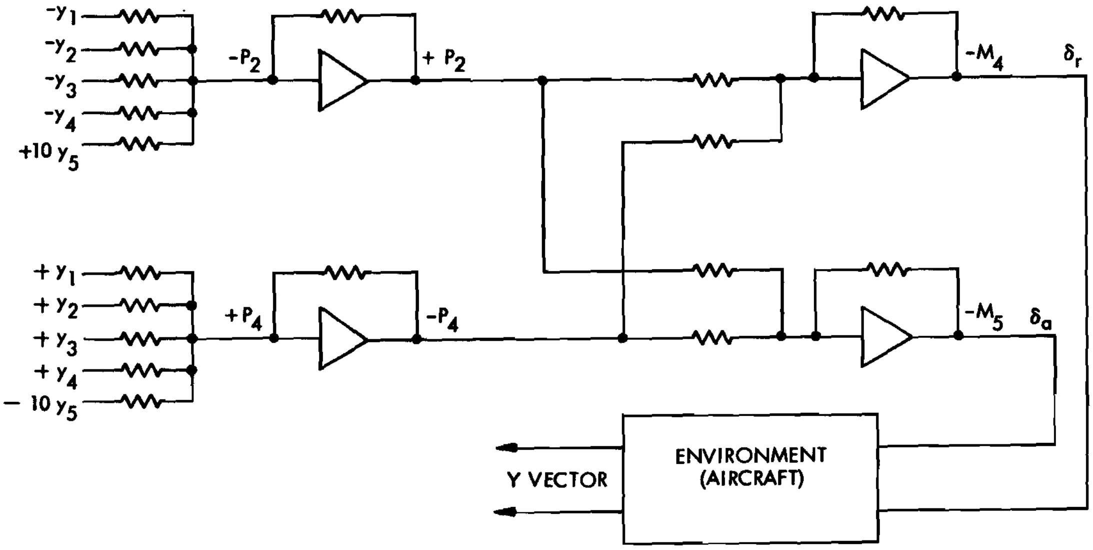


Figure 71. Nonadaptive Controller Simulation

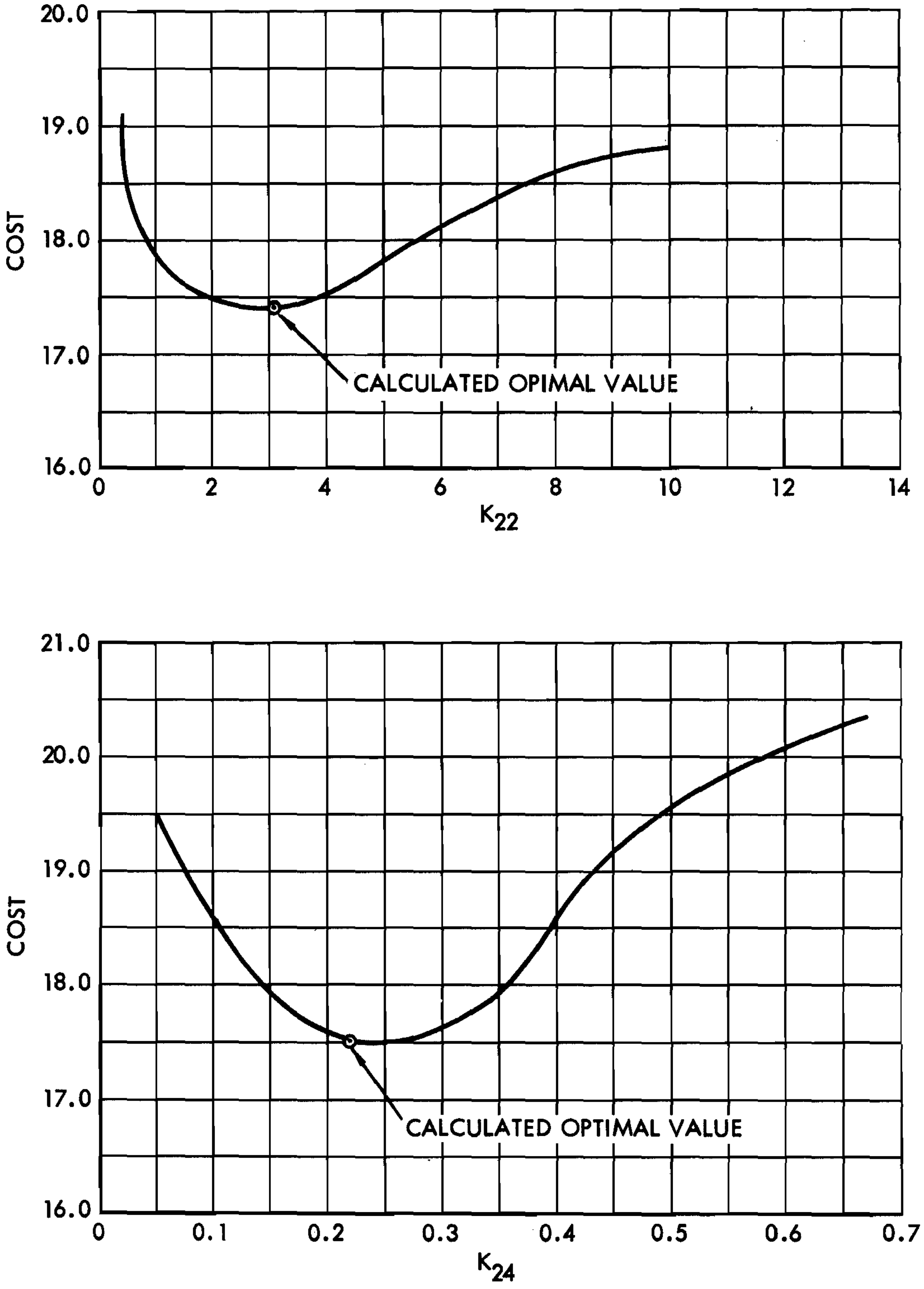


Figure 72. Typical Cost Curves Derived from Nonadaptive System

are similar to those of previous experiments and illustrate that the parameters computed by the digital computer are indeed optimum.

Following the successful checkout of the aircraft dynamics and the nonadaptive optimal controller, the adaptive capabilities were instrumented into the system and preliminary tests begun. Two basic equations were used to provide the adaptive capabilities. The error equation derived from the system equations and the parameter adjustment equation derived from the gradient method of steepest descent are shown in matrix form:

$$K\dot{Y} + A^t P + Y = \phi \tag{65}$$

$$\dot{K} = - \frac{\partial \phi}{\partial K} \text{Sgn}\, \phi \tag{66}$$

where K is the optimal control matrix, $\phi$ is a generalized error vector, and Sgn $\phi$ is the signum function with the following properties:

$$\begin{aligned} \text{Sgn}\,\phi &= \frac{\phi}{|\phi|} \\ \text{Sgn}\,\phi &= 0 \qquad \text{iff } \phi = 0 \end{aligned} \tag{67}$$

Figure 73 shows the instrumentation of the above equations using the analog computer and the transistor gating circuits. Only three of these parameters were made adjustable, because of the demand on the analog computer nonlinear equipment.

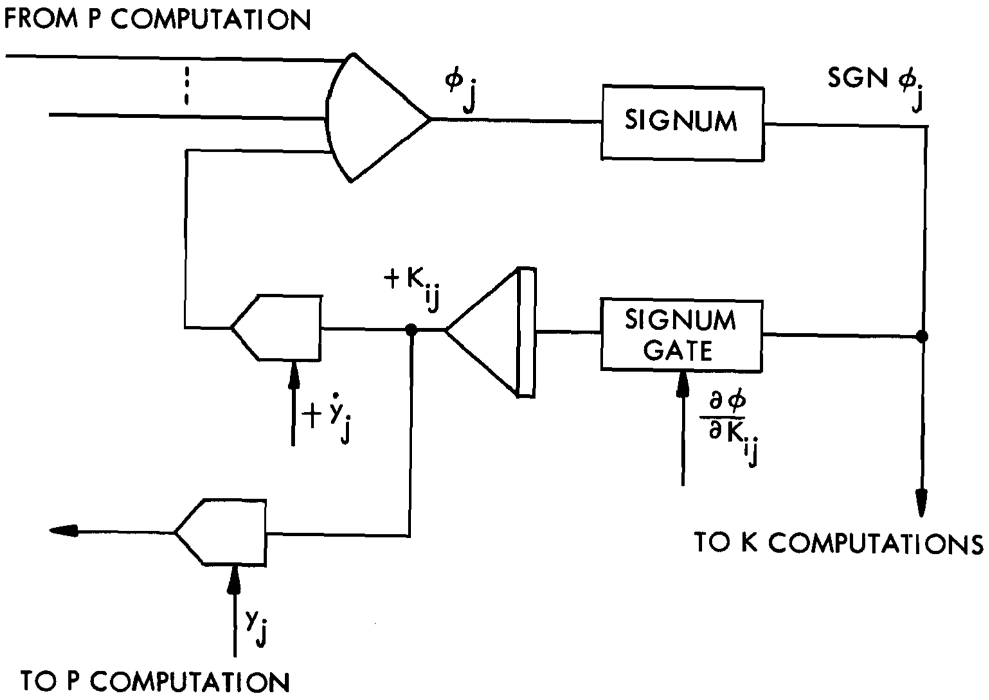


Figure 73. Instrumentation of Equations Providing Adaptive Characteristics

The first series of preliminary tests on the adaptive controller uncovered several problem areas. For these first tests no disturbances were used. Initial conditions were set on the aircraft dynamics and the system was allowed to return to normal. These tests showed that some of the parameters adjusted to values close to the predicted optimum, while others adjusted to values that differed drastically from their predicted values. In the second series of tests, in which disturbances were injected into the aircraft, some of the parameters diverged.

Most of the tests on the parameter adjustment were run using a disturbance vector with a single entry. To provide enough information for the parameters to adjust correctly, the fifth-order space (the total system) must be spanned by the disturbance vector. Analysis showed that the aircraft simulation required a disturbance with two entries to span the space. A transistorized pulse generator was developed to provide a periodic disturbance vector with two entries. Tests on the aircraft simulation using the new disturbance vector showed little improvement over previous results. Although the underlying problems were not completely defined, it is evident that hardware problems such as noisy gating circuits, circuit sensitivity, and transistor amplifier asymmetry significantly degraded system performance.

As in previous experiments, the adaptive controller instrumentation was plagued with noisy error detection circuits. A close analysis of the instrumentation revealed that to increase the signal-to-noise ratio in one section of the instrumentation would mean decreasing it somewhere else. Although the overall signal-to-noise ratio could be improved, different error detection techniques, such as correlation and prediction techniques, would probably have to be incorporated in the model to significantly improve the present error detection.

# APPENDIX VIII

## SOLUTION OF THE MATRIX RICCATI EQUATION

The set of linear system equations for the optimal control approach discussed in Section II led to the equation

$$KA + A^tK - KBB^tK + R = \phi,$$

which gives the relationship of the optimal control parameter matrix K to the process parameter matrices A and B. R is a diagonal matrix (in most cases, R = I, the identity matrix) and $\phi$ is an error matrix. This equation is clearly a matrix Riccati equation.

In general, an explicit solution to this matrix equation does not exist, and iterative techniques using the digital computer must be used to obtain an approximation.

A digital computer program was written which iterated the K matrix until $\phi = \epsilon$, where $\epsilon$ was a preselected tolerance. The adjustment of the K matrix was effected by letting $\phi = \dot{K}$.

$$\dot{K} \cong \frac{\Delta K}{\Delta t}$$

$$\Delta K = \dot{K}\Delta t = \phi\Delta t.$$

After each iteration, the new K matrix would then be

$$K_{new} = K_{old} + \Delta K$$

The new K matrix was then plugged into the Riccati equation and computation resumed. The time increment, $\Delta t$, chosen at the beginning of the program must be small enough to ensure convergence of the iteration process. Choosing $\Delta t$ small enough for a given problem was the major drawback of the computer program. Before very large matrices (10 x 10) can be handled efficiently, some method of accelerating the convergence process is needed.

Following is a list of the program's Fortran statements. This program was written for the IBM 360 computer.

```
//          EXEC  FORTRAN
      DIMENSION D(10,10),A(100),B(100),Q(100),ID(20),AT(100),
     1TEMP(100),R(10),AK(100),TEMP1(100),TEMP2(100),AKD(100),BT(100),
     2SAVE(100),T(100)
      DOUBLE PRECISION D,A,B,AT,Q,BT,TEMP,R,AK,DELT,TEMP1,TEMP2,S,AKD,
     1AK22,DAK22,ALAST,DUM,SAVE,T
      COMMON M,N,NIT,NNL,NNP,LINE,DELT,S,ID
 1    FORMAT(7F10.0)
 2    FORMAT(20A4)
 3    FORMAT(A4,10A1)
 4    FORMAT(9I4)
 5    FORMAT(1H0,'MAX ITERATIONS REACHED',I5)
 6    FORMAT(6F20.9)
      ALAST = 1.D-5
      LINE = 0
      CALL MASK(0)
C     MASK(0) STOPS JOB TERMINATION ON UNDERFLOW
      M1 = 10
      READ(1,3)IBLANK,IA,IB,IR,IK,IQ
      NNT = 10
 10   READ(1,2) ID
      IF(ID(1)-IBLANK)20,160,20
 20     READ(1,4) M,N,NIT,NNL,NNP,IOPT,IN,INLAST
        IF(M-M1)30,160,160
 30     READ(1,1) DELT,S
        CALL LOC (2,2,IJ,M,M,0)
        CALL READ(D,M)
 C      READ THE A MATRIX, PRINT, AND COMPACT
        CALL ARRAY (2,M,M,M1,M1,A,D)
        CALL PRINT(D,M,IA)
 C      READ THE B MATRIX, PRINT, AND COMPACT
        M2 = M*M
        CALL READ(D,M)
        CALL ARRAY(2,M,M,M1,M1,B,D)
        CALL PRINT(D,M,IB)
 C      READ DIAGONAL MATRIX
        READ(1,1) (R(I),I=1,M)
        CALL PRINT(R,M,IR)
 C      CLEAR THE K MATRIX
        DO 40 I=1,M2
        T(I)=0.D0
 40     AK(I) = 0.D0
        IF(IOPT)41,42,41
 41     CALL READ(D,M)
        CALL PRINT(D,M,IK)
        CALL ARRAY(2,M,M,M1,M1,AK,D)
 42     CONTINUE
 C      TRANSPOSE THE A MATRIX
        CALL MTRA(A,AT,M,M,0)
 C      TRANSPOSE THE B MATRIX
        CALL MTRA(B,BT,M,M,0)
 C      COMPUTE Q
        CALL MATA(BT,Q,M,M,0)
        NP = 0
        ISW = 0
        OAK22 = 1.D10
        N1 = 0
        NLAST = 0
 C      COMPUTE K-DOT
 45     CALL MPRD(AT,AK,TEMP,M,1,0,0,M)
        CALL MPRD(AK,A,TEMP1,M,M,0,0,M)
        CALL MPRD(AK,Q,TEMP2,M,M,0,1,M)
        CALL MPRD(TEMP2,AK,B,M,M,0,0,M)
        CALL MADD(TEMP,TEMP1,AKD,M,M,0,0)
        CALL MADD(AKD,R,AKD,M,M,0,2)
        CALL MSUB(AKD,B,AKD,M,M,0,0)
        DO 50 I=1,M2
 50     AKD(I)=AKD(I)*DELT
        CALL MADD(AK,AKD,AK,M,M,0,0)
        IF(ISW)140,60,140
 60     IF(NP-N1)80,70,80
 70     CALL ARRAY(1,M,M,M1,M1,AK,D)
        CALL PRINT(D ,M,IK)
        NP = NP  + NNP
 80     CONTINUE
 100    IF(N1-NIT)110,120,120
 110    N1 = N1+1
        GO TO 45
 120    WRITE(3,5)N1
 130    ISW = 1
        GO TO 45
 140    CALL ARRAY(1,M,M,M1,M1,AK,D)
        CALL PRINT(D,M,IK)
        IF(NLAST-NNL)150,10,10
 150    NLAST = NLAST + 1
        GO TO 45
 160    CALL PUNCH(D,M)
        CALL EXIT
        END
```

```
//        EXEC  FORTRAN
       SUBROUTINE PRINT (A,M,IC)                                         EA230
       DIMENSION A(10,10),ID(20)
       DOUBLE PRECISION A,DELT,S                                         EA230
       COMMON J,N,NIT,NNL,NNP,LINE,DELT,S,ID
  1    FORMAT(//3X,1HR,8X,6(  A1,I2,17X))                                EA230
  2    FORMAT(1X,I3,2X,F12.6)
  3    FORMAT(1X,I3,2X,F12.6, F20.6)
  4    FORMAT(1X,I3,2X,F12.6,2F20.6)
  5    FORMAT(1X,I3,2X,F12.6,3F20.6)
  6    FORMAT(1X,I3,2X,F12.6,4F20.6)
  7    FORMAT(1X,I3,2X,F12.6,5F20.6)
  8    FORMAT(1H1,52X,'JOB EA230'//10X,20A4//10X,'DELT-T',F10.6,10X,'S',
      1F10.3,10X,'ITERATIONS BETWEEN PRINTS',I5)
       L = M
       LK = L + 1
       DO 90 L1 = 1,L,6
       IF(LINE-LK)99,99,9
 99    WRITE(3,8) ID,DELT,S,NNP
       LINE = 40
  9    L2 = L1+5                                                         EA230
       IF(L-L2)10,20,20                                                  EA230
 10    L2=L                                                              EA230
 20    WRITE(3,1) (IC,N,N = L1,L2)
       JFORK = L2-L1+1
       GO TO (30,40,50,60,70,80),JFORK
 30    WRITE(3,2) (I,(A(I,K),K=L1,L2),I=1,M)
       GO TO 90
 40    WRITE(3,3) (I,(A(I,K),K=L1,L2),I=1,M)
       GO TO 90
 50    WRITE(3,4) (I,(A(I,K),K=L1,L2),I=1,M)
       GO TO 90
 60    WRITE(3,5) (I,(A(I,K),K=L1,L2),I=1,M)
       GO TO 90
 70    WRITE(3,6) (I,(A(I,K),K=L1,L2),I=1,M)
       GO TO 90
 80    WRITE(3,7) (I,(A(I,K),K=L1,L2),I=1,M)
 90    LINE = LINE-LK
       RETURN                                                            EA230
       END                                                               EA230

//        EXEC  FORTRAN
       SUBROUTINE PUNCH(D,M)
       DIMENSION D(10,10)
       DOUBLE PRECISION D
  1    FORMAT(5F10.6)
  9    FORMAT(//)
       DO 10 I=1,M
 10    WRITE(2,1) (D(I,J), J=1,M)
       WRITE(2,9)
       RETURN
       END

//        EXEC  FORTRAN
C         SUBROUTINE MADD                                               MADD 004
C                                                                       MADD 005
C         PURPOSE                                                       MADD 006
C            ADD TWO MATRICES ELEMENT BY ELEMENT TO FORM RESULTANT      MADD 007
C            MATRIX                                                     MADD 008
C                                                                       MADD 009
C         USAGE                                                         MADD 010
C            CALL MADD%A,B,R,N,M,MSA,MSB<                               MADD 011
C                                                                       MADD 012
C         DESCRIPTION OF PARAMETERS                                     MADD 013
C            A - NAME OF INPUT MATRIX                                   MADD 014
```

```
C           B - NAME OF INPUT MATRIX                                    MADD 015
C           R - NAME OF OUTPUT MATRIX                                   MADD 016
C           N - NUMBER OF ROWS IN A,B,R                                 MADD 017
C           M - NUMBER OF COLUMNS IN A,B,R                              MADD 018
C           MSA - ONE DIGIT NUMBER FOR STORAGE MODE OF MATRIX A         MADD 019
C                  0 - GENERAL                                          MADD 020
C                  1 - SYMMETRIC                                        MADD 021
C                  2 - DIAGONAL                                         MADD 022
C           MSB - SAME AS MSA EXCEPT FOR MATRIX B                       MADD 023
C                                                                       MADD 024
C        REMARKS                                                        MADD 025
C           NONE                                                        MADD 026
C                                                                       MADD 027
C        SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED                  MADD 028
C           LOC                                                         MADD 029
C                                                                       MADD 030
C        METHOD                                                         MADD 031
C           STORAGE MODE OF OUTPUT MATRIX IS FIRST DETERMINED. ADDITION MADD 032
C           OF CORRESPONDING ELEMENTS IS THEN PERFORMED.                MADD 033
C           THE FOLLOWING TABLE SHOWS THE STORAGE MODE OF THE OUTPUT    MADD 034
C           MATRIX FOR ALL COMBINATIONS OF INPUT MATRICES               MADD 035
C                       A                B                R             MADD 036
C                    GENERAL          GENERAL          GENERAL          MADD 037
C                    GENERAL          SYMMETRIC        GENERAL          MADD 038
C                    GENERAL          DIAGONAL         GENERAL          MADD 039
C                    SYMMETRIC        GENERAL          GENERAL          MADD 040
C                    SYMMETRIC        SYMMETRIC        SYMMETRIC        MADD 041
C                    SYMMETRIC        DIAGONAL         SYMMETRIC        MADD 042
C                    DIAGONAL         GENERAL          GENERAL          MADD 043
C                    DIAGONAL         SYMMETRIC        SYMMETRIC        MADD 044
C                    DIAGONAL         DIAGONAL         DIAGONAL         MADD 045
C                                                                       MADD 046
C     ..................................................................MADD 047

//       EXEC  FORTRAN
C        SUBROUTINE MSUB                                                MSUB 004
C                                                                       MSUB 005
C        PURPOSE                                                        MSUB 006
C           SUBTRACT TWO MATRICES ELEMENT BY ELEMENT TO FORM RESULTANT  MSUB 007
C           MATRIX                                                      MSUB 008
C                                                                       MSUB 009
C        USAGE                                                          MSUB 010
C           CALL MSUB%A,B,R,N,M,MSA,MSB<                                MSUB 011
C                                                                       MSUB 012
C        DESCRIPTION OF PARAMETERS                                      MSUB 013
C           A - NAME OF INPUT MATRIX                                    MSUB 014
C           B - NAME OF INPUT MATRIX                                    MSUB 015
C           R - NAME OF OUTPUT MATRIX                                   MSUB 016
C           N - NUMBER OF ROWS IN A,B,R                                 MSUB 017
C           M - NUMBER OF COLUMNS IN A,B,R                              MSUB 018
C           MSA - ONE DIGIT NUMBER FOR STORAGE MODE OF MATRIX A         MSUB 019
C                  0 - GENERAL                                          MSUB 020
C                  1 - SYMMETRIC                                        MSUB 021
C                  2 - DIAGONAL                                         MSUB 022
C           MSB - SAME AS MSA EXCEPT FOR MATRIX B                       MSUB 023
C                                                                       MSUB 024
C        REMARKS                                                        MSUB 025
C           NONE                                                        MSUB 026
C                                                                       MSUB 027
C        SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED                  MSUB 028
C           LOC                                                         MSUB 029
C                                                                       MSUB 030
C        METHOD                                                         MSUB 031
C           STRUCTURE OF OUTPUT MATRIX IS FIRST DETERMINED. SUBTRACTION MSUB 032
C           OF MATRIX B ELEMENTS FROM CORRESPONDING MATRIX A ELEMENTS   MSUB 033
C           IS THEN PERFORMED.                                          MSUB 034
```

```
C           THE FOLLOWING TABLE SHOWS THE STORAGE MODE OF THE OUTPUT      MSUB 035
C           MATRIX FOR ALL COMBINATIONS OF INPUT MATRICES                 MSUB 036
C                         A                 B                 R           MSUB 037
C                      GENERAL           GENERAL           GENERAL        MSUB 038
C                      GENERAL           SYMMETRIC         GENERAL        MSUB 039
C                      GENERAL           DIAGONAL          GENERAL        MSUB 040
C                      SYMMETRIC         GENERAL           GENERAL        MSUB 041
C                      SYMMETRIC         SYMMETRIC         SYMMETRIC      MSUB 042
C                      SYMMETRIC         DIAGONAL          SYMMETRIC      MSUB 043
C                      DIAGONAL          GENERAL           GENERAL        MSUB 044
C                      DIAGONAL          SYMMETRIC         SYMMETRIC      MSUB 045
C                      DIAGONAL          DIAGONAL          DIAGONAL       MSUB 046
C                                                                         MSUB 047
C     ..................................................................MSUB 048.

//       EXEC  FORTRAN
C        SUBROUTINE MPRD                                                  MPRD 004
C                                                                         MPRD 005
C           MULTIPLY TWO MATRICES TO FORM A RESULTANT MATRIX              MPRD 007
C                                                                         MPRD 008
C        USAGE                                                            MPRD 009
C           CALL MPRD%A,B,R,N,M,MSA,MSB,L<                                MPRD 010
C                                                                         MPRD 011
C        DESCRIPTION OF PARAMETERS                                        MPRD 012
C           A - NAME OF FIRST INPUT MATRIX                                MPRD 013
C           B - NAME OF SECOND INPUT MATRIX                               MPRD 014
C           R - NAME OF OUTPUT MATRIX                                     MPRD 015
C           N - NUMBER OF ROWS IN A AND R                                 MPRD 016
C           M - NUMBER OF COLUMNS IN A AND ROWS IN B                      MPRD 017
C           MSA - ONE DIGIT NUMBER FOR STORAGE MODE OF MATRIX A           MPRD 018
C                  0 - GENERAL                                            MPRD 019
C                  1 - SYMMETRIC                                          MPRD 020
C                  2 - DIAGONAL                                           MPRD 021
C           MSB - SAME AS MSA EXCEPT FOR MATRIX B                         MPRD 022
C           L - NUMBER OF COLUMNS IN B AND R                              MPRD 023
C                                                                         MPRD 024
C        REMARKS                                                          MPRD 025
C           MATRIX R CANNOT BE IN THE SAME LOCATION AS MATRICES A OR B    MPRD 026
C           NUMBER OF COLUMNS OF MATRIX A MUST BE EQUAL TO NUMBER OF ROWMPRD 027
C           OF MATRIX B                                                   MPRD 028
C                                                                         MPRD 029
C        SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED                    MPRD 030
C           LOC                                                           MPRD 031
C                                                                         MPRD 032
C        METHOD                                                           MPRD 033
C           THE M BY L MATRIX B IS PREMULTIPLIED BY THE N BY M MATRIX A   MPRD 034
C           AND THE RESULT IS STORED IN THE N BY L MATRIX R. THIS IS A    MPRD 035
C           ROW INTO COLUMN PRODUCT.                                      MPRD 036
C           THE FOLLOWING TABLE SHOWS THE STORAGE MODE OF THE OUTPUT      MPRD 037
C           MATRIX FOR ALL COMBINATIONS OF INPUT MATRICES                 MPRD 038
C                         A                 B                 R           MPRD 039
C                      GENERAL           GENERAL           GENERAL        MPRD 040
C                      GENERAL           SYMMETRIC         GENERAL        MPRD 041
C                      GENERAL           DIAGONAL          GENERAL        MPRD 042
C                      SYMMETRIC         GENERAL           GENERAL        MPRD 043
C                      SYMMETRIC         SYMMETRIC         GENERAL        MPRD 044
C                      SYMMETRIC         DIAGONAL          GENERAL        MPRD 045
C                      DIAGONAL          GENERAL           GENERAL        MPRD 046
C                      DIAGONAL          SYMMETRIC         GENERAL        MPRD 047
C                      DIAGONAL          DIAGONAL          DIAGONAL       MPRD 048
C                                                                         MPRD 049
C                                                                         MPRD 051
      SUBROUTINE MPRD%A,B,R,N,M,MSA,MSB,L<                                MPRD 052
      DIMENSION A%1<,B%1<,R%1<                                            MPRD 053
      DOUBLE PRECISION A,B,R
C                                                                         MPRD 054
```

```
C        SPECIAL CASE FOR DIAGONAL BY DIAGONAL                             MPRD 055
C                                                                          MPRD 056
      MS#MSA*10&MSB                                                        MPRD 057
      IF%MS-22< 30,10,30                                                   MPRD 058
   10 DO 20 I#1,N                                                          MPRD 059
   20 R%I<#A%I<*B%I<                                                       MPRD 060
      RETURN                                                               MPRD 061
C                                                                          MPRD 062
C        ALL OTHER CASES                                                   MPRD 063
C                                                                          MPRD 064
   30 IR#1                                                                 MPRD 065
      DO 90 K#1,L                                                          MPRD 066
      DO 90 J#1,N                                                          MPRD 067
      R%IR<#0                                                              MPRD 068
      DO 80 I#1,M                                                          MPRD 069
      IF%MS< 40,60,40                                                      MPRD 070
   40 CALL LOC%J,I,IA,N,M,MSA<                                             MPRD 071
      CALL LOC%I,K,IB,M,L,MSB<                                             MPRD 072
      IF%IA< 50,80,50                                                      MPRD 073
   50 IF%IB< 70,80,70                                                      MPRD 074
   60 IA#N*%I-1<&J                                                         MPRD 075
      IB#M*%K-1<&I                                                         MPRD 076
   70 R%IR<#R%IR<&A%IA<*B%IB<                                              MPRD 077
   80 CONTINUE                                                             MPRD 078
   90 IR#IR&1                                                              MPRD 079
      RETURN                                                               MPRD 080
      END                                                                  MPRD 081
```

```
//       EXEC  FORTRAN
C        SUBROUTINE MTRA                                                   MTRA 004
C                                                                          MTRA 005
C        PURPOSE                                                           MTRA 006
C           TRANSPOSE A MATRIX                                             MTRA 007
C                                                                          MTRA 008
C        USAGE                                                             MTRA 009
C           CALL MTRA%A,R,N,M,MS<                                          MTRA 010
C                                                                          MTRA 011
C        DESCRIPTION OF PARAMETERS                                         MTRA 012
C           A - NAME OF MATRIX TO BE TRANSPOSED                            MTRA 013
C           R - NAME OF OUTPUT MATRIX                                      MTRA 014
C           N - NUMBER OF ROWS OF A AND COLUMNS OF R                       MTRA 015
C           M - NUMBER OF COLUMNS OF A AND ROWS OF R                       MTRA 016
C           MS  - ONE DIGIT NUMBER FOR STORAGE MODE OF MATRIX A %AND R<    MTRA 017
C                  0 - GENERAL                                             MTRA 018
C                  1 - SYMMETRIC                                           MTRA 019
C                  2 - DIAGONAL                                            MTRA 020
C                                                                          MTRA 021
C        REMARKS                                                           MTRA 022
C           MATRIX R CANNOT BE IN THE SAME LOCATION AS MATRIX A            MTRA 023
C                                                                          MTRA 024
C        SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED                     MTRA 025
C           MCPY                                                           MTRA 026
C                                                                          MTRA 027
C        METHOD                                                            MTRA 028
C           TRANSPOSE N BY M MATRIX A TO FORM M BY N MATRIX R BY MOVING    MTRA 029
C           EACH ROW OF A INTO THE CORRESPONDING COLUMN OF R. IF MATRIX    MTRA 030
C           A IS SYMMETRIC OR DIAGONAL, MATRIX R IS THE SAME AS A.         MTRA 031
C                                                                          MTRA 032
C     ..................................................................MTRA 033
```

```
//       EXEC  FORTRAN
C        SUBROUTINE MCPY                                                MCPY 004
C                                                                       MCPY 005
C        PURPOSE                                                        MCPY 006
C           COPY ENTIRE MATRIX                                          MCPY 007
C                                                                       MCPY 008
C        USAGE                                                          MCPY 009
C           CALL MCPY %A,R,N,M,MS<                                      MCPY 010
C                                                                       MCPY 011
C        DESCRIPTION OF PARAMETERS                                      MCPY 012
C           A - NAME OF INPUT MATRIX                                    MCPY 013
C           R - NAME OF OUTPUT MATRIX                                   MCPY 014
C           N - NUMBER OF ROWS IN A OR R                                MCPY 015
C           M - NUMBER OF COLUMNS IN A OR R                             MCPY 016
C           MS  - ONE DIGIT NUMBER FOR STORAGE MODE OF MATRIX A %AND R< MCPY 017
C                  0 - GENERAL                                          MCPY 018
C                  1 - SYMMETRIC                                        MCPY 019
C                  2 - DIAGONAL                                         MCPY 020
C                                                                       MCPY 021
C        REMARKS                                                        MCPY 022
C           NONE                                                        MCPY 023
C                                                                       MCPY 024
C        SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED                  MCPY 025
C           LOC                                                         MCPY 026
C                                                                       MCPY 027
C        METHOD                                                         MCPY 028
C           EACH ELEMENT OF MATRIX A IS MOVED TO THE CORRESPONDING      MCPY 029
C           ELEMENT OF MATRIX R                                         MCPY 030
C                                                                       MCPY 033
      SUBROUTINE MCPY%A,R,N,M,MS<                                       MCPY 034
      DIMENSION A%1<,R%1<                                               MCPY 035
      DOUBLE PRECISION A,R
C        COMPUTE VECTOR LENGTH, IT                                      MCPY 037
C                                                                       MCPY 038
      CALL LOC%N,M,IT,N,M,MS<                                           MCPY 039
C                                                                       MCPY 040
C        COPY MATRIX                                                    MCPY 041
C                                                                       MCPY 042
      DO 1 I#1,IT                                                       MCPY 043
    1 R%I<#A%I<                                                         MCPY 044
      RETURN                                                            MCPY 045
      END                                                               MCPY 046
```

```
//       EXEC  FORTRAN
C        SUBROUTINE MATA                                                MATA 004
C                                                                       MATA 005
C        PURPOSE                                                        MATA 006
C           PREMULTIPLY A MATRIX BY ITS TRANSPOSE TO FORM A             MATA 007
C           SYMMETRIC MATRIX                                            MATA 008
C                                                                       MATA 009
C        USAGE                                                          MATA 010
C           CALL MATA%A,R,N,M,MS<                                       MATA 011
C                                                                       MATA 012
C        DESCRIPTION OF PARAMETERS                                      MATA 013
C           A  - NAME OF INPUT MATRIX                                   MATA 014
C           R  - NAME OF OUTPUT MATRIX                                  MATA 015
C           N  - NUMBER OF ROWS IN A                                    MATA 016
C           M  - NUMBER OF COLUMNS IN A. ALSO NUMBER OF ROWS AND        MATA 017
C                NUMBER OF COLUMNS OF R.                                MATA 018
C           MS  - ONE DIGIT NUMBER FOR STORAGE MODE OF MATRIX A         MATA 019
C                  0 - GENERAL                                          MATA 020
C                  1 - SYMMETRIC                                        MATA 021
C                  2 - DIAGONAL                                         MATA 022
C                                                                       MATA 023
```

```
C        REMARKS                                                        MATA 024
C           MATRIX R CANNOT BE IN THE SAME LOCATION AS MATRIX A         MATA 025
C           MATRIX R IS ALWAYS A SYMMETRIC MATRIX WITH A STORAGE MODE#1 MATA 026
C                                                                       MATA 027
C        SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED                  MATA 028
C           LOC                                                         MATA 029
C                                                                       MATA 030
C        METHOD                                                         MATA 031
C           CALCULATION OF %A TRANSPOSE A< RESULTS IN A SYMMETRIC MATRIXMATA 032
C           REGARDLESS OF THE STORAGE MODE OF THE INPUT MATRIX. THE     MATA 033
C           ELEMENTS OF MATRIX A ARE NOT CHANGED.                       MATA 034
C                                                                       MATA 035
C     ..................................................................MATA 036


//       EXEC  FORTRAN
C        SUBROUTINE ARRAY                                               ARRAY004
C                                                                       ARRAY005
C        PURPOSE                                                        ARRAY006
C           CONVERT DATA ARRAY FROM SINGLE TO DOUBLE DIMENSION OR VICE  ARRAY007
C           VERSA.  THIS SUBROUTINE IS USED TO LINK THE USER PROGRAM    ARRAY008
C           WHICH HAS DOUBLE DIMENSION ARRAYS AND THE SSP SUBROUTINES   ARRAY009
C           WHICH OPERATE ON ARRAYS OF DATA IN A VECTOR FASHION.        ARRAY010
C                                                                       ARRAY011
C        USAGE                                                          ARRAY012
C           CALL ARRAY %MODE,I,J,N,M,S,D<                               ARRAY013
C                                                                       ARRAY014
C        DESCRIPTION OF PARAMETERS                                      ARRAY015
C           MODE - CODE INDICATING TYPE OF CONVERSION                   ARRAY016
C                        1 - FROM SINGLE TO DOUBLE DIMENSION            ARRAY017
C                        2 - FROM DOUBLE TO SINGLE DIMENSION            ARRAY018
C           I      - NUMBER OF ROWS IN ACTUAL DATA MATRIX               ARRAY019
C           J      - NUMBER OF COLUMNS IN ACTUAL DATA MATRIX            ARRAY020
C           N      - NUMBER OF ROWS SPECIFIED FOR THE MATRIX D IN       ARRAY021
C                    DIMENSION STATEMENT                                ARRAY022
C           M      - NUMBER OF COLUMNS SPECIFIED FOR THE MATRIX D IN    ARRAY023
C                    DIMENSION STATEMENT                                ARRAY024
C           S      - IF MODE#1, THIS VECTOR CONTAINS, AS INPUT, A DATA  ARRAY025
C                    MATRIX OF SIZE I BY J IN CONSECUTIVE LOCATIONS     ARRAY026
C                    COLUMN-WISE.  IF MODE#2, IT CONTAINS A DATA MATRIX ARRAY027
C                    OF THE SAME SIZE AS OUTPUT.  THE LENGTH OF VECTOR S ARRAY028
C                    IS IJ, WHERE IJ#I*J.                               ARRAY029
C           D      - IF MODE#1, THIS MATRIX %N BY M< CONTAINS, AS OUTPUT, ARRAY030
C                    A DATA MATRIX OF SIZE I BY J IN FIRST I ROWS AND   ARRAY031
C                    J COLUMNS.  IF MODE#2, IT CONTAINS A DATA MATRIX OF ARRAY032
C                    THE SAME SIZE AS INPUT.                            ARRAY033
C                                                                       ARRAY034
C        REMARKS                                                        ARRAY035
C           VECTOR S CAN BE IN THE SAME LOCATION AS MATRIX D.  VECTOR S ARRAY036
C           IS REFERRED AS A MATRIX IN OTHER SSP ROUTINES, SINCE IT     ARRAY037
C           CONTAINS A DATA MATRIX.                                     ARRAY038
C           THIS SUBROUTINE CONVERTS ONLY GENERAL DATA MATRICES %STORAGEARRAY039
C           MODE OF 0<.                                                 ARRAY040
C                                                                       ARRAY041
C        SUBROUTINES AND FUNCTION SUBROUTINES REQUIRED                  ARRAY042
C           NONE                                                        ARRAY043
C                                                                       ARRAY044
C        METHOD                                                         ARRAY045
C           REFER TO THE DISCUSSION ON VARIABLE DATA SIZE IN THE SECTIONARRAY046
C           DESCRIBING OVERALL RULES FOR USAGE IN THIS MANUAL.          ARRAY047
C                                                                       ARRAY048
C                                                                       ARRAY050
      SUBROUTINE ARRAY %MODE,I,J,N,M,S,D<                               ARRAY051
      DIMENSION S%1<,D%1<                                               ARRAY052
      DOUBLE PRECISION S,D
      NI#N-I                                                            ARRAY053
C                                                                       ARRAY054
```

```
C         TEST TYPE CF CCNVERSION                                   ARRAY056
C                                                                   ARRAY057
      IF%MODE-1< 100, 100, 120                                      ARRAY058
C                                                                   ARRAY059
C        CONVERT FROM SINGLE TO DOUBLE DIMENSION                    ARRAY060
C                                                                   ARRAY061
  100 IJ#I*J&1                                                      ARRAY062
      NM#N*J&1                                                      ARRAY063
      DO 110 K#1,J                                                  ARRAY064
      NM#NM-NI                                                      ARRAY065
      DO 110 L#1,I                                                  ARRAY066
      IJ#IJ-1                                                       ARRAY067
      NM#NM-1                                                       ARRAY068
  110 D%NM<#S%IJ<                                                   ARRAY069
      GO TO 140                                                     ARRAY070
C                                                                   ARRAY071
C        CONVERT FROM DOUBLE TO SINGLE DIMENSION                    ARRAY072
C                                                                   ARRAY073
  120 IJ#0                                                          ARRAY074
      NM#0                                                          ARRAY075
      DO 130 K#1,J                                                  ARRAY076
      DO 125 L#1,I                                                  ARRAY077
      IJ#IJ&1                                                       ARRAY078
      NM#NM&1                                                       ARRAY079
  125 S%IJ<#D%NM<                                                   ARRAY080
  130 NM#NM&NI                                                      ARRAY081
C                                                                   ARRAY082
  140 RETURN                                                        ARRAY083
      END                                                           ARRAY084
```

```
//       EXEC FORTRAN
C        SUBROUTINE LOC                                             LOC 004
C                                                                   LOC 005
C        PURPOSE                                                    LOC 006
C           COMPUTE A VECTOR SUBSCRIPT FOR AN ELEMENT IN A MATRIX OF   LOC 007
C           SPECIFIED STORAGE MODE                                  LOC 008
                                                                    LOC 009
C        USAGE                                                      LOC 010
C           CALL LOC %I,J,IR,N,M,MS<                                LOC 011
C                                                                   LOC 012
C        DESCRIPTION OF PARAMETERS                                  LOC 013
C           I   - ROW NUMBER OF ELEMENT                             LOC 014
C           J   - COLUMN NUMBER OF ELEMENT                          LOC 015
C           IR  - RESULTANT VECTOR SUBSCRIPT                        LOC 016
C           N   - NUMBER OF ROWS IN MATRIX                          LOC 017
C           M   - NUMBER OF COLUMNS IN MATRIX                       LOC 018
C           MS  - ONE DIGIT NUMBER FOR STORAGE MODE OF MATRIX       LOC 019
C                  0 - GENERAL                                      LOC 020
C                  1 - SYMMETRIC                                    LOC 021
C                  2 - DIAGONAL                                     LOC 022
C                                                                   LOC 023
C        REMARKS                                                    LOC 024
C           NONE                                                    LOC 025
C                                                                   LOC 026
C        SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED              LOC 027
C           NONE                                                    LOC 028
C                                                                   LOC 029
C        METHOD                                                     LOC 030
C           MS#0   SUBSCRIPT IS COMPUTED FOR A MATRIX WITH N*M ELEMENTS LOC 031
C                  IN STORAGE %GENERAL MATRIX<                      LOC 032
C           MS#1   SUBSCRIPT IS COMPUTED FOR A MATRIX WITH N*%N&1</2 IN LOC 033
C                  STORAGE %UPPER TRIANGLE OF SYMMETRIC MATRIX<. IF  LOC 034
C                  ELEMENT IS IN LOWER TRIANGULAR PORTION, SUBSCRIPT IS LOC 035
C                  CORRESPONDING ELEMENT IN UPPER TRIANGLE.         LOC 036
C           MS#2   SUBSCRIPT IS COMPUTED FOR A MATRIX WITH N ELEMENTS  LOC 037
                   IN STORAGE %DIAGONAL ELEMENTS OF DIAGONAL MATRIX<.  LOC 038
```

```
C                  IF ELEMENT IS NOT ON DIAGONAL %AND THEREFORE NOT IN  LOC  039
C                  STORAGE<, IR IS SET TO ZERO.                         LOC  040
C                                                                       LOC  041
C     ..................................................................LOC  042

//       EXEC  FORTRAN
      SUBROUTINE READ (A,M)                                             EA2
      DIMENSION A(10,10)                                                FA230
      DOUBLE PRECISION A                                                EA230
 1    FORMAT (4F18.12)
      DO 10 I=1,M                                                       EA230
 10   READ(1,1) (A(I,J), J=1,M)                                         EA230
C     READ  A  MATRIX  WITH  M  ROWS  AND  M  COLS  BY  ROWS            EA230
      RETURN                                                            EA230
      END                                                               EA230
```

# APPENDIX IX

## CONVERGENCE OF THE LINEAR SYSTEM

We are given a linear system described by equation

$$\dot{Y} = AY + AM$$

and cost functional

$$C = \int_0^\infty (Y^tY + M^tM)dt,$$

with optimal controller described by

$$P = KY$$

$$M = -B^tP,$$

where it is known that the auxiliary variables P must satisfy

$$\dot{P} = -A^tP - Y, \quad P(\infty) = 0.$$

It is desired to investigate the convergence of the rule set forth in Section II for adjusting the computed values of $\mathcal{A}$, $\beta$, and $\mathcal{K}$, which are estimates of A, B, and K.

The adjustment rules are as follows.

Let

$$\dot{Y} - \mathcal{A}Y - \beta M = \mathcal{E}_1$$

and

$$\mathcal{K}\dot{Y} + \mathcal{A}^tP + Y = \mathcal{E}_2.$$

We note that

$$\frac{\partial|\mathcal{E}_i|}{\partial a_{ij}} = \frac{\partial \mathcal{A}Y}{\partial a_{ij}} \quad \text{sgn} \quad \left[\mathcal{E}_{1i}\right] = y_j \quad \text{sgn} \quad \left[\mathcal{E}_{1i}\right]$$

and that similar results obtain for the other variables.

It appears that an appropriate gradient method might be to set

$$\dot{a}_{ij} = -ry_i \quad \text{sgn} \quad \mathcal{E}_1, \quad r > 0,$$

since such a form would seem to tend to drive each of the entries in the matrices $\mathcal{A}$, $\beta$, and $\mathcal{K}$ toward their respective values in A, B, and K. To investigate whether such is actually the case, let us state formally the following expression.

$$\dot{Y} - \mathcal{A}Y - \beta M - \dot{Y} + AY + AM = \mathcal{E}_1$$

Rearrangement yields that

$$(A - \alpha) Y + (B - \beta) M = \mathcal{E}_1 .$$

From the fact that $M = -\beta^t \mathcal{X} Y$ in the operating system, one can obtain that

$$(A - \alpha) + (B - \beta) \beta^t \mathcal{X} Y = \mathcal{E}_1$$

It is going to be shown that the magnitude of $\mathcal{E}_1$ as given by the above equation almost satisfies the requirements for a Liapunov function for the parameter adjustment system, and that some additional peculiarities tend to produce convergence along somewhat unusual trajectories. Note first that various values of Y are presented to the system from time to time, and presumably span the $\Omega_Y$. Note then that $\mathcal{E}_1$ is continuous and has continuous derivatives everywhere except where $\mathcal{E}_1$ is zero. The discontinuity in the derivatives can be removed by postulating that in every region close to $\mathcal{E}_1 = 0$ the function $\mathcal{E}_1$ is replaced by an appropriate "rounded" function that satisfies the necessary continuity conditions $\mathcal{E}_1$ on the boundaries of the region. This rounded region can be arbitrarily small, and in the limit clearly goes to $\mathcal{E}_1$. Thus $\mathcal{E}_1$ satisfies the continuity requirement of a Liapunov function in the limit. It does not satisfy the requirement that $\mathcal{E}_1 = 0$ only at the point $A = \alpha$, $B = \beta$. Instead we see that

$$(A - \alpha) + (B - \beta) \beta^t \mathcal{X} = 0$$

dictates only that

$$\alpha = A + (B - \beta) \beta^t \mathcal{X} \qquad (68)$$

i.e., for each $\mathcal{X}$, A and B, $\alpha$ is a quadratic function of $\beta$.

Suppose, however, that $\mathcal{X}$ takes on various values from time to time. For a sufficiently wide range of values $\mathcal{X}$, $A = \alpha$ and $B = \beta$ are clearly the only values of $\alpha$ and $\beta$ that will always satisfy Equation 68. Examination of the one-to-one case (see Figure 60 in Appendix VII) yields some insight into the situation. One sees in this figure that two values of $\mathcal{X}$ give two $\mathcal{E}_1 = 0$ curves, having common points at $\alpha = a$, $\beta = b$, and $\beta = 0$. If the values of $\alpha$ and $\beta$ are driven along a trajectory that uses the gradient method previously discussed and that alternates between the two values of $\mathcal{X}$, the system will converge for all initial values of $\alpha$ and $\beta$ in the right-half plane, excluding the $\beta$-axis. This follows from the fact that for $\mathcal{X} \neq 0$, the terms $(A - \alpha)$ and $(B - \beta)$ in Equation 68 ensure that a component of the motions of $\alpha$ and $\beta$ will be in the directions of a and b at each step in the alternation procedure. The fact can also be confirmed readily by systematically plotting vectors corresponding to

$$\dot{a} = -ky \ \mathrm{sgn} \ \mathcal{E}_1$$

$$\dot{b} = -kbky \ \mathrm{sgn} \ \mathcal{E}_1$$

in the various regions of the figure. If the above two-dimensional argument holds for larger systems, they too will converge. Although of considerable interest, this problem was not pursued, since it was not directly pertinent to the main argument. The various systems that were simulated did converge, which was considered adequate for the present purpose.

# APPENDIX X

## LEARNING OF THE EXCLUSIVE-OR FUNCTION IN RATS

The hierarchy implementation discussed in Section III offers an explanation of interference phenomena that are observed in psychological experimentation.

An experiment was performed at Goodyear Aerospace with company funds to test whether observed behavior would match behavior predicted from a hierarchy method for generating the state identification function. The results were quite satisfactory. They are included here to demonstrate the agreement of the algebraic structure with experimental results.

The theoretical conclusions concerning adjustment of hierarchies of threshold elements discussed in the text were tested by an animal experiment. The authors were assisted in conducting the experiment by Messrs. R. H. Kause and J. R. Davis (Reference 31).

The subjects (Ss), 12 albino rats from three to five months of age, had not been used in any previous experiments. The apparatus used was a single choice-point Y-maze (Figure 74). Guillotine doors, which are visible on the figure, separated the various sections of the maze. Dimensions of the maze are shown on the photograph. Various portions of the maze were painted white, gray, and black. Two 3v flashlight bulbs at the choice-point were operated by toggle switches beside the start box.

Prior to experimentation, Ss were reduced to 80 percent ad libitum body weight by 48-hour food deprivation followed by limited daily feedings for 10 days (Purina lab chow). After weight reduction the Ss ranged from 256 to 326 grams. Weights were held to within ±1 to 3 grams of the calculated 20 percent weight reduction during the experiment.

Ss were familiarized with the maze situation for 15 minutes each on each of two successive days before training. On the first day, all guillotine doors were removed, and Ss were placed in the start box. Since most Ss went to the black arm of the maze, on the second day they were placed in the white goal box.

The Ss were divided into two squads of six animals each. All received 15 training trials per day for twenty days. Rewards were four food pellets. To counterbalance the inherent side preference, the reward locations were reversed between the two squads; as shown later:

| | Reward Locations | |
|---|---|---|
| | Squad 1 | Squad 2 |
| One Light | L | R |
| 0, Two Lights | R | L |

During each trial, subjects were kept in the starting box for 15 seconds and allowed to remain in the goal box for 30 seconds or until food was consumed, whichever was longer. Upon completion of each trial Ss were returned to their cages, where water was available. Approximately 25 to 30 minutes separated the trials for each S. After the 20-day training period, Ss were randomly divided into two groups. Both groups were run an additional eight days. Group 1 received 50 percent reinforcement of correct responses; Group 2 received no reinforcements. After the eight-day interval, Group 1 was run an additional eight days with no reinforcement.

The histories of percentages of correct responses to each discriminable light condition are shown in Figure 75, together with theoretical curve fits. The interference phenomenon predicted in Section III for a hierarchy is clearly seen on the figure.

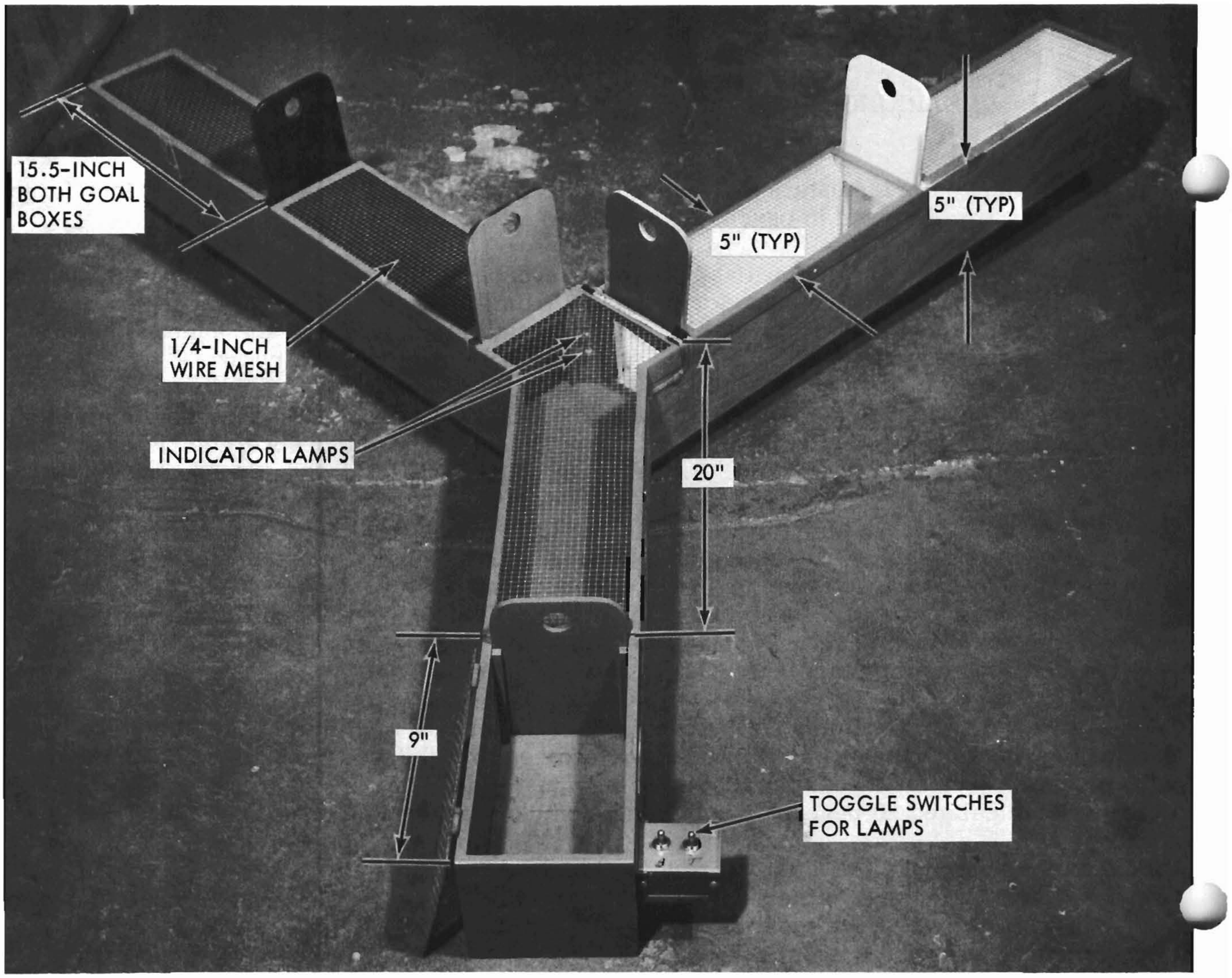


Figure 74. Single-Choice Point Y-Maze for Testing Learning of the Exclusive-Or Function

If one assumes that the strength of an S-R connection is proportional to the probability of the response, and that the rate of change of the strength is proportional to the percentage of responses that are mistakes, one obtains an exponential described by

$$\dot{p} = k_1 (p - p). \quad p(0) = 0.5$$

where $\dot{p}$ is the asymptote, p is the instantaneous probability of correct response on the one-light case, and $k_1$ is the inverse time constant. For the 0, two-light case one can write that

$$\dot{q} = k_2 (q - q) - f k_1 (p - p). \quad q(0) = 0.5$$

where $k_2$ is the inverse time constant, f is a positive constant, and q is the asymptote. The second term of the right-hand member expresses the intuitive notion that changes in the one-light probability are reflected as interference with the 0, two-light probability. One would predict intuitively that the value of f might be 2, since twice as many inputs are present for the two-light case, and that $k_2$ might be about the same as $k_1$.

A standard non-linear regression computer program was used for curve fitting. Values obtained were

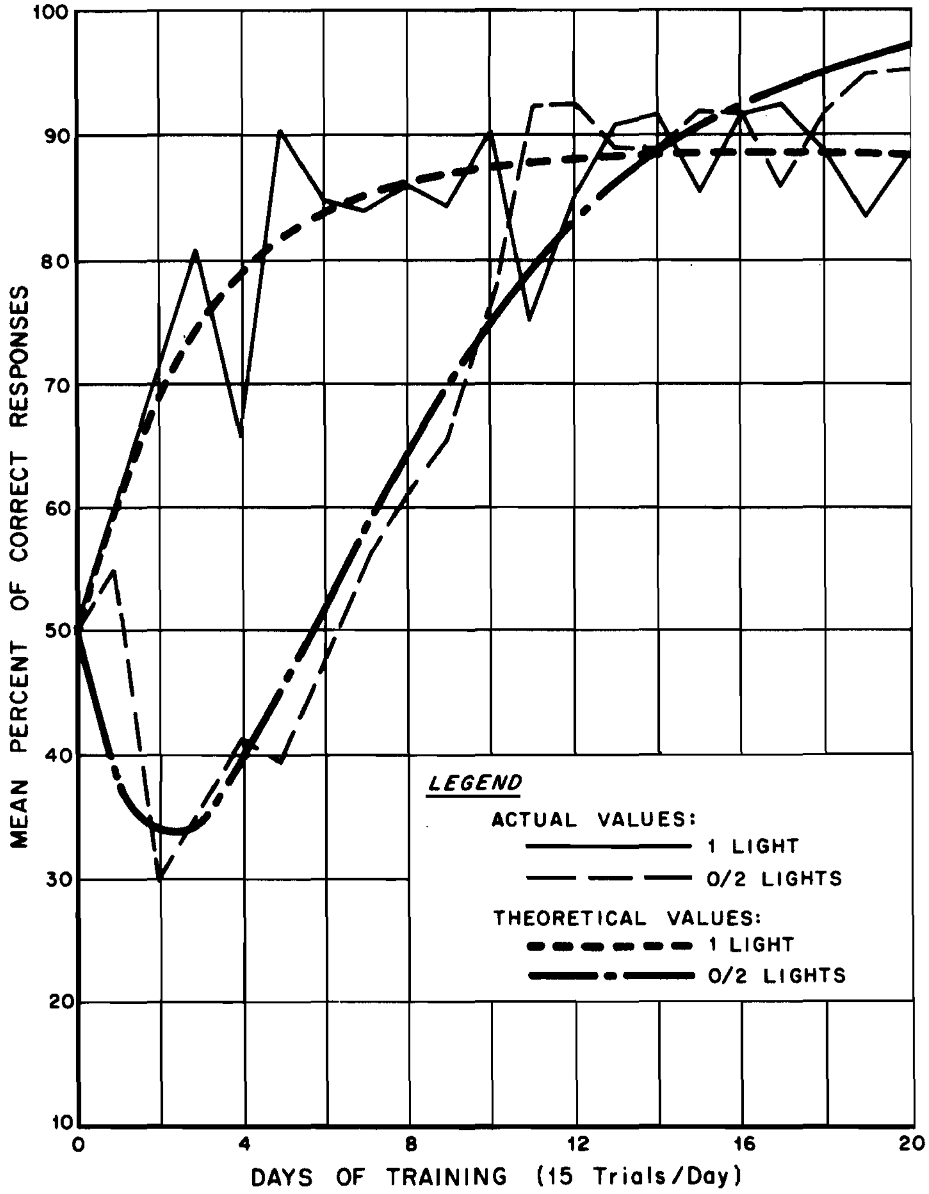


Figure 75. History of Percentages of Correct Responses versus Theoretical Curves

$p = 0.88$<br>
$k_1 = 0.35$/day<br>
$q = 1.009$<br>
$k_2 = 0.205$<br>
$f = 2.18$

The asymptote q is clearly slightly in error, since 1.0 is its maximum possible value. However, if the data had been extended beyond 20 days with no change in reinforcement schedule, the computed value of q would have been more realistic.

The experimental results support the hypothesis that a decision-making hierarchy of at least two levels models the discrimination learning task of the experiment. Further experimentation with logic functions requiring three or more separating planes could determine whether the hierarchy instrumentation scheme is a satisfactory model for more sophisticated tasks.

# APPENDIX XI

## MARKOV PROCESSES

The discussion presented here shows ways in which a cost functional can be defined as an expectation for Markov processes. Although the implications of the derivation have not been completely investigated, a hand-computed example indicates that further research along these lines might be fruitful.

Consider a system that has a finite number of states, $z_1, z_2 \ldots, z_n$. At any instant in time it occupies one (and only one) state. There exists a vector for each $z_i$, the components of which are in order the conditional probabilities that at the next instant of time the system will occupy any particular state of the n possible states. The system is described by an n x n matrix, $P = p_{ij}$ where $p_{ij}$ is the conditional probability that the system will occupy state $z_i$ at time t + 1, given the fact that the system is in state $z_j$ at time t (Reference 32).

From the above description the following formulation can be written. Assume that the respective probabilities that the system is in one or another of its possible states at time t be represented by a vector

$$Z = \mathrm{col}(p(z_1),\ p(z_2),\ \ldots,\ p(z_n)).$$

From Bayes theorem,

$$\mathcal{P}(b) = \mathcal{P}(b|a_1) + \mathcal{P}(b\ a_2) + \ldots + \mathcal{P}(b|a_n)\ ,$$

one can write that

$$Z(t+1) = P \cdot Z(t)\ .$$

Such systems have been extensively studied. The above system is free, i.e., it receives no interference from an outside source, $v_1, v_2, \ldots, v_m$, which alter the probabilities that various transitions will occur, i.e., $P_1, P_2, \ldots, P_m$, such that

$$(P_k)_{ij} = (\text{state i follows state j} \mid \text{input k is present}).$$

If the probabilities of the various inputs occurring are known, such a system can be reduced to a free Markov process by the fact that

$$p_{ij} = (P_1)_{ij}(v_1|z_j) + (P_2)_{ij}(v_2|z_j) + \ldots + (P_m)_{ij}(v_m|z_j)\ .$$

We wish to show that if costs are associated with the various transitions or with occupying certain states, then optimal input sequences are defined under some circumstances.

We consider first a free Markov process. It is well-known that the probability that the process will occupy a certain given state after a very long time is given by the solution to setting P(t + 1) = P(t), yielding,

$$Z_1(I - P) = 0$$

where $\langle Z_1, Z_1 \rangle$ is normalized to 1. This eigenvector for the unity eigenvalue of P gives, if unique, the probabilities of occupying any state after a long time, starting from any state. If the eigenvector is not unique, the probabilities after a long time are dependent on the starting state.

Assume now there is a cost,

$$D = \mathrm{col}(d_{11}, d_{12}, \ldots, d_{1n}, d_{21}, \ldots, d_{nn}) ,$$

associated with each transition between states. Then the expected cost at each instant of time s

$$\phi_1 = \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}(P_k)_{ij} P(y_k z_j) P(z_i) .$$

Minimization of $\phi$ yields a schedule of probabilities $P(y_k z_i)$ which will produce the lowest expected cost at each instant in time. To the authors' knowledge, no general computational rule exists for minimization of such an expression, other than exhaustive computation.

By analogy with some results of Game theory, it seems likely that the probabilities $P(y_k|z_i)$ would tend to become selections, in that for each state $z_i$, some certain $y_k$ would usually be the "best" response and its probability should be maximized. As an alternative to the above formulation, one can assume that costs $C = \mathrm{col}(c_1, c_2, \ldots, c_n)$, are associated with occupying respectively each of the states, and that transitions do not contribute to costs. The expected cost at each instant is then

$$\phi_2 = \langle C, Z_1 \rangle$$

and symbol probabilities can be chosen to minimize $\phi$. Both these formulations lead naturally to the concept that the "desired" state is simply the one that has the highest probability of occupancy when $\phi$ is minimized.

In a third possible formulation, costs could be assigned both to state occupancy and to transitions. It is immediately apparent that one should minimize

$$\phi = \phi_1 + \phi_2$$

by appropriate selection of the input symbol probabilities.

None of these formulations has been studied in any detail in this research. Some relatively crude hand-computed examples, which are discussed next, indicate that the general results obtained for the deterministic case are still applicable.

In the discussion of specific methods of instrumenting "learning" procedures presented in Section III, one idea that is used is that the controller can "explore" the environmental behavior more or less randomly, and "learn" by changing the conditional probabilities of its responses.

The major result of the linear investigations presented earlier was to illustrate the necessity for the mappings

$$\psi : S \times D \longrightarrow R$$

$$I : S^n \longrightarrow D$$

to be non-trivial in D. The usual picture given in behavior theory texts is one of stimuli mapping directly to responses. Although motivation is mentioned, its role is not clear. The following paragraphs outline how the concepts might be extended to Markov processes.

We assume that a process to be controlled is a finite state Markov process described by a matrix B. The elements $b_{ij}$ are respectively the probabilities that state i will follow state j.

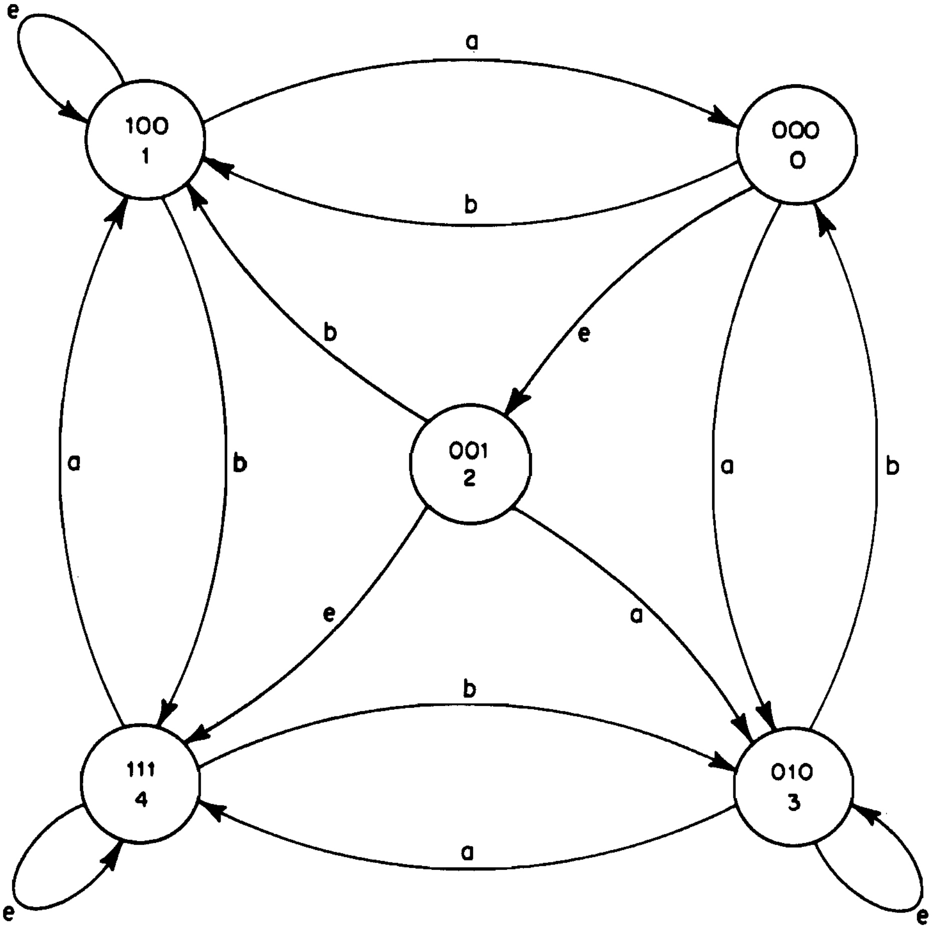


Figure 76. Example of Markov Processes

The controller is able to alter certain of the elements $b_{ij}$. It is desired to find those values of the $b_{ij}$ that cause the process to spend the maximum percentage of the time in one particular state, selected a priori. The specific example discussed below illustrates why the empirical rules that were first tested failed, and why an approach based on the methods of the present analytical model promise utility. A process (Figure 76) is described by the matrix:

$$B = \begin{bmatrix} x_1 & a_2 & 0 & b_4 & 0 \\ b_1 & x_2+e & b_3 & 0 & a_5 \\ e & 0 & x_3 & 0 & 0 \\ a_1 & 0 & a_3 & e+x_4 & b_5 \\ 0 & b_2 & e & a_4 & e+x_5 \end{bmatrix}$$

where $a_i$ and $b_i$ are the probabilities that inputs a and b will occur in state i, and e is the probability that the environment will make a "spontaneous" transition. The $x_i$'s are chosen so as to make each column add to 1.

Assuming that the probabilities of each of the transitions between states on the figure are 0.1, we obtain that

$$B = \begin{bmatrix} 0.7 & 0.1 & 0 & 0.0 & 0 \\ 0.1 & 0.8 & 0.1 & 0 & 0.1 \\ 0.1 & 0 & 0.7 & 0.1 & 0 \\ 0.1 & 0 & 0.1 & 0.8 & 0.1 \\ 0 & 0.1 & 0.1 & 0.1 & 0.8 \end{bmatrix}$$

The eigenvector corresponding to the unity eigenvalue is computed to be

$$S = \begin{bmatrix} 1/6 \\ 1/4 \\ 1/18 \\ 1/4 \\ 5/18 \end{bmatrix} \cong \begin{bmatrix} 0.19 \\ 0.25 \\ 0.05 \\ 0.25 \\ 0.28 \end{bmatrix}$$

It was shown previously that this eigenvector (normalized) is the vector of the probabilities that after a long time the process will be in each of the states.

In the empirical model discussed previously, the weights associated with the various input to an element and its particular output were to become more positive if past history indicated that production of that output tended to remove the inputs. If,on the other hand, production of an output tends to increase the input, the input weight would tend to become negative. Let $\alpha$, $\beta$, and $\gamma$ be the signals from the process that are inputs to the controller. One can list what effect each of the two controller outputs will have on each of the three process signals, starting in each of the five states, in tabular form (Table II.) A + 1 corresponds to increasing the signal, a -1corresponds to decreasing it.

Table II. Changes in Inputs Caused by Various Control Signals in Various States of the Markov Process in Figure 76.

| State | Control | $\alpha$ | $\beta$ | $\gamma$ |
|---|---|---|---|---|
| 0 | a<br>b | 0<br>1 | 1<br>0 | 0<br>0 |
| 1 | a<br>b | -1<br>0 | 0<br>1 | 0<br>1 |
| 2 | a<br>b | 0<br>1 | 1<br>0 | -1<br>-1 |
| 3 | a<br>b | 1<br>0 | 0<br>-1 | 1<br>0 |
| 4 | a<br>b | 0<br>-1 | -1<br>0 | -1<br>-1 |

If each of these changes is multiplied by the probability of occupying the particular state in which it occurs and the values are summed, we obtain

| Control | $\alpha$ | $\beta$ | $\gamma$ |
|---|---|---|---|
| a | 0 | -1/18 | -1/12 |
| b | -1/18 | 0 | 1/12 |

This last matrix is clearly proportional to the long-term determination as to whether a weight should be increased or decreased from its initial value by the empirical rule.

An illustration of the flaw in this empirical rule is given by the following crude example of such a rule. Assume that as a result of such observations over a period of time, some mechanisms alter the probabilities of occurrences of the various transitions so as to increase the probabilities of those transitions that usually lead to a reduction in the input signals (and also to decrease the probabilities of those transitions that cause increases in the input signals). We assume as a crude approximation that those transitions that show correlations of -1/18 have their probabilities increased by 0.2 (from 0.1 to 0.3), and that those showing the correlations of -1/12 have their probabilities increased by 0.3 to 0.4. If multiple correlations are present, the transition probability is increased by the sum of all the indicated transition probability changes. The new matrix of the Markov process becomes

$$B' = \begin{bmatrix} 0.7 & 0.1 & 0 & 0.1 & 0 \\ 0.1 & 0.6 & 0.4 & 0 & 0.5 \\ 0.1 & 0 & 0.1 & 0 & 0 \\ 0.1 & 0 & 0.4 & 0.6 & 0.5 \\ 0 & 0.3 & 0.1 & 0.3 & 0 \end{bmatrix}$$

with eigenvector

$$S' = \frac{1}{904} \begin{bmatrix} 180 \\ 270 \\ 20 \\ 270 \\ 164 \end{bmatrix}$$

We note that the probability of occupying the 3-state is considerably reduced, but that the probability of occupying the 0-state is not significantly increased. Further, the changes in transition probabilities that might result from the new correlation matrix will not significantly alter this qualitative result.

Simple computation of weights of stimuli on responses does not appear from this example to be an effective method for generating appropriate trajectories.

Consider now a scheme which includes the idea of a "drive reduction," where the drive level in any process state is the number of 1's in the output of the process when in that state. Table III lists the "drive reduction" produced by each of the possible outputs when in each of the possible states, and whether any particular input is present in that state.

Table III. Drive Reduction

| State | Control | Drive Change | Signals Present in that State |
|---|---|---|---|
| 0 | a<br>b | +1<br>+1 | 000 |
| 1 | a<br>b | -1<br>+2 | 100 |
| 2 | a<br>b | 0<br>0 | 001 |
| 3 | a<br>b | +2<br>-1 | 010 |
| 4 | a<br>b | -2<br>-2 | 111 |

One can first multiply each of these possible "drive changes" by the probability of that state occurring, and can then sum the drive changes which will be seen when each input is present, yielding

| Control | $\alpha$ | $\beta$ | $\gamma$ |
|---|---|---|---|
| a | -29/36 | -1/18 | -5/9 |
| b | - 1/18 | -29/36 | -5/9 |

Note the drastic change produced in the 11 and 22 terms by this method of computation. Examination of the sequential machine shows that an acceptable logic function for generating optimal trajectories is

$$a = \alpha \vee \gamma$$

$$b = \beta \vee \gamma$$

where the ambiguity (both a and b should appear) in state 4 is resolved either way.

# REFERENCES

1. Study of Distributed Adaptation, GAP 1932, Goodyear Aerospace Corporation, Akron, Ohio, February, 1963.

2. Harmon, L.D., "Studies with Artificial Neurons, I: Properties and Functions of an Artificial Neuron," Kybernetik 1, No. 3, 89-101, 1961.

3. Merriam, C.W., Optimization Theory and the Design of Feedback Control Systems, McGraw-Hill Book Co., New York, 1964.

4. Halkin, H., A Maximum Principle of the Pontryagin Type for Systems Described by Nonlinear Difference Equations, First International Conference on Programming and Control, USAF Academy, Colorado Springs, Colorado, 15 April 1965.

5. Nelson, R. J., Introduction to Automata, Case Institute of Technology, Cleveland, Ohio, 1964.

6. Dale, H., "Pharmacology and Nerve Endings," Walter Ernest Nixon memorial lecture, Procedures of the Royal Society of Medicine, Vol. 28, January 1935, pp. 319-332.

7. Goffman, C., Real Functions, Holt, Rinehart, and Winston, New York, 1963.

8. Windeknicht, T. G., Concerning an Algebraic Theory of Systems, Report SRC 78-A-65-30, Case Institute of Technology, Cleveland, Ohio.

9. Birta, L. G., A Formal Approach to Concepts of Interaction, Report SRC 81-4-65-31, Case Institute of Technology, Cleveland, Ohio, 1965.

10. Hebb, D. O., The Organization of Behavior, John Wiley and Sons, New York; 1949.

11. Milner, P. M., "The Cell Assembly: Mark II," Psychological Review, 64, July 1957, pp. 242-252.

12. Thorndike, E. L., Educational Psychology, Lemcke and Buechner, New York, 1963.

13. Hilgard, Ernest R., Theories of Learning, second edition, Appleton-Century Crofts, Inc., New York, 1956.

14. Dollard, J. and Miller, N. E., Social Learning and Imitation, Yale University Press, New Haven, Conn., 1941.

15. Spence, K. W., Behavior Theory and Conditioning, Yale University Press, New Haven, Conn., 1956.

16. Aristotle, Ethics, Book I.

17. Tinklepaugh, O. L., "An Experimental Study of Representative Factors in Monkeys," Journal of Comp. Psychology 8, 1928, pp. 197-236.

18. Guthrie, E.R., The Psychology of Learning, Harper & Brothers, New York, 1935, p. 26.

19. Voeks, V.W. "Formalization and Clarification of a Theory of Learning," Journal of Psychology 30, 1950, pp. 341-362.

20. Skinner, B. F., Science and Human Behavior, The Macmillan Company, New York, 1953, pp. 144-146.

21. Wiener, Norbert, Cybernetics, Second Edition, Massachusetts Institute of Technology Press, 1961.

22. Von Neumann, J. and Morgenstern, O., Theory of Games and Economic Behavior, revised edition, Princeton University Press, 1947.

23. Flood, M.M. "On Game-Learning Theory and Some Decision-Making Experiments," Decision Processes, R.M. Thrall, C.H. Coombs, and D. L. Davis, editors; John Wiley & Sons, New York, 1954.

24. Hovland, C. I., "A Communication Analysis of Concept Learning," Psychological Review 59, 1952, pp 461-472.

25. Hull, C. L., A Behavior System: An Introduction to Behavior Theory Concerning the Individual Organism, Yale University Press, New Haven, Conn., 1952.

26. McCulloch, W.S. and Pitts, W., "A Logical Calculus of the Ideas Immanent in Nervous Activity," Bulletin of Math. Biophysics 5, 1943.

27. Rosenblatt, F., Principles of Neurodynamics, Spartan Press, 1962.

28. Widrow, B., An Adaptive "Adaline" Neuron Using Chemical "Memistors," TR No. 1553-2, Stanford Electronics Labs, Stanford, Calif., 17 October 1960.

29. Gilstap, L. O., Jr. and Lee, R. J., "Learning Machines," Bionics Symposium, USAF WADD Technical Report 60-600, 1960, pp. 437-450.

30. Fein, L., "The Structure and Character of Useful Information Processing Systems," Simulation 5, No. 6, December 1965.

31. Griffith, V.V., Kause, R.H.; and Davis, J. A.; "Learning of the Exclusive-Or Function in Rats," Bionics Symposium, Wright-Patterson AFB, Ohio (to be published).

32. Rosenblatt, M., Random Processes, Oxford University Press, New York, 1962.

Security Classification

## DOCUMENT CONTROL DATA - R&D

*(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)*

| | |
|---|---|
| 1. ORIGINATING ACTIVITY *(Corporate author)*<br>Avionics and Electronics Research and Development Division, Goodyear Aerospace Corporation<br>Akron, Ohio | 2a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED<br>2b. GROUP<br>N/A |

3. REPORT TITLE

FURTHER STUDIES ON
DISTRIBUTED ADAPTATION IN NEUROMINE NETWORKS

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Final Report, June 1964-May 1967

5. AUTHOR(S) *(Last name, first name, initial)*

Griffith, V. V. Wuerthele, W. E.
Bolen, G. H.

| | | |
|---|---|---|
| 6. REPORT DATE<br>August 1967 | 7a. TOTAL NO. OF PAGES<br>132 | 7b. NO. OF REFS<br>32 |

| | |
|---|---|
| 8a. CONTRACT OR GRANT NO.<br>AF 33 (615)-1891<br>b. PROJECT NO. 7232<br>c. Task No. 723203<br>d. | 9a. ORIGINATOR'S REPORT NUMBER(S)<br>9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*<br>AMRL-TR-67-121 |

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

| | |
|---|---|
| 11. SUPPLEMENTARY NOTES | 12. SPONSORING MILITARY ACTIVITY<br>Aerospace Medical Research Laboratories<br>Aerospace Medical Div., Air Force Systems Command, Wright-Patterson AFB, Ohio 45433 |

13. ABSTRACT

This report describes investigations of networks with adaptive ability distributed through them. It is thought that large-scale adaptive systems can be constructed of adaptive building blocks. These adaptive systems would be flexible in function, reliable and would resist severe damage characteristics of living creatures. Neuron models were tested by interconnecting them into various networks to perform simple control tasks. The test results were evaluated and the evaluation used to improve the theory and the neuron model. The distributed adaption concept was analyzed from an abstract algebraic approach, using optimal control theory. The combined approach, when studied in depth, contributed to the understanding of the problem. Although the conclusions of this report are at best tentative, one conclusion seems reasonably valid: any required adaptive controller can be built using iterative elements provided only that all terminal segments of optimal trajectories of the process are themselves optimal trajectories, and that the process in controllable and observable.

Security Classification

| 14. KEY WORDS | LINK A | | LINK B | | LINK C | |
|---|---|---|---|---|---|---|
| | ROLE | WT | ROLE | WT | ROLE | WT |
| Bionics | | | | | | |
| Neuromime Net | | | | | | |
| Adaptive Controller | | | | | | |
| Distributed Adaptation | | | | | | |
| Optimal Control | | | | | | |

Security Classification