Computatio 1 (1964), 57—71

WŁADYSŁAW TURSKI

A LEARNING AUTOMATON FOR SOLVING STABILITY PROBLEMS OF DIFFERENTIAL EQUATIONS

РЕЗЮМЕ

В статье описывается алгорифм приближённого нахождения области практической устойчивости динамической системе поднерженной случайным возмущениям некоторого типа. Алгорифм основан на принципе самообучения вычислительной машины распознаванию образов.

1. Statement of the problem.

Consider a dynamic system whose behaviour is mathematically represented by a solution of the set of S normal differential equations

$$\frac{dx_k}{dt} = f_k(x_1, x_2, \ldots, x_S; t) \quad (k = 1,2,\ldots,S) \tag{1.1'}$$

or, more concisely

$$x' = f(x;t), \tag{1.1''}$$

where, and further on in this paper, it is to be understood that x and f are vectors: $x = (x_1, x_2, \ldots, x_S)$; $f = (f_1, f_2, \ldots, f_S)$

Under the well-known assumptions concerning the behaviour of functions f there exists certain region **R** with the property that for any initial conditions belonging to this region there exists unique and continuous solution of eq. (1.1'').

From all normal differential equations we single out the equations defined by the following additional property of functions f,

$$f(0;t) = 0 \tag{1.2}$$

In addition we suppose that initial conditions

$$x(0) = 0 \tag{1.3}$$

belong to the region **R**.

We shall henceforth assume, that the function $f(x;t) \not\equiv 0$ in the region **R**, except along the $(0,t)$ axis. Such equations will be denoted by

$$x' = F(x;t) \tag{1.1}$$

Solutions of eq. (1.1) with initial conditions (1.3) are

$$x(t) = 0 \qquad 0 \leqslant t \leqslant \tau, \tag{1.4}$$

where constant τ depends on particular form of F, (eventually $\tau = \infty$). Solutions (1.4) will be called the equilibrial solutions.

It should be observed that quite a large class of differential equations may be reduced to the form (1.1) admitting equilibrial solution. Indeed, any equations describing deviations of a process from a prescribed behaviour of the system, due to badly controlled initial conditions, are of the form (1.1) and admit equilibrial solution, which in such cases represents just the prescribed behaviour of the system.

If the deviations from the prescribed behaviour are due to some unexpected or ignored factors as well as to badly controlled initial conditions, the process is described by solution of equation

$$x' = F(x;t) + p(x;t), \tag{1.5}$$

where $x' = F(x;t)$ admits as before an equilibrial solution, and exact form of $p(x;t)$ — the perturbing function — is not known. We shall, tentatively, assume that $|p(x;t)| \leqslant \min(|F(x;t)|, \delta)$ in the region **R**. The non-negative constant δ may be supposed to be small because of its physical meaning. The equilibrial solution (1.4) is said to be practically stable if there exists such S — dimensional region $\mathbf{Q}_0$, entirely embedded in **R** and containing the point $x = 0$ that if initial conditions

$$x(0) = x^0 \tag{1.6}$$

belong to $\mathbf{Q}_0$ the solution of equation (1.5)

$$x = (x^0;t) \tag{1.7}$$

satisfies condition

$$x \in \mathbf{Q}_1 \qquad (0 \leqslant t \leqslant T < \tau), \qquad \mathbf{Q}_0 \subset \mathbf{Q}_1. \tag{1.8}$$

in remaining parts of this paper we shall say that the region $\mathbf{Q}_0$ exists if, and only if, there exists a positive Integer ε such that if $|x| < \varepsilon$ then $x \in \mathbf{Q}_0$; this excludes the trivial case $\{x = 0\} = \mathbf{Q}_0$
The region $\mathbf{Q}_0$ represents all admissible initial conditions that give rise to solutions not leaving in a finite time the region $\mathbf{Q}_1$. The latter region may be called the region with satisfactory behaviour of the system. It is well worthwhile to observe that $\mathbf{Q}_0$ depends on choice of $\mathbf{Q}_1$ and T; this choice being determined by considerations involving practical meaning of the solution and character of the system.

If the so-called Ljapunov function for equation (1.5) were known, the problem of determination of $\mathbf{Q}_0$ for given $\mathbf{Q}_1$ and T would be solved by an elegant analytical procedure (cf., e.g. [3]) valid even for an infinite T. Unfortunately, there is no general process for determining the Ljapunov function, and thus the probelm is as difficult as it was.

In this paper we shall restrict ourselves to the following problem. Given equations (1.5), region $\mathbf{Q}_1$, finite constant T, and particular set of initial conditions (1.6). Determine whether condition (1.8) will be satisfied. It seems that great variety of control engineering and management problems can be reduced to thus stated question.

2 Some remarks on numerical solution of equation (1.5).

We shall discuss three principal kinds of function p entering equations (1.5): deterministic, stochastic and random. Function p is here considered as the deterministic one if there exists a precise rule for its evaluation — exact formula, convergent series etc. We call function p stochastic if no such rule is known, but there are some ways for estimating the distribution of values of p which is to be expected if distribution of arguments follows certain pattern. In other words, we know the correlation coefficients between $(x;t)$ and $p(x;t)$. If no such relationship exists, or is not known, we shall call p random. For deterministic p we may hope to find an analytic solution of equations (1.5). For stochastic p there are some techniques that can supply us with "correlated" solution (cf., eg. [4]). It seems, however, that even these techniques may turn out to be of rather limited value for random p. Moreover, when numerical processes are involved in determination of the solution, even the deterministic case becomes random due all sorts of errors unavoidably encountered in using numerical integration procedures.

Thus we suppose that p is random (in the sense defined above). This means that we do not know anything of values of $p(x;t)$, except that, as already mentioned, we assume $|p(x;t)| \leqslant \min(|F(x;t)|, \delta)$. The truncated equation (1.5), i.e. $x = F(x;t)$ may be solved by means of any of the widely used numerical integration procedures. Each of these methods requires that values of F be evaluated for a number of different $(x;t)$. If the obtained values of F are every time summed up with random numbers z, $|z| \leqslant \min(|F(x;t)|, \delta)$ we shall get a curve which will be called „solution" of equation (1.5). Of course it would be precarious to expect that this solution faithfully describes behaviour of the system, but we may hope that sufficiently numerous samples of such trajectories will represent some statistical properties of the system.

It may be worthwile to notice that when performing numerical integration the accuracy aimed at does not need be too great, since if the random errors produced by the integration procedure do not exceed δ, their effect will be masked by additions of z's. Below, we give an example of the "special integration" procedure, based on Runge-Kutta-Gill's method, that may be used to perform the numerical integration of equation (1.5) over one step.

```
procedure special RKG (delta, S, RHF, h, t, X, Y, firststep);
real procedure RHF; real delta, h, t; integer S; array X, Y;
Boolean firststep;
comment This procedure integrates the set of S differential equations x'k = fk(t,x1,x2,... xs) + pk
over time interval [t,t + h] under the assumption that the functions pk are unknown perturbations
pk ≤ min (|fk|, delta). Initial conditions, i.e. xk(t) are represented by array X which on the exit from
the procedure is replaced by approximate values of xk(t + h). Non-local real procedure RHF(i) eva-
luates function fi using as arguments the values of elements of array Y. When computing over many
steps, the Boolean firststep may be set to false on every entry into the procedure, except when the
first step is taken, thus resulting in both better accuracy and some economy of the execution time;
begin
own array Z[0 : S], A, B, C [1 : 4]; array K [0 : S];
real procedure random; begin comment This procedure generates one random number -1 ≤ random
≤ 1; end random; integer i,j; comment repeated is a non-local Boolean, it is to be set to false at the
begining of a programme to which it belongs;
if repeated then go to L1;
A[1] : = C[1] : = C[4] : = 0.5;
A[2] : = C[2] : = 1 - sqrt (0.5);
A[3] : = C[3] : = 1 + sqrt (0.5);
B[1] : = B[4] : = 2;
B[2] : = B[3] : = 1;   A[4] : = 1/6;
repeated : = true;
L1: if firststep then for i : = 0 step 1 until S do Z[i] : = 0;
Y[0] : = t;
for i : = 1 step 1 until S do Y[i] : = X[i];
K[0] : = 1;
for j : = 1 step 1 until 4 do begin for i : = 1 step 1 until S do begin real temp;
temp : = RHF(i);
K[i] : = temp + random × (if abs (temp) > delta then delta else temp)
end;
for i : = 0 step 1 until S do begin
Y[i] : = Y[i] + h×(A[j]×(K[i] - B[j]×Z[i])) ;
Z[i] : = Z[i] + 3×(A[j]×(K[i] - B[j]×Z[i])) - C[j]×K[i]
end i;
end j;
t : = Y[0];
for i : = 1 step 1 until S do X[i] : = Y[i];
end special RKG;
```

3. Geometric background of the learning automaton.

In this section we shall consider some binary properties of the domains formed in Euclidean space by intersection of a number of hyperplanes, when the points in the space can be dichotomously separated with respect to a certain property σ. By this we shall understand that every point in the space either has or has not the property σ. No point can be neutral or ambiguous, and the separation of the points with respect to the property σ is independent of any factors but the property itself and coordinates of the points.

In the sequel of this section we shall confine ourselves to a simple case of Euclidean plane E^2 and straight lines, the results obtained will be easily extrapolated into any dimensions.

Let us denote an arbitrary point in the plane by P. This means that P denotes an ordered pair of real numbers (x,y). Similarly, let L denote an arbitrary straight line in the plane; we understand that L denotes an ordered triple of real numbers (a,b,c) such that $ax + by - c = 0$ is an equation of the line.

Define the binary product of a given point P_i and a given line L_j (denoted as $P_i * L_j$) as

$$P_i * L_j = 1 + \operatorname{sgn}(a_j x_i + b_j y_i - c_j) \pmod 1, \tag{3.1}$$

where, as usual,

$$\operatorname{sgn}(\xi) = \begin{cases} -1 \text{ if } \xi < 0 \\ 0 \text{ if } \xi = 0 \\ 1 \text{ if } \xi < 0. \end{cases}$$

Consider now a compact set Ω in E^2 composed of region **U** and its boundary, and a separating property σ We shall employ notation P^{σ} to indicate that point P possesses the property σ and notation $P^{\bar{\sigma}}$ otherwise. Notation P without a superscript will denote either a point whose dichotomous affiliation is irrelevant, or a point of which no knowledge as to possession of property σ is available.

If the region **U** is crossed by n lines $L_1, L_2, \ldots, L_n$ then Ω is divided into K compact sets Ω_k called domains, boundaries of Ω_k being segments of L_j and parts of the boundary of **U**. The number K satisfies the condition $K \leqslant M$ where M for E^2 is $n(n-1)/2 + 1$. In general case, M depends on a number of dimensions of the space and on a number of the hyperplanes drawn.

Consider now a set of points $\Pi = \{P_i^{\Sigma_i}\}$ $(i = 1.2,\ldots, m;\ m \leqslant K;\ \Sigma_i =$ either σ or $\bar{\sigma})$ such that none of domains contains more than one $P \in \Pi$. If Ω_k contains a point $P_i^{\Sigma_i} \in \Pi$, then we shall say that Ω_k is classified by this point. This is to be understood as it follows: if Ω_k is classified by $P_i^{\Sigma_i}$, then if $P_l^{\Sigma_l} \in \Omega_k$ we have $\Sigma_l = \Sigma_i$.

Of course, such classification of domains will be correct only for especially chosen lines L_i and set Π· and not every σ will permit to construct them at all. We leave the detailed topological analysis of possibility of such construction to be discussed in another paper.

Let us now introduce the binary matrix $\|A_{ij}\|$ $(i = 1,2,\ldots, m;\ j = 1,2,\ldots n)$ $A_{ij} = P_i * L_j$. To each row of this matrix uniquely corresponds one of the classifield domains Ω_v. Indeed, if we take any $P_\mu \in \Omega_v$ and form products $P_\mu * L_i (j = 1,2,\ldots, n)$ we shall have $P_\mu * L_j = A_{lj}$, where l is a subscript of the point which classifies Ω_v. Reversely, if we take point P_λ belonging to any of the classified domains, we can determine to which one it belongs by computing the components of vector $v(P_\lambda) = \{P_\lambda * L_j\}$ $(j = 1,2,\ldots, n)$ and comparing them with the rows of matrix $\|A_{ij}\|$.

This correspondence permits us to see whether a point P_λ possesses the property σ without checking it straightforward (what may in some cases become a cumbersome procedure), indeed, all we need to do is to determine to which of the classified domains it belongs (this may be easily done by simple arithmetics described above) and use the definition of classified domain for determining of the affiliation of P_λ.

If, however, the number of classifiel domains, m, is less than the total number of domains K, we are not able to establish the affiliation of all points $P \in$ **U** in this manner, for the vectors representing unclassified domains are not present in $\|A_{ij}\|$. Suppose now that the process by which we have arrived at the classified domains (i.e. the construction of lines L_j) and at the set **II** (i.e. the construction of points $P_i^{\Sigma_i}$) is such that i cannot be extended to give $m = K$ and preserve the correctness of the classification procedure, or that we

cannot guarantee that in a finite number of steps we shall be able to achieve $m = K$ (this is the case with Braverman's algorithm described in the next section). We may try then to employ a procedure which will generalize the existing classification so as to include yet unclassified domains. It cannot be expected that such a procedure would yield fully correct classification even if the starting classification were hundred percent reliable, but in cases when a "statistically reliable" classification is aimed at, the following procedure may be very helpful.

Consider the μ-th row of matrix $\|A_{ij}\|$, i.e. vector ν_μ, and vector $\tilde{\nu}_\mu^\lambda$ obtained from ν_μ by the following transformation:

$$\tilde{A}_{\mu j} = \begin{cases} A_{\mu j}, & j \neq \lambda \quad 1 \leqslant \lambda \leqslant n \\ |A_{\mu j} - 1|, & j = \lambda. \end{cases} \tag{3.2}$$

The vector $\tilde{\nu}_\mu^\lambda$ represents either domain $\tilde{\Omega}_\mu^\lambda$, adjacent to Ω_μ and separated from it by a segment of the λ-th line L_λ, or a non-existing domain. E.g. vector ν_7 for the situation depicted in Fig. 1 is (1,1,1) and $\tilde{\nu}_7^2 = (1,0,1)$ represents domain $\tilde{\Omega}_7^2 = \Omega_5$, whereas $\tilde{\nu}_2^2 = (0,0,0)$ represents the non-existing domain $\tilde{\Omega}_2^2$. Generally, the transformed vector $\tilde{\nu}_\mu^\lambda$ will represent a non-existing domain if λ corresponds to a line no segment of which forms a part of the Ω_μ boundary.

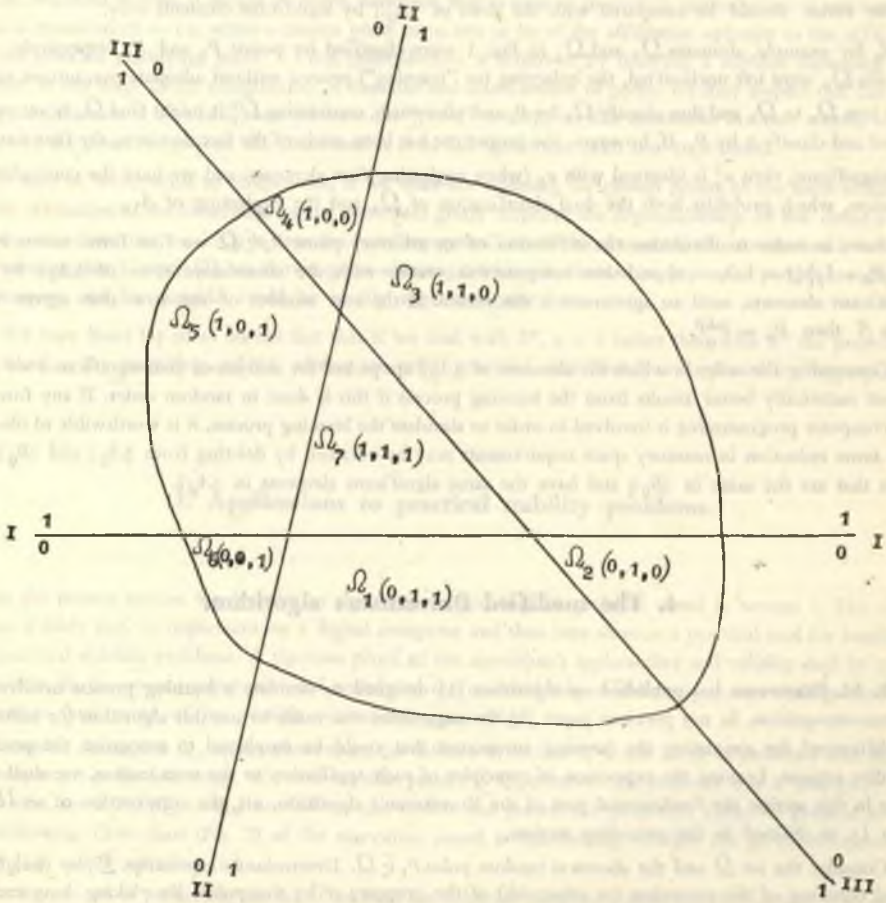

Fig. 1. A simple case of seven domains formed by three lines.

For reasons that will become apparent later we shall not make any distinction between existing and non-existing domains. Instead, we shall carefully distinguish two situations arising from the examination of the classifications of domains Ω_μ and $\tilde{\Omega}_\mu^\lambda$. Let the domain Ω_μ be classified by point $P_\mu^{\Sigma\mu}$. If domain $\tilde{\Omega}_\mu^\lambda$ is classified, i.e. if among the rows of $\|A_{ij}\|$ there is vector $\nu_\nu = \tilde{\nu}_\mu^\lambda$, we check whether $\Sigma_\mu = \Sigma_\nu$. If no — we call this case contradictory and take no further action with these values of μ and λ. If the answer is yes or if domain $\tilde{\Omega}_\mu^\lambda$ is unclassified, we "erase" the separating segment and consider thus enlarged domain as an entirety. In matrix $\|A_{ij}\|$ this entirety will be represented by either of two vectors ν_μ and ν_ν provided that we neglect the values of elements $A_{\mu\lambda}$ and $A_{\nu\lambda}$. In order to mark the neglected elements of the array $\|A_{ij}\|$ we create a new binary matrix $\|B_{ij}\|$ of the same dimensions as the $\|A_{ij}\|$ and fill it with elements according to the following prescription: if A_{ij} is to be neglected $B_{ij} = 0$, otherwise $B_{ij} = 1$. The elements of $\|A_{ij}\|$ for which the corresponding B are not zeros will be called significant. It is quite clear that if transformation (3.2) leads to a non-existing domain the corresponding element $A_{\mu\lambda}$ may safely be considered as insignificant and neglected in all identification processes.

The just described procedure is to be repaeted for all elements of $\|A_{ij}\|$. When repeating it, however one should remember that the adjacent domain indicated by transformation (3.2) may already be joint to another of its neighbours — and thus be classified inspite of the absence of the corresponding row in $\|A_{ij}\|$. In order to avoid any ambiguity such as dual (and perhaps opposite) classification of the originally unclassified domains, in establishing which domain is represented by a given transformed vector, the components of the vector should be compared with the rows of $\|A_{ij}\|$ by significant elements only.

If, for example, domains Ω_5 and Ω_1 in Fig. 1 were classified by points P_5 and P_1 respectively, and domain Ω_6 were left unclassified, the enlarging (or "learning") process without adequate precautions might have join Ω_6 to Ω_5 and thus classify Ω_6 by P_5 and afterwards, considering $\tilde{\Omega}_1^2$ it might find Ω_6 as yet unclassified and classify it by P_1. If, however, the proper use has been made of the fact that in ν_5 the first element is insignificant, then $\tilde{\nu}_1^2$ is identical with ν_5 (when neglecting first elements) and we have the contradictory situation, which probihits both the dual classification of Ω_6 and the neglecting of A_{12}.

Now, in order to determine the affiliation of an arbitrary point $P_a \in \Omega$ we first form vector $\nu_a = \{P_a * L_j\} (j = 1,2,\ldots,n)$ and then compare this vector with the consecutive rows of $\|A_{ij}\|$ by the significant elements, until an agreement is discovered. If the row number of the row that agrees with ν_a is β then $P_a = P_a^{\Sigma\beta}$.

Concerning the order in which the elements of $\|A_{ij}\|$ are picked for analysis of their significance we may expect statistically better results from the learning process if this is done in random order. If any form of the computer programming is involved in order to simulate the learning process, it is worthwihle to observe that some reduction in memory space requirements may be obtained by deleting from $\|A_{ij}\|$ and $\|B_{ij}\|$ the rows that are the same in $\|B_{ij}\|$ and have the same significant elements in $\|A_{ij}\|$.

4. The modified Braverman's algorithm.

E. M. Braverman has published an algorithm [1] designed to simulate a learning process involved in pattern-recognition. In my previous paper [5] the suggestion was made to use this algorithm (or rather its modification) for simulating the learning automaton that could be employed to recognize the practical stability regions. Leaving the exposition of principles of such application to the next section, we shall consider in this section the fundamental part of the Braverman's algorithm, viz. the construction of set Π and lines L_j, as defined in the preceding section.

Consider the set Ω and the chosen at random point $P_1 \in \Omega$. Determine its superscript Σ_1 by straightfor ward checking of the possession (or otherwise) of the property σ by this point. By picking long enough asequence of random points belonging to Ω we will arrive at point P_μ such that $\Sigma_\mu = \Sigma_1$ (we assume that

neither of the two classes of points is empty in Ω and both are of the same number of dimensions as Ω). Disregarding points $P_2, P_3, \ldots, P_{\mu-1}$ we may define $\mu = 2$ and draw a line $L_1 = (a_1, b_1, c_1)$, where a_1 and b_1 are random numbers $-1 \leqslant \max(|a_1|, |b_1|) \leqslant +1$ and $c_1 = (a_1(x_1 + x_2) + b_1(y_1 + y_2))/2$. This line divides Ω into two domains Ω_1 and Ω_2, classified by points $P_1^{\Sigma 1}$ and $P_2^{\Sigma 2}$. Next, we shall apply repeatedly the following procedure: Take the random point $P_m^{\Sigma m}$. This point belongs to one of the already existing domains $\Omega_1, \Omega_2, \ldots, \Omega_k$, $k \geqslant m-1$, formed by n lines. Among these domains there are m—1 domains which are classified by the points $P_1, P_2, \ldots, P_{m-1}$. If point P_m belongs to certain unclassified domain Ω_j we shall use this point to classify the domain. If domain Ω_j is already classified by point $P_s^{\Sigma s}$, $s < m$, and $\Sigma_s^* = \Sigma_m$ then we shall reject point P_m from our consideration and pick another random point assigning to it the same subscript m. If, on the contrary, $\Sigma_m = \bar{\Sigma}_s$ then we draw $(n+1)$st line $L_{n+1} = (a_{n+1}, b_{n+1}, c_{n+1})$ where $c_{n+1} = (a_{n+1}(x_m + x_s) + b_{n+1}(y_m + y_s))/2$ and a_{n+1}, b_{n+1} are again random numbers from the interwal $[-1,1]$, and thus increase a number of the existing domains. The point P_m classifies that one of the newly formed domains to which it belongs.

By repeating this process we shall always have the situation formally identical with the one considered in the preceding section, viz. n lines defining K domains, m—1 of which contain one (and only one) $P \in \Pi$, where Π denotes the set of our randomly chosen and checked against the property σ points.

Of course, our theoretical assumption concerning uniform affilation of all the points beloning to a particular classified domain will not be generally fullfilled by the domains and points generated at random in the described process. It should be noted, however, that as soon as an explicit contradiction to our sasumption is encountered — i.e. when a chosen point turns out to be of the affiliation opposite to the affiliation of the relevant classifying point — this contradiction is removed by drawing a suitable separating line. Hence, at any stage of the construction of domains and classification of points we may suspect that the existing classification is unreliable, but we have no "proof" to support this statement. In other words, at any stage of the described process the classification is in full agreement with our experience.

It may be worthwhile to observe that if we were not rejecting the chosen points of the same affiliation as the affiliation of the classifying points, we could gratly improve the dependableness of the classification by drawing lines that would have been the best "separators" in statistical sense. In this connection it is worth mentioning that Cooper [2] has shown great advantages of hyperplanes as separators in pattern recognition techniques, and his remarks could be readily applied to our case.

We once more lay stress on the fact that if we deal with E^n, $n > 2$ rather than with E^2, the procedures described in the preceding and this section are valid provided that "lines" are replaced by "hyperplanes"

5. Applications to practical stability problems.

In the present section we describe an algorithm for solving the problem stated in section 1. This algorithm is fairly easy to implement on a digital computer and thus may serve as a practical tool for handlinig the practical stability problems. A rigorous proof of the algorithm's applicability and validity shall be given elsewhere, in the present paper, devoted mainly to the practical side of the problem, we shall present the algorithm (and some of its possible extensions) intuitively.

Consider the region $\mathbf{Q}_1$ of the phase-space (as defined in section 1) as the set Ω of section 3 and define the property σ in the following. We shall say that point $P \in \mathbf{Q}_1$ possesses the property σ if it belongs to $\mathbf{Q}_0$, otherwise, i.e. if $P \in (\mathbf{Q}_1 - \mathbf{Q}_0)$ we shall say that it does not possess this property. Thus, we propose to use the following flow-chart (Fig. 2) of the algorithm aimed at establishing whether the given initial conditions (1.6) satisfy criterion (1.8).

In the process of generating the set Π we shall use a special integration procedure (cf. section 2) to determine the affilation of a chosen point in Π.

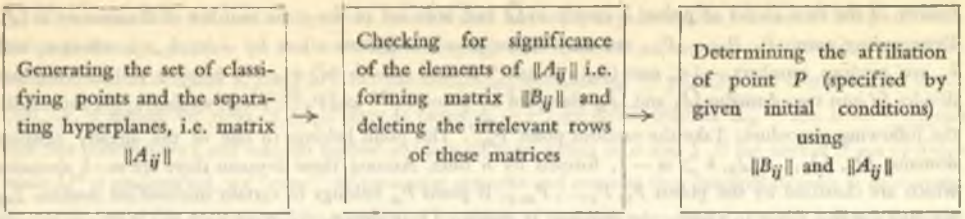

Fig.2

This algorithm may seem to be greatly uneconomical since it requires many numerical integrations of equation (1.5) to be performed in order to save one integration, that would have been needed to check criterion (1.8) directly, If, however, equation (1.5) represents the random case, the straighforward integration employed for checking of criterion (1.8) may yield completely erroneous results, whereas if it is used to determine the affiliation of many points in II we may expect statistically better results. And besides, drawing the random hyperplanes may smooth over (at least partly) the possible bias arising from application of special integration procedure. Secondly, if we need to solve the problem of performance of the real system for great many different initial conditions (1.6), and a number of different cases is greater than a number of integrations needed in order to complete the first box of a given flow-chart (fig. 2), the over-all gain of computing time may be very substantial. Another important factor is that the determination of the point's affilation by means of the method described in section 3 requires (when performed on digital computer) a very few arithmetic and some logical operations whereas the integration procedure will usually require rather a large number of arithmetic operations, thus the straightforward checking for fullfillment of (1.8) requires much longer a time than checking by means of the method from section 3. This fact may turn out to be of utmost importance when the "real time" computing is required.

Suppose now that we already have a set of separating hyperplanes and the corresponding classifying points. We can test the reliability of the determination of points affiliation by taking a sample of random initial conditions and determinig their affiliation by straightforward checking and comparing thus obtained affiliations with the ones determined by the method from section 3. The straightforward checking must here be performed by observing a behaviour of the real system, or by simulating this behaviour by integrating equations (1.5) using a method similar to a special integration routine described in section 2, where the procedure random should be replaced by a procedure which simulates the real perturbances occuring in the course of the process. Needless to say that when such simulation becomes impossible due to, e.g., lack of the precise knowledge of the perturbing factors, the special integration method may be used again in the described form; this time however we understand that it simulates the real behaviour, i.e. that the behaviour of the system would have been fully consistent with the numerical solution if the numbers supplied by the procedure random were to represent exactly the relevant perturbations.

If such test indicates too low a reliablity of indirect determination of the point's affiliation, then we may draw some more hyperplanes and classifying points.

Consider now another flow-chart (Fig. 3). It is pretty clear that the part of the flow-chart included in the broken-line rectangle and denoted by A is independent of specific system and may, be made quite general (this part of the flow-chart will be hencefort called the "learning process"). An ALGOL programme that describes possible computer programme simulating the learning process is given as an appendix to this paper.

The computer programme for the complete process described by the flow-chart in Fig. 3 may be considered as a digital model of an automaton working in two modes. Mode I is the learning process: input consists of the data describing the system to which the automaton is to be joined, i.e. the functions f, region $\mathbf{Q}_1$, and constants T and δ; output consists of matrices $\|A_{ij}\|$ and $\|B_{ij}\|$, and information on classification of the domains represented by the rows of $\|A_{ij}\|$. This information may be in fact included into the matrix $\|A_{ij}\|$ as its zero column, cf. the Appendix.

The operation of the automaton in Mode I is independent of the real system and may be performed "off line" with respect to the examined system.

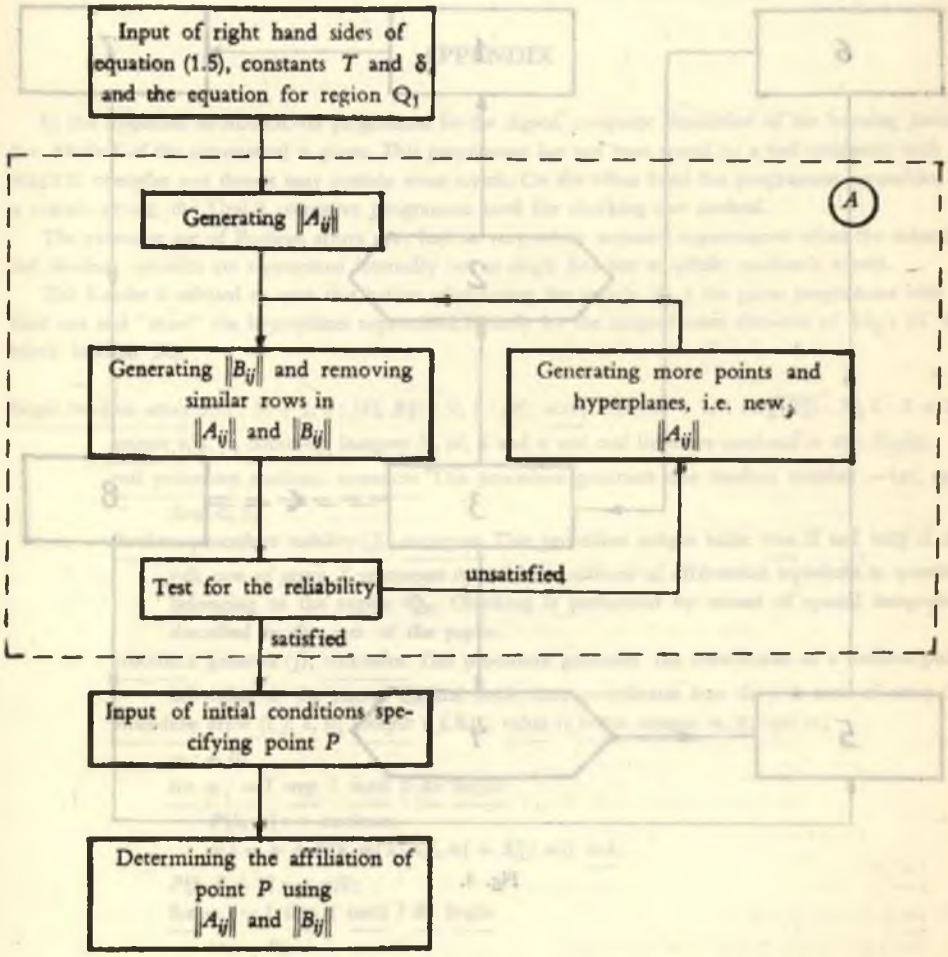

Fig. 3

Mode II is the determination process which tells whether given initial conditions bring about a satisfactory performance of the system. This mode uses as input a particular set of initial conditions (1.6) and it gives as output the affiliation of this point. Thus the second mode may be called a decision making or retognizing process, and should be performed in practice "on line" with the system it examines. In fact, the input to Mode II of the automaton is the ouptut of initial conditions measuring device, and the automaton's output in this mode is the decision whether the system is to be allowed to proceed from given initial conditions.

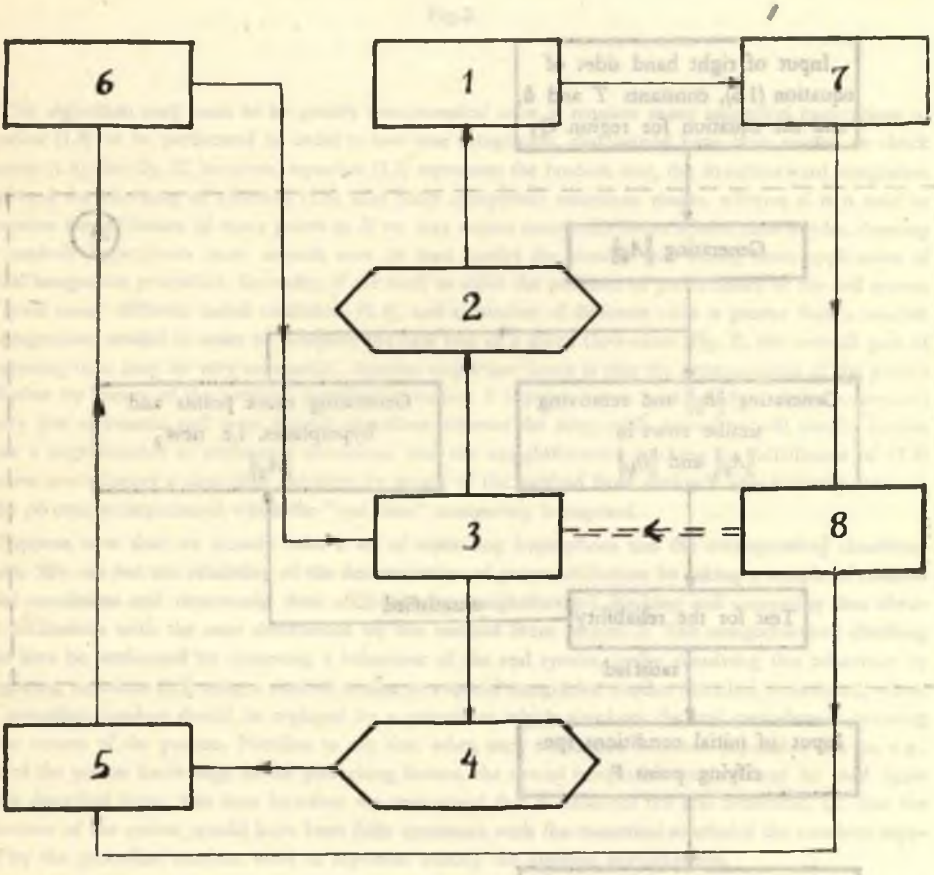

Fig. 4.

Meaning of the boxes is the following:

1. Performance of the system
2. Decision: to proceed with given initial conditions
3. Automaton mode II
4. Decision: not to proceed with given initial conditions
5. Creating new initial conditions
6. Initial conditions measuring device
7. Checking whether the performance of the system is satisfactory
8. Drawing new hyperplanes and performing the necessary augumentations and changes in $\|A_{ij}\|$ and $\|B_{ij}\|$ (as in Mode I after the discovery of a contradictory point)

When the automaton is simulated on a digital computer we encounter the usual difficulties arising from a finite size of the computer memory that limits a number of hyperplanes and classifying points we may use. If, however, in the learning process we have not used all memory cells allowed to the output information we may employ another variation of mode II, viz, a combination of decision making and learning. The flow-chart for such a variation will be like the one presented in Fig. 4 where the double broken line denotes the transfer of information within the model of the automaton, viz. between its first and second mode.

Thus it seems that in practical applications we may derive some profit by not using all the available storage for the learning process and permitting the automaton to "learn from its misjudgements".

APPENDIX

In this Appendix an ALGOL-60 programme for the digital computer simulation of the learning process (i.e. Mode I of the automaton) is given. This programme has not been tested on a real computer with an ALGOL compiler and thus it may contain some errors. On the other hand the programme ressembles, to a certain extent, the Ural-2 computer programme used for checking the method.

The extensive use of Boolean arrays may lead to very severe memory requirements when the subscripted Boolean variables are represented internally not as single bits but as whole machine's words.

The Reader is advised to note that before constructing the matrix $\|B_{ij}\|$ the given programme tries to find out and "erase" the hyperplanes represented entirely by the insignificants elements of $\|A_{ij}\|$ (cf. the block labelled B3).

```
begin Boolean array A[1 : N + 1, 0 : M], B[1 : N, 1 : M]; array X[1 : N + 1, 1 : S], P[1 : M, 1 : S + 1];
        integer r, j, p; comment Integers N, M, S and n and real limit are nonlocal to this block;
        real procedure random; comment This procedure generates one random number —1 ⩽ ran-
            dom ⩽ 1;
        Boolean procedure stability (j); comment This procedure assigns value true if and only if the
            j-th row of array X represents the initial conditions of differential equations in question
            belonging to the region Q0. Checking is performed by means of special integration
            described in the text of the paper;
        procedure generate (j); comment This procedure generates the coordinates of a random point
            belonging to the region Q1 and loads these coordinates into the j-th row of array X;
        procedure draw (i, j, k, l); integer i, j, k, l; value i; begin integer m, n; real w;
            w := 0;
            for m := 1 step 1 until S do begin
                P[k, m] := random;
                w := w + P[k, m]×(X[i, m] + X[j, m]) end;
            P[k, S + 1] := w/2;
            for m := 1 step 1 until l do begin
                w := 0;
                for n := 1 step 1 until S do
                    w := w + P[k, n]×X[m, n];
                A[m, k] := w—P[k, S + 1] ⩾ 0 end;
            r := r + 1
        end draw;
        procedure update (l); integer l;
            begin integer k, l; real w;
                for k := 1 step 1 until r — 1 do begin
                w := 0;
```

```
                for l := 1 step 1 until S do
                    w := w + P[k, l]×X[i,l];
                A[i,k] := w—P[k,S + 1] ⩾ 0 end
end update;
    for i := 1, j while j ⩽ 2 do begin
        generate (j); A[j,0] := stability (j);
        if j = 1 then go to L1;
        if A[1,0] ≡ A[2,0] then j := j—1
        L1 :j := j + 1 end j;
    r := 1; draw (1,2,1,2); p := 3;

B1: for j := p, j while j ⩽ N ∧ r < M do
    begin integer l,k;
        generate (j); A[j,0] := stability (j); update (j);
        for k := 1 step 1 until j — 1 do begin
            for l := 1 step 1 until r — 1 do
            if ¬A[k,l] ≡ A[j,l] then go to L1;
            if A[k,0] ≡ A[j,0] then j := j — 1 else
            if r ⩽ M then draw (k,j,r,j);
            go to L2;
        L1: end k;
    L2: j := j + 1 end B1;

B2: begin integer l,k,m,n;
    for l := 1 step 1 until r — 1 do begin
        for k := 1 step 1 until j — 2 do begin
            for m := k + 1 step 1 until j — 1 do begin
                for n := 1 step 1 until l — 1, l + 1 step 1 until r — 1 do
                    if ¬A[k,n] ≡ A[m,n] then go to L1;
                if A[k,0] ≡ A[m,0] then go to L2; L1: end m end k;
        for k := l step 1 until r — 2 do begin
            for m := 1 step 1 until j — 1 do A[m,k] := A[m,k + 1];
            for m := 1 step 1 until S + 1 do P[k,m] := P[k + 1,m] end k;
        r := r — 1;          L2: end l
    end B2;

B3: begin integer k,l,m,n;
    for k := 1 step 1 until j — 2 do begin
        for l := k + 1 step 1 until j — 1 do begin
            for m := 0 step 1 until r — 1 do
                if —A[k,m] ≡ A[l,m] then go to L1;
            for m := 1 step 1 until j — 2 do begin
                for n := 0 step 1 until r — 1 do A[m,n] := A[m + 1,n];
                for n := 1 step 1 until S do X[m,n] := X[m + 1,n] end m;
            j := j — 1; L1: end l end k;
    end B3;
```

```
if r < M ∧ j ≤ N then begin p := j; go to B1 end;

B4: begin integer i,k,l,m; Boolean array a,b,c[1 : M];
    Boolean procedure compare (a,b,i); Boolean array a,b; integer i;
        begin integer l;
            for l := 1 step 1 until i do
                if ¬ a[l] ≡ b[l] then go to L1;
            compare := true; go to L2;
            L1: compare := false;
        L2: end compare;
    real procedure test (n); integer n;
        begin integer success, trial, i; Boolean guesstab;
            success := 0;
            for trial := 1 step 1 until n do begin
                generate (N + 1); update (N + 1);
                for i := 1 step 1 until j - 1 do begin
                    Boolprod (A,B,a,r - 1,N + 1,i, false);
                    Boolprod (A,B,b,r - 1,i,i, false);
                    if compare (a,b,r - 1) then guesstab := A[i,0];
                    go to L1; end i;
                L1: if guesstab ≡ stability (N + 1) then
                    success := success + 1
                end trial;
                test := success/n
            end test;
    procedure Boolprod (A,B,C,i,j,k, special);
        Boolean array A,B,C,; integer i,j,k; Boolean special;
        begin integer l;
            if special then begin for l := 1 step 1 until i do
                                    C[l] := A[l] ∧ B[k,l] end
                       else for l := 1 step 1 until i do
                                    C[l] := A[j,l] ∧ B[k,l]
            end Boolprod;
        for i := 1 step 1 until j - 1 do
            for k := 1 step 1 until r - 1 do B[i,k] := true;
        for i := 1 step 1 until r - 1 do
            for k := 1 step 1 until j - 1 do begin
                B[k,i] := false;
                Boolprod (A,B,a,r - 1,k,k, false);
                for l := 1 step 1 until k - 1, k + 1 step 1 until j - 1 do begin
                    if A[k,0] ≡ A[l,0] then go to L1;
                    Boolprod (A,B,b,r - 1,l,k, false);
                    Boolprod (a,B,c,r - 1, 0,l, true);
                    Boolprod (b,B,b,r - 1,0,l, true);
                    if compare (b,c,r - 1) then begin B[k,i] := true; go to L2 end;
```

```
            L1: end l;
        L2: end k;
    for i := 1 step 1 until j − 2 do begin
        Boolprod (A,B,a,r − 1,i,i, false);
        for k := i step 1 until j − 1 do begin
            if ¬A[k,0] ≡ A[i,0] then go to L3;
            Boolprod (A,B,b,r − 1,k,k, false);
            if compare (a,b,r − 1) then
                begin for l := k step 1 until j − 2 do begin
                    for m := 1 step 1 until r − 1 do begin
                        A[l,m] := A[l + 1,m];
                        B[l,m] := B[l + 1,m] end m;
                    for m := 1 step 1 until S do
                        X[l,m] := X[l + 1,m] end l;
                j := j − 1
            end if;
        L3: end k
    end i;
    if test (n) < limit ∧ j ≤ N ∧ r < M then begin p := j;
            go to B1 end
end B4
end programme
```

References

[1]. Braverman, E. M., Автоматика и телемеханика, XXXIII, 3, (1962) p. 349—364.

[2]. Cooper, P. W., Cybernetica (Namur), V, 4, (1962), p. 215—238.

[3]. La Salle, Joseph, and Lefschetz, Solomon, Stability by Liapunov's Direct Method, Academic Press, N. Y., London, 1961.

[4]. Srinivasan, S. K., ZAMM, 43, 1963, p. 259—263.

[5]. Turski, Władysław, Algorytmy, S1, (1963), p. 21-28