Computer Aided Design and Operation of Reservoirs

by

M.J. Harley, B.Sc.

A thesis submitted for the degree of

Doctor of Philosophy

Department of Civil Engineering,

The University of Aston in Birmingham.

September, 1973.

Synopsis

This thesis describes the application of dynamic programming to the design and operation of water resource systems, and investigates the use of a version of the simple user-orientated problem solving language, HYDRO, as developed by the author, in which hydrological and hydraulic procedures may be embedded.

Stochastic dynamic programming methods are explored and the policy iteration technique of Howard is developed and applied to a two reservoir system to obtain long term operating rules. The author found no previous applications of this method to reservoir systems design in the current literature.

Because of the high computing costs involved in the stochastic methods, the author researched into the use of deterministic dynamic programming applied to historical or synthetically generated inflow sequences for finding long term rules.

It is concluded that the method described produces satisfactory results and may in fact be more accurate than stochastic dynamic programming in some cases because more reliable representations of the basic data structure may be used.

Acknowledgments

The author would like to thank Dr. T.R.E. Chidley, B.Sc., Ph.d., M.I.C.E., for his encouragement and advice throughout this research.

Thanks are also due to the staff of the University Computer Centre for their help in the early part of this work.

Finally, the author is indebted to the Science Research Council for their financial support.

Notation

The notation is defined in the text.

Only the main variables are repeated here.

x	state variable.
u	control variable.
t	time variable.
σ	dummy time variable
f, g	state transition functions.
l	scalar function of cost per unit time.
$\overline{X}$	set of admissable states.
$\overline{U}$	set of admissable controls.
I	function of optimum costs.
ψ	function of values at end of process.
J	objective function.
K	total number of stages in process.
k, n	stage variables.
$\in$	in the set.
$\overline{w}$	vector of inflows.
E	expectation operator
v	total expected cost of going to end of process.
q	expected immediate cost.
$\overline{P}$	transition probability matrix.
p, b	probabilities.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 The Field of Study

The design of a water resources system is one of the more interesting aspects of civil engineering at the present state of the art since no precise mathematical formulation can yet be applied to many of the problems involved in the efficient management of a natural and uncertain phenomenon such as stream flow. Even if it were possible to describe accurately the statistical behaviour of run-off at some point in a catchment, the mathematical model constructed would be so complex that modern computational methods would be totally inadequate to solve the problems with any degree of accuracy.

Since it is not generally feasible to obtain the perfect answer, then some compromise must be found between a purely subjective solution and the ideal, but impracticable, mathematical analysis. At this point, care must be taken not to fall into the trap of expending excessive effort to achieve over-precise answers to the parts of the problem which may be explained by mathmematical formulae at the cost of overlooking the fact that the uncertain part of the problem should be treated with the most well-informed judgment possible. In this respect, it is better that the simpler problems be executed by computers in order to leave the hydrologist more time to consider the value of his method of approach, for no matter how precise his solutions may be they are still worthless if the wrong problem has been solved. In order to investigate the application of computer methods in water resources projects it is first necessary to

establish the types of problem involved in the design process. The first stage in the design of a water resources system is to identify the objectives of the scheme. Having identified objectives one may look at a number of alternatives which may feasibly meet them. To this end, some means of comparison of each alternative is required, a so-called objective function. If a range of objectives, which may be conflicting in nature, is required, some way of weighting the various aims to form a unified whole must be determined. Alternatively, one can make a set of major choices on main objectives and sieve these through a set of minor objective functions. The determination of an objective function consistent with one's aims is of very great importance in the application of mathematical methods and needs further research. This thesis has taken the standpoint of finding what should be done given that an objective function has been defined.

When a good description of a system, or the broad outline of a set of possible systems, together with some criteria with which to judge the efficiency of each system, has been obtained, then the next step is to collect data defining the inputs and outputs of the system, the control rules and other system parameters.

Usually, the inputs are the hydrological data sequences associated with various parts of the system and the ideal demands on the system. The actual outputs are produced by operating the system and the efficiency of the system is a measure of the correspondence between these outputs and the required outputs or demands.

Much of this thesis has been devoted to finding ways of using computers to assist in the preparation of hydrologic inputs and the determination of control rules and system parameters for a class of systems dealing with reservoir regulation.

Ideally, the raw hydrological data for the catchment area in question should include historic records of rainfall, infiltration rates, groundwater discharges, surface run-off and evaporation rates, or records of the total rates of flow of streams and rivers at the sites of the proposed reservoirs and at river regulation points. Although the most important records are usually stream flows, which are, fortunately, easy to measure and are therefore often available, it is still useful to have the other records when data analyses and data generation have to be carried out in order that trends in the components of total flow may be identified and separated from purely random fluctuations. However, assuming that records have been taken, care must still be exercised in their use. In Chapter 2 work done by the author for the Dee and Clwyd River Authority will be described and it will be shown that the existing run-off records at different points in the catchment areas considered had to be adjusted because of incorrectly calibrated flow measuring instruments and due allowance had to be made for changing patterns of afforestation during the period of record.

A great deal of research has been carried out in recent years into developing sophisticated measuring equipment in order to obtain true records, and also into developing mathematical methods of analysing and correlating the records available, since it is often found

that long rainfall records may exist for a catchment but only a short total flow record is available. Therefore, ways must be found of estimating the required data from that which has been recorded.

After records have been collected and verified short-term information must be extracted from the data in the form of unit hydrographs and return periods of floods and droughts must be found. These data may be used to decide on spillway design flows or to assess whether the historic record itself covers a critical enough period to be used in designing the system.

The next process in design is to evaluate the demands and constraints which will affect the system and to decide the weights to be given to satisfying each demand if sufficient water is not available at any time to satisfy all demands. As stated above, it is this area of study which presents the greatest problems at the present time since many of the constraints and demands conflict in their requirements. For instance, there is an obvious conflict between a constraint which requires that the level in a reservoir should be kept below a certain limit in order that flood water may be accommodated and a demand for the supply of water which requires that a reservoir should be as full as possible. Even more difficult a problem is posed when socio-economic data must be taken into account as is becoming necessary at present. This type of data occurs as recreational and amenity requirements such as sailing and fishing which demand that rapid fluctuations in reservoir levels are avoided. Fortunately, however, these benefits are not likely to conflict with water supply requirements.

In the work done for the Dee and Clwyd River Authority, the demand was assumed to be to maintain a fixed minimum flow in the River Dee at a point some way downstream of the reservoir sites considered. This meant that the actual release from the reservoirs varied with the natural inflow into the River Dee between the reservoir sites and the regulation point. The idea of maintaining a fixed flow might not bear too much scrutiny, especially if the same flow is specified for every month of the year, since it becomes uncertain how to evaluate the worth of maintaining the flow and the cost of not maintaining it. A fixed flow may be specified because it is easier for the various industries and private concerns involved to appreciate how variations in the flow might affect them, but it may be possible to find an alternative and less wasteful method which would still satisfy the water users. The final area of study may be described as the actual design of the water resources system using the data available. In this part of the problem may be included any data generation which may be necessary if the historic records are inadequate. When the data to be used has been decided upon then it is necessary to choose a method of using it to arrive at a meaningful solution to the problem.

This thesis has looked at the application of computers, both in the general field of hydrology and in the particular case of using the hydrological data to obtain operating rules and optimal designs of reservoir systems.

1.2. Current Design Methods

For a system consisting of more than one source, it is known that a greater yield may be obtainable by the conjunctive use of resources in a catchment rather than by their independent operation, but the standard methods of design are not able to

take into account the allocation of releases between resources except in a very subjective manner, or for certain very special cases.

The well-tried mass curve analysis of Rippl simply assumes that one combined source exists and a total required storage is calculated for the worst conditions of inflow and demand. Having ascertained the total storage required, the problem still remains of allocating the total storage between reservoirs and of finding an operating rule to control the fraction of demand released from each reservoir.

Therefore, with the advent of multi-reservoired catchments, simulation of the performance of the system over a period of historic or synthetically generated streamflows with various possible control rules has been practised, the best combination of reservoir sizes and applied control rules being chosen, as measured by some kind of objective function. However, since the choice of control rules is so wide, generalised control rules, the same for several possible system parameters, tend to be applied, with a consequent loss of accuracy in determining the optimum operational policy. In the case of a new reservoir being constructed to add to an existing system, it is impossible, with generalised, non-optimum rules, to determine objectively the correct size of the new reservoir, since for every size tried a different optimum rule will in general be required. It is apparent that, if the rules employed in each simulation are not at the optimum for that particular configuration, then the costs and benefits of various configurations cannot be compared with true objectivity.

New developments in operational research, however, make it possible to find the absolute optimum rule for a system, and to calculate the expected costs which will be incurred by the use of such a rule. Hence, it now becomes possible to

compare different reservoir configurations operating under their own optimum operational rules.

The author will describe work carried out in conjunction with the Dee and Clwyd River Authority using simulation methods, and the development of a mathematical programming procedure, namely dynamic programming, and its possible integration into existing design methods. The work described in this thesis is not meant to replace simulation exercises completely, since the computational effort involved in dynamic programming is high even for a two reservoir case, but it is felt that for smaller projects, or for sub-systems of larger problems, a more efficient solution is obtained by this procedure than by any other method used at the present time.

Dynamic programming allows the determination of a true optimum operating policy for a multi-reservoir, multi-use system, using general constraints and benefit functions.

Methods such as linear and quadratic programming have been well-developed in recent years, but their use may lead to a severe simplification of the cost and benefit functions associated with an operating policy.

The determination of a long term operating policy is not beyond the scope of these methods, but even over a fixed time period, a larger amount of computer time is generally required for their solution than for the dynamic programming procedure.

1.3. Review of Current Literature

A paper by Butcher (38) which describes the different methods of mathematical programming available concludes that the dynamic programming computations are simpler and less time consuming than the corresponding linear or quadratic programming calculations, whether stochastic or deterministic models are used, even with the one multi-purpose reservoir case.

The various programming routines of use in water resources planning are briefly described in a paper produced by Larsen and Keckler (13) and examples are given for the simpler cases. The modifications of the basic computational procedures which may reduce the volume of calculation in certain cases are given but are not always applicable. Only the methods of general use have been used in this thesis.

Most of the resources operation literature is concerned with one multi-purpose reservoir but methods are described for obtaining near optimal rules for multi-reservoir, multi-purpose systems.

A paper by James (35) describes a marginal analysis of reservoir benefits for flood control, water supply and recreation, which yielded a rule curve showing maximum levels to be aimed for each month for one reservoir case, and maximum volume allocation between uses. He mentions that a cyclic process of optimising one reservoir at a time may be used for multi-reservoir operation. Although dynamic programming is not used in this paper it can easily be seen that it could be used instead of marginal analysis and the same method of cyclic optimisation could be employed for a multi-reservoir site and is basically the same as the successive approximations technique described by Bellman (10). This methodology was not adopted for the purpose of this thesis, but it could be of great use in reducing the computational requirements of dynamic programming when more than two reservoirs are to be operated conjunctively.

The tendency at present is to try to achieve long term operational rules by using stochastic data rather than relying on a deterministic historical or synthetically generated trace. Loucks (34) shows how to set up a linear program for the

stochastic one reservoir case using a four season cycle with random serially correlated inflows. The calculation produces steady state target lake levels and discharges but the volume of computation is large and the computer time involved is expensive.

Lloyd (26) has considered the stochastic approach to design with a one reservoir case and pumped storage but does not consider optimising the control rule. The aim of this work was to assess the long term probability of failure of a reservoir of given size using the statistical distribution of inflows rather than a deterministic trace. His paper illustrates the technique involved in solving for steady state probabilities of contents assuming no serial correlation in the inflows, but he briefly describes how correlation may be taken into account and states that the probability of failure may be increased when correlation is considered because of the tendency for high and low flows to form clusters. He also points out that more work is required on the speed of convergence to the steady state solutions which may be relevant for systems of relatively short life. Lloyd's work concerned only one reservoir and the operating rule is specified so that the transition matrices could be set up. No optimisation is involved in the work.

The method of application of Lloyd's work to the case of serially correlated inflows for the one reservoir case is described by Harris, Dearlove and Morgan (27) and an attempt is made to investigate the significance of serial correlation in estimating reservoir behaviour. They carried out a steady state solution of a two season model assuming independent inflows to obtain the long term probabilities of being at specified reservoir levels. They then repeated the calculations

but assuming various levels of serial correlation between the seasons. It was found that the inclusion of serial correlation may greatly increase the probability of failure, as expected, but the probabilities of being at the higher reservoir levels was not much affected by any correlation.

The transition matrix approach of Lloyd, based on Moran's work, is very similar to the policy iteration method of dynamic programming described by Howard (12) in that a set of simultaneous equations describing the state transitions is solved, but Howard's method leads to a way of determining the optimum long term operating policy. In the same way as the steady state probabilities of reservoir levels may be obtained by repeated multiplication of the transition matrices instead of solution of the equations, Howard's value iteration method of dynamic programming employs multiplication of similar matrices in order to arrive at the long term optimum operational policy.

Howard's latter method has been illustrated for a two source system by Schweig and Cole (17) and seems to be the only stochastic example of a multi-reservoir application of dynamic programming in the literature, apart from a similar paper written by Burley and Cole (20). A finite reservoir operated in conjunction with a limited aquifer is investigated in order to obtain the optimal long term monthly decisions describing the quantities of water to be released from each source given the reservoir level and whether the previous month's inflow to the reservoir was higher or lower than the average for that month. Four levels of reservoir contents formed the discrete points at which optimum controls were derived and five possible inflows with their associated

probabilities were considered in each month. The example allowed for four possible combinations of releases from the reservoir and aquifer which constituted the decision possibilities.

Schweig and Cole stated that convergence to the long term policy was mostly achieved in their experiements within five years of iteration, where convergence is assumed to have occurred when two consecutive years policies are the same. However, Burley and Cole point out in their paper that this might not be a sufficient test of convergence and that Howard's policy iteration might be required.

Schweig and Cole mention that the transition matrices which lead to the optimum policy may be multiplied together to obtain the long term probabilities of reservoir levels and hence to arrive at the expected costs of operating the system, which is similar to Lloyd's work. The costs may also be obtained by simulating the optimal policy over a period of synthetic or historic data. In this way alternative systems operating under then optimal policies may be compared, but Schweig and Cole state that the greatest hurdle is computer running time, which forces the use of large state divisions with the consequent inaccuracy, so that research is required into speeding the convergence of the policy.

In some cases, where only one stochastic source is involved, efficient methods of application of dynamic programming may be possible. Some examples are given in a technical memorandum of the Water Research Association (16).

Young (28) has described a method of using a deterministic dynamic programming approach, combined with regression analysis of the results, to formulate an annual release policy

based on synthetically generated inflow traces for a one reservoir case.

By standard statistical analysis of historical records Young was able to generate 1000 years annual inflow traces. Also, by specifying a correlation or reliability between any year's inflow and a forecast of that inflow made in the previous year, he was able to generate 1000 years of forecast data corresponding to the inflow trace, for any level of reliability.

The first step in his methodology was to perform a forward looking deterministic dynamic programming calculation on the first inflow trace starting from some given reservoir level. However, conventional backward looking dynamic programming may be used instead and may be used with more generality. Having thus obtained the best policy to follow, based on some economic loss function, in each year of the 1000 year record, he applied a regression analysis to the results. The last 100 years of the dynamic programming was ignored to avoid end effects. The left hand side of the regression function was the rule obtained from the record and the right hand side was a combination of the storage level at which this rule was chosen, the inflow, and, if required, the forecasted inflow for the next year obtained from the second trace generated. It is a disadvantage of Young's investigations that the current year's inflow is assumed to be known.

Young employed two types of economic loss function in his analyses. One was a quadratic loss function of the type $Li = (7-Di)^2$ where Li is the loss in the i th year and Di is the rule, which is the amount of water to release in the i th

year and the other was the piecewise linear loss function $Li = 4|7-Di|$. The constant, 7, represents a target release figure.

Young also investigated two types of regression function, linear and quadratic. The linear function was:

$$Di = a_o + a_1 Si + a_2 Xi + a_3 \tilde{X}i+1$$

where Si is the storage in the i th year and $\tilde{X}i + 1$ is the forecasted inflow for the (i + 1)th year. The a's are the regression coefficients.

The quadratic regression used the same variables and was of the form:

$$Di = a_o + a_1 (Si + Xi) + a_2 \tilde{X}i+1 + a_3 \tilde{X}^2 i+1 + a_4 (Si + Xi)^2 + a_5 (Si + Xi) \tilde{X}i+1$$

Young tabulated his results in a most compact and precise manner, the implications of which will be described briefly.

To investigate the errors involved in using an estimate of a policy rather than the population policy, Young generated twelve 1000 year inflow traces from the same population parameters, using the same reservoir size of four units and the quadratic loss function to determine the linear regression function applicable to each. Young did not include forecasting in the regression for this investigation. He then routed each policy through a 1000 year simulation to obtain the economic losses involved. Different inflow traces are used for each simulation but Young does not say whether he uses the same inflow traces for the simulation as he used for the equivalent dynamic program, although it is implied that they are different.

In order to compare the losses obtained to some fixed reference, he carried out each simulation again but using the

standard queuing theory policy, which is to release the target draft if possible, to release all the available water if this is less than the target, and to release more than the target if a spillage would occur.

Young then calculated the ratio of losses obtained by the dynamic programming method and those obtained from the standard policy. He found that the average ratio over the 12 trials was about 0.88 with a standard deviation of 1.05%, which is a very consistent result. He then generated 100 inflow sequences and ran simulations with this data using one fixed regression function. This was intended to find the errors involved in using a sample policy instead of the population policy. The average loss ratio was again about 0.88 and the standard deviation was 0.17%.

These two experiments show that the main source of error is in the regression function estimation.

Young repeated the method of using a fixed linear regression function and 100 simulations with various reservoir sizes and with several forecast reliabilities using the quadratic loss function. In all cases the standard deviation of the loss ratio was small, the maximum being 0.32% and the loss ratio itself showed that the regression policies gave improvements over the standard policy of from about 10%-30%, the larger improvements being obtained for larger reservoir sizes.

Using a reservoir size of four units, Young studied the loss ratios obtained for linear regressions and quadratic loss functions with increasing forecast reliabilities. Twelve regressions were carried out with reliabilities from 0 to 1. The results showed that an improvement over the standard policy

was achieved in each case. Young then calculated the percentage improvement in losses obtained by the forecasting cases over the no forecasting case. He deduced that improvements were zero or small up to a certain breakpoint reliability but after this the improvements increased monotonically with the increase in reliability, the maximum improvement achieved being 8.2% for perfect reliability. A study of Young's tabulated results does not show a monotonic increase, improvements being the same for some reliabilities and even falling for higher reliabilities. However, these fluctuations may be due to sampling errors and Young's deduction seems logical but more results would be required to show that this is a general trend.

He repeated the experiment using reservoir sizes of 7,10 and 15 units, and the same type of results were obtained, showing that for the quadratic loss function, the dynamic programming method combined with regression analysis leads to a better policy than the standard queuing theory policy, and that the higher the forecast reliability the more the improvement.

Young then studied the results obtained using linear regression and the piecewise linear loss function. He found that in all cases tried the standard policy was as good as or slightly better than the dynamic programming - regression analysis policy, but because of the consistent results obtained with the quadratic loss function, it can be assumed that the standard policy is optimal or near optimal and that the errors caused by discretisation of the reservoir levels and the errors involved in the regression estimation do not allow this method

to achieve the standard policy exactly. The maximum difference in losses between the standard policy and the dynamic programming policy was 6% and the range was 2%. Further there does not appear to be any pattern in the differences for the various reliability levels. Bearing in mind the error of 6% resulting from discretisation of the dynamic program, it is possible that this method could achieve even better results in the quadratic loss case if the intervals of discretisation were made smaller. Since the standard policy appears to be the best for every level of reliability of forecasting, this confirms that 'hedging' against future losses is less important with a linear loss function but Young's statement that forecasting has no bearing on the policy for the piecewise linear loss function is only true when any spillage that occurs is assumed to add to the draft. When spillage is regarded as waste water hedging may become important.

Using the quadratic regression function and the quadratic loss function with a reservoir size of 10 units, Young then carried out five trials with varying forecast reliabilities. He found that no forecast case gave very little improvement over the standard policy but large improvement (23%-27%) with forecasting. Young's interpretation of this result was that the large improvements with forecasting were misleading since the no forecast case only gave an improvement of 4%, and that the large improvements could be due to the addition of linear forecasting terms in the regression equation. He then states that the large values are not critical since comparative improvements over the standard policy using the linear regression function were equal to or greater than these. Hence,

he deduced that the quadratic regression results were less desirable since both sets of results represented answers to the same problem. By inspection of Young's table of results it can be seen that it is simply not true that the improvements gained by linear regression are equal to or greater than those gained by quadratic regression, even within 'sampling error'. However, Young's further deduction that it can be inferred that linear regression is as good as or better than more complicated (i.e. quadratic) regressions when a quadratic loss function is used is probably true because the multiple correlation coefficients are much lower for the quadratic regression.

Using the results obtained, Young plotted the percentage improvements achieved over the standard policy for the perfect forecasting and no forecasting cases for various detention times. His detention time was the reservoir size divided by the average annual inflow. He found that both curves showed an increase in improvements with increase in detention time until a ceiling was reached at 35% improvement at a detention time of about two years for the perfect forecast, and an improvement of around 30% at 2.5 years for the no forecasting case. He noted that the perfect forecasting and no forecasting curves approached each other as detention time increased and he concluded that large storage capability is a sufficient hedge against an uncertain future.

Young also plotted the improvements of various levels of forecasting over the no forecasting cases against detention time. The curves form a skewed bell shape relative to the time axis. The curve for perfect forecasting reached a maximum

of about 10% improvement when the storage was equal to the target draft figure and then tapered off. The decreasing percentage improvement of forecasting over no forecasting shows again that the impact of one year forecasting diminishes with increasing reservoir size(S_m).

In conclusion Young states that the optimal policy is not necessarily the standard policy of queuing theory and the penalty of using the standard policy when it is not optimal may be as much as 35% additional loss. Experiments indicate forecasting to be of value for quadratic loss functions but not for the piecewise linear loss function used. For quadratic loss, the importance of one year forecasting increases with S_m to a maximum and then decreases. Forecasting has little significance for large reservoirs, but for smaller ones perfect forecasting can almost double the percentage improvement due to using an optimal policy rather than the standard policy.

Young's work is of great use but he did not investigate the relationship between the policies obtained by his deterministic dynamic programming method combined with regression analysis and the optimal policies which might have been achieved with stochastic dynamic programming.

Some indication of this comparison is given by the consistency of his results when simulating the system using the regression functions obtained from various inflow traces. However, it is believed that the results presented for this one reservoir case are not sufficient to show the great improvements which can be obtained by using dynamic programming, since the optimum policy will never by very different from the standard policy for a water supply reservoir.

The author has investigated the application of both Howard's value iteration and policy iteration methods to a system of two reservoirs and, independently of Young, came to the same conclusion that the computational effort was not justified in view of the inherent inaccuracies involved because of the gross discretisation necessary to confine computer running time to realistic proportions, and that some kind of Monte Carlo deterministic dynamic programming approach is preferable when more than one source is stochastic.

1.4. The Objective Function

It has been stated above that whichever method of determining an optimum sizing of resources and the relevant operating policy is used, the performance of the system must be measured against some kind of objective loss or benefit function. Very often, though, only a subjective method of appraisal is used in simulation exercises because the hydrologist cannot define his objectives in mathematical terms but only has a basic intuitive feeling of the result required. However experienced the hydrologist may be this method of approach is obviously open to inefficiency if not error. In view of this, and because more mathematical methods demand it, it is felt that as much as possible of the problem should be expressed in objective terms and that any part of the problem which must still be thought of in a subjective way should be considered only after several tentative solutions have been obtained using only the objective data. At least in this way the hydrologist will be able to see clearly what proportion of his solution depends on fact and what proportion depends on assumptions.

1.5. Use of Computer Languages

Because hydrological problems usually involve the manipulation of large volumes of data, the use of computers becomes especially important to the hydrologist in order to cut down the time involved in what is generally a repetitive and laborious process. However, the writing of computer programs to carry out the calculations can become extremely complicated and time consuming in itself, and because of this the author has investigated methods of facilitating the communication between the hydrologist and the computer. Because of the successful use of simulation languages in inventory fields and other queuing problems it appeared that a language might help with the reservoir simulation problem. Properties of languages, and the properties of the problem were compared and experiments were made with the general purpose Hydro Language developed by Bugliarello (1) at the Carnegie Institute of Technology. Modifications to the basic language were made to further evaluate the possibilities since the original structure was not flexible enough to allow sufficient user control over the course of calculations, and, more important, because the original system contained flaws in the logic which would not allow it to be used in the way that Bugliarello envisaged.

Routines may be incorporated with the language to manipulate raw data and convert it to a form suitable for reservoir regulation problems and in particular into a form suitable for simulation or mathematical programming exercises.

The specialist simulation languages looked at had very little appeal in regard to the water resources problems because of the more complex data structure required in this

field than in the industrial problems to which the simulation languages are applicable, and because of the greater complexity of decisions used in operating a water resources system.

After a thorough study of the different levels of problem-orientated languages (POL'S) it would seem that the stages leading up to the design of reservoirs and their operating systems lend themselves to efficient treatment using POL'S, but the actual design process would be best approached using only a data interface with the POL. The simulation of a reservoir system and the application of dynamic programming procedures, which have much in common with simulation, are best carried out using a universal language, such as Fortran or Algol, but using the simple data structure of a system like Hydro or Genesis. With this in mind, the author has allowed for the use of pure Algol code in the Hydro language.

A real problem was considered and current methods and newly developed methods of solution applied to it, and levels of utility of the language were defined. The applicability of the language in data checking, correlation analysis, manipulation and generation of synthetic data was found good. Once a simulation or description of the system had been set up for the problem, modification to program and data are facilitated with a language. This can be done to some extent with an advanced computer operating system, such as the George 3 System used on I.C.L. machines, and it can be seen how new developments in hardware and software can corrupt work being done or render it obsolescent. However, it can also be seen that a system which remains stable as far as the user is concerned is desirable and languages such as Hydro can be updated to take advantage of new methods while the users input can remain the same.

CHAPTER 2

THE SIMULATION METHOD

2.1. Introduction

At the present time the most favoured method of designing water resources systems and ascertaining the appropriate operating rules is to use simulation with either the historical records of flows or synthetic records. An operating rule is chosen, releases are made accordingly, and the consequent levels of the reservoirs throughout the period of records are analysed, partly objectively and partly subjectively to determine whether the system satisfies certain constraints.

The Water Research Association have used search methods coupled with simulation to assist in the finding of a good operational plan and system configuration. System and control parameters are fixed and a simulation is performed, resulting in an overall operating cost measured by some objective function. The parameters are changed, in turn, to find the improvement in the objective function. The search methods are designed to prohibit backward moves on the response surface once a feasible move has been made. These methods are useful and relatively cheap in computer time when only a few system parameters exist, but complicated control rules and configurations might lead to excessive computer running times.

It is felt that systems might be unrealistically over-simplified if designers attempt to use these methods in all instances. As an illustration of the simulation method, the author will describe work carried out in conjunction with the Dee and Clwyd River Authority over a period of about twelve

months for the determination of the optimum sizing of the proposed Brenig Reservoir in North Wales, and for the deduction of a suitable operating policy for the allocation of releases between the Brenig and the other major source in the system, Llyn Celyn.

2.2. The problem

Figure 2.I. shows the River Dee and its major tributaries, its two regulating storages at Llyn Tegid and Llyn Celyn, the amounts and locations of the principal abstractions for public water supply, and the location of the principal river Dee gauging station at Erbistock.

Releases to the river are controlled by sluices at Bala; as these are downstream of the Dee / Tryweryn junction, the sluices control not only the level and flow from Llyn Tegid (Lake Bala), but also the flow from the Tryweryn, releases to which are in turn controlled by outlet valves at Llyn Celyn.

Since the existing reservoirs in the area are used to regulate the river Dee in order to maintain a prescribed minimum flow upstream of the Erbistock gauging station, the yield obtainable by construction of Brenig reservoir can only be measured in terms of the increase in maintainable flow which is obtainable at Erbistock. The preparation of the data necessary for the determination of the behaviour of the system for various prescribed flows is described below.

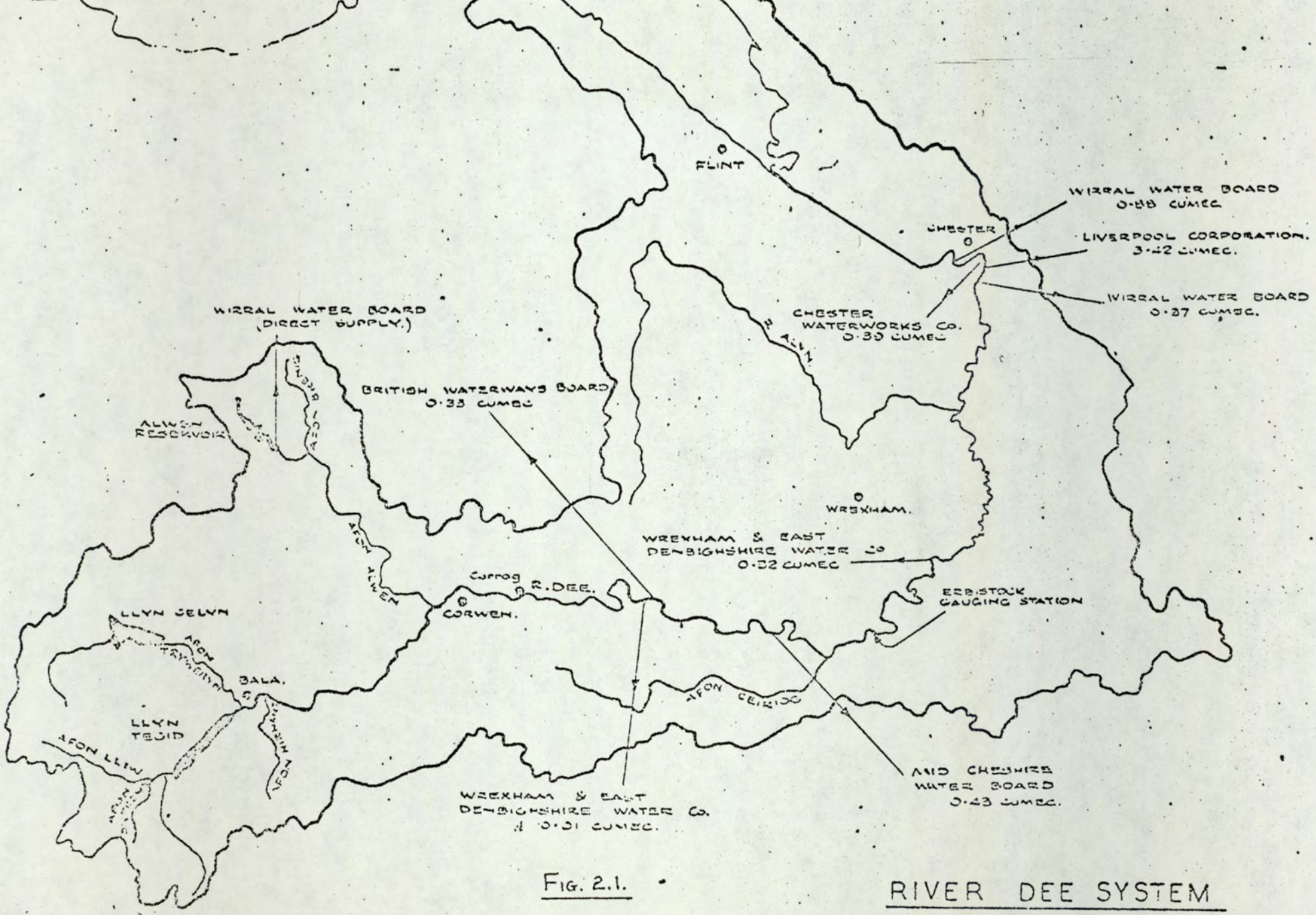


FIG. 2.1.

SCALE :- KILOMETRES 0 5 10 15

RIVER DEE SYSTEM

PRINCIPAL RESERVOIRS AND ABSTRACTIONS FOR PUBLIC WATER SUPPLY.

2.3. Brenig catchment

The proposed reservoir site is situated on the river Brenig, and has a catchment area of 20.2sq.km. This catchment area does not include Llyn Bran, which periodically overflows into the Brenig headwaters, or the stream diversion at Pant-y-Maen taking water from the Brenig catchment into the river Clwyd area. Measurements of these two flows since January 1968 show that the net effect is small, and amounts to an annual outflow of 3 cusec day. The 1916-1950 annual average rainfall over the catchment area is 1308 mm and the average discharge is 0.539 cumecs.

The discharge from the river Brenig, measured at the Pont-y-Rhuddfa gauging station, was recorded from 1922 until 1968 (Table 2.I.) by the Birkenhead Corporation and later the Wirral Water Board, until the Dee and Clwyd River Authority assumed responsibility for its operation. Checks on the theoretical rating curve for the gauge revealed that the curve underestimated the true flow by 0 to 15% at flows less than 1.5 times the average discharge and flows greater than 10 times the average discharge, and consequently the data in the surface water year book series will give underestimates of daily, monthly and annual flows with the percentage error depending upon the individual flow distributions. However, for several reasons, it was decided that the recorded data would suffice for yield calculations without ammendment.

Firstly, the flow data required for yield assessment should be representative of flows which would occur when a large reservoir exists in the catchment area. The evapo-transpiration losses from a large reservoir would exceed similar losses from a heather and moorland area, leading to a reduction in the total runoff from the catchment area, and hence the flows required for design purposes should be less than the

true historic flows.

Secondly, the increasing afforestation of the Brenig valley with evergreen conifers will have enforced a gradual change on the rainfall/runoff relationships for the river, and because of this it is likely that evapo-transpiration and interception losses have been gradually increasing over this period of years, and will probably continue to do so until the trees reach maturity.

Thirdly, analyses of the losses from 1922-1955 showed certain inhomogeneities in the Brenig runoff data, the issue being confused by progressive changes in land management from 1922 onwards.
Consideration of these points reveals that the underestimation of discharges in the historic record is qualitatively compensated for by the fact that the data are to be used to represent a catchment area with different land use and increased evapo-transpiration losses.

2.4. Erbistock data

Daily discharges have been recorded at the Erbistock gauging station since October I937 but the measured flows differ from the natural daily flows which would have occurred if there had been no interference with the river and its tributaries. Therefore, the measured flows have been adjusted, as described below, to allow for the historical increase in control of the river, in order that the natural daily flows could be estimated for the purpose of deriving the effective releases required, over and above the compensation water, to maintain the prescribed flows.

The Alwen reservoir is a direct supply reservoir operative for the whole period of the Erbistock record. However, it is assumed that during low river flow periods, when regulation water releases are required, the compensation water is equivalent to the natural flows from the Alwen catchment area, and that no adjustment to the Erbistock measured flows on a daily basis is warranted.

Upstream of Erbistock, abstractions are taken from the river by the Llantysilio Canal Intake and the Fron Pumping station, which commenced operation in I959. Monthly total abstractions from the river were converted to daily means for each month for which abstraction data existed (from October I950) and the daily means were added to the individual measured daily Erbistock flows. For the period I938-I950 the mean daily values were estimated as being 0.3I cumec (Jan. Feb. March. Oct. Nov. Dec.), 0.34 cumec (April. May) and 0.43 cumec (June).

Another factor which affected the historical Erbistock record was the flow from Llyn Tegid (Bala Lake). Prior to I956 the lake level fluctuated naturally with the discharge through its outlet at Dee Bridge, and was marginally adjusted during dry periods by the Canal Company to provide

slightly increased outflows. No adjustment for these effects was made to the Erbistock recorded flows for this period. As a result of the Bala Lake works, completed in I956, the outlet of the lake was reduced and the lake level controlled by sluices ; under the operating rules during spring and summer, lake retention levels were increased to provide regulation water for maintaining a prescribed flow of 28.9 cumecs at Erbistock. Although individual daily flow corrections at Erbistock were not readily calculable for this period (I956-I964) an approximate monthly adjustment to effective releases during dry summer months when regulation water was being released from storage can be made.

After the construction of Llyn Celyn which was completed by Autumn I964 and commenced refilling at that time, it was impossible to adjust measured Erbistock data other than on a daily basis and this was a laborious task of questionable accuracy. The only justification for this effort would have been to derive data for the dry summer of I969, but in terms of total runoff at Erbistock the July to October dry period was comparable with several other years already included in the I938-I964 data period ; it did not represent an event of the order of a I% or 2% critical period. Accordingly, effective release data were calculated on estimated natural Erbistock flows for the period I938-I964 only.

The quantity of water requiredto be added to the estimated natural Erbistock flow in order to maintain the prescribed flow was determined for each day and these daily values were summed over the calendar months to yield the effective release data. In deriving these data it was assumed that perfect regulation, or addition of exactly the right amount of water, was possible. Monthly effective releases for various prescribed flows are given in tables 2.3. to 2.7.

The effective releases are only the releases required in addition to the

natural flows at Erbistock, which include Brenig and Celyn runoffs, since the compensation water releases from Brenig and Celyn are assumed to be equivalent to the natural runoffs from these catchments in dry periods when regulation is necessary.

For a simulation, the longer the record of flows available, the more accurate will the yield calculations be. However, the existing record of Erbistock flows and the deduced effective releases cover only the period I938-I964 (table 2.2.) but analysis of the nearby Lake Vyrnwy runoff record from I9I0-I964 (table 2.9.) showed that no less than 7 of the 8 most severe droughts lasting 8 months were recorded before gauging started at Erbistock, and it is therefore essential to estimate the river flows as far back as possible in order to derive more reliable estimates of the behaviour of the reservoirs in individual or concurrent dry years.

Since the Brenig runoff record was the longest, being measured since I922, it was decided to try to extend all records back to this time, so that simulations could be carried out with over 40 years data.

It was first necessary to estimate the individual monthly runoffs at Erbistock ; regression equations of the form :

Erbistock runoff = A Vyrnwy runoff + B Brenig runoff

(cusec days I03) (ins) (cusec days I03)

were estimated as follows :

Month	A	B
January	0.6I25	53.59
February	5.2878	2I.23
March	3.9386	3I.08
April	7.46I0	0.00
May	2.9984	32.946I
June	4.2I95	2I.8247

July	4.0388	2I.9383
August	2.73I8	33.8980
September	3.6645	27.3745
October	I.43I5	44.3067
November	2.8295	35.4793
December	4.8350	22.72I3

where C is the correlation coeficient.

These equations gave good agreement between generated and recorded runoff data from I937 to I964 at medium and high runoffs, but they tended to consistently underestimate Erbistock monthly runoff during dry months. To improve the accuracy of the generated data, in all cases where the above equations forecast a monthly Erbistock runoff of less than I0000 cusec days a revised equation relating Erbistock runoff to Brenig runoff only was devised :

Erbistock runoff = 83×Brenig runoff
(cusec days I03) (cusec days I03)

The generated data, incorporating this revision where required, are given in table 2.II.

Since regression analysis yielded very poor results for determining effective releases from the Erbistock monthly flows another method of estimation had to be employed. The effective releases corresponding to the generated Erbistock record and their daily flow distributions within individual months, for use in pumping calculations, were obtained by replacing the individual months in table 2.II by the most similar month in the measured Erbistock record I938-I955.

The procedure used in selecting these months was as follows :

a. Take Erbistock predicted monthly runoff from Table 2.II. for (say) April I938 : 9.95.

b. Select April runoff from Table 2.2. closest to 9.95, i.e. April I946 (9.8I), ensuring that previous months runoff (March I946) in Table 2.2. is not grossly dissimilar to that of March I938 in Table 2.II.

c. If no suitably close April record existed in Table 2.2. use the best record in either of the adjacent months (March or May) in Table 2.2.

d. In autumn, during the month when a drought breaks, runoff is a very poor guide to effective release ; the modified selection rule in such cases is :

   i. If predicted Erbistock runoff for September, October or November exceeds I0000 cusec days and if predicted Erbistock for the previous month is less than I0000 cusec days then selection must take account of previous month's effective release.

   ii. Find from Brenig daily flow records at what time the drought broke midway through the month, and if this is $\frac{1}{4}$, $\frac{1}{2}$, or $\frac{3}{4}$ way through assume that effective release is 25%, 50% or 75% of the effective release for the previous month ; use Table 2.5. for comparing effective releases and select a suitable month from Table 2.2.

The revised synthesised Erbistock record is made up of months from Table 2.2. selected as described, and is itemised in Table 2.I2: the corresponding runoff record is given in Table 2.I3. and its comparison with recorded annual runoff data for the period I938-68 is given in Figure 2.3. The graph highlights wide deviations from recorded data in the years I957-I960 which suggests over recording of discharges, a possibility supported by water balance calculations in those years.

However, apart from these years (I957-I960) there is agreement between synthetic and measured runoffs to within $\mp$ 40,000 cusec days, approximately $\pm$ I0% of the annual average.

The months specified in Table 2.I2. by the use of Table 2.5. corresponding to a prescribed flow of 400 cusecs at Erbistock, are used for the other prescribed flows tried out in the yield calculations.

The effective releases for a maintained flow of 400 cusecs, derived from Tables 2.I2. and 2.5. are given in Table 2.I4; the comparison of synthesised and recorded effective releases is given in Figure 2.4. This shows a standard deviation of $\pm$ 4000 cusec days in the accuracy of individual annual effective releases, which is notunreasonable considering the difficulties in synthesising accurate annual effective releases.

Figure 2.5. shows the comparison of the cumulative probability distribution of annual effective releases ; the synthesised I939-64 data compares acceptably with the recorded I939-64 data, and the final mixed record to be used for simulation purposes (I923-38 and I956-64 synthesised, I939-64 measured data) is also seen to have a similar probability distribution, erring on the conservative side.

The final combined synthetic and recorded Erbistock record is therefore made up as follows :

I923-38 : synthesised data

I939-55 : recorded data

I956 - 64 : synthesised data

The tables giving this data are as follows :

Table 2.I5 : Erbistock monthly and annual runoff (cusec days I03)

Table 2.I6 : Erbistock monthly and annual effective releases, maintained flow 400 cusecs.

Table 2.I7 : Erbistock monthly and annual effective releases,

maintained flow 450 cusecs.

Table 2.18 : Erbistock monthly and annual effective releases maintained flow 500 cusecs.

2.5. Celyn data

Runoff from the Llyn Celyn catchment area has been measured since September I962 ; the reservoir commenced filling in September I964 and since that time natural monthly runoff has been deduced from releases, storage levels and catchwater inflows. The calculated monthly natural runoffs are given in Table 2.8. For the purpose of simulating the performance of the reservoir system over any historic data period it is necessary to estimate the monthly runoffs from the Llyn Celyn catchment area assuming that a reservoir existed in the catchment ; the chosen method of extending the Celyn record back to I923 is that of correlating the Llyn Celyn runoff record for the period I962-I968 with the Brenig runoff record in Table 2.I. and the Lake Vyrnwy runoff record shown in Table 2.9.
The regression equations obtained were of the form :

Celyn runoff = A × Vyrnwy runoff + B × Brenig runoff
(cusec days $I0^3$) (ins) (cusec days $I0^3$)

and the values of A and B for various months were :

Month	A ($I0^3$)	B
January	0.654	+I.298
February	0.69I	
March	0.759	-I.242
April	0.4I0	+I.909
May	0.659	
June	0.785	
July	I.303	-4.I53
August	0.744	
September	0.634	
October	0.760	
November	0.688	
December	0.076	+5.5I0

However, using these coefficients it is possible to obtain negative runoffs in March and July ; a further regression analysis was performed for these months using Vyrnwy only, which yielded coefficients of A = 0.634 and A = I.03 respectively.

The monthly runoff data generated from these equations are given in Table 2.I0; the comparison of the generated and recorded data is good, but this is to be expected because of the limited amount of recorded data available for Celyn.

As an addition to the Celyn flows as calculated above, a further 8% of the individual flows was used to allow for inflows to Celyn from the Hesgin catchwater.

2.6. The Design method

Using the data as obtained above, a computer program was written to simulate the behaviour of the Brenig and Celyn reservoirs over a period of forty years, with specified monthly operational rules.

The required retention levels for Llyn Celyn and the statutory compensation releases for both reservoirs were stipulated.
Since the required maximum storage for the Brenig reservoir was not known, an arbitrary high figure of 100,000 cusec days was used. The use of such a figure allowed spillage but did not allow the reservoir to run dry over the period of record. The size of the existing Celyn reservoir was 31859 cusec days.
Various control rules were tried for each prescribed flow in order to achieve the best sequence of levels over the period, so that the reservoirs were not drawn down for long periods of time or to ensure that one reservoir was not empty while the other was full. Also, large, rapid fluctuations in the reservoir levels were not desirable in view of the proposed use of the reservoirs for recreational and amenity purposes.

For the Brenig/Celyn system, it was obvious that more water could be taken from Celyn than Brenig in the normal operation of the reservoirs, since Celyn has approximately five times the annual inflow of Brenig, but, bearing in mind the amenity use of the reservoirs, it was thought that the percentage contents of each reservoir should be equalised as far as possible.

The first tentative rule to be tried was to treat Brenig as a reserve storage and hence take all effective releases from Celyn unless Celyn fell to the dead storage level of 5259 cusec days, when the remainder of the effective release in any month would be taken from Brenig.

Analysis of the resulting levels showed that at the higher prescribed flows the contents of the two reservoirs were not in balance and that there was a sudden switch over from use of one reservoir storage to the other in the middle of a year.
Consequently, it was decided to make use of Brenig storage more frequently in order to equalise the drawdown in the reservoirs.

It was known that a total of approximately 5600 cusec days of storage could be released from Brenig during the spring and summer seasons and that this would certainly be replaced during the autumn and winter months when high precipitations occured, so that it was decided that from March to September the first 700 cusec days of effective release in any month would be taken from Brenig as long as Brenig was not overdrawn by 5600 cusec days. In order that Celyn would not be drawn down to dead water level before large releases were made from Brenig, a threshold level in Celyn, below which a fraction of the demand would be taken from each reservoir, was applied. Because Celyn refills much faster than Brenig, the threshold level could not be too high or loss of water by spillage would have occurred. Therefore, several levels were tried out in the range of 10000 to 20000 cusec days of storage in Celyn for the various prescribed flows.

From the simulations, it was possible to construct graphs of return period against total combined storage depletion in the reservoirs for each prescribed flow, as shown in Figure 2.2.(a)

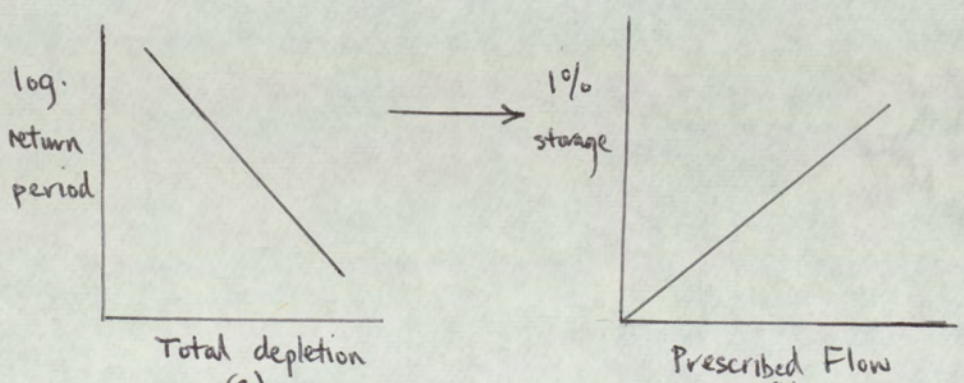


Fig. 2.2.

Extrapolation on these graphs yielded the total depletion and hence required storage, for a return period of once in one hundred years, for each prescribed flow, (Fig. 2.2 (b)).

It was assumed that before I98I a major decision would be taken by the regional planning authorities concerning the development of the River Dee, and so it was decided to construct as a possible first stage a reservoir at Brenig which was capable of supplying the water required for regulation only as far as I98I.

Between I975 and I978 an increased flow of 50 mgd will be required and between I979 and I98I an increase of 73 mgd.

From the graph of storage required against prescribed flow an increase of 50 mgd in the prescribed flow would require an active storage of 2I000 cusec days at Brenig. Since the average annual inflow to Brenig would be only 6500 cusec days, a reservoir at Brenig with a storage much greater than about 20000 cusec days could not be constructed and refilled before I975. Therefore, it was decided that a reservoir of active capacity 2I000 cusec days, capable of maintaining an increase in prescribed flow of 50 mgd at a risk of I% over the period I975-I978 would be constructed at Brenig, and the required increase in flow of 73 mgd between I979 - I98I would be supplied at a risk greater than I% for two years, until a decision had been made either to increase the size of Brenig reservoir or to supply the demand from another source.

The control rule applicable to the two reservoirs for an increase in flow of 50 mgd was found to be to take all effective releases from Celyn until the level fell to I4000 cusec days when two thirds of the effective releases would be taken from Brenig.

Subsequently, Binnie & Partners developed a daily simulation model for the system and it was found that this monthly policy was still applicable with minor modifications.

FIGURE 2.3

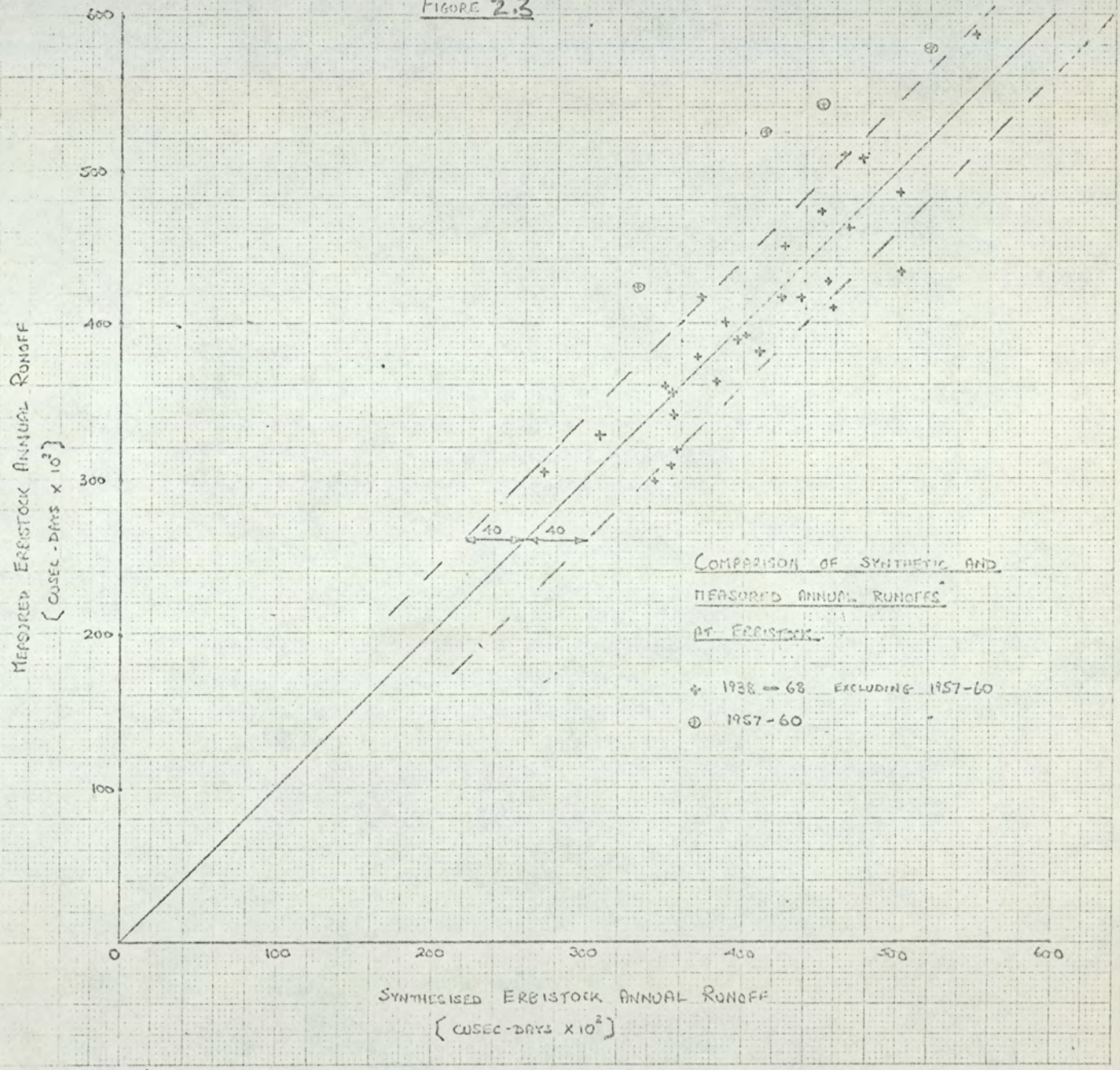
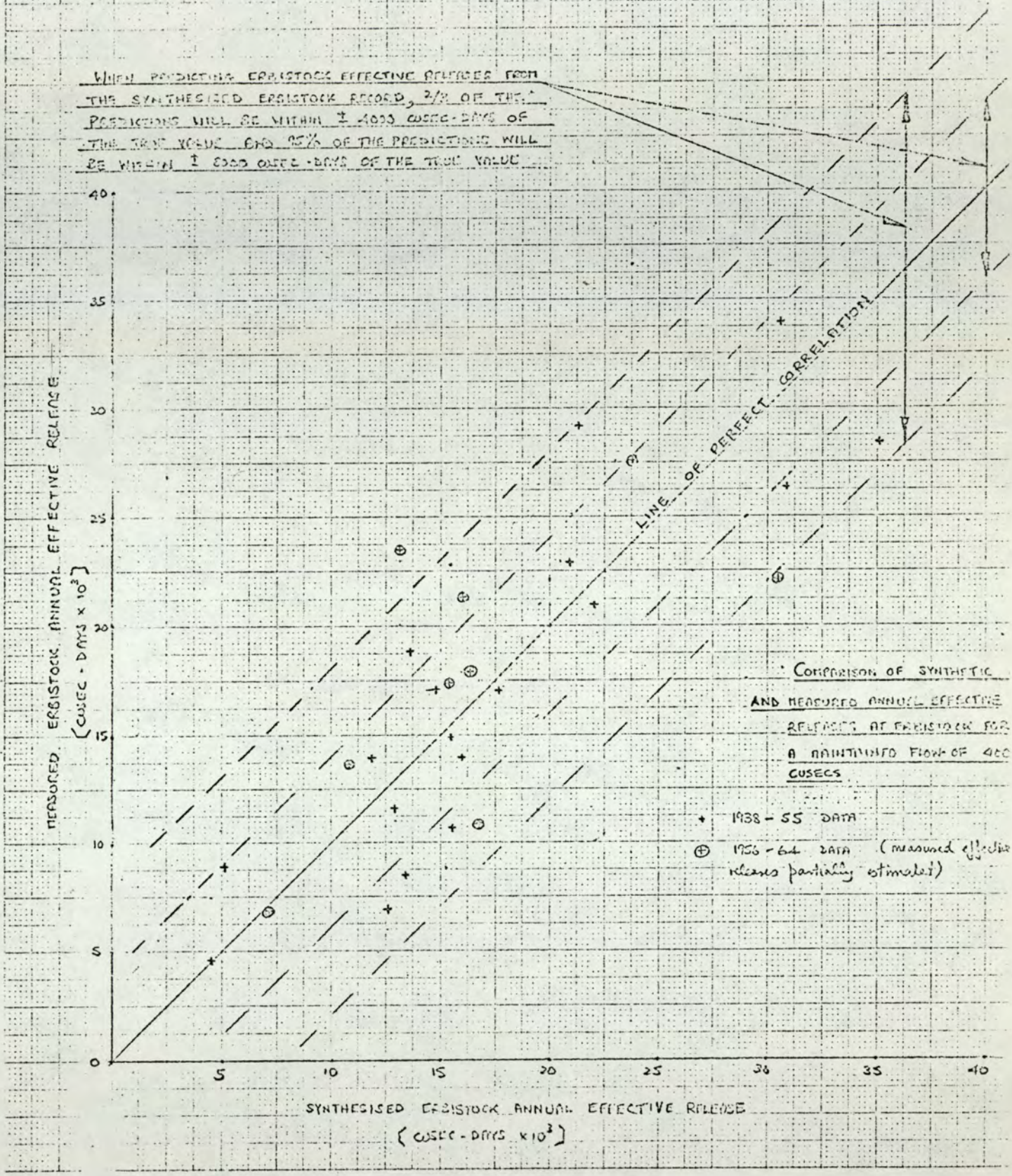


Fig. 2.3

FIGURE 2.4



COMPARISON OF SYNTHETIC AND MEASURED ANNUAL EFFECTIVE RELEASES AT ERBISTOCK FOR A MAINTAINED FLOW OF 400 CUSECS

+ 1938 - 55 DATA

⊕ 1956 - 64 DATA (measured effective releases partially estimated)

Fig. 2.4.

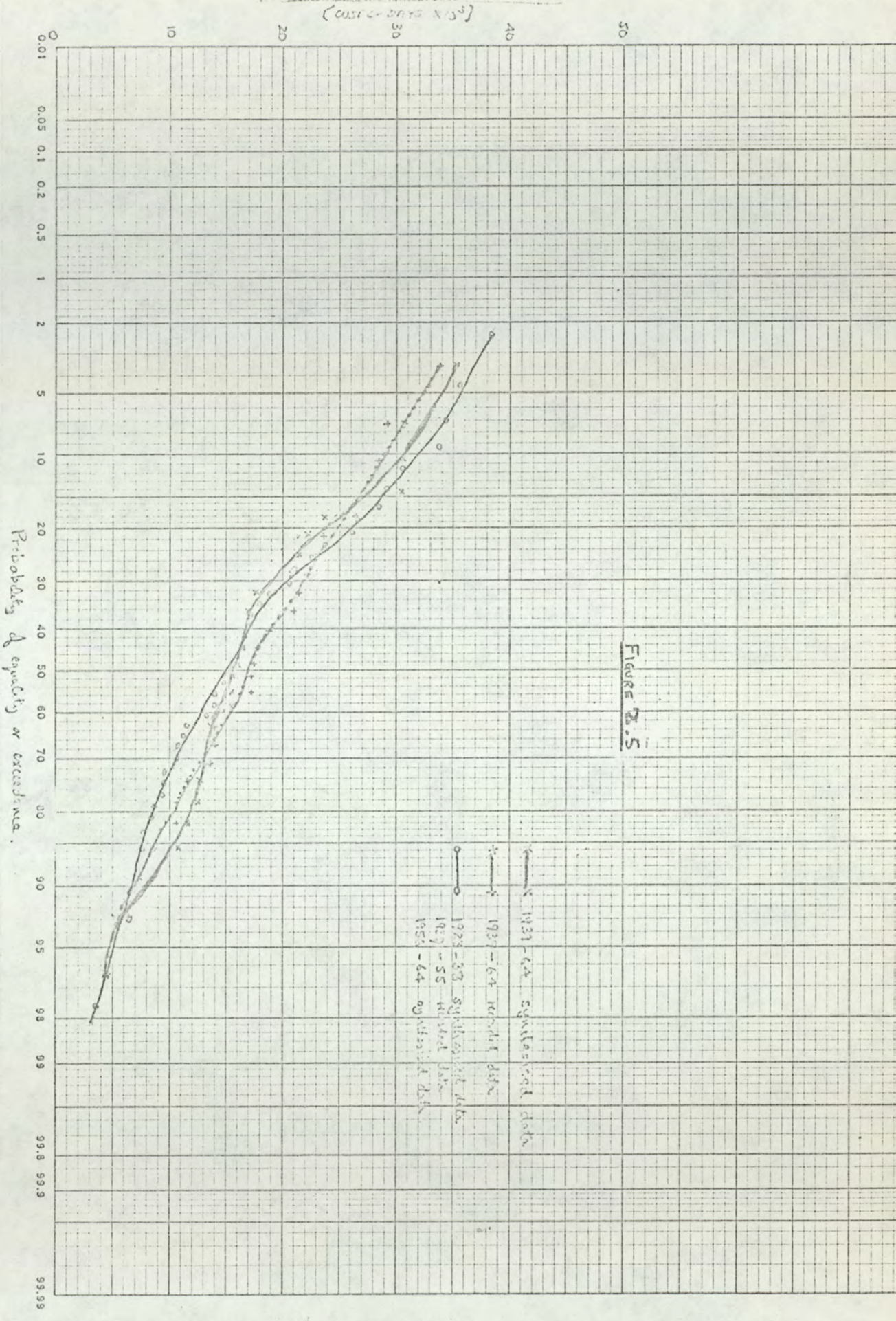

ERSISTOCK ANNUAL MAXIMUM RUNOFF
FOR 400 DAYS MAINTAINED FLOW
(COST OF DAMS X 10⁶)
0
10
20
30
40
50
0.01
0.05
0.1
0.2
0.5
1
2
5
10
20
30
40
50
60
70
80
90
95
98
99
99.8
99.9
99.99
Probability of equality or exceedance.
1931-64 synthesized data
1923-30 synthesized data
1931-64 recorded data
1950-55 recorded data
1950-64 synthesized data
FIGURE 2.5

Fig. 2.5

Table 2.I.

Measured Monthly Runoffs (cusec days × I000) - River Brenig at Pont-y-Rhuddfa

Year	Jan.	Feb	Mar	Ap.	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec	Ann.
I922												I.2I	
I923	.78	I.74	.6I	.I5	.80	.22	.09	.29	.5I	I.52	I.32	I.I3	9.I6
I924	.95	.23	.24	.33	.7I	.5I	.30	.86	I.04	.90	.66	I.I8	7.9I
I925	.85	I.55	.45	.33	.38	.I9	.04	.05	.53	.95	.72	I.2I	7.25
I926	I.20	.60	.50	.I2	.39	.I2	.48	.33	.62	.75	I.42	.42	6.95
I927	.97	.46	I.20	.5I	.24	.45	.40	I.24	I.0I	I.25	I.34	.37	9.44
I928	I.7I	I.44	.32	.28	.I3	.44	.3I	.I8	.I8	I.32	I.98	.78	9.07
I929	.45	.29	.II	.07	.I4	.07	.05	.39	.I7	.92	I.47	2.06	6.I9
I930	I.54	.4I	.66	.68	.I8	.07	.37	.89	.64	I.26	I.48	I.20	9.38
I93I	.96	I.03	.34	.7I	.52	.60	.I2	.94	.99	.29	I.49	.58	8.57
I932	I.38	.I4	.36	I.08	.5I	.II	.37	.38	.63	I.38	.67	.7I	7.72
I933	.89	.9I	.74	.II	.I0	.27	.I5	.06	.05	.34	.36	.2I	4.I9
I934	.87	.I5	.49	.30	.48	.I0	.05	.25	.II	.94	.88	I.20	5.82
I935	.76	I.I7	.46	.52	.I6	.23	.20	.07	.5I	I.65	I.37	.74	7.84
I936	I.23	.68	I.II	.48	.27	.89	.5I	.I6	.49	.53	I.34	I.20	8.89
I937	.97	I.I3	I.38	.9I	.I8	.I2	.08	.04	.04	.I5	.22	.65	5.87
I938	I.25	.40	.27	.I2	.08	.54	.47	.23	.I5	I.I9	I.04	I.0I	6.75
I939	I.93	.7I	.97	.4I	.23	.I4	.8I	.33	.08	.33	I.36	.94	8.24
I940	.23	.86	.67	.4I	.36	.I0	.I3	.06	.20	.80	I.66	.54	6.02
I94I	.38	I.67	I.02	.38	.2I	.I6	.06	.23	.II	.8I	.63	.70	6.36
I942	.86	.88	.53	.48	.I8	.I0	.I4	.50	.39	.65	.28	I.05	6.04
I943	I.I8	.94	.I5	.I7	.63	.23	.20	.24	.93	.78	.73	.6I	6.79
I944	I.I8	.44	.36	.20	.I0	.I8	.I5	.08	.77	I.05	I.70	I.20	7.4I
I945	.82	I.23	.27	.42	.38	.34	.I9	.09	.I4	.77	.25	.83	5.73
I946	I.I2	I.58	.34	.I2	.II	.48	.I4	.69	.90	.26	I.07	I.I8	7.99
I947	I.05	.I3	I.84	.83	.46	.I6	.II	.05	.05	.04	.94	.57	6.23

Year	Jan	Feb	Mar	Ap	My	Jn	Jy	Aug	Sep	Oct	Nov	Dec	Ann
I948	2.27	.88	.I8	.36	.09	.43	.I5	.37	.43	.37	.65	.93	7.II
I949	.99	.32	.50	.85	.22	.I5	.06	.I3	.04	.72	I.05	I.53	6.56
I950	.45	I.27	.45	.63	.25	.07	.08	.38	I.36	.78	.95	.99	7.66
I95I	I.0I	.87	I.I5	.62	.45	.I3	.07	.33	.63	.24	I.6I	I.47	8.58
I952	I.09	.77	.25	.20	.33	.08	.04	.07	.4I	I.I0	.72	.97	6.03
I953	.34	.65	.24	.43	.I9	.27	.I7	.I4	.6I	.25	.88	.34	4.5I
I954	.55	.73	.64	.27	.45	.39	.35	.8I	.4I	I.60	I.55	I.I8	8.93
I955	.75	.39	.6I	.2I	.80	.33	.I0	.04	.04	.I7	.34	.67	4.45
I956	.96	.2I	.43	.I9	.II	.06	.3I	I.I5	.58	.55	.30	.75	5.60
I957	.97	.87	.42	.I2	.II	.04	.30	I.05	I.05	.66	.74	.58	6.9I
I958	.85	I.35	.33	.I2	.35	.33	.37	.54	.88	.84	.49	.43	6.88
I959	I.37	.I7	.27	.87	.3I	.I0	.I4	.05	.03	.30	.87	I.45	5.93
I960	I.54	I.0I	.46	.43	.I0	.05	.07	.I6	.83	.88	I.72	I.I8	8.43
I96I	I.07	.87	.I8	.36	.48	.06	.I7	.32	.I7	.88	.64	I.07	6.27
I962	I.53	.60	.23	.66	.33	.II	.07	.40	.52	.28	.80	.8I	6.34
I963	.I3	.08	I.2I	.62	.38	.54	.3I	.I5	.32	.38	I.4I	.27	5.80
I964	.22	.27	.58	.20	.28	.I0	.II	.2I	.I2	.48	.4I	I.8I	4.79
I965	I.49	.27	.63	.54	.40	.28	.29	.I6	.95	.49	.67	2.I6	8.33
I966	.56	.83	.46	.80	.29	.I4	.I4	.53	.30	.86	.93	I.69	7.53
I967	.55	.84	.47	.29	.69	.I4	.08	.I6	.49	I.87	.98	I.25	7.8I
I968	I.60	.38	.6I	.39	.65	.22	.6I	.I2	.80	.97	.76	.6I	7.72

Erbistock Gross Monthly Runoff (cusec days x 10^3)

Table 2.2

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec
37										5.30	12.71	38.68
38	63.54	27.96	20.02	8.62	4.98	25.83	26.72	20.29	8.41	59.93	60.91	54.28
39	93.56	39.49	49.19	20.80	9.55	6.99	38.35	36.36	7.01	14.07	70.03	59.58
40	27.42	56.27	36.85	26.46	18.22	6.22	12.18	4.81	11.76	29.55	95.36	37.11
41	19.44	68.94	53.31	21.10	12.67	10.85	4.15	10.60	6.93	32.55	27.82	35.76
42	39.94	37.50	23.88	31.89	17.29	11.10	9.95	19.57	19.80	40.60	13.92	52.89
43	73.62	60.57	9.23	13.24	39.82	23.65	18.25	19.58	47.40	40.96	36.89	34.13
44	66.62	26.74	17.04	13.29	8.71	16.61	11.86	8.59	33.32	56.28	83.82	73.55
45	34.98	86.38	17.83	25.75	18.37	26.44	11.84	7.09	16.65	39.87	17.22	38.92
46	56.68	106.58	27.25	9.81	6.74	25.87	13.09	29.26	67.30	22.41	71.81	70.67
47	76.54	10.10	115.00	56.69	31.69	12.78	15.78	6.33	4.72	4.81	46.13	36.38
48	122.51	59.80	17.83	28.77	8.01	26.90	13.86	26.36	28.60	20.92	40.35	56.07
49	63.69	23.42	27.11	45.86	13.05	13.50	3.60	8.29	3.59	43.69	62.10	85.23
50	31.49	82.35	32.50	30.16	19.55	5.37	15.81	37.02	75.87	46.97	42.47	53.77
51	58.59	55.61	63.41	43.12	17.57	8.14	4.35	11.64	33.70	13.00	92.08	83.51
52	69.79	38.95	21.22	12.75	16.76	9.04	5.01	11.80	14.62	56.63	44.16	55.53
53	21.57	39.65	18.43	28.10	17.33	13.31	25.08	19.96	38.85	20.32	64.23	21.58
54	35.31	47.74	43.90	22.66	17.20	31.86	27.31	41.26	35.19	99.67	98.85	86.09
55	43.21	29.85	35.96	20.15	41.08	32.28	11.59	4.50	4.41	12.34	22.73	45.57
56	60.45	17.53	28.50	9.18	8.69	6.00	16.61	49.88	39.06	37.63	23.74	61.13
57	71.92	68.80	46.38	11.32	10.53	9.72	45.01	66.90	74.61	49.35	51.99	66.30
58	60.14	79.49	23.72	8.68	30.05	31.05	36.27	49.17	73.89	71.64	25.37	35.37
59	89.75	13.61	20.72	47.40	20.39	10.09	15.17	9.17	2.68	23.07	52.44	116.37
60	96.04	72.62	35.59	30.06	7.10	5.59	9.15	14.62	42.18	54.13	115.77	74.43
61	63.46	57.57	15.44	27.90	26.95	4.70	8.44	22.71	14.67	52.95	39.90	64.91
62	80.51	39.87	13.32	44.01	22.90	8.87	6.02	31.25	34.95	19.53	39.33	51.80
63	17.83	8.26	74.49	42.36	23.78	17.62	20.78	14.65	20.77	26.28	90.26	21.37
64	15.85	14.62	29.23	13.83	21.31	6.74	12.34	9.31	7.67	28.22	23.34	115.66
65	101.09	19.79	47.40	35.70	32.98	20.01	17.11	15.14	39.60	28.43	36.00	115.94
66	35.34	52.92	28.21	39.30	23.41	14.64	9.39	18.14	16.89	38.13	45.00	88.35
67	36.89	54.32	33.79	15.81	42.47	11.67	8.03	15.81	33.00	102.77	46.89	61.32
68	85.28	24.30	54.31	8.68	32.61	14.67	27.25	7.72	37.68	47.71	34.91	42.05
69	58.71	41.24	39.47	30.52	47.49	21.15	5.54	9.27	7.41	8.62	69.43	58.94

Oct. 1937 to Sept 1950 : Surface Water Year Book Data with additional correction for Canal and Alwen Reservoir.

Oct. 1950 to Sept 1953 : Surface Water Year Book Data with additional correction for Alwen Reservoir.

Post 1953 : Surface Water Year Book Data

Table 2.3.

Effective Release (Cusec days)

Prescribed Flow - 300 cusecs.

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec	Total
38				2348	4390	270	24	-	I037	44	-	-	8,II3
39					I02I	2226	I2	48	2026	I4II	-	-	6,744
40	72	-	-	-	-	2360	I630	4208	257I	374	-	-	II,2I5
4I				I3	2633	I63I	5392	2290	3I30	908	44	-	I6,04I
42		6	-	57	I788	II82	2630	-	29	-	-	-	5,692
43	-	-	I053	697	5	-	284	8	-	-	-	-	2,047
44	-	-	-	5	I50I	775	5I9	I949	-	-	-	-	4,749
45	-	-	-	I98	-	-	33I	3053	597	470	I4	47	4,7I0
46	-	-	-	658	3I56	-	469	-	-	-	-	-	4,283
47	-	252	-	-	-	I38	I7I	4I30	4986	4344	I096	-	I5,II7
48	-	-	6	-	I869	I33	267	499	-	I4	-	-	2,788
49	-	-	-	-	I373	I742	5489	2630	5347	2760	-	-	I9,34I
50	-	-	-	-	337	3768	I388	II8	-	-	-	-	5,6II
5I	-	-	-	-	-	I792	4893	2357	-	427	-	-	9,469
52	-	-	-	208	57I	I704	4257	I770	706	-	-	-	9,2I6
53	-	-	8I7	-	-	73	I54	228	-	28	-	-	I,300
54	-	-	-	328	96I	-	-	-	-	-	-	-	I,289
55	-	-	-	59	-	-	I726	5368	5275	I67I	38	I68	I4,305
56	-	I40	-	990	2300	3452	9I5	-	-	-	-	-	7,797
57	-	-	-	980	I83I	I362	2I4	-	-	-	-	-	4,387
58	-	-	600	I9I0	996	-	-	-	-	-	-	-	3,5I6
59	-	-	-	-	-	474	775	II88	5556	3743	-	-	II,736
60	-	-	-	4II	2874	3749	3082	I200	-	-	-	-	II,3I6
6I	-	-	794	35	II04	48I6	356I	744	460	I44	-	-	II,658
62	-	-	396	-	400	I26I	3740	I092	-	5I6	-	-	7,405
63	-	I085	I45	-	I24	I953	53I	II86	408	400	-	-	5,832
64	I5	94	-	26I	709	2493	2339	2589	33II	569	I64	-	I2,544

Table 2.4.

Effective Release (Cusec days)

Prescribed Flow - 350 cusecs

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec	Total
38	-	-	-	3398	5I88	656	I62	-	2235	94	-	-	II,733
39	-	-	-	-	I949	3598	62	286	3376	206I	-	-	II,332
40	342	-	-	-	7I	375I	2440	5657	3445	524	-	-	I6,230
4I	-	-	-	I02	3883	2494	6942	3070	420I	I258	84	-	22,034
42	-	II5	-	248	2777	2046	3730	45	I42	-	II6	-	9,2I9
43	-	-	2080	I377	I02	7	945	I9I	8	-	-	-	4,7I0
44	-	-	75	325	2724	I345	I300	3II5	-	-	-	-	8,884
45	-	-	24	597	23	60	908	4503	I259	I2I2	233	I47	8,966
46	-	-	-	I475	4545	-	I006	-	-	95	28	-	7,I49
47	-	674	9	-	-	493	465	545I	6457	5894	I546	-	20,989
48	-	-	I03	92	2926	233	634	872	-	I59	-	-	5,0I9
49	-	-	22	99	2558	2482	7039	37I3	6847	36I0	-	-	26,370
50	-	-	-	-	I020	5II8	20I0	337	-	-	-	-	8,485
5I	-	-	-	-	-	2942	6443	3527	-	I069	-	-	I3,98I
52	-	-	62	662	I273	2844	5728	2663	I5I8	-	-	-	I4,750
53	-	-	I553	I8	66	363	430	600	I7	256	-	-	3,303
54	-	-	994	I824	-	-	3	-	-	-	-	-	2,82I
55	-	-	-	265	-	-	2782	69I8	6775	2373	368	439	I9,920
56	-	376	-	2I38	3490	4950	I5I2	-	-	-	-	-	I2,466
57	-	-	-	I7I4	2732	2424	29I	-	-	-	-	I3	7,I74
58	-	-	600	2938	I366	-	-	-	-	-	-	-	4,904
59	-	-	-	-	-	I094	I6I3	I906	7056	4696	-	-	I6,365
60	-	-	-	535	43I3	5085	4I37	I924	-	-	-	-	I5,994
6I	-	-	I604	I33	I7II	63I6	4767	I093	970	583	-	-	I7,I77
62	I8	-	II68	-	347	24I4	5290	I542	-	927	-	-	II,796
63	-	2035	34I	-	324	297I	729	I767	604	400	-	I28	9,299
64	325	370	94	8I0	I072	3927	3I42	3676	45II	869	528	-	I9,324

Table 2.5.

Effective Release (cusec days)

Prescribed Flow 400 cusecs

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec	Total
38	-	60	I66	4495	6698	II58	4I6	367	363I	I44	-	-	I7,I35
39	-	-	-	-	3275	5036	I96	950	486I	280I	-	-	I7II9
40	I506	-	-	-	465	5229	3332	7I07	4367	970	-	-	22,876
4I	37I	-	-	390	5I33	3484	8492	3954	5367	I650	2I7	I0	29,068
42	-	379	-	543	3866	3074	4830	347	408	-	5I8	-	I3,965
43	-	-	3339	2I39	476	85	I767	850	-	-	-	-	8,655
44	-	-	3I5	982	3974	I964	2324	4449	-	-	-	-	I4,008
45	-	-	224	II06	267	384	I748	5953	I904	2II7	965	247	I4,9I5
46	-	-	-	25I9	598I	-	I693	-	-	327	2I0	-	I0,730
47	-	I569	28I	-	-	II99	I096	690I	7957	7444	I996	-	28,443
48	-	-	5I0	362	427I	382	II54	I332	-	45I	-	-	8,462
49	-	-	I9I	3I7	3858	3247	8589	4893	8347	4460	-	-	33,902
50	-	-	-	-	I70I	6468	2728	738	-	-	-	-	II,635
5I	-	-	-	-	5I	4I43	7993	4727	-	I897	-	-	I8,8II
52	-	-	20I	I28I	2067	4I49	7228	3594	2540	-	-	-	20,960
53	I03	-	2455	208	389	847	8I4	I085	I8I	640	-	-	6,722
54	22	-	-	I703	2830	-	98	-	-	-	-	-	4,653
55	-	-	-	663	-	-	39I4	8468	8275	3207	I040	834	26,40I
56	-	722	33	3448	4773	6450	2358	-	-	-	-	-	I7,784
57	-	-	-	2690	3792	3576	44I	-	-	-	-	247	I0,746
58	-	-	705	42I6	I766	-	-	-	-	-	-	68	6,755
59	-	I92	74	-	9	2208	258I	2907	8556	5696	-	-	22,223
60	-	-	-	750	5768	6525	5424	2872	-	-	-	-	2I,339
6I	-	-	2482	333	24I9	78I6	6039	I47I	I697	I273	-	-	23,530
62	82	-	2220	-	787	37I2	6840	2I50	29	I597	-	-	I7,4I7
63	75	3075	54I	-	538	4I28	I056	247I	97I	400	-	407	I3,662
64	966	630	426	I6I3	I768	54I8	405I	4937	5740	II69	I008	-	27,726

Table 2.6.

Effective Release (cusec days)

Prescribed Flow 450 cusecs

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec	Total
38	21	244	593	5641	8248	11725	885	667	5081	194	-	-	23,299
39	-	-	-	120	4646	6503	446	1319	6361	3651	-	-	23,046
40	2127	-	-	-	1055	6729	4438	8657	5367	1241	-	-	29,614
41	930	-	-	710	6383	4544	10042	4929	6615	2097	577	99	36,926
42	-	743	136	979	5004	4174	5945	727	750	34	1358	-	19,850
43	-	-	4716	3017	996	245	2667	1500	-	-	-	-	13,141
44	-	-	568	1860	5338	2665	3460	5800	-	-	-	-	19,691
45	-	-	715	1736	773	842	2794	7403	2554	3084	1788	394	22,083
46	-	-	-	3727	7459	-	2651	112	-	754	540	-	15,243
47	-	2650	567	-	-	2016	1663	8425	9457	8944	2446	34	36,202
48	-	-	1169	780	5709	532	1948	1783	58	822	-	-	12,801
49	-	81	430	591	5203	4110	10139	6175	9847	5310	-	-	41,886
50	73	-	-	-	2505	7853	3535	1288	-	-	-	-	15,254
51	-	-	-	-	351	5401	9543	5940	-	2850	-	-	24,085
52	-	-	485	2089	2935	5299	8765	4560	3604	-	-	-	27,737
53	328	-	3473	497	829	1636	1248	1619	481	1269	-	106	11,486
54	134	-	-	2419	3917	51	520	-	12	-	-	-	7,053
55	-	-	3	1156	1	-	5064	10018	9775	4159	1608	1278	33,062
56	-	1242	224	4884	6085	7950	3313	-	14	-	-	-	23,712
57	-	-	-	3790	4919	4787	591	-	-	17	22	587	14,713
58	-	-	1003	5596	2166	-	-	-	-	-	-	432	9,197
59	-	862	140	2	203	2241	3663	4097	10056	6696	-	-	27,960
60	-	-	35	1116	7294	8015	6724	3972	20	-	-	-	27,176
61	-	-	3452	609	3192	9316	7359	2123	2736	2100	-	-	30,687
62	196	1	3466	23	1378	5062	8390	2800	98	2410	-	-	23,824
63	313	4420	741	-	818	5328	1497	3316	1465	439	-	834	19,171
64	1791	1199	952	2509	2615	6918	4934	6237	7038	1511	1518	-	37,222

Table 2.7.

Effective Release (cusec days)

Prescribed Flow 500 cusecs

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec	Total
38	105	494	1127	6840	9798	2325	1215	1167	6549	244	-	-	29,864
39	-	-	-	486	6046	8003	696	1694	7861	4585	-	68	29,439
40	3118	-	-	-	1886	8229	5634	10207	6408	1671	-	-	37,153
41	1668	-	-	1093	7633	5660	11192	6021	8886	2542	1108	307	46,110
42	-	1163	411	1479	5628	5336	7110	1283	1236	178	2458	-	26,282
43	-	-	6260	4003	1601	445	3609	2341	-	-	-	-	18,259
44	-	-	953	2882	6767	3615	4610	7200	-	-	-	-	26,027
45	-	-	1433	2515	1392	1398	4054	8853	3254	4128	2732	460	30,219
46	-	-	-	5113	9009	87	3764	481	-	1298	1078	-	20,830
47	-	3980	1067	39	10	3019	2479	9975	10957	10554	2896	266	45,254
48	-	-	2146	1438	7259	761	2990	2325	130	1247	-	-	18,296
49	-	275	724	939	6557	5067	11689	7475	11347	6160	-	-	50,233
50	267	44	25	-	3441	9260	4477	1953	-	-	-	-	19,467
51	-	-	-	44	1049	6701	11093	7190	41	3923	-	-	30,041
52	-	-	1047	3068	3856	6589	10315	5656	4733	11	-	-	35,275
53	164	-	4588	939	1423	2736	1844	2226	829	2047	-	326	17,572
54	369	42	-	3227	5033	339	1041	-	116	-	-	-	10,167
55	-	50	64	1738	51	-	6215	11568	11275	5337	2228	1728	40,254
56	-	2114	624	6384	7484	9450	4313	70	267	-	49	-	30,755
57	-	-	-	4992	6191	6037	741	-	-	117	149	1237	19,464
58	-	-	1446	6046	2568	-	-	-	-	-	70	922	11,052
59	-	1779	499	90	621	3691	4851	5479	11556	7696	-	-	36,262
60	-	-	183	1529	8844	9515	8056	5172	107	-	-	-	33,306
61	-	-	4541	1033	4051	10816	8709	2650	3697	3000	-	-	38,497
62	394	75	4806	140	2004	6417	9940	3498	257	3509	-	65	30,905
63	987	5920	941	-	1184	6528	1947	4287	2015	580	-	1480	25,769
64	2712	1916	1683	3504	3515	8418	6055	7581	8387	1926	2068	-	47,765

Table 2.8.

Llyn Celyn Inflows

(Cusec days x 10^3)

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec	Annual Total
62									3.91	1.71	3.48	4.69	
63	0.56	0.40	6.84	3.83	3.23	2.48	1.96	2.33	2.50	3.24	8.15	1.34	36.86
64	1.75	1.21	1.72	1.53	2.90	0.79	2.83	1.91	1.31	3.02	3.56	11.22	33.75
65	7.83	0.78	2.96	2.83	2.60	2.61	2.27	2.27	4.07	2.45	3.45	13.96	48.08
66	3.75	6.08	3.05	3.83	2.93	3.02	1.60	2.12	2.24	3.62	4.54	9.48	46.26
67	3.64	5.39	2.55	1.58	4.29	0.91	1.75	2.65	4.85	11.62	3.42	7.42	50.07
68	9.01	2.16	6.16	1.56	2.60	1.68	2.43	0.84	4.05	5.48	2.57	2.95	41.49
69	5.02	2.56	2.61	2.62	2.95	1.44	0.44	1.39	0.89	1.41	6.41	5.05	32.79

Table 2.9

Lake Vyrnwy runoff record 1910-1969

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec
10	6 .05	8.96	3.47	3.14	1.52	1.25	2.42	5.52	0.84	2.62	5.54	9.50
11	2.93	5.07	3.15	2.82	1.81	1.04	0.26	1.31	2.42	4.71	8.73	8.27
12	5.87	4.42	8.54	1.19	0.44	2.85	4.08	7.40	1.24	5.66	5.36	11.90
13	3.42	5.00	8.71	6.20	5.37	3.26	0.30	0.73	1.28	4.06	6.60	4.96
14	7.65	8.80	7.86	3.20	2.59	0.63	1.69	3.54	1.87	1.10	7.84	12.45
15	8.89	9.25	3.54	1.93	2.01	0.21	1.36	3.44	0.50	1.72	3.95	12.38
16	8.87	6.25	3.74	4.28	2.38	1.79	2.04	1.16	2.23	11.26	7.02	4.43
17	3.34	1.71	3.95	3.32	1.30	1.70	0.58	7.49	3.13	7.56	5.58	3.23
18	6.63	8.47	2.11	2.23	2.10	0.73	2.92	0.92	41.47	6.37	4.45	11.12
19	6.84	3.80	6.73	5.05	1.79	0.53	0.20	1.57	2.64	2.41	4.08	11.47
20	9.70	7.72	6.11	8.23	5.80	1.30	7.42	2.83	1.71	2.92	4.03	7.06
21	10.85	1.04	6.82	1.63	1.20	0.29	0.34	4.00	1.57	3.10	4.20	9.25
22	7.82	8.54	5.23	2.78	1.50	0.61	4.52	2.56	3.93	1.50	3.57	7.76
23	5.71	11.77	0.58	2.98	4.68	1.37	1.32	6.15	5.51	10.40	6.00	6.75
24	6.96	0.98	2.58	3.61	6.74	3.67	4.35	5.53	5.99	7.34	4.83	8.77
25	8.63	10.34	3.13	2.58	3.75	0.66	0.22	1.46	2.88	7.31	3.64	7.22
26	9.79	0.96	3.10	1.75	2.82	1.71	3.19	3.61	2.62	5.39	10.51	1.68
27	6.75	4.25	5.58	3.17	1.39	4.44	3.06	8.56	4.94	3.82	8.05	2.76
28	14.21	10.26	3.88	2.67	0.59	3.89	2.79	4.40	1.15	8.06	13.94	5.03
29	3.14	1.98	1.01	0.58	2.59	1.23	0.52	4.73	1.37	7.50	13.30	14.20
30	10.01	1.71	3.56	4.03	2.43	1.18	3.12	5.04	5.67	8.52	8.78	7.48
31	6.51	7.72	1.97	4.40	4.83	4.85	1.22	5.27	2.68	1.53	11.80	3.82
32	11.91	0.49	3.14	6.22	3.88	0.66	3.44	1.34	4.97	8.33	4.98	4.57
33	5.44	6.45	5.98	0.96	0.83	1.02	2.02	0.37	0.21	3.07	2.19	1.21
34	8.61	0.75	5.03	2.85	3.33	0.94	0.29	2.46	3.06	7.39	3.36	10.60
35	3.20	9.80	2.86	5.09	0.73	3.31	0.95	0.33	6.72	10.64	7.49	5.06
36	8.28	3.16	5.02	2.71	0.94	3.36	6.69	1.57	4.68	4.17	8.00	8.86
37	9.28	9.77	5.41	3.72	1.10	1.86	1.01	0.45	0.59	1.52	2.14	6.16
38	11.77	3.14	2.39	0.71	1.51	4.77	4.31	3.75	0.98	11.94	9.48	6.59
39	11.67	5.91	5.69	2.57	0.93	1.25	8.73	2.45	0.77	1.89	11.65	6.94
40	1.57	7.75	4.94	4.08	1.95	0.43	1.61	0.43	2.50	7.61	15.15	5.93
41	1.60	9.67	5.05	2.31	2.76	0.99	0.49	3.58	1.45	6.82	4.68	6.08
42	6.07	3.92	3.92	4.83	3.79	0.78	2.0	4.51	3.34	6.47	1.29	6.62
43	11.48	5.89	1.29	1.87	5.37	5.36	2.84	4.28	6.54	5.38	4.41	4.16

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec
44	9.69	3.15	1.63	1.67	1.34	2.77	2.57	1.24	5.90	9.56	11.64	8.28
45	6.26	10.87	2.55	3.75	3.26	4.93	2.24	2.15	3.93	7.03	1.13	7.20
46	9.97	12.47	3.13	0.77	0.64	4.20	1.61	5.89	8.78	2.37	10.37	7.26
47	7.32	0.79	14.18	6.09	3.00	1.85	2.01	0.46	0.81	0.67	8.44	5.88
48	17.76	7.33	3.14	3.09	1.50	4.97	1.80	5.85	4.72	2.90	5.17	6.90
49	5.80	3.43	2.49	6.46	2.80	1.96	0.21	1.18	0.51	7.92	9.09	10.61
50	3.58	11.52	4.27	4.21	1.74	0.68	4.20	6.90	11.30	4.86	6.05	4.69
51	8.03	6.94	7.33	5.14	2.31	0.90	0.73	3.83	5.73	1.70	12.30	11.38
52	7.27	4.55	3.80	2.09	2.33	1.70	1.07	2.77	2.05	10.33	4.39	7.03
53	2.40	4.64	4.08	3.84	2.30	1.99	5.17	3.42	5.86	2.47	9.78	2.75
54	4.49	6.00	5.74	2.44	2.81	3.95	4.09	4.79	6.90	14.61	11.79	9.22
55	5.42	3.30	4.27	2.68	7.24	4.79	1.21	0.33	0.39	1.85	3.58	7.63
56	8.74	1.37	3.33	0.92	1.22	0.97	3.86	7.13	4.93	4.10	2.96	8.42
57	10.11	7.45	6.89	0.86	1.53	0.53	6.22	6.60	8.63	6.15	4.50	5.57
58	7.23	9.33	2.24	1.58	5.81	3.06	2.75	4.27	8.83	5.05	2.49	4.89
59	8.60	1.12	3.38	5.29	1.20	1.20	2.82	0.75	0.19	4.92	6.45	14.92
60	10.23	7.80	3.91	4.38	0.87	1.00	3.43	4.21	5.66	6.69	13.61	9.86
61	7.90	6.56	1.49	4.82	3.34	0.58	1.02	3.66	3.32	8.61	6.04	7.87
62	9.81	8.21	2.26	8.07	3.17	0.87	0.94	6.74	5.30	2.29	4.91	6.25
63	0.85	0.61	11.30	5.67	4.33	2.56	2.48	2.61	3.63	3.14	12.27	2.51
64	2.32	2.25	3.34	3.03	3.69	1.42	2.44	1.34	1.60	3.94	4.72	14.88
65	9.16	1.31	4.95	3.63	4.14	4.29	2.40	2.54	5.65	3.38	5.11	15.71
66	4.60	9.19	4.13	6.35	4.15	2.35	1.71	3.25	3.04	4.81	6.26	11.53
67	4.51	7.26	3.78	2.34	7.17	1.12	1.80	4.13	7.80	15.32	4.06	7.79
68	10.44	2.75	9.10	2.82	4.12	2.31	3.92	1.55	7.81	7.50	4.41	5.11
69	7.31	3.52	4.85	4.03	5.49	1.85	0.61	1.01	1.31	–	–	–

Table 2.10

Synthesised Llyn Celyn Gross Natural Monthly Runoff (cusec days x 10^3)

Note - Based on linear regressions (calculated by computer from the overlapping data period Sept. '62 to Sept.69) relating Llyn Celyn (1000 cusec days) with Lake Vyrnwy runoff (inches) and Brenig runoff (1000 cusec days).

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec	Ann.Total
10	4.99	6.19	2.08	2.02	1.00	0.98	2.35	4.11	0.53	1.99	3.81	7.59	37.64
11	2.46	3.50	1.89	1.82	1.19	0.82	0.25	0.90	1.53	3.58	6.01	6.61	30.56
12	4.84	3.05	5.13	0.77	0.29	2.23	3.95	5.51	0.79	4.30	3.69	9.51	44.06
13	6.95	3.45	5.23	3.99	3.54	2.56	0.29	0.54	0.81	3.09	4.54	3.96	38.95
14	6.31	6.08	4.72	2.06	1.71	0.49	1.64	2.63	1.18	0.84	5.39	9.94	42.99
15	7.33	6.39	2.12	1.24	1.32	0.16	1.32	2.56	0.32	1.36	2.72	9.90	36.74
16	7.32	4.32	2.25	2.76	1.57	1.40	1.98	0.86	1.41	8.56	4.83	3.54	40.80
17	2.75	1.18	2.37	2.14	0.86	1.33	0.85	5.57	1.99	5.75	3.84	2.58	31.21
18	5.47	5.85	1.27	1.44	1.38	0.57	2.83	0.68	7.27	4.84	3.06	8.89	43.55
19	5.64	2.63	4.04	3.26	1.18	0.42	0.19	1.17	1.67	1.83	2.81	9.17	34.01
20	8.00	5.33	3.67	5.31	3.82	1.02	7.19	2.14	1.08	2.22	2.77	5.64	48.19
21	8.95	0.72	4.09	1.05	0.79	0.23	0.62	2.98	0.99	2.36	2.89	7.39	33.06
22	6.45	5.90	3.14	1.79	0.99	0.48	4.67	1.90	2.49	1.14	2.46	6.20	37.61
23	4.74	8.13	1.96	1.51	3.08	1.07	1.35	4.57	3.49	7.98	4.74	6.74	49.36
24	5.78	0.68	1.66	2.11	4.44	2.88	4.81	4.11	3.80	5.58	3.32	7.17	46.34
25	6.74	7.14	1.82	1.69	2.47	0.52	0.12	1.09	1.82	5.56	2.50	7.22	38.69
26	7.96	4.81	1.73	0.94	1.86	1.34	2.16	2.69	1.66	4.10	7.23	2.44	38.92
27	5.67	2.94	2.75	2.27	0.92	3.48	2.33	6.37	3.13	6.71	5.54	2.25	44.36
28	11.51	7.09	2.55	1.63	0.39	3.05	2.22	3.27	0.73	6.13	9.59	4.68	52.84
29	2.64	1.37	0.65	0.37	1.70	0.96	0.86	3.52	0.87	5.70	9.15	12.43	40.22
30	8.54	1.18	1.88	2.95	1.60	0.96	2.66	4.19	3.59	6.48	6.04	7.18	47.25
31	5.50	5.33	1.07	3.16	3.18	3.80	1.87	3.92	1.70	1.16	8.12	3.49	42.30
32	9.58	0.34	1.94	4.61	2.56	0.52	2.95	0.99	3.15	6.33	3.43	4.26	40.66
33	4.71	4.46	3.62	0.60	0.55	0.80	2.01	0.27	0.13	2.64	1.51	1.25	22.55
34	6.76	0.52	3.21	1.74	2.19	0.74	0.17	1.83	1.94	5.61	2.31	7.42	34 .44
35	3.08	6.78	1.60	3.08	0.48	2.59	0.42	0.24	4.26	8.09	5.15	4.46	40.23
36	7.01	2.18	2.43	2.03	0.62	2.64	6.60	1.17	2.97	3.17	5.50	7.29	43.61
37	7.32	6.75	2.39	3.26	0.72	1.46	1.00	0.33	0.37	1.15	1.47	4.05	30.27
38	9.32	2.17	1.48	0.52	1.00	3.74	3.70	2.79	0.62	9.08	6.52	6.07	47.01
40	10.13	4.08	3.08	1.83	0.61	0.98	8.00	1.82	0.49	1.44	8.02	5.71	46.19
40	1.32	5.35	2.92	2.45	1.28	0.34	1.56	0.32	1.58	5.79	10.42	3.43	36.76
41	1.54	6.68	2.57	1.67	1.82	0.78	0.39	2.66	0.92	5.19	3.22	4.32	31.76
42	5.34	2.71	2.32	2.89	2.49	0.61	2.12	3.36	2.12	4.92	0.89	6.29	36.06
43	9.04	4.07	0.79	1.09	3.54	4.20	2.83	3.18	4.15	4.09	3.03	3.68	43.69

Year	Jan	Feb	Mar	Ap	May	Jn	Jy	Aug	Sep	Oct	Nov	Dec	Ann.Total
44	7.87	2.18	0.79	1.06	0.88	2.17	2.73	0.92	3.74	7.27	8.01	7.24	44.86
45	5.16	7.51	1.60	2.34	2.15	3.87	2.10	1.60	2.49	5.34	0.81	5.12	40.09
46	7.97	8.62	1.95	0.54	0.42	3.29	1.57	4.38	5.57	1.80	7.13	7.11	50.35
47	6.15	0.55	8.48	4.08	1.98	1.45	2.27	0.34	0.51	0.51	5.81	3.59	35.72
48	14.56	5.06	2.16	1.95	1.00	3.90	1.72	4.35	2.99	2.20	3.56	5.65	49.10
49	5.08	2.37	1.27	4.27	1.85	1.54	0.02	0.88	0.32	6.02	6.25	9.24	39.11
50	2.92	7.96	2.68	2.93	1.15	0.53	5.14	5.13	7.17	3.69	4.16	5.81	49.27
51	6.56	4.79	4.14	3.29	1.52	0.71	0.66	2.85	3.63	1.29	8.46	8.97	46.87
52	6.17	3.14	2.57	1.24	1.54	1.33	1.23	2.06	1.30	7.85	3.02	5.88	37.33
53	2.01	3.20	2.80	2.39	1.52	1.56	6.03	2.54	3.72	1.88	6.73	2.08	36.46
54	3.65	4.14	3.56	1.51	1.85	3.09	3.88	3.56	4.38	11.11	8.11	7.20	56.04
55	4.52	2.28	2.48	1.49	4.78	3.76	1.16	0.24	0.25	1.41	2.46	4.27	29.10
56	6.96	0.95	1.99	0.74	0.80	0.76	3.74	5.30	3.13	3.12	2.04	4.77	34.30
57	7.87	5.15	4.71	0.58	1.01	0.42	6.86	4.91	5.47	4.68	3.10	3.62	48.38
58	5.83	6.45	1.29	0.88	3.83	2.40	2.05	3.18	5.60	3.84	1.71	2.74	39.80
59	7.40	0.77	2.23	3.83	0.79	0.94	3.26	0.56	0.12	3.74	4.44	9.13	37.21
60	8.69	5.39	2.40	2.61	0.57	0.78	4.18	3.13	3.52	5.09	9.36	7.25	52.97
61	6.36	4.53	0.91	2.66	2.20	0.46	0.62	2.72	2.11	6.55	4.16	6.50	39.78
62	8.40	3.59	1.43	4.56	2.09	0.68	0.93	5.01	3.36	1.74	3.38	4.94	40.11
63	0.72	0.42	7.08	3.51	2.85	2.01	1.94	1.94	2.30	2.39	8.44	1.68	35.28
64	1.80	1.55	1.81	1.62	2.43	1.11	2.72	0.99	1.01	2.99	3.25	11.11	32.39
65	7.92	0.90	2.98	2.52	2.73	3.37	1.92	1.89	3.58	2.57	3.52	13.10	47.00
66	3.73	6.35	2.56	4.13	2.74	1.84	1.65	2.42	1.93	3.66	4.31	10.19	45.51
67	3.66	5.02	2.29	1.51	4.73	0.88	2.01	3.07	4.94	11.65	2.79	7.48	50.03
68	8.90	1.90	6.15	1.89	2.72	1.81	2.58	1.15	4.95	5.70	3.03	3.75	44.53

Table 2.II

Erbistock monthly natural runoff from regression equations with low flow modifications*

Year	J	F	M	A	My	J	Jy	A	S	O	N	D
23	45.30	99.18	33.06	22.23	16.67	10.58	7.46*	26.63	34.15	82.36	66.36	58.31
24	55.17	10.06	17.62	26.93	43.60	26.62	25.27	44.26	50.42	50.38	37.08	69.21
25	50.84	87.58	26.31	19.25	23.76	15.75*	3.32*	4.15*	25.06	52.56	35.84	62.40
26	70.30	49.54	27.75	13.06	21.30	9.95*	23.27	21.05	26.57	40.95	80.12	17.66
27	56.12	32.24	59.28	23.65	12.07	28.56	21.01	65.42	45.75	68.01	70.32	21.75
28	100.34	84.82	25.23	19.92	10.78*	26.02	17.57	18.12	14.93*	70.02	109.69	42.04
29	26.04	16.63	9.13*	5.81*	12.38	5.81*	4.15*	26.14	14.10*	51.50	89.79	115.46
30	88.66	17.75	34.54	30.07	13.22	5.81*	21.01	45.58	38.20	68.02	77.35	63.43
31	55.43	62.69	18.33	32.83	31.61	33.56	9.95*	46.26	36.92	15.04	86.25	31.65
32	81.25	11.62*	23.56	46.41	28.44	9.13*	21.90	16.54	35.46	73.07	37.86	38.23
33	51.03	53.43	46.55	9.13*	8.30*	10.19	11.40	4.98	4. 15	20.03	18.97	10.62
34	51.90	12.45*	35.04	21.26	25.80	8.30*	4.15*	15.19	14.22	52.23	40.73	78.52
35	42.69	76.66	25.56	37.98	13.28*	18.99	16.60*	5.81*	38.59	88.34	69.80	41.28
36	70.99	31.15	54.27	20.22	11.71	33.60	38.05	13.27*	30.56	29.45	70.18	70.10
37	57.67	75.65	64.20	27.75	14.93*	10.47	6.63*	3.32*	3.32*	12.45*	13.86	44.55
38	74.20	25.10	17.81	9.95*	6.64*	31.91	27.70	18.04	12.43*	69.82	63.72	54.81
39	110.58	46.32	52.40	19.17	10.37	11.61*	52.74	17.88	6.64*	17.33	81.21	54.91
40	13.29	59.24	40.28	30.44	17 .71	8.30*	10.78*	4.98*	14.64	46.34	101.76	40.94
41	21.34	86.59	51.59	17.23	15.19	13.27*	4.98*	17.58	9.13*	45.65	35.59	45.30
42	50.05	39.41	31.91	36.04	17.29	8.30*	11.39	29.27	22.92	38.06	13.58	55.86
43	70.27	51.10	12.43*	13.95	36.86	27.64	15.68	19.83	49.42	42.26	38.38	33.97
44	69.17	26.00	17.61	12.46	8.30*	15.62	13.62	6.63*	42.70	60.21	93.25	67.30
45	47.78	83.59	18.44	27.98	22.29	28.22	13.08	7.47*	18.23	44.18	12.21	53.67
46	66.13	99.48	22.90	9.95*	9.13*	28.20	9.13*	39.48	56.81	14.91	67.30	65.30
47	60.75	10.78*	113.04	45.44	24.15	11.30	10.82	4.15*	4.15*	3.32*	57.23	41.38
48	132.53	57.44	17.96	23.05	7.47*	30.36	10.52	28.52	29.07	20.54	37.69	54.49
49	56.61	24.93	25.35	48.20	15.64	11.54	4.98*	10.78*	3.32*	43.24	62.97	86.06
50	26.31	87.88	30.80	31.41	13.45	5.81*	18.69	31.73	78.64	41.52	50.82	45.17
51	59.04	55.17	64.61	38.35	21.75	10.78*	5.81*	21.65	38.24	13.07	91.92	88.42
52	62.87	40.41	22.74	15.59	17.86	6.63*	3.32*	5.81*	18.73	63.52	37.97	56.03
53	19.69	38.33	23.53	28.65	13.16	14.29	24.56	14.09	38.17	14.61	58.89	21.02
54	32.22	47.22	42.50	18.20	23.25	25.18	24.09	40.54	36.51	91.80	88.35	71.39
55	43.51	25.73	35.78	19.99	48.06	27.41	8.30*	3.32*	3.32*	10.18	22.19	18.11
56	56.80	11.70	26.48	15.75*	9.12*	4.98*	22.30	58.46	33.94	30.24	19.02	59.75
57	58.17	57.86	40.19	9.95*	9.12*	3.32*	31.61	53.62	60.37	38.05	38.99	40.11
58	49.98	77.99	19.08	11.79	28.95	20.11	19.11	29.97	56.45	44.45	24.43	33.41

Year	J	F	M	A	My	J	Jy	A	S	O	N	D
59	78.69	9.53	21.70	39.47	13.81	8.30*	13.55	4.15*	2.49*	24.90*	49.12	105.08
60	88.79	62.69	29.70	32.68	8.30*	4.15*	15.37	16.92	43.09	48.57	99.35	74.48
61	62.00	53.16	11.46	35.96	25.83	4.98*	14.10*	20.85	16.82	51.31	39.80	62.36
62	88.00	40.29	60.05	60.21	20.38	9.13*	5.81	31.97	33.66	15.68	42.48	48.62
63	10.78*	6.64*	82.12	42.30	25.50	22.59	16.72	12.21	22.06	21.33	84.74	18.27
64	13.21	17.63	31.18	22.61	20.29	8.30*	12.23	10.78	9.95*	26.91	27.90	113.07
65	84.46	1[illegible].66	39.08	27.08	25.59	24.21	15.97	12.36	46.71	26.55	38.23	125.03
66	32.83	66.22	30.56	47.38	22.00	12.97	11.61*	26.84	19.35	44.99	50.71	94.15
67	32.24	56.22	29.50	17.46	44.23	11.61*	6.63*	16.71	42.00	104.78	46.26	66.07
68	92.14	22.61	54.80	21.04	33.77	14.55	29.03	9.95	50.52	53.71	39.44	38.57

Table 2.I2

Erbistock synthesised record I923/68 made up of selected historical months e.g.for March I923 in the synthesised record use March I946

YEAR	J	F	M	A	M	J	J	A	S	O	N	D
23	I.55	2.46	3.46	4.54	5.54	6.4I	8.45	8.39	9.54	II.44	I.49	I2.39
24	I.5I	2.47	3.45	4.45	5.55	6.43	7.38	8.54	9.43	I0.44	II.43	I2.46
25	2.5I	2.45	3.46	5.40	6.45	5.54	7.49	8.55	9.4I	I0.44	II.43	I.38
26	I.52	3.4I	3.46	4.52	6.43	6.52	7.53	8.38	9.48	I0.42	II.44	I.4I
27	I.46	2.55	3.5I	4.54	5.4I	6.48	8.38	9.46	I0.50	9.46	I2.46	I2.53
28	2.46	2.50	3.46	4.55	5.39	6.48	7.43	8.43	9.52	II.39	2.46	I.55
29	I.40	3.44	3.43	5.38	5.4I	6.40	7.4I	8.39	9.52	I0.52	II.44	I.48
30	2.45	3.45	3.55	4.50	5.49	6.50	8.42	9.43	I0.43	II.46	I2.46	I.48
3I	I.46	2.43	3.48	4.42	5.47	6.55	7.42	9.43	I0.43	I0.5I	II.5I	I.45
32	2.45	2.47	3.42	4.49	5.47	7.42	8.42	9.45	9.5I	II.46	II.43	I2.47
33	2.54	2.5I	3.54	4.46	5.46	6.42	7.42	8.55	9.55	II.55	II.45	II.42
34	I2.42	2.47	3.55	4.55	4.45	6.5I	7.5I	7.50	9.52	I0.52	II.43	I.47
35	I.55	I2.44	3.46	3.55	5.49	5.50	7.47	8.47	I0.49	II.44	I2.44	I.55
36	I.43	2.55	3.4I	4.4I	5.4I	6.54	8.54	7.75	9.5I	9.48	II.46	I2.46
37	I.5I	I.43	2.43	4.54	5.4I	6.4I	639	7.49	9.49	9.4I	II.42	I2.42
38	I.43	2.44	3.45	4.46	5.46	6.54	7.54	7.43	9.40	II.39	II.49	I2.50
39	2.46	2.54	3.4I	4.55	5.44	6.42	8.50	7.43	9.4I	I0.45	II.44	I2.52
40	2.47	2.48	3.54	4.50	5.5I	6.5I	7.45	8.55	8.5I	I0.50	II.54	I2.45
4I	I.53	2.45	3.39	3.44	5.52	6.42	7.52	9.45	9.38	I0.45	II.43	I2.55
42	I2.42	2.39	3.50	3.40	5.53	6.5I	7.55	8.46	9.42	9.53	I0.39	I2.48
43	I.52	I2.42	2.47	4.44	5.55	6.48	7.47	8.53	I0.42	I0.43	II.48	I2.4I
44	I.52	2.38	3.45	4.43	5.48	6.44	7.48	8.47	9.53	I0.38	II.40	I2.46
45	2.54	2.50	3.53	4.48	4.54	6.48	7.46	8.49	9.52	I0.40	II.42	I2.38
46	I.44	I.39	3.42	4.38	5.44	6.48	7.42	8.54	I0.44	I0.5I	II.53	I.38
47	I.38	2.47	2.46	4.49	4.40	7.45	6.4I	8.55	9.55	9.49	I0.45	I2.55
48	2.46	2.40	3.45	4.54	5.46	6.54	7.42	8.46	9.44	I0.43	II.43	I2.38
49	I.46	2.44	3.42	3.39	5.52	6.42	7.4I	8.4I	7.49	I0.49	II.53	I2.54
50	I.40	2.45	2.55	4.42	5.49	6.40	7.43	8.46	9.46	I0.43	I2.50	I2.55
5I	I.46	2.40	2.43	3.40	4.4I	6.4I	6.50	8.38	9.53	I0.39	II.40	I2.49
52	I.38	2.39	3.42	4.44	5.5I	6.39	7.49	8.47	9.42	I0.38	II.43	I2.48
53	I.4I	2.52	3.42	4.48	5.49	6.49	7.38	7.48	9.54	I0.39	II.38	I.4I
54	I.50	I2.55	4.5I	3.45	4.54	6.38	7.38	8.54	9.53	II.5I	II.5I	I2.44
55	I2.55	2.44	3.40	4.39	5.49	6.48	6.5I	7.49	9.49	9.40	I0.46	I2.42
56	I.46	2.47	3.49	4.44	5.44	7.52	7.43	9.46	9.44	II.4I	I0.48	I2.48
57	I.5I	2.40	3.40	4.46	5.39	7.49	6.54	I0.44	9.46	I0.45	II.48	I2.45
58	I2.42	I.47	3.53	4.44	5.47	5.50	7.43	8.46	I0.44	I0.43	I0.46	I2.43

YEAR	J	F	M	A	M	J	J	A	S	O	N	D
59	I.47	2.47	3.52	4.5I	5.49	6.5I	7.48	8.55	9.49	I0.39	I2.42	2.46
60	I2.44	2.43	3.50	4.52	5.44	7.4I	7.47	9.45	9.43	I0.50	II.40	I2.44
6I	I.38	2.5I	3.43	3.40	4.40	6.50	7.48	8.38	9.45	I0.50	II.43	I2.48
62	2.45	2.39	3.44	4.47	5.50	6.52	6.50	8.46	9.44	9.45	II.50	I2.55
63	2.47	2.47	2.50	4.5I	4.45	6.43	7.43	8.52	9.42	I0.46	II.44	I. 4I
64	2.47	3.49	3.50	4.54	5.50	6.5I	7.40	8.4I	7.42	I0.45	II.4I	I.48
65	2.45	2.47	3.55	4.40	4.40	6.43	7.47	8.52	9.43	I0.48	II.43	I.48
66	I.50	I.44	3.50	4.49	4.54	6.47	7.55	8.48	9.42	I0.42	I2.50	I.39
67	I.50	2.40	3.50	3.45	4.49	6.42	6.39	9.45	9.53	I0.54	II.52	I.44
68	I.39	2.49	3.4I	3.39	5.47	6.49	7.54	8.4I	9.43	I0.44	II.48	I2.45

Table 2.I3

Erbistock runoffs for synthesised record in table 2.I2 (cusec days x $I0^3$)

YEAR	J	F	M	A	M	J	J	A	S	O	N	D
23	43.2I	I06.58	27.25	22.66	I7.20	I0.85	7.09	26.26	35.I9	83.82	63.69	59.58
									annual total 503.38			
24	58.59	I0.I0	I7.83	25.75	4I.08	23.65	26.72	4I.26	47.40	56.28	36.89	70.67
												456.22
25	55.6I	86.38	27.25	I8.22	26.44	I7.20	3.60	4.50	6.93	56.28	36.89	63.54
												402.84
26	69.79	53.3I	27.25	I2.75	23.65	9.04	25.08	20.29	28.60	40.60	83.82	I9.44
												4I3.62
27	56.68	29.85	63.4I	22.66	I2.67	26.90	20.29	66.30	46.97	67.30	70.67	2I.58
												506.28
28	I06.58	82.35	27.25	20.I5	9.55	26.90	I8.25	I9.58	I4.62	70.03	I06.58	43.2I
												545.05
29	27.42	I7.04	9.23	4.98	I2.67	6.22	4.I5	26.26	I4.62	56.63	83.82	I22.5I
												385.55
30	86.38	I7.83	35.96	30.I6	I3.05	5.37	I9.57	47.40	40.96	7I.8I	70.60	63.69
												503.2I
3I	56.68	60.57	I7.83	3I.89	3I.69	32.28	9.95	47.40	40.96	I3.00	92.08	34.98
												469.3I
32	86.38	I0.I0	23.88	45.86	3I.69	9.95	I9.57	I6.65	33.70	7I.8I	36.89	36.38
												422.86
33	47.74	55.6I	43.94	9.8I	6.74	II.I0	9.95	4.50	4.4I	22.73	I7.28	I3.92
												247.63
34	52.89	I0.I0	35.96	20.I5	25.75	8.I4	4.35	4.35	I4.62	56.63	36.89	76.54
												346.37
35	43.2I	73.55	27.25	35.96	I3.05	I9.55	I5.78	6.33	43.69	83.82	73.55	43.2I
												478.95
36	73.62	29.85	53.3I	2I.I0	I2.67	3I.86	4I.26	II.59	33.70	28.60	7I.8I	70.67
												480.04
37	58.89	73.62	60.57	22.66	I2.67	I0.85	6.99	3.6	3.59	6.93	I3.92	52.89
												327.I8
38	73.62	26.74	I7.83	9.8I	6.74	3I.86	27.3I	I8.25	II.76	70.03	62.I0	53.77
												409.82
39	I06.58	47.74	53.3I	20.I5	8.7I	II.I9	37.02	I8.25	I9.80	39.87	83.82	55.53
												50I.88
40	I0.I0	59.80	43.90	30.I6	I7.57	8.I4	II.84	4.50	II.64	46.97	98.85	38.92
												382.39
4I	2I.57	86.38	49.I9	I7.04	I6.76	II.I0	5.0I	I6.65	8.4I	39.87	36.89	45.57
												354.44
42	52.89	39.49	32.50	36.85	I7.33	8.I4	II.59	29.26	I9.80	38.85	I4.07	56.07
												356.84
43	69.79	52.89	I0.I0	I3.29	4I.08	26.90	I5.78	I9.96	56.63	40.96	40.35	35.76
												423.49
44	69.79	27.96	I7.83	I3.24	8.0I	I6.6I	I3.86	6.33	38.85	59.93	95.36	70.67
												438.44
45	47.74	82.35	I8.43	28.77	I7.20	26.90	I3.09	8.29	I4.62	29.55	I3.92	54.28
												355.I4

YEAR	J	F	M	A	M	J	J	A	S	O	N	D
46	66.62	93.56	23.88	8.62	8.7I	26.90	9.95	4I.26	56.28	I3.00	64.23	63.54
											476.55	
47	63.54	I0.I0	I06.58	45.86	26.46	II.84	I0.85	4.50	4.4I	3.59	39.87	45.57
											373.I7	
48	I06.58	56.27	I7.83	22.66	6.74	3I.86	9.I5	29.26	33.32	20.32	36.89	54.28
											425.96	
49	56.68	26.74	23.88	49.I9	I6.76	II.I0	4.I5	I0.60	3.60	43.69	64.23	86.09
											396.7I	
50	27.42	86.38	29.85	3I.89	I3.05	6.22	I8.25	29.26	67.30	40.96	53.77	45.57
											449.92	
5I	56.68	56.27	60.57	36.85	2I.I0	I0.85	5.37	20.29	38.85	I4.07	95.36	85.23
											50I.49	
52	63.54	39.49	23.88	I3.29	I7.57	6.9I	3.60	6.33	I9.80	59.93	36.89	56.07
											347.38	
53	I9.44	38.95	23.88	28.77	I3.05	I3.50	26.72	I3.86	35.I9	I4.07	60.9I	I9.44
											307.78	
54	3I.49	45.57	43.I2	I7.83	22.66	25.83	26.72	4I.26	38.85	92.08	92.08	73.55
											55I.04	
55	45.57	26.74	36.85	20.80	I3.05	26.90	8.I4	3.60	3.59	II.76	22.4I	52.89
											272.30	
56	56.68	I0.I0	27.II	I3.29	8.7I	9.04	I8.25	67.30	33.32	27.82	20.92	56.07
											348.6I	
57	58.59	56.27	36.85	9.8I	9.55	3.60	3I.86	56.28	67.30	39.87	40.35	38.92
											449.25	
58	52.89	76.54	I8.43	I3.29	3I.69	I9.55	I8.25	29.26	56.28	40.96	22.4I	34.I3
											4I3.68	
59	76.54	I0.I0	2I.22	43.I2	I3.05	8.I4	I3.86	4.50	3.59	I4.07	5[illegible].89	70.67
											33I.75	
60	86.09	60.57	32.50	3I.89	8.7I	4.I5	I5.78	I6.65	47.40	46.97	95.36	73.55
											5I9.62	
6I	63.54	55.6I	9.23	36.85	26.46	5.37	I3.86	20.29	I6.65	46.97	36.89	56.07
											387.79	
62	86.38	39.49	I7.04	56.69	I9.55	9.04	5.37	29.26	33.32	I6.65	42.47	45.57
											400.83	
63	I0.I0	I0.I0	82.35	43.I2	25.75	23.65	I8.25	II.80	I9.80	22.4I	83.82	I9.44
										n	370.59	
64	I0.I0	27.II	32.50	22.66	I9.55	8.I4	I2.I8	I0.60	9.95	39.87	27.82	I22.5I
											342.99	
65	86.38	I0.I0	35.96	26.46	26.26	23.65	I5.78	II.80	47.40	20.92	36.89	I22.5I
											464.3I	
66	3I.49	66.62	32.50	45.86	22.66	I2.78	II.59	26.36	I9.80	40.60	53.77	93.56
											457.59	
67	3I.49	56.27	32.50	I7.83	45.86	II.I0	6.99	I6.65	38.85	99.67	44.I6	66.62
											467.99	
68	93.56	23.42	53.3I	20;80	3I.69	I3.50	27.3I	I0.60	47.40	56.28	40.35	38.92
											457.I4	

Table 2.I4

Erbistock effective releases (cusec days x 10^3) for synthetic record

Maintained flow 400 cusecs

YEAR	J	F	M	A	M	J	J	A	S	O	N	D	ANNUAL
23	-	-	-	I.70	2.83	3.48	5.95	0.95	-	-	-	-	I4.9I
24	-	I.57	0.22	I.II	-	0.09	0.42	-	-	-	-	-	3.4I
25	-	-	-	0.47	0.38	2.83	8.59	8.47	5.37	-	-	-	26.II
26	-	-	-	I.28	0.09	4.I5	0.8I	0.37	-	-	-	0.37	7.07
27	-	-	-	I.70	5.I3	0.38	0.37	-	-	-	-	-	7.58
28	-	-	-	0.66	3.28	0.38	I.77	0.85	2.54	-	-	-	9.48
29	I.5I	0.32	3.34	6.70	5.I3	5.23	8.49	0.95	2.54	-	-	-	34.2I
30	-	0.22	-	-	3.86	6.47	0.35	-	-	0.2I	-	-	II.II
3I	-	-	0.5I	0.54	-	-	4.83	-	-	I.90	-	-	7.78
32	-	I.57	-	0.32	-	4.83	0.35	I.90	-	0.2I	-	-	9.I8
33	-	-	-	2.52	5.98	3.07	4.83	8.47	8.28	I.04	0.97	0.52	35.68
34	-	I.57	-	0.66	I.II	4.I4	7.99	2.73	2.54	-	-	-	20.74
35	-	-	-	-	3.86	I.70	I.I0	6.90	4.46	-	-	-	I8.02
36	-	-	-	0.39	5.I3	-	-	3.9I	-	-	0.2I	-	9.64
37	-	-	-	I.70	5.I3	3.48	5.04	8.59	8.45	5.37	0.52	-	38.28
38	-	-	0.22	2.52	5.98	-	0.I0	I.77	4.37	-	-	-	I4.96
39	-	-	-	0.66	3.97	3.07	0.74	I.77	5.37	2.I2	-	-	I7.70
40	I.57	-	-	-	0.05	4.I4	I.75	8.47	4.73	-	-	0.25	20.96
4I	0.I0	-	-	0.32	2.07	3.07	7.23	I.90	3.63	2.I2	-	0.83	2I.27
42	-	-	-	-	0.39	4.I4	3.9I	-	0.4I	0.I8	2.80	-	II.83
43	-	-	I.57	0.98	-	0.38	I.I0	I.09	-	-	-	0.0I	5.I3
44	-	0.06	0.22	2.I4	4.27	I.96	I.I5	6.90	0.I8	0.I4	-	-	I7.02
45	-	-	2.46	0.36	I.70	0.38	I.69	4.89	2.54	0.97	0.52	-	I5.5I
46	-	-	-	4.50	3.97	0.38	4.83	-	-	I.90	-	-	I5.58
47	-	I.57	-	0.32	-	I.75	3.48	8.47	8.28	8.35	2.I2	0.83	35.I7
48	-	-	0.22	I.70	5.98	-	4.83	-	-	0.64	-	-	I3.37
49	-	-	-	-	2.07	3.07	8.49	3.95	8.59	4.46	-	-	30.63
50	I.5I	-	-	0.54	3.86	5.23	I.77	-	-	-	-	-	I2.9I
5I	-	-	-	-	0.39	3.48	6.47	0.37	0.I8	2.80	-	-	I3.69
52	-	-	-	0.98	0.05	5.04	8.59	6.90	0.4I	0.I4	-	-	22.II
53	0.37	-	-	0.36	3.86	3.25	0.42	I.I5	-	2.80	-	0.37	I2.58
54	-	0.83	-	0.22	I.70	I.I6	0.42	-	0.I8	-	-	-	4.5I
55	0.83	-	-	-	3.86	0.38	4.I4	8.59	8.35	4.37	0.33	-	30.85
56	-	I.57	0.I9	0.98	3.97	7.23	I.77	-	-	0.22	0.45	-	I6.38
57	-	-	-	2.52	3.28	8.59	-	-	-	2.I2	0	0.25	I6.76

YEAR	J	F	M	A	M	J	J	A	S	O	N	D	ANNUAL
58	-	-	2.46	0.98	-	I.70	I.77	-	-	-	0.33	-	7.24
59	-	I.57	0.20	-	3.86	4.I4	I.I5	8.47	8.35	2.80	-	-	30.54
60	-	-	-	I.28	3.97	8.49	I.I0	I.90	-	-	-	-	I6.00
6I	-	-	3.34	-	-	6.47	I.I5	0.37	I.90	-	-	-	I3.23
62	-	-	0.32	-	I.70	4.I5	6.47	-	-	I.90	-	0.83	I5.37
63	I.57	I.57	-	-	I.I2	0.09	I.77	3.59	0.4I	0.33	-	0.37	I0.82
64	I.57	0.I9	-	I.70	I.70	4.I4	3.33	3.95	4.83	2.I2	0.22	-	23.75
65	-	I.57	-	-	-	0.09	I.I0	3.59	-	0.45	-	-	6.80
66	-	-	-	0.32	I.70	I.20	3.9I	I.33	0.4I	-	-	-	8.87
67	-	-	-	0.22	0.32	3.07	5.04	I.90	0.I8	-	-	-	I0.73
68	-	-	-	-	-	3.25	0.I0	3.95	-	-	-	0.25	7.55

Table 2.I5

Erbistock monthly natural runoff : Final combined synthetic and recorded data (cusec days x 10^3)

Year	J	F	M	A	M	J	J	A	S	O	N	D
23	43.2I	I06.58	27.25	22.66	I7.20	I0.85	7.09	26.26	35.I9	83.82	63.69 503.38	59.58
24	58.59	I0.I0	I7.83	25.75	4I.08	23.65	26.72	4I.26	47.40	56.28	36.89 456.22	70.67
25	55.6I	86.38	27.25	I8.22	26.44	I7.20	3.60	4.50	6.93	56.28	36.89 402.84	63.54
26	69,79	53.3I	27.25	I2.75	23.65	9.04	25.08	20.29	28.60	40.60	83.82 4I3.62	I9.44
27	56.68	29.85	63.4I	22.66	I2.67	26.90	20.29	67.30	46.97	67.30	70.67 506.28	2I.58
28	I06.58	82.35	27.25	20.I5	9.55	26.90	I8.25	I9.58	I4.62	70.03	I06.58 545.05	43.2I
29	27.42	I7.04	9.23	4.98	I2.67	6.22	4.I5	26.26	I4.62	56.63	83.82 385.55	I22.5I
30	86.38	I7.83	35.96	30.I6	I3.05	5.37	I9.57	47.40	40.96	7I.8I	70.60 503.2I	63.69
3I	56.68	60.57	I7.83	3I.89	3I.69	32.28	9.95	47.40	40.96	I3.00	92.08 469.3I	34.98
32	86.38	I0.I0	23.80	45.86	3I.69	9.95	I9.57	I6.65	33.70	7I.8I	36.89 422.86	36.30
33	47.74	55.6I	43.90	9.8I	6.74	II.I0	9.95	4.50	4.4I	22.73	I7.22 247.63	I3.92
34	52.89	I0.I0	35.96	20.I5	25.75	8.I4	4.35	4.35	I4.62	56.63	36.89 346.37	76.54
35	43.2I	73.55	27.25	35.96	I3.05	I9.55	I5.78	6.33	43.69	83.82	73.55 478.95	43.2I
36	73.62	29.85	53.3I	2I.I0	I2.60	3I.86	4I.26	II.59	33.70	28.60	7I.8I 480.04	70.67
37	58.89	73.62	60.57	22.66	I2.67	I0.85	6.99	3.60	3.59	6.93	I3.92 327.I8	52.89
38	63.54	27.96	20.02	8.62	4.98	25.83	26.72	20.29	8.4I	59.93	60.9I 38I.40	54.28
39	93.56	39.49	49.I9	20.80	9.55	6.99	38.35	26.26	7.0I	I4.07	70.03 438.88	59.58
40	27.42	56.27	36.85	26.46	I8.22	6.22	I2.I8	4.8I	II.76	29.55	95.36 362.2I	37.II
4I	I9.44	68.94	53.3I	2I.I0	I2.67	I0.85	4.I8	I0.60	6.93	32.55	27.82 304.I2	35.76
42	39.94	37.50	23.88	3I.89	I7.29	II.I0	9.95	I9.57	I9.80	40.60	I3.92 3I8.33	52.89
43	73.62	60.57	9.23	I3.24	39.82	23.65	I8.25	I9.50	47.40	40.96	36.89 4I7.34	34.I3
44	66.62	26.74	I7.04	I3.29	8.7I	I6.6I	II.86	8.59	33.32	56.28	83.82 4I6.43	73.55
45	34.98	86.38	I7.83	25.75	I8.37	26.44	II.84	7.09	I6.65	39.87	I7.22 34I.34	38.92

Year	J	F	M	A	M	J	J	A	S	O	N	D
46	56.68	106.58	27.25	9.81	6.74	25.87	13.09	29.26	67.30	22.41	71.81	70.67
									Annual =		507.47	
47	76.54	10.10	115.00	56.69	31.69	12.78	15.78	6.33	4.72	4.81	46.13	36.38
											416.95	
48	122.51	59.80	17;83	28.77	8.01	26.90	13.86	26.36	28.60	20.92	40.35	56.07
											449.98	
49	63.69	23.42	27.11	45.86	13.05	13.50	3.60	8.29	3.59	43.69	62.10	45.23
											393.13	
50	31.49	82.35	32.50	30.16	19.55	5.37	15.81	37.02	75.87	46.97	42.47	53.77
											473.33	
51	58.59	55.61	63.41	43.12	17.57	8.14	4.35	11.64	33.70	13.00	92.08	83.51
											484.72	
52	69.79	38.95	21.22	1275	16.76	9.04	5.01	11.80	14.62	56.63	44.16	55.53
											356.26	
53	21.57	39.65	18.43	28.10	17.33	13.31	25.08	19.96	38.85	20.32	64.23	21.58
											328.41	
54	35.31	47.74	43.90	22.66	17.20	31.86	27.31	41.26	35.19	99.65	98.85	86.09
											587.04	
55	43.21	29.85	35.96	20.15	41.08	32.28	11.59	4.50	4.41	12.34	22.73	45.57
											303.67	
56	56.68	10.10	27.11	13.29	8.71	9.04	18.25	67.30	33.32	27.82	20.92	56.07
											348.61	
57	58.59	56.27	36.85	9.81	9.55	3.60	31.86	56.28	67.30	39.87	40.35	38.92
											449.25	
58	52.89	76.54	18.43	13.29	31.69	19.55	18.25	29.26	56.28	40.96	22.41	34.13
											413.68	
59	76.54	10.10	21.22	43.12	13.05	8.14	13.86	4.50	3.59	14.07	52.89	70.67
											331.75	
60	86.09	60.57	32.50	31.89	8.71	4.15	15.78	16.65	47.40	46.97	95.36	73.55
											519.62	
61	63.54	55.61	9.23	36.85	26.46	5.37	13.86	20.29	16.65	46.97	36.89	56.07
											387.79	
62	86.38	39.49	17.04	56.69	19.55	9.04	5.37	29.26	33.32	16.65	42.47	45.57
											400.83	
63	10.10	10.10	82.35	43.12	25.75	23.65	18.25	11.80	19.80	22.41	83.82	19.44
											370.59	
64	10.10	27.11	32.50	22.66	19.55	8.14	12.18	10.60	9.95	39.87	27.82	122.51
											342.99	

Table 2.16

Erbistock effective releases (cusec days x 10^3) for combined record

Maintained flow 400 cusecs

YEAR	J	F	M	A	M	J	J	A	S	O	N	D	ANNUAL
23	-	-	-	1.70	2.83	3.48	5.95	0.95	-	-	-	-	14.91
24	-	1.57	0.22	1.11	-	0.09	0.42	-	-	-	-	-	3.41
25	-	-	-	0.47	0.38	2.83	8.59	8.47	5.37	-	-	-	26.11
26	-	-	-	1.28	0.09	4.15	0.81	0.37	-	-	-	0.37	7.07
27	-	-	-	1.70	5.13	0.38	0.37	-	-	-	-	-	7.58
28	-	-	-	0.66	3.28	0.38	1.77	0.85	2.54	-	-	-	9.48
29	1.51	0.32	3.34	6.70	5.13	5.23	8.49	0.95	2.54	-	-	-	34.21
30	-	0.22	-	-	3.86	6.47	0.35	-	-	0.21	-	-	11.11
31	-	-	0.51	0.54	-	-	4.83	-	-	1.90	-	-	7.78
32	-	1.57	-	0.32	-	4.83	0.35	1.90	-	0.21	-	-	9.18
33	-	-	-	2.52	5.98	3.07	4.83	8.47	8.28	1.04	0.97	0.52	35.68
34	-	1.57	-	0.66	1.11	4.14	7.99	2.73	2.54	-	-	-	20.74
35	-	-	-	-	3.86	1.70	1.10	6.90	4.46	-	-	-	18.02
36	-	-	-	0.39	5.13	-	-	3.91	-	-	0.21	-	9.64
37	-	-	-	1.70	5.13	3.48	5.04	8.59	8.45	5.37	0.52	-	38.28
38	-	0.06	0.17	4.50	6.70	1.16	0.42	0.37	3.63	0.14	-	-	17.13
39	-	-	-	-	3.28	5.04	0.20	0.95	4.86	2.80	-	-	17.12
40	1.51	-	-	-	0.47	5.23	3.33	7.11	4.37	0.97	-	-	22.88
41	0.37	-	-	0.39	5.13	3.84	8.49	3.95	5.37	1.65	0.22	0.01	29.07
42	-	0.38	-	0.54	3.87	3.07	4.83	0.35	0.41	-	0.52	-	13.97
43	-	-	3.34	2.14	0.48	0.09	1.77	0.85	-	-	-	-	8.66
44	-	-	0.32	0.98	3.97	1.96	2.32	4.45	-	-	-	-	14.01
45	-	-	0.22	1.11	0.27	0.38	1.75	5.95	1.90	2.12	0.97	0.25	14.92
46	-	-	-	2.52	5.98	-	1.69	-	-	0.33	0.21	-	10.73
47	-	1.57	0.28	-	-	1.20	1.10	6.90	7.96	7.44	2.00	-	28.44
48	-	-	0.51	0.36	4.27	0.38	1.15	1.33	-	0.45	-	-	8.46
49	-	-	0.19	0.32	3.86	3.25	8.59	4.89	8.35	4.46	-	-	33.90
50	-	-	-	-	1.70	6.47	2.73	0.74	-	-	-	-	11.64
51	-	-	-	-	0.05	4.14	7.99	4.73	-	1.90	-	-	18.81
52	-	-	0.20	1.28	2.07	4.15	7.23	3.59	2.54	-	-	-	20.96
53	0.10	-	2.46	0.21	0.39	0.85	0.81	1.09	0.18	0.64	-	-	6.72
54	0.02	-	-	1.70	2.83	-	0.10	-	-	-	-	-	4.65
55	-	-	-	0.66	-	-	3.91	8.47	8.28	3.21	1.04	0.83	26.40
56	-	1.57	0.19	0.98	3.97	7.23	1.77	-	-	0.22	0.45	-	16.38

YEAR	J	F	M	A	M	J	J	A	S	O	N	D	ANNUAL
57	-	-	-	2.52	3.28	8.59	-	-	-	2.I2	-	0.25	I6.76
58	-	-	2.46	0.98	-	I.70	I.77	-	-	-	0.33	-	7.24
59	-	I.57	0.20	-	3.86	4.I4	I.I5	8.47	8.35	2.80	-	-	30.54
60	-	-	-	I.28	3.97	8.49	I.I0	I.90	-	-	-	-	I6.74
6I	-	-	3.34	-	-	6.47	I.I5	0.37	I.90	-	-	-	I3.23
62	-	-	0.32	-	I.70	4.I5	6.47	-	-	I.90	-	0.83	I5.37
63	I.57	I.57	-	-	I.I2	0.09	I.77	3.59	0.4I	0.33	-	0.37	I0.82
64	I.57	0.I9	-	I.70	I.70	4.I4	3.33	3.95	4.83	2.I2	0.22	-	23.75
65	-	I.57	-	-	-	0.09	I.I0	3.59	-	0.45	-	-	6.80
66	-	-	-	0.32	I.70	I.20	3.9I	I.33	0.4I	-	-	-	8.87
67	-	-	-	0.22	0.32	3.07	5.04	I.90	0.I8	-	-	-	I0.73
68	-	-	-	-	-	3.25	0.I0	3.95	-	-	-	0.25	7.55

Table 2.17

Erbistock effective releases (cusec days x 10^3) for combined record

Maintained flow 450 cusecs

YEAR	J	F	M	A	M	J	J	A	S	O	N	D	ANNUAL
23	-	-	-	2.42	3.92	4.54	7.40	I.32	0.I2	-	-	-	I9.72
24	-	2.65	0.72	I.74	0.00	0.25	0.89	-	-	-	-	-	6.25
25	-	-	-	I.06	0.84	3.92	I0.I4	I0.02	6.62	-	-	0.2I	32.8I
26	-	-	-	2.09	0.25	5.30	I.62	0.67	0.06	0.03	-	0.93	I0.95
27	-	-	-	2.42	6.38	0.53	0.67	-	-	-	-	0.II	I0.II
28	-	-	-	I.I6	4.65	0.53	2.67	I.50	3.60	-	-	-	I4.II
29	2.I3	0.57	4.72	8.25	6.38	6.73	I0.04	I.32	3.60	-	-	-	43.74
30	-	0.72	-	-	5.20	7.85	0.73	-	-	0.54	-	-	I5.04
3I	-	-	I.I7	0.98	-	-	5.95	-	-	2.85	-	-	I0.95
32	-	2.65	0.I4	0.78	-	5.95	0.73	2.55	-	0.54	-	0.34	I3.68
33	-	-	-	3.73	7.46	4.I7	5.95	I0.02	9.78	I.6I	I.79	I.36	45.87
34	-	2.65	-	I.I6	I.74	5.40	9.54	3.54	3.60	-	-	-	27.62
35	-	-	-	-	5.20	2.5I	I.66	8.43	5.3I	-	-	-	23.II
36	-	-	-	0.7I	6.38	0.05	-	5.06	-	0.06	0.54	-	I2.80
37	-	-	-	2.42	6.38	4.54	6.50	I0.I3	9.85	6.62	I.36	-	47.80
38	0.02	0.24	0.59	5.64	8.25	I.73	0.87	0.67	5.08	0.I9	-	-	23.30
39	-	-	-	0.I2	4.65	6.50	0.45	I.32	6.36	3.65	-	-	23.05
40	2.I3	-	-	-	I.06	6.73	4.44	8.66	5.37	I.24	-	-	29.6I
4I	0.93	-	-	0.7I	6.38	4.54	I0.04	4.93	6.62	2.I0	0.58	0.I0	36.93
42	-	0.74	0.I4	0.98	5.00	4.I7	5.95	0.73	0.75	0.03	I.36	-	I9.85
43	-	-	4.72	3.02	I.00	0.25	2.67	I.50	-	-	-	-	I3.I4
44	-	-	0.57	I.86	5.34	2.67	3.46	5.80	-	-	-	-	I9.69
45	-	-	0.72	I.74	0.77	0.84	2.79	7.40	2.55	3.08	I.79	0.39	22.08
46	-	-	-	3.73	7.46	-	2.65	0.II	-	0.75	0.54	-	I5.24
47	-	2.65	0.57	-	-	2.02	I.66	8.43	9.46	8.94	2.45	0.03	36.20
48	-	-	I.I7	0.78	5.7I	0.53	I.95	I.78	0.06	0.82	-	-	I2.80
49	-	0.08	0.43	0.59	5.20	4.II	I0.I4	6.I8	9.85	5.3I	-	-	4I.89
50	0.07	-	-	-	2.5I	7.85	3.54	I.29	-	-	-	-	I5.25
5I	-	-	-	-	0.35	5.40	9.54	5.94	-	2.85	-	-	24.09
52	-	-	0.49	2.09	2.94	5.29	8.77	4.56	3.60	-	-	-	27.74
53	0.32	-	3.47	0.49	0.83	I.64	I.25	I.62	0.48	I.27	-	0.II	II.49
54	0.I3	-	-	2.42	3.92	0.05	0.52	-	0.0I	-	-	-	7.05
55	-	-	0.00	I.I6	-	-	5.06	I0.02	9.78	4.I6	I.6I	I.28	33.06
56	-	2.65	0.43	I.86	5.34	8.77	2.67	-	-	0.57	0.82	-	23.7I

YEAR	J	F	M	A	M	J	J	A	S	O	N	D	ANNUAL
57	-	-	-	3.73	4.65	10.14	0.05	-	-	3.08	-	0.39	22.04
58	-	-	3.47	1.86	-	2.51	2.67	0.11	-	-	0.75	-	11.37
59	-	2.65	0.49	-	5.20	5.40	1.95	10.02	9.85	3.65	-	-	39.21
60	-	-	-	2.09	5.34	10.04	1.66	2.55	-	-	-	-	21.68
61	0.21	-	4.72	-	-	7.85	1.95	0.67	2.55	-	-	-	17.95
62	-	-	0.57	-	2.51	5.30	7.85	0.11	-	2.55	-	1.28	20.17
63	2.65	2.65	-	-	1.74	0.25	2.67	4.56	0.75	0.75	-	0.93	16.95
64	2.65	0.43	-	2.42	2.51	5.40	4.44	4.93	5.95	3.08	0.58	-	32.39

Table 2.I8

Erbistock effective releases (cusec days x $I0^3$) for combined record. Maintained flow 500 cusecs.

YEAR	J	F	M	A	M	J	J	A	S	O	N	D	ANNUAL
23	-	-	-	3.23	5.03	5.66	8.85	I.69	0.I2	-	-	0.07	24.65
24	-	3.98	I.43	2.52	0.05	0.45	I.22	-	-	-	-	-	9.65
25	-	-	-	I.89	I.40	5.03	II.69	II.57	8.89	-	-	0.I0	40.57
26	-	-	-	3.07	0.45	6.59	I.84	I.I7	0.I3	0.I8	-	I.67	I5.I0
27	-	0.05	-	3.23	7.63	0.76	I.I7	-	-	-	-	0.33	I3.I7
28	-	0.04	-	I.74	6.05	0.76	3.6I	2.34	4.73	-	-	-	I9.27
29	3.I2	0.95	6.26	9.80	7.63	8.23	II.I9	I.69	4.73	0.0I	-	-	53.6I
30	-	I.43	0.06	-	6.56	9.26	I.28	-	-	I.08	-	-	I9.67
3I	-	-	2.I5	I.48	0.0I	-	7.II	-	-	3.92	-	-	I4.67
32	-	3.98	0.4I	0.94	0.0I	7.II	I.28	3.25	0.04	I.08	-	0.27	I8.37
33	0.04	-	-	5.II	9.0I	5.34	7.II	II.57	II.28	2.23	2.73	2.46	56.88
34	-	3.98	0.06	I.74	2.52	6.70	II.09	4.48	4.73	0.0I	-	-	35.3I
35	-	-	-	0.06	6.56	3.44	2.48	9.98	6.I6	-	-	-	28.68
36	-	0.05	-	I.09	7.63	0.34	-	6.22	0.04	0.I3	I.08	-	I6.58
37	-	-	-	3.23	7.63	5.66	8.00	II.69	II.35	8.89	2.46	-	58.9I
38	0.II	0.49	I.I3	6.84	9.80	2.33	I.22	I.I7	6.55	0.24	-	-	29.86
39	-	-	-	0.49	6.05	8.00	0.70	I.69	7.86	4.59	-	0.07	29.44
40	3.I2	-	-	-	I.89	8.23	5.63	I0.2I	6.4I	I.67	-	-	37.I5
4I	I.67	-	-	I.09	7.63	5.66	II.I9	6.02	8.89	2.54	I.II	0.3I	46.II
42	-	I.I6	0.4I	I.48	5.63	5.34	7.II	I.28	I.24	0.I8	2.46	-	26.28
43	-	-	6.26	4.00	I.60	0.45	3.6I	2.34	-	-	-	-	I8.26
44	-	-	0.95	2.88	6.77	3.62	4.6I	7.20	-	-	-	-	26.03
45	-	-	I.43	2.52	I.39	I.40	4.05	8.85	3.25	4.I3	2.73	0.46	30.22
46	-	-	-	5.II	9.0I	0.09	3.76	0.48	-	I.30	I.08	-	20.83
47	-	3.98	I.07	0.04	0.0I	3.02	2.48	9.98	I0.96	I0.55	2.90	0.27	45.25
48	-	-	2.I5	I.44	7.26	0.76	2.99	2.33	0.I3	I.25	-	-	I8.30
49	-	0.28	0.72	0.94	6.56	5.07	II.69	7.48	II.35	6.I6	-	-	50.23
50	0.27	0.04	0.03	-	3.44	9.26	4.48	I.95	-	-	-	-	I9.47
5I	-	-	-	0.04	I.05	6.70	II.09	7.I9	0.04	3.92	-	-	30.04
52	-	-	I.05	3.07	3.86	6.59	I0.32	5.66	4.73	0.0I	-	-	35.28
53	0.I6	-	4.59	0.94	I.42	2.74	I.84	2.23	0.83	2.05	-	0.33	I7.57
54	0.37	0.04	-	3.23	5.03	0.34	I.04	-	0.I2	-	-	-	I0.I7
55	-	0.05	0.06	I.74	0.05	-	6.22	II.57	II.28	5.34	2.23	I.73	40.25
56	-	3.98	0.72	2.88	6.77	I0.32	3.6I	-	-	I.II	I.25	-	30.64
57	-	-	-	5.II	6.05	II.69	0.34	-	-	4.I3	-	0.46	27.78

YEAR	J	F	M	A	M	J	J	A	S	O	N	D	ANNUAL
58	-	-	4.59	2.88	0.01	3.44	3.61	0.48	-	-	1.30	-	16.31
59	-	3.98	1.05	0.04	6.56	6.70	2.99	12.12	11.35	4.59	-	-	49.38
60	-	-	0.03	3.07	6.77	11.19	2.48	3.25	-	-	-	-	26.79
61	0.10	-	6.26	-	-	9.26	2.99	1.17	3.25	-	-	-	23.03
62	-	-	0.95	0.04	3.44	6.59	9.26	0.48	-	3.25	-	1.73	25.74
63	3.98	3.98	0.04	0.04	2.52	0.45	3.61	5.66	1.24	1.30	-	1.67	24.49
64	3.98	0.72	0.03	3.23	3.44	6.70	5.63	6.02	7.11	4.13	1.11	-	42.10

CHAPTER 3

REVIEW OF COMPUTER LANGUAGES

3.I. Introduction

When computers were first produced, the only method of communicating with them was through their own language, which was ofter peculiar to a certain model of computer, or even to one particular machine. The instructions consisted of combinations of numbers and special symbols which bore no resemblance to the problem statement written in English and algebraic terms by the engineer or scientist.

Before a problem could be coded for a computer it had to be broken down into the most basic mathematical processes involved, and each instruction then referred to one action, such as addition of two numbers. The computer holds the numbers in which may be imagined as a set of cells, each of which has an absolute position or address in the machine core. When using machine language each possible operation which can be performed by the computer was assigned a number, and an instruction could consist of the address of a data item to be operated on and a number representing the operation to be carried out. Since most operations require two operands, a second number must somewhere be available. The usual convention makes the number standing in a special cell called the accumulator the second operand. The accumulator is also available to store the result of the operation.

It can be seen that the programmer did not then operate on variable names, as in algebra, but on the contents of a cell in the computer store which could only be referred to by its absolute address. In order to write a program in machine language it was necessary to keep a record of what variable or number was represented at each address. With a complex

program, the clerical work involved in detailing the items present in various locations became burdensome and was always open to human error. To overcome someof the labour involved in documentation and to reduce the possibility of errors, a form of language known as symbolic code was written. Instead of referring to absolute addresses, the programmer could now label data and instructions by words or mnemonics which related the information in the computer store to the normal vocabulary of mathematics. However, there still tended to be a one to one correspondence between groups of mnemonics and machine language instructions, and the program was still too detailed to resemble the original problem statement.

Because of the high speed of computers it was thought that the task of translating a problem from scientific terms into machine language could be performed by the computer itself with much greater efficiency than by a programmer. Therefore, the next step in development was to construct languages which resembled ordinary algebraic terminology. These languages are known as procedure orientated, or high level, languages and are more distant from the computer than the symbolic codes. The detailed breakdown of problems is no longer performed by the programmer, and he need not know how the computer stores its information. For the computer to be able to carry out the functions described by these languages it is first necessary to translate the instructions into its own terms.

They way in which this is done is to write a program in machine language which will accept the user's high-level program as data, and replace each statement in the high level language with a set of machine code instructions to carry out the same process. This program, known as a compiler, also controls the addressing required by the machine. Several of these high level languages and their associated compilers

3.2. Requirements of a computer language for a specialist subject

Because practising engineers do not have the time to become fully conversant with modern computing methods, and indeed, because computer installations tend to change very rapidly so that only full time users of the computer can keep up with new developments, and due to the scattered nature of literature concerning specialist procedures, engineering usage of the computer is not as widespread as it could be.
However, it is thought that if a simple, standard system of programming for a specialist subject could be designed, which remained stable as far as the engineer is concerned, and which included all the most commonly used routines of the specialist subject, then this would tend to convey more confidence to the engineer and might lead to increased computer usage, as opposed to the situation at present, where an engineer might feel that he would rather carry out an approximate problem solution by hand than spend time searching for a relevant routine in the literature and then searching through language manuals to find the necessary input and output procedures for data.

In the past few years there has been an increasing tendency for Engineering departments in Universities to teach the basics of computer usage to undergraduates, and, as a consequence, it is felt that in future more practising engineers will look to the computer to solve their problems, and so there is an increasing urgency to provide a well documented set of routines, implanted in simply structured language. It is thought that the knowledge that such systems exist at computing installations available to industry, and that simple documented usage manuals have been written, is more important than the basic knowledge of one or two computer languages such as Fortran or Algol. It is suggested that interest is soon lost in the usage of computers if

engineers cannot obtain the well tried procedures they require without the frustration of weeks of searching.

A simple computer language for use by a specialist in a particular field of engineering should require that the user states his problem and gives his data in essentially the same form as he would to a human assistant. In the same way, the user should be able to specify various courses of action to take depending upon the results obtained at any stage in his problem, and since the human capacityto make decisions based on only partially defined facts cannot be built into the computer, some form of temporary or permanent filing of intermediate results for possible use by later calculations must be provided, so that a human decision can be made without having to reconstruct the data on cards or paper tape. Facilities should also be provided for editing input data held on files so that ammendments or corrections can be made.

Since a user's problemwould generally be solved using several specialist routines, some of which would require as data results from previous routines, the form of the data input and output by each routine must be acceptable to all other routines which might use the same data.

3.3. Existing methods of solving specialist problems

3.3.I. A compatible subroutine library

The Fortran and Algol languages provide the facility to construct a library of routines for a private user, and it is possible to write each routine so that the form of the data is the same for each one. Thus, the routines would be obeyed by simply writing down in a normal Algol or Fortran program the routine names, in the required order of calculation, along with the parameters, or data items necessary for the running of the routines. However, the reading of data into the correct areas in the machine core, and the outputting of results, is left to the user.

With large amounts of data it is not possible to hold all information in the computer core and it is generally in this situation when problems occur for a user. With a complicated program, it may be necessary to run the same sequence of calculations with several data sets and then to carry out further calculations with one of the data sets, to be selected by an analysis of the first results. In this case, where the data to be used for the second sequence of calculations cannot be specified beforehand, it is impossible to arrange the input data in the correct order for a simple Algol or Fortran program, using only the basic input devices, if all the data cannot be held in the core at the same time. The problem of reading data in a random order can be overcome by the use of magnetic tapes or discs as input devices, but this assumes a knowledge of the method of storage on magnetic media and involves keeping a directory of the positions of data blocks on the files. Even using tapes and discs it is difficult to insert identifying information in the data, and the storage of text, for use as titles in outputting results, poses special problems.

It would not be possible to write general routines, for inclusion in the library, to read and file away data, without including an unwieldly number of parameters to cover all the different types and forms of information necessary to a complicated system.

Since it is the aim of a specialist computer language to take some of the burden of machine communication from the user, and to allow him to present his problem in familiar terms, without the necessity to manipulate his data for the machine's benefit, it is thought that the library of compatible subroutines is still too far orientated towards the machine to be of great benefit to the casual user.

3.3.2. Packages

A further step towards user-orientated languages is provided in extremely specialised fields by some computer manufacturers. These 'languages' are usually known as soft ware packages and include packages for Traffic Engineering, Fluid Distribution Network Analysis, Pipe-stressing calculations, Continuous Beam Analysis and Power Systems Analysis.

Each package consists of several programs or routines, including special reading routines, which act upon the numerical data provided in an order specified by the user. The user's input usually consists of several sets of data matrices, identified by standard names, and some simple commands which are recognised by the reading routines.

The commands convey to the package which routines are to be used and which data are to be manipulated. In general, the form of data necessary for one package may be completely different from that required by another, although some standardisation has been attempted between packages in the same special field, where a user might need to use the results from one package as input data for another. In these cases, the

user is allowed to file his results from the first package on magnetic tape and then, in a separate job, read back the data into the second package.

3.3.3. Vehicle Scheduling Package

The ICL I900 Vehicle Scheduling Package does not act in the same way as most of the otherpackages, since it is a complete high level language, with a similar structure to that of Algol.

The user's program, written in the special language, is presented to the Vehicle Scheduling compiler for direct translation into machine code, in much the same way as an Algol program is translated.

This type of language is on thesame level as Fortran and Algol, and only adds yet another language to the many which exist already.

3.3.4. I900 Control and Simulation Language (CSL)

The original CSL language was developed jointly by IBM United Kingdom Ltd. and Esso Petroleum Co. Ltd. for the purpose of carrying out simulation exercises, and was adapted for use on the ICL I900 series of machines.

The CSL system makes use of the facilities provided by Fortran and a program written in CSL language could contain blocks of code written entirely in Fortran, augmented by the special statements peculiar to CSL. The user's input is scanned by a special program or processor which assembles an equivalent Fortran program from the information provided. This program is then presented to the Fortran compiler for translation into machine code and for subsequent execution.

When the CSL processor reads the user's program it must recognise whether the input is Fortran code or a special CSL command. If the input is Fortran code then it is copied directly to the file where the equivalent Fortran program to the user's input is being assembled, but if the input

is a CSL command then this must be replaced by a Fortran statement or set of statements which will carry out the procedure implied by the CSL instruction. This pure Fortran code is then written to the assembly file.

The declarations, or reservation of storage space, for all data is left to the user, as is the selection of the correct input and output devices for thenumerical data.

The effect of CSL is to make available a library of specialised simulation routines and to make their use easier by simplifying the method of using them in a Fortran program.

3.3.5. SIMON (Simulation language)

SIMON was written by P.R. Hills at Bristol College of Science and Technology and provides the same facilities in Algol as does CSL in Fortran. However, Simon is not a language in the grammatical sense, since pre-translation of a program containing Simon routines is not necessary before it is presented to the Algol compiler.

Simon consists simply of a set of routines for manipulating the membership of lists or queues and for generating random inputs to a system from given frequency distributions. The routines are called for use from the Algol program in the same way as normal Algol routines, and are incorporated in a program by inserting card decks declaring the routines at the head of the Algol program which uses them, so that the user of the Simon routines needs to be a competent Algol programmer.

It is thought that Simon might be of use in reducing the possibility of logical errors or omission of clauses in an Algol program for simulating a complex system.

3.3.6. GENESYS

Genesys was developed by Alcock Shearing and Partners for the Ministry of

Public Building and Works, and is administered by the Genesys Centre at Loughborough University. It was originally written to provide a simple user language for structural engineers, but the concepts involved are extendable to other fields of engineering.

In principle, Genesys is similar to CSL, in that the user's program is read by a processing system which translates it into Fortran for presentation to the Fortran compiler. However, Genesys includes many more facilities than CSL, and recognises a far greater number of commands. Furthermore, a user's data is presented in the form of tables which are read by special routines, and which may be edited and filed on magnetic media as desired.

Genesys is by far the most versatile and simple user-orientated system available at the present time.

A user's program consists of a set of commands, some of which concern the reading of the correct data tables into themachine, and some of which specify the calculations to be performed on the data. Following the commands are the tables of data, each identified by a title, and each having at least two standard column headings. The data itself is listed under the relevant column headings.

The commands of Genesys are used in the same way as calls on compatible subroutines in Fortran, and any Genesys program may contain Fortran code and ordinary subroutines. The Fortran facilities for altering the course of programs are available.

The major advantages of Genesys over Fortran lie in the manipulation of the data tables and the provision of a virtual store. The latter facility allows a programmer writing a routine for inclusion in the Genesys library to imagine that the computer has a very large storage capacity, and reduces many of the problems involved in the handling of large amounts of data.

When the user's program is presented to Genesys, the tables are temporarily filed away automatically on magnetic media, and are later accessed by quoting their titles in data reading commands. The tables may be accessed in any order and tables may be used which were produced as results from or input as data to another program as long as they were permanently filed at the time by using the Genesys editing and filing facilities. Tables may contain data in several forms; an item of data may be given as a numerical value, as an arithmetic expression, or if a value is not known, or will vary when the table is used several times, then it may be given as a variable name. In the latter case, the user assigns a value to the variable name before the table is used.

Various facilities exist for reducing the amount of data to be written in a table, and these are all described in the Genesys reference manual, but for the purpose of giving an example of a Genesys program, one particular facility is useful. If a data item under one column heading is the same as the item above it then the '=' sign may be used to represent 'ditto'.

Genesys is composed of many different problem solving groups of routines in the form of a library.

Each group of routines is called a subsystem, and the user must state at the head of his program which subsystem he intends to use. The commands available to the user under a particular subsystem are relevant only to that subsystem and may not make sense to another subsystem.

Example

```
*GENESYS
*START  'CØNTINUØUS-BEAMS'
*TABLES
'B67'
LENGTH,,M  BREADTH,,MM  DEPTH,,MM
  4.5          300          400
  4.5            =            =
  5.8            =          500
'LIVE'
SPAN     LØAD,,KN/M
I,2        25.8
*MASTER
APPLY LØAD CASE 'LIVE' TØ BEAM 'B67'
ALLØW SETTLEMENT ØF 6,,MM AT SUPPØRT 2
PRINT BENDING MØMENTS
*FINISH
*EXIT
```

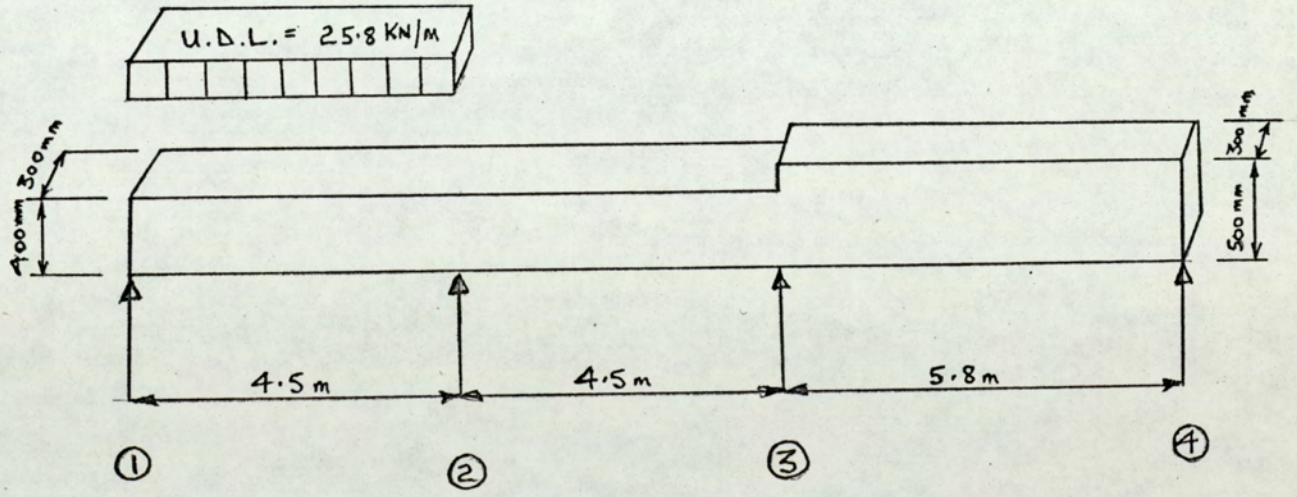


Fig. 3.1.

The GENESYS card simply tells the computer operator that the Genesys system is to be used.

The second line is a command to choose the subsystem required. In this case the system dealing with continuous beams is required.

The * TABLES commands tells the system that the information between * TABLES and the next command beginning with an asterisk are the data tables required by the sybsystem.

Each table begins with thetitle of the table in inverted commas.

The line after the title contains the standard column headings for that particular data table. The headings are each followed by an optional mnemonic indicating the units relevant to that heading.

After the last table the next commandis * MASTER. This command introduces the problem solving commands for that subsystem.

The first line

APPLY LØAD CASE 'LIVE' TØ BEAM 'B67'

tells the subsystem which tables are to be used.

In this case the uniformily distributed load of 25.8 KN/M is to be applied over the span connecting supports I and 2, as described in table 'LIVE'.

The beam itself is described in table 'B67' and is shown in Fig.3.1.

The second line is self explanatory, and allows support 2 to settle by 6 MM.

The bending moments resulting from the load and settlement are now calculated.

The third line tells the system to print the results on the line printer.

The next command * FINISH tells Genesys that the user has performed all his calculations with the subsystem 'CONTINUOUS-BEAMS'.

After * FINISH the user may * START another subsystem or he may write * EXIT which terminates the Genesys run.

3.3.7. Natural language problem-solving systems

Some steps have been taken recently in the field of artificial machine intelligence to produce systems which can understand natural English.

Gelb, of the IBM Corporation, New York, has developed a program called Happiness which is able to solve problems involving probability. The user's problem is input in English, and the program then breaks down the grammatical and idiomatic structure of the input to produce a precise mathematical representation of the problem. The problem may then be solved by the computer.

Winograd, at MIT, has written a program which carries out actions and answers questions about a simple world containing several objects, which include a table, a box, and some blocks and pyramids, stored as a data structure inside the computer and displayed on a television screen to the person conversing with it. The world also contains a 'hand' which is able to pick up and move the objects. The system is known as SHRDLU, and the user may interrogate the system concerning the state of the world at any time, and he may give instructions to alter the state of the world. The user's instructions are not only analysed grammatically, but any instruction is analysed in the context of the previous conversation. This facility makes SHRDLU one of the most advanced language understanding programs yet produced.

However, the nearer a system becomes to understanding conversational English, the more costly becomes its use, and the more likely it becomes for the user to miss out information. It is felt that an engineer should present his problem to a computer in a more mathematical and exact manner if errors are to be avoided.

The use of natural language understanding programs would be better applied to interrogation and restructuring of information filing systems.

3.3.8. HYDRO

A simple language for water resources systems, called Hydro, was constructed by Bugliarello and McNally at the Carnegie Institute of Technology in 1966. The language is not machine based, since the Hydro program translates the user's input into ALGOL, which is then re-translated by the Algol compiler into machine language, ready for execution, so that it carries out a similar process to that of CSL and GENESYS. Although Hydro was originally meant to solve problems only in water engineering, the structure of the system is such that any routine, in any field of science and engineering, can be incorporated into the language.

The Hydro language has been adapted by the author for the ICL 1900 series of machines and is described in the following chapter.

3.3.9. SLANG

SLANG is an analogue computer simulation language developed by Hawker Siddeley Dynamics. This language simulates the operation of a differential analyser. This in turn is a device for solving sets of simultaneous ordinary differential equations. It has been shown elsewhere in this thesis that the reservoir operation problem can be described by a set of simultaneous differential equations. One has been prevented from using ordinary analogue computers for reservoirs problems because of the complex function generators required to simulate data sequences and control rules. To a large extent digital simulations of analogue computers overcome this problem because almost any type of function can be specified. One still has difficulty in specifying control rules, especially of the type used in reservoir regulation where amenity and recreational factors are taken into account.

3.3.IO Ascop

Ascop is an integrated system capable of performing a wide range of data editing operations as well as many of the standard statistical analyses. An Ascop program consists of instructions and data matrices, the instructions indicating to the system which analyses are to be performed on given data sets.

The data matrices are arranged so that the columns represent variables and the rows represent observations on the values of these variables. Each matrix is assigned a name and number by the user and he also provides the column, or variable, names by means of a special statement.

The user normally instructs the system to read in the data matrices and store them on magnetic tapes or discs as a data base for use by the system. The editing operations may then be used to check the data and to specify the subsets that are required for analysis. The subsets may be stored with theoriginal data as new matrices, thus augmenting the data base, or created solely for the purpose of analyses. In the latter case the subset may be envisaged as a matrix which has no physical existence, but only exists in the mind of the user for the limited period that the data is to be used.

The Ascop language has been designed so that the instructions follow normal English as far as possible, and there exist several ways of saying the same thing, all recognisable to the system.

Ascop is a very useful tool for carrying out statistical analyses and for performing arithmetic and editing operations on matrices, but the data structure is so rigid that the system is not readily extendable to other fields.

CHAPTER 4

THE HYDRO LANGUAGE

4.I. Introduction

The Hydro language was the result of a four year pilot effort intended primarily to explore the issues arising in the development of a comprehensive problem-orientated language for the field of water resources, and to provide a framework for rapid future expansion on a library of useful computer routines. A first choice facing the designer of a language is whether or not to subdivide the water resources field into several subfields, such as frequency analysis, precipitation analysis, and channel flow, and to provide for each a problem-orientated language with a structure and data format most suitable to it.

Subdivision has merit when intercommunication between the different sub-areas is limited. For instance, the Integrated Civil Engineering System (ICES) developed at M.I.T.is an assembly of different sublanguages for surveying, structural frame analyses, soil stability analyses etc. The ICES components are integrated into a system by a general compiler, which understands the contents of the component subcompilers and makes them operate jointly. In the case of water resources there is a considerable amount of exchange between the possible sub-areas entering into the solution of a typical problem. The determination of design spillway flows may begin with the analysis of precipitation and proceed through flood routing, which requires open channel flow procedures, and frequency analyses. If separate compilers were used for each of these sub-areas, the user would be required to learn the programming instructions for each to perform a simple routine analysis. Furthermore, if subfields are defined on the basis of type

of problem and corresponding data structure their number would be very large. In this respect, the situation in water resources is far more complex than, say, structural frame analysis, where one deals with many different geometries, but where, once these have been described, the solution is given by the same equations for every frame structure.

The above considerations militate against the subdivision of a water resources language into sub-languages. Thus, Hydro has been designed as a comprehensive language, with a data structure valid for the entire range of water resources problems it covers. This solution is in the direction of greater integration and has potential for further expansion; but it does lead to a slightly more general data structure, which is less responsive to the needs of a given sub-area. On the other hand the user only has to learn one data structure, andis given a tool of considerably more power and flexibility.

The flexibility in Hydro is enhanced by the fact that the Hydro processor which converts the user's program to the universal language of Algol, has only the task of organising the code that will carry out the solutions, but unlike the ICES system it does not have the capability in itself of producing problem solutions. The routines which actually solve the problems are contained in a library, and can be withdrawn, altered or added to much more easily than if they were incorporated directly in the processor. The existence of a library of procedures separated from the processor also facilitates the building of decision links into the language and the use of the language in an on-line conversational mode.

The decision as to the basic structure of the language must be followed by several accessory ones : for example, the amount of logical decisions to be entrusted to the user versus that entrusted to the

language, the size of the individual routines of the language, and the criteria to be followed in maintaining, revising and augmenting the language. In the actual design of a procedure, it must also be decided how general or how specific the procedure should be. A procedure may be designed as a compromise acceptable to most potential users, or several different procedures could be constructed, each satisfying the needs of a particular group of users. In Hydro, both approaches have been followed, according to circumstances.

The development specifications which have guided the design of Hydro demanded that Hydro should be much simpler to program than Fortran or Algol but should have sufficient flexibility to handle different types of problems so as to cover a broad area of water resources. The commands and data have been structured toward the natural language of water resources and there are automatic data transfers between commands. The routines available to Hydro have been written so that the user may choose to use them separately, one after the other, or, when possible so that he may request one complex routine which automatically calls for other routines in an order dictated by intermediate results.

In order that the system be easily augmented, the processor and its associated libraries have been designed to facilitate the design of procedures and the addition of new procedures to the library, or the removal and substitution of existing procedures.

4.2. General description and operation of the System

The main features of the Hydro System are the translator program and the line library.

From the instructions and data supplied by the user, the translator assembles an equivalent Algol program using lines of Algol code stored in the library. The generated program is then presented to the Algol compiler for execution. In addition, the system employs a general working area to which the translator writes the generated program with its associated data, and from which it runs. A further component of the system consists of a copy of the translator program which is used only when alterations or additions are made to the system. This copy is necessary in order to avoid corruption of the working translator if incorrect alterations are made.

All components of the system are located on disc and tape storage facilities when the system is not in use, but when a user submits a program to Hydro all components are transferred to disc devices only, since operations are much more rapid from discs. A more practical consideration is that if the line library was accessed on magnetic tape the multiple rewindings which would be necessary, would quickly cause a deterioration in the quality of the tape.

Although the system was developed on an ICL I905 computer using Algol-60 and the I900 Algol input/output procedures for the transfer of data between the computer fast store and the peripheral units, such as card readers, tapes, printers and discs, it can easily be adapted to run in any language and on any computer installation with at least 32K storage space, as long as there are disc or magnetic tape backing store facilities, and provided that procedures exist or can be written to

perform similar tasks to the I900 input/output system.

A more detailed description of the Hydro translator and the special data reading procedures used by the system may be found in Reference (2).

4.3. Requirements of an Algol program

In order that the operations of the Hydro translator be understood it is necessary to be conversant with the requirements and structure of an Algol program. The method of constructing Algol programs may be found in any computer installations manuals, but a brief description of some of the major requirements will be given here.

So that the computer may reserve sufficient storage space for all the variables which may be used in an Algol program, they must be declared prior to their use. Single variables may take either integer values or continuous decimal, or real, values. Integers require only one storage space in the machine but real numbers require two spaces, since a real number is stored as a fraction between zero and I plus an exponent. The declarations take the form

'INTEGER' variable name,

or 'REAL' variable name.

Declarationsof array variables, which represent vectors and matrices, must include the size of the array as well as the name. Arrays may be of any size and any number of dimensions. Array elements are referred to by subscripts after the array name. The size of the array is declared by quoting the range of each subscript used.

'INTEGER' 'ARRAY' name [-4:I2,0:I3]

or 'REAL' 'ARRAY' name [-4:I2,0:I3]

will declare an integer or real array called 'name' having I7 × I4 = 238 elements. The first subscript ranges from -4 to I2 and the second from 0 to I3 in steps of I. The two figures separated by a colon and giving the range of the subscript are termed a bound pair. Individual bound pairs are separated by commas, and the whole list is enclosed in square

subscript brackets.

As well as declaration of variables, it is also necessary to declare any procedures, or routines, which are to be used. The procedures consist of blocks of code which may be used several times in a program. Instead of having to write out the whole block of code every time it is used, Algol provides the facility to name the block of code, so that only the name of the block needs to be written down when the code is required. The naming of the block is performed in the declaration, which, in the same way as variables, must precede the use of the name in the program, i.e.

```
'PROCEDURE' name ;
    'BEGIN'
        code
    'END';
```

The writing down of the name of the procedure in an Algol program now implies that the whole block of code is to be used at that point, and is known as a call on the procedure.

The data for an Algol program iscompletely separated from the program and is presented as a separate document to the machine.

Any program assembled by the Hydro system has several standard procedures declared at the head of the program. These are blocks of code which are able to read in the data for the program in the form that it is given by the user. The appropriate calls on these procedures read in the user's data in the correct order and assign the data to the correct variables. These procedures have a list of parameters after the procedure name, and it is the parameters which tell the procedure the names and sizes of the arrays with which it is dealing.

The Hydro system automatically sets up the correct parameters to read in

a particular set of data.

4.4 Bugliarello's HYDRO

The version of the HYDRO system available to the author,as written under the direction of Bugliarello at the Carnegie Institute of Technology,was in a form of ALGOL that was not directly usable on the ICL 1900 series of machines,and,because ALGOL does not contain standard reading and writing routines,so that different input and output software tends to be provided for the various types of computer,new routines,incorporating the 1900 ALGOL reading and writing procedures,had to be written to perform the equivalent operations to Bugliarello's program.Also,several grammatical constructions used by Bugliarello were not available on the 1900 machine, but,when the intention was determined from the context of the surrounding program statements,code could be written to carry out similar tasks.

Because of the complexity of the HYDRO translator,Bugliarello's flow diagrams and his related description involved extremely lengthy documentation and it proved a difficult task to formulate an accurate overall picture of what the translator was actually doing.However,by breaking down the program into several self-contained groups of statements,the action of each was determined and,by re-assembling these groups in their various combinations, the general principles were realised.It then became possible to start re-writing the program with some confidence,and it was found that Bugliarello's version contained several logical and grammatical errors which would not allow thesystem to operate in the intended manner.One error was found impossible to correct without restructuring the system to some extent and affected the flexibility of the language.It concerned the method of dealing with array declarations and the full implications of the error are detailed at the end of this chapter.

This chapter describes the HYDRO system written by the author to perform the same operations as Bugliarello's language,and does not include the revision necessary for dealing with arrays.The method of overcoming the error is dealt with in the next chapter where the author's complete revision of the HYDRO language is described.

The following description of the translator and line library should be read in conjunction with the examples given later in the chapter.

4.5. The Line library

The line library contains code for all of the Hydro procedures and procedure calls, the variable and array reading code, declarations of all variables and the declarations of the special data reading procedures used by the system. For the convenience of the translator program, which has to calculate the position in the library of the code necessary to assemble the final Algol program, all code of a similar nature is grouped together. For instance, all array declarations are together and each is composed of the same number of lines so that it is only necessary to know the position of the first declaration in the library in order to find the array declaration required, since every array variable is given a unique reference number which determines its relative position to the first one. The first material on the library is the declaration of all single variables which may be used by an assembled program. All possible single variables are declared at the head of every assembled program because the storage space involved is small and thereforeit is not necessary that they be declared as they are required.

Next on the library are the special data reading procedures which assign data to the correct variables when the program is executed. These are also declared at the head of every program assembled by Hydro.

Following this is a short section of code which is necessary to initialise certain variables used by the reading procedures.

After this preliminary code are the lines which read data into all the single read-in variables used by the Hydro procedures. These consist of calls on one of the data reading procedures. There are the same number of lines of code for each variable.

Following the single read calls are the sections of data reading code for each of the read-in arrays which may be used in Hydro. Each section of code consists of the array declaration and a call on a data reading procedure.

After the array read lines come the sections of code representing material which must be assembled globally to any of the procedures. This code mainly consists of array declarations for arrays not of the read in type, the calculations of bounds on internal arrays and array initialisation procedures.

The main section of code remaining, which is the largest section, contains the bodies of all procedures which exist in the Hydro System. After each procedure body is a call on that procedure. The final section of code contains a list of Algol 'END'; lines which are used to terminate the assembled program.

Diagram of line library

I. Declaration of global single variables used by Hydro and any general array declarations which are used by all Hydro programs

2. Declaration of the seven data reading procedures

3. BAS I card
Single variable reading procedure calls

4. BAS 2 card
Declaration of arrays and
calls on array reading procedures

5. BAS 3 card
Declaration of global material necessary for some procedures

6. BAS 4 card
Procedure bodies and
calls on procedures

7. Algol 'END' cards for terminating program

A copy of a line library which includes only a few procedures and arrays is given in Appendix 4.

4.6. The Translator Program

The translator program is split into three distinct components or phases. The first phase of the translator, which is automatically entered when a user submits a program to the system, reads, one by one, the input cards, each of which must contain either a procedure name, variable names and associated data or data only. By comparing any procedure or variable names which occur with a list of permissable names, the translator is able to check for spelling mistakes or non-existent names which may occur. The cards are also checked for grammatical errors, such as misplaced commas or decimal points in data. All input cards, except those which contain procedure names are written away to a disc file to be used later as data by the generated Algol program. When a procedure or variable name is recognised by the translator it places a number corresponding to the procedure or variable into a list which is used by the second phase of the translator. If any errors have been detected in phase one, relevant error messages are written to the lineprinter, detailing exactly where the error occurred and explaining what may have happened. The execution of the Hydro system is then terminated since the operation of the next phase of the translator assumes that no errors were found in the previous phase.

For the program which the system assembles to run correctly, then all of the necessary data must be present and all the procedures and variables which are required must be declared before they are used. Also the data supplied must be read in the correct sequence and assigned to the correct variables. The checking needed and the assembly of code to perform the required actions constitute the main function of phase two of the translator.

Phase two, therefore, performs the major task of the Hydro system, since

this is the part of the translator which actually decides which lines of Algol code must be moved from the library to solve the user's problems.

Data for a Hydro procedure may be supplied in two ways :

(a) All of the data necessary for the running of a procedure may be given directly after the procedure name in the input deck, or

(b) some data may be omitted from one procedure because it may be calculated by the operation of an earlier procedure upon its own data, or the data required may be the same as that given to an earlier procedure.

In the case where data is calculated by a previous procedure, the number corresponding to this data variable will not have been placed in the list of procedures and variables by phase one since it does not appear in the input deck.

Similarly, there are several internal procedures which are not available to the user but are used by other procedures, so that the numbers for these internal procedures do not appear in the input list. In some cases it would be possible to declare these internal procedures within the bodies of the main procedures, but this would lead to an unnecessary duplication of effort, since the same internal procedures may be used by several external procedures. In addition, some external procedures call other external procedures, and although if all the data is given for the major procedure it will automatically satisfy the smaller procedure, the number for the auxilary procedure will not appear in the list unless it has been used in its own right.

Therefore, it is necessary to use the input list in some way in order to check that for every procedure number which appears there, all the data variables required by it have been given in the input deck, or that the necessary data may be calculated from the data given to a

previous procedure, and to check that all of the procedures necessary to the solution of the problem will be available during execution of the assembled program. Since these checks can only be carried out on the input list then continuous additions to it will be necessary, so that when checks are made, the numbers representing procedures and variables which are used or generated by the program, but not given by the user, will be present in the list. This modification is carried out by special blocks of code which are written into the translator for every possible procedure available to the user.

When a procedure number is encountered for the first time in the input list, the variables calculated by it and any auxiliary procedures which it uses must be placed into the top of the list, since these are now available as data for later procedures in the same way as if they had been supplied by the user. Also, the numbers of the auxiliary procedures must be placed in the list. This modification of the input list is necessary for two reasons. Firstly, if allows the translator to check that all variables required for the running of this and later procedures will be available and, secondly, a check may be made whether this is the first time that a variable or procedure has been used. This second point is used when procedures or variables have to be declared. If procedure or variable numbers have not appeared before, then they must be declared at this stage, but if they have appeared before then no declaration is necessary since they will have been declared at their first occurence. As well as declaring procedures and variables if necessary, code must also be assembled to read any data which the user has given immediately following the procedure being processed. The reading of data is performed by special reading procedures which are incorporated into every program assembled by Hydro. Therefore, for every data variable which appears in the input under a particular procedure

all that is needed is a call on the reading procedure relevant to that data variable.

After all of the declarations of variables and other procedures used by this procedure have been made, then the declaration of the main procedure itself may be carried out, followed by a call on the procedure.

Now, if the main procedure has been employed before by the user then the above checks for declarations will not be necessary since they will have been carried out the first time the procedure was used, and all that is necessary is the assembly of code to read any new data for the procedure and a further call on the procedure.

As in phase one, if any errors occur, such as missing data or procedures then relevant error messages are printed out and the Hydro system terminates. Phase two assembles the necessary code by effectively noting down the serial numbers of the lines of code required from the line library to perform the tasks of declaring variables and procedures, reasing data into the variables, and calling the procedures, as well as any code necessary for special operations such as initiallising arrays.

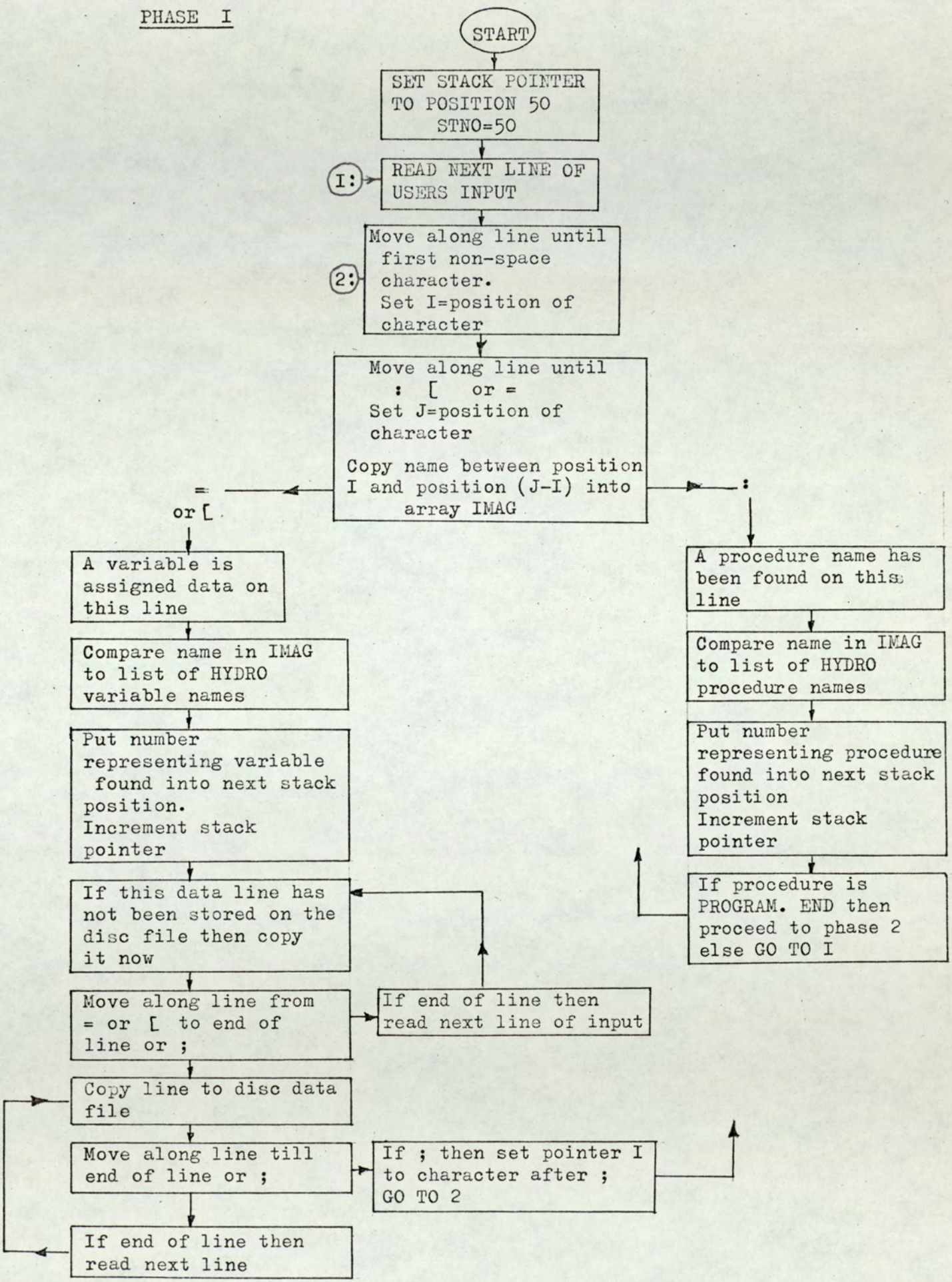
Phase three of the translator is the section which actually writes the assembled program which will perform the tasks required by the user. The lines of code required, whose line numbers in the library have been calculated by phase two, are retrieved from the library and copied onto a disc file in preparation for presentation of the assembled program to the Algol compiler for execution. The data for this program, which is simply a copy of those input cards containing data, has already been written to another disc file by Phase one.

All that remains is for the computer to execute the program and to print

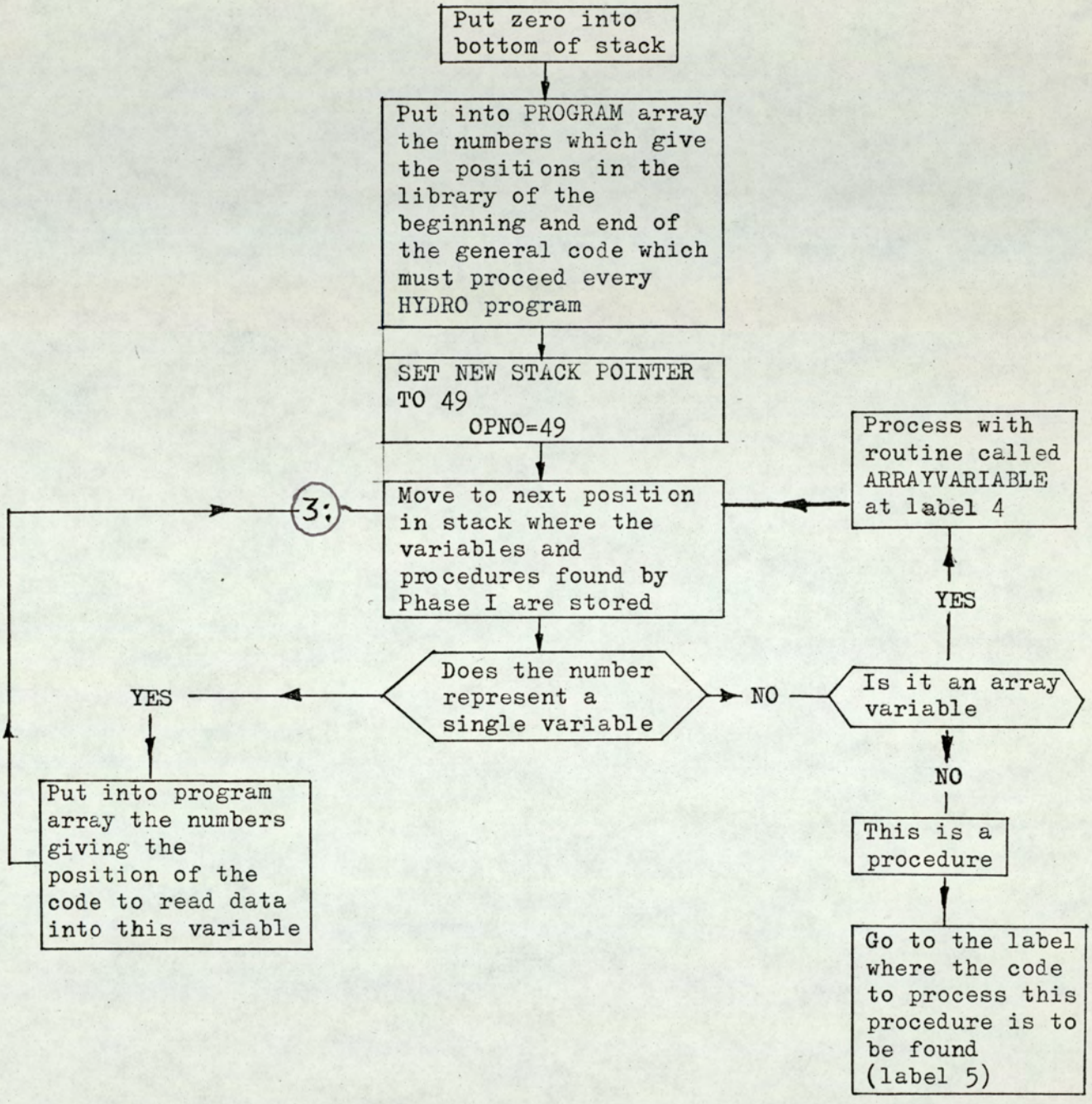
out the results on theline printer.

It is still possible for errors to occur in the execution of the users' program since, although the translator checks that all variables and procedures required are present, the user may have supplied too much of too little data for a particular variable. Because of the special data reading procedures used by the system, this type of error may be detected and an error message will be output to the line printer.

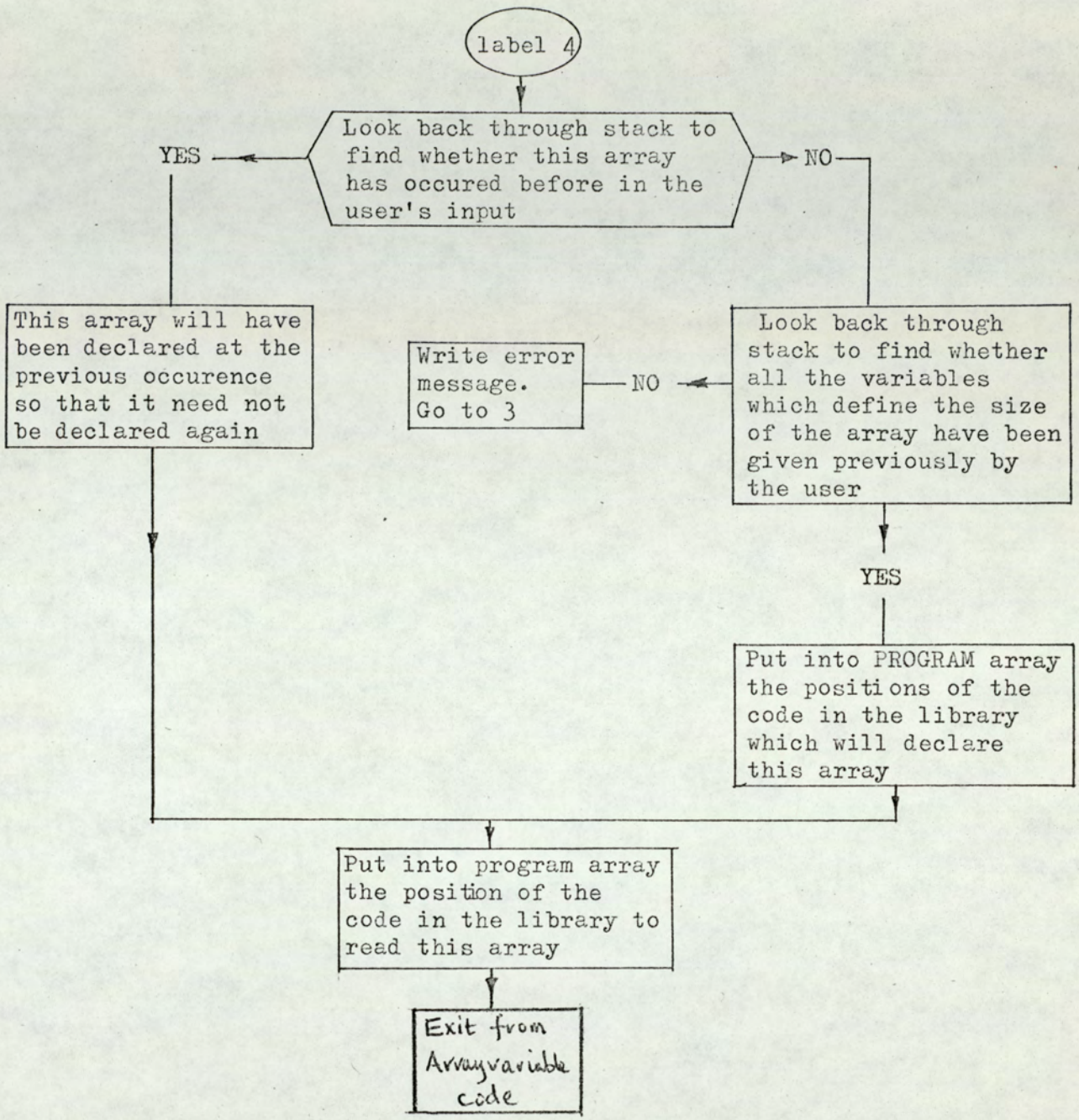
PHASE I


START
SET STACK POINTER TO POSITION 50
STNO=50
I:
READ NEXT LINE OF USERS INPUT
2:
Move along line until first non-space character.
Set I=position of character
Move along line until
: [or =
Set J=position of character
Copy name between position I and position (J-I) into array IMAG
=
or [
:
A variable is assigned data on this line
Compare name in IMAG to list of HYDRO variable names
Put number representing variable found into next stack position.
Increment stack pointer
A procedure name has been found on this line
Compare name in IMAG to list of HYDRO procedure names
Put number representing procedure found into next stack position
Increment stack pointer
If procedure is PROGRAM. END then proceed to phase 2 else GO TO I
If this data line has not been stored on the disc file then copy it now
Move along line from = or [to end of line or ;
If end of line then read next line of input
Copy line to disc data file
Move along line till end of line or ;
If ; then set pointer I to character after ;
GO TO 2
If end of line then read next line

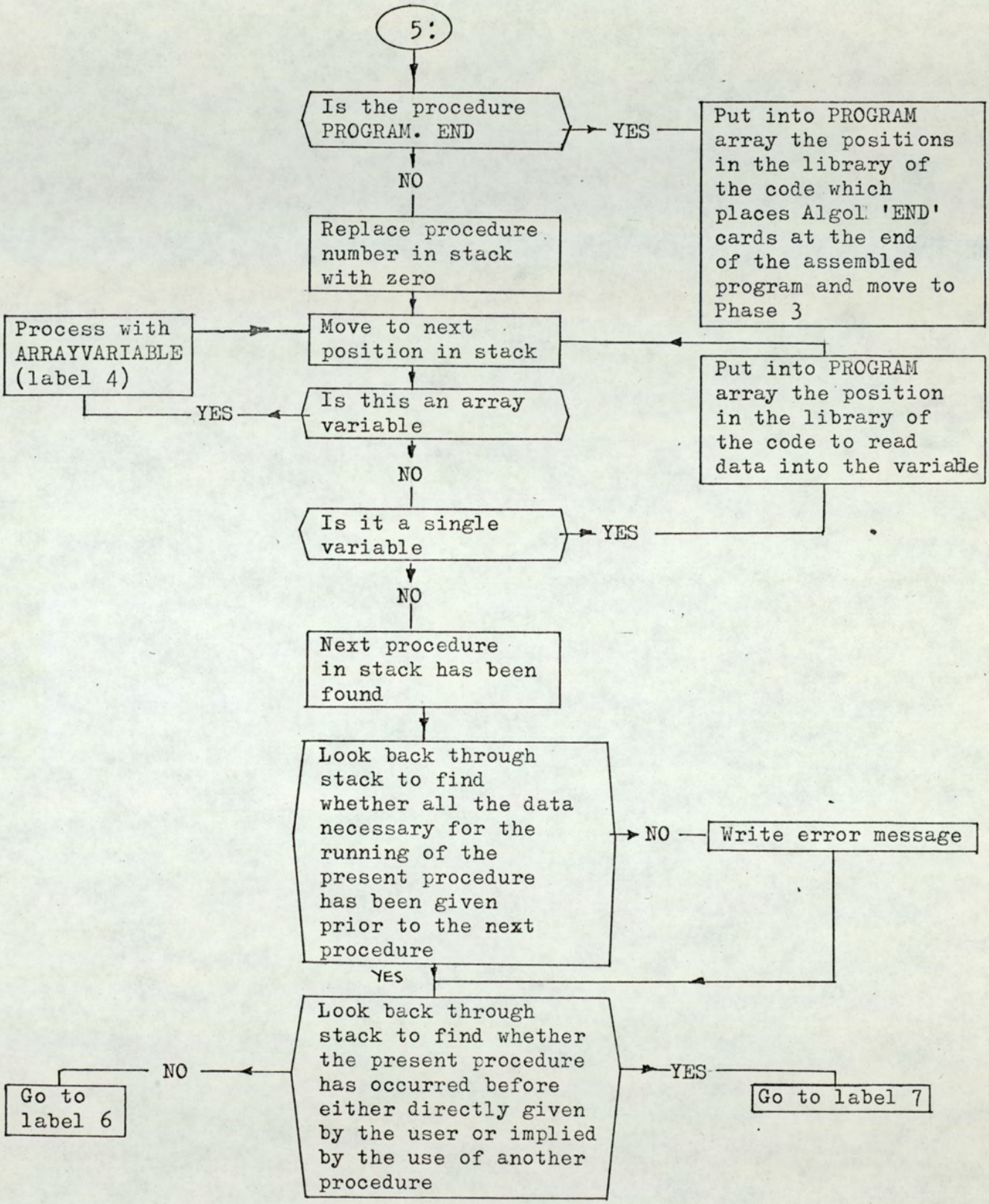
PHASE 2

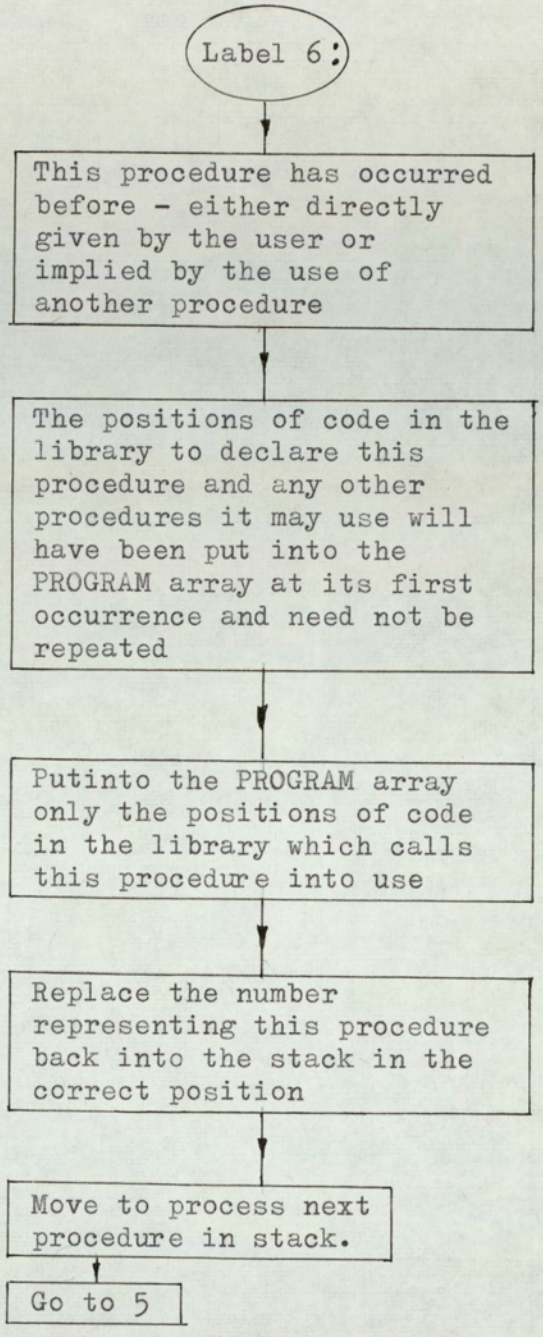

Put zero into bottom of stack
Put into PROGRAM array the numbers which give the positions in the library of the beginning and end of the general code which must proceed every HYDRO program
SET NEW STACK POINTER TO 49
OPNO=49
3:
Move to next position in stack where the variables and procedures found by Phase I are stored
Process with routine called ARRAYVARIABLE at label 4
YES
Is it an array variable
NO
Does the number represent a single variable
YES
NO
Put into program array the numbers giving the position of the code to read data into this variable
This is a procedure
Go to the label where the code to process this procedure is to be found (label 5)

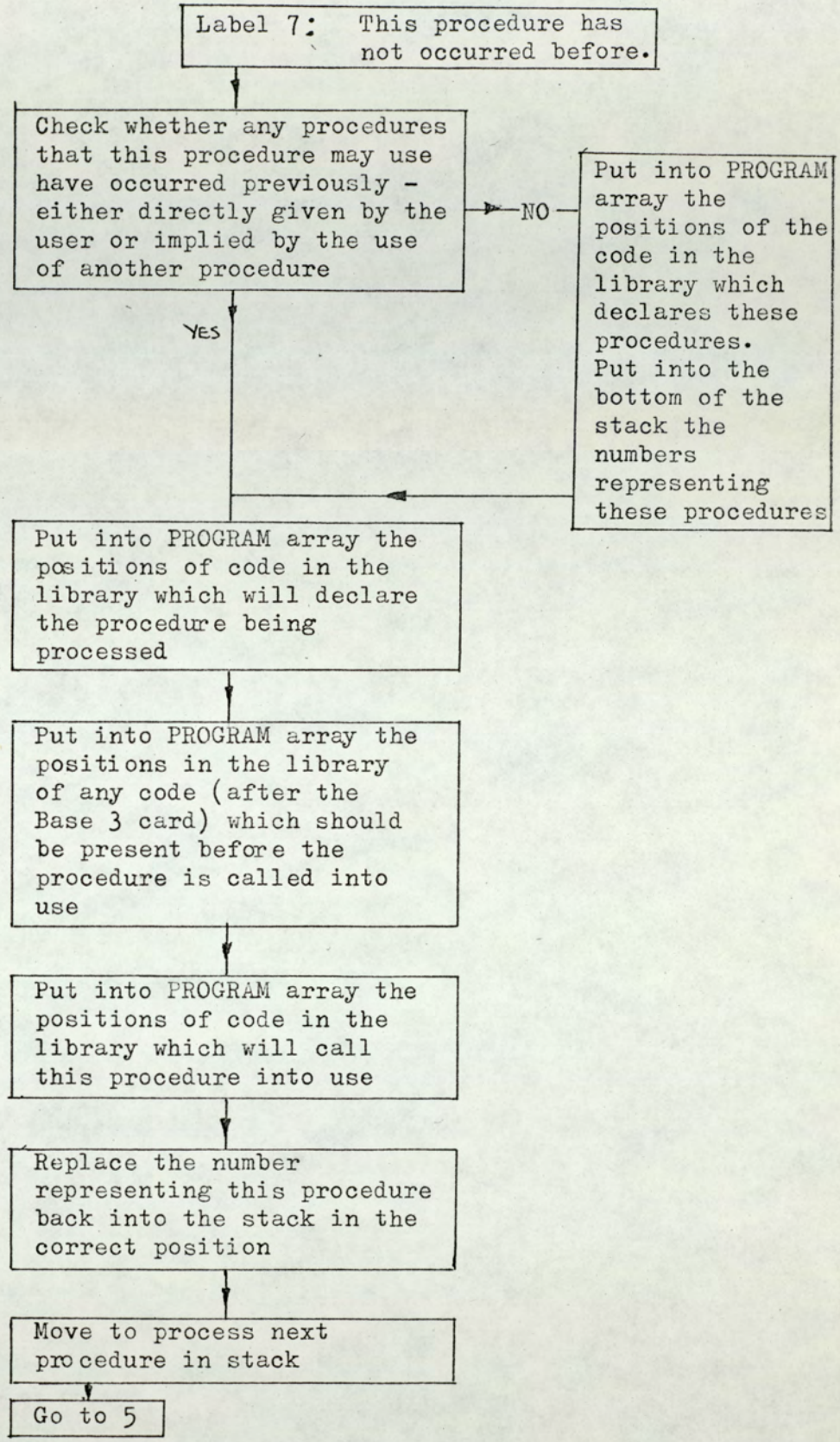
Code to process an array variable



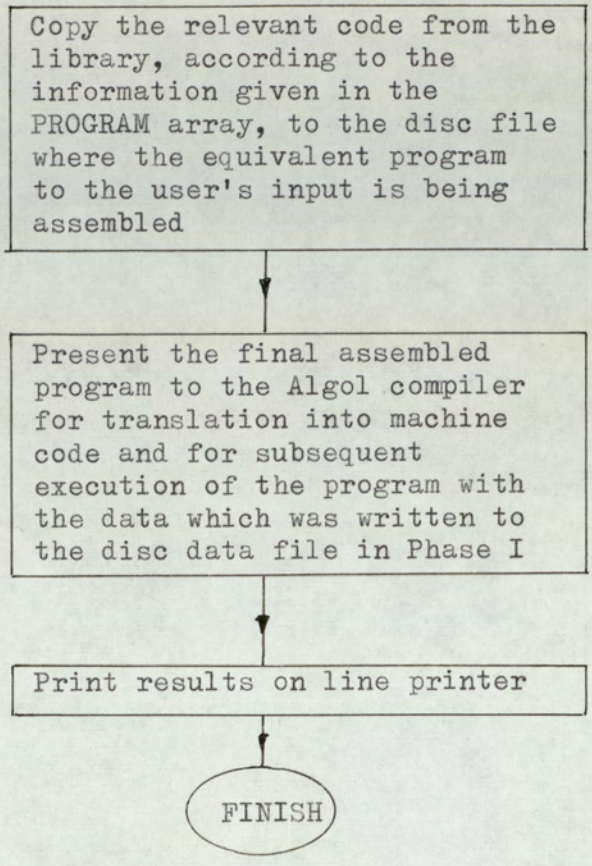
Code to process a typical procedure




Label 6:
This procedure has occurred before - either directly given by the user or implied by the use of another procedure
The positions of code in the library to declare this procedure and any other procedures it may use will have been put into the PROGRAM array at its first occurrence and need not be repeated
Putinto the PROGRAM array only the positions of code in the library which calls this procedure into use
Replace the number representing this procedure back into the stack in the correct position
Move to process next procedure in stack.
Go to 5


Label 7: This procedure has not occurred before.
Check whether any procedures that this procedure may use have occurred previously - either directly given by the user or implied by the use of another procedure
NO
Put into PROGRAM array the positions of the code in the library which declares these procedures. Put into the bottom of the stack the numbers representing these procedures
YES
Put into PROGRAM array the positions of code in the library which will declare the procedure being processed
Put into PROGRAM array the positions in the library of any code (after the Base 3 card) which should be present before the procedure is called into use
Put into PROGRAM array the positions of code in the library which will call this procedure into use
Replace the number representing this procedure back into the stack in the correct position
Move to process next procedure in stack
Go to 5

PHASE 3


Copy the relevant code from the library, according to the information given in the PROGRAM array, to the disc file where the equivalent program to the user's input is being assembled
Present the final assembled program to the Algol compiler for translation into machine code and for subsequent execution of the program with the data which was written to the disc data file in Phase I
Print results on line printer
FINISH

4.7. Example

The following example is taken directly from Bugliarello's Hydro manuals and illustrates the concept of stringing procedure calls and the retention of data and computed results. In addition it illustrates input of data to arrays of multiple dimension as well as the use of alphanumeric data.

This program combines six procedures to compute a set of depth-area-duration curves for a large storm over a watershed. The data used are taken from the drawings of Fig. 4.1 and the program is given in Fig. 4.2. The process of solution is the following :

a. From inputted hourly masscurves of rainfall at each station in the area covered by the storm isohyetal map, compute six-hour masscurves of rainfall.

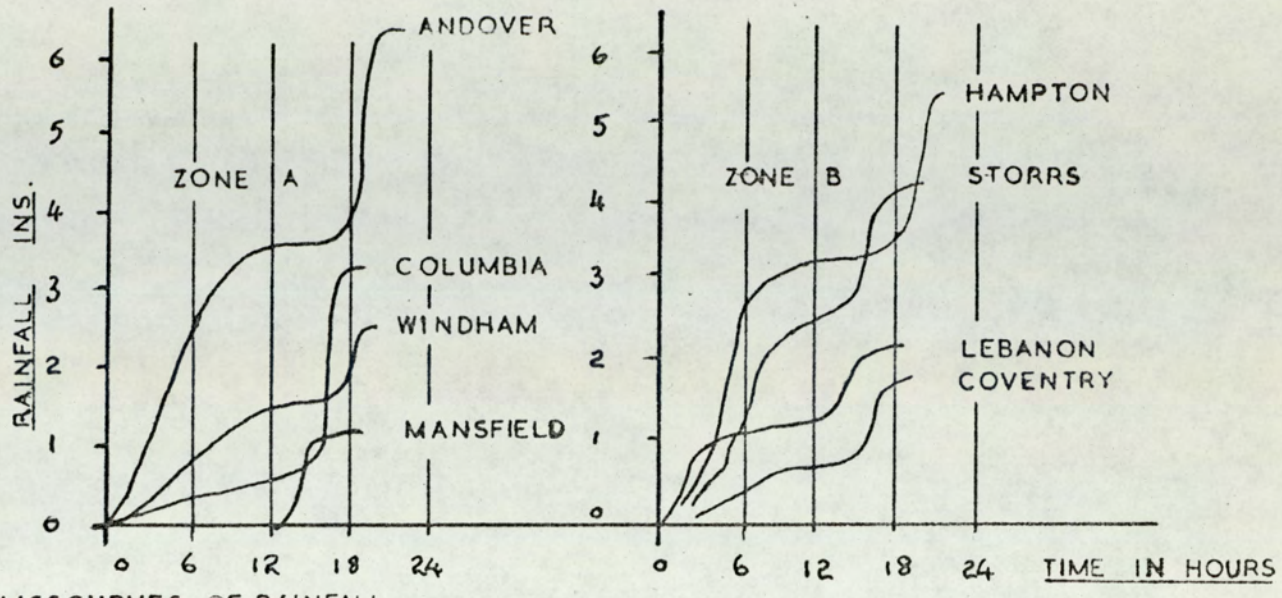
b. Find the maximum rainfall at each station for 6, I2, I8, and 24 hour periods throughout the storm.

c. Compute a curve of rainfall depth versus surface area from data planimetered from the isohyetal map. This curve relates average depth of rain to the areas surrounded by each isohyetal line.

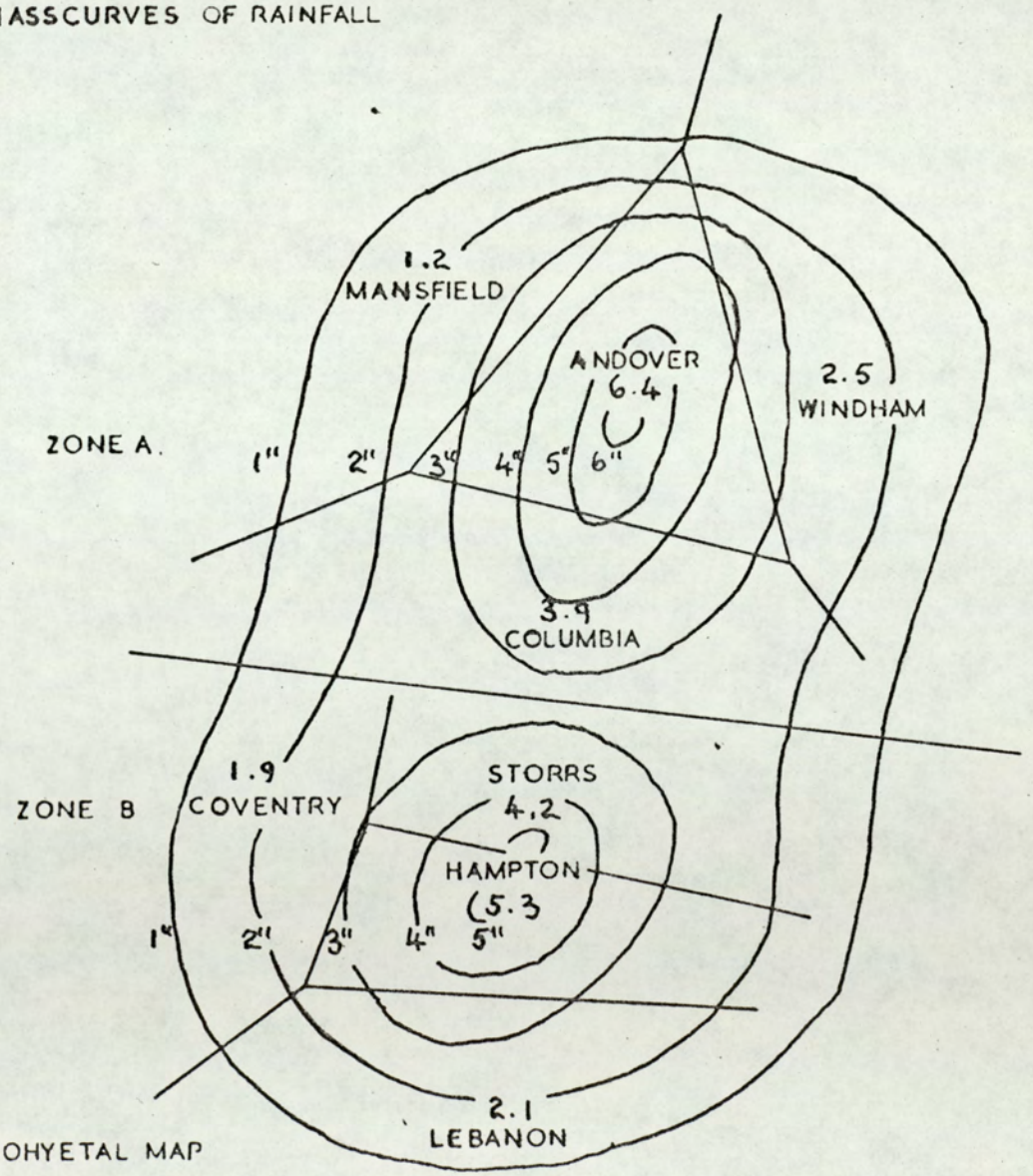
d. Compute depth-duration curves for the isohyets within individual zones of the isohyetal map using the six-hour masscurves at individual stations and planimetered areas about those stations within given isohyets.

e. Compute depth-duration curves for isohyets crossing combinations of zones of the isohyetal map.

f. Rearrange the computed depth-duration curves according to the area associated with each to get a complete set of depth-area-duration curves.



A) MASSCURVES OF RAINFALL



B) ISOHYETAL MAP

FIG. 4.1

```
SIX.HOUR.MASSCURVES:
TOTAL11=8;  TOTAL10=20;
ZONE1=*A*,*A*,*A*,*A*,*B*,*B*,*B*,*B*;
STATION1=*ANDOVER*,*COLUMBIA*,*WINDHAM*,*MANSFIEL*,
         *HAMPTON*,*STORRS*,*LEBANON*,*COVENTRY*;
MASSINPUT=0.5,0.8,1.1,1.5,1.9,2.3,2.7,3.0,3.3,3.4,
          3.4,3.4,3.4,3.4,3.4,3.4,3.4,3.4,3.5,6.4,
          0.2,0.2,0.2,0.3,0.4,0.4,0.5,0.6,0.7,0.8,
          0.8,0.8,0.8,0.8,0.8,0.9,1.5,3.0,3.9,3.9,
          0.2,0.3,0.5,0.7,0.8,1.0,1.2,1.3,1.4,1.5,
          1.6,1.6,1.6,1.6,1.6,1.6,1.7,2.5,2.5,2.5,
          0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
          0.0,0.0,0.0,0.0,0.2,1.1,1.2,1.2,1.2,1.2,
          0.2,0.5,0.8,1.4,2.3,2.8,3.1,3.2,3.2,5.2,
          3.2,3.2,3.2,3.2,3.2,3.2,3.2,3.2,5.2,5.3,
          0.2,0.3,0.4,0.8,1.0,1.4,2.0,2.6,2.8,2.8,
          2.8,2.8,2.8,2.8,2.8,2.8,2.8,3.8,4.2,4.2,
          0.3,0.6,0.7,0.8,1.0,1.1,1.2,1.3,1.3,1.3,
          1.3,1.3,1.3,1.3,1.3,1.8,2.1,2.1,2.1,2.1,
          0.0,0.1,0.2,0.4,0.5,0.7,0.8,0.9,0.9,0.9,
          0.9,0.9,0.9,0.9,1.0,1.9,1.9,1.9,1.9,1.9;
MAXIMUM.STATION.PRECIPITATION:
DEPTH.AREA.CURVE:
  TOTAL13=15;  CONVERTER=1.543;
  CENTER2=*A*,*A1*,*A1*,*A1*,*A1*,*A1*,*A1*,*A1*,*B*,*B1*,
          *B1*,*B1*,*B1*,*B1*,*B1*;
  ISOHYET2=0,6,4,6,5,4,3,2,1,0,5,3,5,4,3,2,1;
  PLANIMETER2=0,0,16,123,531,1580,3652,6704,0,0,36,
          319,1171,3122,6749;
DEPTH.DURATION.SINGLE.ZONES:
  TOTAL14=32;
  ZONE3=*A*,*A*,*A*,*A*,*A*,*A*,*A*,*A*,*A*,*A*,*A*,*A*,
        *A*,*A*,*A*,*A*,*A*,*B*,*B*,*B*,*B*,*B*,*B*,*B*,
        *B*,*B*,*B*,*B*,*B*,*B*,*B*,*B*;
  STATION3=*ANDOVER*,*ANDOVER*,*ANDOVER*,*ANDOVER*,*COLUMBIA*,
           *ANDOVER*,*COLUMBIA*,*WINDHAM*,*MANSFIEL*,
           *ANDOVER*,*COLUMBIA*,*WINDHAM*,*MANSFIEL*,
           *ANDOVER*,*COLUMBIA*,*WINDHAM*,*MANSFIEL*,
           *HAMPTON*,*HAMPTON*,*HAMPTON*,*STORRS*,
           *HAMPTON*,*STORRS*,*LEBANON*,
           *HAMPTON*,*STORRS*,*LEBANON*,*COVENTRY*,
           *HAMPTON*,*STORRS*,*LEBANON*,*COVENTRY*;
  ISOHYET3=0,6,5,4,0,3,0,0,0,2,0,0,0,1,0,0,0,
           0,5,4,0,3,0,0,2,0,0,0,1,0,0,0;
  PLANIMETER3=0,16,123,411,120,984,376,182,38,
              1522,929,751,540,0,0,0,0,0,36,207,112,
              698,392,81,1244,872,667,342,0,0,0,0;
DEPTH.DURATION.COMBINED.ZONES:
  TOTAL15=6;
  ZONE4=*A*,*B*,*A*,*B*,*A*,*B*,*A*,*B*;
  ISOHYET4=2,2,2,1,1,1;
  PLANIMETER4=1,1,1,0,0,0;
DEPTH.AREA.DURATION.CURVES:
PROGRAM.END:
```

Fig.4.2 HYDRO PROGRAM.

It can be seen from Fig. 4.2 how a program was developed to execute these six tasks. All rules of data transfer apply; all inputs given and outputs computed are always available for use by procedures called later in the program.

A large part of the data for the program, including some alphanumeric data and two-dimensional array data, are required for the first procedure, SIX.HOUR.MASSCURVES. The number of raingage stations in the watershed (8) is given by the variable TOTALII, and the number of hours of rainfall at these stations (20hours) is given by TOTALIO. ZONEI and STATIONI are alphanumeric variables; STATIONI is given the raingage station names, and ZONEI the rainfall zones in which each is located. It is easy to see how each piece of alphanumeric data is given, surrounded by asterisks. Finally the 8 masscurves of rainfall, each containing 20 values, are given continuously to the two-dimensional MASSINPUT array variable. There is no distinction within the data given for this variable between one masscurve and the next. The 8 curves are simply given back to back in an order corresponding to the station names given to STATIONI. Two lines are taken for each masscurve, but this of course is only done for order and convenience. Any number of lines could have been used.

The second procedure, MAXIMUM.STATION.PRECIPITATION, is given no data; it uses many of the same inputs given to SIX.HOUR.MASSCURVES.

The other procedures of the program are called in order, each followed by the data necessary for their execution. The data for DEPTH.AREA.CURVE, DEPTH.DURATION.SINGLE.ZONES, is alphanumeric. In all cases inputs and outputs from previous procedures are retained and used, eliminating the need for the user to re-submit data already inputted or computed

elsewhere in the program. The final procedure called, DEPTH.AREA. DURATION.CURVES, requires no data; its data are the results of previous procedures.

4.8. The ICL I900 Algol input/output procedures used by Hydro

In this section the procedures required by the translator program and the assembled program for the transference of information between the computer core store and the peripheral devices will be described. These procedures, or procedures which can perform similar functions are essential to the implementation of the Hydro system on any computer installation.

Disc handling procedures

These procedures allow numbers to be written to or fetched from a disc file so that they may be accessed in any order.

Using these procedures the disc may be thought of as a one-dimensional list. Each element in the list is numbered serially starting from element I, and each element contains one item of data.
It is possible for the disc procedures to fetch items in the list back into core store, starting at any specified element and ending at any element. Similarly, it is possible to write information away starting at any position in the list. This facility is used for storing the line library, so that the lines necessary to assemble a program may be accessed in any order. Each character is identified to the machine by using an integer code cypher instead of the actual character.
In Hydro the information is stored one character to an element, and there are 8I characters per line.
The elements in the disc store are transferred from and to arrays in the machine core. The Hydro system uses only a one dimensional array for this purpose, so that part of the list on the disc is copied directly to the array or list in the core.

The relevant procedures are :

a) WORKSTORE (N,S,L) which sets up a list of length L on a file on medium S, which is a disc for Hydro, and the file is referenced by a number N, assigned by the programmer.

b) PUTPART (N,K,A,X,Y)

This procedure transfers the part of the array A, in core, starting from array element X and ending with array element Y to the disc file, N, and writes it starting at element K on the file.

c) GETPART (N,K,A,X,Y)

This procedure reads information from the disc file, N, starting at element K on the file, and stores it in the array A, starting at array element X and finishing at array element Y.

Character handling procedures

These procedures are used to read the user's Hydro program one symbol at a time and to assign a unique integer number to different symbols, so that the machine can manipulate numbers instead of character information.

a) INBASIC (I); where I is a channel number assigned to a peripheral unit.

This procedure is usedin the form

X:= INBASIC (3);

INBASIC reads the next symbol from the input peripheral specified as its parameter and assigns to X the integer code value which represents the symbol in the machine.

b) OUTBASIC (I,X); where I is a channel number and X is a variable containing an integer code representing a basic symbol.

OUTBASIC prints out on channel I the symbol represented by the integer code in X.

4.9. The method of dealing with arrays in theuser's data

It was shown earlier in this chapter that an array which occured in the user's data was declared only once in the assembled program, at its first occurrence, and at later occurrences of that array in the user's data only reading procedures were incorporated into the program.

It can be seen that if an array increases in size, by using the same procedure in which it is employed, or another procedure which uses it, with different numerical data, at a later stage in theprogram, then the storage reserved for the array at its first occurrence will not be sufficient for later occurrences.

This error cannot be overcome by declaring the array every time it is used, because it would lead to wastage of the storage space and would mean that a previously read in array could not be updated even in one element, since the original array would not be made available by the Algol compiler after that point in the program at which it was re-declared.

There are two methodsof overcoming this difficulty, one of which requires the user to provide additional information and the other which could be performed automatically by the Hydro system.
The first method is employed in the author's revision of Hydro and is described in the next chapter.

CHAPTER 5

Revision of Hydro

5.I. Introduction

The original version of Hydro as written by Bugliarello was a first step in the investigation of high level computer languages for application in the field of water resources and hydraulic engineering, but was by no means the complete answer to the problem of bringing engineers into closer and more confident contact with computers. Although the language was simple to use, there was very little flexibility allowed in the sequencing of proceedures. The user had no way of altering the course of his program depending upon intermediate results and it was impossible to repeat sequences of procedures with different data without having to write down the same procedure names again. Furthermore, as was stated in the previous chapter, the Hydro translator program as written originally would not haveoperated in the manner intended for all cases of user input. An additional disadvantage was the inability of the system to store information in the form of results from one user program for use in later programs, or, in other words, no data bank could be built up.

When the original Hydro system was constructed, the aim was to find some simple language for use by engineers, which could be understood by a translator program and then rewritten into one of the universally accepted and standardised computer languages. The facilities which should be provided in such a simple language thus had to be examined before a translator program could be written, and because it is difficult to foresee every eventuality some helpful facilities were overlooked or deemed unimportant at the time. It has become obvious in recent years that data management is as difficult, if not more difficult, than actually writing a program to carry out the specialist procedures and

the author felt that the original Hydro system did not go far enough in simplifying data input to the machine. Therefore, the Hydro system has been rewritten to facilitate the control of the program during execution and to provide more useful data handling methods. The user may now employ any of the basic Algol - 60 statements within his program, the most useful being the conditional 'IF' statement, the 'FOR' statement and the 'GO TO' statement.

A further aid to rapid programming is the provision for tabulation of data. Identified data may be written in a table in the normal format for the original Hydro translator. Any number of variable names and associated data may be present in one table, which is assigned a unique integer number by the user. However, if the user does not wish to tabulate his data, or if there is no need for it, then he may simply write the data anywhere before or immediately after the procedure name in the same way as for the original system. If any data is not presented in a tabular form then the new Hydro translator automatically assigns a table number to it for its own reference.

As a consequence of including 'GO TO' statements in the language and to provide more flexibility than the original Hydro language it has become necessary for the user to give the maximum values which some of the Hydro input single variables may take during the course of the program at the head of the input. This is necessary because these values are used to reserve enough space in the computer core for all the data given by the user. These values will always be known since they are only counters of the number of data itemsto be input for the Hydro array variables.

Some typical examples of user input for the New Hydro are given at the end of this chapter.

Since the basic design and philosphy of the original Hydro, and the underlying method of assembling an equivalent Algol program from the user's input remains similar in the revised version, this chapter does not reiterate the information to be found in the last chapter and in the Hydro manuals written by Bugliarello.

5.2. Special Symbols Employed by the Revised Hydro

A procedure name given by a user of Hydro is given on a separate line and is followed by the symbol ← .

The use of this symbol facilitates the search for procedure names in the user's program. The translator simply scans a line of input for a left arrow and if it is found the code for processing a procedure is effected. If no data assignments are to be given after the procedure name the left arrow may be immediately followed by three asterisks (***) on the same line or on the next line.

If data assignments are given after the procedure name then the three asterisks must terminate the block of assignments for that procedure. The combination of left arrow and three asterisks have the same grammatical effect as the Algol (and) symbols after a procedure name and only indicate to the translator that all the information between them is relative to the procedure. The rulesof grammar for the Algol brackets apply to these symbols.

The 'TAB' symbol indicates that the following information up to the two asterisks (**) line is a table of data.

The 'TAB' symbol is followed immediately by a number assigned by the user for this table, and this number is used in any reference to the table.

A special case of the 'TAB' symbol occurs when a multiple table is used. In this case the user gives two numbers, separated by commas, after 'TAB'. A multiple table is allowed purely for convenience to avoid the user's having to provide a large number of tables, written separately, if one data array variable is to be assigned different sets of values on different passes through a procedure.

The user simply has to give the starting number of the tables and the

finishing table number. He may then write down the name of the variable to which the data is to be assigned at the head of the table, followed by := . From the next line onwards he may write the separate sets of data, each terminated by a semi-colon. After the last data set has been given, the multiple table is concluded by the two asterisks in the normal way.

5.3. The Form of Input

The user's input to the revised Hydro begins with a call on the procedure MAXVALS which is followed by a list of the single valued Hydro variables which are employed by the user's program, together with the maximum values which each takes. As mentioned previously, these values will always be known by the user since they are only counters of the number of data items given for the array variables in the user's input. The list of variables is terminated with three asterisks.

Example 5.1.

```
MAXVALS ←
GRIDROWS := I2; NOGAUGES := 8;
GRIDCOLUMNS := I4;
*** ;
```

After the MAXVALS procedure the user has the option of calling the DECLARATIONS procedure. At this stage, any single or array variables which the user wishes to employ, and which are not Hydro variables, may be declared in the normal Algol manner. As with MAXVALS, the procedure is terminated with three asterisks.

Example 5.2.

```
DECLARATIONS ←
'INTEGER' I,J;
'ARRAY' DATA [1:100,1:3];
'REAL' X;
*** ;
```

Following the MAXVALS and DECLARATIONS procedures the user may begin to use the normal Hydro procedures with the relevant data. Pure Algol code may also be employed after this point.

Procedural instructions are all given in the same way. First, the procedure name is given on a separate line, followed by a left arrow. On

the next and subsequent lines any necessary data is assigned to the procedure variables, or the tables where the data may be found are given alongside the variable names.

If the results from the procedure are required to be filed in the data bank for later use then the table number to which the procedure must write its results should be given in brackets between the procedure name and the left arrow. When all data assignments for a procedure have been made, the three asterisks line is given to terminate the group.

The only other code which may occur on the same line as a procedure name is a label.

Example 5.3.

```
LAB: THIESSEN. RAINFALL. AVERAGE (RESULTS TO 'TAB' IO) ←
GRIDROWS := 3; GRIDCOLUMNS := I2;
GRID:= 'TAB' 3;
*** ;
```

Since the left arrow and three asterisks combination is grammatically the same as the Algol round brackets then the three asterisks must be followed by an Algol terminator, (i.e. ; 'END' or 'ELSE').

A label may be used against any Hydro procedure name or against a data assignment which is not included between a left arrow and three asterisks.

Tables of data may be given at any point in the program and take the same form as normal assignments. The table number is given on a separate line and on subsequent lines the data assignments are given.

A table is terminated by a two asterisks line.

Example 5.4. Simple table

```
*'TAB' IO
GRID := 0,0,0,I,I,I,I,I,0,0,0,0,
        0,0,I,I,I,I,I,I,I,I,I,0,
        0,I,I,I,I,I,I,I,0,0,0,0;
```

```
GAUGES := I,4,
          I,7,
          I,8,
          2,5,
          2,6,
          3,3,
          3,4,
          3,7;

RAIN:= 0.II, 0.40, 0.30, 0.25, 0.4I, 0.35, 0.20, 0.36;

** ;
```

Example 5.5. Multiple Table

Each data set is terminated by a semi-colon.

```
* 'TAB' I,4

RAIN:= 0.3, 0.4, 0.6, 0.5;
       0.2, 0.3, 0.5, 0.4;
       0.6, 0.8, 0.8, 0.5;
       0.2, 0.I, 0.3, 0.I5;

** ;
```

Every user's program must be terminated by the procedure call

PROGRAM.END ← followed by a four asterisks line.

```
****
```

5.4. The translator program

Since tabular data, given in any position in the user's program, is allowed in the revised version of Hydro, it is not possible to process the user's input in the same way as in the original version. In the last chapter, it was shown that the first part, or phase one, of the original Hydro translator simply compiled a list describing the order of occurrence of the Hydro data variables and procedure names in the user's input, while copying array to a disc file the linesof input containing data assignments. Having done this it was only necessary to use the list in some way to assemble an equivalent Algol program. This simplicity was achieved because the data was read by the assembled program in the same order as it was supplied by the user and because no conditional clauses were allowed in the Hydro language. However, since the data for the revised version of Hydro may be given in tabular form, and Algol control statements are acceptable, then some method must be foundto indicate to the assembled program which data table to read from.

In order to achieve the desired result, it was found that in some circumstances the essential operations of all three phases of the original Hydro translator must be combined. A list is still compiled but this only records the identification numbers of the Hydro procedures occurring in the user's input.

All code for reading data from tables is assembled on a special disc scratch file immediately the relevant data variable assignment is found in the user's program. Similarly, the code for calling a procedure is written to this file when the procedure name occurs in the input. In this case, however, the procedure call is held back until the code dealing with all the data assignments following the procedure name has been assembled. Any pure Algol statements are copied directly, with no

alteration, to the file. If a table of data is given by the user, the translator copies this to a separate data disc file but inserts some necessary information at the head of the table for later use by the assembled program. The number of the table, supplied by the user, is recorded in a list together with its position in the file, and this list is made available to the assembled program. If any data is given which is not in a table, then the translator assigns a table number to it, copies it to the tables file in the normal way, and writes the necessary reading code to the file where the equivalent Algol program is being assembled.

When all of the user's input has been read, the translator processes the list of procedures which has been set up. For each procedure occurring in the list the translator assembles code to declare the body of the procedure and any other procedures it might use. All arrays used by the procedures are also declared at this stage. These declarations are assembled on a final disc scratch file. After all procedures have been processed, the code assembled previously on the program file for reading data and calling procedures, is copied after the declarations. The assembled program is then complete and in the correct order, and may be presented to the Algol compiler for translation into machine code and execution. The data for this Algol program is automatically taken from the tables file.

Examples of user input for the revised version of Hydro are given at the end of this chapter.

5.5. Examplesof User Input

Example 5.6.

The following example demonstrates a possible user's program for the repetitive use of the revised Thiesson method for calculating areal rainfall averages.

The GRID array represents a reference grid placed over the area under consideration and which completely covers the area. If a zero is given for an element in the GRID array this means that the grid point falls outside the area and if a one is given the point is inside the area.

The GAUGES array describes the positions of rain gauges in the area by supplying the nearest grid coordinates to the gauges.
GRIDROWS and GRIDCOLUMNS simply give the number of rows and columns in the grid so that sufficient space may be reserved in the computer for the information supplied by the user, and GRIDAREA is the ground area covered by one grid square.

The THIESSEN.RAINFALL.AVERAGE procedure is called ten times with different values for rainfall at the gauge points.
The rainfall values given in the RAIN array are written in the same order as the rain gauge coordinates are supplied to the GAUGES array.
After the Thiessen procedure has been repeated ten times the RAIN array is set to the values given in table 4 and the ARITHMETIC.RAINFALL. AVERAGE procedure is called to take the direct averageof these rainfall figures.

```
MAXVALS ←

GRIDROWS:=I2; NOGAUGE:=8;

GRIDCOLUMNS:=I4;

***;

DECLARATIONS ←

'INTEGER'I;

***;

GRIDAREA:=25.0;

GRID:=

      0,0,0,0,0,0,0,0,0,I,I,I,0,0,
      0,0,0,0,0,0,0,0,I,I,I,I,I,0,
      0,0,0,0,0,I,I,I,I,I,I,I,I,I,
      0,0,I,I,I,I,I,I,I,I,I,I,I,I,
      I,I,I,I,I,I,I,I,I,I,I,I,I,0,
      I,I,I,I,I,I,I,I,I,I,I,I,0,0,0,
      I,I,I,I,I,I,I,I,I,I,I,0,0,0,0,
      I,I,I,I,I,I,I,I,I,0,0,0,0,0,0,
      0,I,I,I,I,I,I,I,I,I,0,0,0,0,0,
      0,0,I,I,I,I,I,I,I,I,I,0,0,0,0,
      0,0,0,0,0,I,I,I,I,I,0,0,0,0,0,
      0,0,0,0,0,0,I,I,I,0,0,0,0,0;

GAUGES:=2,II,
        4,4,
        4,I0,
        5,7,
        7,I,
        7,9,
        9,4,
       II,9;

'FOR' I:=I,I0'DO'

THIESSEN.RAINFALL.AVERAGE ←

RAIN:='TAB' I;

***;
```

```
 'TAB' I
RAIN:=I.2,I.35,0.46,0.6,I.3,I.5I,0.84,0.67;
**;
*'TAB' 2
RAIN:=I.5,I.6,0.3,0.7,I.5,I.4,0.7,0.7;
**;
*'TAB' 3
 . . . . . .
 . . . . . .
 .
 .
*'TAB' IO
RAIN:=I.4,I.5,0.35,0.65,I.8,2.0,0.6,0.9;
**;
RAIN:='TAB' 4;
ARITHMETIC.RAINFALL.AVERAGE ←
***;
'IF' RAINFALLAVERAGE<I.O 'THEN' WRITETEXT ('('DRY YEAR')');
PROGRAM.END  ***;
****
```

Example 5.7.

This example gives a possible user's input program for a sequence of routines which could generate I00 years of random inflows to a two reservoir system given the histograms of flow probabilities determined from a historic record, then uses this data in a dynamic programming calculation to obtain a near optimum operational policy for the system which is then employed in a simulation of the system to find the costs incurred by using this policy for I00 years with the inflows previously generated.

The reservoir inflows are assumed to be independent in one time increment and it is assumed that serial correlation may be taken into account by using an index which records whether the previous inflow was higher or lower than average for that month, and a histogram of inflow probabilities is given for each of these two cases for each monthof the year.

The Hydro variables used in the input are as described below:

NI is the number of discrete inflows used in the histograms.

NSTATA and NSTATB are the number of discrete levels used in the two reservoirs for the dynamic programming calculation.

RANDA and RANDB are merely starting points for the generation of random numbers.

The arrays FLOWSA and FLOWSB record the inflowhistograms for the two reservoirs.

Two rows of data are given for each month and for each reservoir. The first row gives NI sets of data where a set represents a discrete flow in the histogram followed directly by its probability. This row is the histogram for the case when the previous month's inflow was lower than average. The second row is the histogram for a higher than average

previous month's inflow.

Thus there are 24 rows of data for each reservoir giving 2 histograms for each month.

URA, URB, UDF, USA and USB are unit costs of releases from each reservoir, deficits to demand, and spillage costs in each reservoir respectively.

PWF is a present worth or discount factor.

The arrays STATESA and STATESB record the discrete levels to be used in the reservoirs.

The DEMAND array gives the demand to be used for each month.

The ORIGVALUES array records the terminating values of being in certain system states at the end of a dynamic programming calculation and corresponds to the array described in the chapters on dynamic programming.

```
DOC DATA
MAXVALS ←
NI:=5;
NSTATA:=4;
NSTATB:=4;
***;
GENERATE ←
RANDA:=0.23;
RANDB:=0.87;
FLOWSA:=I.47,0.I7,3.44,0.I4,6.I0,0.34,8.42,0.24,I2.06,0.II,
       3.I7,0.I7,5.I6,0.24,6.28,0.I4,7.I3,0.I7,8.43,0.28,
       0.4I,0.II,2.46,0.34,4.63,0.22, . . . . . ,
       0.74,0.25,2.97,0.28, . . . . . ,
        . . .  .  .  .  .  . . . .
        . . .  .  .  .  .  . . . .
        . . . . . ;
FLOWSB:=0.33,0.30,0.87,0.I7,I.03,0.26,I.3I,0.I7,I.99,0.I0,
       0.52,0.I3, . . . . . ,
        . . .  .  .  .  .  . . . ,
        . . .  .  .  .  .  . . . ,
        . . .  .  .  .  .  . . . ,
        . . . . . ;
***;
DYNAMIC.PROGRAM ←
URA:=0;URB:=0;UDF:=I00;
USA:=0;USB:=0;PWF:=0.985;
STATESA:=0,I0000,20000,30000;
STATESB:=0,I0000,20000,30000;
DEMAND:=0.22,0.42,0.85,I.62,3.33,3.85,4.I6,3.46,3.0I,I.76,
       0.36,0.I4;
```

```
ORIGVALUES:=0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
***;
SIMULATE ←
***;
PROGRAM.END ←
****
```

Example 5.8.

Example of program assembled by Hydro.

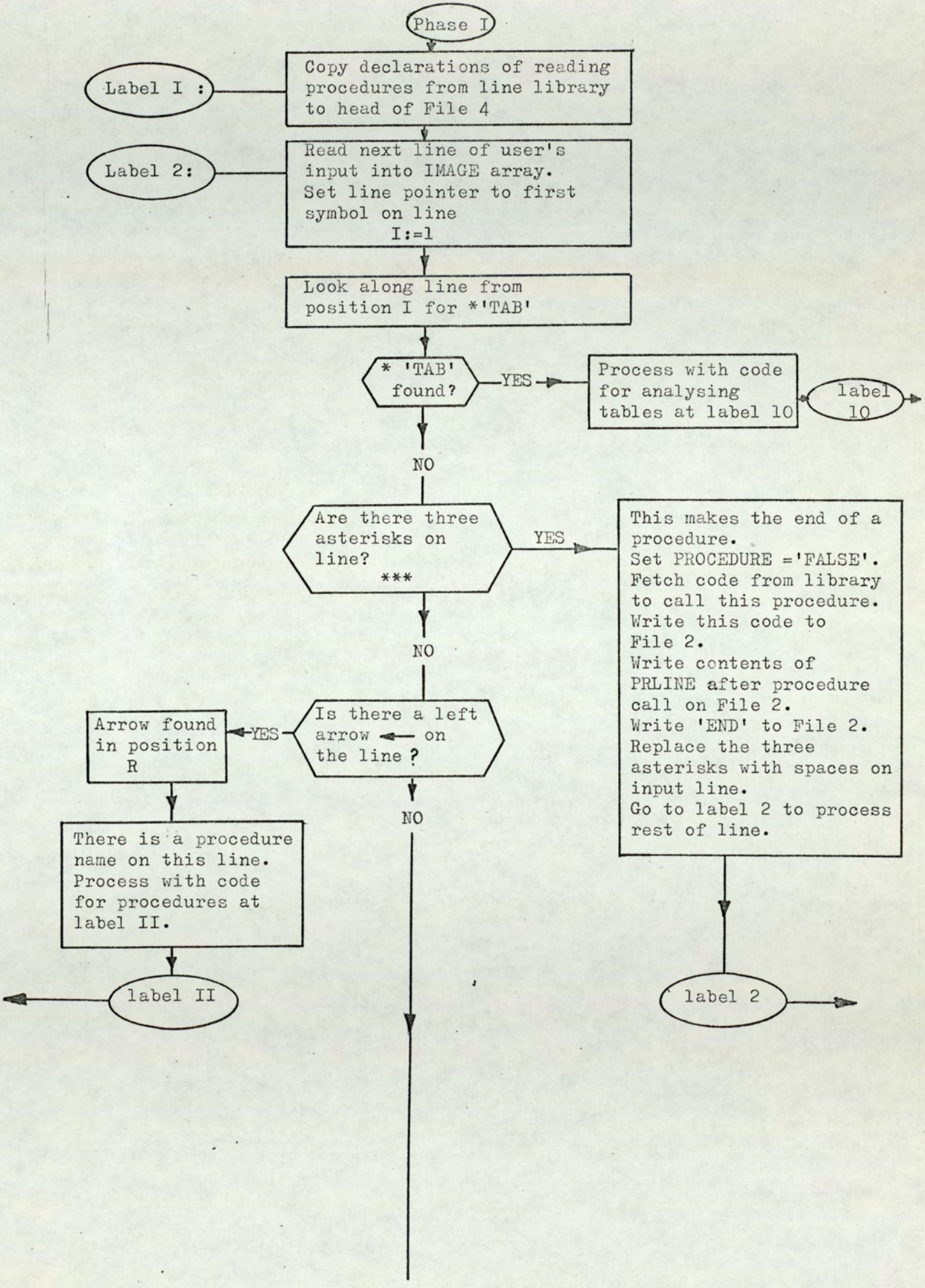
Program assembled for previous example of user's input:

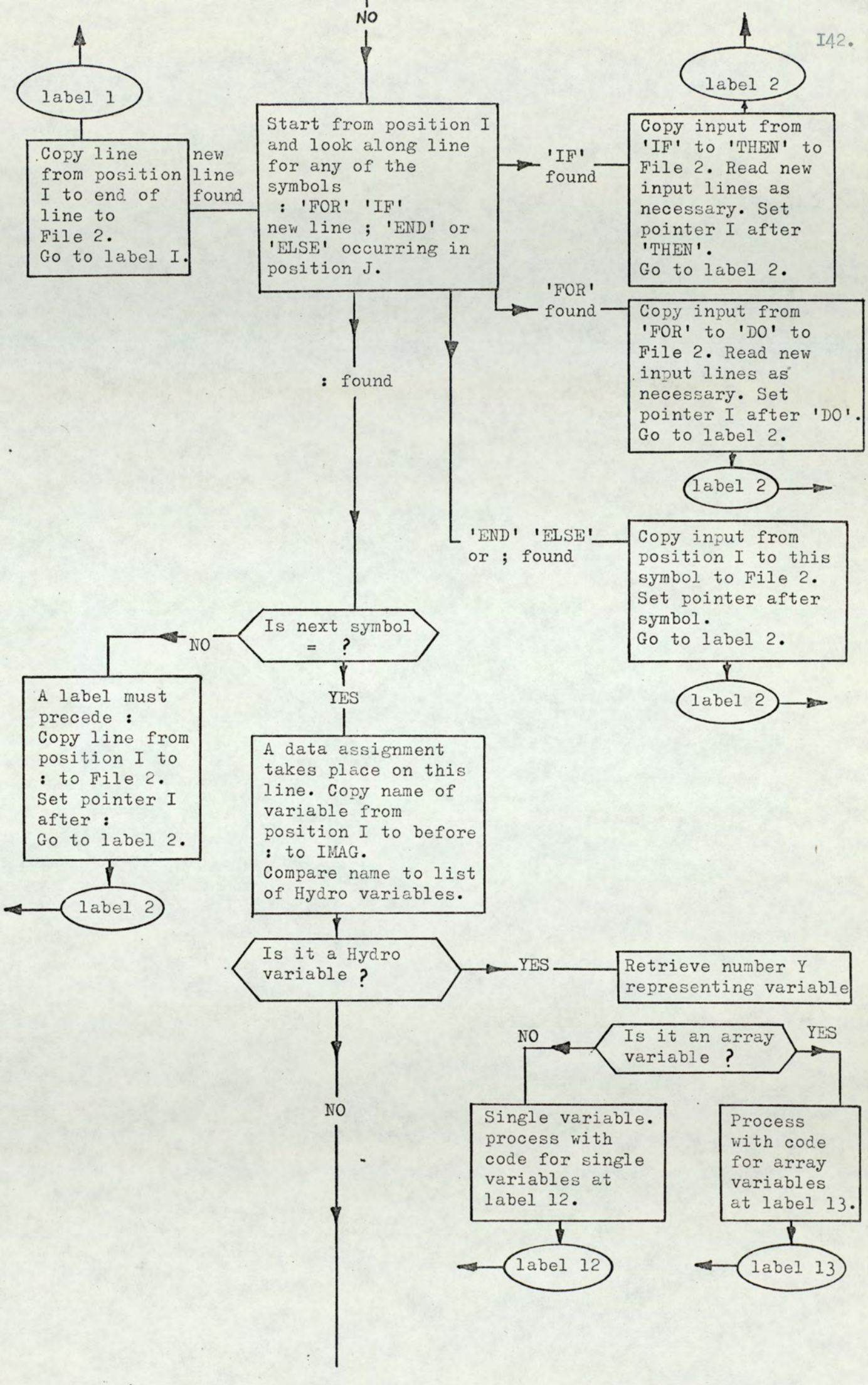
```
'BEGIN'
'COMMENT' GENERAL DECLARATIONS FOLLOW;
     General code to declare reading procedures etc
     (as shown in APPENDIX 4 before BAS I line)
'BEGIN'
'COMMENT' ASSEMBLED PROGRAM FOLLOWS ;

NI:=5;
NSTATA:=4;
NSTATB:=4;
TABLEPOINTER:=15000;
'BEGIN'
'PROCEDURE' GENERATE (RANDA, RANDB, FLOWSA, FLOWSB, AVA, AVB, TABLEPOINTER,
      TABNO);
'VALUE' TABNO;
'ARRAY' FLOWSA, FLOWSB, AVA, AVB;
'INTEGER' TABLEPOINTER, TABNO;
'REAL' RANDA, RANDB;
'ALGOL';
'PROCEDURE' DYNAMICPROGRAM (AVA, AVB, STATESA, STATESB, DEMAND,ORIGVALUES,
   VALUES, DECISIONS,URA,URB,UDF,USA,USB,PWF,NSTATA,NSTATB,NI,TABLEPOINTER,
   TABNO);
'VALUE' TABNO;
'ARRAY' AVA,AVB,STATESA,STATESB,DEMAND,ORIGVALUES,VALUES,DECISIONS;
'INTEGER' NSTATA,NSTATB,NI,TABLEPOINTER,TABNO;
'REAL' URA,URB,UDF,USA,USB,PWF;
'ALGOL';
'PROCEDURE' SIMULATE (AVA,AVB,STATESA,STATESB,DEMAND,DECISIONS,URA,URB,
      UDF,USA,USB,NSTATA,NSTATB,TABLEPOINTER,TABNO);
'VALUE'TABNO;
'ARRAY' AVA,AVB,STATESA,STATESB,DEMAND,DECISIONS;
'INTEGER' NSTATA,NSTATB,TABLEPOINTER,TABNO;
```

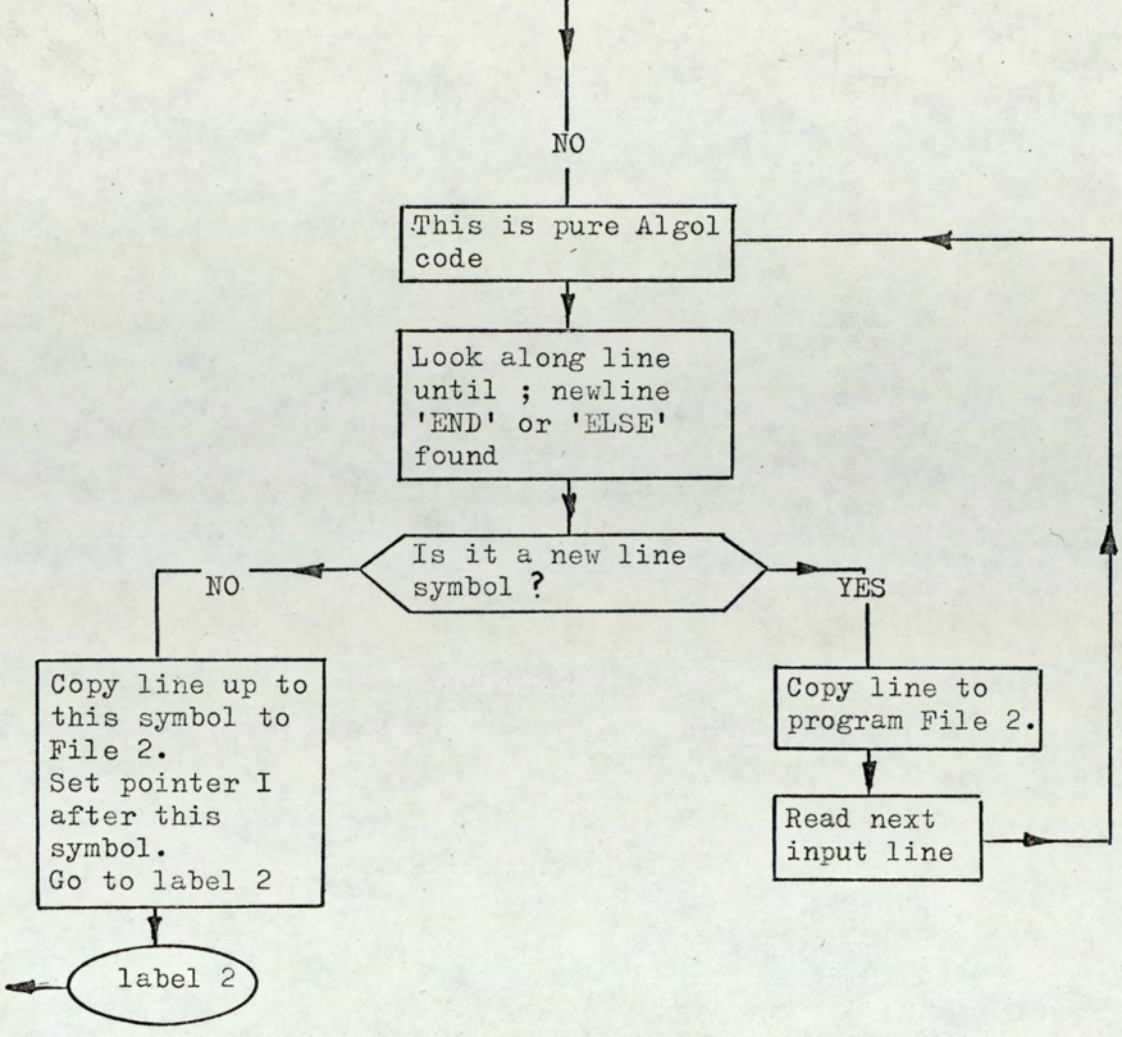
```
'REAL' URA,URB,UDF,USA,USB;
'ALGOL';
'ARRAY' FLOWSA [I:I2,I:4,I:NI] ;
'ARRAY' FLOWSB [I:I2,I:4,I:NI] ;
'ARRAY' AVA[I:I2] ;
'ARRAY' AVB[I:I2] ;
'ARRAY' FLOWDATA[I:3] ;
'ARRAY' STATESA [I:NSTATA];
'ARRAY' STATESB [I:NSTATB];
'ARRAY' DEMAND [I:I2];
'ARRAY' ORIGVALUES [I:(NSTATA*NSTATB)];
'ARRAY' VALUES [I:(NSTATA*NSTATB)] ;
'ARRAY' DECISIONS [I:(NSTATA*NSTATB*4,I:I2] ;
'BEGIN'
RANDA:=0.23;
RANDB:=0.87;
ARRAYREAD(FLOWSA, DAI,I,I00I3,I2,4,NI,0,
COLM,CHAR,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,BEG,END,
GETPARTI,BINARY,ALPHANUMERICALDATA,DATAINPUT,PROGEND,FINISHED,
DEFINESUBSCRIPTS,LOC,TOP,BOT,
I00I);
ARRAYREAD(FLOWSB,DAI,I,I00I4,I2,4,NI,0,
COLM,CHAR, . . . . . ,
GETPART , . . . . . ,
DEFINESUBSCRIPTS, . . . . . ,
I002);
GENERATE(RANDA,RANDB,FLOWSA,FLOWSB,AVA,AVB,TABLEPOINTER,0);
'END';
'BEGIN'
URA:=0;URB:=0;UDF:=I00;
USA:=0;USB:=0;PWF:=0.985;
ARRAYREAD(STATESA,DAI,I,I00I5,NSTATA,0,0,0,
COLM,CHAR, . . . . . ,
GETPART, . . . . . ,
DEFINE , . . . . . ,
I003);
```

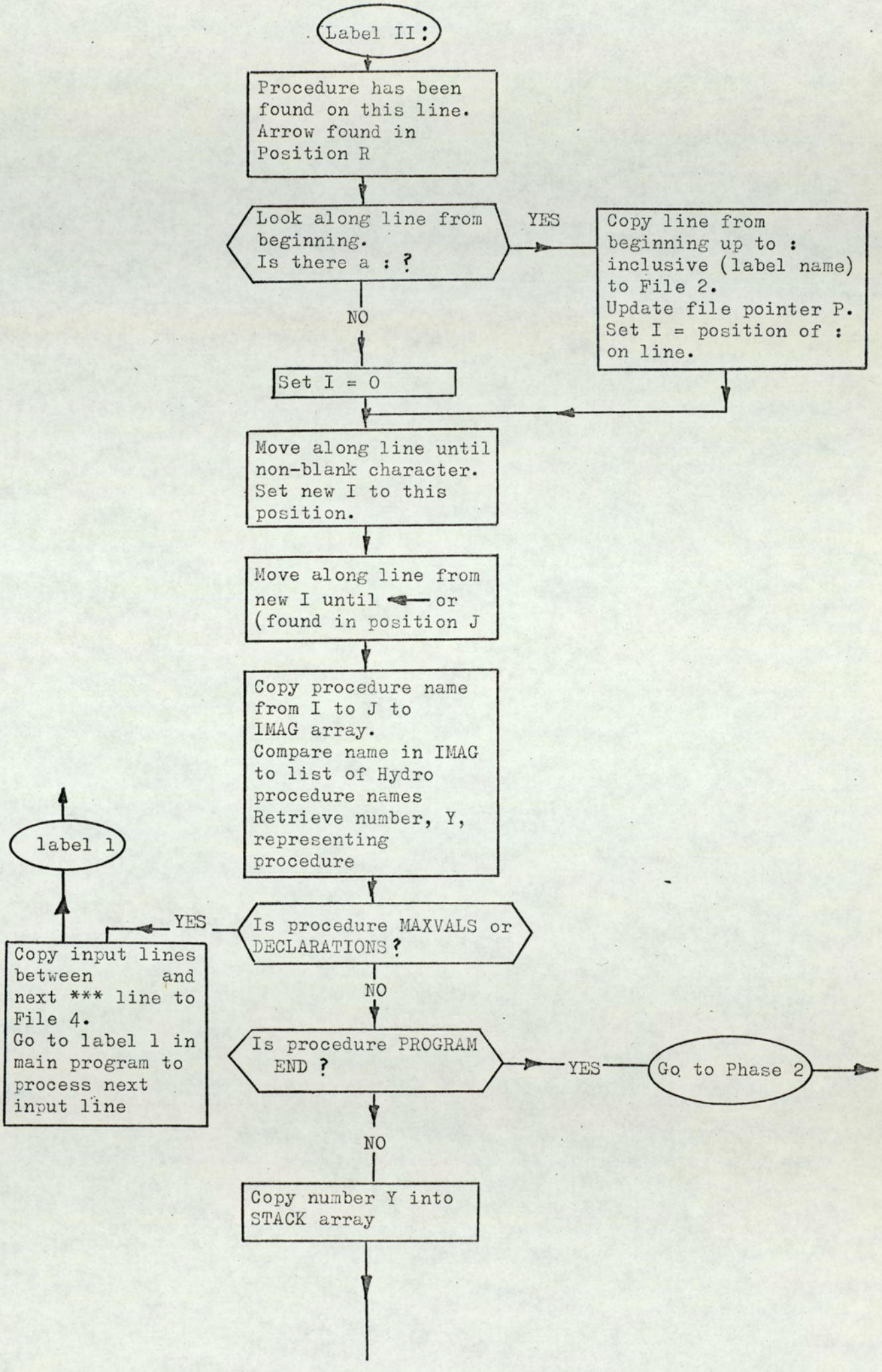
```
ARRAYREAD(STATESB,DAI;;I,I00I6,NSTATB,0,0,0,  .  .  .  ,
COLM,CHAR,  .  .  .  .  .  .  ,
GETPART,  .  .  .  .  .  .  .  .  ,
DEFINE ,  .  .  .  .  .  .  .  .  .  .  ,
I004);
ARRAYREAD(DEMAND,DAI,I,I00I7,I2,0,0,0,
COLM,CHAR,  .  .  .  .  .  .  .  .  .  ,
GETPART,  .  .  .  .  .  .  .  .  .  .  ,
DEFINE ,  .  .  .  .  .  .  .  .  .  .  ,
I005);
ARRAYREAD(ORIGVALUES,DAI,I,I00I8,(NSTATA*NSTATB),0,0,0,
COLM,CHAR,  .  .  .  .  .  .  .  .  .  ,
GETPART,  .  .  .  .  .  .  .  .  .  .  ,
DEFINE,  .  .  .  .  .  .  .  .  .  .  ,
I006);
DYNAMIC PROGRAM(AVA,AVB,STATESA,STATESB,DEMAND,ORIGVALUES,VALUES,
      DECISIONS,URA,URB,UDF,USA,USB,PWF,NSTATA,NSTATB,NI,TABLEPOINTER,0);
'END' ;
'BEGIN'
SIMULATE (AVA,AVB,STATESA,STATESB,DEMAND,DECISIONS,URA,URB,UDF,USA,USB,
       NSTATA,NSTATB,TABLEPOINTER,
       0);
'END';
'END';
'END';
PROGEND:'END';
```

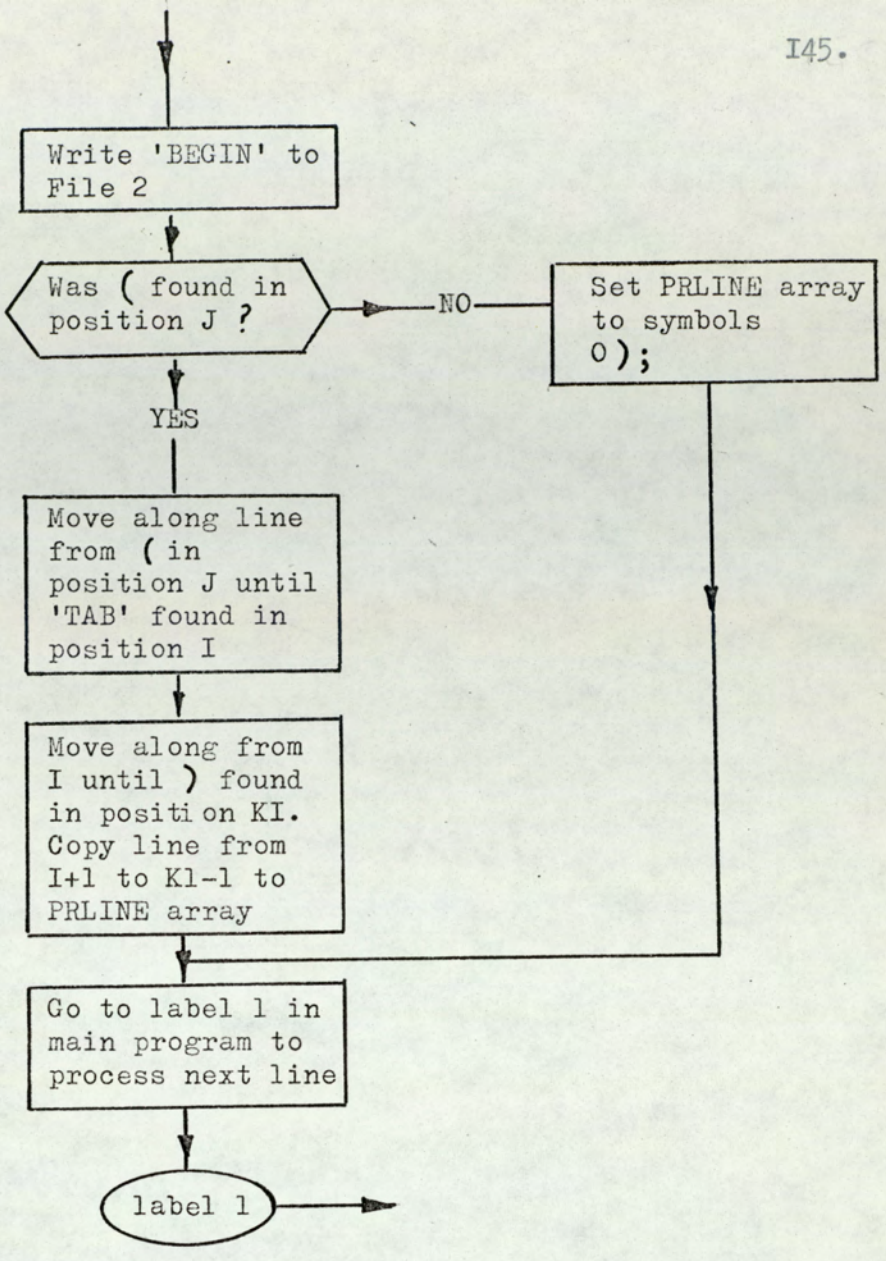
5.6. Flow Diagram for Translator Program

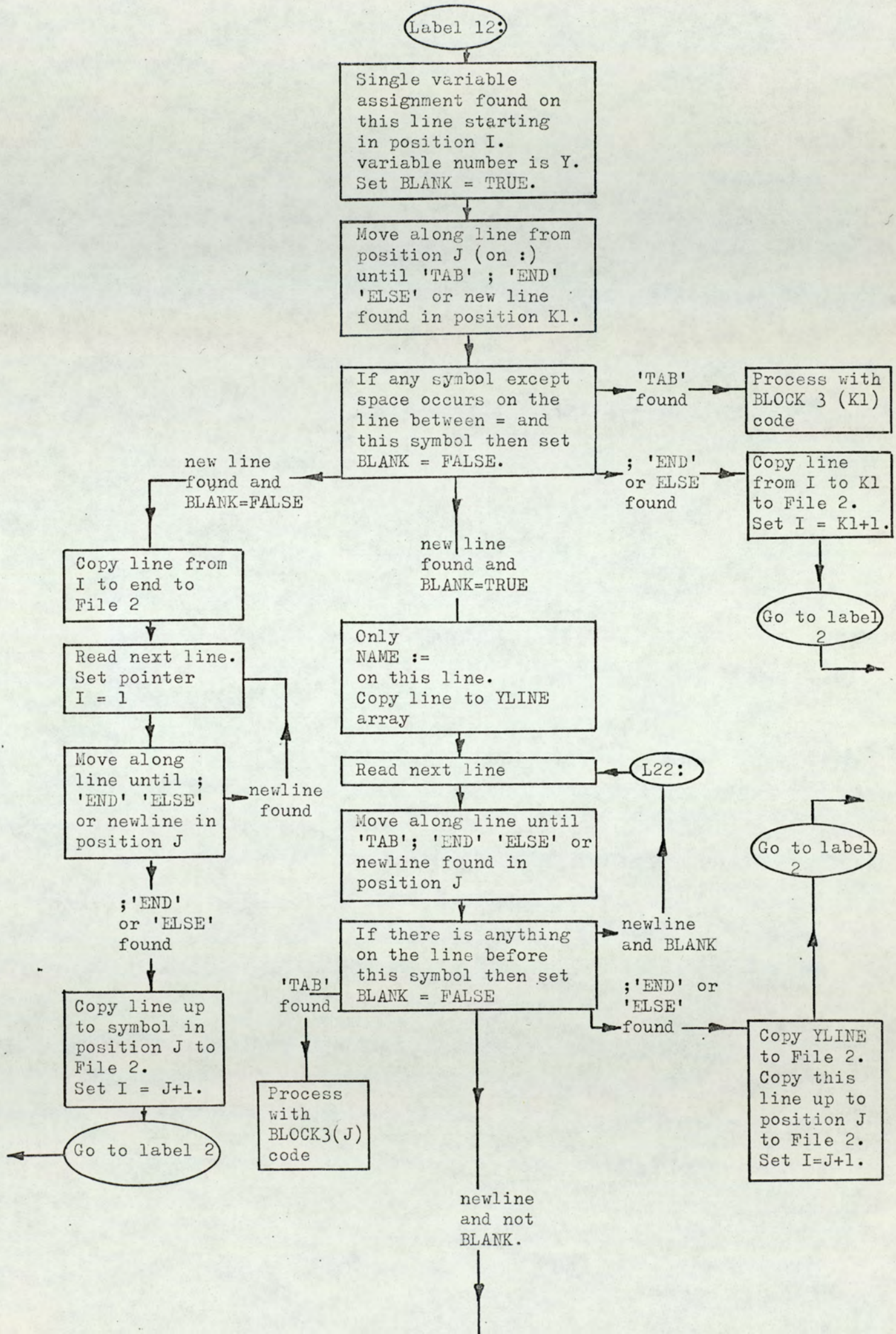


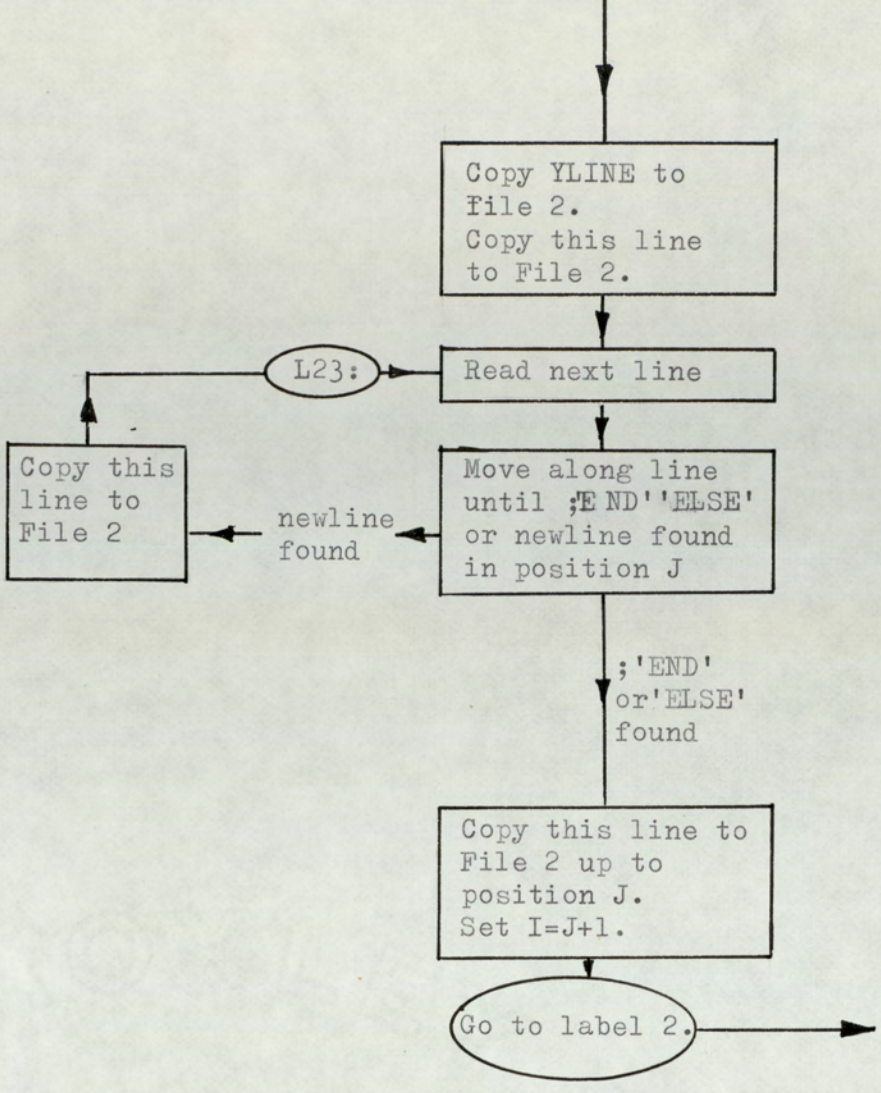

NO
label 1
label 2
Copy line from position I to end of line to File 2. Go to label I.
new line found
Start from position I and look along line for any of the symbols : 'FOR' 'IF' new line ; 'END' or 'ELSE' occurring in position J.
'IF' found
Copy input from 'IF' to 'THEN' to File 2. Read new input lines as necessary. Set pointer I after 'THEN'. Go to label 2.
'FOR' found
Copy input from 'FOR' to 'DO' to File 2. Read new input lines as necessary. Set pointer I after 'DO'. Go to label 2.
label 2
: found
'END' 'ELSE' or ; found
Copy input from position I to this symbol to File 2. Set pointer after symbol. Go to label 2.
label 2
Is next symbol = ?
NO
YES
A label must precede : Copy line from position I to : to File 2. Set pointer I after : Go to label 2.
label 2
A data assignment takes place on this line. Copy name of variable from position I to before : to IMAG. Compare name to list of Hydro variables.
Is it a Hydro variable ?
YES
Retrieve number Y representing variable
NO
Is it an array variable ?
NO
YES
Single variable. process with code for single variables at label 12.
Process with code for array variables at label 13.
label 12
label 13

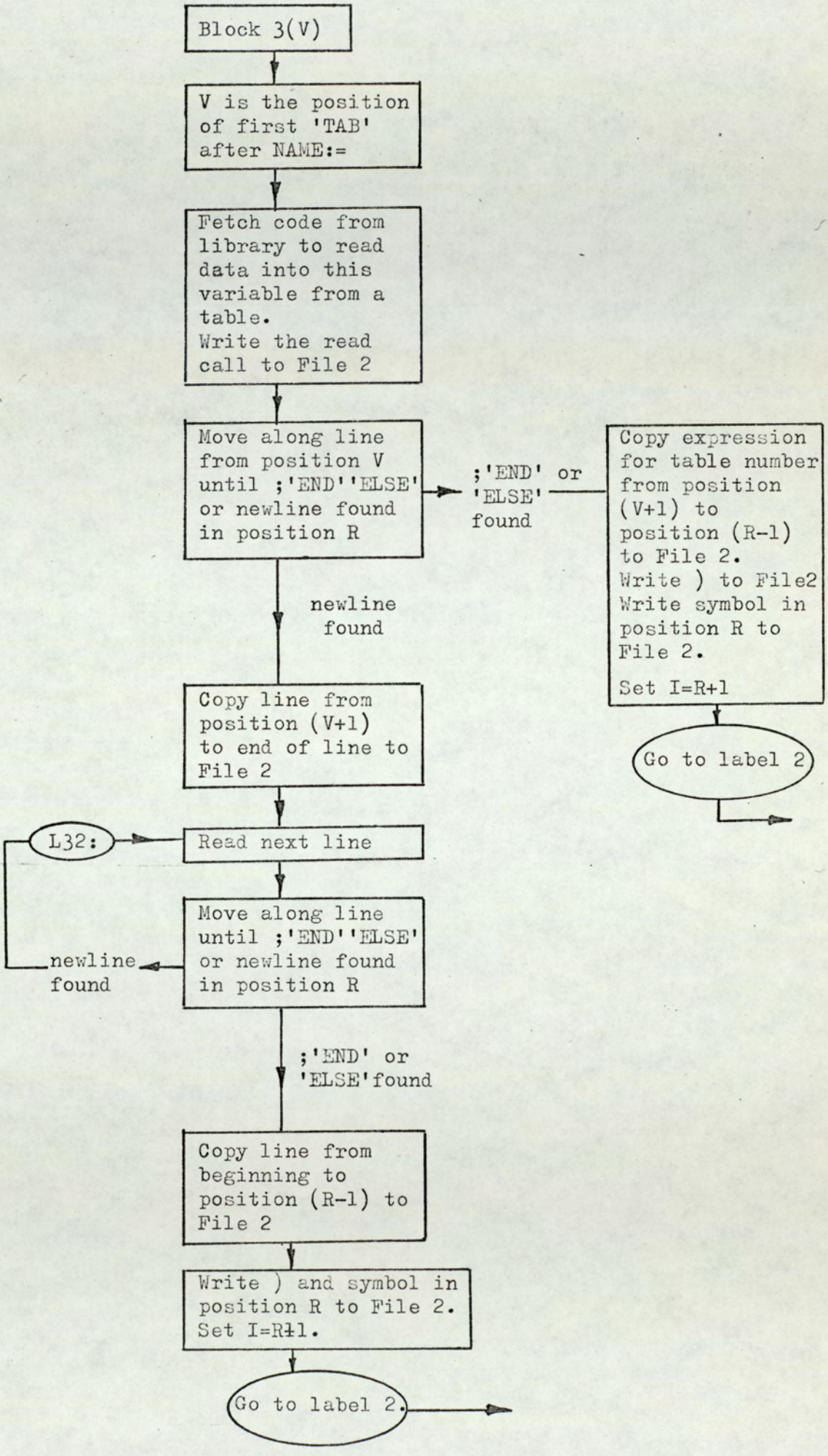

NO
This is pure Algol code
Look along line until ; newline 'END' or 'ELSE' found
Is it a new line symbol ?
NO
YES
Copy line up to this symbol to File 2.
Set pointer I after this symbol.
Go to label 2
label 2
Copy line to program File 2.
Read next input line

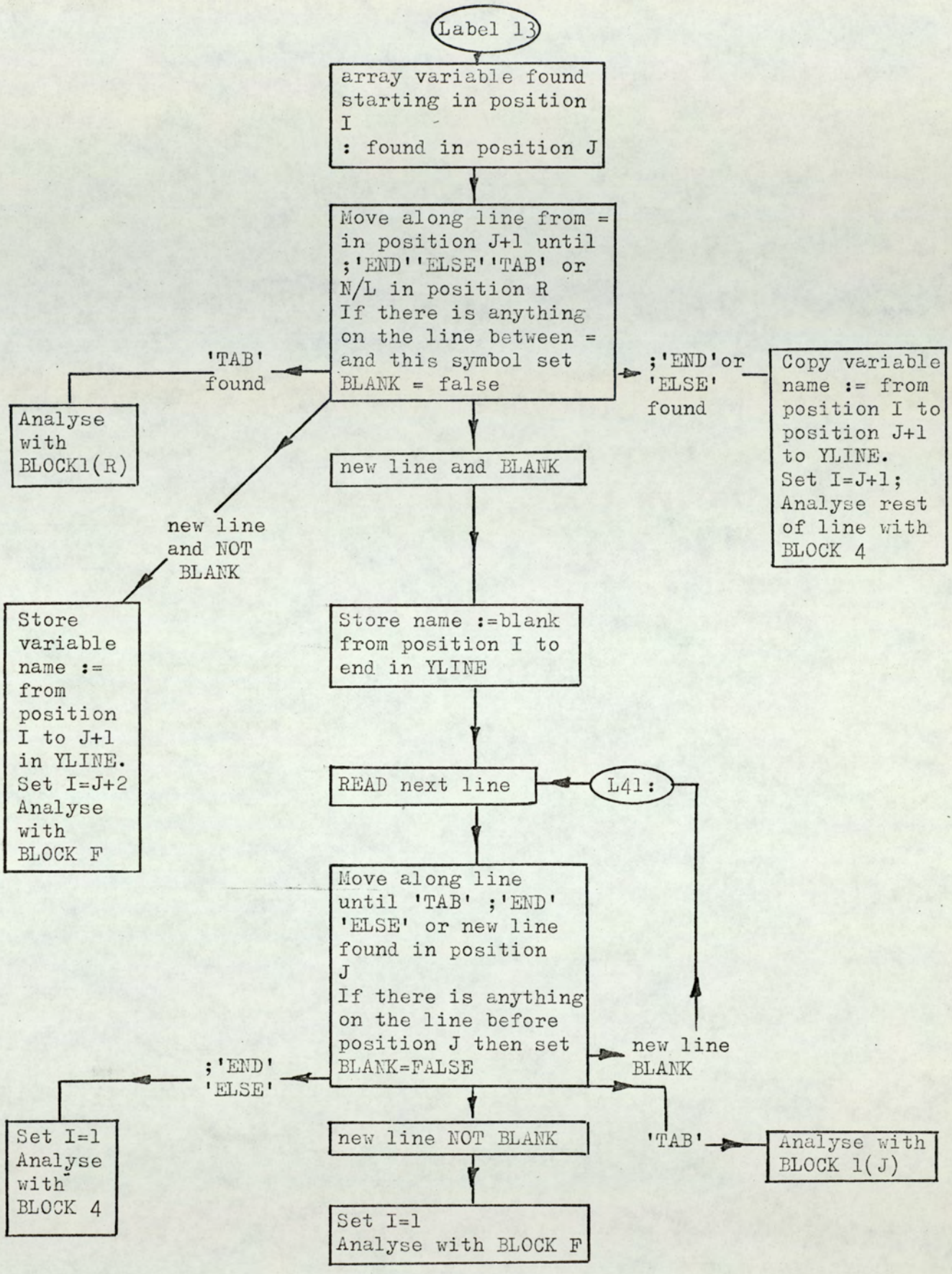

Label II:
Procedure has been found on this line. Arrow found in Position R
Look along line from beginning. Is there a : ?
YES
Copy line from beginning up to : inclusive (label name) to File 2. Update file pointer P. Set I = position of : on line.
NO
Set I = 0
Move along line until non-blank character. Set new I to this position.
Move along line from new I until ← or (found in position J
Copy procedure name from I to J to IMAG array. Compare name in IMAG to list of Hydro procedure names Retrieve number, Y, representing procedure
label 1
YES
Is procedure MAXVALS or DECLARATIONS ?
Copy input lines between and next *** line to File 4. Go to label 1 in main program to process next input line
NO
Is procedure PROGRAM END ?
YES
Go to Phase 2
NO
Copy number Y into STACK array

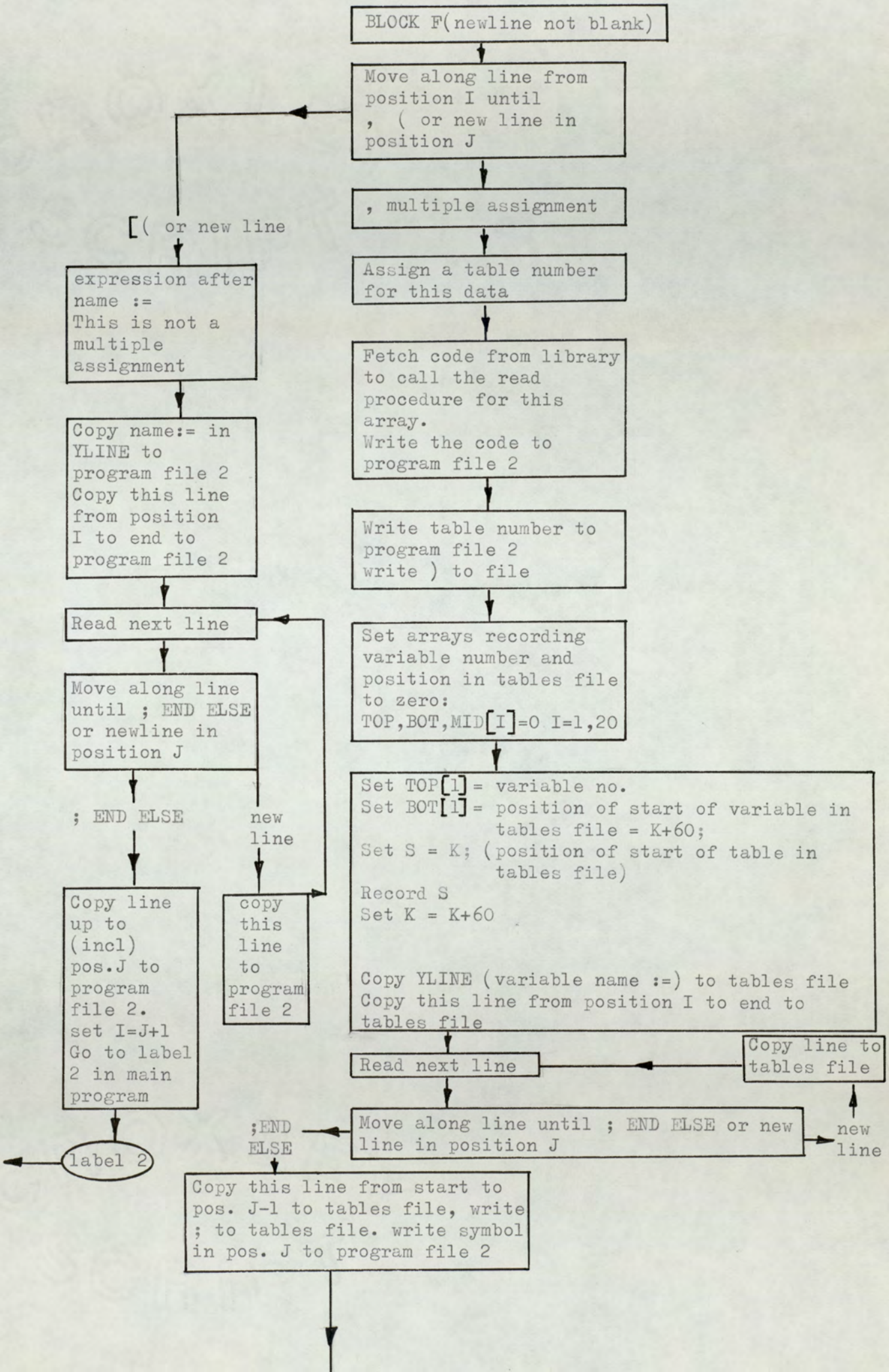

Write 'BEGIN' to File 2
Was (found in position J ?
NO
Set PRLINE array to symbols 0);
YES
Move along line from (in position J until 'TAB' found in position I
Move along from I until) found in positi on KI. Copy line from I+1 to Kl-1 to PRLINE array
Go to label 1 in main program to process next line
label 1

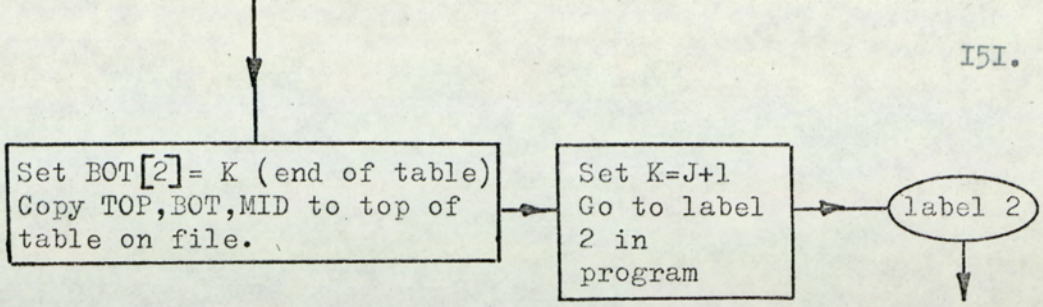

Label 12:
Single variable assignment found on this line starting in position I. variable number is Y. Set BLANK = TRUE.
Move along line from position J (on :) until 'TAB' ; 'END' 'ELSE' or new line found in position K1.
If any symbol except space occurs on the line between = and this symbol then set BLANK = FALSE.
'TAB' found
Process with BLOCK 3 (K1) code
; 'END' or ELSE found
Copy line from I to K1 to File 2. Set I = K1+1.
Go to label 2
new line found and BLANK=FALSE
Copy line from I to end to File 2
Read next line. Set pointer I = 1
Move along line until ; 'END' 'ELSE' or newline in position J
newline found
;'END' or 'ELSE' found
Copy line up to symbol in position J to File 2. Set I = J+1.
Go to label 2
new line found and BLANK=TRUE
Only NAME := on this line. Copy line to YLINE array
Read next line
L22:
Move along line until 'TAB'; 'END' 'ELSE' or newline found in position J
If there is anything on the line before this symbol then set BLANK = FALSE
newline and BLANK
'TAB' found
Process with BLOCK3(J) code
;'END' or 'ELSE' found
Copy YLINE to File 2. Copy this line up to position J to File 2. Set I=J+1.
Go to label 2
newline and not BLANK.

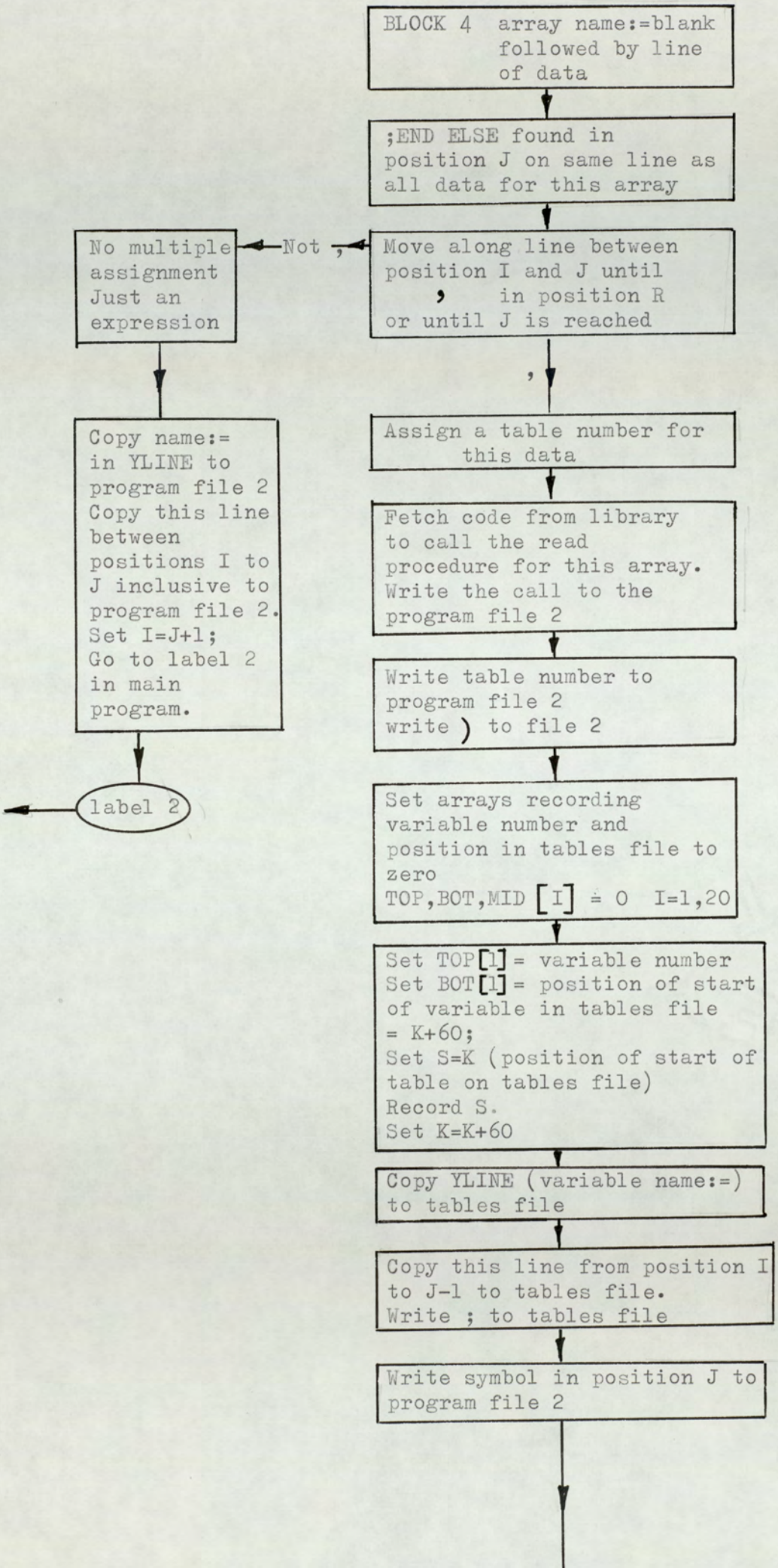

Copy YLINE to
file 2.
Copy this line
to File 2.
L23:
Read next line
Copy this
line to
File 2
newline
found
Move along line
until ;'END''ELSE'
or newline found
in position J
;'END'
or'ELSE'
found
Copy this line to
File 2 up to
position J.
Set I=J+1.
Go to label 2.

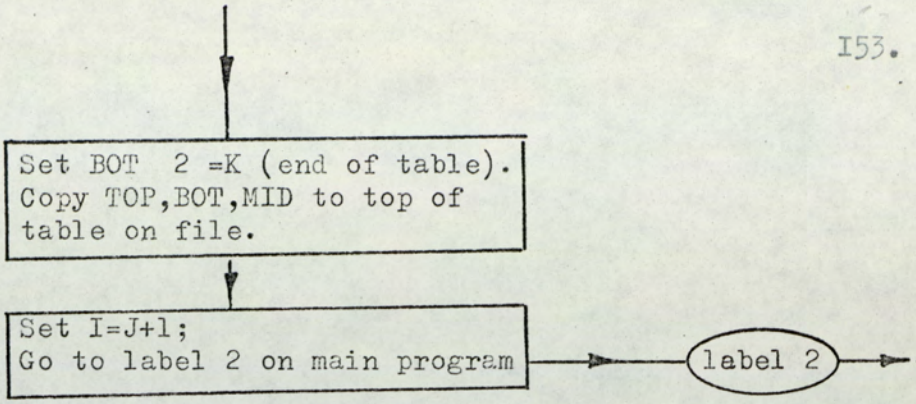

Block 3(V)
V is the position of first 'TAB' after NAME:=
Fetch code from library to read data into this variable from a table.
Write the read call to File 2
Move along line from position V until ;'END''ELSE' or newline found in position R
;'END' or 'ELSE' found
Copy expression for table number from position (V+1) to position (R-1) to File 2.
Write) to File2
Write symbol in position R to File 2.
Set I=R+1
Go to label 2
newline found
Copy line from position (V+1) to end of line to File 2
L32:
Read next line
Move along line until ;'END''ELSE' or newline found in position R
newline found
;'END' or 'ELSE'found
Copy line from beginning to position (R-1) to File 2
Write) and symbol in position R to File 2.
Set I=R+1.
Go to label 2.

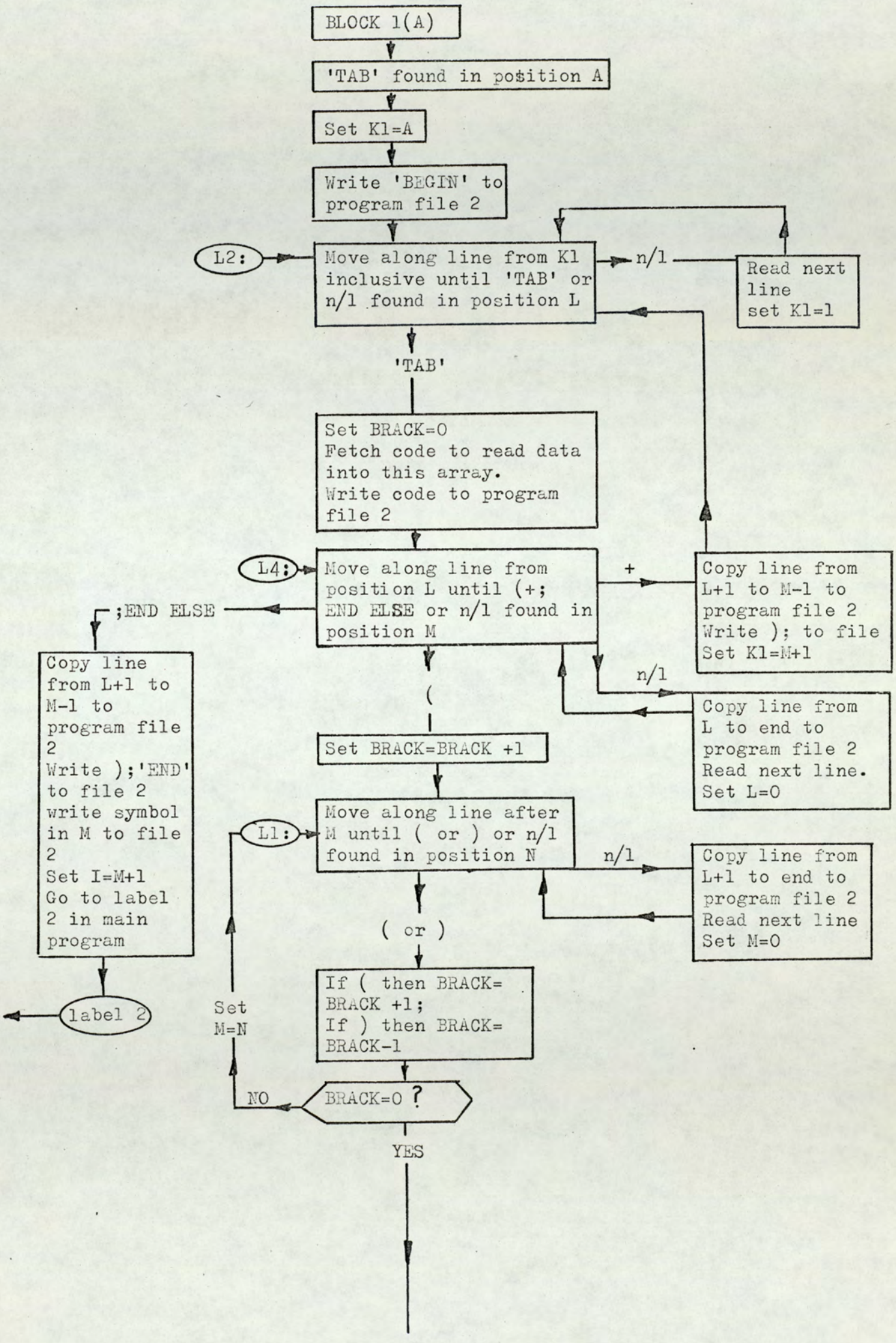

Label 13
array variable found starting in position I
: found in position J
Move along line from = in position J+1 until ;'END''ELSE''TAB' or N/L in position R
If there is anything on the line between = and this symbol set BLANK = false
'TAB' found
Analyse with BLOCK1(R)
;'END'or 'ELSE' found
Copy variable name := from position I to position J+1 to YLINE.
Set I=J+1;
Analyse rest of line with BLOCK 4
new line and BLANK
new line and NOT BLANK
Store variable name := from position I to J+1 in YLINE.
Set I=J+2
Analyse with BLOCK F
Store name :=blank from position I to end in YLINE
READ next line
L41:
Move along line until 'TAB' ;'END' 'ELSE' or new line found in position J
If there is anything on the line before position J then set BLANK=FALSE
new line BLANK
;'END' 'ELSE'
Set I=1
Analyse with BLOCK 4
new line NOT BLANK
'TAB'
Analyse with BLOCK 1(J)
Set I=1
Analyse with BLOCK F

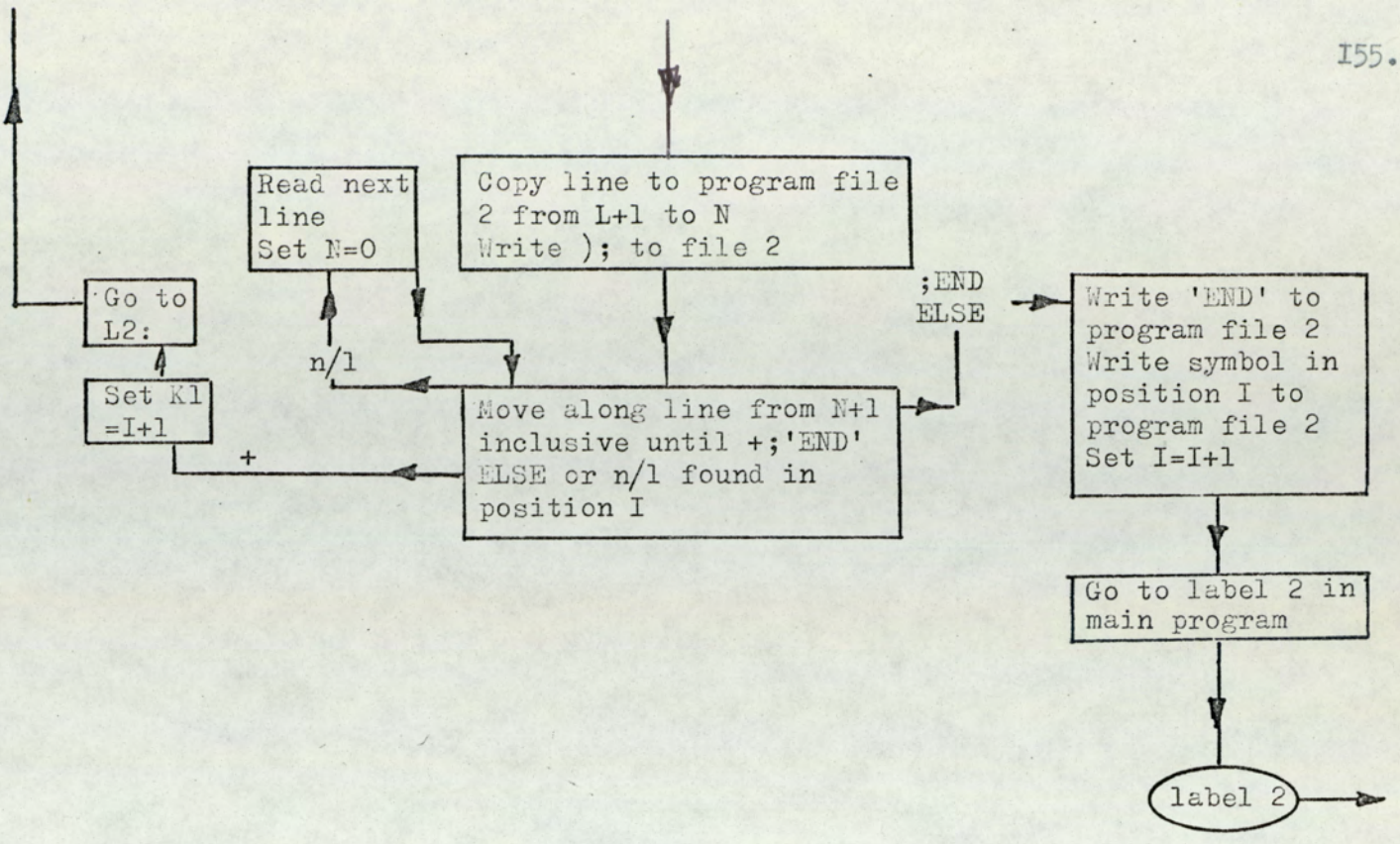

BLOCK F(newline not blank)
Move along line from position I until , (or new line in position J
, multiple assignment
Assign a table number for this data
Fetch code from library to call the read procedure for this array. Write the code to program file 2
Write table number to program file 2 write) to file
Set arrays recording variable number and position in tables file to zero: TOP,BOT,MID[I]=0 I=1,20
Set TOP[1] = variable no.
Set BOT[1] = position of start of variable in tables file = K+60;
Set S = K; (position of start of table in tables file)
Record S
Set K = K+60
Copy YLINE (variable name :=) to tables file
Copy this line from position I to end to tables file
Read next line
Copy line to tables file
new line
Move along line until ; END ELSE or new line in position J
;END ELSE
Copy this line from start to pos. J-1 to tables file, write ; to tables file. write symbol in pos. J to program file 2
[(or new line
expression after name := This is not a multiple assignment
Copy name:= in YLINE to program file 2 Copy this line from position I to end to program file 2
Read next line
Move along line until ; END ELSE or newline in position J
; END ELSE
new line
Copy line up to (incl) pos.J to program file 2. set I=J+1 Go to label 2 in main program
copy this line to program file 2
label 2

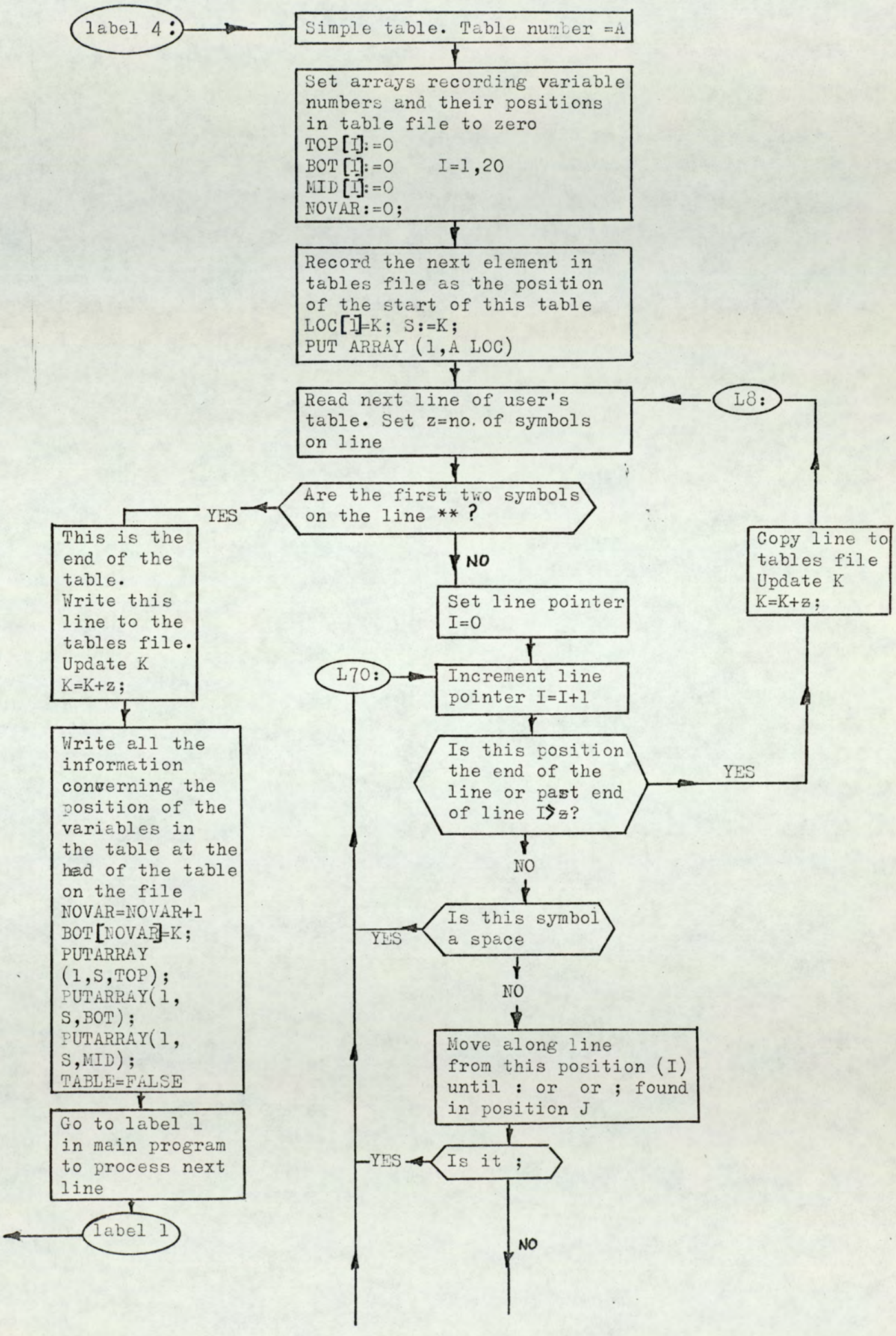

Set BOT[2] = K (end of table)
Copy TOP,BOT,MID to top of table on file.
Set K=J+1
Go to label 2 in program
label 2

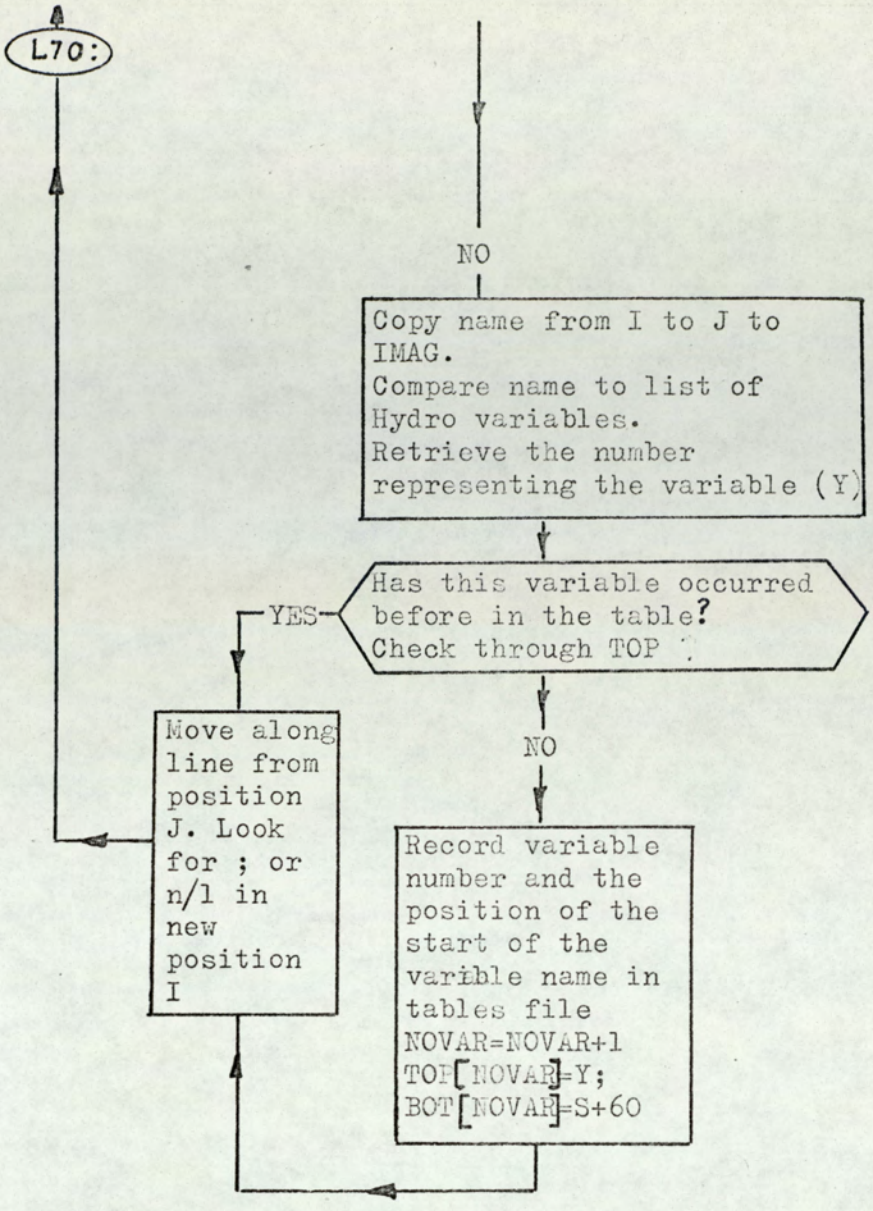

BLOCK 4 array name:=blank followed by line of data
;END ELSE found in position J on same line as all data for this array
Move along line between position I and J until , in position R or until J is reached
Not ,
No multiple assignment Just an expression
,
Assign a table number for this data
Fetch code from library to call the read procedure for this array. Write the call to the program file 2
Write table number to program file 2 write) to file 2
Set arrays recording variable number and position in tables file to zero TOP,BOT,MID [I] = 0 I=1,20
Set TOP[1] = variable number Set BOT[1] = position of start of variable in tables file = K+60; Set S=K (position of start of table on tables file) Record S. Set K=K+60
Copy YLINE (variable name:=) to tables file
Copy this line from position I to J-1 to tables file. Write ; to tables file
Write symbol in position J to program file 2
Copy name:= in YLINE to program file 2 Copy this line between positions I to J inclusive to program file 2. Set I=J+1; Go to label 2 in main program.
label 2

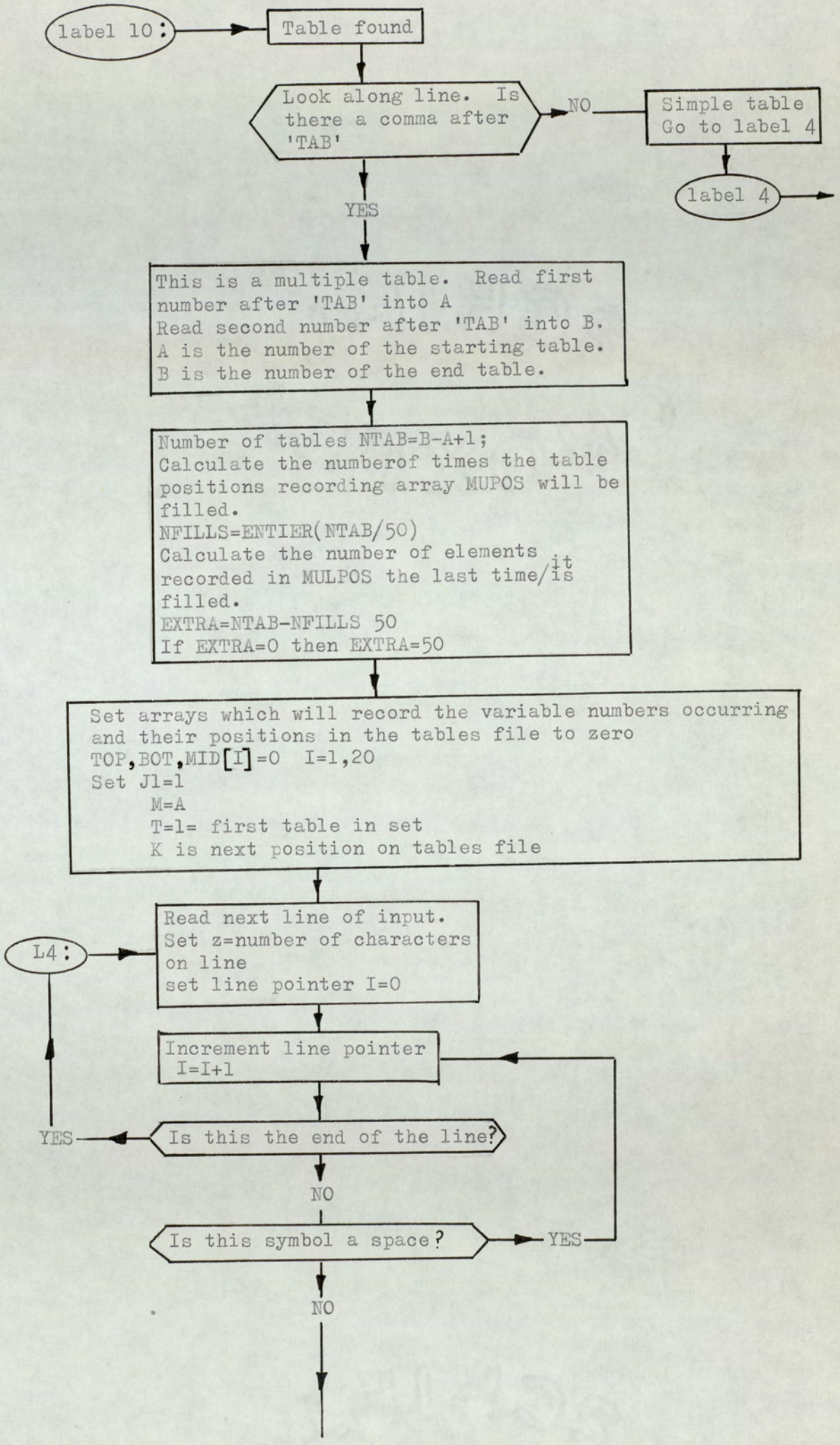

Set BOT 2 =K (end of table).
Copy TOP,BOT,MID to top of table on file.
Set I=J+1;
Go to label 2 on main program
label 2

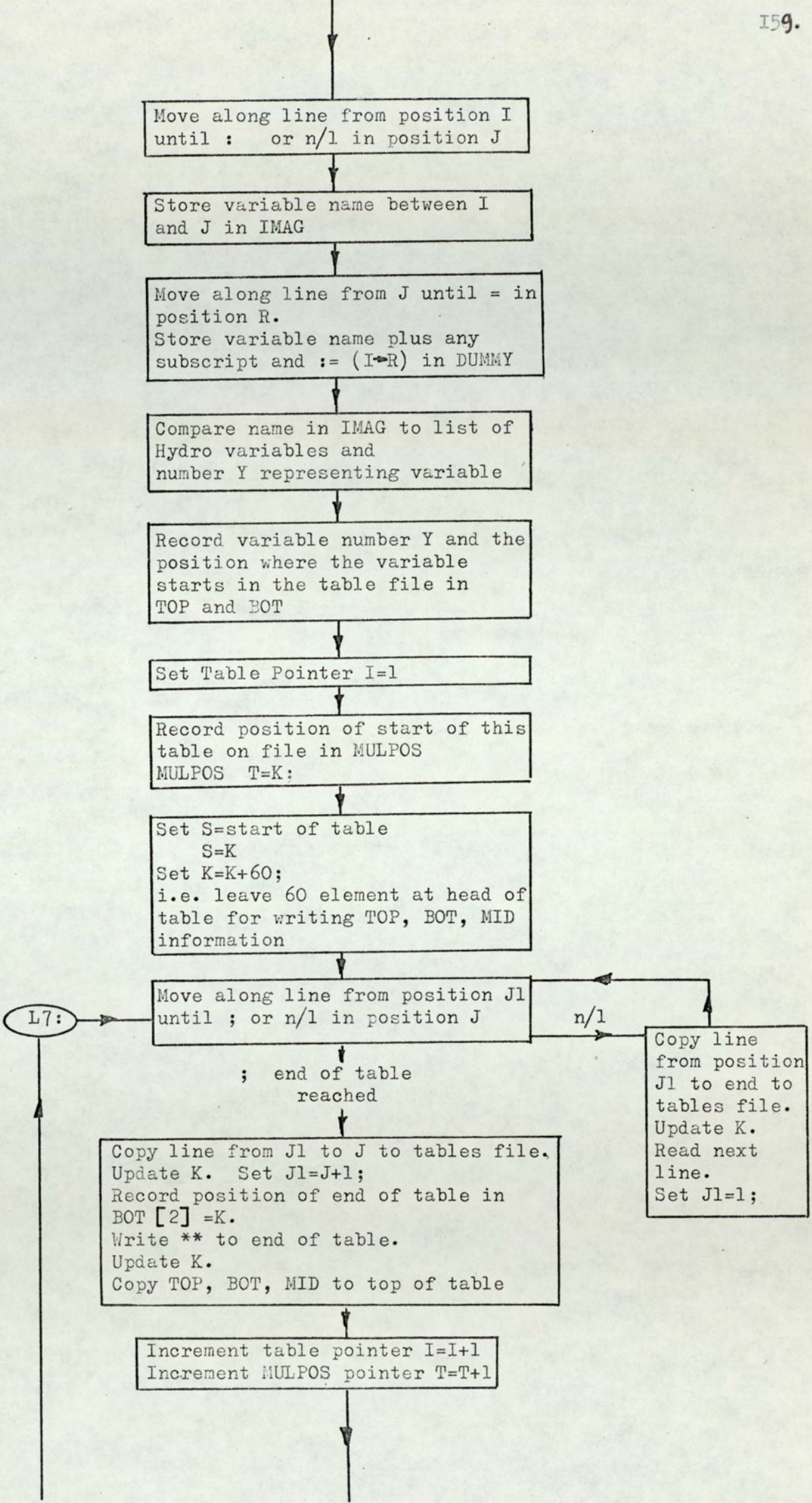

BLOCK 1(A)
'TAB' found in position A
Set K1=A
Write 'BEGIN' to program file 2
L2:
Move along line from K1 inclusive until 'TAB' or n/l found in position L
n/l
Read next line set K1=1
'TAB'
Set BRACK=0
Fetch code to read data into this array.
Write code to program file 2
L4:
Move along line from position L until (+; END ELSE or n/l found in position M
+
Copy line from L+1 to M-1 to program file 2 Write); to file Set K1=M+1
;END ELSE
n/l
(
Copy line from L to end to program file 2 Read next line. Set L=0
Set BRACK=BRACK +1
Copy line from M-1 to program file 2 Write);'END' to file 2 write symbol in M to file 2 Set I=M+1 Go to label 2 in main program
L1:
Move along line after M until (or) or n/l found in position N
n/l
Copy line from L+1 to end to program file 2 Read next line Set M=0
(or)
label 2
Set M=N
If (then BRACK= BRACK +1; If) then BRACK= BRACK-1
NO
BRACK=0 ?
YES

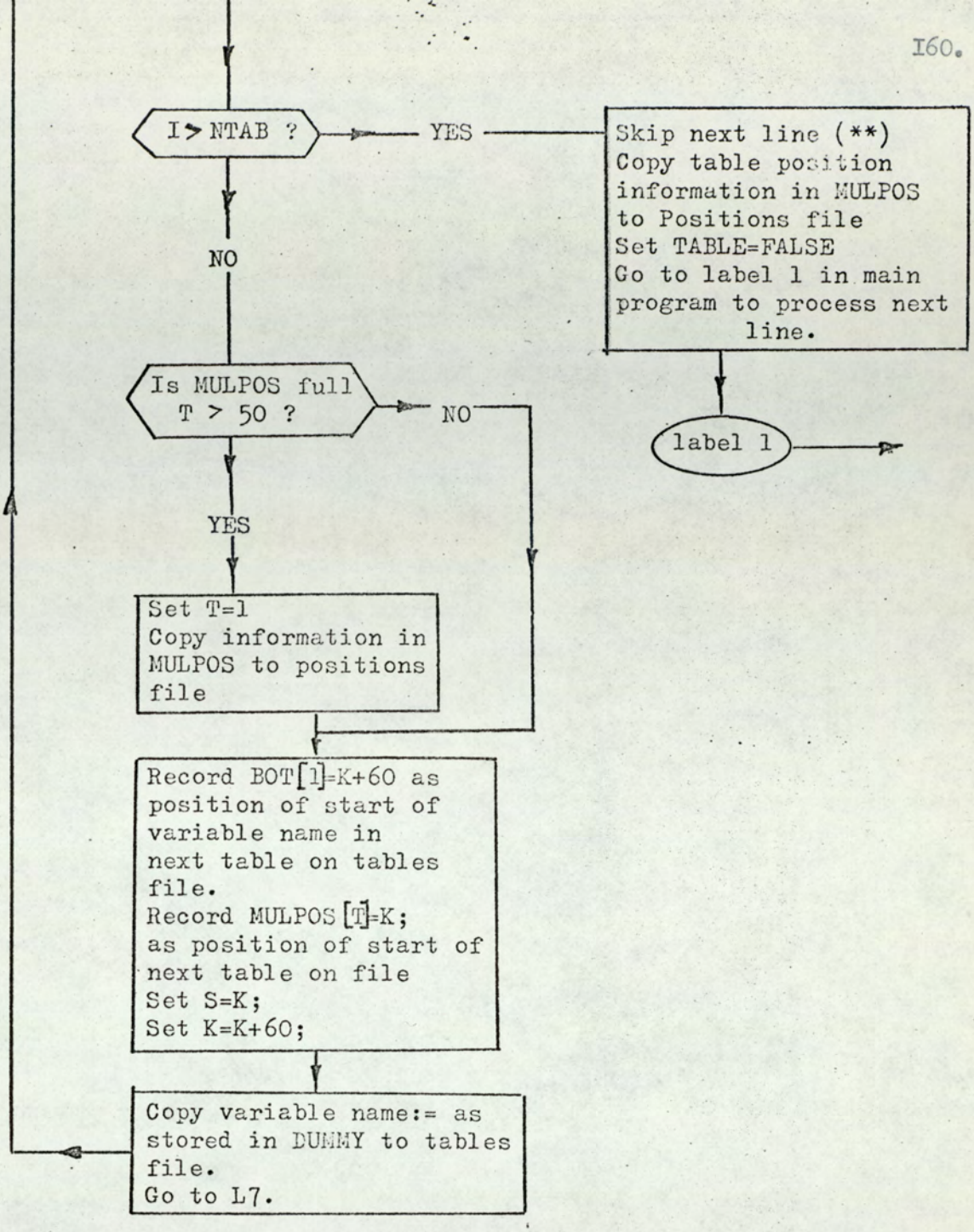

Read next line
Set N=0
Copy line to program file 2 from L+1 to N
Write); to file 2
;END
ELSE
Write 'END' to program file 2
Write symbol in position I to program file 2
Set I=I+1
Go to label 2 in main program
label 2
Go to L2:
Set K1 =I+1
n/l
+
Move along line from N+1 inclusive until +;'END' ELSE or n/l found in position I

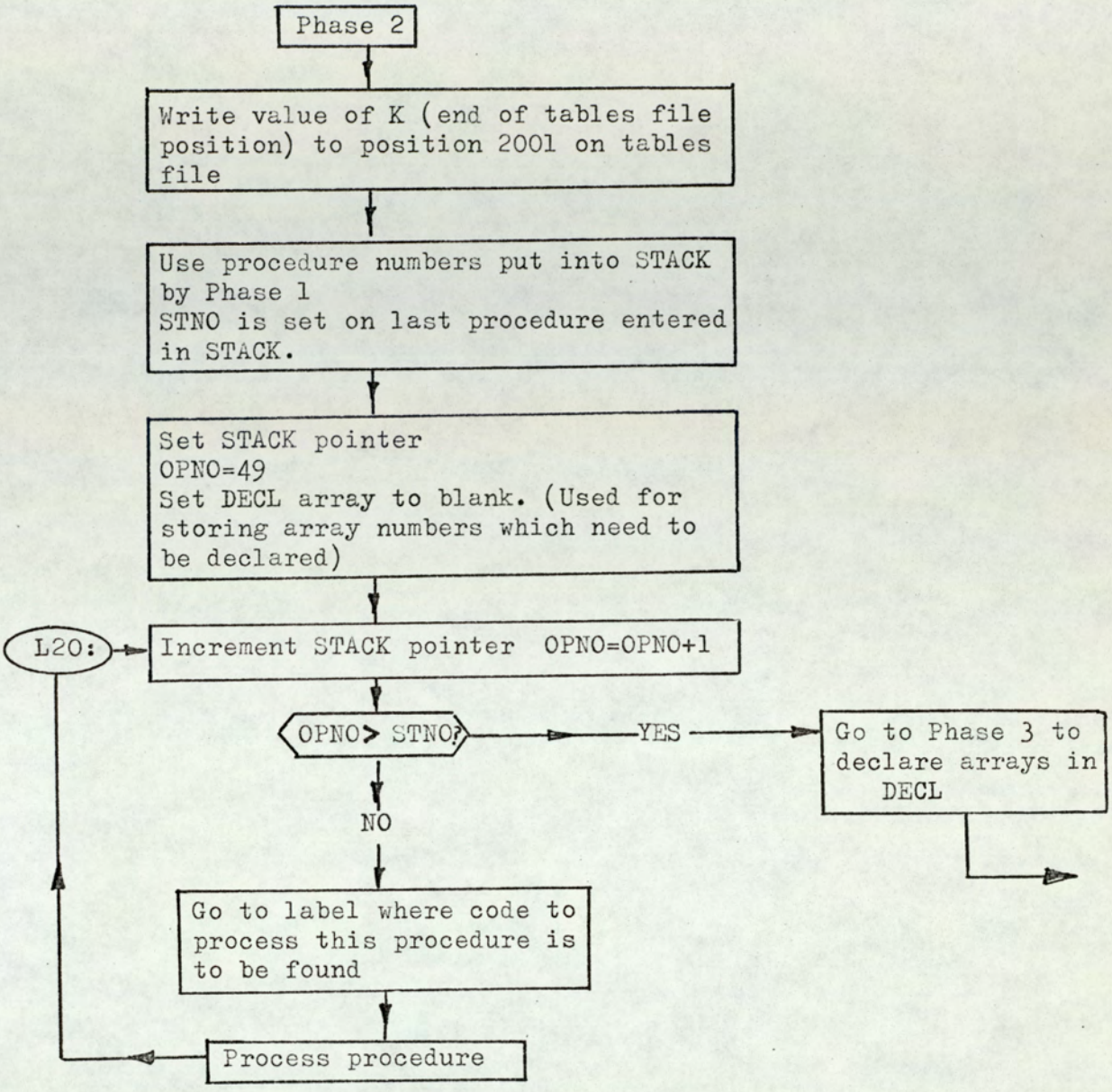

label 4:
Simple table. Table number =A
Set arrays recording variable numbers and their positions in table file to zero
TOP[I]:=0
BOT[I]:=0 I=1,20
MID[I]:=0
NOVAR:=0;
Record the next element in tables file as the position of the start of this table
LOC[I]=K; S:=K;
PUT ARRAY (1,A LOC)
L8:
Read next line of user's table. Set z=no. of symbols on line
Are the first two symbols on the line ** ?
YES
NO
This is the end of the table. Write this line to the tables file. Update K
K=K+z;
Copy line to tables file Update K
K=K+z;
Set line pointer
I=0
L70:
Increment line pointer I=I+1
Is this position the end of the line or past end of line I>z?
YES
NO
Is this symbol a space
YES
NO
Move along line from this position (I) until : or or ; found in position J
Is it ;
YES
NO
Write all the information concerning the position of the variables in the table at the head of the table on the file
NOVAR=NOVAR+1
BOT[NOVAR]=K;
PUTARRAY (1,S,TOP);
PUTARRAY(1, S,BOT);
PUTARRAY(1, S,MID);
TABLE=FALSE
Go to label 1 in main program to process next line
label 1

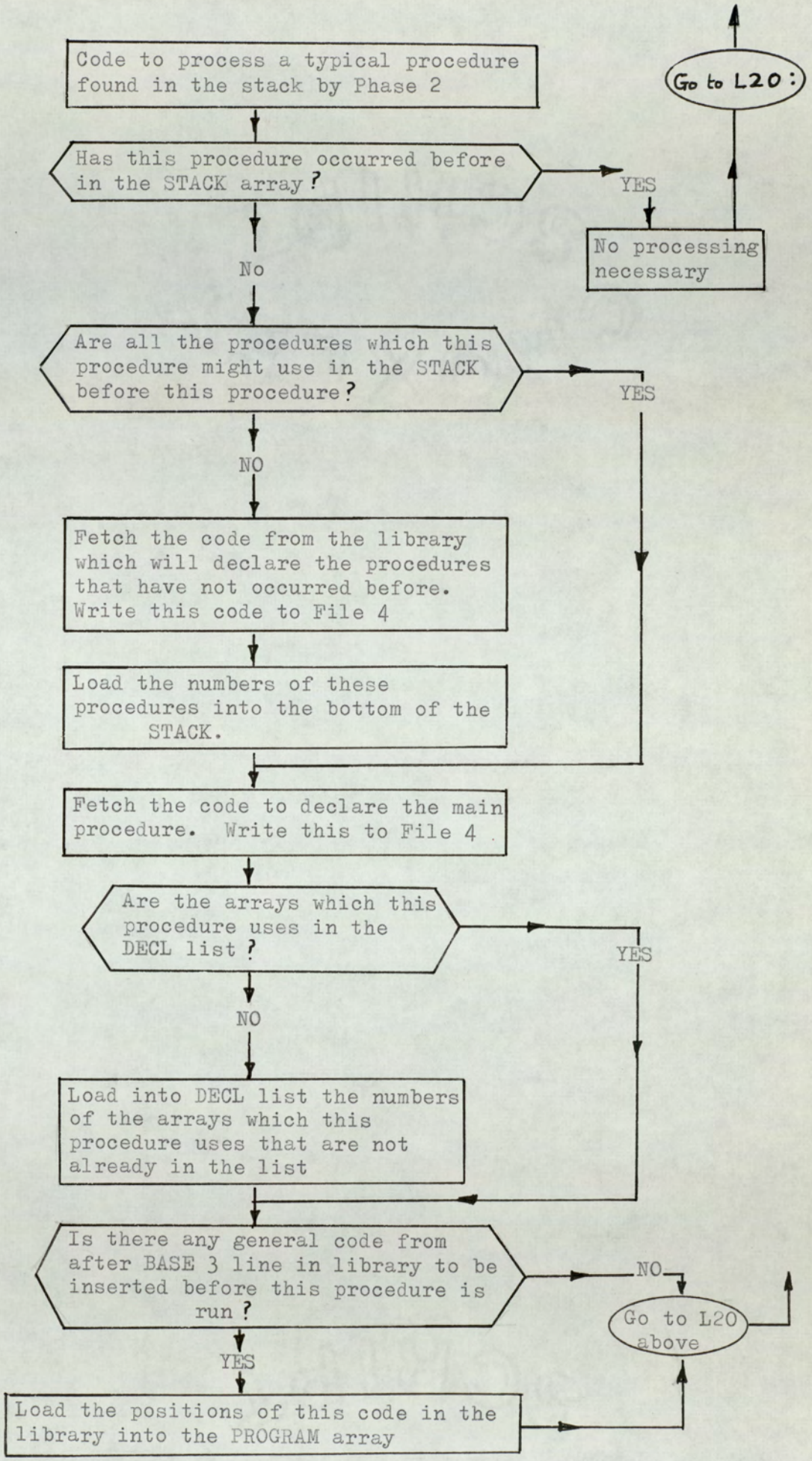

L70:
NO
Copy name from I to J to IMAG.
Compare name to list of Hydro variables.
Retrieve the number representing the variable (Y)
Has this variable occurred before in the table? Check through TOP
YES
NO
Move along line from position J. Look for ; or n/l in new position I
Record variable number and the position of the start of the varible name in tables file
NOVAR=NOVAR+1
TOP[NOVAR]=Y;
BOT[NOVAR]=S+60

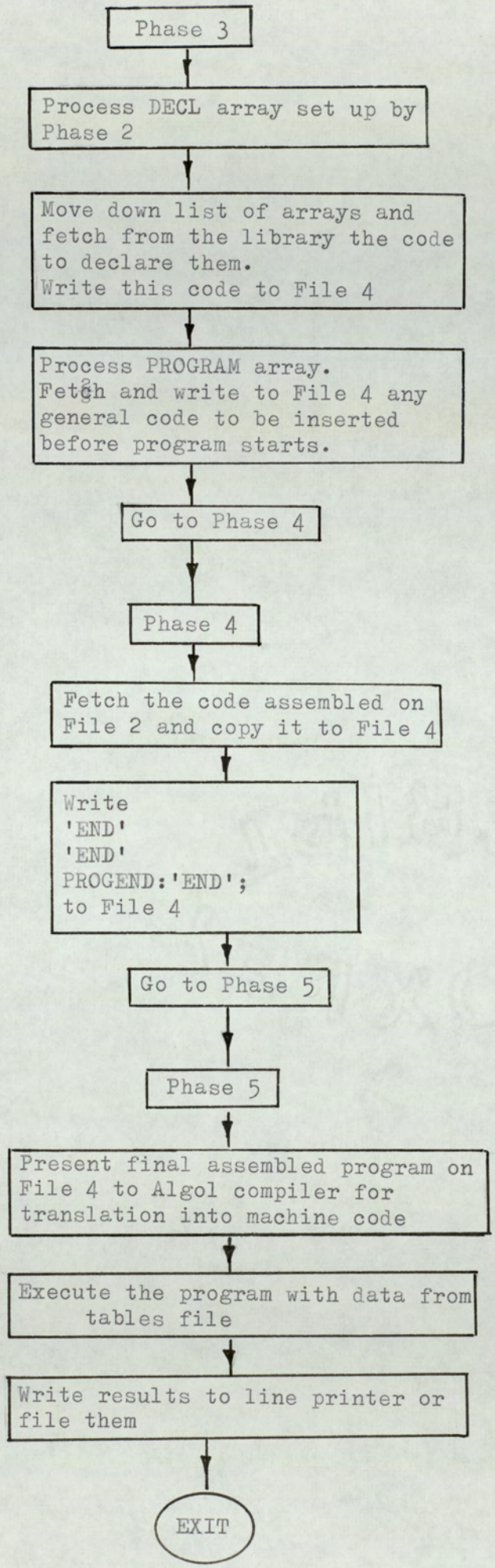

label 10:
Table found
Look along line. Is there a comma after 'TAB'
NO
Simple table Go to label 4
label 4
YES
This is a multiple table. Read first number after 'TAB' into A
Read second number after 'TAB' into B.
A is the number of the starting table.
B is the number of the end table.
Number of tables NTAB=B-A+1;
Calculate the numberof times the table positions recording array MUPOS will be filled.
NFILLS=ENTIER(NTAB/50)
Calculate the number of elements recorded in MULPOS the last time it is filled.
EXTRA=NTAB-NFILLS 50
If EXTRA=0 then EXTRA=50
Set arrays which will record the variable numbers occurring and their positions in the tables file to zero
TOP,BOT,MID[I]=0 I=1,20
Set J1=1
M=A
T=1= first table in set
K is next position on tables file
L4:
Read next line of input.
Set z=number of characters on line
set line pointer I=0
Increment line pointer I=I+1
YES
Is this the end of the line?
NO
Is this symbol a space?
YES
NO


Move along line from position I until : or n/l in position J
Store variable name between I and J in IMAG
Move along line from J until = in position R.
Store variable name plus any subscript and := (I→R) in DUMMY
Compare name in IMAG to list of Hydro variables and number Y representing variable
Record variable number Y and the position where the variable starts in the table file in TOP and BOT
Set Table Pointer I=1
Record position of start of this table on file in MULPOS MULPOS T=K:
Set S=start of table
S=K
Set K=K+60;
i.e. leave 60 element at head of table for writing TOP, BOT, MID information
L7:
Move along line from position J1 until ; or n/l in position J
n/l
Copy line from position J1 to end to tables file.
Update K.
Read next line.
Set J1=1;
; end of table reached
Copy line from J1 to J to tables file.
Update K. Set J1=J+1;
Record position of end of table in BOT [2] =K.
Write ** to end of table.
Update K.
Copy TOP, BOT, MID to top of table
Increment table pointer I=I+1
Increment MULPOS pointer T=T+1


I > NTAB ?
YES
Skip next line (**)
Copy table position information in MULPOS to Positions file
Set TABLE=FALSE
Go to label 1 in main program to process next line.
label 1
NO
Is MULPOS full T > 50 ?
NO
YES
Set T=1
Copy information in MULPOS to positions file
Record BOT[1]=K+60 as position of start of variable name in next table on tables file.
Record MULPOS[T]=K; as position of start of next table on file
Set S=K;
Set K=K+60;
Copy variable name:= as stored in DUMMY to tables file.
Go to L7.


Phase 2
Write value of K (end of tables file position) to position 2001 on tables file
Use procedure numbers put into STACK by Phase 1
STNO is set on last procedure entered in STACK.
Set STACK pointer
OPNO=49
Set DECL array to blank. (Used for storing array numbers which need to be declared)
L20:
Increment STACK pointer OPNO=OPNO+1
OPNO> STNO?
YES
Go to Phase 3 to declare arrays in DECL
NO
Go to label where code to process this procedure is to be found
Process procedure


Code to process a typical procedure found in the stack by Phase 2
Has this procedure occurred before in the STACK array?
YES
No processing necessary
Go to L20:
No
Are all the procedures which this procedure might use in the STACK before this procedure?
YES
NO
Fetch the code from the library which will declare the procedures that have not occurred before. Write this code to File 4
Load the numbers of these procedures into the bottom of the STACK.
Fetch the code to declare the main procedure. Write this to File 4
Are the arrays which this procedure uses in the DECL list?
YES
NO
Load into DECL list the numbers of the arrays which this procedure uses that are not already in the list
Is there any general code from after BASE 3 line in library to be inserted before this procedure is run?
NO
Go to L20 above
YES
Load the positions of this code in the library into the PROGRAM array


Phase 3
Process DECL array set up by Phase 2
Move down list of arrays and fetch from the library the code to declare them.
Write this code to File 4
Process PROGRAM array.
Fetch and write to File 4 any general code to be inserted before program starts.
Go to Phase 4
Phase 4
Fetch the code assembled on File 2 and copy it to File 4
Write
'END'
'END'
PROGEND:'END';
to File 4
Go to Phase 5
Phase 5
Present final assembled program on File 4 to Algol compiler for translation into machine code
Execute the program with data from tables file
Write results to line printer or file them
EXIT

5.7. Description of the Flow Diagram for the translator program

The translator program is entered at Phase 1 which first copies the declarations of the special Hydro data reading procedures from the line library to File 4 which is to be used for assembling the final program equivalent to the user's input. Some general code used by all programs assembled by Hydro is also copied across at this stage.

After this preliminary operation the translator passes to Label 1 where the next, or first, line of the user's data is read in. The user's data is only read in one line at a time and the symbols are stored in a line array called IMAGE. An imaginary pointer is then set to the first symbol on the line.

At label 2, the translator looks along the line for the two symbols *'TAB'. If these are found at the start of the line then a table has been found and the translator starts to process the input with the code for analysing tables at label 10.

If a table is not found then the translator looks for the procedure terminating three asterisks. If they are found then a procedure has just been processed and all that is now required is to fetch the relevant code from the line library to call this procedure. This code is copied to the file where the program is being assembled. When the procedure name was first encountered the results table number in brackets after the procedure name, if present, would have been stored in an array called PRLINE in character form. If no results table number is given then the results are assumed to go only to the lineprinter, and the character zero is stored in PRLINE.

The number or expression that was stored in PRLINE is now copied at the end of the procedure call as the last parameter to the procedure. The

call is terminated by writing a left bracket and semi-colon to the file, followed by an Algol 'END' symbol. The 'END' symbol is written because the translator would have automatically written a 'BEGIN' symbol to the file when the procedure name was found in the input. It is necessary to surround the data and call with 'BEGIN' and 'END' in the assembled program so that all the information given by the user for that procedure, between the left arrow and the terminating three asterisks, is related to that procedure in the assembled program.

If neither a table nor the three asterisks line is found at label 2 then the translator next looks for a left arrow signifying that a procedure name has been written on the line.

If the arrow is found then the translator processes the procedure with the code at label 11. If an arrow is not found then either Hydro data assignments or pure Algol code, or a mixture of the two must occur on the line.

Starting from the position at which the pointer is at present set the translator moves along the line looking for any of the Algol symbols 'FOR' 'IF' 'END' 'ELSE';: or newline. If a new line is the first symbol to occur then the line contains only Algol code which is copied intact to the program file. The translator will then go back to label 1 to read the next line of data and will proceed as before. Similarly, if any of the symbols 'END' 'ELSE' or ; are found then the translator copies the line beginning at the original position of the pointer up to and including the symbol found to the program file. The pointer is set at the next symbol and processing starts again from label 2 in the translator.

However, if either 'IF' or 'FOR' is found then the translator looks along the line for the related 'THEN' or 'DO' symbols. If a new line occurs before the symbols are found then the line is copied, from the pointer to the new line, to the program file, and the next line is read from the

input, the pointer being set at the beginning of the line. As before, the translator looks along the line for the 'THEN' or 'DO' symbols. As many new lines are read and copied as necessary to locate these symbols. When they are found the translator copies the line from the present position of the pointer up to and including this symbol to the program file. The pointer is then set after the symbol and processing of the input again continues from label 2.

The only other symbol which could have been found is the colon symbol :. When this symbol is found two possibilities exist. Either the colon is followed by the = symbol, in which case a data assignment of some kind follows the equals sign, or it is not in which case the colon is terminating a label name. Thus, the translator first looks after the colon for the equals sign. If it is not present then the translator copies the label name from the position of the pointer up to and including the colon to the program file. The pointer is then set at the position after the colon and processing begins again from label 2.

If the equals sign is found then the translator must check whether the variable name preceding the colon is a Hydro variable name or a name employed by the user in the normal Algol way. This is done by copying the name from the position of the pointer to the colon into a dummy array called IMAG. Any spaces which occur in the input before the variable name are omitted. The IMAG array is then compared to strings of Hydro variable names by means of the procedure DATREC. If a match is not found then the variable is not a recognised Hydro variable. The translator moves along the input line until one of the Algol 'END' 'ELSE' or ; symbols is found. If a new line occurs before any of these symbols the line is copied to the program file and the next line is read, the pointer is set at the beginning of this line, and the search continues. When one of the symbols is found the translator copies the line from the

pointer to the symbol to the program file. The pointer is set after the symbol and processing recommences from label 2 of the translator. However, if the variable name is recognised by DATREC then the number representing the variable is recovered from this procedure and set into a translator variable Y. If the number representing a Hydro array variable then the translator starts to process the input with the code at label 13, but if it represents a single variable the input is processed with the code at label 12.

The code at label 11 is used for processing an input procedure. If an arrow was found on an input line then a procedure name occurs on the line. The translator sets the pointer at the beginning of the line and looks along from there for a colon which would signify that a label precedes the procedure name. If a colon is found then the label, including spaces, starting at the beginning of the line and ending at the colon, is copied to the program file 2. The colon is also copied across. The pointer is then set after the colon. If a colon is not found the pointer is left at the start of the line. The translator then looks for the first non space character after the present position of the pointer. The pointer is now reset at this new position, which is the start of the procedure name. The translator looks along the line to locate the position of the left arrow, or a left round bracket. The name between the position of the pointer and this name is copied into IMAG and is then compared by the translator procedure PRREC to a list of permissible Hydro procedure names. When a match is found the number representing the procedure is retrieved. A check is made to see whether the procedure is either MAXVALS, DECLARATIONS or PROGRAM.END. If it is PROGRAM.END. then the end of the user's input has been reached and the translator may pass on to the next phase of analysis, but if the procedure is MAXVALS or DECLARATIONS then the

user's input between the left arrow and the procedure terminating three asterisks line is copied to the program file4. The pointer is set at the beginning of the next line and processing continues from label 1 of the translator. However, if the procedure is an ordinary Hydro routine its representative number is copied into the next available position in an array called STACK for later use by the translator. The Algol symbol 'BEGIN' is then written to the program file 2. If a left round bracket was found after the procedure name then the user has supplied the number of a table to which he requires his data to be sent, as well as to the lineprinter. The translator moves along the line until the 'TAB' symbol is found. It then copies the number or expression between 'TAB' and the right closing bracket into an array called PRLINE. However, if the brackets were not present, PRLINE is set to the character 0 which indicates that the results are to be written to the line printer only.

The translator then reverts to its label 1 position to read in the next line of the users data.

PRLINE is used later by the translator when all data given for the procedure by the user has been processed, and the translator is assembling code for the procedure call itself. The expression in PRLINE is copied to the program file 2 at that point as the last parameter to the procedure.

The code at label 10 is used when a table has been supplied by the user. If the *'TAB' symbols are found at the beginning of a line then the user has given data in the form of a table. The translator looks past the 'TAB' symbol for a comma which would indicate that there are two numbers separated by a comma in which case a multiple table follows. If a simple table has been found it is processed with the code at label 4.

If a comma is found then a multiple table has been given and the translator sets into A the number of the first table number and into B the second number. At this point the translator initialises three twenty element arrays, TOP, BOT and MID to zeros. These arrays will later be written at the head of the user's table when it is copied to the tables file. During the analysis of the table, numbers will be set into these arrays which will form a directory of the representative numbers of the variables in the table, their locations in the table and the form of the data given to each variable respectively. The MID array is always left with its elements zero by the translator, since a zero signifies to the assembled program that the data for the variable is to be read in character form which is always true of data input to the translator by the user. It is the assembled program which may write results to a table other than in character form and the MID array is then necessary to indicate to the assembled program whether it is to read data from an input table in character form or from a previously generated results table which may have been written in binary form. After the arrays have been initialised the translator begins to process the contents of the tables. In the case of a multiple table, the first line of the table will contain at least a variable name followed by the symbols :=. Data may also occur on the same line. The translator first stores the variable name up to the : symbol, minus any preceding spaces, into the IMAG array for comparison by the DATREC procedure to the list of Hydro variable names. The part of the line containing the variable name plus := is stored in an array called DUMMY which is written into each simple table which is expanded from the user's compact table. From the DATREC procedure, the variable number is retrieved and is stored in the first element of the TOP array.

The next available location for storage of data on the tables file is recorded in a special tables list as the position at which the table starts on the file. The first data stored for the table will be the three directory arrays TOP, BOT and MID, but the information to be stored in these arrays will not be known until the table has been processed. Therefore, space on the file is left at the head of the table for the insertion of these arrays. The location on the file after this space is recorded in the first element of the BOT array as the position of the start of the variable in that table.
Having done this the translator looks past the = sign in the user's input to the actual data. Each data set in the multiple table is separated by a semicolon. The translator first copies the DUMMY array to the table and then copies the first data set to the tables file and will record in the second element of the BOT array the next available location in the file after the end of this set. The TOP, BOT and MID arrays are then inserted in the space left at the head of the table. This latter procedure is repeated for all the data sets, the starting positionof each table being recorded in the special list, and the BOT array being revised as each table is written. The TOP and MID arrays stay the same for all the tables, since the TOP array only contains the one element recording the variable number, and the MID array contains all zeros for user input.

When the last table has been written away the translator moves past the two asterisks line, which terminates a table and begins to process the next line starting at label 1 of the translator program.

The code at label 4 is employed when a simple table has occurred. The translator again initialises the directory arrays TOP, BOT and MID to zeros, and records in the special list the next available location

on the tables file as the start of the user's table. The translator then looks at the first data line in the table. As with the multiple table, the first variable name is located and stored in IMAG and is then compared to the list of Hydro variables by the DATREC procedure. The number representing the variable is retrieved and stored in the first element of the TOP array. A space is left on the tables file for the directory arrays and the next location in the file after this space is recorded in the first element of the BOT array as the position of the start of the first variable of this table on the file. The translator then copies the user's input to the tables file until a semi colon is found, indicating that all the data for that variable has been given. The next variable name in the table is then located and compared to the list of Hydro variables. If the same variable as before has been given then no additions are made to the TOP and BOT arrays, but the data for the variable including the name up to the next semi colon, is copied to the tables file. This situation can occur when different elements of an array variable are given separately, when the variable name in the table will be followed by subscript brackets.

If a different variable is found then the next available location on the tables file is recorded as the start of that variable in the file in the next element of the BOT array and the variable number is stored in the next element of the TOP array. The variable name and data is copied to the tables file up to the next semi colon.
This procedure is repeated until the two asterisks line is found, indicating that the table has ended. At this point the translator returns to label 1 to read the next line of input.

The code at label 13 is used when an array variable assignment is found. If an array variable name, which is not in a table, is found in the input

two possibilities exist. Either the user has given a string of data to the variable, as in a table, or he has given a table number where the necessary data may be found. The translator now looks past the := symbols after the variable name for a 'TAB' symbol or any of the Algol symbols ;'END' or 'ELSE'. If the 'TAB' symbol is found before any of the others then the user has given at least one table number for this variable. In this case the translator first writes an Algol 'BEGIN' symbol to the program file 2.

Then for every 'TAB' symbol found, the translator retrieves the appropriate array reading code from the line library and writes it to program file 2. The array reading code consists of a call on the procedure ARRAYREAD which has several parameters. Each call in the library has different parameters, which depend on the variable found in the input, but the last parameter is omitted in the line library. This parameter is now filled in by the translator since it represents the table number given by the user after the 'TAB' symbol. One ARRAY READ call is incorporated into the table for every table number given by the user for this variable.

After the last 'TAB' symbol and its following table number have been given the user will have written either a ;'END' or 'ELSE' symbol. When the translator finds this symbol, it writes an Algol 'END' symbol to match the 'BEGIN' which was written when the variable was found. The translator then sets its line pointer to the symbol found and reverts to label 2 to process the rest of the user's input.

If the 'TAB' symbol is not found after the array variable name then actual data must be given at this point in the user's program. The translator must now ascertain whether only one data item is given or several. This is done by checking for a comma in the data. If a comma occurs then several items are given, each separated by a comma. If any

of the symbols ;'END' or 'ELSE' occur before a comma is found then only one data item is given to the array variable. The variable name and its data assignment may be copied directly to the program file 2 in this case in the same way as normal Algol code. The translator then sets its pointer after the terminating symbol found and begins to process the rest of the user's input with the code at label 2.

If a comma is found then a different approach must be made. The translator assigns a table number to this data and copies it to the tables file in the same way as if it had been given as a table by the user. However, code must also be fetched from the library to read the data from the assigned table at this point. The variable number is retrieved in the normal way by comparing its name, using the DATREC procedure, to the list of Hydro variables. The appropriate ARRAYREAD call is fetched from the line library, and the assigned table number is filled in as the last parameter. This code is then copied to the program file 2.

The translator sets its pointer after the data given and reverts to label 2 to process the remainder of the user's input.

The code at label 12 processes the input when a single variable occurs on a line. When a Hydro single variable is found in the user's input then either a table number where the data may be found is given, or the data itself is given. For a single variable, only one table number will be given unlike the position for an array variable where several table numbers may be supplied. Therefore, the translator checks whether a 'TAB' symbol or any of the Algol symbols ;'END' or 'ELSE' occurs first. If a 'TAB' symbol is not found then it is not necessary to assemble reading code. The variable and its data up to and including the symbol found may be copied directly to the program file 2 as though it were ordinary Algol code. However, if the 'TAB' symbol is found, the

translator fetches the appropriate SINGLEREAD call from the line library and writes this to the program file 2. As with all other calls fetched from the library the last parameter to the procedure is omitted, since it represents the table number from which the data must be read, and can only be supplied by the user. The table number between the 'TAB' symbol and any of the symbols ;'END' or 'ELSE' is now written as the last parameter.

The translator sets its pointer after the last symbol found, and recommences processing the user's input with the code at label 2. When all the user's input has been processed the translator moves on to its Phase two analysis. As mentioned earlier, the phase one part of the translator compiles a list,known as the STACK, of all the procedures numbers which occurs in the user's data. This information is now used to assemble code to declare the procedures used and later to declare the arrays used by these procedures.

The translator looks at each procedure number in the list and goes to a position in the translator where a block of code is written which relates to the declaring of that procedure. For the procedure under consideration the translator looks back through the STACK to ascertain whether the procedure has occurred at an earlier stage. If it has, then nothing need be done at this point since all the necessary operations for declaring the procedure body and its associated array variables will have been carried out at the first occurrence in the STACK. However, if this is the first occurrence then several operations must be carried out. Since some Hydro procedures may use other procedures it is necessary to insert at the top of the STACK, when a procedure is processed by phase two, the representative numbers of all procedures it may use. This action is carried out because a procedure used by another at one stage in a program may occur in its own right at a later stage,

or may be used by a different procedure, but all declarations for the smaller procedure will have been carried out at the earlier occurrence of the larger procedure, so that they need not be done again. Thus, when the translator looks back through the STACK it scans through implied procedures as well as actual input procedures, and when any one input procedure is processed by phase two only the procedures which do not occur above it in the list and which are used by this procedure are declared. The body of the main procedure is declared after all of the lower order procedures have been declared. The declarations are made by fetching the relevant code from the line library, the location of the code being related to the procedure number, and writing this code to program file 4. For every Hydro procedure which may be used, the translator contains code indicating which array variables are required for the running of the procedure. When an input procedure is found in the STACK compiled by phase one, the translator stores in a list called DECL the numbers representing the array variables which need to be declared. Only the numbers which are not already in the DECL list are inserted.

When the processing of the STACK list is complete, the phase three section of the translator uses the DECL list compiled by phase two to assemble code for the declaration of the arrays present. For every array number present in the DECL list the translator fetches the appropriate declaring code from the line library and writes this to program file 4.

At this stage all processing of the user's input is complete, and only a final collection of all the assembled code is necessary.

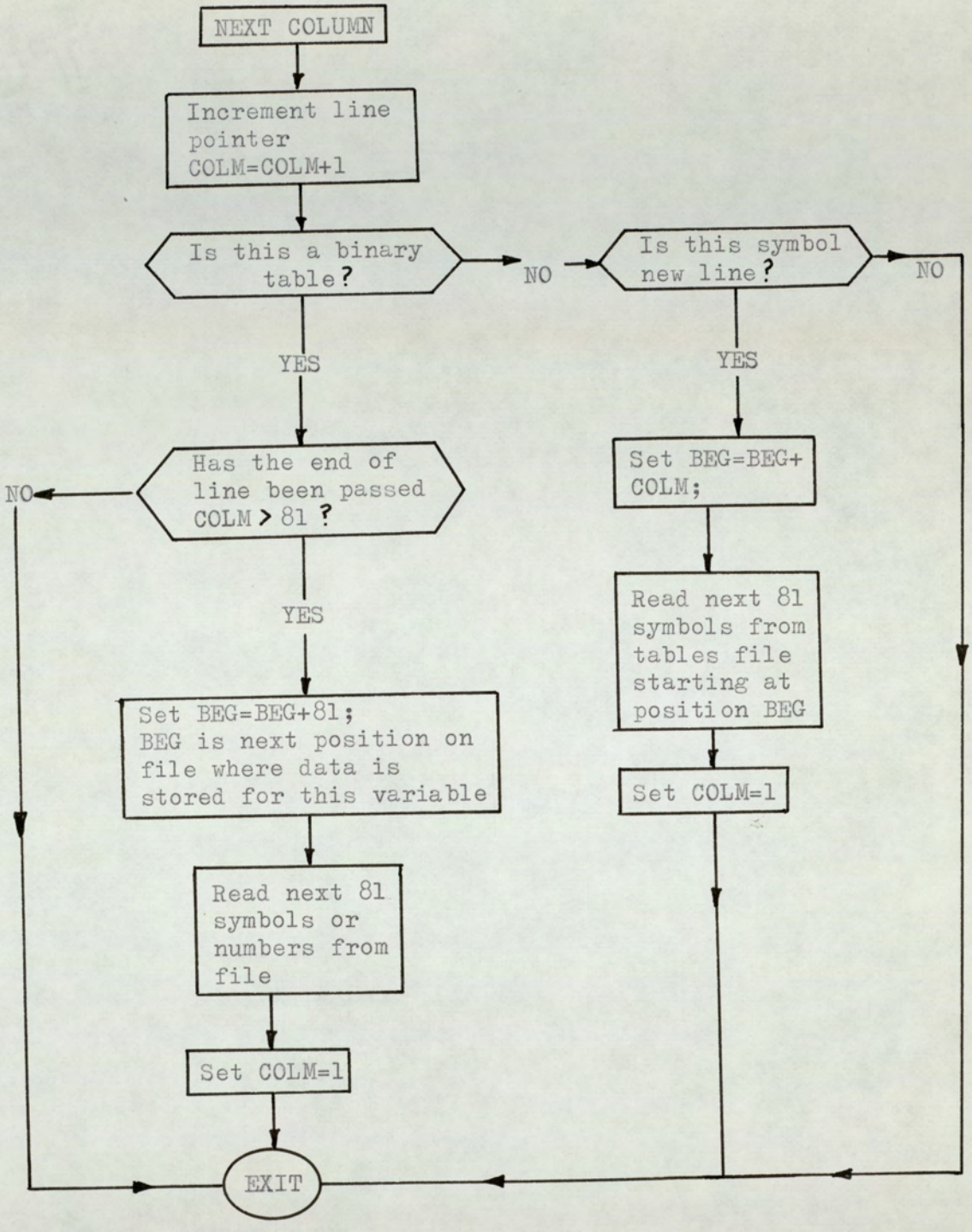
The code assembled on file 4 contains the declarations of single variables and general code used by the assembled program when executed, together with the declarations of the data reading procedures used by

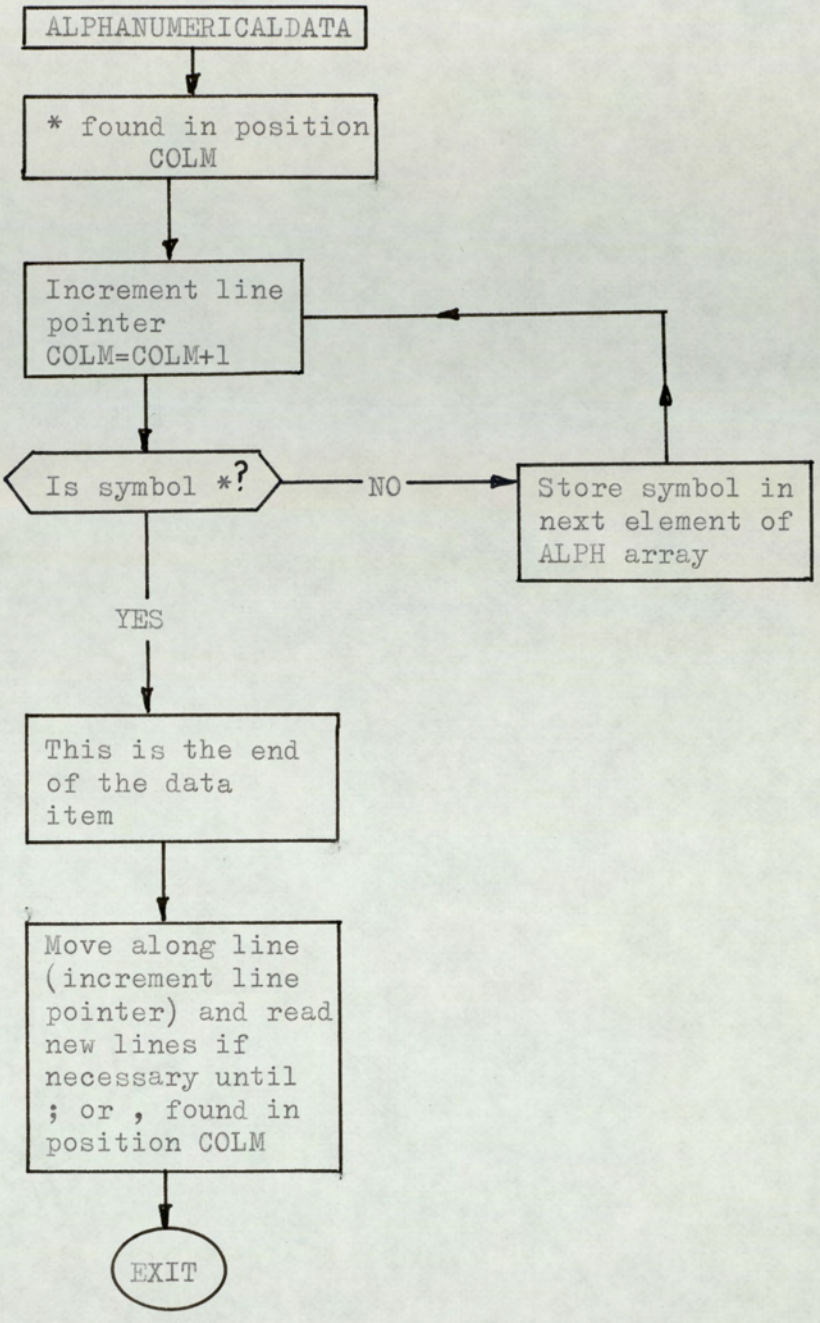
every program assembled by Hydro. After this on file 4 are the data assignments and declarations given by the user under the MAXVALS and DECLARATIONS procedures.

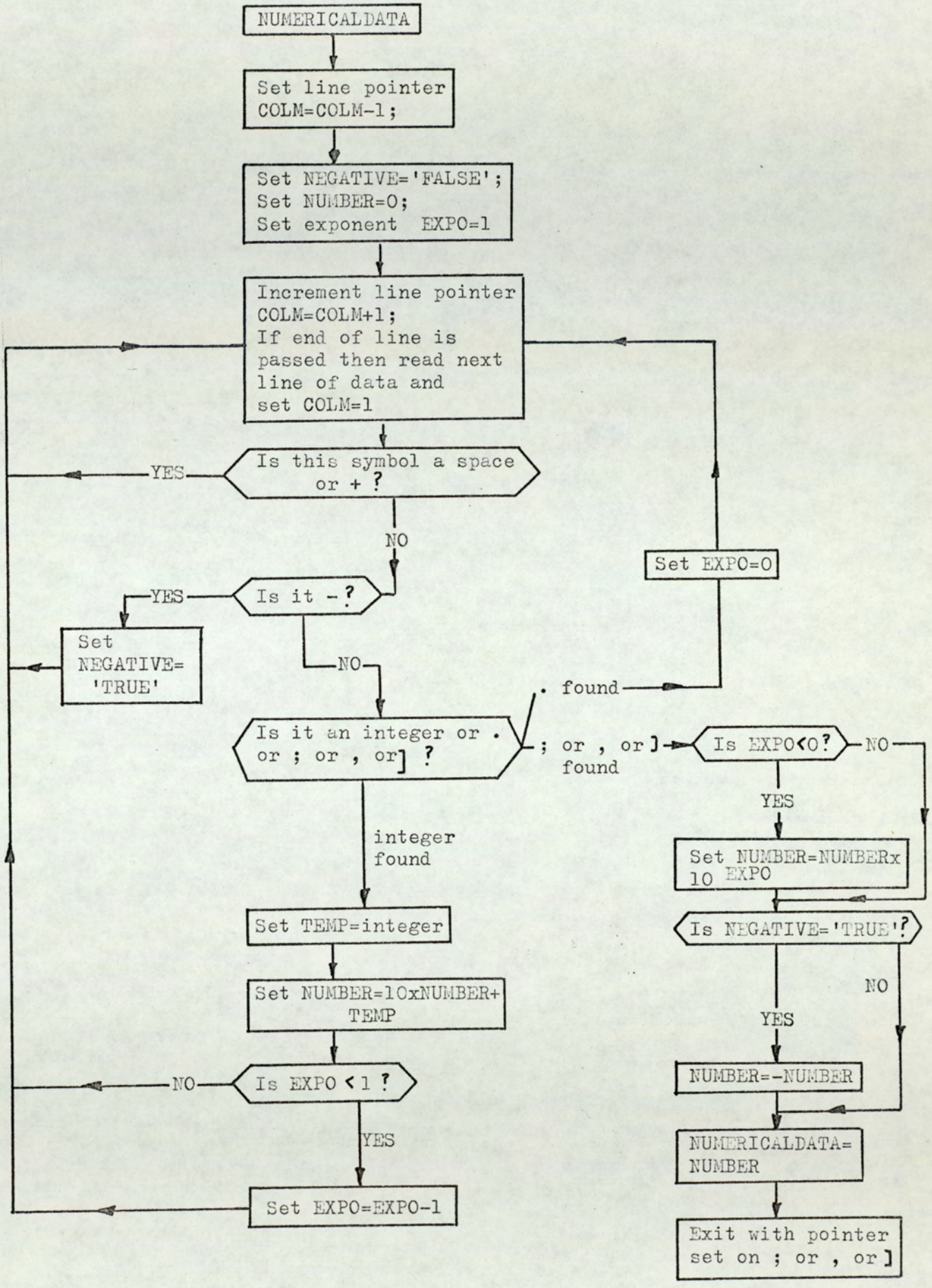
Next are the declarations of all the Hydro procedures which will be used, followed by the declarations of the array variables.

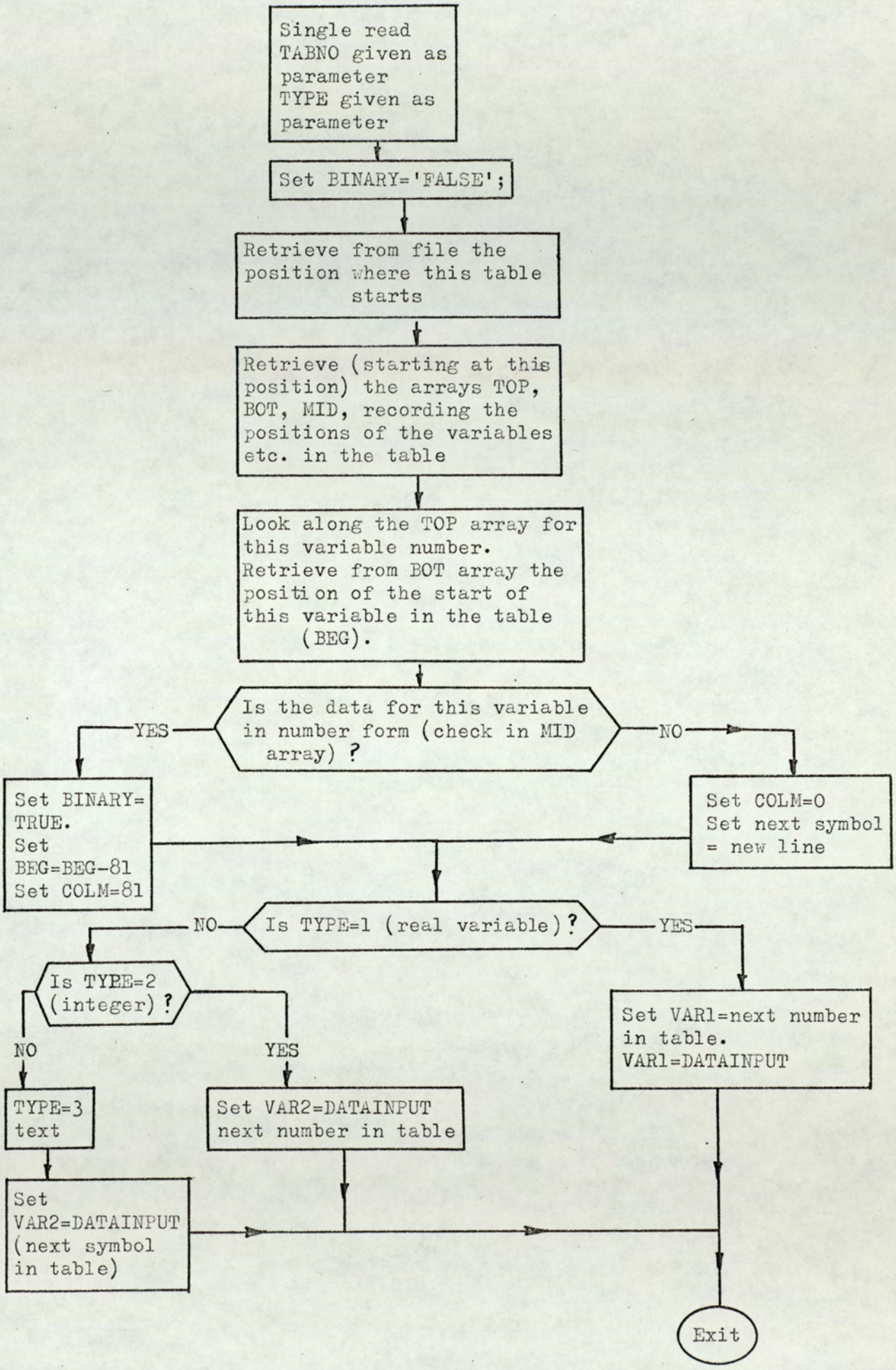
The code assembled on file 2 contains only pure Algol code, the calls on the Hydro procedures, and the calls on the data reading procedures, ARRAYREAD and SINGLEREAD. This code is now copied from file 2 to file 4 after the declarations. The only remaining operation is to write several Algol 'END' lines at the end of the program. After this the complete assembled program on file 4 may be presented to the Algol compiler for translation into machine code and for subsequent execution. By means of the data reading procedure calls the program automatically locates the necessary numerical and text information it requires from the tables file.

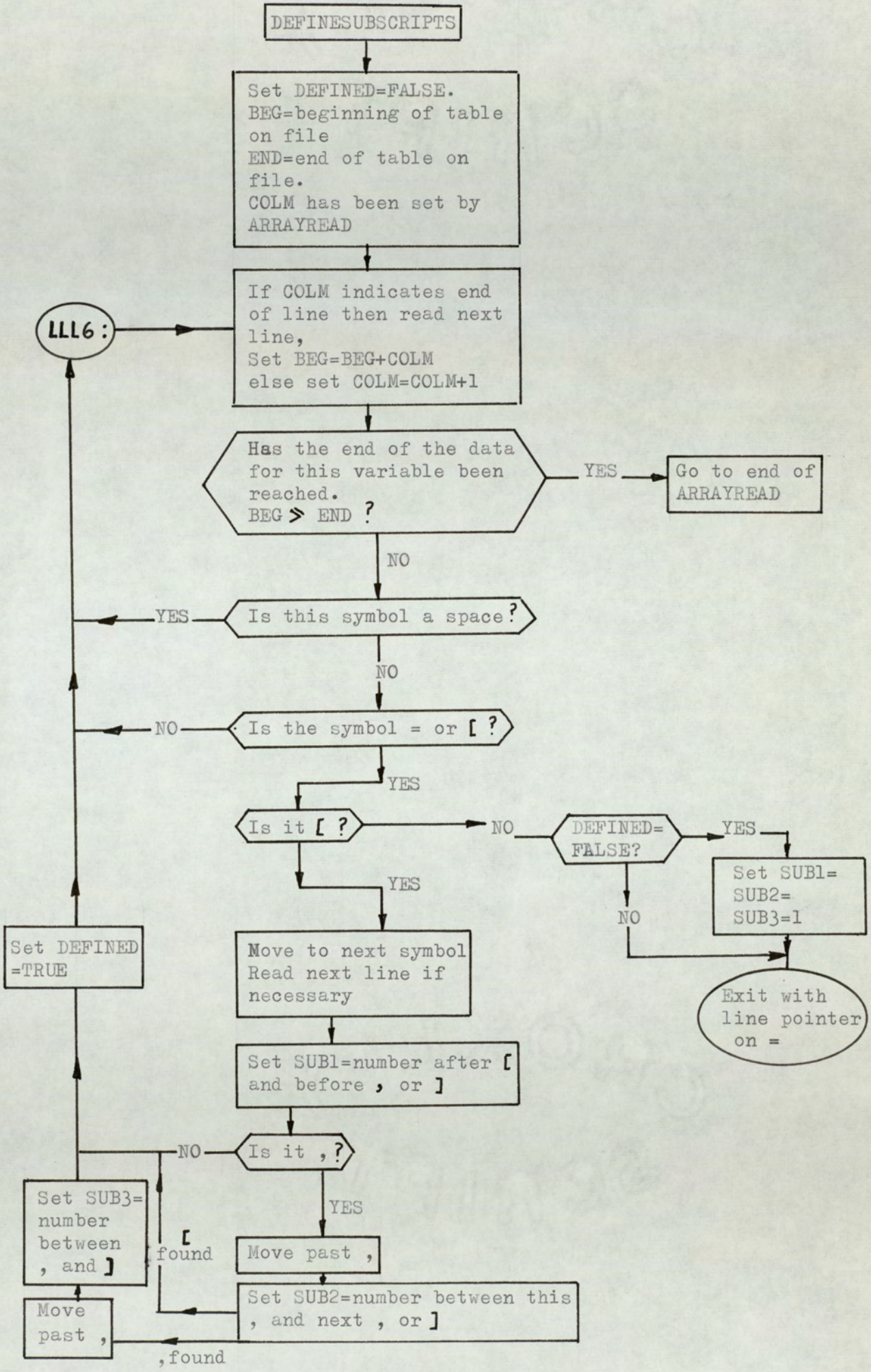
5.8. Flow Diagrams for the Hydro Reading Routines

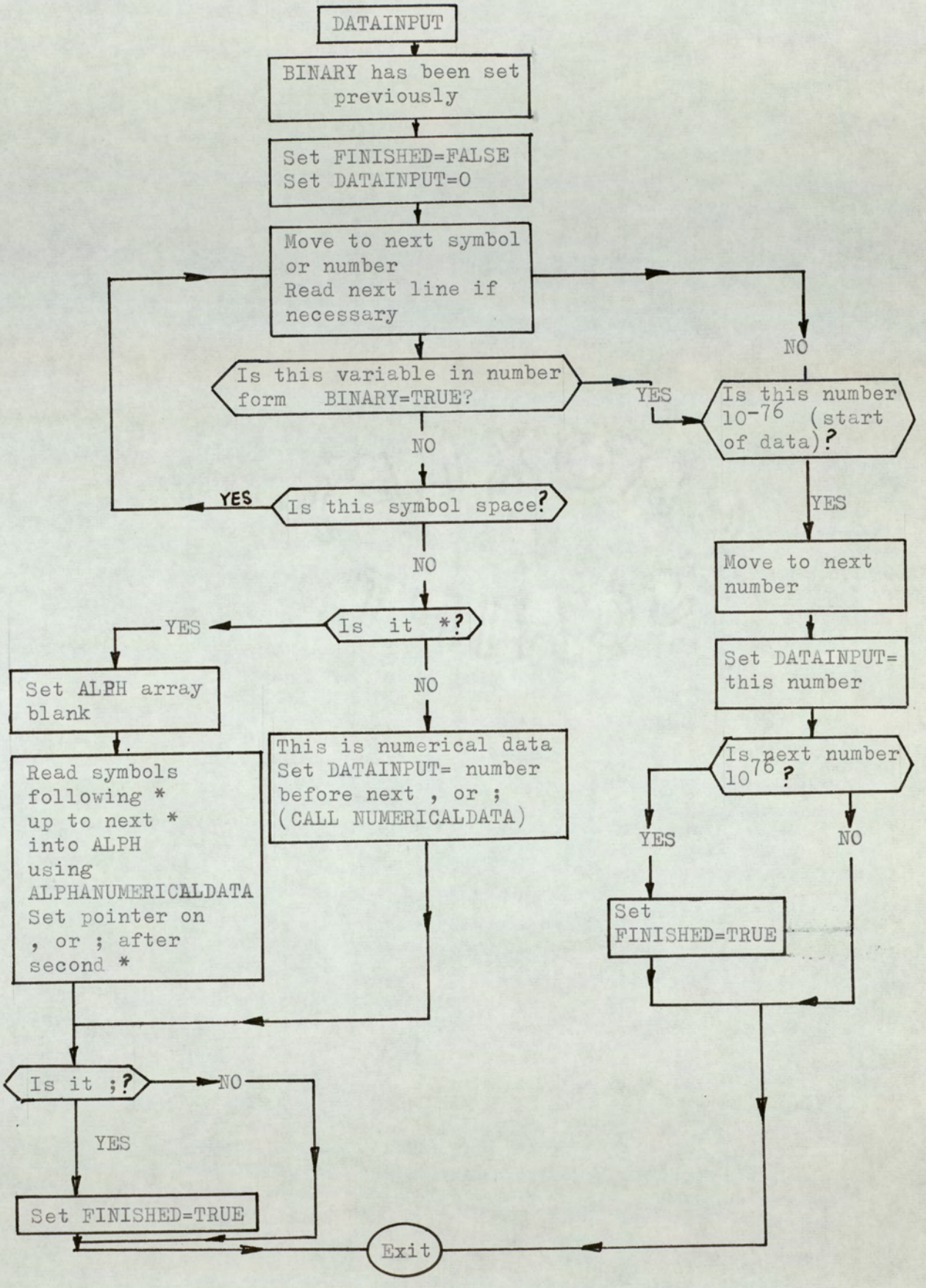


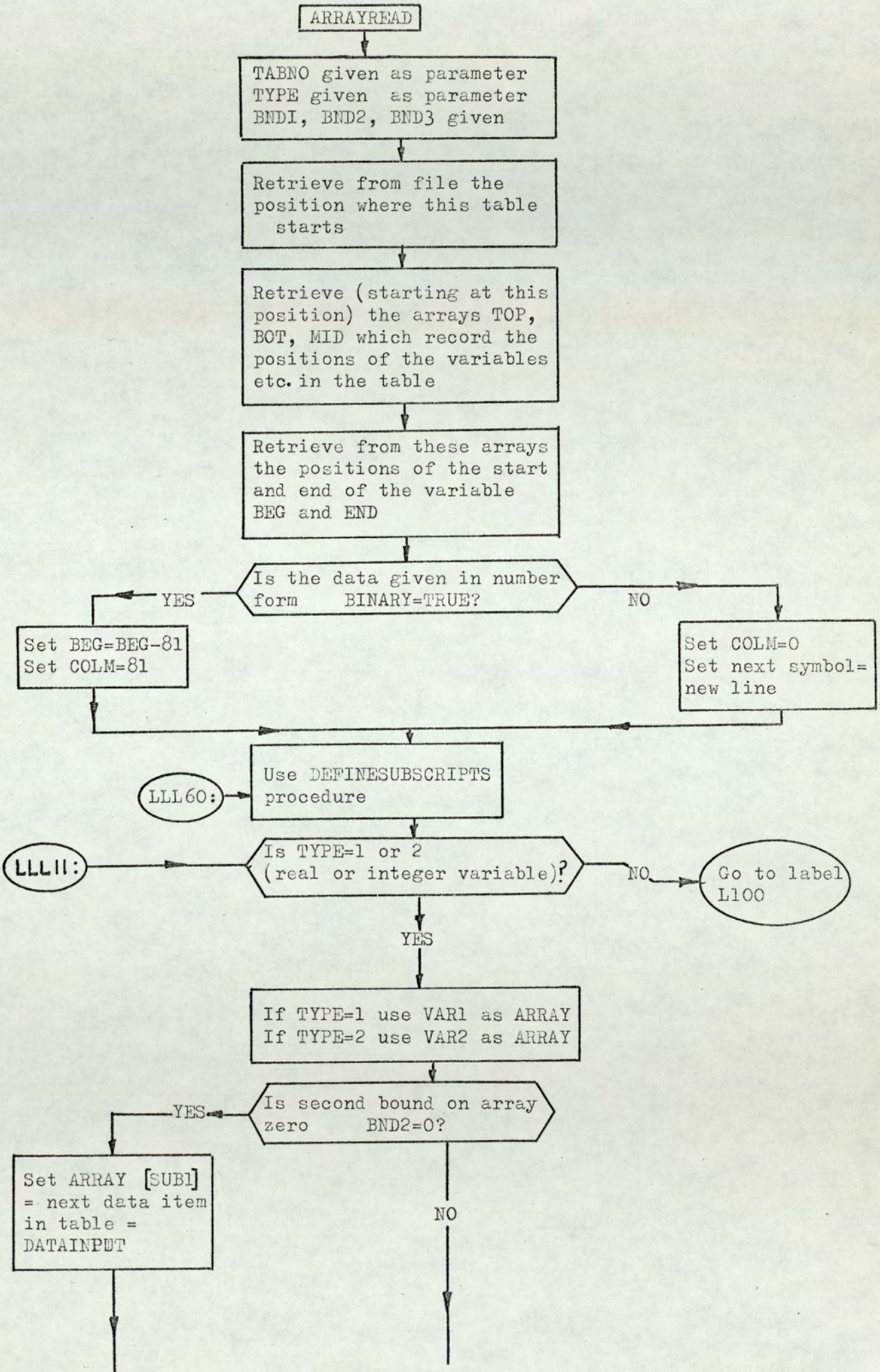

ALPHANUMERICALDATA
* found in position COLM
Increment line pointer COLM=COLM+1
Is symbol *?
NO
Store symbol in next element of ALPH array
YES
This is the end of the data item
Move along line (increment line pointer) and read new lines if necessary until ; or , found in position COLM
EXIT

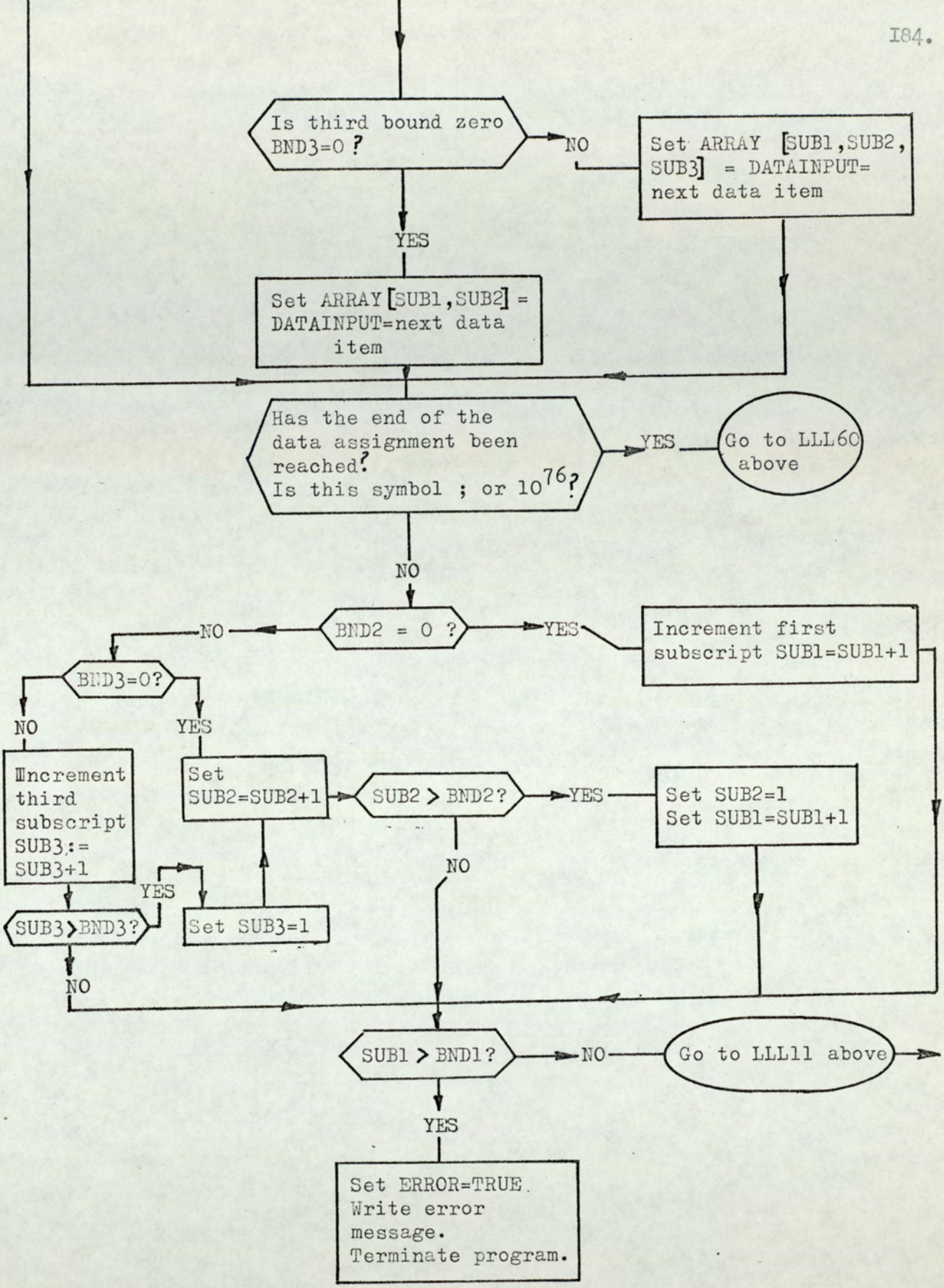

NUMERICALDATA
Set line pointer COLM=COLM-1;
Set NEGATIVE='FALSE';
Set NUMBER=0;
Set exponent EXPO=1
Increment line pointer COLM=COLM+1;
If end of line is passed then read next line of data and set COLM=1
Is this symbol a space or + ?
YES
NO
Is it - ?
YES
Set NEGATIVE= 'TRUE'
NO
Is it an integer or . or ; or , or] ?
. found
; or , or] found
Set EXPO=0
integer found
Set TEMP=integer
Set NUMBER=10xNUMBER+ TEMP
Is EXPO < 1 ?
NO
YES
Set EXPO=EXPO-1
Is EXPO<0?
NO
YES
Set NUMBER=NUMBERx 10 EXPO
Is NEGATIVE='TRUE'?
NO
YES
NUMBER=-NUMBER
NUMERICALDATA= NUMBER
Exit with pointer set on ; or , or]

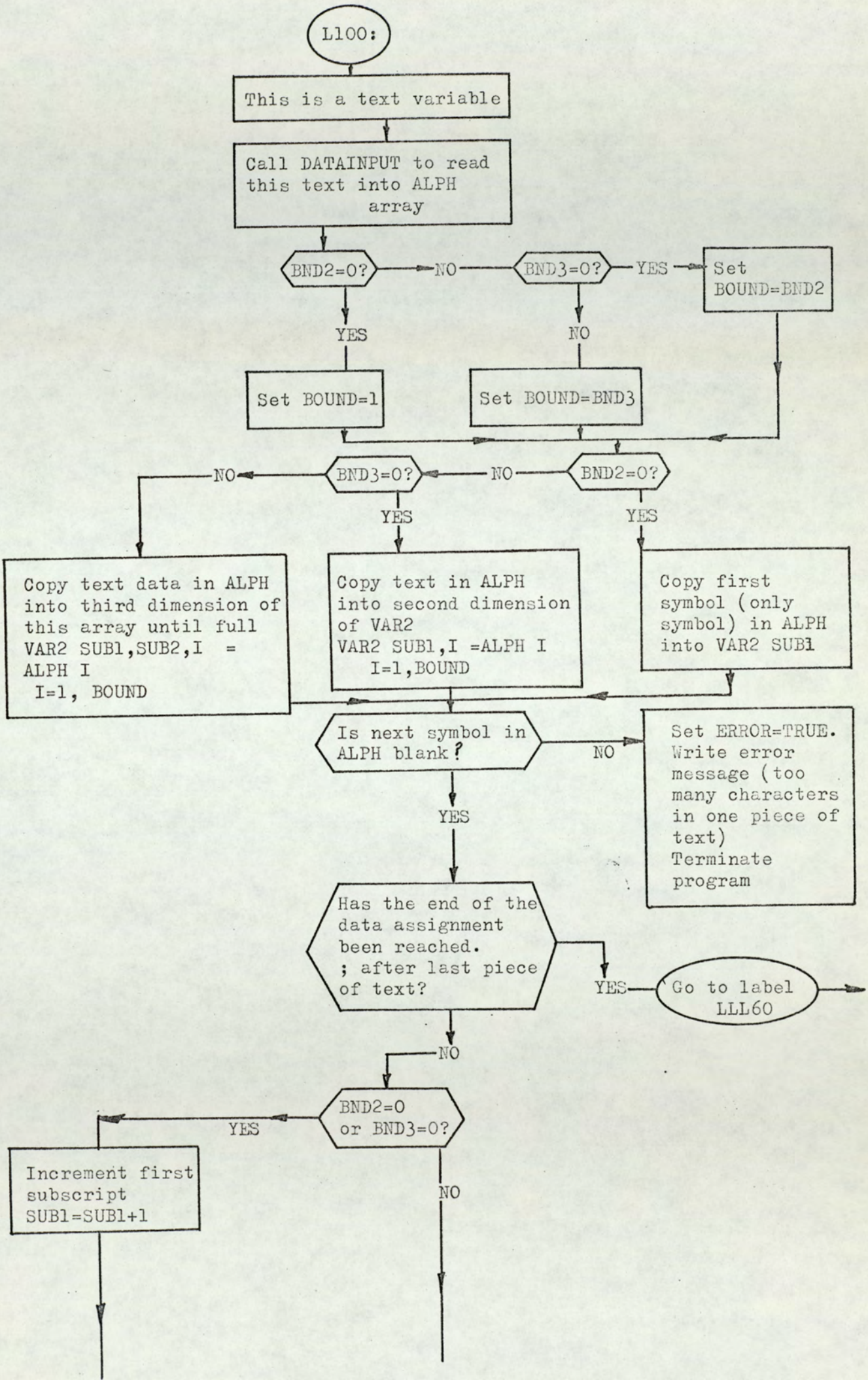

Single read
TABNO given as parameter
TYPE given as parameter
Set BINARY='FALSE';
Retrieve from file the position where this table starts
Retrieve (starting at this position) the arrays TOP, BOT, MID, recording the positions of the variables etc. in the table
Look along the TOP array for this variable number. Retrieve from BOT array the position of the start of this variable in the table (BEG).
Is the data for this variable in number form (check in MID array) ?
YES
NO
Set BINARY= TRUE. Set BEG=BEG-81 Set COLM=81
Set COLM=0 Set next symbol = new line
Is TYPE=1 (real variable) ?
NO
YES
Is TYPE=2 (integer) ?
NO
YES
Set VAR1=next number in table. VAR1=DATAINPUT
TYPE=3 text
Set VAR2=DATAINPUT next number in table
Set VAR2=DATAINPUT (next symbol in table)
Exit

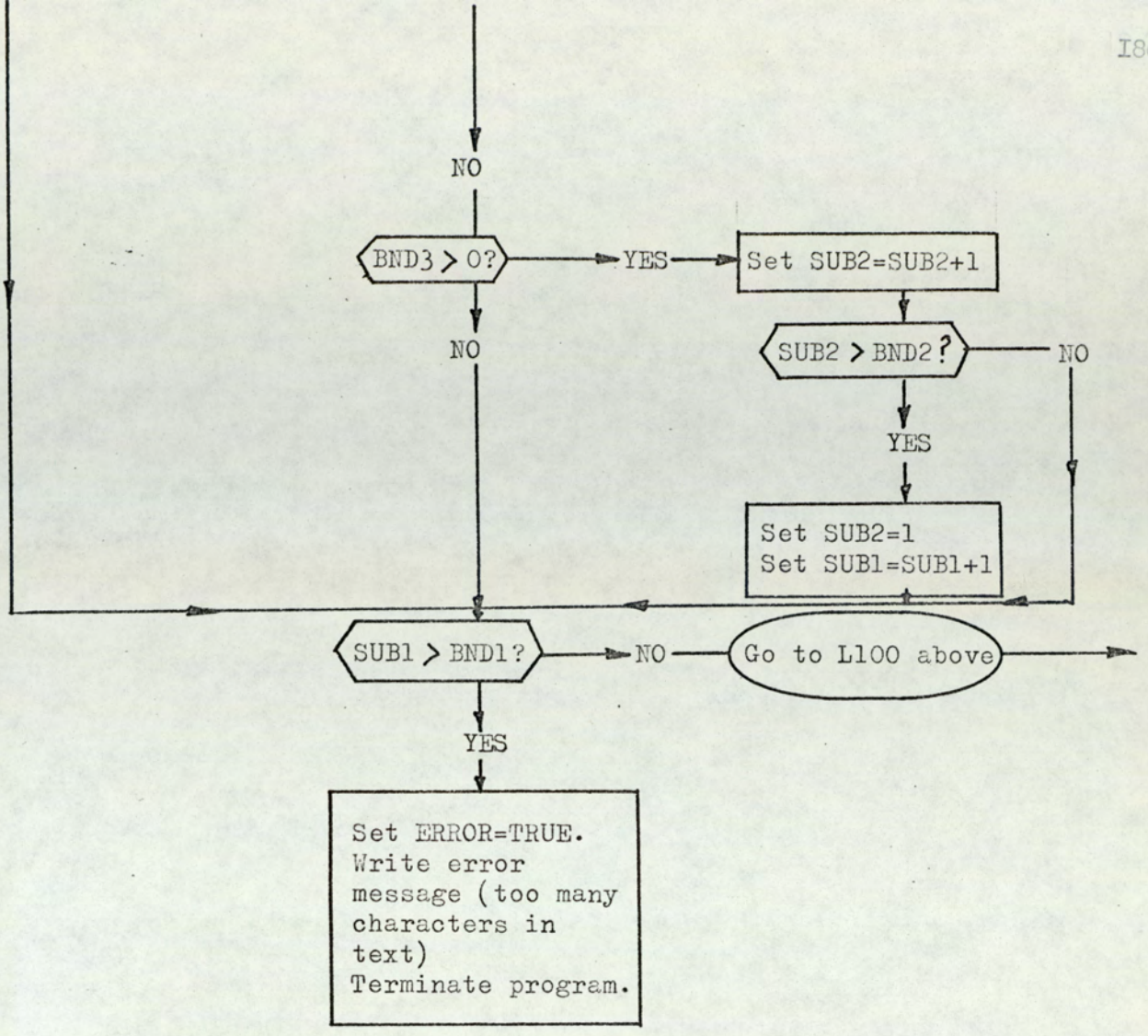

DEFINESUBSCRIPTS
Set DEFINED=FALSE.
BEG=beginning of table on file
END=end of table on file.
COLM has been set by ARRAYREAD
LLL6:
If COLM indicates end of line then read next line,
Set BEG=BEG+COLM
else set COLM=COLM+1
Has the end of the data for this variable been reached.
BEG ≫ END ?
YES
Go to end of ARRAYREAD
NO
YES
Is this symbol a space?
NO
NO
Is the symbol = or [?
YES
Is it [?
NO
DEFINED= FALSE?
YES
Set SUB1= SUB2= SUB3=1
NO
YES
Set DEFINED =TRUE
Move to next symbol
Read next line if necessary
Exit with line pointer on =
Set SUB1=number after [and before , or]
NO
Is it , ?
YES
Set SUB3= number between , and]
[found
Move past ,
Set SUB2=number between this , and next , or]
Move past ,
, found


DATAINPUT
BINARY has been set previously
Set FINISHED=FALSE
Set DATAINPUT=0
Move to next symbol or number
Read next line if necessary
Is this variable in number form BINARY=TRUE?
YES
NO
Is this number 10^{-76} (start of data)?
YES
NO
Is this symbol space?
YES
NO
Move to next number
Set DATAINPUT= this number
Is next number 10^{76}?
YES
NO
Set FINISHED=TRUE
Is it *?
YES
NO
Set ALPH array blank
Read symbols following * up to next * into ALPH using ALPHANUMERICALDATA Set pointer on , or ; after second *
This is numerical data Set DATAINPUT= number before next , or ; (CALL NUMERICALDATA)
Is it ;?
NO
YES
Set FINISHED=TRUE
Exit


ARRAYREAD
TABNO given as parameter
TYPE given as parameter
BND1, BND2, BND3 given
Retrieve from file the position where this table starts
Retrieve (starting at this position) the arrays TOP, BOT, MID which record the positions of the variables etc. in the table
Retrieve from these arrays the positions of the start and end of the variable BEG and END
Is the data given in number form BINARY=TRUE?
YES
NO
Set BEG=BEG-81
Set COLM=81
Set COLM=0
Set next symbol= new line
LLL60:
Use DEFINESUBSCRIPTS procedure
LLL11:
Is TYPE=1 or 2 (real or integer variable)?
NO
Go to label L100
YES
If TYPE=1 use VAR1 as ARRAY
If TYPE=2 use VAR2 as ARRAY
Is second bound on array zero BND2=0?
YES
NO
Set ARRAY [SUB1] = next data item in table = DATAINPUT


Is third bound zero BND3=0 ?
NO
Set ARRAY [SUB1,SUB2, SUB3] = DATAINPUT= next data item
YES
Set ARRAY [SUB1,SUB2] = DATAINPUT=next data item
Has the end of the data assignment been reached? Is this symbol ; or 10^{76}?
YES
Go to LLL60 above
NO
BND2 = 0 ?
NO
YES
Increment first subscript SUB1=SUB1+1
BND3=0?
NO
YES
Increment third subscript SUB3:= SUB3+1
Set SUB2=SUB2+1
SUB2 > BND2?
YES
Set SUB2=1 Set SUB1=SUB1+1
NO
YES
SUB3>BND3?
Set SUB3=1
NO
SUB1 > BND1?
NO
Go to LLL11 above
YES
Set ERROR=TRUE. Write error message. Terminate program.


L100:
This is a text variable
Call DATAINPUT to read this text into ALPH array
BND2=0?
NO
BND3=0?
YES
Set BOUND=BND2
YES
NO
Set BOUND=1
Set BOUND=BND3
BND3=0?
NO
BND2=0?
NO
YES
YES
Copy text data in ALPH into third dimension of this array until full VAR2 SUB1,SUB2,I = ALPH I I=1, BOUND
Copy text in ALPH into second dimension of VAR2 VAR2 SUB1,I =ALPH I I=1,BOUND
Copy first symbol (only symbol) in ALPH into VAR2 SUB1
Is next symbol in ALPH blank?
NO
Set ERROR=TRUE. Write error message (too many characters in one piece of text) Terminate program
YES
Has the end of the data assignment been reached. ; after last piece of text?
YES
Go to label LLL60
NO
BND2=0 or BND3=0?
YES
Increment first subscript SUB1=SUB1+1
NO


NO
BND3 > 0?
YES
Set SUB2=SUB2+1
NO
SUB2 > BND2?
NO
YES
Set SUB2=1
Set SUB1=SUB1+1
SUB1 > BND1?
NO
Go to L100 above
YES
Set ERROR=TRUE.
Write error message (too many characters in text)
Terminate program.

5.9. Description of the flow diagram for the reading routines

The first reading routine is the NEXTCOLUMN procedure. This routine is able to read into an array called CHAR, lines of data from a table. It is assumed that there are two types of information in a table. One type is in character form and the other is in binary form. If the routine is reading character information then one line of data of variable length is read into the CHAR array. The line is terminated by a newline symbol. However, if binary information is being read 81 elements are read from the tables file. In this case each element contains one complete real or integer number, whereas for character information each element contains only one digit of a number, a decimal point, a sign, a comma, semicolon or a character of the variable name. The routine not only reads in the information but also increments a pointer on the CHAR array. When a line of data is read in the pointer is set to the first element of the CHAR array. Later calls on the procedure move the pointer one element along the array. If the pointer is moved past the end of a line for character information or past the 81st number for binary information then more data is read and the pointer is set at the beginning of the data. No analysis of the data is carried out by this procedure.

The ALPHANUMERICALDATAis called when an asterisk is found in the CHAR array. This indicates that the information is text which is to be used to identify results sent to the lineprinter. Each mnemonic or word is given between asterisk symbols. The routine copies the text between the asterisk found and the next asterisk to an array called ALPH. The pointer on the CHAR array is then moved past the text until a semicolon or a comma is found. The pointer is then left on this symbol and an exit is made from the procedure.

The NUMERICALDATA procedure is used to construct a complete number from the digits supplied as character information by the user. On entry to the procedure the pointer on the CHAR array has been set by another procedure or by earlier use of this procedure to the beginning of a number in character form. The pointer could be set to a space preceding the number, a plus or minus sign or the first digit. The routine first moves the pointer along the CHAR array to the first non-space character, if necessary. The first check made is for the presence of a sign. If a sign is present and it is negative then the variable NEGATIVE is set 'TRUE'. If no sign is present the routine reads the following number up to the terminating symbol which can be a comma, semicolon or bracket, by the method detailedin the flow diagram. The final assembled number is set into NUMERICALDATA and an exit is made from the routine.

The SINGLEREAD procedure uses the NEXTCOLUMN procedure to read in a line of dataor 81 symbols, depending upon the type of information from the table specified as the last parameter to the procedure. The procedure first retrieves from the list of table positions the position of the start of this table on file. Having determined this position the TOP, BOT and MID arrays are read from the head of the table. The location in the table of the start of the particular variable to be read is found by looking along the TOP array for the variable number. When this is found, the position of the start of this variable is determined by looking in the equivalent element of the BOT array, and the type of data given is determined from the MID array. If the data is in binary form then BINARY is set 'TRUE'.
In either case, the procedure then checks whether the variable is a realinteger or text variable. This information is supplied automatically in the procedure call for that particular variable.

Data is then read into the appropriate variable by means of the DATAINPUT procedure and an exit is made from SINGLEREAD.

The DEFINESUBSCRIPTS procedure is used when an array variable is given in a table. The user may have given an element of the array as the starting position for filling the rest of the array with the data provided. The subscripts for this element will have been supplied, separated by commas, within square brackets immediately following the array name. This procedure is called from the ARRAYREAD procedure when the array has been located in a table. When DEFINESUBSCRIPTS is entered, the boolean variable DEFINED is set to 'FALSE' and the variables BEG and END will have been set to the positions on file where data begins and ends for this array in the table. This information is retrieved from the directory arrays for this table by ARRAYREAD. If DEFINESUBSCRIPTS has not been called before by the ARRAYREAD procedure then the line pointer on the CHAR array will have been set to a newline symbol. The first action of DEFINESUBSCRIPTS is to check whether the pointer indicates a newline symbol. If it does, then the next line of data for the array is read into CHAR and the pointer is set to the first symbol on the line, but if not then the pointer is moved to the next symbol on the line being processed. The next operation checks whether the data for this array has been exhausted. Every time a new line of data is read the procedure adds the number of symbols on the line to the variable BEG. BEG is then compared to END. If BEG is greater than END then a jump is made to the end of the ARRAYREAD procedure. If the data has not been exhausted then DEFINESUBSCRIPTS begins to process the subscripts given. The line pointer is moved across the line of input in the CHAR array until the first non-space character is found. If the symbol is = then no subscripts have been supplied so that they are all assumed to be equal to one, and an exit

is made from the procedure with the pointer set on =.
If the symbol found was [then subscripts have been given and the NUMERICALDATA procedure is called to read into SUB1 the number between [and the next , or] symbols. If a comma is found after the first subscript then more subscripts are given and a similar procedure is carried out to find the value of SUB2, but if a] symbol was found only one subscript was given and an exit may be made from the procedure. In the same way a check is made for the presence of a third subscript. When all given subscripts have been found an exit is made from the procedure.

The DATAINPUT procedure determines the type of data given for a variable and, depending upon this information, calls the relevant ALPHANUMERICALDATA or NUMERICALDATA procedures to assemble a data item for the variable. At entry to the procedure, FINISHED is set to 'FALSE' and the real variable DATAINPUT is set to zero. A check is first made to find whether the data is in character or binary form. If it is binary information then a check is made to locate the start of the data. Any information which a previous program has written in binary form will contain mixed data. An identifying variable name will have been written first in character form. Then a very small number, 10^{-76} is written in binary form after the variable name and this indicates that the real data for the variable immediately follows. The data is terminated by the large number 10^{+76}. Therefore, if binary data is present the CHAR array is checked for the number 10^{-76}. When this is located, DATAINPUT is set to the next number, which is in binary form, on the line. A further check is now made for the value of the following number on the line. If it is 10^{+76} then FINISHED is set 'TRUE'. In either case, an exit is then made from the procedure.

If the data on file was found to be in character form then a more complex procedure is carried out. The line pointer is moved along the line of data until a non-space character is found, newlines being read as necessary. When a symbol is encountered the procedure checks whether it is the * symbol. If it is, then the text array ALPH is set to blanks and the ALPHANUMERICALDATA procedure is called to transfer the next piece of text on the line to the ALPH array. The line pointer is set on the comma or semi-colon following the data item. If a semicolon is present FINISHED is set to 'TRUE', and in both instances an exit is made from the procedure.

If an asterisk is not found then the data is numerical and the NUMERICALDATA procedure is called to assemble a number from the digits given and to assign this value to the DATAINPUT variable.

The line pointer is set on the comma or semicolon following the data item and if it is a semicolon FINISHED is set 'TRUE'. An exit is then made from the procedure.

The ARRAYREAD procedure is the most complex data reading procedure and is best described in general terms. ARRAYREAD is supplied with several parameters which include the table number where the data is to be found, the data type of the array, and the upper bounds on the array subscripts. The first check made is to determine whether the data is given in character or binary form in the table. The DEFINESUBSCRIPTS procedure is then called to find from the user's input the element from which the array will be filled. The next test made is for the type of the array variable in question. If it is a real or integer variable then similar paths are taken but if it is a text variable then slightly different code must be employed. If real data is given then the first procedure parameter is used as the array name for assignment of data. Otherwise, the second parameter is used as the array name. A process is then

followed which reads data, using DATAINPUT, into the array, starting at the subscripts found by DEFINESUBSCRIPTS. These subscripts are then incremented and data again read in until either the data is exhausted, marked by a semi-colon, or until the subscripts reach their upper bound values. If a semi-colon is found and the end of the data for this variable has not been reached on the file, as determined by the value of the END variable, then DEFINESUBSCRIPTS is used again and the whole procedure is repeated. When the end of data is finally found an exit is made from the procedure.

CHAPTER 6

DETERMINISTIC DYNAMIC PROGRAMMING APPLIED TO RESERVOIR CONTROL

6.1. Introduction

Deterministic dynamic programming deals with the case where the effective control of a system depends only on parameters with known values, and can be applied only to systems where a direct simulation with the same data can be made. In the reservoirs control problem the fixed quantities are the inflows to the reservoirs at any time, which may be obtained from the historical record or be synthetically generated in some way, and the demands on the system.

A system in which inflows or demands can only be given as probability distributions cannot be solved by deterministic dynamic programming. The method of solution where uncertainty is introduced into the system is known as stochastic dynamic programming, which is discussed in the next chapter.

When a data sequence and the sizes of the system modules have been specified, the possible release or control decisions which can be made for each reservoir level or combination of reservoir levels, or other system 'state', are decided upon by the designer.

Deterministic dynamic programming can then be applied to find the optimal decision for any 'state' and time, or stage, if the inflows and demands are routed through the system.

6.2. The Dynamic Programming Principle

It is not always evident that the operation of a reservoir or water resources complex can be taken as the set of non linear, time-varying differential equations

$$\dot{\bar{x}} = \bar{f}(\bar{x},\bar{u},t) \quad \ldots . \quad \text{equ. 6.I}$$

where $\bar{x}$ = n - dimensional state vector

$\bar{u}$ = m - dimensional control vector

t = time variable measuring time elapsed starting at an arbitrary datum

$$\dot{(\ )} = \frac{d}{dt}(\ ).$$

This formulation tends to loosen many preconceived notions that one might have about the reservoirs control problem. It is easier to see analagous problems in different fields and possibly make use of advances in these areas. It also demonstrates more clearly the consequences of discretisation of the problems in the more familiar finite differences notation. For these reasons, the dynamic programming principle is introduced along these lines.

The state vector is a record of every relevant piece of information which might affect the decision to be made at a given time concerning the operation of the system. In a simple case, this vector might only contain the two elements which give the levels in each reservoir for a two reservoir system. A more sophisticated approach might include elements which describe whether the previous inflow to each reservoir was higher or lower than average for a particular time, thus allowing for serial correlation of inflows.

The state vector at time t is represented by

$$\bar{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix} \quad \text{where } x_J(t) \text{ is one known feature at time t}$$

The control vector contains any number of elements, each of which describes one operation on a given partof the system

In the caseof a two reservoir system supplying one demand then onlyone operation may be required for each reservoir i.e. the release from each, but if two demands are to be satisfied, then there may be two operations for each reservoir, namely the releases to be made to each demand.

The control vector at time t is denoted as

$$\bar{u}(t) = \begin{bmatrix} u_I(t) \\ u_J(t) \\ u_m(t) \end{bmatrix}$$ where $u_J(t)$ is one control variable

For the two reservoir case and a single demand this might be

$$\bar{u}(t) = \begin{bmatrix} u_I(t) \\ u_2(t) \end{bmatrix}$$ where $u_I(t)$ is the release from reservoir I at time t and $u_2(t)$ is the release from reservoir 2 at time t

For the two demands case the control vector might be

$$\bar{u}(t) = \begin{bmatrix} u_I(t) \\ u_2(t) \\ u_3(t) \\ u_4(t) \end{bmatrix}$$

where $u_I(t)$ is the release from reservoir I to demand I at time t

$u_2(t)$ is the release from reservoir I to demand 2 at time t

$u_3(t)$ is the release from reservoir 2 to demand I at time t

$u_4(t)$ is the release from reservoir 2 to demand 2 at time t

The system variables and fixed quantities are related by the system equations 6.I. which describe how the system changes from one time period to the next when a control is applied.

The system controller can choose values of $\bar{u}$ at any time in such a way that over a long period of time some measure of performance is optimised. This measure takes the form of an integral of a scalar functional of the

state variables, control variables and time plus a scalar functional of the final state and final time :

$$J = \int_{t_o}^{t_f} l\left[\bar{x}(\sigma), \bar{u}(\sigma), \sigma\right] d\sigma + \Psi\left[\bar{x}(t_f), t_f\right] \quad . \quad . \quad . \quad . \quad \text{Equ. 6.2.}$$

where t_o = initial time

t_f = final time

σ = dummy variable for time

l = scalar functional for cost per unit time

Ψ = scaler functional for final value cost

In order to solve the set of equations 6.I. bearing in mind the objectives contained in equation 6.2., it is necessary to use finite difference approximations. Equation 6.I. can be written as :

$$\bar{x}(t+\delta t) = \bar{x}(t) + \bar{f}(\bar{x}(t), \bar{u}(t), t)\delta t \quad . \quad . \quad . \quad . \quad \text{Equ. 6.3.}$$

The integration of equation 6.2. may be simplified by the expression

$$\int_{t}^{t+\delta t} l\left[\bar{x}(\sigma), \bar{u}(\sigma), \sigma\right] d\sigma = l\left[\bar{x}(t), \bar{u}(t), t\right] \delta t \quad . \quad . \quad . \quad . \quad \text{Equ. 6.4.}$$

These are the simplest possible finite difference forms that can be written for equations 6.I. and 6.2. When integrating difference equations by numerical techniques many problems arise over the choice of step size. Frequently they are resolved by solving the equations at different spacings and comparing the results.

There may be other considerations to take into account in the solution but these can generally be written in the form of constraints on the possible states and policies :

$$\bar{x}(t) \in \bar{X}(t)$$

$$\bar{u}(t) \in \bar{U}(\bar{x}, t)$$

Where $\bar{X}$, the set of admissible states, can vary with t, and where $\bar{U}$, the set of admissible controls, can vary with $\bar{x}$ and t.

It is possible to solve the problem by an application of Bellman's

principle of optimality, which states that if we know an optimal trajectory, or sequence of states, from a state $\bar{x}(t_o)$ to a state $\bar{x}(t_f)$ then the portion of the trajectory from any intermediate state $\bar{x}(t)$ on that trajectory to state $\bar{x}(t_f)$ is the optimal trajectory from state $\bar{x}(t)$ to state $\bar{x}(t_f)$.

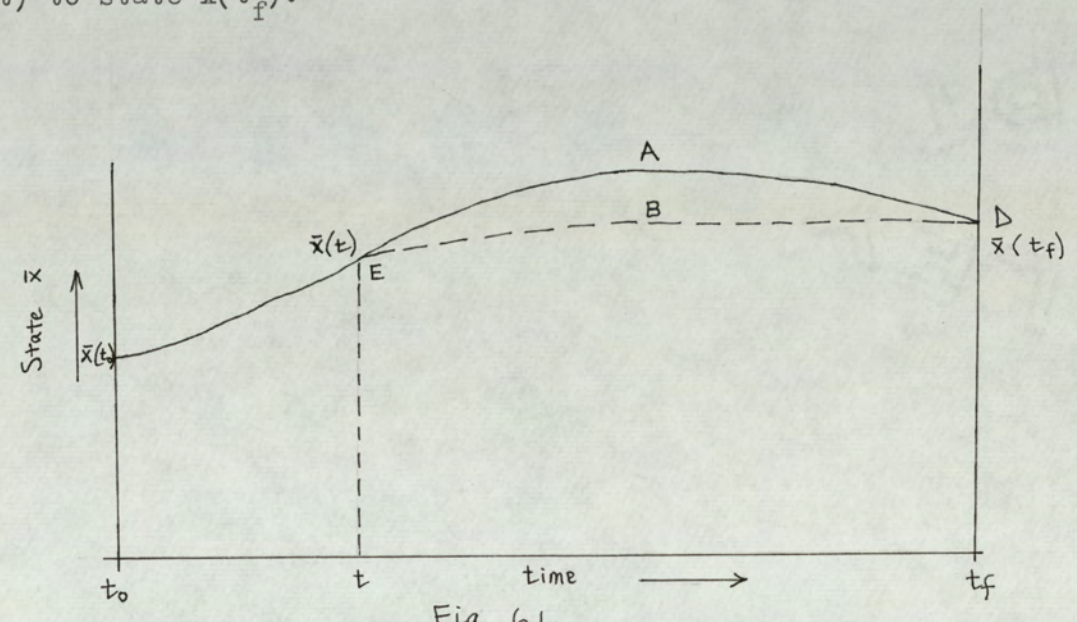


Fig. 6.1

This principle can easily be proved by considering the logic of the situation. Referring to Fig. 6.I., if we know CEAD is the optimal trajectory from $\bar{x}(t_o)$ to $\bar{x}(t_f)$ then let us assume that EBD is the optimal trajectory from $\bar{x}(t)$ to $\bar{x}(t_f)$. Then path CEBD must have cost less than path CEAD, but this contradicts the fact that CEAD is the optimal path from $x(t_o)$ to $\bar{x}(t_f)$ and hence EAD must be the best path from $\bar{x}(t)$ to $\bar{x}(t_f)$.

In order to apply the principle to the reservoir operation problem it is necessary to define a minimum cost function $I\left[\bar{x}(t),t\right]$.
This function determines the minimum cost that could be incurred in going to the time t_f if the present time is t and the present state is $\bar{x}$.

The principle of optimality may now be written as

$$I\left[\bar{x}(t),t\right] = \underset{\substack{\bar{u}(\sigma)\in\bar{U} \\ t\leq\sigma\leq t_f}}{\text{Min}} \int_t^{t_f} l\left[\bar{x}(\sigma),\bar{u}(\sigma),\sigma\right] d\sigma \quad + \Psi\left[\bar{x}(t_f),t_f\right] \qquad \text{. . . . Equ. 6.5.}$$

Using the approximations of equations 6.3. and 6.4. the iterative functional equation of dynamic programming may be written as

$$I\left[\bar{x}(t),t\right] = \min_{\bar{u}\in\bar{U}}\left[l\left[\bar{x}(t),\bar{u}(t),t\right]\delta t + I\left[\bar{x}(t)+\bar{f}(\bar{x}(t),\bar{u}(t),t)\delta t, t+\delta t\right]\right] \quad \ldots \text{ Equ. 6.6.}$$

The development of this equation is treated more fully in the next section. The interpretation of this equation is that the minimum cost at a given state $\bar{x}$ and the present time t is found by minimising, through the choice of the present control $\bar{u}(t)$, the sum of $l\left[\bar{x}(t),\bar{u}(t),t\right]\delta t$, the cost over the next time interval δt, plus $I\left[\bar{x}(t)+\bar{f}(\bar{x}(t),\bar{u}(t),t)\delta t, t+\delta t\right]$, the minimum cost of going to t_f from the resulting next state, $\bar{x}(t)+\bar{f}(\bar{x}(t),\bar{u}(t),t)\delta t$.

This iterative equation is solved backwards in time because $I\left[\bar{x}(t),t\right]$ depends on values of the minimum cost function at future times. Consequently, the iterations begin by specification of the minimum cost function at the final time t_f.

Using equation 6.5. $$I\left[\bar{x}(t_f),t_f\right] = \Psi\left[\bar{x}(t_f),t_f\right] \quad \ldots \text{ Equ. 6.7.}$$

The minimum cost function for all $\bar{x}$ and t can be evaluated by iteratively solving equation 6.6. with equation 6.7. as a boundary condition. The optimal control at every $\bar{x}$ and t, denoted by $\bar{u}_{OPT}(\bar{x},t)$, is obtained as the value of $\bar{u}(t)$ which minimises equation 6.6. for the given $\bar{x}$ and t.

6.3. Discretisation of the variables

Since the levels and releases in a reservoir may take any continuous values within their possible ranges at any time, the dynamic programming method reduces the computational volume by fixing the variables at discrete values and carries out calculations of optimum rules, using equation 6.6., only at these values. The costs, $I\left[\bar{x}(t)+\bar{f}(\bar{x}(t),\bar{u}(t),t)\delta t, t+\delta t\right]$, if the resulting state, $\bar{x}(t)+\bar{f}(\bar{x}(t),\bar{u}(t),t)$, is not a discrete value, are obtained by interpolating between the costs for the nearest discrete states. The time variable is also reduced to discrete values, but the increments in time, δt, between each calculation of costs need not be the same for all t.

The accuracy of the solution of the system equations depends, of course, on the discrete step sizes chosen for state, control and time variables and some attempt has been made in this thesis to investigate the effects of varying step sizes.

In essence, dynamic programming contains the same steps as simulation, except that instead of applying one specified rule to the set of levels existing in the system at any time, dynamic programming applies several rules to a particular set of levels and chooses the one which is the best at that time. No assumption is made about the state of the system in one period of time, the best rule being determined for every state which could occur.

The simulation approach would involve fixing the controls, $\bar{u}$, at each time and running a simulation of the system starting from some given state $\bar{x}(t_o)$. This simulation would produce a cost

$$J_I = \int_{t_o}^{t_f} l\left[\bar{x}(\sigma),\bar{u}(\sigma),\sigma\right] d\sigma + \Psi\left[\bar{x}(t_f),t_f\right]$$

Further simulations can be run with every possible control sequence,

and the costs, J, obtained. The sequence with the minimum cost is the optimal one.

Hill climbing and directed search techniques have been used to modify the sequence of $\bar{u}$'s to find the best sequence .

The number of trajectories possible depends upon the number of controls which may be applied at each time, and upon the number of time increments chosen. For a system where four controls may be applied at one time, and with a twelve months total time in one monthly increments the number of trajectories is 4^{12}, approximately 17 million. So it is clear that, even with this short process, it would be computationally impossible to evaluate all the trajectories. However, the dynamic programming method does in fact perform an equivalent calculation with far less computational effort.

6.4. The discretised formulation of dynamic programming

In the discretised dynamic programming notation, the time variable is called the stage variable (k) and this measures the time elapsed from the beginning of the process. A stage in the process is the period of time over which a control is applied and corresponds to δt in the continuous case. In the conventional dynamic programming procedure δt is normally taken to be constant for all t and becomes Δt. In the reservoirs control problem a fixed control rule may be applied over a stage length of one month. Since the stage variable is discretised it is not necessary to know the absolute value of the time, but only the integer number of stages elapsed from the beginning of the process.

The absolute time is related to the stage variable by the equations

$$t = t_o + k\Delta t$$

and $$K\Delta t = t_f - t_o$$

where t_o = time at which the process begins

t_f = time at which the process ends

Δt = the length of time for one stage

K = the total number of time periods (stages) in the process

t = absolute time

Equation 6.3., the system equations may be written as

$$\bar{x}(k+I) = \bar{x}(k) + \bar{f}(\bar{x}(k),\bar{u}(k),k)\Delta t \quad . \quad . \quad . \quad . \qquad \text{Equ. 6.8.}$$

For convenience we may write

$$\bar{x}(k+I) = \bar{g}\left[\bar{x}(k),\bar{u}(k),k\right] \quad . \quad . \quad . \quad . \qquad \text{Equ. 6.9.}$$

where $\bar{g}\left[\bar{x}(k),\bar{u}(k),k\right] = \bar{x}(k) + \bar{f}(\bar{x}(k),\bar{u}(k),k)\Delta t.$

Expanding equation 6.9. gives

$$x_I(k+I) = g_I\left[x_I(k),x_J(k),\ldots x_n(k),u_I(k),u_J(k),\ldots u_m(k),k\right]$$

$$x_i(k+I) = g_i\left[x_I(k), x_J(k), \ldots x_n(k), u_I(k), u_J(k), \ldots u_m(k), k\right]$$

$$x_n(k+I) = g_n\left[x_I(k), x_J(k), \ldots x_n(k), u_I(k), u_J(k), \ldots u_m(k), k\right]$$

where x_J is a discretised value of any one observable feature of the system

u_J is a discretised value of one operation on one part of the system.

Example I.

For a two reservoir system where the reservoirs are not interconnected, and one demand is to be satisfied :

$x_I(k)$ = level in reservoir I at stage k

$x_2(k)$ = level in reservoir 2 at stage k

$$\bar{x}(k) = \begin{bmatrix} x_I \\ x_2 \end{bmatrix}$$

$u_I(k)$ = release made from res.I at stage k

$u_2(k)$ = release made from res.2 at stage k

$$\bar{u}(k) = \begin{bmatrix} u_I \\ u_2 \end{bmatrix}$$

$$x_I(k+I) = g_I\left[x_I(k), u_I(k), k\right]$$

$$x_2(k+I) = g_2\left[x_2(k), u_2(k), k\right]$$

The functions, g, only contain the variables for one reservoir, since the level at stage (k+I) in one reservoir does not depend upon the level and operation of the other reservoir at stage (k) because they are not interconnected.

The levels at stage (k+I) depend upon the stage variable, k, because both the levels and the controls at any time may be restricted to a particular range of values, which change with time.

For instance, the level in a reservoir may have to be constrained below a flood level which may change for every month, or there might be a minimum acceptable release in any month to provide for compensation water. Obviously, the range of control may also depend on the level itself since it is not possible to release more water than is available and it may not be allowed to release water below a minimum storage level. These restrictions are called the constraints on the problem and are represented by

$$\bar{x}(k) \in \bar{X}(k) \quad . \quad . \quad . \quad . \quad \text{Equ. 6.I0}$$

$$\bar{u}(k) \in \bar{U}(\bar{x},k) \quad . \quad . \quad . \quad . \quad \text{Equ. 6.II}$$

Although it is not generally true, these constraints may be separated for each reservoir or for each level in a reservoir, in which case the constraints become

$$x_I(k) \in \bar{X}_I(k) \qquad u_I(k) \in \bar{U}_I(x_I,k)$$

$$x_2(k) \in \bar{X}_2(k)$$

$$x_n(k) \in \bar{X}_n(k) \qquad u_n(k) \in \bar{U}_n(x_n,k)$$

In order to choose the optimal rule for every possible combination of levels at each time stage, it is necessary to have some kind of performance criterion for the system. In some cases this will be a cost function where a penalty is applied, for example, for not releasing enough water to supply a demand, or a benefit function where a reward is gained for each unit of water released or power generated in a hydro electric scheme. The purpose of dynamic programming is to find the best operating rules at each stage, starting from some given state, so that the sequence of states throughout the process is the best which can be obtained to minimise or maximise the performance criterion.

Therefore, the performance criterion may be written as

$$J = \sum_{k=o}^{K} l\left[\bar{x}(k),\bar{u}(k),k\right] + \Psi\left[\bar{x}(k),k\right] \quad \text{where } l \text{ is any function} \ldots \text{Equ. 6.I2.}$$

and $\bar{u}(k)$ is the control at stage (k).

In the two reservoirs problem the quantity to be minimised may be the deficits to supply, in which case the performance criterion becomes

$$J = \sum_{k=o}^{K} \left[D(k) - u_1(k) - u_2(k) \right] \quad \text{where}$$

$D(k)$ is the demand at stage k

$u_i(k)$ is the release from reservoir i at stage k.

and $\sum_i u_i(k) \leqslant D(k)$

6.5. Derivation of the iterative functional equation for the discrete case

Consider stage K at the end of the process :

The system may be in any state in the range $\bar{X}(K)$.

It is conceivable that a penalty will be attached to being in certain states at the end of the process. For instance, an operator may want the reservoirs to be completely empty at this stage and the penalty to be attached to each state will then be the cost of discharging the amount of water attachedto that state until the reservoir is empty. Other types of penalty may be used but this will depend upon the system. The penalty to be attached to each state will be designated as $\phi\left[\bar{x}(K),K\right]$ which is a special case of the general cost function applied at each stage, $l\left[\bar{x}(k),\bar{u}(k),k\right]$.

Thus, the best state to be in at stage K is the one with the minimum penalty function, although this does not necessarily lie on the complete optimal trajectory.

Now consider stage (K-I) :

Again, the system may be in any state in the allowable range. It is not yet known which of the states will lie on the optimal trajectory from $\bar{x}(o)$ so that it is necessary to calculate the best trajectory to the end of the process from all possible states at this stage.

If the system is in state $\bar{x}_I(K-I)$ then it may make a transition to several states at stage K, depending upon the control vector, $\bar{u}_J$, applied. Let these states be

$$\begin{aligned}
\bar{g}\left[\bar{x}_I(K-I),\bar{u}_I(K-I),(K-I)\right] &= \bar{y}_I(K) \\
\bar{g}\left[\bar{x}_I(K-I),\bar{u}_2(K-I),(K-I)\right] &= \bar{y}_2(K) \qquad \text{. . . .} \quad \text{Equ. 6.13.} \\
\bar{g}\left[\bar{x}_I(K-I),\bar{u}_m(K-I),(K-I)\right] &= \bar{y}_m(K)
\end{aligned}$$

The associated costs involved in the transition will be

$$1\left[\bar{x}_I(K-I),\bar{u}_I(K-I),(K-I)\right] = C_I$$
$$1\left[\bar{x}_I(K-I),\bar{u}_2(K-I),(K-I)\right] = C_2 \quad \ldots\ldots \text{Equ. 6.I4.}$$
$$1\left[\bar{x}_I(K-I),\bar{u}_m(K-I),(K-I)\right] = C_m$$

Now, in order to compute the least cost trajectory of going from stage $\bar{x}_I(K-I)$ to the end of the process, the costs of the transitionsto time k are added to thepenalty functions at time K.

i.e. Under decision $\bar{u}_I$the total cost is

$$C_I+\phi\left[\bar{y}_I(K),K\right] = TC_I\left[\bar{x}_I(K-I)\right] \quad \ldots\ldots \text{Equ. 6.I5.}$$

Under decision $\bar{u}_2$ the total cost is

$$C_2+\phi\left[\bar{y}_2(K),K\right] = TC_2\left[\bar{x}_I(K-I)\right] \quad \ldots\ldots \text{Equ. 6.I6.}$$

These total costs are computed for every decision at time (K-I) and the minimum one is chosen as the best trajectory from $\bar{x}_I(K-I)$ to the end of the process. The costof this trajectory is $TC_{OPT}\left[\bar{x}_I(K-I)\right]$.

The above calculation is repeated for every possible discrete state $\bar{x}(K-I)$, so that we obtain the minimum cost trajectory from all $\bar{x}(K-I)$ to the end of the process.

The calculation then proceeds to stage (K-2) where identical computations to those carried out at stage (K-I) are performed.

If the system is in state $\bar{x}_I(K-2)$ then it may make transitions under different decisions to the states

$$\bar{g}\left[\bar{x}_I(K-2),\bar{u}_I(K-2),(K-2)\right] = \bar{y}_I(K-I)$$
$$\bar{g}\left[\bar{x}_I(K-2),\bar{u}_2(K-2),(K-2)\right] = \bar{y}_2(K-I) \quad \ldots\ldots \text{Equ. 6.I7.}$$
$$\bar{g}\left[\bar{x}_I(K-2),\bar{u}_m(K-2),(K-2)\right] = \bar{y}_m(K-I)$$

The immediate costs of the transitions will be

$$\begin{aligned} 1\left[\bar{x}_I(K-2),\bar{u}_I(K-2),(K-2)\right] &= D_I \\ 1\left[\bar{x}_I(K-2),\bar{u}_2(K-2),(K-2)\right] &= D_2 \\ &\vdots \\ 1\left[\bar{x}_I(K-2),\bar{u}_m(K-2),(K-2)\right] &= D_m \end{aligned} \qquad \text{Equ. 6.I8.}$$

From the previous stage, we know the optimum costs of going from any state at (K-I) to the end of the process, i.e. the $TC_{OPT}\left[\bar{x}(K-I)\right]$, so that the total cost of going from state $\bar{x}_I(K-2)$ to the end is the minimum of

$$\begin{aligned} &D_I+TC_{OPT}\left[\bar{y}_I(K-I)\right] \\ &D_2+TC_{OPT}\left[\bar{y}_2(K-I)\right] \\ &\vdots \\ &D_m+TC_{OPT}\left[\bar{y}_m(K-I)\right] \end{aligned} \qquad \text{Equ. 6.I9.}$$

which is denoted by $TC_{OPT}\left[\bar{x}_I(K-2)\right]$.

Again, this calculation is repeated for all states $\bar{x}(K-2)$, so that we will know all $TC_{OPT}\left[\bar{x}(K-2)\right]$.

It can now be clearly seen that this dynamic programming approach is an iterative process which, at any stage, links with the following stage by means of the minimum cost functions TC_{OPT}.

Mathematically, this can be written as

$$TC_{OPT}\left[\bar{x}(k)\right] = \underset{\substack{\bar{u}_J\in\bar{U} \\ J=I,m(\bar{x},k)}}{\text{Min}} \; 1\left[\bar{x}(k),\bar{u}_J(k),k\right] + TC_{OPT}\left[\bar{g}\left[\bar{x}(k),\bar{u}_J(k),k\right]\right] \quad \text{Equ. 6.20.}$$

where $m(\bar{x},k)$ is the number of decisions which may be made for this state and stage and $TC_{OPT}\left[\bar{x}(k)\right]$ represents the minimum cost, starting in state $\bar{x}(k)$ at time k, of going to the end of the process.

In dynamic programming notation $TC_{OPT}\left[\bar{x}(k)\right]$ is usually represented by $I\left[\bar{x}(k),k\right]$ so that the equation becomes

$$I\left[\bar{x}(k),k\right] = \underset{\substack{\bar{u}_J \in \bar{U}(\bar{x},k) \\ J=I,m(\bar{x},k)}}{\text{Min}} \; l\left[\bar{x}(k),\bar{u}_J(k),k\right] + I\left[\bar{g}\left[\bar{x}(k),\bar{u}_J(k),k\right],k+I\right] \quad \ldots \text{ Equ. 6.2I.}$$

This equation is known as the iterative functional equation of dynamic programming. It states that the minimum cost of going from a state $\bar{x}$ at stage k to the end of the process is obtained by minimizing the sum of the cost incurred during stage k, and the minimum cost in going to the end of the process from the resulting state at stage (k+I).

It can be seen from the iterative functional equation that if we know the penalty function at stage K then the equation may be applied in a backward direction starting from stage K and finishing at stage 0. After all calculations are complete we will have the optimum decision to be made at every stage and for every state which may occur at that stage. If the state of the system $\bar{x}(0)$ is known at stage o then all that is necessary is to apply the now known optimum decision for that state and stage and find the resulting state. The known optimum decision for that state at that stage is applied and the resulting state computed. This process is repeated until stage K is reached, when we will have found the optimum decision which will have to be made at each stage and the resulting optimum sequence of states.

Since the optimum rules starting from any stage and any state are found by the d.p. then if an error is made in the operation of the system, so that the optimal sequence of states is broken at some stage then all that is necessary is to find from the dynamic programming results the optimal rules to be applied starting from that stage and that state which is the result of the incorrect operation.

Example 2

Consider a two reservoir system with no interconnections, supplying one

demand.

Let the discretised levels be 0,I,2 units in each reservoir so that the possible state vectors at any stage are

$$\bar{x}_I=\begin{bmatrix}0\\0\end{bmatrix}\ \bar{x}_2=\begin{bmatrix}0\\I\end{bmatrix}\ \bar{x}_3=\begin{bmatrix}0\\2\end{bmatrix}\ \bar{x}_4=\begin{bmatrix}I\\0\end{bmatrix}\ \bar{x}_5=\begin{bmatrix}I\\I\end{bmatrix}\ \bar{x}_6=\begin{bmatrix}I\\2\end{bmatrix}\ \bar{x}_7=\begin{bmatrix}2\\0\end{bmatrix}\ \bar{x}_8=\begin{bmatrix}2\\I\end{bmatrix}\ \bar{x}_9=\begin{bmatrix}2\\2\end{bmatrix}$$

Let there be three decisions at each stage :

I. Take all demand from res. I. i.e. $\bar{u}_I=\begin{bmatrix}D\\0\end{bmatrix}=\begin{bmatrix}u_I\\u_2\end{bmatrix}$ where D is the demand at any stage.

2. Take all demand from res. 2. i.e. $\bar{u}_2=\begin{bmatrix}0\\D\end{bmatrix}=\begin{bmatrix}u_I\\u_2\end{bmatrix}$

3. Take half of the demand from each reservoir i.e.

$$\bar{u}_3=\begin{bmatrix}D/2\\D/2\end{bmatrix}=\begin{bmatrix}u_I\\u_2\end{bmatrix}$$

Assume that the reservoirs only have to be operated for I year and let the time between operations be one season of 3 months, so that there are four stages to the process.

k = 0,I,2,3.

Let the inflows in these time periods be

	Res.I	Res.2.
k = 0	INF = 2 units	INF = I unit
k = I	INF = I unit	INF = 2 units
k = 2	INF = I unit	INF = 2 units
k = 3	INF = 3 units	INF = I unit

Let the demands in these time periods be

k = 0 D = 4 units

k = I D = 4 units

k = 2 D = 4 units

k = 3 D = 4 units

We wish to minimise the deficits to demand over the year.

Assume that the operator wishes to have both reservoirs full at the end of the period, andif they are not full, then water will have to be pumped into them from some other source, thus incurring pumping costs.

Let the pumping cost be £190 / unit pumped for each reservoir.

Let the cost of a deficit to demand be £300 / unit.

The reservoirs are both full at the beginning of the process.

The penalty costs at the end of the process, k = 4, are the costs of pumping water into the reservoirs until they are full. Therefore :

$$\phi[\bar{x}_1(4),4] = 4 \times 290 = £1160 = I[\bar{x}_1,4]$$
$$\phi[\bar{x}_2(4),4] = 3 \times 290 = £870 = I[\bar{x}_2,4]$$
$$\phi[\bar{x}_3(4),4] = 2 \times 290 = £580 = I[\bar{x}_3,4]$$
$$\phi[\bar{x}_4(4),4] = 3 \times 290 = £870 = I[\bar{x}_4,4]$$
$$\phi[\bar{x}_5(4),4] = 2 \times 290 = £580 = I[\bar{x}_5,4]$$
$$\phi[\bar{x}_6(4),4] = 1 \times 290 = £290 = I[\bar{x}_6,4]$$
$$\phi[\bar{x}_7(4),4] = 2 \times 290 = £580 = I[\bar{x}_7,4]$$
$$\phi[\bar{x}_8(4),4] = 1 \times 290 = £290 = I[\bar{x}_8,4]$$
$$\phi[\bar{x}_9(4),4] = 0 \times 290 = 0 = I[\bar{x}_9,4]$$

The expression to be minimised is

$$J = \left\{ \sum_{k=0}^{K-1} (D(k)-R_1(k)-R_2(k)) \times 400 \right\} + \phi[\bar{x}(4),4]$$

where R_1 and R_2 are the actual releases from reservoir I and reservoir 2, which may be different from $u_1(k)$ and $u_2(k)$ since there may not be enough water in the reservoirs to release all of the stipulated release.

The state transition function $\bar{g}$ is given by the following :

$$x_I(k+I) = g_I[x_I(k), u_I(k), k]$$

$$= x_I(k) + \text{INFLOW I } (k) - u_I(k)$$

$$x_2(k+I) = g_2[x_2(k), u_2(k), k]$$

$$= x_2(k) + \text{INFLOW 2 } (k) - u_2(k)$$

with the constraints :

a. If $x_I(k+I) \geqslant o$ then

$R_I = u_I$

b. If $x_I(k+I) < o$ then

$R_I = x_I(k) + \text{INFLOW I } (k)$ and next state $x_I(k+I) = o$

c. If $x_2(k+I) \geqslant o$ then

$R_2 = u_2$

d. If $x_2(k+I) < o$ then

$R_2 = x_2(k) + \text{INFLOW 2}$ and next state $x_2(k+I) = o$

e. If $x_I(k+I) > x_{I\ max}$ then

SPILL I $= x_I(k+I) - x_{I\ max}$ and next state $x_I(k+I) = x_{I\ max}$

$x_{I\ max} = 2$ units

f. If $x_2(k+I) > x_{2\ max}$ then

SPILL 2 $= x_2(k+I) - x_{2\ max}$ and next state $x_2(k+I) = x_{2\ max}$

$x_{2\ max} = 2$ units

The dynamic programming procedure is set out in the following table. The costs in column (II) are initially set to the penalty costs since these may be viewed as being the costs to go from stage K to the end of the process.

STARTING STATE		AVAILABLE WATER		DECISIONS		ACTUAL RELEASES		SPILLS		RESULTING STATE		COST OF GOING FROM RESULTING STATE TO END OF PROCESS	DEFICIT COST (D-R1-R2) x 300	TOTAL COST
RES 1	RES 2	RES 1 COL(1)+INF1	RES 2 COL(2)+INF2	u_1	u_2	R_1	R_2	SPILL 1	SPILL 2	RES 1	RES 2			
0	0	3	I	4	0*	3	0	0	0	0	I	870	300	II70
				2	2	2	I	0	0	I	0	870	300	II70
				0	4	0	I	I	0	2	0	580	900	I480
	I	3	2	4	0	3	0	0	0	0	2	580	300	880
				2	2*	2	2	0	0	I	0	870	0	870
				0	4	0	2	I	0	2	0	580	600	II80
	2	3	3	4	0	3	0	0	I	0	2	580	300	880
				2	2*	2	2	0	0	I	I	580	0	580
				0	4	0	3	I	0	2	0	580	300	880
I	0	4	I	4	0*	4	0	0	0	0	I	870	0	870
				2	2	2	I	0	0	2	0	580	300	II80
				0	4	0	I	2	0	2	0	580	900	I480
	I	4	2	4	0*	4	0	0	0	0	2	580	0	580
				2	2	2	2	0	0	2	0	580	0	580
				0	4	0	2	2	0	2	0	580	600	II80
	2	4	3	4	0	4	0	0	I	0	2	580	0	580
				2	2*	2	2	0	0	2	I	290	0	290
				0	4	0	3	2	0	2	0	580	300	880
2	0	5	I	4	0*	4	0	0	0	I	I	580	0	580
				2	2	2	I	I	0	2	0	580	300	880
				0	4	0	I	3	0	2	0	580	900	I480
	I	5	2	4	0*	4	0	0	0	I	2	290	0	290
				2	2	2	2	I	0	2	0	580	0	580
				0	4	0	2	3	0	2	0	580	600	II80
	2	5	3	4	0*	4	0	0	I	I	2	290	0	290
				2	2	2	2	I	0	2	I	290	0	290
				0	4	0	3	3	0	2	0	580	300	880

INF I = 3 D = 4
INF 2 = I

* denotes the optimum decision for a state

STAGE 3

STARTING STATE		AVAILABLE WATER		DECISIONS		ACTUAL RELEASES		SPILLS		RESULTING STATE		COST OF GOING FROM RESULTANT STATE TO END OF PROCESS	DEFICIT COST $(D-R_1-R_2)\times 300$	TOTAL COST
RES 1	RES 2	RES 1 COL(1)+ INF 1	RES 2 COL(2)+ INF 2	u_1	u_2	R_1	R_2	SPILL 1	SPILL 2	RES 1	RES 2			
0	0	I	2	4	0	I	0	0	0	0	2	580	900	I480
				2	2*	I	2	0	0	0	0	II70	300	I470
				0	4	0	2	0	0	I	0	870	600	I470
	I	I	3	4	0	I	0	0	I	0	2	580	900	I480
				2	2*	I	2	0	0	0	I	870	300	II70
				0	4	0	3	0	0	I	0	870	300	II70
	2	I	4	4	0	I	0	0	2	0	2	580	900	I480
				2	2	I	2	0	0	0	2	580	300	880
				0	4*	0	4	0	0	I	0	870	0	870
I	0	2	2	4	0	2	0	0	0	0	2	580	600	II80
				2	2*	2	2	0	0	0	0	II70	0	II70
				0	4	0	2	0	0	2	0	580	600	II80
	I	2	3	4	0	2	0	0	I	0	2	580	600	II80
				2	2*	2	2	0	0	0	I	870	0	870
				0	4	0	3	0	0	2	0	580	300	880
	2	2	4	4	0	2	0	0	2	0	2	580	600	II80
				2	2*	2	2	0	0	0	2	580	0	580
				0	4	0	4	0	0	2	0	580	0	580
2	0	3	2	4	0	3	0	0	0	0	2	580	300	880
				2	2*	2	2	0	0	I	0	870	0	870
				0	4	0	2	I	0	2	0	580	600	II80
	I	3	3	4	0	3	0	0	I	0	2	580	300	880
				2	2*	2	2	0	0	I	I	580	0	580
				0	4	0	3	I	0	2	0	580	300	880
	2	3	4	4	0	3	0	0	2	0	2	580	300	880
				2	2*	2	2	0	0	I	2	290	0	290
				0	4	0	4	I	0	2	0	580	0	580

INFI = I D = 4

INF2 = 2

STAGE 2

STARTING STATE		AVAILABLE WATER		DECISIONS		ACTUAL RELEASES		SPILLS		RESULTANT STATE		COST OF GOING FROM RESULTANT STATE TO END OF PROCESS	DEFICIT COST (D-R_1-R_2) × 300	TOTAL COST
RES 1	RES 2	RES 1 COL(1)+ INF1	RES 2 COL(2)+ INF2	u_1	u_2	R_1	R_2	SPILL 1	SPILL 2	RES 1	RES 2			
0	0	I	2	4	0	I	0	0	0	0	2	870	900	I770
				2	2*	I	2	0	0	0	0	I470	300	I770
				0	4	0	2	0	0	I	0	II70	600	I770
	I	I	3	4	0	I	0	0	I	0	2	870	900	I770
				2	2*	I	2	0	0	0	I	II70	300	I470
				0	4	0	3	0	0	I	0	II70	300	I470
	2	I	4	4	0	I	0	0	2	0	2	870	900	I770
				2	2	I	2	0	0	0	2	870	300	II70
				0	4*	0	4	0	0	I	0	II70	0	II70
I	0	2	2	4	0	2	0	0	0	0	2	870	600	I470
				2	2*	2	2	0	0	0	0	I470	0	I470
				0	4	0	2	0	0	2	0	870	600	I470
	I	2	3	4	0	2	0	0	I	0	2	870	600	I470
				2	2*	2	2	0	0	0	I	II70	0	II70
				0	4	0	3	0	0	2	0	870	300	II70
	2	2	4	4	0	2	0	0	2	0	2	870	600	I470
				2	2*	2	2	0	0	0	2	870	0	870
				0	4	0	4	0	0	2	0	870	0	870
2	0	3	2	4	0	3	0	0	0	0	2	870	300	II70
				2	2*	2	2	0	0	I	0	II70	0	II70
				0	4	0	2	I	0	2	0	870	600	I470
	I	3	3	4	0	3	0	0	I	0	2	870	300	II70
				2	2*	2	2	0	0	I	I	870	0	870
				0	4	0	3	I	0	2	0	870	300	II70
	2	3	4	4	0	3	0	0	2	0	2	870	300	II70
				2	2*	2	2	0	0	I	2	580	0	580
				0	4	0	4	I	0	2	0	870	0	870

INFI = I D = 4

INF2 = 2

STAGE I

STARTING STATE		AVAILABLE WATER		DECISIONS		ACTUAL RELEASES		SPILLS		RESULTANT STATE		COST OF GOING FROM RESULTANT STATE TO END OF PROCESS	DEFICIT COST $(D-R_1-R_2)$ x 300	TOTAL COST
RES 1	RES 2	RES 1 COLUMN (1) + INF1	RES 2 COLUMN (2) + INF2	u_1	u_2	R_1	R_2	SPILL 1	SPILL 2	RES 1	RES 2			
				4	0*	2	0	0	0	0	I	I470	600	2070
0	0	2	I	2	2	2	I	0	0	0	0	I770	300	2070
				0	4	0	I	0	0	2	0	II70	900	2070
				4	0*	2	0	0	0	0	2	II70	600	I770
	I	2	2	2	2	2	2	0	0	0	0	I770	0	I770
				0	4	0	2	0	0	2	0	II70	600	I770
				4	0	2	0	0	I	0	2	II70	600	I770
	2	2	3	2	2*	2	2	0	0	0	I	I470	0	I470
				0	4	0	3	0	0	2	0	II70	300	I470
				4	0*	3	0	0	0	0	I	I470	300	I770
I	0	3	I	2	2	2	I	0	0	I	0	I470	300	I770
				0	4	0	I	I	0	2	0	II70	900	2070
				4	0*	3	0	0	0	0	2	II70	300	I470
	I	3	2	2	2	2	2	0	0	I	0	I470	0	I470
				0	4	0	2	I	0	2	0	II70	600	I770
				4	0	3	0	0	I	0	2	II70	300	I470
	2	3	3	2	2*	2	2	0	0	I	I	II70	0	II70
				0	4	0	3	I	0	2	0	II70	300	I470
				4	0*	4	0	0	0	0	I	I470	0	I470
2	0	4	I	2	2	2	I	0	0	2	0	II70	300	I470
				0	4	0	I	2	0	2	0	II70	900	2070
				4	0*	4	0	0	0	0	2	II70	0	II70
	I	4	2	2	2	2	2	0	0	2	0	II70	0	II70
				0	4	0	2	2	0	2	0	II70	600	I770
				4	0	4	0	0	I	0	2	II70	0	II70
	2	4	3	2	2*	2	2	0	0	2	I	870	0	870
				0	4	0	3	2	0	2	0	II70	300	I470

INFI = 2 D = 4

INF2 = I

STAGE 0

Retrieval of optimal trajectory

Let us assume that the reservoirs are both full at the beginning of the process

i.e. $\bar{x}(o) = \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \bar{x}_9$

then from the stage o chart it can be seen that the optimum decision is to take half the demand from each reservoir

i.e. $\bar{u}_{OPT}(\bar{x}_9, o) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ with $\bar{R}_{OPT}(\bar{x}_I, o) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

The resultant state is $\bar{x}(I) = \begin{bmatrix} 2 \\ I \end{bmatrix} = \bar{x}_8$

From the stage I chart the optimal decision for this state is

$\bar{u}_{OPT}(\bar{x}_8, I) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ with $\bar{R}_{OPT}(\bar{x}_8, I) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

The resultant state is $\bar{x}(2) = \begin{bmatrix} I \\ I \end{bmatrix} = \bar{x}_5$

From the stage 2 chart the optimal decision for this state is

$\bar{u}_{OPT}(\bar{x}_5, 2) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ with $\bar{R}_{OPT}(\bar{x}_5, 2) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

The resultant state is $\bar{x}(3) = \begin{bmatrix} 0 \\ I \end{bmatrix} = \bar{x}_2$

From the stage 3 chart the optimal decision for this state is

$\bar{u}_{OPT}(\bar{x}_2, 3) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ with $\bar{R}_{OPT}(\bar{x}_2, 3) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

The resultant state is $\bar{x}(4) = \begin{bmatrix} I \\ 0 \end{bmatrix} = \bar{x}_4$

Hence, at the end of the process the operator will have reservoir 2 empty and reservoir I with one unit, so that he will have to incur the penalty cost for this state, which involves pumping a total of 3 units at a cost of £870.

Thus the total minimum cost is £870 since no deficits occurred.

As a check on this figure, it can be seen from the stage o chart that the total cost of going from stage $\bar{x}(o) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ to the end of the process is

indeed £870 (column (I5)).

Consider the case where subjective decisions are taken :

Since reservoir I. has 4 units and reservoir 2. has 3 units, including inflows, at stage o, there is no obvious decision at this stage.
We could take all from reservoir I. or half from each.
Let the subjective decision be to take all from reservoir I.

Therefore $\bar{x}(I) = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$, $\bar{u}(o) = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$, $\bar{R}(o) = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$

There is no deficit cost.
At stage I reservoir I. has I unit and reservoir 2. has 4 units, including inflows. Thus, the obvious decision is to take all the demand from reservoir 2.

Therefore $\bar{x}(2) = \begin{bmatrix} I \\ 0 \end{bmatrix}$, $\bar{u}(I) = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$, $\bar{R}(I) = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$

There is no deficit cost.

At stage 2 reservoir I. has 2 units and reservoir 2. has 2 units, including inflows. The obvious decision is to take half from each reservoir.

Therefore $\bar{x}(3) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\bar{u}(2) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ $\bar{R}(2) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

There is no deficit cost.

At the 3rd stage reservoir I. has 3 units and reservoir 2. has I unit, including inflows. Therefore, we may take all from reservoir I. or half from each, at the same cost. Let the decision be to take all from reservoir I.

Therefore, $\bar{x}(4) = \begin{bmatrix} 0 \\ I \end{bmatrix}$, $\bar{u}(3) = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$, $\bar{R}(3) = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$

There is a deficit of I unit at a cost of £300.

The cost of pumping the reservoirs full is £870.

Therefore, the total cost is £1170.

Therefore, even with this simple process, the dynamic programming solution finds a more economical sequence of decisions, than a subjective method. In the example, it is possible to see what the optimal sequence of decisions is, because of the limited period of time, but in a long historical record, it would be impossible, without a simulation of each policy, to accurately determine the optimum policy.

The number of possible combinations of decision for any process is m^N where m is the number of decisions which may be made at any stage and N is the number of stages. This means that m^N simulations with different combinations of decisions, would have to be made, and the costs computed, to be sure of obtaining the optimum sequence, so that in the example 2^4 (=16) trials would have been necessary.

In the above example, the inflows, demands and possible decisions have been chosen so that all the discretised states only make transitions to other discretised states. The method is easily extended to cases where transitions are made to intermediate states. The cost of going to the end of the process from the resultant intermediate state is obtained by interpolating between the costs for the nearest discretised states. The extra computer time and cost involved in using interpolation is often worth the reduction in storage achieved.

For example, consider starting state $\bar{x}(o) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ in the example.

Let the inflow to reservoir I be 1.7 units instead of 2 units, considering two decisions only.

The first line of the chart would now be :

STARTING STATE		AVAILABLE WATER		DECISIONS		ACTUAL RELEASES		SPILLS		RESULTANT STATE		COST OF GOING FROM RESULTANT STATE TO END OF PROCESS	DEFICIT COST $(D-R_1-R_2)$ x 300	TOTAL COST
RES 1	RES 2	RES 1 col(1) + INF1	RES 2 col(2) + INF2	u_1	u_2	R_1	R_2	SPILL 1	SPILL 2	RES 1	RES 2			
0	0	1·7	1	4	0	1·7	0	0	0	0	1	1470	(4-1·7)x300	2160
				0	4	0	1	0	0	1·7	0	(V=) 1260	(4-1)x300	2160

The value v = I260 is obtained by interpolation as shown below.

For the second decision, the resultant state is $\bar{x}(I) = \begin{bmatrix} I\cdot 7 \\ 0 \end{bmatrix}$

We now consider the nearest quantised states to this

i.e. $\bar{x}(I) = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ and $\bar{x}(I) = \begin{bmatrix} I \\ 0 \end{bmatrix}$

The cost of going from $\bar{x}(I) = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ to the end of the process is obtained from stage I as £II70.

The cost of going from $\bar{x}(I) = \begin{bmatrix} I \\ 0 \end{bmatrix}$ to the end of the process is obtained from stage I as £I470.

Assuming a linear interpolation procedure, then the cost of going from $\bar{x}(I) = \begin{bmatrix} I\cdot 7 \\ 0 \end{bmatrix}$ to the end of the process is given by

$V\begin{bmatrix} I\cdot 7 \\ 0 \end{bmatrix} = 0.7 \times II70 + 0.3 \quad I470 \quad = £I260$ where $V[\bar{x}(k)]$ represents the cost of going from state $\bar{x}$ at stage k to end of the process.

In general, for a two component vector, the resultant state is

$\bar{x}(k) = \begin{bmatrix} x_I \\ x_2 \end{bmatrix}$

The nearest states are $\bar{x}(k) = \begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} c \\ d \end{bmatrix}, \begin{bmatrix} a \\ d \end{bmatrix}, \begin{bmatrix} c \\ b \end{bmatrix},$

where $a < x_I < c$ and $b < x_2 < d$

The cost of $\bar{x}(k) = \begin{bmatrix} x_I \\ x_2 \end{bmatrix}$ is then given by

$$V\begin{bmatrix} x_I \\ x_2 \end{bmatrix} = \left\{\left(I-\frac{(x_2-b)}{(d-b)}\right) V\begin{bmatrix} a \\ b \end{bmatrix} + \frac{(x_2-b)}{(d-b)} V\begin{bmatrix} a \\ d \end{bmatrix}\right\}\left(I-\frac{(x_I-a)}{(c-a)}\right)$$
$$+\left\{\left(I-\frac{(x_2-b)}{(d-b)}\right) V\begin{bmatrix} c \\ b \end{bmatrix} + \frac{(x_2-b)}{(d-b)} V\begin{bmatrix} c \\ d \end{bmatrix}\right\}\left(\frac{(x_I-a)}{(c-a)}\right) \quad \text{Equ. 6.22.}$$

Other interpolation functions may be used but linear interpolation is sufficient if the discretised states are near to one another.

Chapter 7

Extension to the Stochastic Case

7.I. Introduction

Stochastic dynamic programming is a similar process to deterministic programming but deals with the case where uncertainty is present in a system. The stage, state and control variables are the same as for the deterministic case but the system equations are affected by stochastic inputs, and possibly stochastic outputs. In the reservoirs problem, the stochastic inputs are those streamflows which cannot be predicted exactly. Instead of using fixing inflows to each reservoir at a particular time as in deterministic programming, a discretised probability distribution of inflows is specified. The inflows are now denoted by the vector $\bar{w}_J$, J=I,r, where r is the number of discretised sets of inflows in the probability distribution. Each element in the vector $\bar{w}_J$ contains one random input to the system at a state and stage and the vector has one probability, b_J, attached to it. There may be any number of elements in the vector and any number of r, of vectors with different probabilities may be possible for a given state and stage. The distributions of these vectors are assumed to be independent from one stage to the next. If there is a correlation in time then this can be overcome by defining additional state variables which are independent of time. The distribution of inputs at the stage and state under consideration in the dynamic programming calculation is then given by

$$\bar{w}_J(k) \in W(\bar{x}(k),k) \qquad J = I,r$$

The calculations are carried out, in the same way as for the deterministic case, for every state at each stage so that if the values of the state variables are known from observation at any stage the operation of the system, from that stage to the end of the process, becomes independent of any occurrence before that stage, which is the necessary criterion for

dynamic programming.

7.2. The state transitions

The system equations now become

$$\bar{x}(k+1) = \bar{g}\left[\bar{x}(k),\bar{u}(k),\bar{w}(k),k\right] \quad \ldots \text{Equ. 7.1.}$$

In the stochastic case it can be seen that the resultant state now varies with the applied combination of inputs as well as with the chosen control. Any of the resultant states may be the same for different $\bar{w}$ because of the constraints on the problem.

The costs associated with the state transitions from a specified state, $\bar{x}_i$, stage, k, and control, u_f, may be written as

$$\begin{aligned}
C_I &= 1\left[\bar{x}_i(k),\bar{u}_f(k),\bar{w}_I(k),k\right] + I\left[\bar{g}\left[\bar{x}_i(k),\bar{u}_f(k),\bar{w}_I(k),k\right],\ k+1\right] \\
&\quad \text{with probability } b_I \\
&\vdots \\
C_J &= 1\left[\bar{x}_i(k),\bar{u}_f(k),\bar{w}_J(k),k\right] + I\left[\bar{g}\left[\bar{x}_i(k),\bar{u}_f(k),\bar{w}_J(k),k\right],\ k+1\right] \\
&\quad \text{with probability } b_J \\
&\vdots \\
C_r &= 1\left[\bar{x}_i(k),\bar{u}_f(k),\bar{w}_r(k),k\right] + I\left[\bar{g}\left[\bar{x}_i(k),\bar{u}_f(k),\bar{w}_r(k),k\right],\ k+1\right] \\
&\quad \text{with probability } b_r \qquad \ldots \text{Equ. 7.2.}
\end{aligned}$$

where b_J represents the probability of obtaining the J^{th} inflow vector for a particular stage and stage and where C_J is the cost of going from this state and stage to the end of the process if $\bar{w}_J$ occurs at this stage.

7.3. The expected cost

Some method of combining these costs must be found in order to arrive at one value for the cost of going from state $\bar{x}_i(k)$ to the end of the process for each control, $\bar{u}_f$, which may be applied at stage k, so that the optimum cost may be chosen.

The method used is to combine the various costs, C, for one control into the expected or average cost $E(\bar{x}_i(k),\bar{u}_f(k),k)$ which is obtained by summing the C_J multiplied by their corresponding probabilities, b_J,(J=I,r) so that

$$E(\bar{x}_i(k),\bar{u}_f(k),k) = \sum_{J=I}^{r} C_J \times b_J \quad . \quad . \quad . \quad . \quad \text{Equ.7.3.}$$

The optimum cost then becomes

$$I\left[\bar{x}_i(k),k\right] = \underset{\substack{\bar{u}_f \\ f=I,m}}{\text{Min}} \left\{ E(\bar{x}_i(k),\bar{u}_f(k),k) \right\} \quad . \quad . \quad . \quad . \quad \text{Equ.7.4.}$$

or, in expanded form,

$$I\left[\bar{x}_i(k),k\right] = \underset{\substack{\bar{u}_f \\ f=I,m}}{\text{Min}} \sum_{J=I}^{r} \left(l\left[\bar{x}_i(k),\bar{u}_f(k),\bar{w}_J(k),k\right] + I\left[\bar{g}\left[\bar{x}_i(k),\bar{u}_f(k),\bar{w}_J(k),k\right],k+I\right] \right) \times b_J \quad . \quad . \quad \text{Equ.7.5.}$$

which is the standard form of the iterative functional equation for the stochastic case.

7.4. The Performance Criterion

The performance criterion may be written as

$$J = \underset{\bar{w}(o),\ \bar{w}(k),\ \bar{w}(K)}{E} \sum_{k=0}^{K} l\left[\bar{x}(k),\bar{u}(k),\bar{w}(k),k\right] \quad . \quad . \quad \text{Equ.7.6.}$$

where the expectation is taken over the sequence of stochastic inputs.

All constraints and penalty functions are similar to those for the deterministic case.

The use of the expected cost for determining which is the best policy is a point of controversy. Much discussion may be found in the literature on operational research techniques on whether a real individual makes decisions based upon an expected gain or loss as computed. In practice, a manager may well place such a high cost on failure of any kind that this distorts the average cost. It is assumed in this thesis that failure of a supply is not catastrophic and that 'hedging' is justified.

Example 7.I.

The random vector, $\bar{w}$, for any state, $\bar{x}_i$, and stage, k, may be any of the subsets

$$\bar{w}_I(k) = \left[w_{II}, w_{I2}, \ldots w_{Ia}, \ldots w_{In}\right] \text{ with probability } b_I$$

$$\bar{w}_J(k) = \left[w_{JI}, w_{J2}, \ldots w_{Ja}, \ldots w_{Jn}\right] \text{ with probability } b_J$$

$$\bar{w}_r(k) = \left[w_{rI}, w_{r2}, \ldots w_{ra}, \ldots w_{rn}\right] \text{ with probability } b_r$$

which constitute the main set $\bar{W}(\bar{x}_i(k),k)$,

where w_{Ja} could be the inflow to the a^{th} reservoir under the J^{th} possible combination of inflows,

and where n could be the total number of reservoirs in the system.

The equivalent deterministic vector is

$$\bar{w}_I(k) = \left[w_{II}, w_{I2}, \ldots w_{Ia}, \ldots w_{In}\right] \text{ with probability } b_I = I$$

Example 7.2.

If the possible inflows in time period (k+I) depend upon the inflow in time period k, then state variables must be defined which eliminates the dependence.

Consider a one reservoir system where the inflows in any month depends upon whether the inflow in the previous month was higher or lower than the average for that month.

If previous inflow was higher than average then we might have

inflow in this month = I unit = $\bar{w}_I(k)$ with probability 0.2

2 unit = $\bar{w}_2(k)$ with probability 0.5

3 unit = $\bar{w}_3(k)$ with probability 0.3

where $\bar{w}_J(k)$ contains only one element since there is only one inflow to the system.

If previous inflow was lower than average then we might have

inflow in this month = 0.7 unit = $\bar{w}_I(k)$ with probability 0.3

I.6 unit = $\bar{w}_2(k)$ with probability 0.4

2 unit = $\bar{w}_3(k)$ with probability 0.3

The overall average inflow for this month is therefore

$$\frac{(I \times 0.2 + 2 \times 0.5 + 3 \times 0.3) + (0.7 \times 0.3 + I.6 \times 0.4 + 2 \times 0.3)}{2}$$

$$= \underline{I.775 \text{ units}}$$

One extra state variable is now defined, which can take two values, which denote whether the previous inflow was high or low.

At any stage, dynamic programming calculations are carried out for both values, since, until the computations have been performed for the whole process, when the optimal trajectory can then be chosen from known initial values of all the variables, it is impossible to know which of the values, high or low, will occur at that stage.

The state vector now becomes

$$\bar{x}(k) = \begin{bmatrix} x_I(k) \\ x_2(k) \end{bmatrix}$$ where x_I is the level in the reservoir and x_2 contains a number or letter which denotes high or low previous inflows

Let the reservoir have three discrete levels at any stage, 0,I,2 units and let H denote high previousinflow and L denote low previous inflows, then the possible states at that stage are now

$$\begin{bmatrix} 0 \\ L \end{bmatrix}, \begin{bmatrix} 0 \\ H \end{bmatrix}, \begin{bmatrix} I \\ L \end{bmatrix}, \begin{bmatrix} I \\ H \end{bmatrix}, \begin{bmatrix} 2 \\ L \end{bmatrix}, \begin{bmatrix} 2 \\ H \end{bmatrix}$$

Any vector which contains H has the inflows I,2,3 units with respective probabilities 0.2,0.5, 0.3, and any vector which contains L has the inflows 0.7, I.6, 2 with probabilities 0.3, 0.4 and 0.3.

The resultant states vectors under any control rule must also contain the variables L or H and these are evaluated by finding whether the inflow under consideration at this stage is higher or lower than the average.

Consider being in state $\bar{x}(k) = \begin{bmatrix} 0 \\ H \end{bmatrix}$

Then the inflows are

I unit with probability 0.2

2 unit with probability 0.5

3 unit with probability 0.3

The average inflow for the month is I.775 units

Let the decision be to release I unit from the reservoir.

An inflow of less than the average for the month will lead to a state with L in the vector and an inflow higher than average will lead to a state with H in the vector.

So that the resultant states under a release of I unit will be

$$\bar{x}(k+I) = \begin{bmatrix} 0+I-I \\ L \end{bmatrix} = \begin{bmatrix} 0 \\ L \end{bmatrix}$$ with probability 0.2

$$\bar{x}(k+I) = \begin{bmatrix} 0+2-I \\ H \end{bmatrix} = \begin{bmatrix} I \\ H \end{bmatrix}$$ with probability 0.5

$$\bar{x}(k+1) = \begin{bmatrix} 0+3-1 \\ H \end{bmatrix} = \begin{bmatrix} 2 \\ H \end{bmatrix} \quad \text{with probability } 0.3$$

Each of these resultant states will have a cost attached to it from the previous step in the process, so that the total expected cost for this decision can be calculated.

The concept if high and low previous inflows can be extended to include any number of antecedent indices.

7.5. The long term stochastic process

So far, the methods discussed can only be applied to a process which has a finite number of stages. The results of the calculations yield one optimum control rule for each state in each stage, and the results obtained for each state may not be the same at different stages.

A dynamic program may be required to find the optimum rules at each stage and state for a twelve months process, where the inflows at each stage are given as probability distributions. If the distributions for each month are obtained from historical data then they represent the statistical properties of the inflows over a long period of time. In this case, the distributions in a particular month are identical for the same month in every year.

If a dynamic programming computation is performed over a period of several years, using the same monthly distributions and the same unit costs in each year then it is found that if the number of years is large enough, the control rules chosen for each state in a particular month become the same for every year, so that the rules obtained after a long period of time represent the long term optimal decisions. A proof of this convergence is given by Bellman but an analogy can be drawn between the solution of the iterative functional equation and the iterative solution of an implicit function by Newton's method.

Another analogy is the way in which one can determine the probability of emptiness of a reservoir either by powering the transition matrix or by solving directly for the steady state probabilities.

However, the convergence to a long term policy is asymptotic and it is difficult to determine exactly when the policy has converged, but an analysis of the iterative functional equation for a system with a large number of stages leads to a more direct analytical solution to the problem.

An additional concept of state and stage variables is needed to describe the development of an analytical method for the long term stochastic process.

The stage variable is still considered to be a period of one month but the month is not specified. Instead, the name of the month is included in the state vectors as another state variable, so that at any stage the system can be in any combination of month, levels, antecedent indices etc.

e.g. $\bar{x}(k) = \begin{bmatrix} \text{JANUARY} \\ \text{LEVEL I} \\ \text{HIGH} \\ \text{LEVEL 2} \\ \text{HIGH} \end{bmatrix}$

and each of these state vectors will have a probability distribution of inflows attached to it.

Before the analytical method is described the dynamic programming equations for the iterative slution of the normal fixed period process, with stochastic inflows, using the same monthly distributions in each year, and employing state vectors as described above, will be developed.

7.6. The Long Term Iterative Stochastic Method

In order to achieve convergence of the policy with a long term process, the unit costs employed at each stage must be the same, although they may still vary with the state and control, and since the month is now a part of the state vector the costs may vary from month to month.

Therefore, the unit cost may be written as

$$l\left[\bar{x}_i, \bar{u}_f, \bar{w}_J\right] \quad \text{for all i, f, and J.}$$

Equations 7.I. may now be written as

$$\bar{x}(k+I) = \bar{g}\left[\bar{x}(k), \bar{u}(k), \bar{w}(k)\right] \qquad \text{Equ. 7.7.}$$

Equations 7.2. may be written as

$$C_J = l\left[\bar{x}_i, \bar{u}_f, \bar{w}_J\right] + I\left[\bar{g}\left[\bar{x}_i, \bar{u}_f, \bar{w}_J\right], k+I\right]$$

$$\text{with probability } b_J, \quad J=I,r \qquad \text{Equ. 7.8.}$$

Equations 7.3. may be written

$$E\left(\bar{x}_i(k), \bar{u}_f(k)\right) = \sum_{J=I}^{r} C_J \times b_J \qquad \text{Equ. 7.9.}$$

Equation 7.4. may be written

$$I\left[\bar{x}_i(k), k\right] = \min_{\substack{\bar{u}_f \\ f=I,m}} E\left(\bar{x}_i(k), u_f(k)\right) \qquad \text{Equ. 7.I0.}$$

Equations 7.9. may be expanded as

$$E\left(\bar{x}_i(k), u_f(k)\right) = \sum_{J=I}^{r} l\left[\bar{x}_i, \bar{u}_f, w_J\right] x\ b_J$$

$$\sum_{J=I}^{r} I\left[\bar{g}\left[\bar{x}_i, \bar{u}_f, w_J\right], k+I\right] x\ b_J$$

$$\text{for all i} \qquad \text{Equ. 7.II.}$$

7.7. The infinite time span case

For the development of the general equations for the infinite time span case it is now necessary to restructure the form of the probability distributions.

Let p_{Ji} be the probability of making a transition from state $\bar{x}_i$ to state $\bar{x}_J$ over one stage for a particular control rule $\bar{u}_f$, where $\bar{x}_i$ and $\bar{x}_J$ are both discrete states,

and let l_{Ji} be the immediate cost of making this decision.

Thus, there exists a matrix $\bar{P}$ with elements p_{Ji} which describes the state transitions given a fixed control rule for each state.

This matrix has the form,

$$\bar{P} = \begin{array}{c} \\ \text{Jan} \\ \text{Feb} \\ \text{Mar} \\ \text{Apr} \\ \text{May} \\ \text{Jun} \\ \text{Jly} \\ \text{Aug} \\ \text{Sep} \\ \text{Oct} \\ \text{Nov} \\ \text{Dec} \end{array} \begin{array}{c} \begin{array}{cccccccccccc} \text{Jan} & \text{Feb} & \text{Mar} & \text{Apr} & \text{May} & \text{Jun} & \text{Jly} & \text{Aug} & \text{Sep} & \text{Oct} & \text{Nov} & \text{Dec} \end{array} \\ \begin{bmatrix} & & & & & & & & & & & \bar{M}_{I2} \\ \bar{M}_{I} & & & & & & & & & & & \\ & \bar{M}_{2} & & & & & & & & & & \\ & & \bar{M}_{3} & & & & & & & & & \\ & & & \bar{M}_{4} & & & & & & & & \\ & & & & \bar{M}_{5} & & & & & & & \\ & & & & & \bar{M}_{6} & & & & & & \\ & & & & & & \bar{M}_{7} & & & & & \\ & & & & & & & \bar{M}_{8} & & & & \\ & & & & & & & & \bar{M}_{9} & & & \\ & & & & & & & & & \bar{M}_{I0} & & \\ & & & & & & & & & & \bar{M}_{II} & \end{bmatrix} \end{array}$$

Fig. 7.1.

Only the submatrices, $\bar{M}$, will have non-zero elements since it is only possible to make transitions from states in one month to states in the next month.

It can easily be seen that equations 7.II. may now be written as

$$E(\bar{x}_i(k),\bar{u}_f(k)) = \sum_{J=I}^{a} l_{Ji} x p_{Ji} + \sum_{J=I}^{a} I\left[\bar{x}_J, k+I\right] \times p_{Ji} \quad . \quad . \quad . \quad \text{Equ. 7.I3.}$$

where a is the total number of discretised states at one stage.

The first summation in equations 7.I3. is independent of stage and is therefore constant throughout time for a particular state, $\bar{x}_i$, and control $\bar{u}_f$.

Let $q_i = \sum_{J=I}^{a} l_{Ji} \times p_{Ji}$ for all i . . . Equ. 7.I4.

Therefore $E(\bar{x}_i(k),\bar{u}_f(k)) = q_i + \sum_{J=I}^{a} I\left[\bar{x}_J, k+I\right] \times p_{Ji}$ . . . Equ.7.I5.

7.8. Intermediate transitions

In writing equations 7.I5. it was assumed that under a fixed control, $\bar{u}_f$, there existed a probability, p_{Ji}, of making a transition from a discretised state $\bar{x}_i$ at stage k to every state $\bar{x}_J$ at stage k+I, where $\bar{x}_J$ is a discretised state only. Now the actual inflows at state $\bar{x}_i$ and stage k, represented by $\bar{w}_d(k)$, d=I,r, may not allow transitions to all states at stage k+I and the transitions which exist may not be to discretised states, but it is an easy matter to rearrange the probabilities of the $\bar{w}_d$ so that equations 7.I5 may still be used.

Consider equations 7.8. with some $\bar{w}_d$ which does not lead to a discrete state, Then the corresponding cost under a fixed control, $\bar{u}_f$, is

$$C_d = l\left[\bar{x}_i,\bar{u}_f,\bar{w}_d\right] + I\left[\bar{g}\left[\bar{x}_i,\bar{u}_f,\bar{w}_d\right],k+I\right] \text{ with probability } b_d.$$

Let $\bar{x}_B(k+I) = \bar{g}\left[\bar{x}_i,\bar{u}_f,\bar{w}_d\right]$ be the resultant non-discrete state, so that b_d may now be interpreted as the probability of going from state $\bar{x}_i$ to state $\bar{x}_B$ in one transition.

The value of the optimum cost function, $I\left[\bar{x}_B,k+I\right]$ must be found by interpolation between the nearest discretised states.

If $\bar{x}$ only contains one element, say the level in a one reservoir system, and $\bar{x}_G$ and $\bar{x}_L$ are the two nearest discretised levels, with $\bar{x}_G > \bar{x}_B > \bar{x}_L$, then using linear interpolation :

$$I\left[\bar{x}_B,k+I\right] = \frac{(\bar{x}_B-\bar{x}_L)}{(\bar{x}_G-\bar{x}_L)} \times I\left[\bar{x}_G,k+I\right] + \frac{(\bar{x}_G-\bar{x}_B)}{(\bar{x}_G-\bar{x}_L)} \times I\left[\bar{x}_L,k+I\right]$$

Thus

$$C_d \times b_d = l\left[\bar{x}_i,\bar{u}_f,\bar{w}_d\right] \times b_d + \frac{(\bar{x}_B-\bar{x}_L)}{(\bar{x}_G-\bar{x}_L)} \times I\left[\bar{x}_G,k+I\right] \times b_d + \frac{(\bar{x}_G-\bar{x}_B)}{(\bar{x}_G-\bar{x}_L)} \times I\left[\bar{x}_L,k+I\right] \times b_d$$

It can be seen that the probability $\frac{(\bar{x}_B-\bar{x}_L)}{(\bar{x}_G-\bar{x}_L)} \times b_d$ may be added to any

direct probability p_{Gi} which may exist under one of the $\bar{w}_d$ and $\frac{(\bar{x}_G-\bar{x}_B)}{(\bar{x}_G-\bar{x}_L)} \times b_d$ may be added to any direct probability p_{Li} which may exist, without altering the solution of the problem.

If no transfer is made from state $\bar{x}_i$ to some discrete state $\bar{x}_J$ either directly or indirectly by apportioning part of the probability of going to a non-discrete state, then p_{Ji} is set to zero.

Equation 7.II may now be written :

$$E(\bar{x}_i(k),\bar{u}_f(k)) = \sum_{d=I}^{r} I\left[\bar{x}_i,\bar{u}_f,\bar{w}_d\right] \times b_d + \sum_{J=I}^{a} I\left[\bar{x}_J,k+I\right] \times p_{Ji}$$

The value of $\sum_{d=I}^{r} I\left[x_i,\bar{u}_f,\bar{w}_d\right] \times b_d$ is identical to that $\sum_{J=I}^{a} I_{Ji} \times p_{Ji}$

so that no restructuring is necessary, and

$$q_i = \sum_{d=I}^{r} I\left[\bar{x}_i,\bar{u}_f,\bar{w}_d\right] \times b_d .$$

7.9. Thelong term equations

Now that it has been demonstrated how to set up the $\bar{P}$ matrix, all the equations which will be derived from equations 7.I5 may be used with complete generality.

If it is assumed that a rule has been fixed for every $\bar{x}_i$, so that no minimisation is required then the optimum costs, I, at every stage now become the only possible costs.

Let $v_i(k)$ be the total expected cost of going to the end of the process starting from state $\bar{x}_i$ at stage k.

Then

$$v_i(k) = I\left[\bar{x}_i, k\right]$$

Therefore, for a fixed control system, equations 7.I5. may be written as :

$$v_i(k) = q_i + \sum_{J=I}^{a} v_J(k+I) \times p_{Ji} \qquad i = I, a \quad . \quad . \quad . \quad \text{Equ. 7.I6.}$$

In matrix notation equation 7.I6 is

$$\bar{v}(k) = \bar{q} + \bar{v}(k+I) \times \bar{P} \quad . \quad . \quad . \quad \text{Equ. 7.I7.}$$

where the i^{th} column of $\bar{P}$ contains the p_{Ji} and $\bar{v}$ and $\bar{q}$ are row vectors.

If n is defined as the number of stages to go to the end of the process equations 7.I7. may be written as

$$\bar{v}(n+I) = \bar{q} + \bar{v}(n)\bar{P} \quad . \quad . \quad . \quad \text{Equ. 7.I8.}$$

For the determination of the equations for the solution of the long term process it is now convenient to break down equations 7.I8 into the twelve distinct monthly processes involved.

Let d be the number of discretised states in one month so that d = a/I2, and let t be a stage variable, where a stage is twelve months.

Consider the month of January :

The probability transition matrix to describe a transfer from a state in January of one year to January in the following year is given by

$$\bar{R}_I = \bar{M}_{I2} x \bar{M}_{II} x \bar{M}_{I0} x \bar{M}_9 x \bar{M}_8 x \bar{M}_7 x \bar{M}_6 x \bar{M}_5 x \bar{M}_4 x \bar{M}_3 x \bar{M}_2 x \bar{M}_I$$

and the immediate costs $\bar{Q}_I$ are given by

$$\bar{Q}_I = \bar{q}_I + \bar{M}_I(\bar{q}_2 + \bar{M}_2(\bar{q}_3 + \bar{M}_3(\bar{q}_4 + \bar{M}_4(\bar{q}_5 + \bar{M}_5(\bar{q}_6 + \bar{M}_6(\bar{q}_7 + \bar{M}_7(\bar{q}_8 + \bar{M}_8(\bar{q}_9 + \bar{M}_9(\bar{q}_{I0} + \bar{M}_{I0}(\bar{q}_{II} + \bar{M}_{II}(\bar{q}_{I2} + \bar{M}_{I2}) \ldots))$$

where the subscripts represent the month numbers.

The structure of equations 7.I8 may now be used to describe the yearly transitions for the January matrix :

$$\bar{v}_I(t+I) = \bar{Q}_I + \bar{v}_I(t)\bar{R}_I \qquad \text{. . . . Equ. 7.I9.}$$

For the manipulation of these matrix equations it is convenient to use the technique of z - transformation, where the z-transform of some function $f(n)$ is defined as $F(z) = \sum_{n=0} f(n)z^n$. A table of z - transforms for common functions is given in Appendix 2.

Let the z-transform of the vector $\bar{v}(t)$ be $\bar{y}(z)$.

Taking the z-transformations of equation 7.I9. we obtain

$$z^{-I}\left[\bar{y}_I(z) - \bar{v}_I(0)\right] = \frac{I}{I-z}\bar{Q}_I + \bar{y}_I(z)\bar{R}_I$$

$$\bar{y}_I(z) - z\,\bar{y}_I(z)\bar{R}_I = \frac{z}{I-z}\bar{Q}_I + \bar{v}_I(0)$$

$$\bar{y}_I(z)\left[\bar{I} - z\bar{R}_I\right] = \frac{z}{I-z}\bar{Q}_I + \bar{v}_I(0)$$

where $\bar{I}$ is the unit matrix.

$$\bar{y}_I(z) = \frac{z}{I-z}\bar{Q}_I\left[\bar{I} - z\bar{R}_I\right]^{-I} + \bar{v}_I(0)\left[\bar{I} - z\,\bar{R}_I\right]^{-I} \qquad \text{. . . Equ. 7.20}$$

It is shown in Appendix I that

$$\left[\bar{I} - z\,\bar{R}\right]^{-I} = \frac{I}{I-z}\bar{S} + \bar{J}(z) \qquad \text{. . . Equ. A.I.20}$$

where $\bar{S}$ is a stochastic matrix whose J^{th} column is the vector of limiting state probabilities if the system is started in the J^{th} state, and $\bar{J}(z)$ is a set of matrices representing the transient behaviour of the system. It is also shown in Appendix I that all columns of $\bar{S}$ are identical. Therefore, substituting Equations A.I.20. in Equations 7.20 we obtain

$$\bar{y}_I(z) = \underset{(I-z)2}{\underline{z}}\ \bar{Q}_I\bar{S}_I + \underset{(I-z)}{\underline{z}}\ \bar{Q}_I\bar{J}_I(z) + \underset{(I-z)}{\underline{I}}\ \bar{v}_I(0)\bar{S}_I + \bar{v}_I(0)\bar{J}_I(z)$$

By inspection of this equation for $\bar{y}_I(z)$ it is possible to identify the components of $\bar{v}_I(t)$. The term $\underset{(I-z)2}{\underline{z}}\ \bar{Q}_I\bar{S}_I$ represents a ramp of magnitude $\bar{Q}_I\bar{S}_I$.

Partial fraction expansion (see Appendix 3) shows that the term $\underset{(I-z)}{\underline{z}}\ \bar{Q}_I\bar{J}_I(z)$ represents a step of magnitude $\bar{Q}_I\bar{J}_I(I)$ plus geometic terms that tend to zero as t becomes very large.

The quantity $\underset{(I-z)}{\underline{I}}\ \bar{v}_I(0)\bar{S}_I$ is a step of magnitude $\bar{v}_I(0)\bar{S}_I$ and $\bar{v}_I(0)\bar{J}_I(z)$ represents geometric components that vanish for large t.

Thus, for large t,

$$\bar{v}_I(t) = t\bar{Q}_I\bar{S}_I + \bar{Q}_I\bar{J}_I(I) + \bar{v}_I(0)\bar{S}_I$$

If a now vector, $\bar{G}_I$, with components G_J is defined by $\bar{G}_I = \bar{Q}_I\bar{S}_I$ then

$$\bar{v}_I(t) = t\bar{G}_I + \bar{Q}_I\bar{J}_I(I) + \bar{v}_I(0)\bar{S}_I \qquad \text{Equ. 7.2I.}$$

The quantity G_J is equal to the sum of the immediate costs Q_J weighted by the limiting state probabilities that result if the system is started in the J^{th} state, or

$$G_J = \sum_{i=I}^{d} Q_i\ s_{iJ}$$

where s_{iJ} is an element of $\bar{S}_I$ and represents the long term probability of being in state $\bar{x}_i$ if the system was started in state $\bar{x}_J$.

The G_J is also the average cost per transition if the system is started in

the J^{th} state and allowed to make many transitions. It may be called the average cost of the J^{th} state. Inspecting equation 7.2I. it can be seen that G_J is the slope of the asymptotic of $v_J(t)$.

Since all states in the same Markov chain have identical columns in the $\bar{S}_I$ matrix, such states all have the same average cost.

Therefore,

$$G = \sum_{J=I}^{d} Q_J \pi_J(\infty)$$ where $\bar{\pi}(\infty)$ is the limiting state probability distribution.

The vectors $\bar{Q}_I \bar{J}_I(I)$ and $\bar{v}_I(0)\bar{S}$ represent the intercepts at t=0 of the asymptotes of $\bar{v}_I(t)$.

Let v_i be the asymptotic intercept of $v_i(t)$ so that for large t

$$v_i(t) = tG_i + v_i \qquad i = I,d \qquad . \quad . \quad . \quad . \quad \text{Equ. 7.22.}$$

The row vector with components v_i may be designated by $\bar{v}_I$ so that

$$\bar{v}_I = \bar{Q}_I \bar{J}_I(I) + \bar{v}_I(0)\bar{S}_I$$

Equations 7.22. then become

$$\bar{v}_I(t) = t\bar{G}_I + \bar{v}_I$$

If the system is completely ergodic then all $G_i = G$ and G is the average cost of the process rather than the average cost of a state so that equations 7.22. become

$$v_i(t) = tG + v_i \qquad i = I,d \qquad . \quad . \quad . \quad . \quad \text{Equ. 7.23.}$$

Now that it has been demonstrated that the costs involved in a Markov process tend to increase at a constant rate for large t, a method will be described in which direct use is made of this fact to obtain the optimal long term control.

7.IO. Determination of the optimum long term control

We have deduced the equations 7.23,

$$v_i(t) = tG + v_i, \qquad i = I,d \quad \text{for large } t,$$

and the dynamic programming equation is

$$\bar{v}_I(t+I) = \bar{Q}_I + \bar{v}_I(t)\bar{R}_I \qquad . \quad . \quad . \quad . \qquad \text{Equ. 7.I9}$$

for a fixed set of controls.

Equations 7.I9. may be expanded as

$$v_i(t) = Q_i + \sum_{J=I}^{d} v_J(t-I)R_{Ji} \qquad i = I,d \qquad . \quad . \quad . \quad . \qquad \text{Equ. 7.24}$$

Substituting Equations 7.23 in equations 7.24 we obtain the equations

$$tG + v_i = Q_i + \sum_{J=I}^{d} \left[(t-I)G + v_J\right] R_{Ji}, \qquad i = I,a.$$

$$tG + v_i = Q_i + (t-I)G\sum_{J=I}^{d} R_{Ji} + \sum_{J=I}^{d} v_J R_{Ji}$$

Since $\sum_{J=I}^{d} R_{Ji} = I$ we have

$$G + v_i = Q_i + \sum_{J=I}^{d} v_J R_{Ji} \qquad i = I,d \qquad . \quad . \quad . \quad . \qquad \text{Equ. 7.25}$$

It can be seen that these are d simultaneous equations but (d+I) unknowns. Therefore, one of the v_i, say v_d, may be set to zero and the equations solved for G and the relative values of the v_i. It will be shown that only the relative values are required to find the optimal controls for the system.

Having deduced equations 7.25. by using a yearly transition matrix for January, $\bar{R}_I$, rather than using the complete set of monthly transitions described by $\bar{P}$ it is now possible to revert to the monthly structure of the problem given by equations 7.I6.

$$v_i(k) = q_i + \sum_{J=I}^{a} v_J(k+I) \times p_{Ji}, \text{ for } i = I,a \quad . \quad . \qquad \text{Equ. 7.I6.}$$

or, in n notation,

$$v_i(n+I) = q_i + \sum_{J=I}^{a} v_J(n)p_{Ji} \quad \text{for } i=I,a \qquad \ldots \text{ Equ. 7.26}$$

In normal iterative dynamic programming, if we have a policy with n stages to go then we find the best alternative in the i^{th} state with (n+I) stages to go by minimising

$$v_i(n+I) = \min_{\substack{u_f \\ f=I,m}} \left\{ q_i f + \sum_{J=I}^{a} v_J(n)p_{Ji}f \right\} \qquad \ldots \text{ Equ. 7.27.}$$

Inspecting the matrix form of equations 7.26. given by equations 7.I8. it can be seen that the equations may be written in the $\bar{M}$ notation as

$$\begin{aligned}
\bar{v}_I(n+I) &= \bar{q}_I + \bar{v}_2(n)\bar{M}_I \\
\bar{v}_2(n+I) &= \bar{q}_2 + \bar{v}_3(n)\bar{M}_2 \\
\bar{v}_3(n+I) &= \bar{q}_3 + \bar{v}_4(n)\bar{M}_3 \\
\bar{v}_4(n+I) &= \bar{q}_4 + \bar{v}_5(n)\bar{M}_4 \\
\bar{v}_5(n+I) &= \bar{q}_5 + \bar{v}_6(n)\bar{M}_5 \\
\bar{v}_6(n+I) &= \bar{q}_6 + \bar{v}_7(n)\bar{M}_6 \qquad \ldots \text{ Equ. 7.28.} \\
\bar{v}_7(n+I) &= \bar{q}_7 + \bar{v}_8(n)\bar{M}_7 \\
\bar{v}_8(n+I) &= \bar{q}_8 + \bar{v}_9(n)\bar{M}_8 \\
\bar{v}_9(n+I) &= \bar{q}_9 + \bar{v}_{10}(n)\bar{M}_9 \\
\bar{v}_{I0}(n+I) &= \bar{q}_{I0} + \bar{v}_{11}(n)\bar{M}_{I0} \\
\bar{v}_{II}(n+I) &= \bar{q}_{II} + \bar{v}_{12}(n)\bar{M}_{II} \\
\bar{v}_{I2}(n+I) &= \bar{q}_{I2} + \bar{v}_1(n)\bar{M}_{I2}
\end{aligned}$$

where the subscripts refer to the month numbers.

Considering only the last equation of this set

Equations 7.26. may now be reduced to

$$v_i(n+I) = q_i + \sum_{J=I}^{d} v_J(n)p_{Ji} \qquad \text{for } i = a-d+I,a$$

Now for large n we may use the values obtained for large t in equation 7.23. so that equation 7.27 may be written

$$v_i(n+I) = \underset{\substack{f=I,m \\ f}}{\text{Min}} \left\{ q_i f + \sum_{J=I}^{d} \left(\frac{nG}{I2} + v_J\right) p_{Ji} f \right\} \quad \text{for } i=a-d+I,a$$

Since $\sum_{J=I}^{d} p_{Ji} f = I$ then $\frac{nG}{I2}$ and any constant in the v_J terms

(from $\bar{v}_I(0)\bar{S}_I$) become independent of the decision $\bar{u}_f$.

Thus, we may minimise

$$q_i f + \sum_{J=I}^{d} v_J \, p_{Ji} f \quad \text{for each state} \quad i=a-d+I,a$$

and furthermore we may use the relative values of the v_J, obtained from equations 7.25., for the policy that was used up to stage n.

Having obtained the relative values of the components of $\bar{v}_I$ from equations 7.25. these may be back-substituted in equations 7.28. to find the other $\bar{v}_J$, J=2,I2. and a similar minimisation to the above may be carried out for all other states $\bar{x}_i$, i=I,(a-d).

It can be shown[12] that if the minimisation procedure yields different controls for any state at stage (n+I) from those which were used to assemble equations 7.25. for stage n, then the new controls, if applied over a long period of time, would lead to a lower average cost per transition for the process than the previous controls.

Equations 7.25. may be regarded as performing the same function as a simulation of a long historic sequence. If a simulation was run starting from a known state and using fixed controls for every state then at the end of the run we would have the total cost of operating the system over a long period of time. If the total cost is divided by the number of transitions made then we obtain the average cost per transition.

However, with a simulation, we have no direct method of correcting the controls to obtain a better policy, whereas the solution of equations 7.25. in conjunction with the minimisation procedure leads to a method of

obtaining a better policy with a lower average loss per transition.

If a different set of controls is found by the minimisation procedure than those that were used to set up the probability matrix, $\bar{P}$, used to assemble equations 7.25. then the new controls are used in the formation of a new $\bar{P}$ matrix. The equations 7.25. are again solved using the new $\bar{P}$ matrix, and the minimisation procedure applied to the values obtained. This method is repeated until the controls found by the minimisation are the same as those used in the construction of equations 7.25. At this point the best policy has been found.

7.11. The discounted case

So far the costs incurred in future time periods have been given equal weight to cost incurred at the present time, and do not take into account the interest payable on borrowed capital and the rate of inflation. This section will deal with the case where discounting of future costs is important.

Following Howard's notation, let β be the value at the beginning of a transition interval of a unit cost incurred at the end of the interval, so that $\beta = \frac{I}{1+I}$ where I is the interest rate.

Now equation 7.19. the dynamic programming recurrence equation for the no discounting case can be written as

$$\bar{v}_I(t+I) = \bar{Q}_I + \beta\bar{v}_I(t)\bar{R}_I \qquad \text{Equ. 7.30.}$$

where $\beta\bar{v}_I(t)$ represents the values at stage t+I of the expected costs of going from any state $\bar{x}$ with t stages to go to the end of the process.

Taking the z-transforms of equation 7.30. we obtain

$$z^{-I}\left[\bar{y}_I(z)-\bar{v}_I(0)\right] = \frac{I}{I-z}\bar{Q}_I + \beta\bar{y}_I(z)\bar{R}_I$$

$$\therefore \bar{y}_I(z)-\bar{v}_I(0) = \frac{z}{I-z}\bar{Q}_I + \beta z\,\bar{y}_I(z)\bar{R}_I$$

$$\therefore \bar{y}_I(z)(\bar{I}-\beta z\bar{R}_I) = \frac{z}{I-z}Q_I + \bar{v}_I(0)$$

$$\text{or } \bar{y}_I(z) = \frac{z}{I-z}\bar{Q}_I(\bar{I}-\beta z\bar{R}_I)^{-I} + \bar{v}_I(0)(\bar{I}-\beta z\bar{R}_I)^{-I} \qquad \text{Equ. 7.31.}$$

From the discounting case we know that

$$(\bar{I}-z\bar{R}_I)^{-I} = \frac{I}{1-z}\bar{S}_1+\bar{J}_1(z) \quad \text{where } \bar{S}_I \text{ is the matrix of limiting state}$$

probabilities and $\bar{J}_1(z)$ is the transformed matrix of transient components of the system.

Hence it is obvious that we may write

$$(\bar{I}-(\beta z)\bar{R}_1)^{-1} = \frac{1}{1-(\beta z)}\bar{S}_1 + \bar{J}_1(\beta z) \quad . \quad . \quad . \quad \text{Equ. 7.32.}$$

Equation 7.3I. can then be written as

$$\bar{y}_1(z) = \frac{z}{(1-z)}\bar{Q}_1\left[\frac{1}{1-\beta z}\bar{S}_1 + \bar{J}_1(\beta z)\right] + \bar{v}_1(0)\left[\frac{1}{1-\beta z}\bar{S}_1 + \bar{J}_1(\beta z)\right]. \quad \text{Equ.7.33.}$$

Inspecting equation 7.33 we see that the term

$\bar{v}_1(0)\frac{1}{1-\beta z}\bar{S}_1$ tends towards zero for large t since the inverse transform of

$\frac{1}{1-\beta z}$ is β^t

Similarly,

$\bar{v}_1(0)\bar{J}_1(\beta z)$ represents terms that decay to zero since the eigenvalues of $\bar{R}_1$ are all less than or equal to unity and so the eigenvalues of $\beta\bar{R}_1$ are certain to be less than unity.

$\frac{z}{1-z}\bar{Q}_1\frac{1}{1-\beta z}\bar{S}_1$ represents a step component of magnitude $\frac{1}{1-\beta}\bar{Q}_1\bar{S}_1$ plus a term that tends to zero as t becomes large.

$\frac{z}{1-z}\bar{Q}_1\bar{J}_1(\beta z)$ represents a step of magnitude $Q_1\bar{J}_1(\beta)$ plus terms that decay to zero as t becomes large

Therefore, for large t

$$\bar{v}_1(t) = \frac{1}{1-\beta}\bar{Q}_1\bar{S}_1 + \bar{Q}_1\bar{J}_1(\beta) = Q_1\frac{1}{1-\beta}\bar{S}_1 + \bar{J}_1(\beta) \quad . \quad . \quad . \quad \text{Equ. 7.34.}$$

From equation 7.34 it can be seen that, for large t, the present day costs of operating the system with t stages to go to the end of the process become constant.

Therefore, for large t, letting $\bar{v}_1$ be the constant values, we may write equation 7.30. as

$$\bar{v}_1 = \bar{Q}_1 + \beta\bar{v}_1\bar{R}_1 \quad . \quad . \quad . \quad \text{Equ.7.35.}$$

These equations are solved for the $\bar{v}_1$ and the calculations then proceed as for the no-discounting case with the probabilities in the monthly transition matrices, $\bar{M}$, multiplied by $\beta/12$.

Note that it is not now necessary to set one of the v_i to zero to solve the equations, since there are now only d unknowns and d equations. The values of v_i thus obtained are the absolute values of operating the process over a long period of time starting in state $\bar{x}_i$ under a fixed set of controls.

Chapter 8

The Application of Dynamic Programming to the Design & Operation of Reservoir Systems

8.1. The Explicit Stochastic Approach

The only explicit stochastic dynamic programming multi-source models in the literature at the present time are those investigated by the Water Research Association. The earliest models are those described by Schweig and Cole in 1968 (17), which require the assumption of very simple streamflow dependencies.

The problems involved the determination of the long term operating rules for a system comprising two linked reservoirs meeting a common demand. Draw off was permitted from either of the reservoirs directly to supply and transfers were allowed from the smaller to the larger reservoir. Because of the lengthy calculations involved in these problems, only a flow diagram of the necessary computer program was given, but an example of the method of computation was described for a simpler case involving a small surface reservoir operated in conjunction with a major underground source. Transfers of water between the sources were considered redundant. The method of solution used was the conventional reverse time sequence stochastic dynamic programming value iteration approach, as described in Chapter 7. Schweig and Cole assumed that the policy had converged to an optimum when, in all seasons of the year, the revised control rules at step N of the iteration were identical to those at step (N-12). However, a paper by Burley and Cole (20) describing the same work stated that the results of the value iteration approach needed to be tested further as O'Kane (41) had demonstrated that the necessary criterion of convergence described above might not be sufficient,

and that application of Howard's policy iteration method (12) might be called for.

8.2. Simple Reservoir - Aquifer System - Value Iteration

With this in mind, the author developed a set of programs to investigate the behaviour of the policies and costs in the long term situation. The water resource system used was the same as that in the example given by Schweig and Cole, but a two season model only was considered.

Size of surface reservoir = 30000 units
States (levels) used for dynamic program = 0, 10000, 20000, 30000.
Maximum release from Aquifer in any season = 10000 units

Inflows:-

Season 1		Season 2	
Inflow	Prob.	Inflow	Prob.
4000	0.3	500	0.2
5000	0.6	5000	0.5
7000	0.1	8000	0.3

Demands:-

Season 1	Season 2
12500	15000

Possible decisions at any state and stage:-

	Season 1		Season 2		
Decision Reference Number	Release from Reservoir	Release from Aquifer	Release from Reservoir	Release from Aquifer	Decision Ref. Number
1	12500	0	12500	2500	5
2	7500	5000	7500	7500	6
3	2500	10000	2500	10000	7
4	0	10000	0	10000	8

Costs:- Release from reservoir = £40/unit
Release from aquifer = £80/unit
Deficit to demand = £100/unit
Discount factor = 0.985

The release costs were calculated on the size of the release aimed for and not on the actual release possible for a given reservoir level.

As in Cole's example, the problem was considered to be solved when the policies obtained for one year were identical to those obtained in the previous year. If one considers a solution surface in multi-dimensional space this means that the solution obtained occurs at the nearest low point on the surface to the given starting conditions.

The results obtained for this example are given below. It was found that the fourth year's solutions were the same as those for the third year. Therefore the process was considered to have optimised and the procedure was terminated.

Table of Optimum Decisions Obtained in Each Year

(Numbers represent decision reference numbers)

	Season 1				Season 2			
Reservoir level	0	10000	20000	30000	0	10000	20000	30000
Year 1	3	2	1	1	6	5	5	5
Year 2	3	3	1	1	7	6	5	5
Year 3	3	3	2	1	7	6	5	5
Year 4	3	3	2	1	7	6	5	5

The CPU time taken on the ICL 1900 computer was 16 seconds but only 8 seconds of this was used in the problem solution, the rest being used in system organisation and program compilation prior to loading the program for execution. The total time involved was 1 minute 35 seconds.

No attempt was made in this first program to store any information calculated for the first year and which could be used by every iteration in order to make the program more efficient. Any basic costing information required in later years had to be re-calculated.

The program instructions occupied 4672 words of core store after consolidation and the maximum total amount of core store used in execution was 14,016 words.

8.3. Long Period Value Iteration

This first program was also checked by a hand calculation to ensure that no errors of logic were occurring in the computer calculation. Having found that the program was correct and knowing the time and core store taken, it was decided to run a second value iteration program which would continue past the apparent policy convergence in years 3 and 4 to produce costs and decisions for fifty years. It was thought that this program would also provide an indication of the speed of convergence of the discounted costs to the constant values predicted by the theory.

In fact, it was found that cost convergence did not take place within 50 years and not even within 80 years, as later tried. However, it could be seen from the costs printed out for each year that the costs were increasing at a decreasing rate as the number of years became greater. Figure 8.1. shows a graph of discounted costs versus number of years for the case of starting from an empty reservoir in season 1.

It was interesting to note that after about 60 years' iterations a change over occurred and the costs showed that it was better to start in a given state in season 2, thus having an extra season to go to the end of the process, than to start in the same state in season 1.

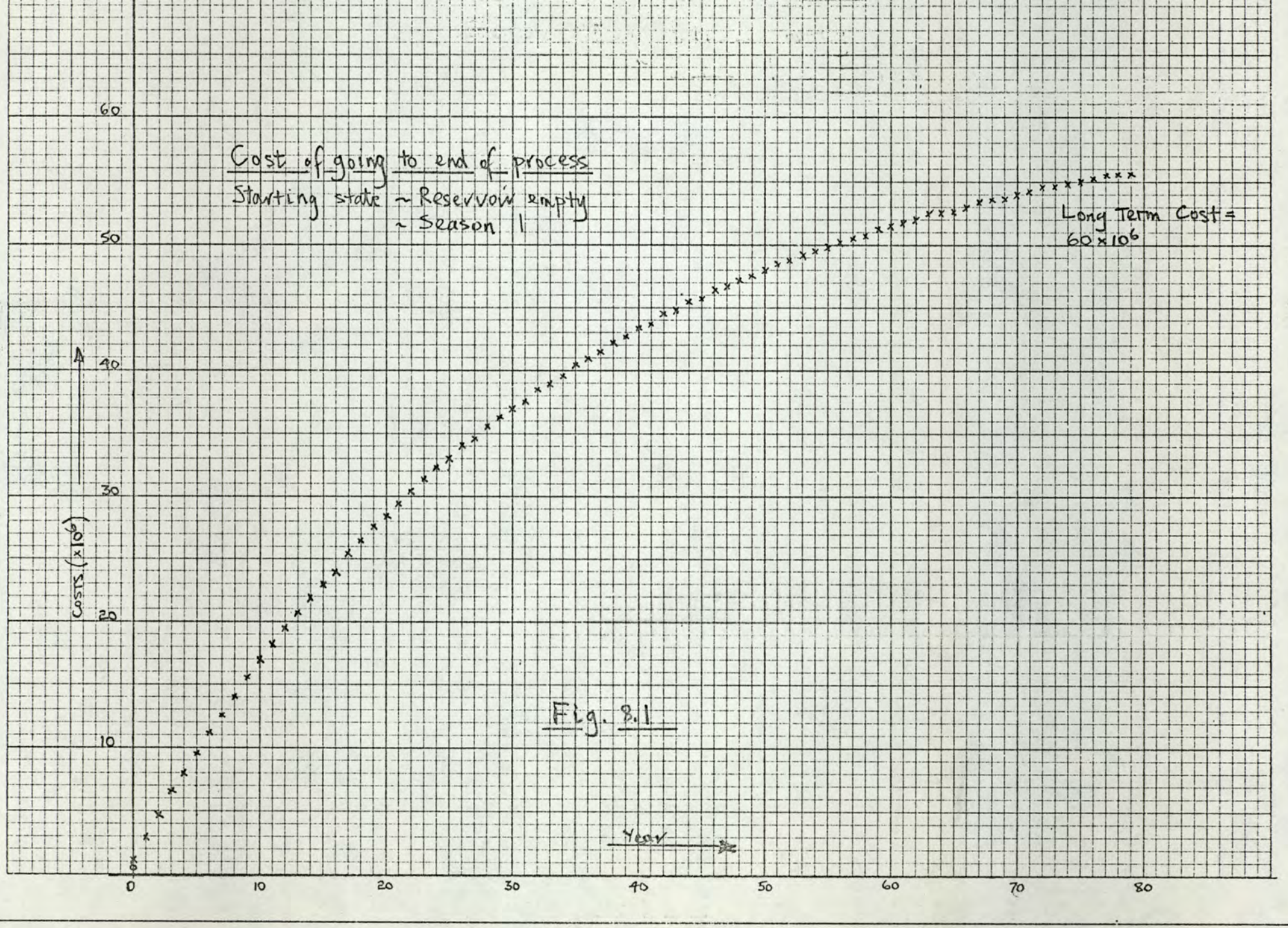

Cost of going to end of process
Starting state ~ Reservoir empty
~ Season 1
Long Term Cost =
60 × 10^6
Cost (×10^6)
Year
0
10
20
30
40
50
60
70
80

Fig. 8.1

The computer program for these long period calculations was made more efficient by storing the immediate costs and the coefficients involved in interpolating between state costs for intermediate state transitions, as described in Chapter 7, when they were calculated in the first year's iterations.

The C.P.U. time for the 50 year case was 39 seconds, 22 seconds of which were used in the problem calculations. The C.P.U. time per one year's iterations was 0.333 seconds and the initial time to read in data and calculate the information needed in every year was 5.4 seconds. The total computer time for 50 years was 2 minutes 5 seconds.

As well as showing the speed of convergence of costs and providing calculation times, the long period computations showed that the policy, which had apparently converged after 3 years when the first program was run, changed in year 5 and again in year 6. From year 6 onwards the policy was the same for all years.

Table of Optimum Decisions Obtained in Each Year (Year 4 Onwards)
(Numbers represent decision reference numbers)

	Season 1				Season 2			
Reservoir Level	0	10000	20000	30000	0	10000	20000	30000
Year 4	3	3	2	1	7	6	5	5
Year 5	3	3	2	1	7	6	6	5
Year 6	3	3	3	1	7	6	6	5
Year 7	3	3	3	1	7	6	6	5

8.4. The Policy Iteration Method

With the information that a long term steady state policy exists

and knowing that at least one sub optimum solution may be found, it is now possible to apply Howard's policy iteration method and to verify in practice that this method does in fact converge monotonically to the true optimum policy.

The information general to every year was again stored for computational efficiency. Because of the small size of the problem it was possible to store all the data in the computer central core. The method, as described in Chapter 7, consists of performing yearly value iterations alternating with solving the set of equations describing the system.

It was found that the optimum policy was reached with 4 equation solutions and 5 value iterations, the last being to check that the optimum has been reached.

The optimum policy found by this method agreed with the long term optimum policy found in the 50 year iteration, thus verifying that Howard's policy iteration method is applicable to the water resources problem.

One of the intermediate policies found by this method in moving towards the optimum was the sub optimum policy found by the first trial where iteration was discontinued after 4 years when the policies for the third and fourth years were the same. However, it is not necessary for this to occur and probably only happens because of the simplicity of the problem.

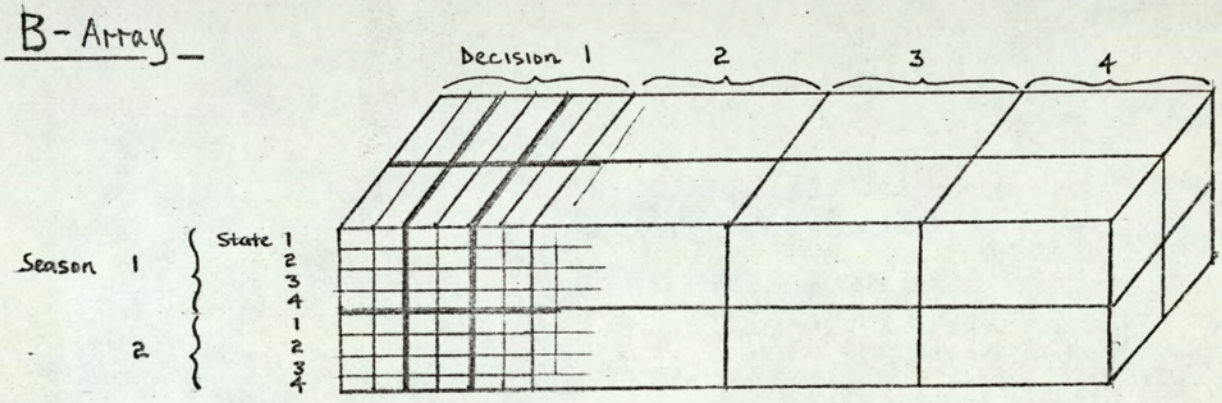
The following table shows the long term costs and associated decisions found by the program. The costs are the equation solutions plus one year's iteration.

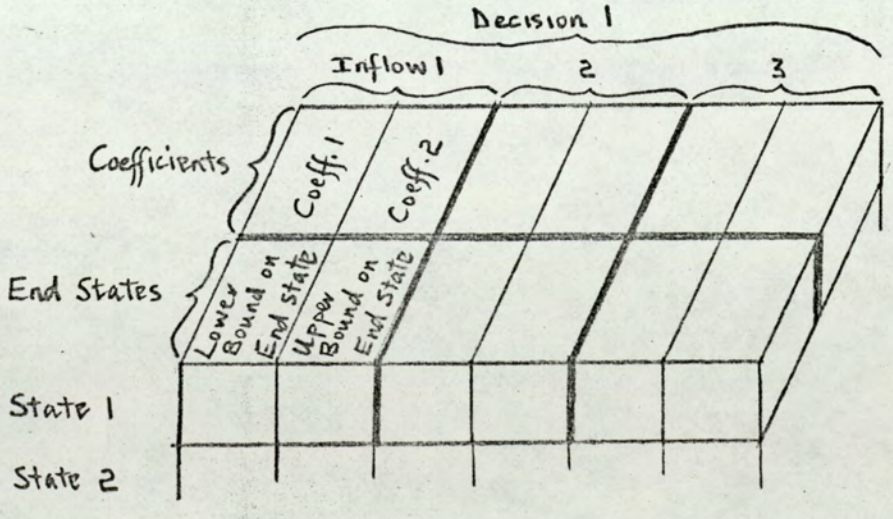
		Optimum Costs				Decision Nos.	Comments
Res. Level		0	10 000	20 000	30000		
Iteration 1	Season 1	116500	740000	700000	700000	3211	Decisions based on immediate costs only. Solve equations with these decs.
	Season 2	1947055	1537742	1219444	1189500	6555	
Iteration 2	Season 1	64238521	63734759	63281511	62830886	3332	Costs produced from previous equation solutions. Now solve with these new decisions to give next long term costs + 1 Iteration.
	Season 2	64055854	63571590	63125761	62703828	7666	
Iteration 3	Season 1	61251380	60747674	60335141	59944510	3321	
	Season 2	61113533	60638936	60235763	59837769	7655	
Iteration 4	Season 1	61229556	60725031	60308129	59909924	3331	Optimum decisions have been found at this point but must carry on before this is known.
	Season 2	61091843	60615600	60209372	59809372	7665	
Iteration 5	Season 1	61198864	60693247	60271952	59872792	3331	
	Season 2	61061353	60583254	60173511	59773511	7665	

These results show that Howard's policy iteration method does in fact find the optimum long term policy and that each step in the method yields a better solution than the previous one. The long term costs obtained from the equation solutions also give an idea of the number of years necessary to reach a stationary cost situation when compared to the costs obtained after 80 years with the value iteration approach.

The computer program took 19 seconds of C.P.U. time, of which 9 seconds were used in the calculations and 10 seconds in system management. The total execution time was 1 minute 23 seconds. The program instructions occupied 10816 words of core store and the total store occupied at run time was 18,240 words. The program, as before, stored general information calculated at iteration 1 for use in later iterations.

8.5. Method of Storage for Policy Iteration





Typical Unit

For any discretised state considered there are three possible inflows with their relative probabilities and therefore three possible end states. Since the end states may not be discretised states they are recorded by storing the nearest discretised states to the intermediate state together with the two interpolation coefficients necessary. If the end state coincides with a discrete state this is taken as the lower bound with a coefficient of 1, and the next highest discrete state is considered to be the higher bound with a coefficient of zero.

For convenience the coefficients obtained are all multiplied by the discount factor.

C - Array

			Dec. 1	Dec. 2	Dec. 3	Dec. 4
Season 1	State	1	Cost			
		2				
		3				
		4				
2		1				
		2				
		3				
		4				

The C array stores the immediate transition costs for each state and each possible decision. These costs are worked out and stored at the same time as the information is stored in the B array.

MATI Array

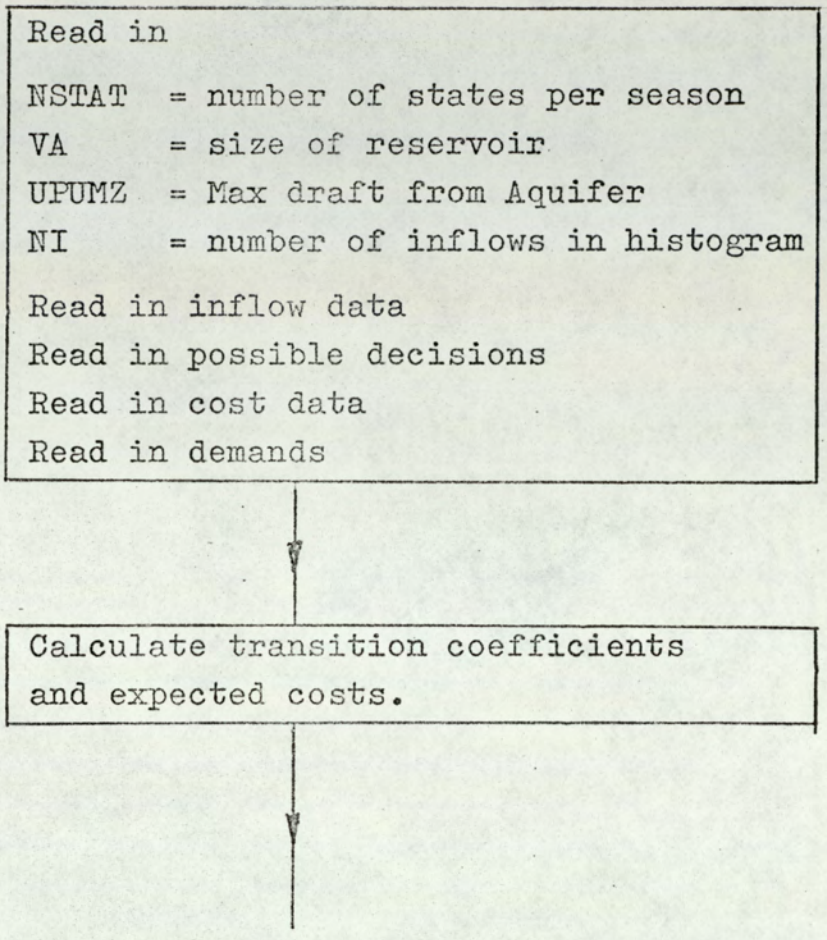
		End States 1	2	3	4
Starting States	State 1	Coeff.1	Coeff.2	Coeff.3	Coeff.4
	2	. . .	. . .	. . .	. . .
	3				
	4	. .	.	. . .	. .

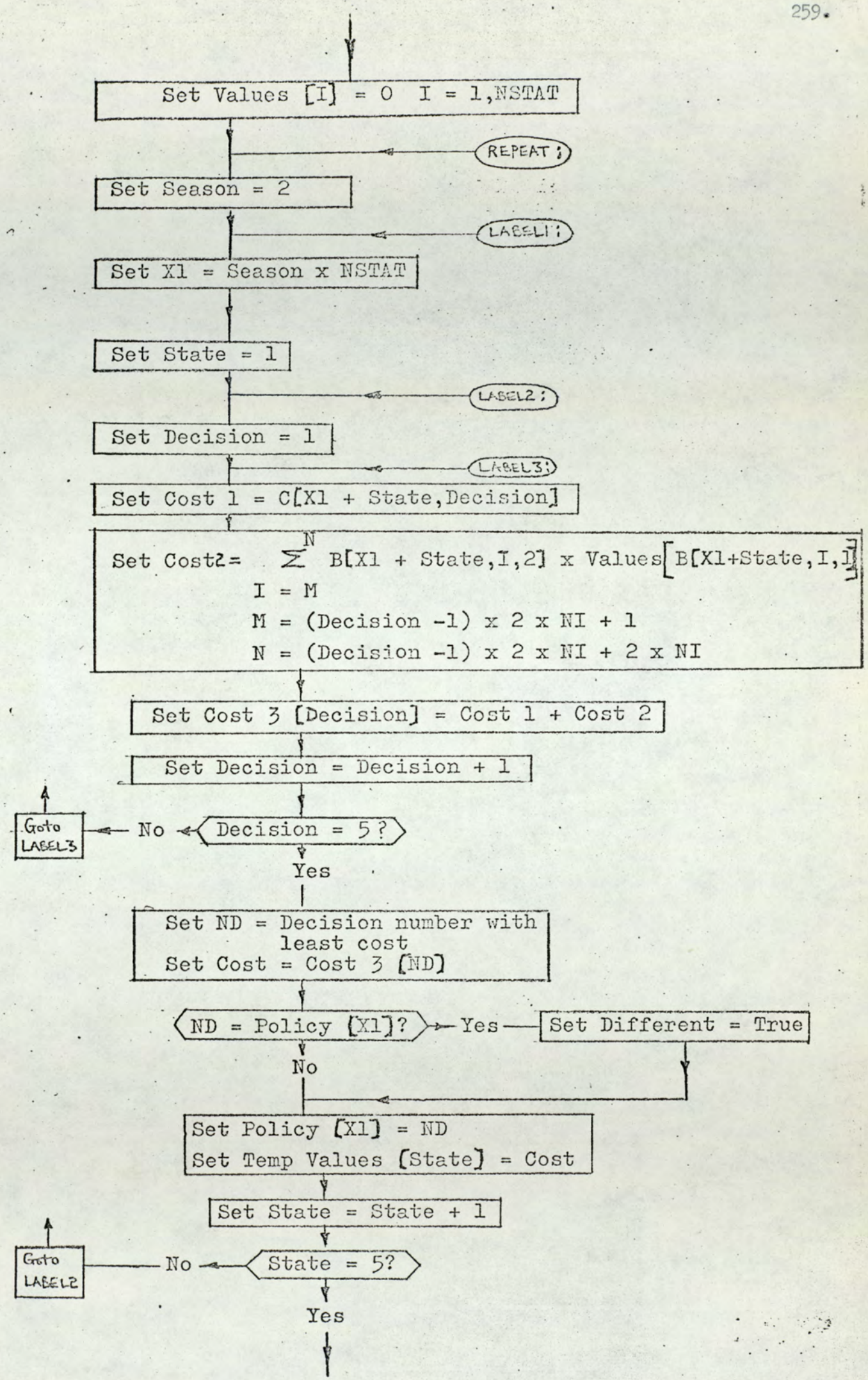
The MATI matrix is used to hold the transition coefficients for the chosen set of policies for each state of season 1.

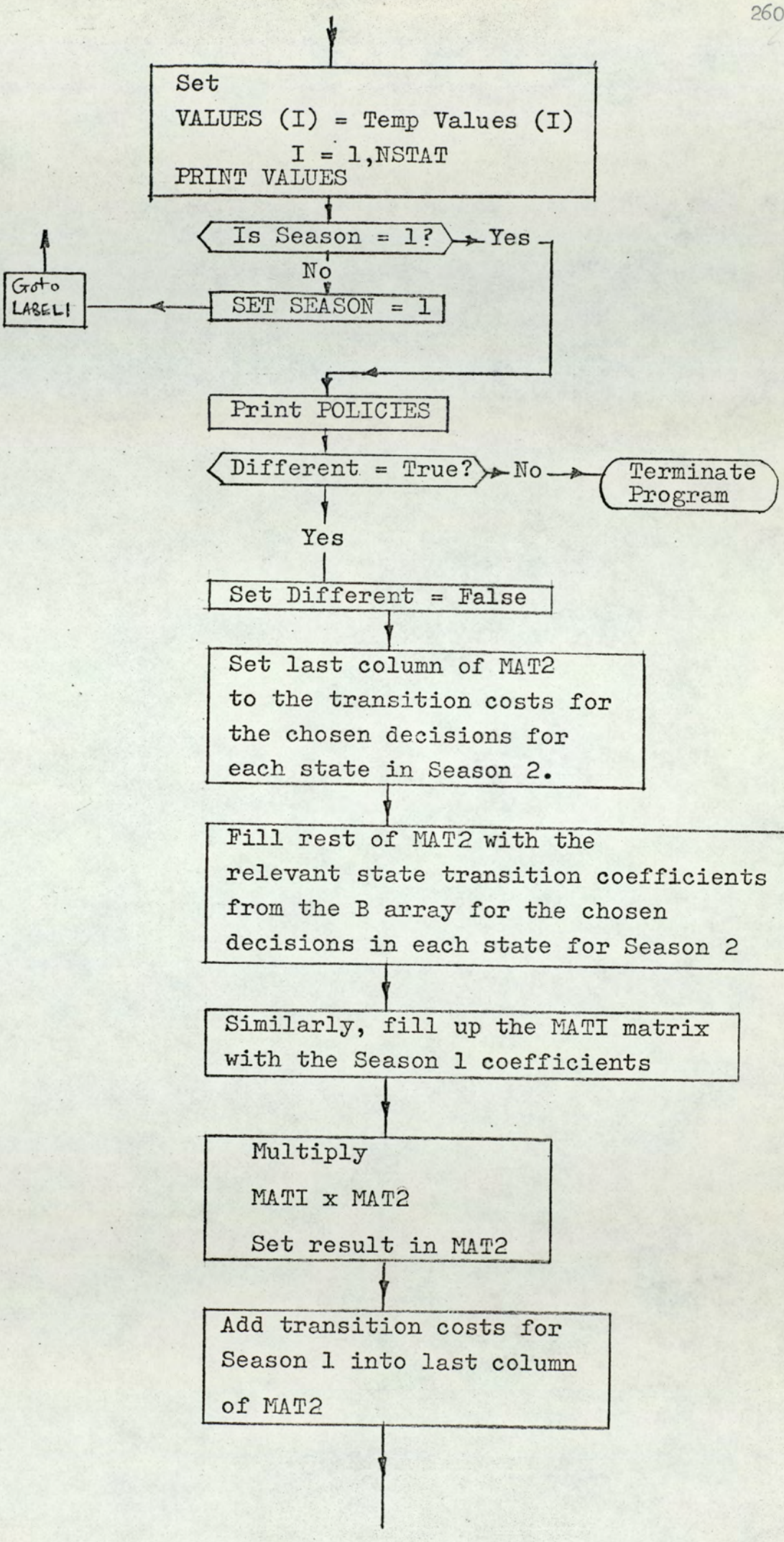
MAT2 is a matrix similar to MATI but with an additional column. The matrix is used to hold transition coefficients for season 2 and the last column is used to hold the average transition costs.

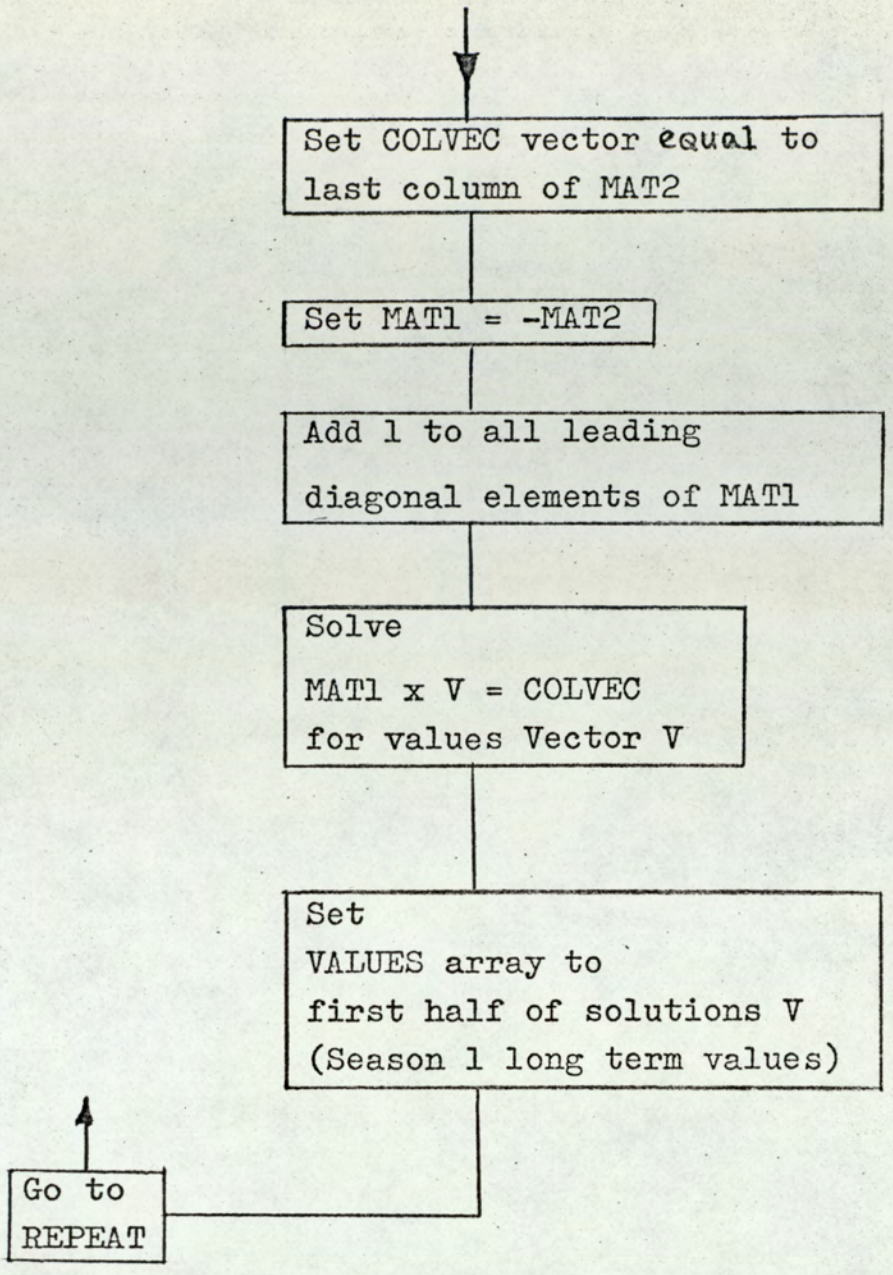
These coefficients are obtained by adding the coefficients in the B array, for the chosen decisions and given state, into the matrix element describing the same end state as that related to the coefficient in the B array.

8.6. Flow Diagram for Policy Iteration Program




Set Values [I] = 0 I = 1,NSTAT
REPEAT;
Set Season = 2
LABEL1;
Set X1 = Season x NSTAT
Set State = 1
LABEL2;
Set Decision = 1
LABEL3;
Set Cost 1 = C[X1 + State,Decision]
Set Cost2 = Σ B[X1 + State,I,2] x Values[B[X1+State,I,1]]
I = M
M = (Decision -1) x 2 x NI + 1
N = (Decision -1) x 2 x NI + 2 x NI
Set Cost 3 [Decision] = Cost 1 + Cost 2
Set Decision = Decision + 1
Decision = 5 ?
No
Goto LABEL3
Yes
Set ND = Decision number with least cost
Set Cost = Cost 3 [ND]
ND = Policy [X1]?
Yes
Set Different = True
No
Set Policy [X1] = ND
Set Temp Values [State] = Cost
Set State = State + 1
State = 5?
No
Goto LABEL2
Yes


Set
VALUES (I) = Temp Values (I)
I = 1,NSTAT
PRINT VALUES
Is Season = 1?
Yes
No
SET SEASON = 1
Goto LABEL1
Print POLICIES
Different = True?
No
Terminate Program
Yes
Set Different = False
Set last column of MAT2 to the transition costs for the chosen decisions for each state in Season 2.
Fill rest of MAT2 with the relevant state transition coefficients from the B array for the chosen decisions in each state for Season 2
Similarly, fill up the MATI matrix with the Season 1 coefficients
Multiply
MATI x MAT2
Set result in MAT2
Add transition costs for Season 1 into last column of MAT2


Set COLVEC vector equal to last column of MAT2
Set MAT1 = -MAT2
Add 1 to all leading diagonal elements of MAT1
Solve MAT1 x V = COLVEC for values Vector V
Set VALUES array to first half of solutions V (Season 1 long term values)
Go to REPEAT

8.7. Dynamic Programming Applied to Two Stochastic Reservoirs

Because of the success of the policy iteration method in the simple case of one reservoir combined with an aquifer, it was decided to attempt to apply this method to a system of two finite stochastic reservoirs meeting a common demand. This system is general enough to cover many practical situations and is able to include the simple aquifer system by treating the second reservoir as the aquifer. This system is also easily adaptable to include the multi-demand or resource allocation problem, and the stochastic demand case which occurs with a regulating reservoir.

The system is similar to that described by Burley and Cole, except that transfers between the reservoirs were not considered since this does not alter the basic method and it was only the efficiency of the various dynamic programming methods which was under investigation. Serial correlation of inflows in each reservoir was allowed for in exactly the same way as that used by Schweig and Cole, the probability distribution of inflows to each source in any month depending upon whether the previous month's inflow was considered to be higher or lower than the average for that month. Of course, more complicated methods of accounting for antecedent flows can be incorporated but this only increases the computation time.

Four discretised levels were used for each reservoir, making the number of states in each reservoir eight in all, where the state comprises a combination of level and an index indicating whether the previous month's inflow was higher or lower than

average. Thus, the total number of combinations of states in the system was sixty-four. Five possible inflows, with their corresponding probabilities were allowed for each month. The objective function allowed for costing the releases from each reservoir, the spills and the deficits to supply. However, in the example used, only deficit was assigned a unit cost greater than zero, so that the dynamic program would minimise deficits only. This is comparable to conventional methods where the probability of emptiness is minimised, except that the dynamic program not only tries to minimise the probability of emptiness but also the size of the deficits. This might mean that more deficits occur but are not as disastrous as, say, one major deficit. If this is not acceptable to some engineers then it is extremely easy to rewrite the objective function to minimise only the number of deficits occurring.

8.8. Value Iteration - Two Stochastic Reservoirs

In order to investigate the rate of convergence to the correct optimum long term policy it was first decided to carry out a value iteration procedure which would be continued until several years'consecutive iterations showed the same results. The inflow and demand data used for these examples was taken from the Celyn/Brenig system described in Chapter 2. Both reservoirs were assumed to have 30000 units capacity, where one unit was a cusec-day. The demands used for each month were the average monthly effective releases to maintain a flow of 450 cusecs in the Celyn/Brenig system. The release decisions were to take 0%, 25%, 50%, 75% or 100% of the demand from Celyn in any month.

Demands (cusec-days)

Jan.	Feb.	Mar.	April	May	June	July	Aug.	Sept.	Oct.	Nov.	Dec.
220	420	850	1620	3330	3850	4160	3460	3010	1760	360	140

Unit Costs:- Release from Celyn URA = £0/unit
Release from Brenig URB = £0/unit
Spill from Celyn USA = £0/unit
Spill from Brenig USB = £0/unit
Deficit cost UDF = £100/unit
Present worth or discount factor PWF (monthly) = 0.985 = 18%/annum

Inflow Data (cusec-days)

a) Brenig

Month	Average Monthly Inflow	Monthly Inflow Histograms			
		Low Previous Inflow		High Previous Inflow	
		Inflow	Probability	Inflow	Probability
January	1000	330	0.30	520	0.13
		870	0.17	790	0.22
		1030	0.26	1050	0.39
		1310	0.17	1540	0.22
		1990	0.10	1930	0.04
February	740	220	0.27	180	0.20
		550	0.15	410	0.20
		880	0.27	670	0.25
		1230	0.19	910	0.25
		1650	0.12	1510	0.10
March	550	230	0.26	170	0.13
		440	0.26	310	0.26
		620	0.26	460	0.35
		1120	0.17	670	0.13
		1840	0.05	1180	0.13
Cont.					

Month	Average Monthly Inflow	Monthly Inflow Histograms			
		Low Previous Inflow		High Previous Inflow	
		Inflow	Probability	Inflow	Probability
April	420	110	0.21	190	0.29
		250	0.31	400	0.24
		430	0.24	510	0.17
		750	0.21	640	0.18
		1080	0.03	870	0.12
May	330	110	0.33	100	0.05
		290	0.22	180	0.32
		430	0.22	270	0.26
		670	0.15	370	0.16
		800	0.08	480	0.21
June	230	80	0.42	90	0.20
		150	0.23	140	0.20
		260	0.12	220	0.20
		470	0.19	330	0.25
		890	0.04	550	0.15
July	210	60	0.43	120	0.19
		140	0.27	160	0.25
		250	0.13	300	0.25
		410	0.10	370	0.19
		710	0.07	490	0.12
August	390	60	0.37	170	0.38
		190	0.27	350	0.18
		350	0.23	540	0.06
		570	0.10	850	0.19
		940	0.03	1150	0.19
September	540	70	0.33	240	0.12
		190	0.20	440	0.25
		480	0.17	620	0.18
		600	0.13	980	0.38
		860	0.17	1360	0.07
Cont.					

Month	Average Monthly Inflow	Monthly Inflow Histograms			
		Low Previous Inflow		High Previous Inflow	
		Inflow	Probability	Inflow	Probability
October	770	120	0.14	260	0.21
		320	0.14	560	0.17
		410	0.13	810	0.21
		880	0.50	970	0.16
		1460	0.09	1490	0.25
November	970	250	0.14	670	0.38
		350	0.18	960	0.21
		720	0.18	1420	0.25
		960	0.23	1690	0.12
		1440	0.27	1980	0.04
December	970	330	0.11	400	0.20
		640	0.31	700	0.15
		800	0.12	980	0.10
		1080	0.31	1190	0.40
		1780	0.15	1690	0.15

b) Celyn

Month	Average Monthly Inflow	Monthly Inflow Histograms			
		Low Previous Inflow		High Previous Inflow	
		Inflow	Probability	Inflow	Probability
January	6140	1470	0.17	3170	0.17
		3440	0.14	5160	0.24
		6100	0.34	6280	0.14
		8420	0.24	7130	0.17
		12060	0.11	8430	0.28
February	3960	410	0.11	740	0.25
		2460	0.34	2970	0.28
		4630	0.22	4920	0.25
		6380	0.22	6280	0.12
		7870	0.11	7620	0.10
Cont.					

Month	Average Monthly Inflow	Monthly Inflow Histograms			
		Low Previous Inflow		High Previous Inflow	
		Inflow	Probability	Inflow	Probability
March	2690	1210	0.21	1070	0.16
		1900	0.18	1890	0.29
		2610	0.36	2460	0.26
		4930	0.18	3330	0.19
		7780	0.07	4520	0.10
April	2200	480	0.08	750	0.20
		940	0.13	1800	0.30
		1530	0.28	2410	0.20
		2590	0.38	3630	0.25
		4280	0.13	5310	0.05
May	1810	480	0.18	640	0.16
		950	0.29	1130	0.16
		1650	0.24	1610	0.20
		2580	0.15	2500	0.40
		4260	0.14	3680	0.08
June	1600	400	0.27	620	0.27
		870	0.31	1020	0.15
		1420	0.15	1670	0.23
		2410	0.12	2860	0.19
		3490	0.15	3910	0.16
July	2390	510	0.34	360	0.10
		1420	0.13	1780	0.42
		2440	0.32	2540	0.24
		4430	0.10	4080	0.19
		7020	0.11	6600	0.05
August	2530	590	0.32	920	0.34
		1860	0.13	1950	0.14
		2640	0.18	2910	0.19
		3420	0.21	3950	0.14
		5040	0.16	5210	0.19
Cont.					

Month	Average Monthly Inflow	Monthly Inflow Histograms			
		Low Previous Inflow		High Previous Inflow	
		Inflow	Probability	Inflow	Probability
September	2430	510	0.36	770	0.29
		1640	0.28	1910	0.16
		2680	0.18	3570	0.39
		4130	0.14	5390	0.13
		7270	0.04	7170	0.03
October	4350	1120	0.21	1670	0.23
		2430	0.24	3410	0.23
		4210	0.21	5360	0.27
		5880	0.24	7300	0.19
		8490	0.10	11380	0.08
November	4650	1680	0.12	850	0.07
		2730	0.22	2980	0.37
		3870	0.31	4540	0.19
		6250	0.22	6090	0.15
		8260	0.13	9110	0.22
December	6120	2190	0.09	2110	0.17
		4240	0.31	3860	0.25
		6000	0.20	6840	0.42
		8260	0.31	9380	0.12
		11460	0.09	12430	0.04

Since Schweig and Cole suggested in their paper that the computer running time might be high for this type of problem it was decided to write the computer program for the S.R.C. Atlas computer to avoid a monopoly of the ICL 1900 machine with excessive running times. The program stored all input data in the central core of the computer but any cost or system transition data calculated in the first iteration was not stored for use in later iterations but had to be re-calculated when required.

The program run time was limited to 288000 instructions and the compilation store was 81K. The store occupied at execution time was 20K and 13 blocks of magnetic storage were used. The actual number of instructions carried out was 288078. Exactly 14 years of iterations were performed at approximately 20000 instructions each, where 10000 instructions take about one minute of run time. However, only half the policy for iteration 14 was printed out before the program time terminated. It was thought at first that the system had optimised with the yearly policies after and including iteration eight the same, but when the policy iteration method had been carried out and the optimum policy was found to be different from that at iteration 13, the value iteration results were investigated more thoroughly. It was found that iterations 8,9,10, 11 and 12 produced the same policy but the policy changed in iteration 13, but still not producing the optimum. Since the policy was the same for five consecutive iterations it is thought that the policy was very near the optimum or that the system was such that many policies could have been close to the optimum. Later investigations implied that the total reservoir storage was far too large for the inflows and demands considered, so that almost any reasonable policy would suffice.

Exactly the same program was run on the ICL 1900 computer at Aston University when it was found that 5 $^{7}/_{12}$ years' value iterations were carried out in a C.P.U. time of 1805 seconds with a total machine occupation time of 54 minutes, which gives an approximate total time per iteration of 9.7 minutes, compared to the equivalent Atlas time of 2 minutes.

A trial was then made to find the decrease in time produced by storing intermediate transition and cost data, calculated at the first year's iteration, for use in each iteration, rather than re-calculating the same data at each iteration.

The first program run on the ICL 1900 machine with stored results produced 7½ iterations with the same C.P.U. time as before and with a total time of 59 minutes. A second program was limited to 580 seconds of C.P.U. time and produced 2 5/12 iterations in a total of 19 minutes.

From these results it can be shown that one iteration absorbs about 240 seconds of C.P.U. time and about 7.86 minutes of total time. Further, the preliminary time to calculate and store the transition and cost data was almost zero. However, the latter figure may be grossly inaccurate because initial compilation time was not included in the calculations, but the other figures will be approximately correct since they are much larger than compilation time.

The comparisons show that the extra time in retrieving and filing information on discs is still preferable to recalculating it at every iteration. It should be noted that the information was stored in such a way that to carry out the complete dynamic programming calculations for one monthly stage two blocks of information containing 600 pieces of data each needed to be retrieved from disc files.

8.9. Policy Iteration - Two Stochastic Reservoirs

The policy iteration method carried out on the same data allowed for storage of intermediate calculated data since the calculations were expected to consume enough computer time with efficient programming without the necessity of recalculation of intermediate data.

The Atlas program itself occupied 12436 words of core store and the total number of instructions carried out at run time was 500178, the store used in execution being 61K.

The optimum policy was reached within four iterations where an iteration comprised a value iteration over 12 months to obtain a policy and the solution of a set of equations developed from the policy. An extra value iteration was required at the end to prove that the system had actually optimised. It was found from inspection of the results for January that the values obtained from solution of the equations at iteration 3 were not different from the final results to any practical degree, the solutions only differing in the second decimal place with values of 340 at the minimum and 261500 at the maximum.

The solutions from iteration 2 were from 25 to 75 units different from the final results, the values ranging from 381 to 261534 units.

At iteration 1, larger differences were found. These varied from 9700 to 13000 units, the solutions being in the range 12413 to 271254.

These results suggest that the first solution, based on minimising the immediate costs, is a poor one, but the fact that the solutions from iterations 2 and 3 are both close to the optimum suggests that several sub optimum policies might exist which yield very similar long term costs and that the optimum is only marginally the best policy.

8.10. Convergence of Policy Iteration

A second program was then run with a Brenig size reduced to 20000 cusec-days to investigate the convergence of the solutions. The optimum was reached in three policy iterations instead of four, which suggested that there were fewer steep ascent paths to the optimum than before. The first solutions differed from the optimum by from 7300 to 9000, the actual solution values being in the range 9278 to 198617.

The second iteration produced values from 476 to 191315 which differed from the optimum by from 17 to 29.

Since these results still seemed to show that many sub-optimum policies might exist which were close to the optimum, it was decided to reduce the reservoir sizes and to increase the demands so that the optimum policy choice would be much more critical.

8.11. A Critical System

Brenig was reduced to 15000 cusec-days and Celyn to 12000 cusec-days, while the demand was increased by 50%.

The optimum was again reached in only three iterations, the first solutions being in the range 991503 to 1444985, which differed from the optimum by from 401000 to 550000. This result showed

a substantially higher percentage difference range between the first solution and the optimum than had been found before. This confirmed that the optimum choice was more critical than before.

The second iteration yielded solutions in the range 444054 to 1045137, with differences from the optimum being from 1080 to 1630, which may be considered minor differences when compared to the solutions.

The three programs described above lead one to conclude that solutions close to the optimum are reached within only two or three iterations, but the rate of convergence to the optimum after this stage depends on the sensitivity of the system. Thus, when relatively large sources of water are present the dynamic program may spend several iterations in achieving the absolute optimum policy when a sub-optimum policy might be adequate for all practical purposes, but if the sources are small compared with the demand, the optimum policy is found quickly.

From the number of instructions carried out at execution time in the above programs an estimate of the number of instructions per full iteration, which includes one year's value iteration and one solution to a set of simultaneous equations, was found to be 115000.

The run with Brenig 15000, Celyn 12000, was investigated for policy changes between the iterations.

Iteration 2 started with a policy which was different to the final policy by 15 decisions different by one decision interval, one different by two intervals and two different in four intervals. The total number of decisions in a policy is 768. Therefore, only 18 in 768, or 2.3% of decisions, were different to the optimum.

The policy used by iteration 1 was the immediate cost minimisation policy and this was also compared to the optimum. It was found that 41 decisions were one interval, ten decisions were two intervals, 38 decisions were three intervals and 190 decisions were four intervals from the optimum, giving a total of 279 differences or 36.3%. Most of the differences, 193, were concentrated in the last few months of the year, when the greatest inflows occur, and the decisions are not so critical.

8.12. Stochastic Demands

A further program with reservoir sizes of 30000 units was run using five stochastic demands for each month instead of using the monthly means. The optimum policy showed that in November, December and January when the inflows are high and demands smaller, since they are regulating demands, more water tends to be taken from Celyn, which has the higher rates of inflows, than before. Presumably, this allows Brenig to fill and Brenig is then used in the summer months as more of a standby supply. This is to be expected since it is logical to hold more water in emergency supply as the system demands become more uncertain. Celyn is used as the working supply because it has a faster filling capability than Brenig. In the early wetter months of the year, the tendency is to use Brenig more than before and to let Celyn fill if necessary.

Stochastic Demand Data (Section 8.12.)

Month	Demand Data (cusec-days)					
January	Demand	0	220	930	2130	2650
	Probability	0.81	0.07	0.02	0.05	0.05
February	Demand	0	240	500	730	2650
	Probability	0.71	0.02	0.05	0.05	0.17
March	Demand	10	520	940	3470	4720
	Probability	0.59	0.19	0.10	0.05	0.07
April	Demand	10	680	1080	2170	5020
	Probability	0.26	0.14	0.14	0.33	0.13
May	Demand	70	1140	3050	5120	6880
	Probability	0.21	0.17	0.14	0.29	0.19
June	Demand	270	2180	4810	7280	10090
	Probability	0.29	0.14	0.33	0.19	0.05
July	Demand	550	2160	4190	6770	9910
	Probability	0.21	0.36	0.12	0.17	0.14
August	Demand	30	1200	3550	5750	9470
	Probability	0.24	0.29	0.12	0.16	0.19
September	Demand	10	660	3250	5900	9760
	Probability	0.48	0.07	0.14	0.17	0.14
October	Demand	20	660	2370	4190	7780
	Probability	0.48	0.14	0.24	0.09	0.05
November	Demand	0	560	1070	1730	2450
	Probability	0.71	0.10	0.10	0.07	0.02
December	Demand	0	130	370	930	1310
	Probability	0.71	0.10	0.07	0.05	0.07

This assures that Brenig will not be drawn down too much early in the year, but allows Celyn to fill more rapidly, in order to ensure a full Celyn reservoir at the beginning of summer.

If Celyn is still low in summer, Brenig is drawn down more than in the average demands case.

In October, just before the wetter months, if Celyn is low, more water is taken from Brenig. This is because of the fact that Brenig will be used less in winter and will therefore refill, but Celyn will be required to satisfy the winter demands more than before and must therefore be allowed to refill as much as possible before November.

In the summer months, if Celyn is low, most of the water is drawn from Brenig to allow Celyn to refill for the rest of the dry spell, but if Celyn is more than a quarter full this is mainly used, with Brenig being used only when Celyn falls below this level.

The expected costs involved in the stochastic demand case are greater than in the average demands case. This is to be expected since higher demands are involved in the demand histograms. However, because of the low probabilities of higher demands and the smoothing effect of the corresponding low demands, the effect is not too great in terms of total expected deficits at any stage, but the effect on total costs, of course, depends on the objective function used. In the problem described above, after 7 years value iteration, the deficits involved for the average demand and stochastic demands cases were 2610 and 4080 units respectively, if the system started with both reservoirs empty. The difference

is 1470 units, which is only about 6% of the average annual demand, but when a unit cost of £100 is applied, the difference in operating costs over seven years becomes £147,000, which is substantial. It is not argued here that a deficit cost of £100 per unit is realistic, but it can be seen that small changes in system variables might lead to high cost increases which are not negligible.

It is useful to point out that the value iterations in the stochastic demand case do not take any longer than the average demands case in computation time because the size of the matrices involved depend only on the number of states in the system. Once the matrices have been set up the computations are the same. It is only in the preliminary stages of the dynamic program that extra calculations are involved.

8.13. Antecedent Flows

It was noticed throughout the dynamic programming investigations that the benefit derived from incorporating high and low previous flow indices was small, since the costs and decisions found for the two cases were always similar. However, this may only apply for the particular inflows occurring in the Celyn-Brenig system. The inflow histograms tabulated previously show that the histograms for the two cases were not significantly different in most months for either reservoir.

8.14. Rate of Convergence of Stochastic Value Iteration

With the values from 14 value iterations for Celyn and Brenig sizes of 30000 units each, and the long term discounted values from the policy iteration with the same data, it is possible to show the convergence of the values for January starting from both reservoirs empty.

Values for January Starting from State of Both Reservoirs Empty

Celyn Storage 30000 cusec-days
Brenig Storage 30000 cusec-days
Average monthly demands used.

Iteration No.	Value (cost) £	Differences £
1	233580	
2	254367	20787
3	258620	4253
4	260072	1452
5	260724	652
6	261050	326
7	261221	171
8	261316	95
9	261371	55
10	261406	35
11	261428	22
12	261444	16
13	261456	12
14	261466	10
⋮	⋮	⋮
Long Term	261509	43

From the above table it can be seen that the 'long-term' costs are approached very quickly in the first few years but the curve tends to flatten out after this to approach the long term asymptotically.

8.15. Minimising Drawdown

A stochastic value iteration dynamic programming calculation was carried out for reservoir sizes of 30000 cusec-days each, with an objective function which ignored deficits but minimised the draw down in the reservoir. The optimum long term policy

found was quite different from the policy found by minimising deficits for the same input data.

Figure 8.1. shows a brief summary of the decisions reached in the deficits minimisation case for the four seasons..

Figures are fractions of demand to be taken from Celyn

Celyn Level	Brenig Level	Spring March-May	Summer June-Aug.	Autumn Sept.-Nov.	Winter Dec.-Feb.
0	0	0.50-0.75	0.75	0.75-1	0
	10000	0.50-0.75	0 - 0.25	0-0.50	0
	20000	0.50-0.75	0	0.50	0
	30000	0.25-0.50	0	0-0.25	0
10000	0	0.75	1	1	0
	10000	0.75	1	0.75	0
	20000	0.75	1	0.75	0
	30000	0.75	0-0.50	0.50-0.75	0
20000	0	1	1	1	1
	10000	0.75-1	1	1	1
	20000	0.75	1	1	1
	30000	0.50-0.75	1	0.75	0
30000	0	1	1	1	1
	10000	1	1	1	1
	20000	1	1	1	1
	30000	0.50-0.75	1	1	0

Fig. 8.1.

It can be seen that in winter when Celyn and Brenig are drawn down after the dry months, Celyn is allowed to fill first to about 15000 cusec-days, while Brenig satisfies the small winter demands. After this, when Celyn contains sufficient water to supply the spring demands, Brenig is allowed to fill, while all the demand is taken from Celyn, unless Brenig is about to overflow, when all the demand is taken from Brenig.

In the spring, if Celyn is still below one third full, the demand is shared between the reservoirs, with a tendency to take more from Celyn, so that Brenig can fill ready for the summer. In fact, all through the Celyn range, more water tends to be taken from Celyn. The aim is to have Brenig full first because it has smaller inflows. Even if Celyn is low in the spring, it is probable that it will fill sufficiently for the summer because it has high rates of inflow.

In the summer, all the demand is taken from Celyn, with Brenig as a standby supply. If Celyn is drawn down below one third full and Brenig is still high, all the water is taken from Brenig, but if Brenig is nearly emptied a higher fraction of demand is again allocated to Celyn.

In autumn, if Celyn is low after summer, the fraction of demand taken from Celyn is inversely proportional to the level of Brenig. As Celyn becomes more than half full, all the demand is taken from Celyn, so that Brenig can begin to refill if necessary, ready for supplying the low winter demands and for the following summer.

Figure 8.2. shows the policy obtained by minimising the draw-down in the reservoirs.

Celyn Level	Brenig Level	Spring March-May	Summer June-Aug.	Autumn Sept.-Nov.	Winter Dec.-Feb.
0	0	0	0	1	1
	10000	1	1	1	1
	20000	1	1	1	1
	30000	0.75	1	0.50	0
10000	0	0	0	1	1
	10000	1	1	1	1
	20000	1	1	1	1
	30000	0.75	1	0.50	0
20000	0	1	0	1	1
	10000	1	1	1	1
	20000	1	1	1	1
	30000	0.75	1	0.75	0
30000	0	1	0	1	1
	10000	1	1	1	1
	20000	1	1	1	1
	30000	1	1	1	0

Eg. 8.2.

In winter, Brenig is always allowed to fill before Celyn, all the demand being taken from Celyn. If Brenig is about to overflow, the demand is taken from Brenig. Thus, Celyn level is allowed to rise to avoid wastage of water.

In the spring, if both reservoirs are still very low, the demand is taken from Brenig but generally most or all of the demand is taken from Celyn whatever the levels in the reservoir. Thus, Brenig is again allowed to fill before Celyn, as in winter.

In the summer and autumn the demand is taken from Celyn unless Brenig is about to overflow, when it is shared.

The interesting point to note is that in summer, when Brenig is empty and Celyn is at any level, even full, the demand is all allocated to Brenig. This is in direct conflict to the minimisation of deficits but is logical for minimising draw down since the objective function in this case tries to supply the demand if sufficient water is available but, if a conflict arises between supplying demand and maximising the reservoir levels when either of the reservoirs is below about 4000 cusec-days, the maximisation takes precedence. This type of objective function may be applicable in the case of hydro-electric power generation, when water supply is not the main purpose of the reservoir, but would probably be more complicated than the linear function used here, a higher penalty being placed on higher draw downs than implied by a linear rule.

8.16 The Deterministic Approach

From the discussions in the previous sections it is evident that the solution of the stochastic control case for more than two or possibly three reservoirs or components is virtually impossible even using efficient methods of computation. However, the deterministic problem for a given system can be solved with far less effort and this leads to the possibility of increasing the size of the systems which may be analysed. A further benefit which ensues is that the intervals of discretisation of the state variables may be decreased to include more states so that a more accurate solution is obtained.

It was therefore decided to investigate the application of deterministic dynamic programming to the solution of the long term policy. If it is possible to achieve a policy which is reasonably close to the optimum by analysing deterministic results, then the method has distinct advantages over the stochastic method. The main problem in trying to apply stochastic dynamic programming to a real system is to summarise the probability distribution of inflows in a histogram and to include serial and cross correlations between the inflows in a multi-unit system while attemptingto keep the amount of data to a minimum so that the computer program does not reach unmanageable proportions. It is obvious that gross inaccuracies and over simplifications may occur, so that the optimum policy found by stochastic dynamic programming might not be the optimum for the real system.

However, if deterministic dynamic programming can be

applied, the actual historic data, with its real correlations, may be used. Even if long sequences of historic data are not available it is possible to generate synthetic inflow and demand traces by using regression analysis or auto correlation techniques. Although the statistical parameters have to be estimated from the historic data this can be done in far more detail, and thus with more accuracy, than can be allowed for in the stochastic method.

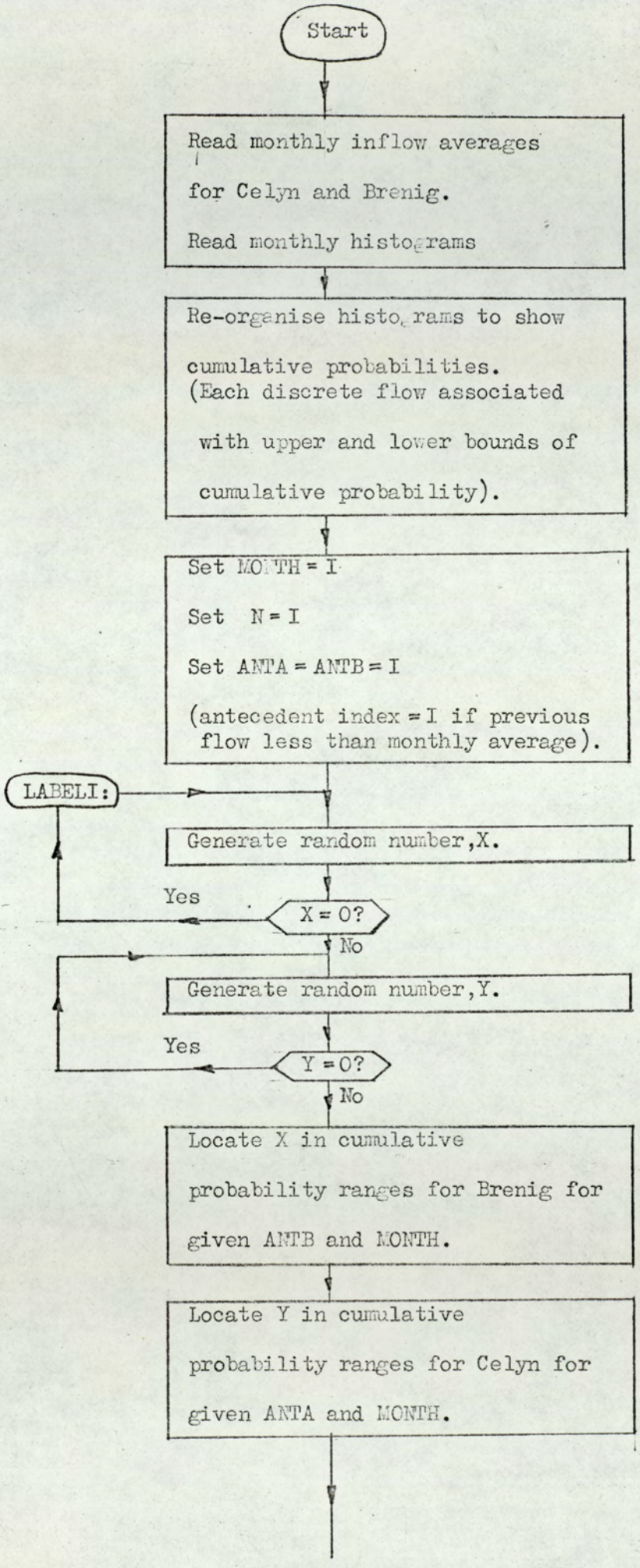
In order to be able to compare the worth of policies obtained by analysing deterministic dynamic programming results to those obtained by the stochastic method it is necessary to use data with the same statistical properties for both approaches. For this purpose the Celyn/Brenig histograms described in Section 8.8 were taken as the basic data structure. The deterministic data used was generated by a very simple procedure which only produced inflows of magnitudes equal to the discrete flows given by the histograms. In this way, it was ensured that the deterministic traces would be as close to the statistical properties of the histograms as possible.

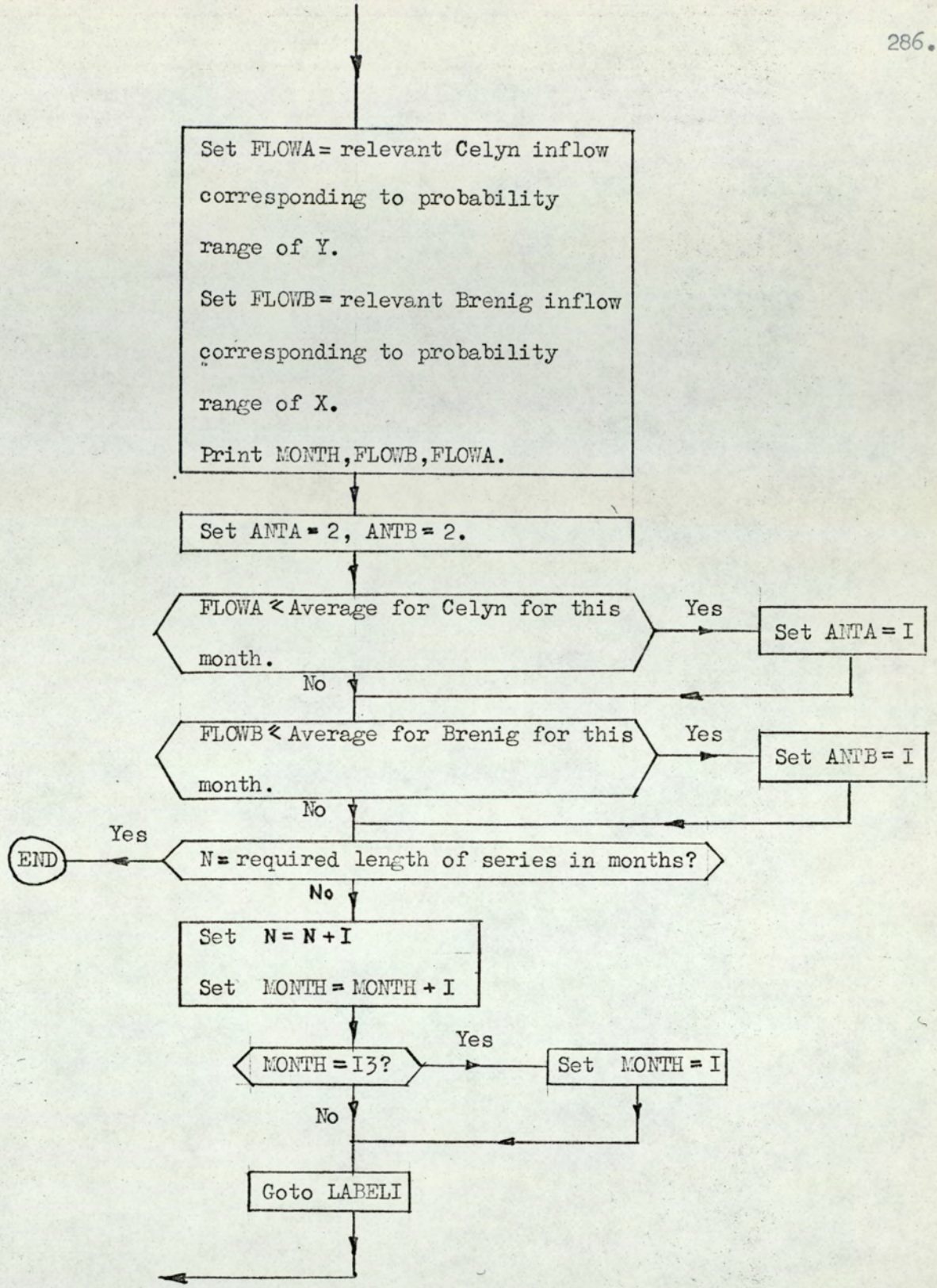
8.17 Data generation

A standard random number generating routine was employed which produced numbers between 0 and 1. The flow diagram for the complete routine is shown below and is self explanatory when read in conjunction with the data structure of section 8.8.

The demand data used were the twelve average monthly demands, also given in section 8.8.

Flow Diagram for Data Generation Routine




Set FLOWA = relevant Celyn inflow corresponding to probability range of Y.
Set FLOWB = relevant Brenig inflow corresponding to probability range of X.
Print MONTH, FLOWB, FLOWA.
Set ANTA = 2, ANTB = 2.
FLOWA < Average for Celyn for this month.
Yes
Set ANTA = I
No
FLOWB < Average for Brenig for this month.
Yes
Set ANTB = I
No
N = required length of series in months?
Yes
END
No
Set N = N + I
Set MONTH = MONTH + I
MONTH = I3?
Yes
Set MONTH = I
No
Goto LABELI

8.18 Extraction of Long Term Policy

Assuming that the best decision for the system for every state at every stage has been found by deterministic dynamic programming for a long data sequence, whether synthetic or historic, the hurdle remains of using it to operate the system in the future. The problem of the Controller when running a reservoir system is to decide upon the best way of operating the system in the immediate day, week or month, ahead based upon the data available to him at the time. One way of doing this is to look back over the historical or synthetic data sequence used for the deterministic dynamic program to try to identify one or more similar situations to the current one and implement the decision, or some average of the decisions if several similar situations were found, of those determined by the dynamic program for those situations. This kind of approach would probably require a computer to be available for searching rapidly through the past data to compare the present situation with the record of situations.

Another method would be to analyse the decisions implemented by the dynamic program for given sets of available data and attempt to extract the average or most frequently occurring decisions for given situations.

Young (28), as described in chapter 1, applied a regression analysis to his deterministic results for a simple one reservoir system. The decision for any system state and known previous inflows, as many as thought necessary, was made a function of these known variables. However, it is thought that the use of the same regression

function for all situations might lead to gross inaccuracies. It would appear better to break down the policies into a group of regression functions, say one for each combination of levels, so that the regression would only then be upon the previous inflows. It is the latter type of approach which has been used mainly in this thesis, although regression analysis was not the basis of 'averaging' the decisions. A simple average of the decisions for a particular set of results for given system variables was used at first, but a refinement of this was applied to later results. The method consisted of weighting the possible decisions at any time according to the saving in cost each produced over the worst decision which could have been made at the time, and goes a step towards the idea of investigating the effect of applying each possible decision for a particular state in turn through all time while holding the decisions for all other states at their optimum at each time.

The mechanical extraction of the policy is made by setting up an array which includes an element for every possible decision for every possible state. The costs computed in the dynamic program for each decision for a given state are added into the relevant array element. This procedure is followed at every stage of the calculation so that the costs accumulate in the array elements.

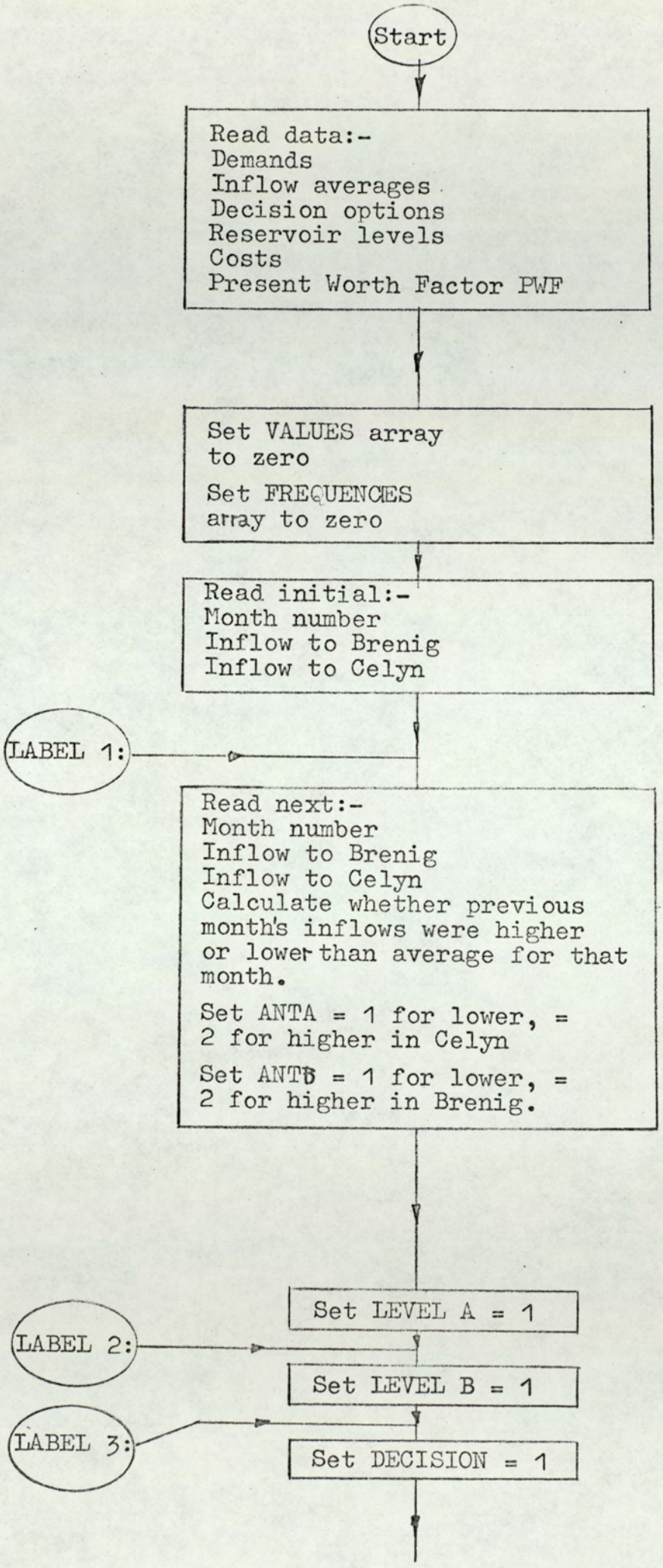
Throughout the dynamic program calculation, the optimum decisions are used as normal, the long term policy extraction being independent of the intermediate calculations. When the dynamic programming calculation is complete the

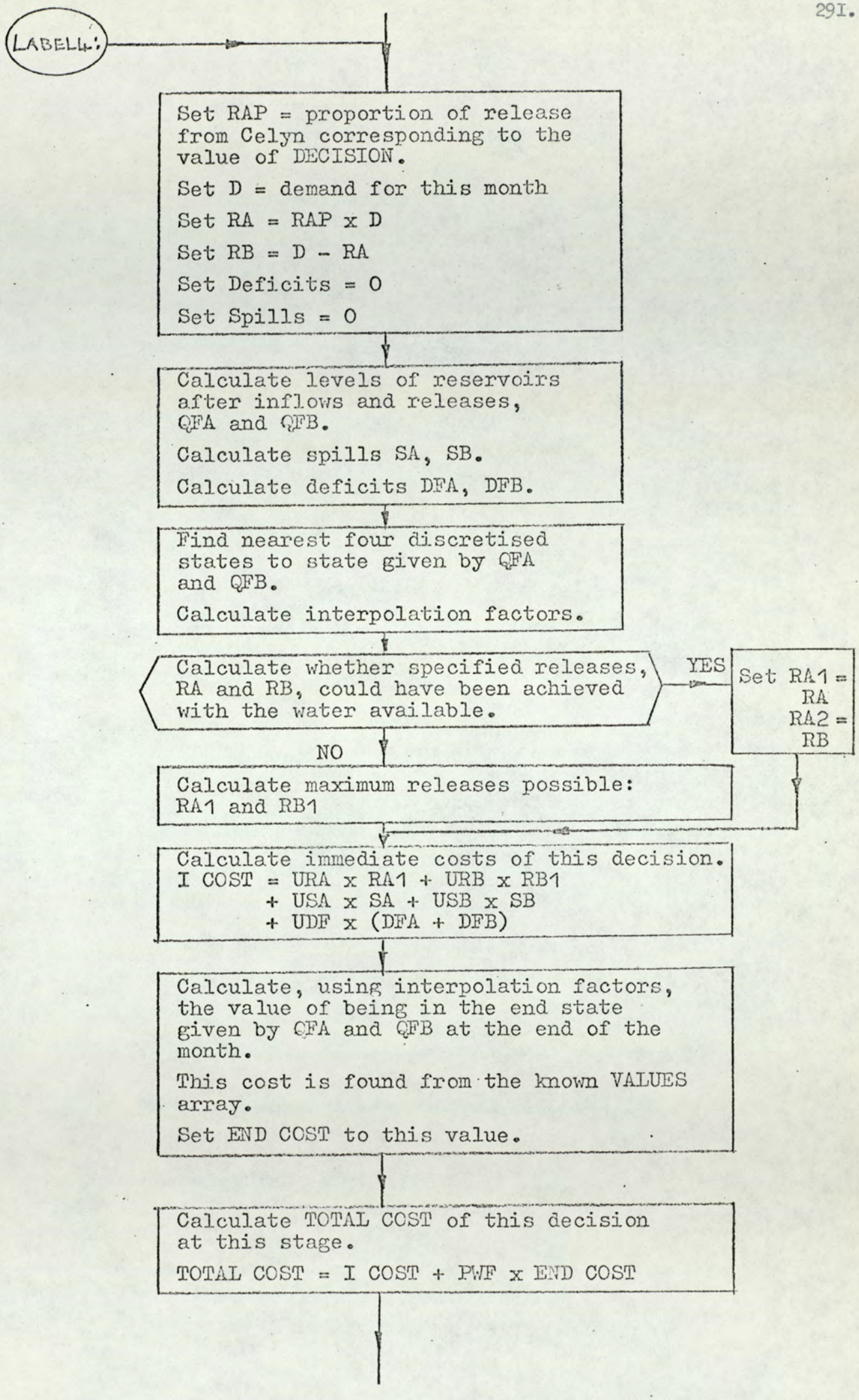
long term policy is found by choosing the optimum decision for each state as the one corresponding to the minimum accumulated cost for that state.

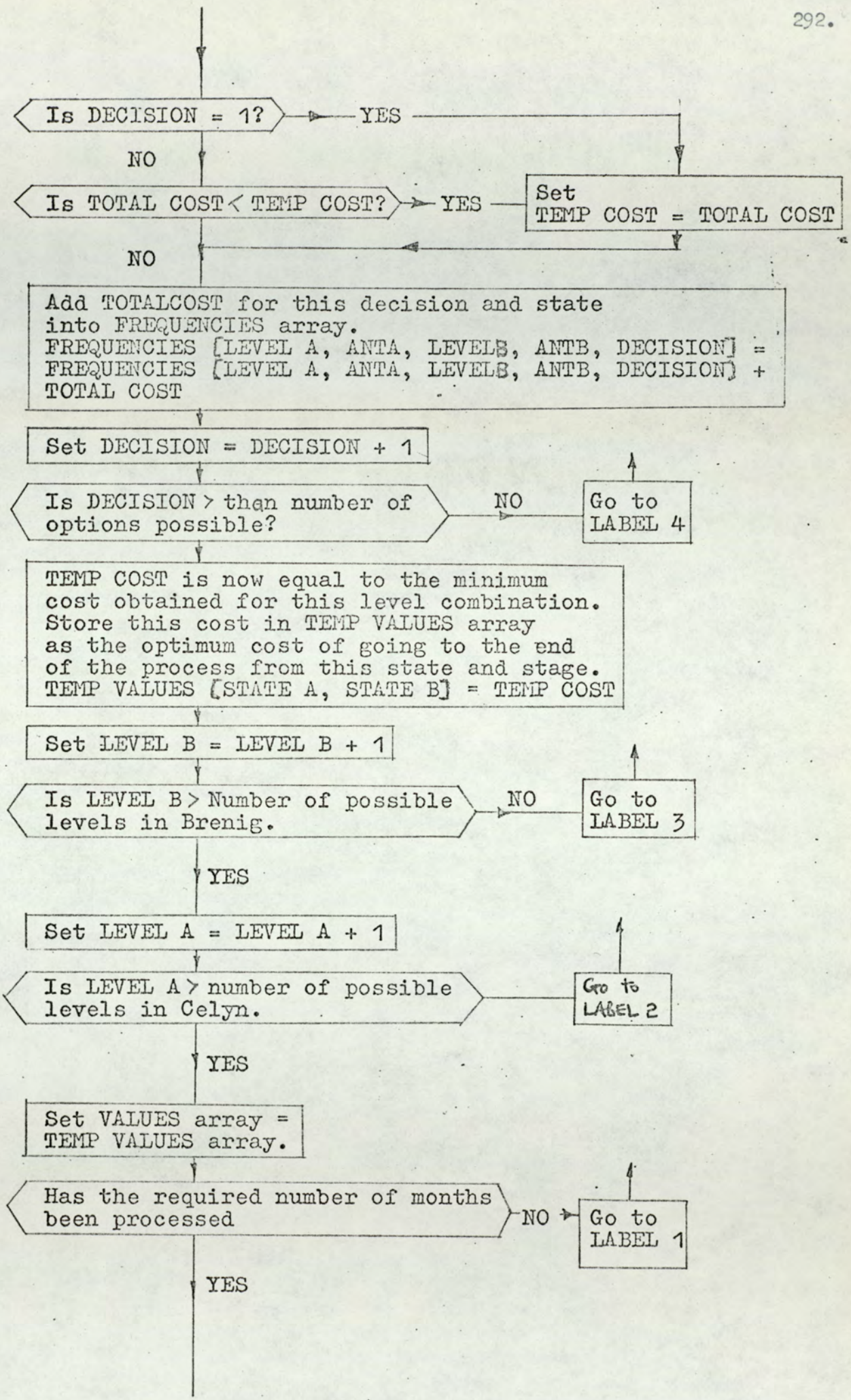
8.19 Comparison of methods of policy extraction

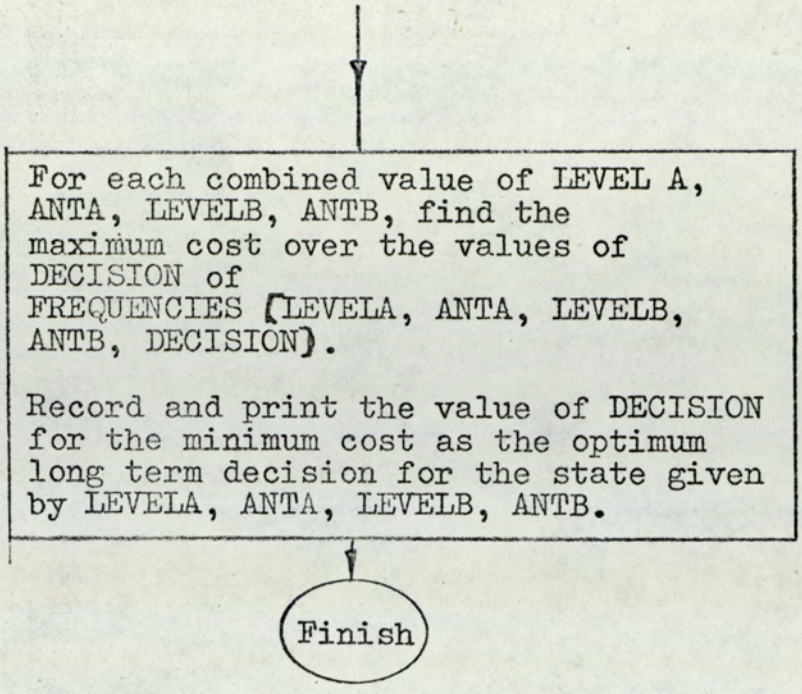
In order to verify that the author's method of long term policy extraction from deterministic dynamic programming runs was a reasonable method, several 50 year trials were carried out to compare the policies thus obtained to those obtained by choosing, for each state, the decision which occurred most frequently. The reservoir sizes used were 30000 units for both sources, and the inflow and demand data were those of section 8.8, the 50 year inflow traces being generated by the method described in section 8.15. The objective was to minimise deficits only.

The policies obtained were very similar for both methods, the author's method yielding marginally better costs in simulations over 50 years, but because of the large storages involved compared to the demands, these results cannot lead to any general conclusions. Far more rigorous tests must be applied to determine whether one method is better than the other. However, the results did show that the author's method produced comparable policies and since it is not the purpose of this thesis to determine the best method of policy extraction, but only to show that consistent near optimum policies are extractable from deterministic dynamic programming calculations, with great computational reductions over the stochastic methods, any logical extraction method is acceptable. It is useful to reiterate here that because inaccuracies occur in trying to simplify the


Start
Read data:-
Demands
Inflow averages
Decision options
Reservoir levels
Costs
Present Worth Factor PWF
Set VALUES array to zero
Set FREQUENCES array to zero
Read initial:-
Month number
Inflow to Brenig
Inflow to Celyn
LABEL 1:
Read next:-
Month number
Inflow to Brenig
Inflow to Celyn
Calculate whether previous month's inflows were higher or lower than average for that month.
Set ANTA = 1 for lower, = 2 for higher in Celyn
Set ANTB = 1 for lower, = 2 for higher in Brenig.
Set LEVEL A = 1
LABEL 2:
Set LEVEL B = 1
LABEL 3:
Set DECISION = 1


LABEL4:
Set RAP = proportion of release from Celyn corresponding to the value of DECISION.
Set D = demand for this month
Set RA = RAP x D
Set RB = D - RA
Set Deficits = 0
Set Spills = 0
Calculate levels of reservoirs after inflows and releases, QFA and QFB.
Calculate spills SA, SB.
Calculate deficits DFA, DFB.
Find nearest four discretised states to state given by QFA and QFB.
Calculate interpolation factors.
Calculate whether specified releases, RA and RB, could have been achieved with the water available.
YES
Set RA1 = RA
RA2 = RB
NO
Calculate maximum releases possible: RA1 and RB1
Calculate immediate costs of this decision.
I COST = URA x RA1 + URB x RB1
+ USA x SA + USB x SB
+ UDF x (DFA + DFB)
Calculate, using interpolation factors, the value of being in the end state given by QFA and QFB at the end of the month.
This cost is found from the known VALUES array.
Set END COST to this value.
Calculate TOTAL COST of this decision at this stage.
TOTAL COST = I COST + PWF x END COST


Is DECISION = 1?
YES
NO
Is TOTAL COST < TEMP COST?
YES
Set
TEMP COST = TOTAL COST
NO
Add TOTALCOST for this decision and state into FREQUENCIES array.
FREQUENCIES [LEVEL A, ANTA, LEVELB, ANTB, DECISION] = FREQUENCIES [LEVEL A, ANTA, LEVELB, ANTB, DECISION] + TOTAL COST
Set DECISION = DECISION + 1
Is DECISION > than number of options possible?
NO
Go to LABEL 4
TEMP COST is now equal to the minimum cost obtained for this level combination. Store this cost in TEMP VALUES array as the optimum cost of going to the end of the process from this state and stage.
TEMP VALUES [STATE A, STATE B] = TEMP COST
Set LEVEL B = LEVEL B + 1
Is LEVEL B > Number of possible levels in Brenig.
NO
Go to LABEL 3
YES
Set LEVEL A = LEVEL A + 1
Is LEVEL A > number of possible levels in Celyn.
Go to LABEL 2
YES
Set VALUES array = TEMP VALUES array.
Has the required number of months been processed
NO
Go to LABEL 1
YES


For each combined value of LEVEL A, ANTA, LEVELB, ANTB, find the maximum cost over the values of DECISION of FREQUENCIES (LEVELA, ANTA, LEVELB, ANTB, DECISION).
Record and print the value of DECISION for the minimum cost as the optimum long term decision for the state given by LEVELA, ANTA, LEVELB, ANTB.
Finish

statistical properties of the data for use in stochastic programming, the results obtained by using synthetically generated traces together with the deterministic dynamic programming method may in fact be the more accurate.

8.20 The consistency of the deterministic method

As stated previously, before deterministic dynamic programming can be applied with confidence to finding long term policies for a system, some experiments must be carried out to compare the results obtained by this method to those found by the equivalent stochastic approach.

For this purpose, it was decided to apply deterministic dynamic programming to the simplified Celyn/Brenig system with reservoir sizes of 12000 and 15000 cusec-days respectively. Four levels and two antecedent indices were used in exactly the same way as the stochastic run described in Section 8.11. The demands used were again 1½ times those given in Section 8.8. Sequences of 100 years inflow data were generated by the method of Section 8.15 and the deterministic dynamic programming method applied, the long term policies being extracted by the author's method.

The reservoir sizes and demands were such that the policies obtained would be critical. The objective function was to minimise the deficit costs incurred, a discound factor of 1.5% per month being used.

The policies obtained for four of the inflow traces (S1 - S4 in Table 8.1) were applied to the same traces in the simulations. After this, three more traces (S5 - S7) were generated and the long term policies found from dynamic programming, but instead of applying each policy to the

inflow data used to calculate that policy, the policies were applied to different sequences to show whether the policies obtained in each case were general to the inflow population. In Section 8.11 a policy for this same system was determined by the stochastic method. This policy was now applied in a simulation to the same generated data as for S1 of Table 8.1 and the results are shown as S8. In all the simulations described previously the policy used was stated as a matrix of decisions, one for each possible discretised state in each month. When intermediate, or non-discrete states, were encountered in the simulation the relevant decisions were obtained by interpolation among the nearest discretised states. However, the simulations of S1 and S8, which used the same generated data, were repeated with the decision for any intermediate state being taken as that for the nearest discrete state, no interpolation being used. The costs in these instances are shown in brackets after the costs for the interpolation method.

The 100 year simulations took a total computer occupation time of from 4 to 5 minutes with C.P.U. times of from 45 to 55 seconds.

The computer programs for S2 to S4 were complete data generation, dynamic programming and simulation runs in the same jobs. The C.P.U. time varied from 1418 to 1555 seconds. Separate programs for 100 years data generation, and simulations took about 33 seconds and 55 seconds respectively for one instance. Unfortunately, the total computer occupation time for the complete program was not ascertainable because of the introduction of a time sharing system.

Table 8.1

RUN NUMBER	SIMULATION COSTS OVER 100 YEARS (Deficits x £100)
S1	1241200 (2734500)*
S2	2131560
S3	1489940
S4	3258790
S5	2562060
S6	2796170
S7	2810810
S8	1251820 (2594500)*

* Costs with no interpolation.

Reservoir Sizes:-

Celyn 12000cusec day

Brenig 15000 cusec day

Demands: $1\frac{1}{2}$ times those of Section 8.8.

Inflows: Generated from histograms of Section 8.8.

The actual policies derived by the stochastic and deterministic methods for cases S8 and S1 were compared and it was found that about 10% of the decisions were different, nearly two-thirds of these being different in only one decision interval.

Table 8.1 shows that the deficit costs incurred over 100 years of simulation in the same data by policies S1 and S8 are very nearly the same, whether interpolation for policies for intermediate states is used or not.

The costs shown in Table 8.1 are the actual costs incurred since the discount factor used in the dynamic program was not applied in the simulations. Therefore, the figures show the real sizes of the deficit multiplied by the unit cost of £100. The figure of £1251820 represents a total deficit of 12518.2 cusec-days over 100 years, which is less than 0.3% of the total demand over the period.

Expressing the deficit as a percentage of the demand shows that the spread of the figures in Table 8.1 is in fact small, the highest figure only being about 1% of the demand.

The costs for trials S4 to S6, for which the policies derived from one set of data were applied to a different data sequence for the simulations, fall well within the cost range for the other simulations and indicate that the deterministic policies are applicable over the inflow population as a whole, but because of the small number of trials carried out it would be unsafe to conclude that this is always so. However, the results show that further research in this area would be worthwhile.

The policies obtained for runs S2 to S4 were analysed and compared in detail to that for the stochastic policy of S8 and were also compared to one another. The results are shown below.

No. of Decision Intervals Different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	6	8	II	0	6	7	I	3	8	I	0	51
2	0	5	0	3	0	2	0	0	I	0	I	0	I2
3	0	2	0	0	0	I	I	0	0	0	0	0	4
4	I	0	0	0	2	0	0	0	0	0	0	0	3
TOTAL	I	I3	8	I4	2	9	8	I	4	8	2	0	70

S2 compared to S4

Table 8.2

No. of Decision Intervals Different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	6	I4	5	0	3	7	3	4	5	2	0	49
2	0	5	0	3	0	I	I	0	0	0	I	0	II
3	I	I	0	0	0	I	3	0	0	0	0	0	6
4	I	0	0	0	4	0	0	0	0	0	0	0	5
TOTAL	2	I2	I4	8	4	5	II	3	4	5	3	0	7I

S2 compared to S3

Table 8.3

No. of Decision Intervals Different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	6	I2	8	0	4	4	4	4	6	2	0	50
2	0	2	0	3	0	2	2	0	I	0	I	0	II
3	0	I	0	0	0	4	4	0	0	0	0	0	9
4	I	0	0	0	I0	I	0	0	0	0	0	0	I2
TOTAL	I	9	I2	II	I0	II	I0	4	5	6	3	0	82

S2 compared S8

Table 8.4

No. of Decision Intervals Different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	2	I0	9	0	6	6	2	I	4	3	0	43
2	0	3	0	0	0	2	2	I	0	0	0	0	9
3	I	3	0	0	0	I	I	0	0	0	0	0	6
4	I	0	0	0	I	0	0	0	0	0	0	0	2
TOTAL	2	8	I0	9	I	9	9	3	I	4	3	0	60

S3 compared to S4

Table 8.5

No. of Decision Intervals Different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	4	5	9	0	3	6	5	2	4	2	0	40
2	0	3	I	0	0	I	3	0	0	0	0	0	8
3	I	0	0	0	0	3	3	I	0	0	0	0	8
4	0	0	0	0	I0	I	0	0	0	0	0	0	II
TOTAL	I	7	6	9	I0	8	I2	6	2	4	2	0	67

S3 compared to S8

Table 8.6

No. of Decision Intervals Different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	4	4	I0	8	0	5	6	3	I	2	I	0	44
2	0	2	I	0	0	I	3	0	0	0	0	0	7
3	0	I	0	0	0	6	4	0	0	0	0	0	II
4	0	0	0	0	I2	I	0	0	0	0	0	0	I3
TOTAL	4	7	II	8	I2	I3	I3	3	I	2	I	0	75

S4 compared to S8

Table 8.7

	S2	S3	S4	S8
S2	-	71(9.2%)	70(9.1%)	82(10.7%)
S3	71(9.2%)	-	60(7.8%)	67(8.7%)
S4	70(9.1%)	60(7.8%)	-	75(9.8%)
S8	82(10.7%)	67(8.7%)	75(9.8%)	-

Total number of decision differences between policies.

Table 8.8

8.21 Length of data sequence

Following this set of experiments, it was decided to investigate the effects on the policies and simulation cost of varying the length of the data series used in the deterministic dynamic programs. In these experiments, the same 100 years generated trace, S4, was used throughout as a data base. For the dynamic program, the required length of data series was taken from the end of the 100 year trace. The policies produced in each case were applied over the full 100 year sequence in the simulation runs.

The same system as in the previous section was used, all inflow, size, and cost data being identical. The unit deficit costs was again £100.

The same composite computer program, which generated data, performed the dynamic program and ran the simulation, as in Section 8.19 for runs S2 to S4 was used.

The total C.P.U. time was 796 seconds for a 50 year program, 483 seconds for 25 years, 359 seconds for 15 years and 297 seconds for 10 years.

The simulation costs are given in Table 8.9.

Length of d.p. data series in years	Average Yearly costs over 100 year simulation
100 (S4)	32588
50	35826
25	40448
15	52277
10	117904

Table 8.9

The following tables show the policies derived from the 50 to 10 year programs compared to S4, the 100 year policy, and S8, the stochastic policy.

No. of Decision Intervals Different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	5	I2	I0	0	6	4	6	3	7	0	0	53
2	0	2	0	0	0	2	2	0	2	0	3	0	II
3	0	I	0	0	0	5	5	0	0	0	0	0	II
4	5	I	0	0	II	I	0	0	0	0	0	0	I8
TOTAL	5	9	I2	I0	II	I4	II	6	5	7	3	0	93

50 year d.p. compared to S8

Table 8.I0

No. of Decision Intervals Different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	5	6	I0	0	4	2	2	I	5	0	0	35
2	0	I	0	0	0	I	2	0	2	0	2	0	8
3	0	I	0	0	0	I	0	I	0	0	0	0	3
4	4	0	0	0	I	0	0	0	0	0	0	0	5
TOTAL	4	7	6	I0	I	6	4	3	3	5	2	0	51

50 year d.p. compared to S4

Table 8.II

No. of Decision Intervals Different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	7	20	26	2	8	5	4	6	I0	0	0	88
2	0	5	3	0	0	I	3	I	2	I	2	0	I8
3	0	4	0	0	0	7	2	0	0	0	6	0	I9
4	4	2	0	0	I2	I	0	0	0	0	0	0	I9
TOTAL	4	I8	23	26	I4	I7	I0	5	8	II	8	0	I44

25 year d.p. compared to S8

Table 8.I2

No. of Decision Interval different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	7	I5	2I	I	9	6	3	5	I0	0	0	67
2	0	4	I	0	0	0	3	I	2	2	I	0	I4
3	0	3	I	0	0	I	I	0	0	0	7	0	I3
4	3	I	0	0	2	0	0	0	0	0	0	0	6
TOTAL	3	I5	I7	2I	3	I0	I0	4	7	I2	8	0	I00

25 year d.p. compared to S4

Table 8.I3

No. of decision Interval different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	I	I9	I9	4	8	5	3	8	II	0	0	78
2	0	4	2	2	0	3	0	I	0	3	2	0	I7
3	0	6	I	I3	I	3	6	0	0	0	6	0	36
4	7	4	0	3	9	8	0	0	0	0	0	0	3I
TOTAL	7	I5	22	37	I4	22	II	4	8	I4	8	0	I62

I5 year d.p. compared to S8

Table 8.I4

No. of decision Intervals different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	2	9	20	5	9	8	4	7	I2	0	0	76
2	0	3	2	2	0	I	3	0	0	2	I	0	I4
3	0	6	2	II	0	4	3	I	0	I	7	0	35
4	6	3	0	5	2	9	0	0	0	0	0	0	25
TOTAL	6	I4	I3	38	7	23	I4	5	7	I5	8	0	I50

I5 year d.p. compared to S4

Table 8.I5

No. of Decision Intervals Different	No. of Decisions Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	4	I4	I9	3	6	9	8	I2	9	0	0	84
2	0	8	2	2	7	3	0	6	0	4	2	0	34
3	0	6	0	I3	2	3	4	I	3	I	6	0	39
4	I8	6	0	3	II	8	9	0	0	I3	0	I2	80
TOTAL	I8	24	I6	37	23	20	22	I5	I5	27	8	I2	237

I0 year d.p. compared to S8

Table 8.I6

No. of Decision Intervals Different	No. of Decision Differences												
	J	F	M	A	M	J	J	A	S	O	N	D	TOTAL
I	0	5	II	20	5	8	7	9	I0	I0	0	0	85
2	0	6	I	3	7	2	3	6	0	3	I	0	32
3	0	6	I	II	0	4	4	I	3	2	7	0	39
4	I7	5	0	4	2	I0	I0	0	0	I3	0	I2	73
TOTAL	I7	22	I3	38	I4	24	24	I6	I3	28	8	I2	229

I0 year d.p. compared to S4

Table 8.I7

Length of data series used in dp. in years	Total number of decisions different from:	
	S4	S8
10	229	237
15	150	162
25	100	144
50	51	93
100	0	75

Table 8.18

It can be seen from these tables that the policy for the 50 year sequence is very similar to the stochastic policy and to the 100 year policy, the simulation cost being close to that for run S4. Even the 25 year policy produces a low simulation cost although the policy is different in twice as many decision elements as the 100 year policy from the stochastic policy. This, combined with the results of comparisons of policies from different 100 year sequences, leads one to believe that the policy differences which do occur between data sets and those caused by length of series, at least with series greater than 50 years in length, are minor differences and generally occur for states where there is a marginal choice between the possible decisions.

8.22 Effect of reservoir sizes

A series of 50 year deterministic dynamic programs combined with 50 year simulations was carried out, using the same data sequence throughout for both the dynamic programs and the simulations, to investigate the convergence to optimal reservoir sizes using the near optimum policy determined for each case. At the same time, further tests were made to compare simulation costs obtained when policy interpolation for intermediate states was used to those obtained when the decision for an intermediate state was taken as the same decision as that for the nearest discretised state.

In both cases, experiments were also carried out to show the results of using a policy which was optimum for one reservoir size combination on all other size combinations. The results are given in Table 8.19. The demands used were 1½ times those of section 8.8 and the objective was to minimise deficit costs, the unit deficit cost being £100. A discount factor of 1.5% per month was employed in the dynamic programs but not in the simulations.

The dynamic programs of Series 2 of Table 8.19 using 8 levels in each reservoir with two antecedent indices for previous inflow, HIGH and LOW, giving 256 states in all, used about 900 seconds of C.P.U. time and one hour total time on the ICL 1905 computer. The simulations used about 30 seconds of C.P.U. time and 6 minutes total time.

Reservoir Sizes Cusec - days		Total Storage Cusec - days	D.P. Values after 50 years Series 1. 6 levels in each reservoir starting from both:-		D.P. Values after 50 years Series 2 8 levels in each reservoir starting from both:-		Simulation Series 2 Average Yearly Costs with Policy Interpolation	As Col.8 but without interpolation for policies	Simulation costs using policy for Series 2 15000/ 18000 throughout With interpolation for policies	As Col.10 but without interpolation for policies
Celyn	Brenig		Full Antecedent Indices both low	Empty Antecedent Indices both low	Full Antecedent Indices both low	Empty Antecedent Indices both low				
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
30000	24000	54000	648	973971						
30000	21000	51000	919	972172						
30000	18000	48000	1331	970577						
30000	15000	45000	2051	968864						
24000	24000	48000	516	968374						
24000	21000	45000	763	966940						
24000	18000	42000	1149	965396						
24000	15000	39000	2018	965682						
21000	24000	45000	522	961421						
21000	21000	42000	779	960424						
21000	18000	39000	1180	959439			-	-	-	-
21000	15000	36000	2115	959834			-	-	-	-
18000	24000	42000	490	958090	0.1	954892	0	25500	0	0
18000	21000	39000	688	957224	0.3	954557	0	25500	0	0
18000	18000	36000	1007	956661	0.7	954309	0	35000	0	0
18000	15000	33000	1732	956942	2.0	954192	0	0	0	0
15000	24000	39000	-	-	17.6	957730	0	28000	0	43250
15000	21000	36000	-	-	27.8	955869	0	28000	0	43250
15000	18000	33000	3858	964193	40.5	955181	0	43250	0	43250
15000	15000	30000	6102	964371	71.0	954689	0	49250	0	43250
12000	24000	36000	-	-	4265	1001449	0	178500	363120	794750
12000	21000	33000	-	-	5443	992100	32572	104750	336297	794750
12000	18000	30000	45412	1038116	6797	984898	33252	81500	353428	857750
12000	15000	27000	56842	1036404	8725	976914	33273	192000	403834	912250

Table 8.19

The first series of programs was run with 6 levels in each reservoir, which, combined with the antecedent indices, gave a total of 144 system states. Only the deterministic dynamic programs were carried out for this trial, no attempt being made to extract a long term policy for simulation purposes. For twenty combinations of reservoir capacities, the fifty year optimum costs of running the system, starting from states of both reservoirs full and both reservoirs empty, with initial antecedent indices representing low previous inflows to both reservoirs, were extracted. These costs are, of course, the absolute minimum costs which could be achieved with the given data and discretisation level. Any long term policy applied to the same data and problem structure would necessarily produce higher operating costs than these. The results of the first series of programs are shown in columns (4) and (5) in Table 8.19.

It can be seen that for a fixed Celyn size and decreasing Brenig size the costs of starting from both reservoirs full tends to increase with decreasing Brenig size, as expected. However, although this occurs for each chosed Celyn size, there is no relationship between the lowest costs incurred in each group for the various Celyn sizes. This may be explained by the varying accuracy of the problems as the size combinations change, because the same number of states are used in each reservoir whatever the capacity. Therefore, the problems with smaller reservoir capacities yield the more accurate results. Neither is the accuracy of the problem the same for, say, a problem with Celyn 24000 and

Brenig 18000, and a problem with Celyn 18000 and Brenig 24000, since the accuracy is also related to the inflow rates to each reservoir, and consequently the 'usefulness' of the reservoir, which is determined by the policy solution itself. The accuracy is also affected by discretisation of the decision possibilities.

The figures in column (5), for starting from both reservoirs empty, are likely to be more affected by the accuracy of the problem since the costs (of deficits) are only incurred when the reservoirs are depleted. The figures show that the general tendency is for the costs to fall for smaller reservoir sizes, which is clearly not logical. However, in spite of this, because the costs fall over a wide range as Celyn size decreases, whatever the capacity of Brenig, it may be deduced that Celyn is the more important reservoir of the two, which is true because it has much higher rates of inflow than Brenig, and that the level of discretisation of Celyn affects the costs even more than the capacity. When Celyn becomes very small, the accuracy of the problem becomes greater, and the influence of Brenig on the system becomes more pronounced, as Celyn reaches its critical capacity, which appears to be about 15000 cusec-days. The last two figures in column (5) show that the costs are less for a Brenig size of 15000 cusec-days than for a size of 18000 cusec-days which indicates that the accuracy of Brenig discretisation still affects the system more than the capacity. From this one can assume that even at a capacity of 15000 cusec-days Brenig is still too large.

The figures shown in column (7), using 8 levels in

each reservoir instead of 6, still show the same tendencies, the critical level of Celyn, before costs increase at a fast rate, being about 15000 cusec-days. One can see again that Brenig capacity does not appear to affect the costs unduly at a capacity of 15000 cusec-days.

Because the change in accuracy of the problems affects the final optimum costs, and since the absolute optimum policy is used throughout the dynamic programming solution, rather than the best long term fixed policy obtainable, which would have to be applied in practise, it is not strictly fair to compare the optimum costs incurred by each reservoir size combination to determine the optimal configuration. Therefore, a practical way to determine costs which are comparable is to extract the long term policy as described in previous sections and to apply this to a sequence of data in a simulation to obtain the operating costs.

Column (8) of Table 8.19 shows the costs obtained from 50 year simulations starting from a state of both reservoirs full with antecedent indices representing low previous flows. In this set of simulations, linear interpolation was used to determine a decision for states that were not discrete from the decisions for the nearest surrounding discretised state decisions. The figures show that no costs are incurred until Celyn capacity is 12000 cusec-days and Brenig capacity is 21000 cusec-days. The costs are nearly the same whatever the capacity of Brenig after this. These results indicate that the critical level in Celyn probably lies between 15000 and 12000 cusec-days, while Brenig can

be reduced below 15000 cusec-days. Further experiments in this range would determine the optimal sizes.

The simulations described above were repeated without interpolation for decisions for intermediate states, and it was found that the costs incurred started at a much greater Celyn size of 18000 cusec-days and the costs for lower Celyn sizes were much higher then before (Col. (9)). Although some of the costs for a fixed Celyn size increase with decreasing Brenig size this is not always so, especially when Celyn has a capacity of 12000 cusec-days, when no cost pattern can be seen. It is apparent that policy interpolation is desirable and, because there is not always a pattern to the costs when no interpolation is used, that in some instances the no-interpolation method might yield reasonable results but in others it might not. Because there is a tendency, when simulation combined with trial and error policy determination is used as a design method, to apply a generalised policy to several reservoir sizes to find comparable operating costs, it was decided to apply one long term policy, determined for a size combination of 15000 cusec-days in Celyn and 24000 cusec-days in Brenig, to a range of combinations in simulation exercises. The results are shown in Column (10) of Table 8.19 and show that no costs are incurred until Celyn has capacity 12000 cusec-days and Brenig has capacity 24000 cusec-days. The costs are then ten times greater, with policy interpolation, than those incurred in the same circumstances with the optimum long term policies for each system. The figures again confirm that the optimal capacity of Celyn lies between 15000 and

12000 cusec-days. They also show that, in this case, any reasonable policy could be used to locate the critical level, an obvious increase in costs occurring at this point.

These simulations were repeated using no policy interpolation and, as before, the figures show that costs are incurred at higher size combinations, but another obvious increase occurs when Celyn falls to 12000 cusec-days.

The results of Table 8.19 thus indicate that the cheapest and easiest method of determining optimal reservoir sizes may be to apply a reasonable operating policy, say the space, or reservoir balancing rule, to various size combinations until a sudden large increase in costs is indicated. This will show the area for detailed investigation in one reservoir. The experiment can then be repeated to obtain a critical range for the other reservoir, or reservoirs.

The determination of the best sizes can then be carried out by dynamic programming in a restricted range.

Experiments to investigate the level of discretisation necessary are described in Appendix 5.

8.23 The Celyn/Brenig System

Because a policy had been found by the simulation method for the Celyn/Brenig system, as described in Chapter 2, it was decided to apply the deterministic dynamic programming method to the same 42 years inflow and demand data. The objective function used was to minimise the deficit costs, but a higher cost weighting was given to deficits which involved compensation water, the unit cost ratio of deficits in compensation to deficits in make-up water being 10, with costs of £1000/unit and £100/unit. A discount factor was not used. The long term policy was extracted from the dynamic programming results by the author's method.

The reservoirs were divided into 11 levels each, and HIGH and LOW antecedent indices were taken into account when the policy was extracted. The reservoir sizes used were 21000 cusec-days and 31859 cusec-days in Brenig and Celyn respectively. The monthly retention levels, compensation waters, and low water limit in Celyn were used in the same way as in Chapter 2.

Simulations over the 42 years data were then carried out using the dynamic programming policy, with interpolation for intermediate states, and the policy found by the methods of Chapter 2.

The policy of Chapter 2 produced a total cost of £1,291,167 over 42 years, or £30,742 per year on average. Three consecutive monthly deficits occurred in 1934 from July to September, the total compensation deficit being 138 cusec-days and the ordinary make-up deficit being

8814 cusec days. In 1937, make-up deficits were 2118 cusec-days over the two months, October and November, and there were no compensation deficits. In 1938, there was a compensation deficit, only, of 60 cusec-days in June. Thus, deficits occurred in three separate years, the total compensation deficit being 198 cusec-days, and the make-up deficit being 10932 cusec-days.

The dynamic programming policy with the same data produced simulation costs of £979,800 over 42 years, or an average yearly cost of £23,329. Deficits only occurred in the three months, July to September, of 1934. The compensation deficits were 94 cusec-days, and the make-up deficits were 8858 cusec-days.

The results show that the dynamic programming method produces a better policy for this system if deficits are minimised. The true value of dynamic programming is not reflected in the difference in costs, however, because the work described in previous sections indicates that the size of Celyn reservoir is greater than required and that the policy is not very critical.

The computer program for the deterministic dynamic program used above consumed a total of 1 hour 52 minutes, and a C.P.U. time of 2160 seconds on the ICL 1905 machine, and the simulation with the resulting policy took 6 minutes 13 seconds total time and 39 seconds C.P.U. time, 22 seconds being used in computation.

The simulation using the policy of Chapter 2 took 2 minutes 46 seconds of total time and 32 seconds C.P.U. time, 19 seconds being used in computation.

Chapter 9

State Increment Dynamic Programming

9.1. Introduction

The theory & application of state increment dynamic programming is described by Larsen (II). He states that this method leads to a computational procedure that has significant advantages in terms of computational requirements over the conventional procedure. The main objective of the method is to reduce the high speed memory requirements by arranging the calculations so that the maximum number of computations may be carried out with the minimum immediate information.

9.2. The Theory

The basic iterative functional equation is the same as that given by equation 6.6. The discretisation of state, control and time variables is done in exactly the same way as for the conventional case. The minimum cost and optional control are computed at all quantised values of $\bar{x}$ and t. The minimisation in equation 6.6. is performed by applying all admissable discretised controls and choosing the minimum value of the right hand side of the equation. The generality of the problem formulation and the form of the solution are in no way changed.

As described by Larsen, the reduction in computational requirements is obtained by combining the two basic concepts of the new approach. The first of these is related to the choice of δt in equation 6.6. This time increment, which is the interval over which a control is applied, is not now set equal to Δt, the time increment between successive computations of optimal control. Thus, two time intervals have been introduced into the calculation instead of one, Δt, as for the conventional method. The dynamic programming

calculations are still carried out at fixed time intervals, or stages. It is only the computations within these stages which are modified.

However, the δt in equation 6.6., over which a control is applied, is not fixed but is chosen as part of the calculation. As described in Chapter 6, in the conventional method δt is fixed at Δt, and a particular state may make a transition to any other state during this time, but the state increment method finds the value of δt during which the original state makes a transition to the next nearest discretised state. This value may be less than, equal to, or greater than Δt, but if it is greater than Δt, the value used for the subsequent optimisation is set at Δt. This procedure ensures that the state transition is over one or less state increments. Formally,

$$\delta t = \underset{i=1,n}{\text{Min}} \left\{ \left| \frac{\Delta x_i}{f_i[\bar{x}, \bar{u}, t]} \right| \right\} \quad \ldots . \quad \text{Equation 9.1.}$$

The value of δt is calculated for each state variable and the minimum is chosen. Thus, the next state, $\bar{x} + \bar{f}(\bar{x}, \bar{u}, t)\delta t$, is always close to the present state. Larsen describes it as lying on the surface of an n- dimensional hypercube centred at $\bar{x}$, with length $2\Delta x_i$ along the xi-axis. The minimum cost at the next state is found using interpolation in (n-1) of the state variables and time, using values at the discretised states that lie in this hypercube at times $(t + \Delta t)$ and $(t + 2\Delta t)$. In other words, a state transition beginning at time t is made to a time for which there is no value of the function, I, stored in the machine since the calculations for I are only carried out at fixed intervals, Δt. Therefore, the value of I at the time under consideration is found by

extrapolation of the values calculated at times $(t + \Delta t)$ and $(t + 2\Delta t)$. Because the transitions are to states within a confined hypercube, only the minimum costs corresponding to these points need to be stored in the high speed memory, as opposed to the minimum costs for every admissable discretised state as required by the conventional method.

However, the saving in high speed storage by using this method is offset by the need to transfer these small blocks of data from low speed to high speed memory at frequent intervals, if the calculations are carried out in the same order as for the conventional procedure, because each state for which calculations are carried out might need a different set of values. The computing time involved in these transfers is so large that the method is not attractive. But if large amounts of data are transferred infrequently, the waiting time is reduced and if the computations can be arranged to take advantage of this then the method might become useful.

State increment dynamic programming uses this concept by carrying out the computations not according to the ordering of the time increments, as in the conventional procedure, but in blocks which cover a small number of discretised states buty many time increments. If the present state under consideration is in the interior of a block, then the discretised states required for extrapolation of the minimum cost at the resulting state after transition are always in the block. It is then possible to compute the minimum cost and the optimal decisions throughout the block on the basis of an initial set of minimum costs at the two largest discretised values of time at the end of the block. By modifying the computations at the boundaries of the block it is possible to allow the trajectories to pass from block to block. Consequently, an

efficient computational procedure is obtained in which the transfer of a relatively small amount of initial date enables computations to take place throughout the block.

Other interpretations of extrapolation methods than that described above may be used for finding the optimum cost at the resulting state after transition. Some of these involve using only the optimum costs found at stage $(t + \Delta t)$ instead of for both $(t + \Delta t)$ and $(t + 2\Delta t)$. Advantage can also be taken of the fact that interpolation can be carried out at time t when sufficient costs have been computed for some states by other methods. The procedures are described in detail by Larsen.

Interblock transitions are easily made into previously computed blocks by storing the optimal costs previously calculated at the boundary of the blocks, as well as the costs at the end of the block under consideration. It is also not necessary to re-compute the values along that boundary for the new block. Larsen states that transitions into blocks which have not been previously calculated are more complicated and the calculations are not so accurate. However, it is possible to a limited extent to order the blocks so that some calculations can be carried out where it is known that the transitions will not go out of the block. For instance, in the reservoirs problem, the transitions at the extreme states cannot go outside the block, since negative reservoir storages cannot exist. The methods where transitions must be to blocks not yet computed are not described here but are detailed by Larsen. Only the general principle of state increment dynamic programming needs to be given here, since the author's experiement with this method only concerned the extra time required over the

conventional procedure caused by introducing the minimisation of δt and the subsequent time extrapolations. Otherwise, the computations were exactly the same as for the conventional procedure. All cost data was held in the computer fast store.

9.3. Stochastic Example

The system and data used was exactly the same as that described in Section 8.8. The reservoir sizes were again both 30,000 cusec-days and the decision options were identical.

The number of instructions carried out was 288038 in the requested execution time. In this time, 10¼ annual iterations were produced but the system did not optimise. However, the policy produced at iteration 10 was only different in one decision from the policy found at iteration 10 in Section 8.8. The policies found at each iteration should be the same as those in Section 8.8. but the differences will be caused by the difference between the extrapolation over two time periods when δt is less than Δt, and the interpolation at one time as with the conventional procedure. In fact, in the example taken, most of the δt were equal to Δt because of the large state increments, so that, although the computations still included the same number of steps, and therefore absorbed the same computer time, no extrapolation was necessary. The values of the states in January at iteration 10 were slightly different from the values in the conventional procedure, ranging from 262459 to 253 instead of 261405 to 253. From the number of iterations carried out in both programs it can be seen that the state increment method takes about 1.4 times longer than the conventional method. However, this is the maximum time difference which can be expected since no transition data was stored at the first year's iteration for use in subsequent iterations. If this was done, only the

first iteration's time would be significantly different for the two methods, but the values of δt for each state and each decision would have to be stored. However, when the transition data is stored to save computing time, the amount of data which has to be transferred from low speed storage even for one state makes the use of state increment dynamic programming impractical. Also, for the efficient use of this method, the state increments should be small so that δt is less than Δt.

Therefore, for the stochastic approach, where transition data, which is the same for each iteration, can be stored and all iterations act on similar data, the state increment method is not very useful, but in a deterministic case, where the data continually varies at each iteration and a large number of states exist in the system, this method might reduce the computation time by reducing the number of transfers of data from low speed to high speed storage when only a small amount of the data can be stored in the high speed store at one time.

For a comparison of the times consumed on the Atlas computer to the time taken on the ICL 1900 machine for this method a short run of the same program was made on the ICL computer. It was found that $3^5/12$ iterations were carried out in a CPU time of 1780 seconds and a total time of 53 minutes. This gives a time of 15.5 minutes per yearly iteration compared to the Atlas figure of 2.8 minutes per iteration.

Chapter 10

Conclusions

10.1 Specialist Languages

Details of a language which meets many of the requirements for an ideal specialist computing language have been given. It was found that such a powerful language is necessary for reservoir design problems, unless one were to have to revert to a more general language as ALGOL or FORTRAN. Various discrete and continuous simulation languages were not useful for such problems.

The most important conclusion to be drawn from this study, which was only realised well into the work, was the extraordinary power of a POL in the hands of an expert in its use. A POL is the 'Seven League Boots' of the computer user. While it can be true that a POL simplifies computer usage for the ordinary user he can probably gain almost as much from the use of a package or even a compatible library of sub-routines as from the use of a POL. To some extent developments in computer hardware and system software have changed the original climate in which POL's were spawned.

The desk top computer, the mini computer and the terminal have all had the effect of bringing low cost computing into the design office. The small computer favours the use of a series of small programs, one for each task with minimum linkage. Though linkage can be achieved through inexpensive tape storage devices now available.

System software has improved to a vast extent making the compatible subroutine idea look attractive. For

instance the George 3 and 4 system software available on the ICL 1900 series computers has made forms of virtual store and linkage of programs more available to the user. By use of the file-store in such systems, which operate similarly to the data files in the HYDRO described herein, one can link the output from one package to the input of another with very little programming. The user has many of the problems of peripheral assignment removed from him or made easier. Once again it is true to be said that these new system control languages lend proportionally more power to the expert than the novice.

We see the situation that while attempts are being made to simplify computer usage by the provision of Problem Oriented Languages and powerful System Control Languages the power available to the expert user is growing at a greater rate than it is becoming available to the non-expert. Whether this situation is desirable or not is debatable; it does however seem inevitable.

10.2 Dynamic Programming

10.2.1. The deterministic dynamic programming procedures described in Chapter 8 were structured to fit into the HYDRO language. However, for every different system configuration a new dynamic program must be written so as to include all of its special peculiarities. Because of this, the author allowed for the use of pure Algol code in the user's input to HYDRO. The computer user could then write his dynamic programming and simulation system descriptions using the flexibility of Algol while still being able to employ the simple data structure of HYDRO.

10.2.2. A computer program was developed for the stochastic value iteration method describing two finite reservoirs supplying one demand point. It was found that although the method converged to the optimum policy, the point of convergence could not be defined with precision. In particular, it is unsafe to assume that convergence has taken place when two year's consecutive policies are the same.

10.2.3. A program has been written for the application of Howard's policy iteration method to the two finite stochastic reservoirs problem, which involves the solution of a large number of simultaneous equations. However, because of the monthly grouping of the equations it was possible to apply matrix partitioning and an efficient method of solution has been developed by the author. With the method of using policy iteration developed in this thesis, it is not necessary, once one month's solutions have been obtained, to substitute them in the other eleven month's equations to find the complete solution. Because one year's value iteration always follows the solution of the equations it is only necessary to know the solutions for January. The value iteration procedure itself, if carried out on a monthly basis, will update these costs for the other eleven months as the calculations proceed. This method is more efficient and will lead to the optimum more quickly than performing the value iteration on a yearly basis with the complete equation solutions as starting values.

The policy iteration method, as described above, was applied to two finite stochastic reservoirs. Four levels were used in each reservoir, and two antecedent indices of inflow were included in the state space. The number of system states was therefore 64 in each month. Five inflows, with their associated probabilities were allowed to each reservoir, and five possible decision options were considered at each state and stage.

It was found, for the Celyn/Brenig data given in Section 8.8, that the optimum policy was achieved after 4 years' policy iterations. This policy corresponded in all but one decision in 768 to the policy found by value iteration after 13 years.

The number of times a set of equations has to be solved before the optimum is obtained may be reduced by a combination of value and policy iteration. Value iteration may be applied until the first suboptimum is reached, when two consecutive years' policies are the same, and then policy iteration can take over fully, or, when the suboptimum is reached, one policy iteration may be used to check whether it is in fact the optimum. If it is not, then value iteration may be applied again until the next local optimum is reached, when another policy iteration check may be used, and so on.

10.2.4. The computer time taken by the policy iteration method to reach the optimum was even longer than that required by the value iteration procedure to carry out 13 years' iterations.

In view of the fact that the value iteration policy,

after 8 years, produced a policy only different in two decisions out of 768 from the optimum policy, it is considered that value iteration is the more practical approach. If this method is continued until five years' policies are the same, it is unlikely that the policy will change to any significant degree after this.

10.2.5. Unless it is important to find short term policies for stochastic data, then it is suggested that even value iteration consumes excessive computer times. In order to use the method in a real problem involving two or more reservoirs with stochastic inflows, the number of states which it would be necessary to employ would be so small that the reliability of the results would be questionable.

Furthermore, the requirement of stating the inflows, and perhaps demands, in histogram form might lead to severe oversimplifications of the statistical properties of the data. It must also be borne in mind that the program would have to be run for various trial configurations, so that the design cost would soon become prohibitive.

10.2.6. Because of the gross discretisation necessary for the stochastic methods of dynamic programming and the computing costs involved, the author was led to explore the idea of obtaining near optimum policies from the results of the deterministic dynamic programming method applied to historical or synthetically generated data.

As a result of the reduced computational requirements involved in this method, the intervals of discretisation of the state variables could be made smaller, thus increasing

the accuracy of the solution. It is also thought that the problem accuracy is greater because the data is not forced into simple histograms. The historical data itself can be used complete with its true serial and cross correlations. Even if the historical sequence is not long enough, or critical enough, and has to be extended by synthetic methods, it is thought that generation techniques are sufficiently advanced to yield traces that incorporate more of the true statistical properties of the real data than histograms, combined with states describing correlations, can ever do.

10.2.7. Whatever, the method of design used, careful thought must be given to the objective function.

Two completely different policies were obtained when deficits were minimised in one case, and reservoir total drawdown was minimised in another. On the surface it might appear that the two objectives would achieve the same ends, but, depending upon the size of, and inflows to, the reservoirs, this might not be true.

The real objective for a water supply reservoir is to minimise the deficits. If there is more than one objective the weighting to be given to each must be decided.

10.2.8. If deterministic dynamic programming is used on a historical or synthetically generated data trace, then some method of extracting a long term policy from the results must be used. Using regression analysis to find one single function which relates the decision to be made to the known values of the system variables at any time is thought to be too gross a method. Much of the detailed information

would be smoothed over in this way.

A better method, still using regression analysis, would be to split the system variables into smaller units and produce a different function for each. It is also necessary to decide upon which system variables to include in the regression analysis. The reservoir levels must obviously be included, but the effect of including previous inflows and forecasts of future inflows must be investigated by some form of factor analysis. Efficient statistical routines for regression and factor analysis are available at most computer installations.

For early studies in reservoir system design it is not considered necessary to use such techniques. It is only when the final design has been decided that such refinements should be used to improve the policy. It appears from the results of Chapter 8 that it would be sufficient only to include the reservoir levels, and perhaps crude measures of previous inflows, in the system variables, until the major design parameters have been found.

A good method of policy extraction in these early stages has been described in Section 8.18.

10.2.9. Deterministic dynamic programming was applied to a system of two reservoirs with fixed sizes, but with different inflow traces generated from the same histograms. The long term policies derived were applied to simulations using the corresponding inflow traces. For states which were not discrete, the simulation interpolated for decisions between the nearest discretised states.

The consistency of the policies and simulation costs

was found to be good.

10.2.10. Experiments carried out using different data sequences in the simulations from those used in the dynamic programs showed that the simulation costs fell within the range of costs obtained when the same sequences were used for both.

This again indicates that the policies found from deterministic dynamic programming applied to synthetic data traces are consistent.

10.2.11. Simulations were carried out using the policy found from stochastic policy iteration for the equivalent system. The costs and the policy itself were comparable to those achieved with the deterministic method.

10.2.12. Simulations of all policies without interpolation for intermediate states produced much higher costs than with interpolation. This indicates that the policies tend to be continuous and that interpolation may be used to advantage.

10.2.13. Experiments were carried out to investigate the length of data series required in the deterministic programs.

The results showed that a 50 year sequence produced similar costs to thosefor 100 years which appeared to be equivalent to costs incurred using the policy found by policy iteration. Even the costs from a sequence as short as 15 years were not excessive. However, shorter series produced much higher costs.

It seems that sequences of 50 years' data would be adequate, at least until the final reservoir sizes have been fixed, longer traces being used for 'polishing' the operating

policy if thought necessary.

10.2.14. For a fixed, generated 50 year data trace, the two reservoir sizes were varied and deterministic dynamic programming applied to find the appropriate long term policies. Simulations with these policies were then carried out to obtain the 50 year operating costs.

The policy from an intermediate combination of reservoir sizes was applied to all the size combinations and simulation costs again calculated. It was found, using the policy applicable to each configuration, that costs were not incurred until the reservoirs were small. It can be inferred that at least one reservoir size had become critical at that point.

The costs obtained when the one fixed policy was used throughout started at a slightly greater size in one reservoir and subsequent costs were much greater than those found using the correct policy in each case.

Therefore, it is suggested that any reasonable operating rule, say the space rule or balancing rule, may be applied to various trial configurations until a large increase in costs occurs when the reservoirs are made slightly smaller. This method will yield a limited range of possible reservoir sizes. The optimum sizes may then be found by applying deterministic dynamic programming.

10.2.15. State Increment dynamic programming was investigated but was not found to be of great advantage for water resources problems. The computer programs are also more difficult to write, from the data management point of view, than for the conventional case. The method would be

more advantageously applied to problems where the inputs may be described as smooth, continuous functions, or where the system is such that a large number of states may be used. In the latter case considerable savings in computer time might be obtained.

10.3 Future Work

10.3.1. The results described in this thesis indicate that deterministic dynamic programming is a viable method of obtaining operational policies. Many aspects of its application have been investigated and present a wide scope of topics in which research in depth would be worthwhile.

10.3.2. More work is required on the effect of varying the number of states considered in the system. The optimum number of states and their distribution in space may depend upon system parameters such as inflow, reservoir size, and demands. The distribution of the states will also be related to the objective function in some way.

10.3.3. Further experiments are necessary to investigate the consistency of the policies when synthetic data traces are used.

10.3.4. The policies derived from stochastic dynamic programming should be compared more thoroughly to those derived from synthetic data traces.

10.3.5. More experiments should be carried out on the length of data series required.

10.3.6. Far more work is needed into the effects of different objective functions. Data needs to be collected from practising engineers concerning their requirements from water resource systems.

10.3.7. Thought should be given to the idea of maintaining prescribed flows in rivers, and the penalties to he applied for not meeting these flows.

10.3.8. Research is needed into the use of the 'critical period' as a basis for deriving a policy.

10.3.9. The effect of including variables to describe previous inflows or forecasted inflows should be studied.

10.3.10. If possible, it would be informative to apply deterministic dynamic programming to existing systems, operating under rules derived by other means. In this way the benefits of this method could be demonstrated in a powerful manner.

10.3.11. Changes in policies found for various reservoir sizes should be investigated.

References

1. Bugliarello,G., McNally,W.D., Onstott,J.T., & Gormley,J.T. 'Design Philosophy, Specifications, and Implications of "Hydro", a Pilot Computer Language for Hydrology and Hydraulic Engineering' Carnegie Institute of Technology, Pittsburgh, Pa. 1966.

2. McNally,W.D., 'Hydro User's Manual', Carnegie Institute of Technology, Pittsburgh, Pa., 1966.

3. McNally,W.D., 'Hydro Reference Manual', Carnegie Institute of Technology, Pittsburgh, Pa., 1966.

4. Roos,D., 'ICES System Design', Massachusetts Institute of Technology Press, Cambridge, Massachusetts,1966.

5. 'Computer Languages and Program Libraries', Task Committee, Journal, A.S.C.E. Hydraulics Div., HY7, July, 1972.

6. Cooper,B.E., 'Ascop User's Manual', Science Research Council Atlas Computer Laboratory, Chilton, Didcott, Berks.

7. 'Genesys Reference Manual', March, 1970. The Genesys Centre University of Technology, Loughborough.

8. 'Hydrologic/1' - A proposal for a subsystem of Genesys Draft

9. Bellman,R., 'Dynamic Programming', Princeton University Press, Princeton, New Jersey, 1957.

10. Bellman,R., 'Applied Dynamic Programming', Princetown, University Press, Princetown, New Jersey, 1962.

11. Larsen,R.E., 'State Increment Dynamic Programming', Stanford Research Institute, Menlo Park, California, 1968.

12. Howard,R.A. 'Dynamic Programming and Markov Processes', Cambridge (Mass.) Technology Press, 1960.

13. Larsen,R.E., & Keckler,W.G., 'Application of Dynamic Programming to the Control of Water Resources Systems', Technical Memorandum 1967, Stanford Research Institute Menlo Park, California.

14. Keckler,W.G., & Larsen,R.E., 'Dynamic Programming Applications to Water Resources System Operation and Planning', Journal of Mathematical Analysis and Applications 24, 80 - 109 (1968).

15. Korsak,A.J. & Larsen,R.E., 'Convergence Proofs for a dynamic programming successive approximations technique', Fourth I.F.A.C. Congress, Warsaw, Poland, June, 1969.

16. Mawer,P.A., Sherriff,J.D.F., Thorn,D., Wyatt,T. 'Multiple Resource Studies', Technical Memorandum T.M.67 Economics Group, September, 1971, Water Research Association.

17. Schweig,Z., & Cole,J.A., 'Optimal Control of Linked Reservoirs', Water Resources Research, Vol 4, Nr 3 June, 1968.

18. Schweig,Z., 'Optimisation of Control Rules for Water Storage Systems by dynamic programming', Water Research Association, (1968), Reservoir Yield, Part 3, Technical Paper T.P.61.

19. Cole,J.A., 'Probability of Short Term Inflows - Decisions in Time of Drought', Water Research Association, Reservoir Yield Symposium, Part 1, 1965, Paper 9.

20. Burley,M.J. & Cole,J.A., 'Alternative digital computer applications to evaluate linked water resources'. International Association of Scientific Hydrology, Tucson Symposium, 1969.

21. Buras,N.,'Conjunctive operation of dams and aquifers' Proc., A.S.C.E. Hydraulics Div., 89, HY6, ;; 111-131.

22. Meier,L., 'Combined Optimum Control and Estimation', 1965 Allerton Conference, Univ. of Illinois (Oct. 1965) pp 109-120.

23. Buras,N., 'The Conjunctive use of Surface and Ground Waters', W.R.A., Reservoir Yield Symposium, 1965, Paper 10, Part 1.

24. Aron,G. & Scott, V.H., 'Dynamic Programming for conjunctive Water Use', Journal, Hydraulics Div., Proc. A.S.C.E., HY5, May, 1971.

25. Roefs,T.G. & Bodin,L.D., 'Multireservoir Operation Studies', Water Resources Research, Vol 6, Nr 2 April, 1970.

26. Lloyd,E.H. 'Probability of Emptiness - I' W.R.A. Reservoir Yield Symposium, 1965 Part I Paper 5.

27. Harris,R.A., Dearlove,R.E. & Morgan,M. 'Serially Correlated Inflows and subsequent attainment of steady state probabilities',W.R.A. Technical Memorandum T.P.45, Reservoir Yield - II.

28. Young,G.K., 'Finding Reservoir Operating Rules', Proc., A.S.C.E. Hydraulics Div. 93,HY6, Nov. 1967.

29. Hall,W.A. & Howell, D.T., 'The Optimisation of single purpose reservoir design with application to synthetic hydrology examples'. Journal of Hydrology, 1963 pp 355 - 363.

30. 'Some factors influencing required reservoir storage' Proc., A.S.C.E., Hydraulics Div. HY7, July, 1971.

31. Wilson,T.T., & Kirdar,E.,
'Use of Runoff Forecasting in Reservoir operations',
Journal, A.S.C.E., Irrigation and Drainage Div., 96
IR3, Sept.1970.

32. Gablinger,M & Loucks,D.P., 'Markov models for Flow
Regulation', Journal A.S.C.E., Hydraulics Div., HY1
January, 1970.

33. Harboe,R.C., Mobasheri,F. & Yeh, W.W-G
'Optimal Policy for Reservoir Operation',
Journal, A.S.C.E., Hydraulics Div., HY.11, Nov.1970.

34. Loucks,D.P.,'Computer Models for Reservoir Regulation',
Journal, A.S.C.E., Sanitary Div., SA4, August, 1968.

35. James, L.D., 'Economic Derivation of Reservoir
Operating Rules', Journal A.S.C.E., Hydraulics Div.
HY9, Sept. 1968.

36. Fiering,M.B., 'Forecasts with Varying Reliability',
Journal A.S.C.E. Sanitary Div, June 1969.

37. Perez., A.I., Schaake,J., & Pyatt,E.E.,
'Simulation Model for Flow Augmentation Costs',
Journal, A.S.C.E. Hydraulics Div. HY1, Jan.1970.

38. Butcher,W.S. 'Mathematical Models for optimising
the allocation of stored water', Internation Assoc.
Scientific Hydrology. 'The use of Analog and Digital
Computers in Hydrology', Tucson Symposium, 1969.

39. Burges,S.J. & Linsley,R.K., 'Some factors influencing
required reservoirs storage', A.S.C.E. Hydraulics
Div. July 1971, HY7.

40. Hall,W.A., Askew,A.J., Yeh,W.W.-G.
'Use of the Critical period in Reservoir Analysis',
Water Resources Research, Vol 5, Nr 6, December, 1969.

41. O'Kane, Personal Communication

APPENDIX 1

Appendix I

Proof of convergence of stochastic matrices

Assume that we have a probability transition matrix, $\bar{R}$, with all non zero elements where R_{iJ} is the probability of jumping from state J to state i in one time increment, and the probabilities are independent of time.

Then $R_{iJ} > 0 \quad \sum_{i=I}^{n} R_{iJ} = I$ for all J where $\bar{R}$ is an nxn matrix

$i,J = I, \quad n.$

Let k be a discrete time variable.

The probability of moving from state J at time k to state i at time $(k + S)$ is obviously the $(i,_J)$th element of the matrix $(\bar{R})^S$.

Let $\bar{p}(0)$ be a column vector whose elements are the probabilities of the system being in the states J $(J = I,n)$ at time k=0.

Then $\bar{p}(k)$, the probabilities of being in states J $(J = I,n)$ at time k, is given by

$$\bar{p}(k) = (\bar{R})^k \bar{p}(0)$$

To prove that $\bar{p}(k)$ reaches a limiting distribution for large k it is necessary to investigate the latent roots, or eigenvalues, of the matrix $\bar{R}$. The latent roots of the matrix $\bar{R}$ are given by the equations

$|\bar{R} - \lambda \bar{I}| = 0$ where $||$ represents a determinant.

Let $\lambda_1, \lambda_2, \ldots \lambda_n$ be the n distinct roots where $\prod_{J=1}^{n} \lambda_J = |\bar{R}|$

The equations

$$\bar{R}\,\bar{x} = \lambda \bar{x} \quad \text{and} \quad \bar{y}^{-1}\,\bar{R} = \lambda \bar{y}^{-1} \qquad . \quad . \quad . \text{(A.I.I.)}$$

where $\bar{x}$ and $\bar{y}$ are column vectors, have solutions $\bar{x}$ and $\bar{y}$ other than zero, if and only if λ is a root (eigenvalue) of $\bar{R}$. For each eigenvalue, $\lambda_J (J=I,n)$, therefore, we can solve for the corresponding left and right eigenvectors $\bar{x}_J$ and $\bar{y}_J$.

Let $\bar{H} = [\bar{x}_I, \bar{x}_2, \bar{x}_3 \ldots \bar{x}_n]$

and $\bar{K} = [\bar{y}_I, \bar{y}_2, \bar{y}_3, \ldots y_n]$

From equation (A.I.I.) it follows that

$$\bar{R}\bar{H} = \bar{H}\bar{L} \quad \text{and} \quad \bar{K}^{-1}\bar{R} = \bar{L}\bar{K}^{-1} \qquad . \quad . \quad . \quad \text{(A.I.2.)}$$

where $\bar{L}$ is the diagonal matrix

$$\bar{L} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \lambda_n \end{bmatrix}$$

From (A.I.2.) we thus have the alternative expression for $\bar{R}$ given by

$$\bar{R} = \bar{H}\bar{L}\bar{H}^{-I} = (K^{-1})^{-I}\,\bar{L}\,\bar{K}' \qquad . \quad . \quad . \quad \text{(A.I.3.)}$$

From (A.I.I.)

$$\bar{R}\bar{x}_i = \lambda_i \bar{x}_i$$

Premultiplying by y_J' we have

$$y_J'\,\bar{R}\bar{x}_i = \bar{y}_J'\,\lambda_i\bar{x}_i$$

Also from (A.I.I.) we have $\bar{y}_J'\bar{R} = \lambda_J\bar{y}_J'$. Postmultiplying by $\bar{x}_i$ we have

$$\bar{y}_J'\,\bar{R}\bar{x}_i = \lambda_J\bar{y}_J'\bar{x}_i = \bar{y}_J'\lambda_J\bar{x}_i$$

$$\therefore \quad \bar{y}_J'\,\bar{R}\bar{x}_i = \bar{y}_J'\lambda_i\bar{x}_i = \bar{y}_J'\,\lambda_J\,\bar{x}_i$$

If the eigenvalues λ_i and λ_J are different then obviously

$$\bar{y}_J'\,\bar{x}_i = 0 \qquad i \neq J \qquad . \quad . \quad . \quad \text{(A.I.4.)}$$

We can obviously choose the scales of $\bar{x}_J$ and $\bar{y}_J'$ so that

$$\bar{y}_J'\,\bar{x}_J = I \qquad . \quad . \quad . \quad \text{(A.I.5.)}$$

Using equations (A.I.4.) and (A.I.5.) we can write

$$\bar{K}'\bar{H} = \bar{I}$$

and equation (A.I.3.) can be put in the form

$$\bar{R} = \bar{H}\bar{L}\bar{K}' = \sum_{J=I}^{n} \lambda_J\,\bar{A}_J \qquad . \quad . \quad . \quad \text{(A.I.6.)}$$

where $\bar{A}_J = \bar{x}_J \bar{y}_J'$

and

$$\left.\begin{aligned} \bar{A}_i \bar{A}_J &= \bar{x}_i(\bar{y}_i' \bar{x}_J)\bar{y}_J' = 0 \text{ (from (4)) for } i \neq J \\ &= \bar{x}_i(\bar{y}_i' \bar{x}_i)\bar{y}_i' = \bar{x}_i \bar{y}_i' = \bar{A}_i \text{ (from A.I.5.) for } i \neq J \end{aligned}\right\} \quad \text{. Equ.A.I.7.}$$

$$\sum_{J=I}^{n} \bar{A}_J = \bar{I}$$

This last expression follows from

$$\sum_{J=I}^{n} \bar{A}_J = \sum_{J=I}^{n} \bar{x}_J \bar{y}_J' = \bar{H}\bar{K}' = \bar{H}(\bar{K}' \bar{H})\bar{H}^{-I} = \bar{I}$$

Because of the relations in A.I.7. we may write, using equations A.I.6.

$$(\bar{R})^k = \sum_{J=I}^{n} (\lambda_J)^k \bar{A}_J \qquad \text{since } (\bar{A}_J)^k = \bar{A}_J$$

$$(\bar{R})^k = \sum_{J=I}^{n} (\lambda_J)^k \bar{x}_J \bar{y}_J' \qquad \text{. . . A.I.8.}$$

Now, the probability distribution of being in the states, J, at time k is given by

$$\bar{p}(k) = (\bar{R})^k \bar{p}(0)$$

$$= \left\{ \sum_{J=I}^{n} (\lambda_J)^k \bar{x}_J \bar{y}_J' \right\} \bar{p}(0) = \sum_{J=I}^{n} (\bar{y}_J' \bar{p}(0)) \lambda_J^k \bar{x}_J$$

$$\bar{p}(k) = \sum_{J=I}^{n} \alpha_J \lambda_J^k \bar{x}_J \quad \text{where } \alpha_J = \bar{y}_J' \bar{p}(0) \qquad \text{. . . A.I.9.}$$

Since the columns of $\bar{R}$ must always sum to unity then it is possible to write

$$\left[I, I, \ldots . I\right] \bar{R} = \left[I, I, \ldots . I\right]$$

$$\text{or} \quad \left[I, I, \ldots . I\right] \bar{R} = I \left[I, I, \ldots . I\right] \qquad \text{. . . A.I.IO.}$$

From equation A.I.I. we have

$$\bar{y}_i' \bar{R} = \lambda_i \bar{y}_i' \qquad \text{. . . A.I.I.}$$

Comparing A.I.I. and A.I.IO. we see that one non trivial solution to

equation A.I.I. is

$$\bar{y}_I' = [I, I, \ldots . I]$$

$$\lambda_I = I$$

In this case, $\alpha_I = \bar{y}_I' \bar{p}(0) = I$ since the elements of $\bar{p}(0)$ must sum to unity.

Therefore, we may write from equation A.I.9.

$$\bar{p}(k) = \alpha_I \lambda_I^k \bar{x}_I + \sum_{J=2}^{n} \alpha_J \lambda_J^k \bar{x}_J$$

$$\bar{p}(k) = \bar{x}_I + \sum_{J=2}^{n} \alpha_J \lambda_J^k \bar{x}_J \qquad . \quad . \quad . \quad \text{A.I.II.}$$

The root $\lambda_I = I$ cannot be exceeded by any other root.

Since $|\bar{R} - \lambda \bar{I}| = 0$ there is at least one non trivial solution of the equation $\bar{y}'\bar{R} = \lambda \bar{y}'$

Let y_m' be the maximum element of $\|\bar{y}'\|$ where $\| \ \|$ denotes the absolute values of the components of $\bar{y}'$.

Then, because the columns of $\bar{R}$ sum to unity,

$$y_m' \geq (\bar{y}'\bar{R})_m = \|\lambda\| \, y_m'$$

from which $\|\lambda\| \leq 1$.

From equation A.I.II. it follows that the eigenvector, $\bar{x}_I$, will give a limiting distribution for $\bar{p}(k)$ independent of $\bar{p}(0)$ as $k \to \infty$ if $\lambda_2 \to \lambda_n$ are all less than unity.

Frechet (1938) has shown that $\lambda_2 \to \lambda_n$ are all less than unity if the $\bar{R}$ matrix or any power of the $\bar{R}$ matrix has all non zero elements.

However, even if this were not the case, if the physical process has any stability then it is possible to insert very small positive elements in place of the zeros which occur, without altering the behaviour of the system.

It also follows from equation A.I.8. that

$$\bar{R}^k = \sum_{J=I}^{n} \lambda_J^k \, \bar{x}_J \, \bar{y}_J' = \lambda_I^k \, \bar{x}_I \, \bar{y}_I' + \sum_{J=2}^{n} \lambda_J^k \, \bar{x}_J \, \bar{y}_J' \quad . \quad . \quad . \text{ A.I.I2.}$$

or $\bar{R}^k = \bar{x}_I [I,I,I,\ldots.I]$ As $k \to \infty$ since $\lambda_I = I$ and $\bar{y}_I' = [I,I,\ldots I]$.

. . . A.I.I3.

so that $\bar{R}^k$ converges to a stochastic matrix, with all columns identicalin and equal to the limiting state distribution $\bar{p}(k)$.

Equation A.I.I2. may be written as

$$\bar{R}^k = \bar{x}_I [I,I,I,\ldots I] + \sum_{J=2}^{n} \lambda_J^k \, \bar{x}_J \, \bar{y}_J'$$

or $\bar{R}^k = \bar{S} + \bar{T}(k)$ . . . A.I.I4.

where $\bar{S}$ is a stochastic matrix whose J^{th} column is the vector of limiting state probabilities if the system is started in the J^{th} state, and $\bar{T}(k)$ is a set of differential matrices with geometrically decreasing coefficients.

For the determination of the analytical equations of dynamic programming for a long term process it is now necessary to write equation A.I.I4. in terms of z-transforms where $F(z) = \sum_{k=0}^{\infty} f(k) z^k$.

A table of z-transforms for commonly occurring functions $f(k)$ is given in Appendix 2.

Now the basic equation for state distributions at any time for a fixed transition matrix $\bar{R}$ is

$$\bar{p}(k+I) = \bar{R} \, \bar{p}(k) \quad . \quad . \quad . \text{ A.I.I5.}$$

Let $\bar{\pi}(z)$ be the z-transform of $\bar{p}(k)$.

Then by taking the z-transformationsof equation A.I.I5. we have

$$z^{-I} \left[\bar{\pi}(z) - \bar{p}(0)\right] = \bar{R} \, \bar{\pi}(z)$$

Expanding, we have

$$\bar{\pi}(z) - \bar{p}(0) = z\bar{R} \, \bar{\pi}(z)$$

$$\therefore \bar{\pi}(z) = (\bar{I} - z\bar{R})^{-I} \, \bar{p}(0) \quad . \quad . \quad . \text{ A.I.I6.}$$

where $\bar{I}$ is the unit matrix.

Let $\bar{H}(k)$ be the inverse transformation of $(\bar{I} - z\bar{R})^{-I}$ on an element by element basis. Then, taking the inverse transformation of equation A.I.I6. we have

$$\bar{p}(k) = \bar{H}(k)\bar{p}(0) \quad . \quad . \quad . \quad \text{A.I.I7.}$$

But we know that

$$\bar{p}(k) = \bar{R}^{k}\bar{p}(0) \quad . \quad . \quad . \quad \text{A.I.I8.}$$

Therefore, comparing equations A.I.I7. and A.I.I8.

$$\bar{H}(k) = \bar{R}^{k}$$

From equation A.I.I4. we may say that

$$\bar{H}(k) = \bar{R}^{k} = \bar{S} + \bar{T}(k) \quad . \quad . \quad . \quad \text{A.I.I9.}$$

Taking the transform of equation A.I.I9. gives

$$(I-zR)^{-I} = \frac{1}{1-z}\bar{S} + \bar{J}(z) \quad . \quad . \quad . \quad \text{A.I.20.}$$

where $\bar{J}$ is the transform of $\bar{T}$.

It can be seen that equation A.I.20. is an equation describing the system under consideration which is now independent of k.

This equation will be of use in showing how analytical equations may be formed for the long term dynamic programming problem.

APPENDIX 2

Appendix 2

The z-transform

$$F(z) = \sum_{n=0}^{\infty} f(n) z^n$$

where $F(z)$ is independent of n.

The re-transformation of a transform will produce the original function.

Consider some typical functions :

a) $$f(n) = \begin{cases} 1, & n = 0,1,2,3, \\ 0, & n < 0 \end{cases}$$

The z-transform is

$$F(z) = \sum_{n=0}^{\infty} f(n) z^n = 1 + z + z^2 + z^3 + \ldots\ldots\ldots = \frac{1}{1-z}$$

b) $$f(n) = \alpha^n , \quad n \geq 0$$

$$F(z) = \sum_{n=0}^{\infty} f(n) z^n = \sum_{n=0}^{\infty} (\alpha z)^n = \frac{1}{1-\alpha z}$$

c) If $F(z) = \sum_{n=0}^{\infty} (\alpha z)^n$ then

$$\frac{d}{dz} F(z) = \sum_{n=0}^{\infty} n \alpha^n z^{n-1}$$

$$\therefore \quad \sum_{n=0}^{\infty} n \alpha^n z^n = z \frac{d}{dz} F(z) = z \frac{d}{dz}\left(\frac{1}{1-\alpha z}\right) = \frac{\alpha z}{(1-\alpha z)^2}$$

Thus if $f(n) = n \alpha^n$ then

$$F(z) = \frac{\alpha z}{(1-\alpha z)^2}$$

The z-transform of $f(n+1)$ is given by

$$\sum_{n=0}^{\infty} f(n+1) z^n = \sum_{m=1}^{\infty} f(m) z^{m-1} \qquad \text{where } m = n+1$$

Now $$\sum_{m=1}^{\infty} f(m) z^{m-1} = f(1) + f(2) z + f(3) z^2 + \ldots$$

and $$\sum_{n=0}^{\infty} f(n) z^n = f(0) + f(1) z + f(2) z^2 + \ldots$$

Thus $\sum_{m=1}^{\infty} f(m)z^{m-1} = z^{-1}\left\{\sum_{n=0}^{\infty} f(n)z^{n} - f(0)\right\}$

If $F(z) = \sum_{n=0}^{\infty} f(n)z^{n}$ then

the z-transform of $f(n+1)$

$$\sum_{n=0}^{\infty} f(n+1)z^{n} = z^{-1}\left(F(z) - f(0)\right)$$

A table of z-transforms for common functions occurring in dynamic programming is presented in Table I.

Table I

z-transforms

function for $n \geq 0$	z-transform
$f(n)$	$F(z)$
$f_1(n)+f_2(n)$	$F_1(z)+F_2(z)$
$kf(n)$ (k = constant)	$kF(z)$
$f(n-1)$	$zF(z)$
$f(n+1)$	$z^{-1}(F(z)-f(0))$
α^n	$\frac{1}{1-\alpha z}$
1 (unit step)	$\frac{1}{1-z}$
$n\alpha^n$	$\frac{\alpha z}{(1-\alpha z)^2}$
n (unit ramp)	$\frac{z}{(1-z)^2}$
$\alpha^n f(n)$	$F(\alpha z)$

APPENDIX 3

Appendix 3

The inverse transform of $\frac{z}{(I-z)}\ \bar{Q}\ \bar{J}(z)$

$$\bar{T}(n) = \sum_{J=2}^{a} \lambda_J^{\,n}\ \bar{x}_J\ \bar{y}_J' \quad \text{by definition} \qquad \ldots \text{A.3.I.}$$

Taking the z-transform

$$\bar{J}(z) = \sum_{J=2}^{a} \frac{I}{(I-\lambda_J z)}\ \bar{x}_J\ \bar{y}_J' \qquad \ldots \text{A.3.2.}$$

Multiplying throughout by $\frac{z}{I-z}\ \bar{Q}$ we have

$$\frac{z}{(I-z)}\ \bar{Q}\bar{J}(z) = \bar{Q} \sum_{J=2}^{a} \frac{z}{(I-z)(I-\lambda_J z)}\ \bar{x}_J\ \bar{y}_J' \qquad \ldots \text{A.3.3.}$$

Partial fraction expansion gives

$$\frac{z}{(I-z)}\ \bar{Q}\bar{J}(z) = \bar{Q} \sum_{J=2}^{a} \left\{ \frac{\frac{I}{(I-\lambda_J)}}{(I-z)} - \frac{\frac{I}{(I-\lambda_J)}}{(I-\lambda_J z)} \right\}\ \bar{x}_J\ \bar{y}_J' \qquad \ldots \text{A.3.4.}$$

The inverse transform gives

$$\text{I.T.} = \bar{Q} \sum_{J=2}^{a} \left\{ \frac{I}{(I-\lambda_J)} - \frac{\lambda^n}{(I-\lambda_J)} \right\}\ \bar{x}_J\ \bar{y}_J' \qquad \ldots \text{A.3.5.}$$

As $n \to \infty$ the second term in the brackets $\to 0$

Therefore we have

$$\text{I.T.} = \bar{Q} \sum_{J=2}^{a} \frac{I}{(I-\lambda_J)}\ \bar{x}_J\ \bar{y}_J' = \text{constant as } n \to \infty$$

i.e. a step of magnitude $\bar{Q} \sum_{J=2}^{a} \frac{I}{(I-\lambda_J)}\ \bar{x}_J\ \bar{y}_J'$

From equation A.3.2. it can be seen that

$$\bar{J}(I) = \sum_{J=2}^{a} \frac{I}{(I-\lambda_J)}\ \bar{x}_J\ \bar{y}_J'$$

Therefore, the term $\frac{z}{(I-z)}\ \bar{Q}\ \bar{J}(z)$ represents a step of magnitude

$$\bar{Q}\ \bar{J}(I) = \bar{Q} \sum_{J=2}^{a} \frac{I}{(I-\lambda_J)}\ \bar{x}_J\ \bar{y}_J'$$

APPENDIX 4

Appendix 4

Line library

```
'LIBRARY'(ED,SUBGROUPTEMP)
'INPUT'3=ED2(ICLA=DEFAULT(0))
'OUTPUT'0=LP0
'BEGIN'
'REAL'GRIDAREA,RAINVOLUME,SHEDAREA,RAINFALLAVERAGE,ANNUAL,MEAN,DSR;
'INTEGER'A,B,C,D,K,L,SUBI,SUB2,SUB3,COLM,Z,GRIDROWS,GRIDCOLUMNS,
TOTAL4,TOTAL6,DSI;
'BOOLEAN'FINISHED;
'INTEGER''ARRAY'CHAR[I:80],ALPH[I:80],DAI[I:2];'REAL''ARRAY'DAR[I:2];
'PROCEDURE'NEXTCOLUMN(COLM,CHAR,Z);
'INTEGER'COLM,Z;
'INTEGER''ARRAY'CHAR;
'ALGOL';
'REAL''PROCEDURE'NUMERICALDATA(COLM,CHAR,Z,ALPH,NEXTCOLUMN);
'INTEGER'COLM,Z;
'INTEGER''ARRAY'CHAR,ALPH;
'PROCEDURE'NEXTCOLUMN;
'ALGOL';
'PROCEDURE'ALPHANUMERICALDATA(COLM,CHAR,Z,ALPH,NEXTCOLUMN);
'INTEGER'COLM,Z;
'INTEGER''ARRAY'CHAR,ALPH;
'PROCEDURE'NEXTCOLUMN;
'ALGOL';
'PROCEDURE'DEFINESUBSCRIPTS(COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,
SUBI,SUB2,SUB3);
'INTEGER'COLM,Z,SUBI,SUB2,SUB3;
'INTEGER''ARRAY'CHAR,ALPH;
'PROCEDURE'NEXTCOLUMN;
'REAL''PROCEDURE'NUMERICALDATA;
'ALGOL';
'REAL''PROCEDURE'DATAINPUT(COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,
SUBI,SUB2,SUB3,ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D);
'INTEGER'COLM,Z,SUBI,SUB2,SUB3,A,B,C,D;
'INTEGER''ARRAY'CHAR,ALPH;
'BOOLEAN'FINISHED;
'PROCEDURE'NEXTCOLUMN;
'REAL''PROCEDURE'NUMERICALDATA;
'PROCEDURE'ALPHANUMERICALDATA;
'PROCEDURE'DEFINESUBSCRIPTS;
'ALGOL';
'PROCEDURE'SINGLEREAD(VARI,VAR2,TYPE,COLM,CHAR,Z,ALPH,NEXTCOLUMN,
NUMERICALDATA,SUBI,SUB2,SUB3,ALPHANUMERICALDATA,DEFINESUBSCRIPTS,
FINISHED,A,B,C,D,DATAINPUT);
'VALUE'TYPE;
'INTEGER'TYPE;
'REAL'VARI;
'INTEGER'VAR2;
'INTEGER'COLM,Z,SUBI,SUB2,SUB3,A,B,C,D;
'INTEGER''ARRAY'CHAR,ALPH;
'BOOLEAN'FINISHED;
'PROCEDURE'NEXTCOLUMN;
'REAL''PROCEDURE'NUMERICALDATA;
'PROCEDURE'ALPHANUMERICALDATA;
'PROCEDURE'DEFINESUBSCRIPTS;
```

```
'REAL''PROCEDURE'DATAINPUT;
'ALGOL';
'PROCEDURE'ARRAYREAD(VARI,VAR2,TYPE,AA,BB,CC,DD,BNDI,BND2,BND3,
COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,
SUBI,SUB2,SUB3,ALPHANUMERICALDATA,DEFINESUBSCRIPTS,
FINISHED,A,B,C,D,DATAINPUT,PROGEND);
'VALUE'TYPE,BNDI,BND2,BND3,AA,BB,CC,DD;
'INTEGER'TYPE,BNDI,BND2,BND3,AA,BB,CC,DD;
'REAL''ARRAY'VARI;
'INTEGER''ARRAY'VAR2;
'INTEGER'COLM,Z,SUBI,SUB2,SUB3,A,B,C,D;
'INTEGER''ARRAY'CHAR,ALPH;
'BOOLEAN'FINISHED;
'LABEL'PROGEND;
'PROCEDURE'NEXTCOLUMN;
'REAL''PROCEDURE'NUMERICALDATA;
'PROCEDURE'ALPHANUMERICALDATA;
'PROCEDURE'DEFINESUBSCRIPTS;
'REAL''PROCEDURE'DATAINPUT;
'ALGOL';
Z:=0:COLM:=0;SELECT INPUT(3);SELECT OUTPUT(0);
LLL9:NEXTCOLUMN(COLM,CHAR,Z);
'IF'CHAR[COLM]=I58'THEN''GOTO'LLL9;
A:=CHAR[COLM];
B:=CHAR[COLM+I];
C:=CHAR[COLM+2];
D:=CHAR[COLM+3];
COLM:=COLM+3;
DEFINESUBSCRIPTS(COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,
SUBI,SUB2,SUB3);
ALPH[I]:=I6I;
BASI
SINGLEREAD(DSR,GRIDCOLUMNS,2,
COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,
ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D,DATAINPUT);
SINGLEREAD(DSR,GRIDROWS,2,
COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,
ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D,DATAINPUT);
SINGLEREAD(GRIDAREA,DSI,I,
COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,
ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D,DATAINPUT);
SINGLEREAD(DSR,TOTAL4,2,
COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,
ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D,DATAINPUT);
SINGLEREAD(DSR,TOTAL6,2,
COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,
ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D,DATAINPUT);
SINGLEREAD(ANNUAL,DSI,I,
COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,
ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D,DATAINPUT);
BAS2
'BEGIN''INTEGER''ARRAY'GRID[I:GRIDROWS,I:GRIDCOLUMNS];

ARRAYREAD(DAR,GRID,2,I8,29,20,I5,GRIDROWS,GRIDCOLUMNS,0,
```

```
COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,
ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D,DATAINPUT,PROGEND);
'BEGIN''REAL''ARRAY'RAIN[I:GRIDROWS,I:GRIDCOLUMNS];

ARRAYREAD(RAIN,DAI,I,29,I2,20,25,GRIDROWS,GRIDCOLUMNS,0,

COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,
ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D,DATAINPUT,PROGEND);
'BEGIN''REAL''ARRAY'SHEDRAIN[I:TOTAL6];

ARRAYREAD(SHEDRAIN,DAI,I,30,I9,I6,I5,TOTAL6,0,0,

COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,
ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D,DATAINPUT,PROGEND);
'BEGIN''REAL''ARRAY'STATIONRAIN[I:TOTAL4];

ARRAYREAD(STATIONRAIN,DAI,I,30,3I,I2,3I,TOTAL4,0,0,

COLM,CHAR,Z,ALPH,NEXTCOLUMN,NUMERICALDATA,SUBI,SUB2,SUB3,
ALPHANUMERICALDATA,DEFINESUBSCRIPTS,FINISHED,A,B,C,D,DATAINPUT,PROGEND);
BAS3
'FOR'K:=I'STEP'I'UNTIL'GRIDROWS'DO''BEGIN'
'FOR'L:=I'STEP'I'UNTIL'GRIDCOLUMNS'DO''BEGIN'
RAIN[K,L]:=-I;'END''END';
BAS4
'BEGIN'
'PROCEDURE'AVERAGE(VECTOR,LENGTH,MEAN);
'INTEGER'LENGTH;
'REAL'MEAN;
'REAL''ARRAY'VECTOR;
'ALGOL';
'BEGIN'
'PROCEDURE'ARITHMETICRAINFALLAVERAGE(AVERAGE,SHEDRAIN,TOTAL6,
RAINFALLAVERAGE);
'INTEGER'TOTAL6;
'REAL'RAINFALLAVERAGE;
'REAL''ARRAY'SHEDRAIN;
'PROCEDURE'AVERAGE;
'ALGOL';
ARITHMETICRAINFALLAVERAGE(AVERAGE,SHEDRAIN,TOTAL6,RAINFALLAVERAGE);
'BEGIN'
'PROCEDURE'NORMALANNUALPRECIPITATION(AVERAGE,STATIONRAIN,TOTAL4,ANNUAL);
'INTEGER'TOTAL4;
'REAL'ANNUAL;
'REAL''ARRAY'STATIONRAIN;
'PROCEDURE'AVERAGE;
'ALGOL';
NORMALANNUALPRECIPITATION(AVERAGE,STATIONRAIN,TOTAL4,ANNUAL);
'BEGIN'
'PROCEDURE'THIESSENRAINFALLAVERAGE(RAINVOLUME,SHEDAREA,GRID,GRIDAREA,
GRIDROWS,GRIDCOLUMNS,RAIN,RAINFALLAVERAGE);
'INTEGER''ARRAY'GRID;
'INTEGER'GRIDROWS,GRIDCOLUMNS;
'REAL''ARRAY'RAIN;
'REAL'RAINVOLUME,SHEDAREA,GRIDAREA,RAINFALLAVERAGE;
```

```
'ALGOL';
THIESSENRAINFALLAVERAGE(RAINVOLUME,SHEDAREA,GRID,GRIDAREA,
GRIDROWS,GRIDCOLUMNS,RAIN,RAINFALLAVERAGE);
BAS5
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
'END';
PROGEND:'END';
GARBAGE
****  0
E.D.S. SOURCE FILE IS OF LENGTH I4 BUCKETS.
```

APPENDIX 5

APPENDIX 5

Discretisation of state variables

Deterministic experiments were carried out with various numbers of states in the same system. The data used were the 42 years inflows and the demands for a maintained flow of 450 cusecs described in Chapter 2 for the Celyn/Brenig system. The reservoir capacities were 30000 cusec-days for both reservoirs. The objective was to minimise deficits, the unit deficit cost being £100/unit. Table A5.1 shows the dynamic programming costs incurred in the several cases.

No. of levels in each Reservoir	Costs after 1 year Starting from both:		Costs after 42 years Starting from both:	
	Full	Empty	Full	Empty
	£	£	£	£
4	0.4	1069216	20659	120599
6	0	976359	3029	12063
8	-	-	62	756
16	0	974008	0	0
21	0	973086	0	0

Table A5.1

Fig.A5.1 shows the costs for a typical case of one year's dynamic programming. For the Celyn/Brenig system, it was thought that from six to eight levels in each reservoir was sufficient. However, the distribution of the states is significant. Inspection of Fig. A5.1 shows that the sensitive range of levels for minimisation of deficits is from 0 to 10000 cusec-days, as would be expected. For different objective functions different ranges would be

important. It is a cheap and efficient process to investigate the sensitive range for a particular system by running one dynamic program with a relatively large number of states and to fix the states for the design investigation depending upon the results.

It appears, from the results of Chapter 8, that the significant range of reservoir sizes may be ascertained using reasonable state increments, and then, if required, the state increments may be reduced to find the optimum solution.

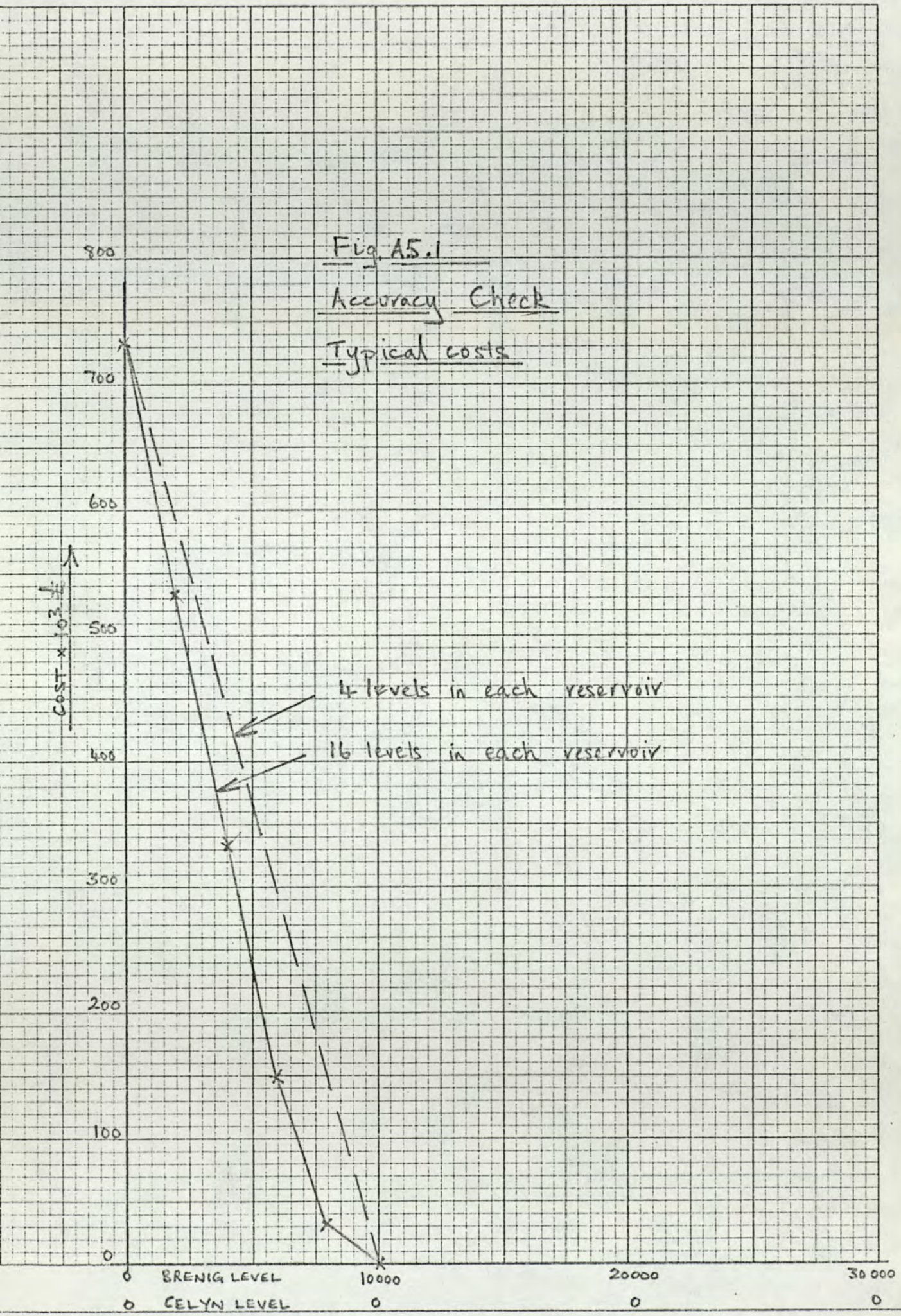


Fig. A5.1

Accuracy Check

Typical costs