OPTIMAL COMPUTER CONTROL OF A GROUP OF INTERACTING HYDRO ELECTRIC POWER STATIONS

A thesis submitted to the Faculty of Engineering of the University of Glasgow for the degree of

Doctor of Philosophy

By

James Clink, B.Sc.

December 1986

To My Parents

Acknowledgements

I would like to thank Professor J. Lamb for provision of the laboratory facilities in the Department of Electronics and Electrical Engineering.

I am indebted to Dr H. Davie for his close supervision of my work. His advice and many useful suggestions have proved invaluable, both during the project and as I prepared this thesis. I am also very grateful to Dr D. Muir and Dr M. Macauley for their many constructive discussions and practical assistance throughout the project.

I would like to thank the North of Scotland Hydro Electric Board and the S.E.R.C. for the provision of the C.A.S.E. award. In particular, I would like to thank Mr P. M. Johnson and Mr J. Painter of the NSHEB for all their help. Special thanks must go to the staff and engineers of Errochty Group, without whose help this work could not have been attempted.

I would also like to thank the members of staff of this department who have advised and assisted me in the past three years, and especially, my fellow research students for the many interesting discussions and nights out, particularly, Mr D. Fraser, Mr D. Loomes, Mr W.C. Michie, Dr J. Duffy, Dr L. Bradley and Mr I. McIntyre.

I would like to thank all my family who have given me so much support and encouragement during my time at university, and, finally, thanks must go to Lisi for all her patience and support, without which this thesis would not have been possible.

Summary

The North of Scotland Hydro Electric Board (NSHEB) are currently upgrading the control equipment for hydro generating plant. An important part of this modernisation is the introduction of optimisation methods to the control system. Highly constrained plant poses an unique optimisation problem. This project reports the application of optimisation techniques to one such highly constrained group, namely, the Tummel Valley Scheme.

The Tummel Valley scheme consists of nine main reservoirs, coupled to each other by the flow through eight hydro power stations. Each individual station of the scheme has a number of unique constraints which must be placed upon its operation.

The problem of optimising the scheme has been simplified by decomposing the problem in time. The operation of the group was studied in three time scales : long term, or strategic, operational planning of the large reservoirs, daily decoupling of the interactions between the valley's reservoirs, and, the hourly scheduling of the available water.

The method used in the long term planning of the large reservoir, Loch Ericht, was a two stage process based upon stochastic dynamic programming. The method was fast, and enabled annual planning of a reservoir's water resources. The optimisation results were compared to the actual annual strategy used by the NSHEB for Loch Ericht.

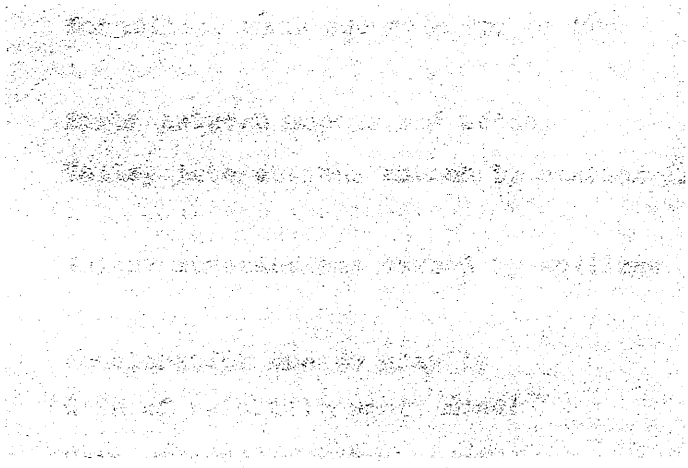
For smaller hydro stations, the water available for generation from day to day is much more dependent upon the natural inflow from the catchment area. The prediction of this natural inflow allows optimal use to be made of the water available in small stations. Methods have been applied to the catchment at Gaur power station which allow a prediction of the inflow to be made from limited data which can be easily collected on site.

The daily decoupling procedure reconciles the daily demand

for power with the optimal water available for generation from each station. The coupling constraints between the reservoirs and other operational constraints are considered. The procedure uses non-linear programming to produce daily outflows from each station, which effectively decouple the reservoirs of the Tummel Valley scheme.

The daily outflow from each station is allocated hourly according to a hourly demand curve to give an hourly schedule for each station. The method of dynamic programming with successive approximations is used to find a solution to the close coupling between Clunie and Pitlochry stations.

A set point controller has been tested for dispatching the hourly schedule produced by the optimisation. Models have been developed of existing Temporary Droop governed hydro turbines. The set point controller has been fully tested against this simulation and real plant for both fast and controlled loading.



List Of Variables

a_i	Auto-regressive parameters of ARMA models
A_j	Sub-region of feasible weekly inflows
AR, ar_j	Area of reservoirs
b_i	Moving Average parameters of ARMA models
BP_k	Benefits from generated power in optimisation
B_j	Sub-region of possible hourly outflows
b_t	Temporary droop of TD governor
b_p	Permanent droop of TD governor
C(I)	Cumulative distribution of probability
c_u	Cost per unit of power
c_b	Cost per unit of power
c_o	Cost per unit of power
c_c	Cost of the change in state of a turbine
c_f	Feasibility constant
c_e	Cost placed upon emptying a reservoir
c_x	Cost placed upon failing to meet a storage target
D_k	Demand curve of merit order costs
DS_k	Sorted demand curve of merit order costs
D^2	Normalised mean square error of PDF
err	Error between models and actual
E, e_{jm}	Valley interactions caused by station flow
F, f_{jm}	Valley interactions caused by spillage
g	Acceleration due to gravity
G_r	Gain of rainfall-runoff model
G_s	Gain of speeder motor model
H_t	Effective head across the turbine
h_w	Height of water of the top of the dam
h_e	Effective head between headpond and tail-race

I_k	Inflow
IN, in_j	Natural inflow or runoff
j	subscript indicating station/reservoir number
k	subscript indicating time period
L, l_j	Level as read from dam
m	subscript indicating station/reservoir number
n_j	Number of samples within A_j
np_j	Theoretical number of samples in A_j
Ns_j	Number of sets in station j
Nh_j	Number of hours station j will generate for
Nu_j	Number of units produced by station j
O, o_j	Outflow from station j
od_j	Daily outflow from station j
P_{jo}	Probability of i_k falling within region A_j
P(I)	Probability distribution of weekly inflow
Pt	Temporary PDF of weekly inflow
PO_c	PDF of weekly outflow from Cuaich station
PO_e	PDF of weekly outflow from Loch Ericht Station
PI_e	PDF of natural inflow to Loch Ericht
PL_k	Penalty for failing to meet storage targets
PX_k	Penalty for spillage or emptying a reservoir
PS_k	Penalty for spillage or loss of head
PF_k	Feasibility Penalty
Po	Power out of the turbine
Popt	Optimum power of a set
Q	Flow through the turbine
Ra	Rainfall

s	Laplace operator
Sx_k	Sum of changes in power
SP, sp_j	Spillage
SCF_j	Spillage to units lost conversion factor
T	Time
T_s	Speeder motor time constant
T_y	TD governor time constant
T_d	TD governor time constant
Tx	Target storage
u_k	Actual units generated
WCF_j	Water to units conversion factor
X, x_j	Storage
z_k	Target units generated
∂_j	Reservoir constant of discharge
ρ	Density of water
Δx	Change in storage
ΔC	Change in station output state (on/off)
$\{x_j\}_{max}$	Maximum storage capacity of the jth reservoir
$\{D_j\}_{max}$	Maximum merit order price per unit

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The North of Scotland Hydro Electric Board (NSHEB) are currently undertaking a program of modernisation and centralisation of their control and monitoring equipment for hydro generating plant. The modernisation must make use of the most up to date digital and computer equipment, for maximum flexibility. The development of computer hardware, particularly in the last few years, results in computers being capable of much more than a simple communication and control role in power systems. The computing power available makes it possible to consider not only how to control plant in real time, but also how best to operate and optimise systems in real time. The optimisation of the power plants must consider all constraints which are placed on the system. In a centralised optimal control scheme, highly constrained sub-systems have a high computational overhead and tend to monopolise the CPU time. One possible solution is to use hierarchical control and distribute the optimisation of the highly constrained sub-systems to lower levels.

1.1. Aims of the project

One such highly constrained sub-system is the Tummel Valley scheme, situated to the west of Pitlochry in Perthshire, Scotland. The scheme has nine main reservoirs, coupled to each other by the flow via eight hydro power stations. The scheme is, at present, controlled manually by Generation Engineers at Errochty Group Control Centre near Tummel Bridge.

The aim of the project was to develop a real time control

system for the Tummel Valley scheme. The control system would optimally schedule the power output of the scheme throughout the day, taking account of the many operating constraints placed on the valley. Under normal operating conditions, the results produced by the optimal control system will follow the same sheduling philosophy as the Generation Engineers. However, the major advantages of an automated system are in its ability to reschedule quickly and accurately on request. The computer system is capable of considering all possible options in deriving the optimum in a very short time. The use of an optimal control system would give rise to better overall control of the Tummel Valley scheme in terms of flexibility and frequency control at both, crisis periods for the national grid, and unique flow patterns within the scheme.

It was envisaged that such a control system would operate as part of a hierarchical system of control for the national grid. At the upper level, an expected power demand for the day is specified. The optimiser then attempts to use the available water to generate power according to this projected demand. The decision on the amount of available water is a complex one which varies from reservoir to reservoir. In large reservoirs, which may be considered to be seasonal storage reservoirs, the available water is decided as a compromise between the strategic plan for the reservoir's storage and the projected need for power. The available water from smaller reservoirs is decided more on the basis of the immediate runoff from the catchment area. When reservoirs of both types are coupled the decisions on available water are made vastly more complicated.

The resulting scheduled use of the water as generated power

was to be implemented by a lower level of the hierarchy. Set point controllers control the power output of each turbine, dispatching the decisions of the main optimising program.

1.2. The Representation of Data and Results.

The complex interactions between the Tummel Valley scheme and the national grid system in meeting consumer demand make an exact objective evaluation of the optimality of the control system impossible in analytic terms. The results of the computer optimisations were, therefore, given to the Generation Engineers for critical assessment.

The whole process of developing and testing the control system was done in close collaboration with the Generation Engineers at Errochty Group Control Centre. The engineers traditionally work in imperial units, therefore all work was carried out in imperial units. A table of conversion to transfer these units to the normal S.I. representation is listed in Table 1-1.

Imperial		S.I.
1 foot (ft)	= 0.305	metres (m)
1 mile (mi)	= 1.609	kilometres (km)
1 square mile (mi^2)	= 2.588	square kilometre (km^2)
1 cubic foot / sec (Cusec)	= 28.288	litres per second (l/s)
1 million cubic feet (Mcf)	= 28.288	million litres (Ml)
1 Mcf/ft	= 0.093	km^2

Table 1-1: Conversion of Imperial units to S.I. units

1.3. Overview of Thesis

The thesis describes the problems involved in controlling the Tummel Valley scheme. In subsequent chapters, particular problems are highlighted and methods of solution given.

The physical disposition of the Tummel Valley scheme is discussed in Chapter 2. Its operational constraints and the existing operational policies are presented in detail , as learned from the Generation Engineers.

The mathematical equations, which are used to model the scheme, are derived in Chapter 3. The chapter then goes on to discuss the current method of costing the generation from hydro power and reviews the optimisation techniques already applied to power system and how they can be applied to this particular problem.

One possible method for long term planning of the storage in

a reservoir is discussed in Chapter 4. The results of which are compared to existing operating policies.

As part of deciding how much water is available for generation it is important to know the amount of run off from the catchment which will result from rainfall events. Chapter 5 looks at several approaches to predicting run off from rainfall data for one particular station of the valley.

Chapter 6 looks at techniques which allow the problem of optimisation to be simplified by decoupling the interactions within the Tummel Valley scheme. This is taken to be an essential prerequisite of the hourly scheduling of plant. The methods used to schedule the power stations hour by hour are discussed in chapter 7. The results of which were reviewed critically by the engineers at Errochty Group Control Centre.

The ultimate dispatch of the optimisation decisions require that an automated set point controller be used. Chapter 8 develops models of Temporary Droop governed hydro turbines and evaluates these models in conjunction with a set point controller developed within the department.

Chapter 9 summarises the progress of the project and suggests areas of further work.

2. THE TUMMEL VALLEY

2.1. Introduction

The Tummel Valley scheme comprises of eight hydro power stations and twelve reservoirs. The cascaded nature of the plant implies complex interactive operational constraints. Additional constraints arise from statutory obligations to environmental groups such as fishery boards. In order to model the scheme it is necessary to document and understand these constraints. This chapter details the plant, the physical, operational and statutory constraints of the scheme [1].

2.2. Plant and Operation

The Tummel Valley catchment area extends over a large part of the southern Grampian Mountains and contains some of the most rugged and desolate areas in Britain. The higher summits and plateau areas have snow cover for many months of the year and they form some of the wettest areas in Scotland.

The Tummel Valley group consists of eight power stations, which use the water from the River Tummel and its tributaries. Fifteen small diversion weirs, eleven large reservoirs, twenty-eight miles of tunnelling, and seventeen miles of aqueducting make up the civil works of the scheme. The oldest two power stations, Rannoch and Tummel, were built by Grampian Electricity Supply Company in the 1930s. The others were built in the 1950s by the North of Scotland Hydro-Electric Board (NSHEB). As an aid to the following sections, which describe the operation of the valley and constraints in detail, a fold out map of the valley and the Tummel

Scheme is shown in Figure 2-1 on page 18.

2.2.1. The Catchments

The logical starting point to describe the scheme is to consider the water available for generation. Nine reservoirs in the scheme have their levels tele-metered to the Errochty Group Control Room at Tummel Bridge. Table 2-1 shows the operational levels for each of these reservoirs, along with their area and maximum storage.

Loch	Absolute Minimum Level (feet)	Minimum Operate. Level (feet)	Maximum Operate. Level (feet)	Loch Area (Mcf/ft)	Maximum Storage (Mcf)
Loch An-T-Seilich	1386.0	1392.0	1400.0	11.1	95.0
Loch Cuaich	1303.0	1303.3	1305.0	5.0	10.0
Loch Garry	1330.0	1340.0	1360.0	14.0	299.0
Loch Ericht	1152.0	1155.0	1174.9	226.1	5710.0
Loch Eigheach	840.0	842.0	850.0	15.0	124.0
Loch Rannoch	664.0	665.0	672.0	204.6	1435.0
Loch Errochty	1040.0	1050.0	1080.0	24.4	795.0
Loch Tummel	465.0	468.0	472.0	63.1	320.0
Loch Faskally	290.0	294.0	300.0	6.9	42.0

Table 2-1: Loch Levels in feet (A.S.M.L), Areas and Storages.

The water taken from the reservoirs for generation is replenished by runoff from the catchment. This natural inflow is dependent on two main factors : catchment size and average precipitation. Both are tabulated in Table 2-2 for each station. Typical inflows are shown in Table 2-3. These are given only as

an indication of what is normal during each season.

Loch	Catchment Area (Sq. mile)	Average Annual Rainfall (mm/yr.)
Loch An-T-Seilich	55.7	1524
Loch Cuaich	2.3	—
Loch Garry	34.5	1778
Loch Ericht	90.5	1676
Loch Eigheach	92.3	1735
Loch Rannoch	106.4	1600
Loch Errochty	90.6	1422
Loch Tummel	41.3	1524
Loch Faskally	198.7	1422

Table 2-2: Area of Catchments and Average Rainfalls for each Reservoir.

Loch	Summer Average Inflow (Mcf/Day)	Summer Peak Inflow (Mcf/Day)	Winter Average Inflow (Mcf/Day)	Winter Peak Inflow (Mcf/Day)
Loch An-T-Seilich	8	40	30	80
Loch Cuaich	—	—	—	—
Loch Garry	8	40	20	50
Loch Ericht	40	120	70	140
Loch Eigheach	20	100	70	160
Loch Rannoch	30	90	70	140
Loch Errochty	30	80	50	120
Loch Tummel	8	20	20	70
Loch Faskally	8	30	60	140

Table 2-3: Typical Average Inflow to each Reservoir.

2.2.2. Stations

The water from the reservoirs is used by one of the eight power stations of the scheme. The generation equipment in each station depends on the amount of water available and the head. Table 2-4 gives the data on the flows, heads and power ratings for each station.

Station	No. Of Sets	Power per Set Optimum (Mw)	Power per Set Maximum (Mw)	Head (ft)	Flow per Unit (Cuft/U)	Max Station Daily Flow (Mcf/Day)
Cuaich	1	2.2	2.5	87	640	38.4
Loch Ericht	1	2.2	2.2	180	370	19.5
Rannoch	3	14.0	14.0	512	105	105.8
Gaur	1	5.3	6.4	90	550	84.5
Tummel	2	16.0	17.0	173	300	244.8
Errochty	3	22.0	25.0	610	91	163.8
Clunie	3	17.0	20.4	165	305	455.3
Pitlochry	2	—	7.5	50	1100	396.0

Table 2-4: Station Data for Stations
Rated Power, Number of Generators, Working Head and Water Flow Rates for each of the Stations of the Tummel Valley Scheme.

Table 2-5 shows the type and manufacturer of the equipment installed in each of the stations.

Station	Turbine Type	Generator Type	Speed (Revs/ Min)	Governor Type
Cuaich	Francis	4-Pole Sync	750	Worm Drive
Loch Ericht	Francis	Induction	3000	Worm Drive
Rannoch	Francis	6-Pole Sync	500	Woodward TD
Gaur	Francis	14-Pole Sync	214	E. Elec. TD
Tummel	Francis	10-Pole Sync	300	Boving TD
Errochty	Francis	7-Pole Sync	428	Boving TD
Clunie	Francis	14-Pole Sync	214	Boving TD
Pitlochry	Kaplan	18-Pole Sync	170	Boving TD

-Pole = number of pole pairs in alternator
Sync = Synchronous generator
TD = Temporary Droop
E. Elec. = English Electric Company

Table 2-5: Station Set Descriptions.

2.2.3. Overview Of The Tummel Valley Scheme

The main long term, or strategic, storage of the Tummel Valley scheme is Loch Ericht, which can store a maximum of 5710.0 Mcf to compensate for seasonal variations in weather. The three small stations, Cuaich, Loch Ericht and Gaur, while not contributing much to the generation of the group, are important in terms of flow control. Rannoch station is the main generating station of the group, with a 75% load factor, that is, the station is expected to generate for 75% of the day. The other two main stations, with a 50% load factor, are Tummel and Clunie. Errochty station is used primarily for peak demand and as such has a low load factor of about 25-30%. Pitlochry station is a run-of-river

station which is used to smooth irregularities in the river flow before it leaves the valley.

2.3. Operational Constraints

The physical shape of the civil works dictates certain operating procedures. A special problem is the social use of the reservoirs by the public, which adds further constraints. The following sections detail these operating procedures and constraints for each station in the Tummel Valley scheme.

2.3.1. Cuaich Power Station and Loch An-T-Seilich

Cuaich hydro-power station is located at the northern edge of the Tummel Valley. Water reaches Cuaich via an aqueduct 1 mile long from its small head pond, Loch Cuaich. It, in turn, is fed via a 4 mile tunnel from the main catchment, whose storage reservoir is Loch An-T-Seilich. The outflow from the power station runs into the northern end of Loch Ericht through an aqueduct 3 miles in length.

The station, when generating, passes 1.6 Mcf/hour into Loch Ericht. A minimum operational level of 1392.0 ft must be maintained in Loch An-T-Seilich to ensure an adequate flow along the tunnel to Loch Cuaich. During the period from approximately early March to late August smolt screens must be placed at the tunnel entrance from Loch An-T-Seilich to protect young fish. These screens are easily damaged by excessive flows, therefore the flow through the tunnel must be restricted to 0.8 Mcf/hour. This restricts the possible number of hours for which the station will be available for generation otherwise the small headpond (Loch Cuaich) will be emptied.

Compensation water of 3.4 Mcf/day must flow from Loch An-T-Seilich down the River Tromie and out of the valley, which is also the route taken by water spilled from the reservoir. The spillage from this reservoir is thus costly as the water spilled is taken out of the valley and lost to the scheme.

Cuaich station is situated high on the plateau of the Grampian mountains and as such most of the winter precipitation falls as snow and is stored in the winter snow pack. In order to ensure that the aqueducts are kept clear of snow and ice during the winter, the station must run for at least 4-6 hours per day. The station and the tunnel intake gate are controlled remotely from Errochty Group Control Room at Tummel Bridge. The station can, therefore, be activated remotely to and from a set power operating point. The control does not, however, allow the power output of the station to be adjusted. This must be done manually at the power station which is near-inaccessible during the winter months.

2.3.2. Loch Ericht Power Station and Loch Garry

Loch Ericht hydro-power station lies on the west bank of Loch Ericht and its isolation makes access difficult in summer and impossible in winter. The station is supplied with water by a 4 mile tunnel/pipe from Loch Garry and its outflow flows directly into the centre section of Loch Ericht. Spillage from Loch Garry flows into the River Garry, to be later re-directed into Loch Errochty.

This station is noted for its unreliablility and its isolation which combine to make repairs both difficult and

expensive.

2.3.3. Rannoch Power Station and Loch Ericht

Rannoch hydro-power station located at the west end of Loch Rannoch was originally built by the Grampian Electricity Supply Company in 1930 as a base load station for local demand. The water from Loch Ericht is taken along a 3 mile combination of tunnel and pipe, then combined with some additional local inflow before flowing through Rannoch station and thence directly into Loch Rannoch. Loch Ericht, being the main storage in the valley is used as long term seasonal storage which allows the valley to generate during periods of drought. 2 Mcf/day is released from Loch Ericht down the River Ericht as compensation water.

Despite refurbishment, age has resulted in some deterioration in the civil works upon which Rannoch Station depends. Although originally designed to generate 16 Mw per set, it has now been limited to 14 Mw in order to reduce the pressure loading on the ageing pipework. Spillage from Loch Ericht erodes the banks of the River Ericht. Further erosion may undermine the foundations of the pipework so spillage must be avoided.

2.3.4. Gaur Power Station and Loch Eigheach

At the west end of the valley, the inflow to Gaur hydro-power station comes from the large catchment of Rannoch Moor. This catchment consists primarily of peat bogs which gives the runoff a highly non-linear response to precipitation. The station, completed in 1953, was the NSHEB's first station designed to be fully automatic. The power operating point must be set manually

at the station. Automation is limited to simply an on/off control. The head pond of the station, Loch Eigheach, is small for the catchment area, as it was originally designed to operate in conjunction with another station at Loch Laidon, which was never built. The reservoir level is, therefore, particularly sensitive to runoff. The outflow from the station, and spill over the dam at Loch Eigheach, flows down the River Gaur to Loch Rannoch.

In addition to the main set, the station also contains a compensation turbine which releases 2.2 Mcf/day as compensation water down the River Gaur. An additional 1.6 Mcf/day is lost through the fish ladder, which, with 70 pools, is the longest in the valley.

2.3.5. Tummel Power Station and Loch Rannoch

Tummel hydro-power station, like Rannoch, was originally built by the Grampian Electricity Supply Company in 1933. It is situated on the River Tummel as it flows into the western end of Loch Tummel. The water for the station is released from Loch Rannoch by remotely controlled gates at Kinlochrannoch. The water then flows through Dunalastair Water to gates just before the Falls of Tummel. The Dunalastair gates feed the water through 3 miles of aqueduct to maintain head by routing the water around the Falls of Tummel. The water takes approximately 1-2 hours to flow from Kinlochrannoch gates to the station. Thus, the gates must be moved 1-2 hours before any changes in the operation of the station, otherwise, Dunalastair Water will spill or empty.

The Falls of Tummel are a local tourist attraction. To

maintain them a compensation of 7.2 Mcf/day is released at Dunalastair Gates and the associated fish ladder. Spillage from the gates is also routed down the River Tummel and into Loch Tummel.

2.3.6. Errochty Power Station and Loch Errochty

At the west end of Loch Tummel, Errochty hydro-power station is fed by a 6 mile tunnel from Loch Errochty. The catchment area of Loch Errochty has been artificially increased by the construction of four diversion weirs and 10 miles of tunnel. This tunnel channels water from the River Bruar, into the River Garry. The intake to Loch Errochty from the River Garry is controlled remotely from Errochty Group Control Room. A compensation water of 2.3 Mcf/day is passed through a compensation turbine, which is situated in its own power house below the dam, at Trinafour. This water and the water spilled over the dam flows down the River Errochty and into the River Garry. The foundations of this small station at Trinafour could easily be weakened if the dam spilled. Therefore, this dam is not allowed to spill. At times of high runoff the River Garry intake tunnel is closed.

Errochty power station is primarily used for peak loads. The operation of this station, although physically controlled from Errochty Group Control Room, is planned by Central Control Room (CCR) in Pitlochry.

2.3.7. Clunie Power Station and Loch Tummel

Clunie hydro-power station is fed by a 1 mile tunnel from Loch Tummel. The water level in Loch Tummel must not fall below

468 ft due to an environmental constraint placed on the reservoir. A compensation turbine at the base of Clunie Dam releases 3.6 Mcf/day down the River Tummel to Loch Faskally. The fish pass of 43 pools releases 2.1 Mcf/day down the same route. Spillage from the reservoir also flows down the River Tummel.

2.3.8. Pitlochry Power Station and Loch Faskally

Loch Faskally is an artifically created reservoir which supplies Pitlochry hydro-power station. The prime function of the reservoir is to smooth the irregularities of flow in the River Tummel caused by Clunie Station. The dam is by-passed by a fish ladder of 35 pools which releases 4.2 Mcf/day down the River Tummel into the River Tay. A compensation set at the base of the dam also releases 1.7 Mcf/day down the river. In order to reduce erosion of the banks of Loch Faskally, when the level of the reservoir rises to 300 ft (A.S.M.L) the drum gates drop a foot which quickly release the excess water down the River Tummel.

The reservoir, since it was created, has become a centre for water sport and angling. This restricts the operational level of the reservoir to greater than 294 ft (A.S.M.L). Downstream of the station has become very popular with salmon fishermen who regularly wade into the middle of the river. Thus, during the fishing season, the turbine output must be held near constant otherwise the changes in flow could easily constitute a danger to these fishermen.

Pitlochry is the last station of the scheme that water flows through before entering the River Tummel, and, later, the River Tay.

2.4. Existing Operation of the Tummel Valley Scheme

The Tummel Valley Scheme is, at present, planned and operated from Errochty Group Control Centre in Tummel Bridge. The Generation Engineers plan the operation of the scheme using their experienced knowledge of expected inflow and target levels.

The daily planning is done by dividing the day into known peaks and troughs in consumer demand. The day is divided into six periods as shown in Table 2-7.

Period	Times
Overnight	23:30 - 07:30
Morning peak	07:30 - 13:30
Afternoon slack	13:30 - 16:30
Tea peak	16:30 - 20:30
Evening slack	20:30 - 22:30
Evening peak	22:30 - 23:30

Table 2-6: Daily Planning Periods.

The Engineers schedule the available power from each station into the six planning periods, taking into account the complex operating constraints of each plant. A different planning schedule is produced for weekends and week days. The variety of plant within the scheme gives rise to individual planning methods for each plant.

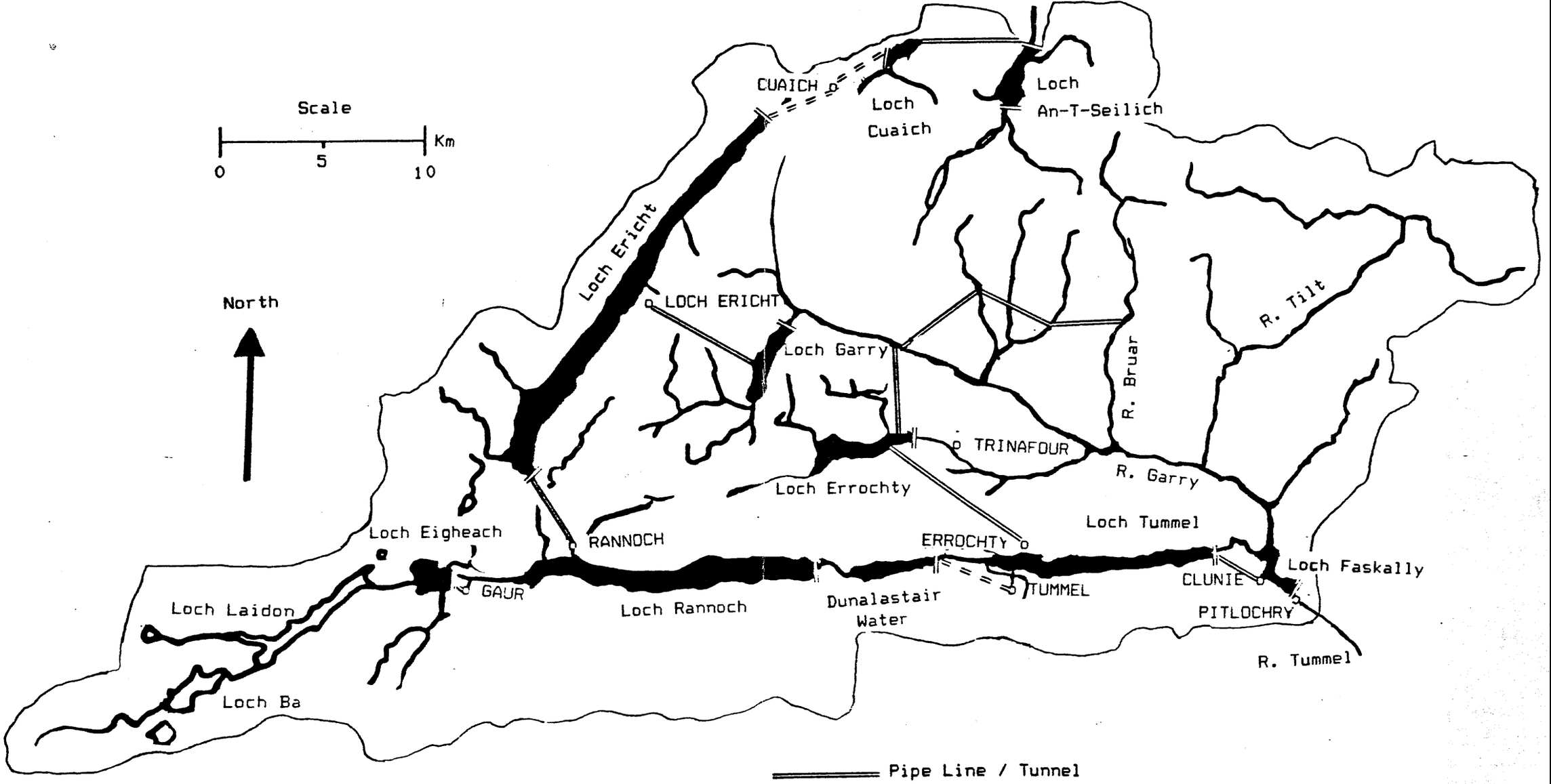


FIGURE 2-1 MAP OF THE TUMMEL VALLEY SCHEME

This map shows the physical configuration of the Tummel Valley Scheme. All the main reservoirs of the scheme are shown and the eight hydro power stations are clearly indicated.

2.4.1. Cuaich, Loch Ericht and Gaur Power Stations

The small size of the storages of Cuaich, Loch Ericht and Gaur station dictates that they be operated as Run-Of-River plant. Any inflow reaching the storage is taken out through the turbines. If the inflow is in excess of the maximum that the station can take out then the station is run until the level can be brought back down to the target level.

Loch An-T-Seilich is kept full in winter to ensure a minimum flow is available to keep the aqueduct clear of snow. In summer the storage is emptied due to the restricted flow through the station.

Loch Ericht station starts generating if the level of Loch Garry rises above 1344 ft and continues to generate until the level falls below 1342 ft.

At times of low flow Gaur is run for 2-4 hours each day. However, if there is a large inflow, or if the level of Loch Eigheach rises sharply then station will run until the level is brought down to below 844 ft.

2.4.2. Rannoch Power Station

Loch Ericht is the main storage of the Tummel Valley scheme. As such, its level is carefully planned to ensure flow within the valley during times of drought. The strategy requires that the level be at 1172.9 ft on the 1st of January and for it to be reduced linearly to 1166.0 ft by the 30th of September. Thereafter, it is linearly increased back to 1172.9 ft by the 1st of January. By this strategy the station would normally be

running all three sets for between 13-18 hours a day.

2.4.3. Tummel Power Station

Tummel Power Station is planned so as to throughput all the water released by Rannoch and Gaur stations down the valley. The level of Loch Rannoch is adjusted for planned maintenance over long periods by slight variations in the number of hours which Tummel runs per day.

2.4.4. Errochty Power Station

The operation of Errochty is dictated by Central Control Room (CCR) in Pitlochry, thus no firm control policy is implemented for this dam. CCR specify in the morning the operation which is required from this station.

2.4.5. Clunie Power Station

In order to allow a margin for flexibility the level of Loch Tummel is always targeted to be 670.0 ft. This allows any plan for Errochty and Clunie to be varied without concern for Loch Tummel. Clunie Power Station is planned to take the flows from Errochty and Tummel down to Loch Faskally.

2.4.6. Pitlochry Power Station

The storage size of Loch Faskally is such that the reservoir varies between its maximum and minimum storages in the course of one day. It is generally planned to have the reservoir at its minimum storage first thing in the morning before Clunie station comes on. This ensures maximum flexibility in the operation of

Clunie station.

Only two changes per day in the output power from Pitlochry are planned. The power is increased to its day-time loading at 6:00 and returns to its night loading at 23:30. The loadings are planned to take the flow from Clunie, plus the additional natural inflow from the River Garry, out of the valley.

2.5. Control from the Central Control Room

The program of operation once formulated at Errochty Group Control Room is passed for approval to the Central Control Room (CCR) in Pitlochry, which controls the whole system for the NSHEB area. CCR will then either accept the total power outputs as sent, or, request a different plan. The program will then be implemented by the engineers from Errochty Group Control Room. CCR in Pitlochry is consulted at each change in operation of any turbine and may request that a set be held on or off longer. A crisis in another part of the system may force CCR in Pitlochry to request some change in the planned operation of the scheme, even if a previous program has already been implemented.

In response to such a request, the engineers at Errochty Group Control Room will attempt first to re-plan Errochty station without changing the operation of any other station. The change in discharge from Errochty will effect the storage of Loch Errochty. The storage will be adjusted in the following day's program of operation.

In the event that the crisis requires further change within the valley, the engineers will then attempt to re-schedule Rannoch Station. Tummel station's operation cannot easily be changed due

to the flow of water from Kinlochrannoch gates, and Clunie station's discharge is very closely tied to the operation of Pitlochry. If, however, the crisis is severe their operation could be temporarily varied and compensated for later in the day.

The relationship between Errochty Group Control Room and the Central Control Room (CCR) in Pitlochry is one of inter-action, trading off the benefit to the group against the benefit to the system. It is, however, CCR which controls the system and as such has the final say in the operation of the Tummel Valley scheme.

3. MODELLING, COSTING AND OPTIMISATION

3.1. Introduction

The object of applying an optimisation method to a problem is to make decisions on the best method of operating the problem system. To ensure that the results of the optimisation are realistically optimum, all optimisation methods must be able to assess the relative benefits from each of the possible decisions. A pre-requisite of optimisation is, therefore, the development of both an accurate mathematical model of the problem system and a realistic penalty function to define the optimum.

3.2. Modelling the Tummel Valley Scheme

The application of optimisation techniques to the Tummel Valley scheme requires that the scheme be described by mathematical equations and variables. This section develops these equations which will be used in later chapters.

3.2.1. Reservoir Models

The most significant variable which can be used to describe any system of reservoirs is their contents. For a reservoir in a cascaded system, the rate of change of its contents is easily obtained by considering the following water balance equation [2],

$$\frac{dx_j(t)}{dt} = in_j(t) + \sum_{m=1}^{n} \left(e_{jm}.o_m(t) + f_{jm}.sp_m(t) \right)$$

—Eq. 3.1.1

where $x_j(t)$ is the storage in the jth reservoir at time t,

$in_j(t)$ is the natural inflow into the reservoir,

$sp_m(t)$ is the spillage from the mth reservoir,

$o_m(t)$ is the outflow from the mth reservoir, and

e_{jm}, f_{jm} are constants which define the reservoir configuration of the cascaded hydro system.

The constants e_{jm} and f_{jm} are derived from the links between each reservoir, and may only take one of the three possible values, [-1, 0, 1]. The values are obtained using the convention of Teneketzis et al. [2], which states that

if o_m flows into reservoir 'j' from reservoir 'm' then $e_{jm}=1$
if o_m flows from reservoir 'j' into reservoir 'm' then $e_{jm}=-1$
if there is no flow between reservoirs 'j' and 'm' then $e_{jm}=0$
f_{jm} is similarly defined for the spillage sp_j

The outflow from each station is limited by the designed maximum flow rate through the turbine. For some stations, the physical construction of the turbine and alternator have resulted in the station having limited operating regions. This can be represented by stating that o_j must be a member of a set of possible outflows.

$$o_j(t) \in \{ B_1, B_2, \ldots, B_n \} \qquad \text{—Eq. 3.1.2}$$

where $B_1..B_n$ are a set of mutually exclusive sub-ranges of outflow.

The eight reservoirs of the Tummel Valley can be more conveniently represented as a group using a modified form of Equation 3.1.1 such as Equation 3.1.3 . The equation is first converted to a matrix form, and the derivative term represented discretely. The flows then represent the total flows within the period T. By considering moving downstream in the valley the

stations can be ordered as in Table 2-5, making the 1st station Cuaich and the 8th Pitlochry.

Let the reservoir storages be represented by the vector $\mathbf{X}(t) = [x_j(t),\ j=1..8]$ and similarly the natural inflow, the outflow and the spillage be denoted by the vectors $\mathbf{IN}(t) = [in_j(t),\ j=1..8]$, $\mathbf{O}(t) = [o_j(t),\ j=1..8]$ and $\mathbf{SP}(t) = [sp_j(t),\ j=1..8]$, respectively. This gives, for the kth period,

$$\mathbf{X}(kT) = \mathbf{X}((k-1)T) + \mathbf{IN}(kT) + \mathbf{E}.\mathbf{O}(kT) + \mathbf{F}.\mathbf{SP}(kT) \qquad \text{---Eq. 3.1.3}$$

where, for the Tummel Valley scheme,

$$\mathbf{E} = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

and,

$$
\mathbf{F} = \begin{bmatrix}
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & -1
\end{bmatrix}
$$

The storage, **X**, has been tabulated by the NSHEB as a function of the levels in the reservoirs, **L**. It was found from these tables that the relationship between **X** and **L** was linear.

$$\mathbf{X}(t) = \mathbf{AR}.\mathbf{L}(t) \qquad \text{—Eq. 3.1.4}$$

where ar_j is the area of the jth reservoir and,

AR is the vector of reservoir areas.

The spillage from each reservoir can be calculated using the broad-crested weir formula [3],

$$sp_j(t) = \partial_j * h_w(t)^{\frac{3}{2}} \qquad \text{—Eq. 3.1.5}$$

where ∂_j is the reservoir constant of discharge, which is a function primarily of the dam geometry,

$h_w(t)$ is the height of the water over the top of the dam, and is defined by,

$$h_w(t) = 0 \qquad \text{if } x_j(t) <= \{x_j\}_{max}$$

and,

$$h_w(t) = (x_j(t) - \{x_j\}_{max})/ar_j \qquad \text{if } x_j(t) > \{x_j\}_{max}$$

—Eq. 3.1.6

where $\{x_j\}_{max}$ is the maximum storage of the jth reservoir.

If the period, T, is of the order of hours then all water in excess of the maximum storage will be spilled within one period. Therefore equation 3.1.6 can be approximated by,

$$sp_j(t) = 0 \qquad \text{if } x_j(t) <= \{x_j\}_{max}$$

and,

$$sp_j(t) = (x_j(t) - \{x_j\}_{max}) \qquad \text{if } x_j(t) > \{x_j\}_{max}$$

—Eq. 3.1.7

3.2.2. Generator Models

The relationship between the water flowing through a turbine and the power produced from the alternator is a complex one [4]. For the purpose of the optimisation work a simpler relationship may be used as power is required only for comparison of costs. The power supplied from each power station can be calculated by considering the potential energy lost by the water as it flows through the station and is given by the equation,

$$po_j(t) = \rho.g.h_e(t).o_j(t) - \text{losses} \qquad \text{—Eq. 3.1.8}$$

where $po_j(t)$ is the power out of the alternator at time t,

ρ is the density of water,

g is the acceleration due to gravity,

$h_e(t)$ is the effective head between the head pond and the tail race, and,

losses are the power losses in the turbine and alternator and is dependent on the number of sets in operation and the type of set.

The form that the equation 3.1.8 takes as a function of head and flow can be found from plant capability curves [5]. An example of which is given for Errochty station in figure 3-1 .

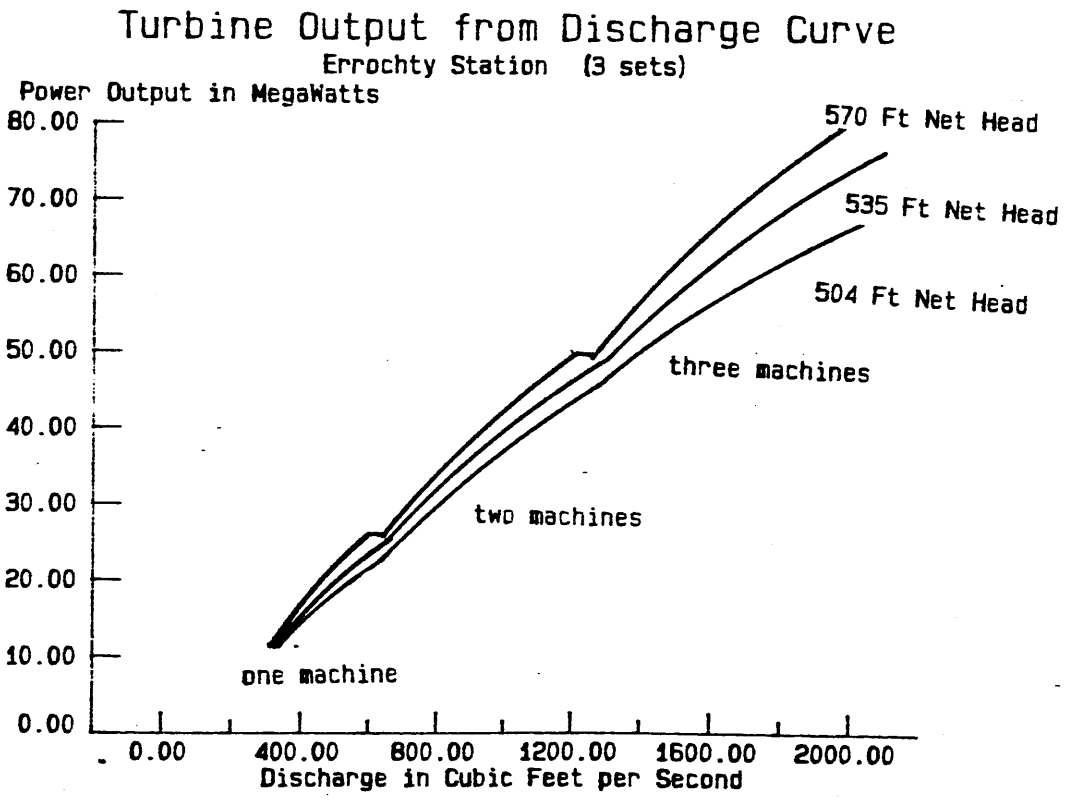


Fig. 3-1: Flow to Power Curve as a Function of Effective Head for Errochty Station with up to three Sets Generating.

The NSHEB use a linear relationship between the water used and the power generated in Units (kilowatt-hour). Average consumption figures are used to produce a Water Conversion Factor (WCF). This allows a simpler relationship than 3.1.8 to be used,

$$\text{No. Of Units} = \frac{\text{Total Flow (Cuft)}}{\text{WCF (Cuft/Unit)}} \qquad \text{— Eq. 3.1.9}$$

or,

$$Nu_j = \frac{o_j(t)}{WCF_j}$$

The WCF_j is tabulated in table 2-4 as flow per unit. The figures are obtained by averaging the performance of the stations over a number of years.

3.3. Costing The Power Output Of Hydro Stations

The purpose of a costing system is to produce a performance index which reflects the rational of operating decisions. The index allows the performance of the plant to be compared from year to year, and aids in the formulation of new or radical planning policies.

In the case of the power generation industry this performance index is purely financial. The industry must meet the consumer demand for power, but receives a price for each unit which is fixed by an outside body. The task of minimising the cost of producing each unit falls to the generation industry.

The actual cost of generating power from hydro plants comprises of two main factors : the repayment of the capital cost of the plant and the cost of operating and maintaining the plant. Hydro plant, however, only contributes less than 25% of the total generation in Scotland [6] and less than 10% of the power requirements of the national grid [7]. The remainder is supplied by plant which has a more significant running cost, fuel.

The representation of hydro in a costing system by its actual production cost does not result in a viable performance index. The actual production cost does not reflect that the major benefit

from hydro is derived from the saving made by not having to use other plant. For example, if hydro produces power overnight, when demand is least, then a base load (or cheap) plant may have to be moved to a less efficient output or turned off completely. Compare this to the situation where the hydro generates only at peak demand times. At these times the hydro plants generation may save expensive plant being used.

The power generation industry has developed a method of artificially pricing the units produced by hydro. The method is used both for scheduling plant and as a performance index, and is known as the Merit Order Scheduling System.

3.3.1. Merit Order Scheduling System

As demand varies throughout the day, plant must be dispatched to match it. Obviously, it is better to allocate the cheapest plant first. A cost per Unit generated can be calculated for each power station which takes account of capital cost, operation and fuel. The stations can then be sorted into ascending order of price to give a merit order. A hypothetical example merit order is shown in Table 3-1.

Station	Price/Unit (Pounds)	Generation capacity (Mw)
Station A (nuclear)	0.011	450-550
Station B (nuclear)	0.012	900-1100
Station C (oil)	0.021	300-500
Station D (oil)	0.024	500-700
Station E (coal)	0.035	1200-1300
Station F (oil)	0.036	700-850
Station G (coal)	0.037	400-600
Station H (coal)	0.041	600-800
Station I (diesel)	0.050	400-800
Station J (gas)	0.058	400-1000

Table 3-1: Sample Merit Order Table showing Cost per Unit and Generation Capacity

It is a simple matter to use the table to find the next to be allocated or shut down as demand changes.

In practice, the table is much larger and much more complex. Additional tables are used in allocating reserve stations. These reserve stations, or spinning spare, have their generators spinning but not actually generating in order that they can be brought on line quickly to cover sudden surges in power demand. Consideration must also be given to the power flow constraints of the overhead lines.

If hydro generation is available then it can replace other plant. The maximum cost saving will be achieved if the plant it replaces is the most expensive, that is, the lowest on the merit order. The units produced by the hydro may then be priced at the

same cost as those of the plant it displaces. This artificial pricing system results in a true performance index for hydro plant.

3.4. Previous Work : Methods of Power System Optimisation

The flexibility in meeting demand and the substantial running costs of a power system combine to make it an ideal application for optimisation. Any improvement in performance can result in significant financial savings[8]. The complexity of even small power systems leads to high dimensionality problems for all optimisation methods[9,10]. Dimensionality may be thought of as the number of variables and constraints which need to be considered. A high dimensionality gives rise to heavy computational requirements, even on large computers. This is reflected in the computation times which have been reported[9,11,12,13,14]. Optimisation methods fall into two main categories : enumeration methods, such as dynamic programming[15,16,17,18,19], and search methods, such as linear[20,21,22,23,24] and non-linear programming[25,26].

3.4.1. Search Methods

Search methods evaluate a performance index for a particular solution of the problem and use this to adjust the solution so as to improve the index while remaining within the feasible solution space. The most widely used of the search methods are linear and non-linear programming. These methods require that each variable be considered explicitly, for example, storage at time kT must be represented as a separate variable from the storage at time (k+1)T. This leads to a large number of variables and

constraints. In order to optimise the Tummel Valley scheme, with its eight stations, hourly over 24 hours requires 192 interdependent variables, and in excess of 768 constraints.

In their linear programming formulations, Maurras et al[27] and Arvanitidis et al[28] attempted to overcome the problem by using a lumped model for reservoirs. The diversity of the physical characteristics of the reservoirs within the Tummel Valley scheme make this approach unrealistic. Other linear programming formulations, such as those of Tyrel[29] and Habibollahzadeh[30], highlight the difficulty of representing inherently non-linear functions as piece wise linear functions. The constraints on outflow, the calculation of spillage, and the representation of power demand can not realistically be approximated by a linearised function.

Non-linear programming relies heavily upon the use of costing function gradients to find the solution. Saha et al[31], Ikura et al[9], and Gagnon et al[11] all suggested formulations using gradient searches. Constraints are incorporated at part of the costing function using lagrangian multipliers[32], as in the methods suggested by Merlin et al[33] and Bertsekas et al[34]. The large number of variables and the use of lagrangian multipliers result in a complex costing function which is difficult to evaluate. The gradients of this complex costing function are difficult to calculate and are normally approximated[25]. These methods are all iterative and lead to long computation times on large computers[12], even for relatively small power systems.

The dependence upon gradients, or linearisation, is a major

drawback of all these methods. If a constraint is discontinuous, or a variable may only hold one of a set of values, (such as outflow Eq 3.4.5) then it is impossible to represent that constraint using linear or non-linear programming. Integer programming[35,36], which optimises with the constraint that some variables may only hold integer values relies heavily upon rounding an unconstrained problem. The branch and bound method[35,37,13] can represent discontinuous constraints. The method successively reduces the solution space by placing upper and lower bounds and branches within the bound region in search of an optimum. The Tummel Valley scheme has 16 turbines. Thus, at any instant in time, there is 2^{16} or 65536 possible combinations of turbines just by considering sets to be either on or off. So over a 24 hour period there is $(2^{16})^{24}$ or 2^{384} possible unique solutions. Here the solution space is so large that even the branch and bound method which sub-divides the solution space can not quickly find an optimum.

3.4.2. Enumeration Methods

Enumeration methods calculate the cost of a number of possible solutions, (effectively enumerating them), and selects the optimum. A large number of possible solutions has to be enumerated unless the problem is simplified.

One such adaption of this type of method is that of the linear network approach of Wakamori et al[38]. A number of small packets of flow are scheduled to pass through a network which represents the actual system. Thus flows are built up to an optimum from an assembly of small packets. The resolution required by the solution of the smallest station determines the

size of the packets. The calculations are, therefore, inefficient for the larger stations which do not require such resolution. The method also suffers from the same drawback as non-linear programming in that it cannot realistically represent discontinuous constraints.

The maximum flexibility in terms of representing constraints has been demonstrated by dynamic programming[39,40]. Although suffering from the 'curse of dimensionality', the method has been applied to a variety of power system problems[41,42,43,10,14]. The method considers how the decisions, or the control variable, effects the system, or the state variable, and finds the optimum sequence of decisions which take the system to some desired state. Various attempts have been made to reduce dimensionality while still retaining the flexibility in representing constraints. The methods of reducing grid[40], progressive optimality[44,45], and successive approximations[46,47,48] all attempt to reduce the dimensionality by iterating within the solution space. The difficulty with these methods is one of conversion. The problem and its constraints must be carefully formulated to ensure that there is a consistent convergence at an optimum solution.

The application of optimisation methods to real systems has shown the greatest success when the approach taken by Pereira et al[49], Le et al[50], Soares et al[52], and Happ et al[51] is used. These methods may be termed heuristic or knowledge based methods. The complexity is reduced by applying simplifications which are derived from an expert knowledge of the problem system. For example, although turbines may generate at a variety of operating points, they are generally only used on, or near, their most efficient power output. Thus, it is possible to say that a

turbine will either be shut down or will generate at its most efficient setting. The simplified system may then be optimised using one of the conventional optimisation methods.

3.5. The Optimisation Procedures used for the Tummel Valley Scheme

By decomposing the system in time, many authors have found that the optimisation problem can be reduced to manageable dimensions[53,54]. It is common to consider two main time scales : long term, or strategic, operation and short term, or tactical, scheduling. The two are nominally linked by the specification of a target storage for each reservoir.

In the case of the Tummel Valley scheme, the problem was decomposed in time to give three separate problems : Strategic reservoir operation, daily decoupling and hourly scheduling. The decomposition of the problem in this way allows additional simplifications to be made about the coupling of the reservoirs.

3.5.1. Strategic Reservoir Operation

The planning of the long term operation of a reservoir is made complex by the uncertainty associated with rainfall events and thus in the the amount of water which will be available for generation. Formulations, such as those of Dondi et al[55] and Agarwar et al[56], which require deterministic inflow data, result in solutions which do not respond optimally to novel inflow sequences. Stochastic Dynamic Programming (SDP) has shown maximum flexibility in solving this planning problem[57,58,59]. The dimensionality of SDP formulations are large, resulting in heavy computational requirements. The further decomposition of the

problem, as described in Chapter 4, reduces the dimensionality substantially and allows SDP to obtain a solution in a relatively short time.

3.5.2. Daily Decoupling Procedure

The object of the daily decoupling procedure is to find the optimum outflow from each reservoir while taking into account the target levels from strategic planning. In this problem, although the number of variables is small, the coupling between them would lead to a high dimensionality problem if dynamic programming were used. The nature of the penalty function, which for the Tummel Valley scheme may be made to be continuous and differentiable, make non-linear programming a more suitable technique. The application of non-linear programming to this problem is discussed in Chapter 6.

3.5.3. Hourly Scheduling

The daily decoupling allows each station to be scheduled independently. The complexity of the hourly scheduling is thus greatly reduced. The problem may be further simplified by noting that generally when a turbine is generating it will be operating at its most efficient power output. Thus it is only necessary to consider if a station is either on-line or off. Larson's [40] methods are applicable and give good results. An Assessment of these results is given in Chapter 7.

4. STRATEGIC WATER MANAGEMENT

4.1. Introduction

The major economic burden on the operation of any power system is the cost of fuel. In meeting consumer demand, the replacement of expensive fossil-fired plant with low cost hydro will produce substantial economic savings. The random nature of water available for hydro generation means, however, that careful long term planning of available water is required in order to take advantage of the cost differential. The problem of realising the potential savings of strategic water management has received considerable attention [60].

This chapter studies the application of optimisation ideas and methods as applied to Loch Ericht, the main storage of the Tummel Valley scheme. The random nature of inflow is simulated using probalistic methods, and the chapter goes on to compare the performance of optimal level strategies with the actual target levels used by the NSHEB.

4.2. Planning Period

In planning the annual operation of a reservoir, the year must be divided into a number of planning periods. The duration of these planning periods is determined by the size of the reservoir in relation to the expected inflow. The period must be such that the reservoir level can be visibly varied from one period to the next, and, that the solution produced has sufficient resolution to enable further planning. For Loch Ericht, the use of a period of one week satisfied these criteria.

4.3. The Probability Distribution of the Weekly Inflow

The NSHEB have recorded, manually, the inflow to each of the reservoirs of the Tummel Valley scheme. This data was copied at Errochty Group Control room and entered into a computer in the laboratory. The data consisted of weekly inflows for the 25 years for which data has been recorded. The data was then used to derive a probability distribution of inflow for each week of the year.

Halliburton et al[61] assumed that the inflows could be approximated by a gaussian distribution, but Wood et al.[62] has shown this only to be valid for periods of a year or more. The distribution is positively skewed for smaller periods. Therefore, non-gaussian distributions were fitted to the inflow data until one was found which was acceptable in terms of a 'goodness of fit' criterion [63]. The 'goodness of fit' criterion may simply be thought of as the mean square error between the distribution of the weekly inflow sample and some proposed theoretical distribution. Consider the testing of some proposed distribution P(x) as a theoretical fit to 25 years of weekly inflow data, $I_1,\ldots,I_{25}$. If the region of possible inflows is divided into k mutually exclusive intervals, $A_1,\ldots,A_k$, then n_j may be used to represent the number of samples from $I_1,\ldots,I_{25}$ in the interval A_j. Similarity, from the proposed distribution P(i), the theoretical number of samples, np_j, which should fall within the region A_j, may be found. If $p_{jo} = p\{i_m \in A_j\}$ is defined as the probability of a sample x_m falling within a region A_j, then for 25 years of inflow data np_j may defined as,

$$np_j = 25 * p_{jo}$$

It is then possible to compute the normalised mean square error between n_j and np_j, given by

$$D^2 = \sum_{j=1}^{k} \frac{(n_j - np_j)^2}{np_j} \qquad \text{—Eq. 4.4.1}$$

If the size of the sample set is large, it has been shown that D^2 has a Chi-square distribution with (k-1) degrees of freedom[63]. Thus the fit of any model distribution can be assessed.

It was found that the gamma distribution given by,

$$P(i) = \frac{\alpha}{(r-1)!}.(\alpha i)^{r-1}.e^{-\alpha i} \qquad \text{— Eq. 4.4.2}$$

had a 43% chance of being correct, ($D^2 = 5.5$), when r equals 2. Compare this to a less than 1% chance, ($D^2 = 32.4$) for a normal distribution. These figures represent average D^2 over all 52 weeks. For r equal to 2, the probability distribution function (PDF) becomes,

$$P(i) = \alpha^2 \; i \; e^{-\alpha i} \qquad \text{— Eq. 4.4.3}$$

where α is known as the distribution parameter and $\alpha > 0$,

and the cumulative distribution function (CDF) or integral of P(x) becomes,

$$C(i) = 1 - e^{-\alpha i}.(1 + \alpha i) \qquad \text{— Eq. 4.4.4}$$

The parameter α of the distribution can be estimated by the maximum likelihood method, which gives

$$\alpha = \frac{2 * 25}{\sum_{j=1}^{25} I_j} \qquad \text{— Eq. 4.4.5}$$

The parameter α was estimated using equation 4.4.5 for all the stations in the valley, over each of the weeks of the year.

4.3.1. Additional Inflows

In addition to natural inflow to Loch Ericht, two other stations, Cuaich and Loch Ericht stations, discharge into the reservoir. The size of the storages behind Cuaich and Loch Ericht Station are small. Thus, in a period of one week all the available inflow must be taken through the stations. The PDF of the discharge into Loch Ericht can, therefore, be approximated by the PDF of the natural inflows to the storages behind these stations. However, the discharge must be limited by the maximum throughput of the turbines which significantly changes the form of the PDF. The additional inflow must be represented by a modified form of equation 4.4.3 .

$P(i) = \alpha^2 \, i \, e^{-\alpha i}$ (i < Maximum Outflow)

P(i) = probability of inflow >= maximum outflow

P(i >= Maximum Outflow) (i = Maximum Outflow)

$P(i) = 0.0$ (i > Maximum Outflow)

The inflows represented by this type of non-linear PDF contribute a significant part of the inflow to Loch Ericht. The actual inflow to Loch Ericht must therefore be given by the sum of three random variables : the natural inflow, the outflow from Loch

Ericht station, and the outflow from Cuaich station. The PDF of the total inflow is calculated by the convolution[63] of the PDF of each of the three random variables which have to be summed. The convolution is more easily calculated using discrete PDFs, and is given by,

$$P_t(i) = \sum_{k=0}^{i} PO_c(k).PO_e(i-k) \qquad \text{— Eq. 4.4.6}$$

$$P(i) = \sum_{k=0}^{i} P_t(k).PI_e(i-k) \qquad \text{— Eq. 4.4.7}$$

where, PO_c is the PDF of the outflow from Cuaich station,

PO_e is the PDF of the outflow from Loch Ericht station,

PI_e is the PDF of the natural inflow to Loch Ericht,

P_t is an intermediate PDF used for calculation,

and P is the PDF of the total inflow to Loch Ericht.

Examples of the PDF P, for a summer and a winter week are given by figure 4-1 and 4-2 . The PDFs were used to formulate expected values in the optimisation procedure.

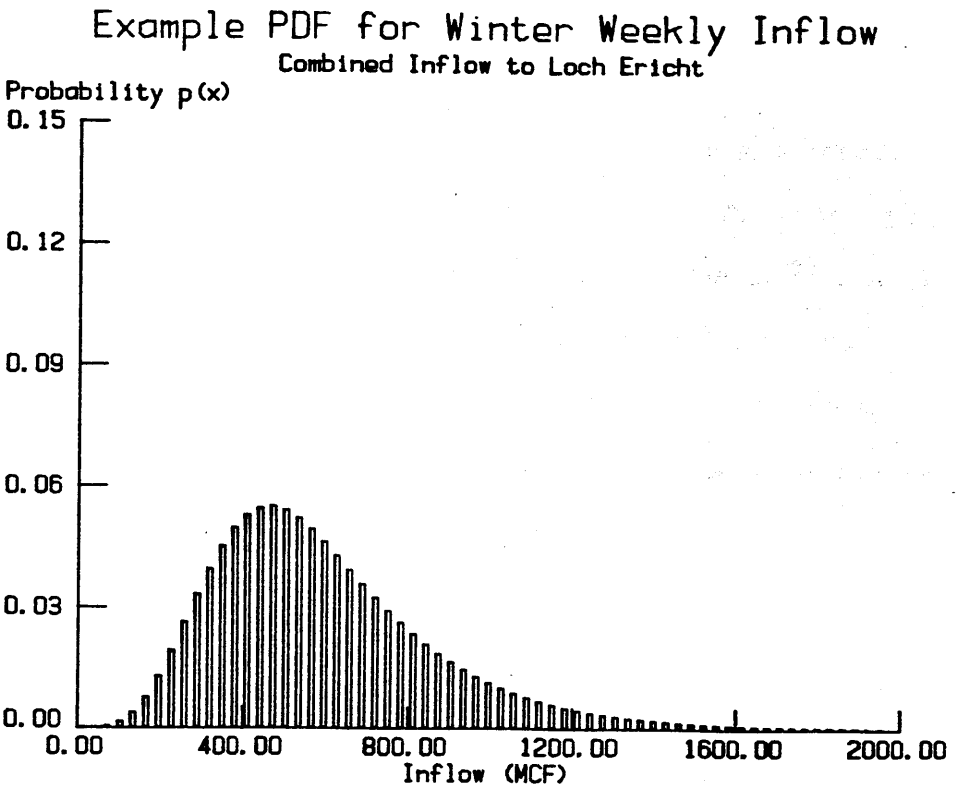


Fig. 4-1: Example PDF of Weekly Inflow for a Winter Week

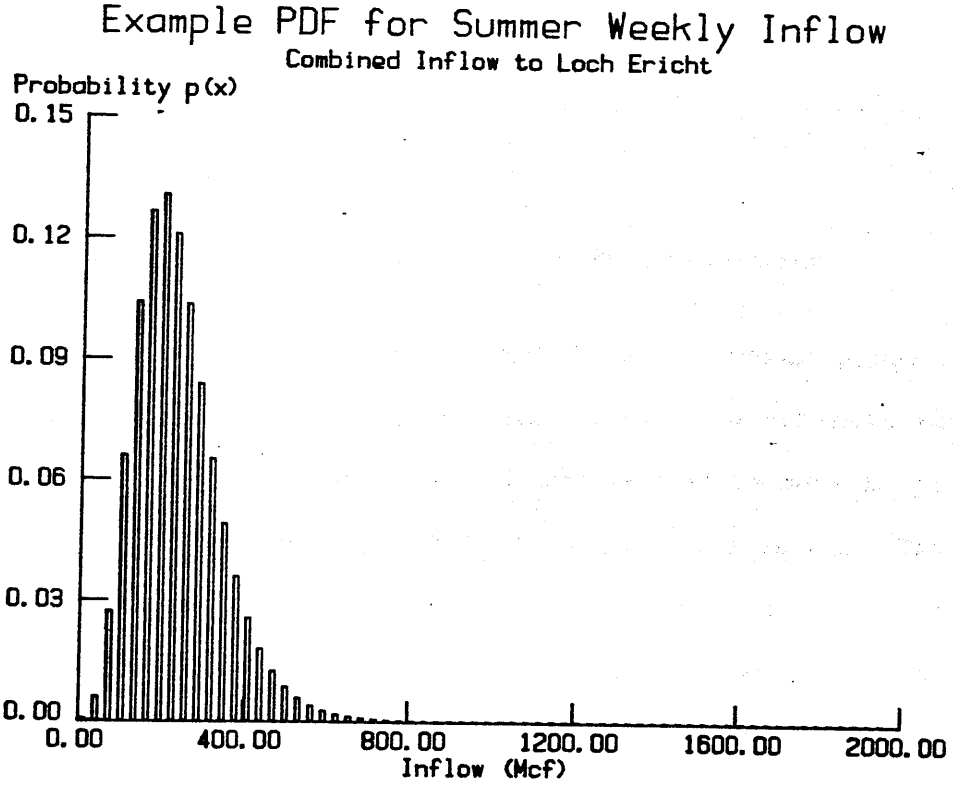


Fig. 4-2: Example PDF of Weekly Inflow for a Summer Week

4.4. Stochastic Dynamic Programming Formulation

In order to optimise any given process a costing function must be defined which defines the penalties and benefits which occur under various operating conditions. The costing function used was based upon that used by Bradley-Russell[57], modified by a feasibility penalty described by Askew[58], and Neto et al[59] as a reliability penalty. The optimisation problem may then be stated as : find the series of states $x_1,\ldots,x_{52}$, which maximise the benefit function,

$$\text{Benefit} = \sum_{k=1}^{52} BP_k - PS_k - PF_k \qquad \text{—Eq. 4.5.1}$$

where, BP_k is the value of the power produced in week k,

PS_k is the penalty due to storage in week k, and,

PF_k is the feasibility penalty in week k.

The sub-costings of the main benefit function are defined in the following sections.

4.4.1. Assessing the Benefit Return from Generation

The benefits to be gained from producing power may be assessed by considering the hydro plant as a separate economic system. Any power which is produced is sold to some larger power system. The price of which is dictated by the demand. The larger system must match the total power generation to the expected consumer demand. To achieve this the larger system will specify a generation target to each of its sub-systems for each time period. Suppose that in the kth period the generation target for the hydro plant is z_k units, and the hydro plant's generation is u_k units. If the hydro plant meets target by generating $u_k = z_k$ units, then

the power will be bought by the main system at a cost of $u_k * c_b$. If, however, the hydro plant fails to reach the target then the main system must make up the deficit. Under this condition the hydro plant is penalised by the system only buying the power generated at a reduced rate of $u_k * c_u$. The hydro plant may supply generation in excess of the target. In which case the main system may be forced to run other plant at a loading with a reduced efficiency and again will penalise the hydro plant by buying the generation at a reduced rate of $u_k * c_o$. Thus BP_k can be defined as,

$$\begin{aligned} BP_k &= u_k * c_b \qquad \text{if } u_k = z_k \\ BP_k &= u_k * c_u \qquad \text{if } u_k < z_k \\ BP_k &= u_k * c_o \qquad \text{if } u_k > z_k. \end{aligned} \qquad \text{—Eq. 4.5.2}$$

In general, it is found that,

$$c_u < c_o < c_b$$

which gives that the benefits from supplying the target power are the greatest. While exceeding the target is more beneficial than not meeting the target. It should be noted that by setting $c_b = c_u = c_o = 0$, it is possible to optimise with respect to no target generations and thus, find optimum generation targets for a sub-system independent of the main system requirements.

4.4.2. Storage Costs

The storage in any period has a cost related to it due to two factors. The water lost to the hydro plant due to spillage must be costed at c_b for the generation it could have produced. Since the power produced is directly proportional to head, the power

lost by not having the maximum head must also be costed at c_b. Therefore, PS_k is defined as,

$$PS_k = c_b * (sp_k * WCF + u_k * (\frac{x_{max}+x_{base}}{x_k + x_{base}} - 1)) \qquad \text{---Eq. 4.5.3}$$

where sp_k is the spillage (eq. 3.1.7),

WCF is the water conversion factor (eq. 3.1.9),

x_{max} is the maximum storage, and,

x_{base} is the storage corresponding to the base head.

The inclusion of this costing function allows power lost to spill to be traded off against the power lost due to a reduced head.

4.4.3. Feasibility Penalties

In the previous two sections the costing functions were defined in terms of u_k, the power produced in the kth period. In obtaining a fixed change in storage over one period, u_k becomes a random variable due to its dependence on the random inflow (eq. 3.1.1). The restrictions on the outflow (eq. 3.1.2) result in a situation where it may not be possible to achieve the desired fixed change in storage. In these circumstances the fixed change in storage may be said to be non-feasible. A feasibility penalty may be introduced into the optimisation process[58] to reflect the probability of a strategy being realistically achieved.

$$PF_k = c_f * P(u_k < 0 \quad OR \quad u_k > Maximum) \qquad \text{---Eq 4.5.4}$$

where c_f is a constant.

The introduction of a feasibility penalty forces the optimal solution to make use of the most probable value of inflow.

4.5. Computation of the Optimal Strategy

The costing function is used in the optimisation procedure to define and compare the relative benefits of various strategies. This section describes the computation of the optimal strategy using stochastic dynamic programming (SDP).

4.5.1. Solution by Enumeration

The basic method of finding the optimal strategy is described by Larson [40]. It considers that the storage, x, is the state variable, and the outflow (or the desired change in storage) is the control variable. The method of enumeration simply calculates the benefits of all possible strategies and selects the most beneficial. The target computer for the program was chosen to be an IBM PC-AT with an 80287 maths coprocessor, due to the high availability of this machine at the university and within the NSHEB. The storage was discretised into 200 states and the outflow was discretised into 24 states. The constants c_u,c_o, and c_b were set to zero to give an optimum individual solution for the reservoir. The optimisation run took approximately 63.5 hours to complete. This computation time is obviously unacceptable.

The results of SDP by enumeration are shown in Figure 4-3. The optimum solution and 4 sub-optimum solutions are shown. The basic form of all the solutions are the same. The optimum solution does not converge to same final value of storage that it specifies as the starting value. Harboe et al[39] has reported that by extending the optimisation to over a number of years, the initial and final value will converge. This would increase the computation time even further.

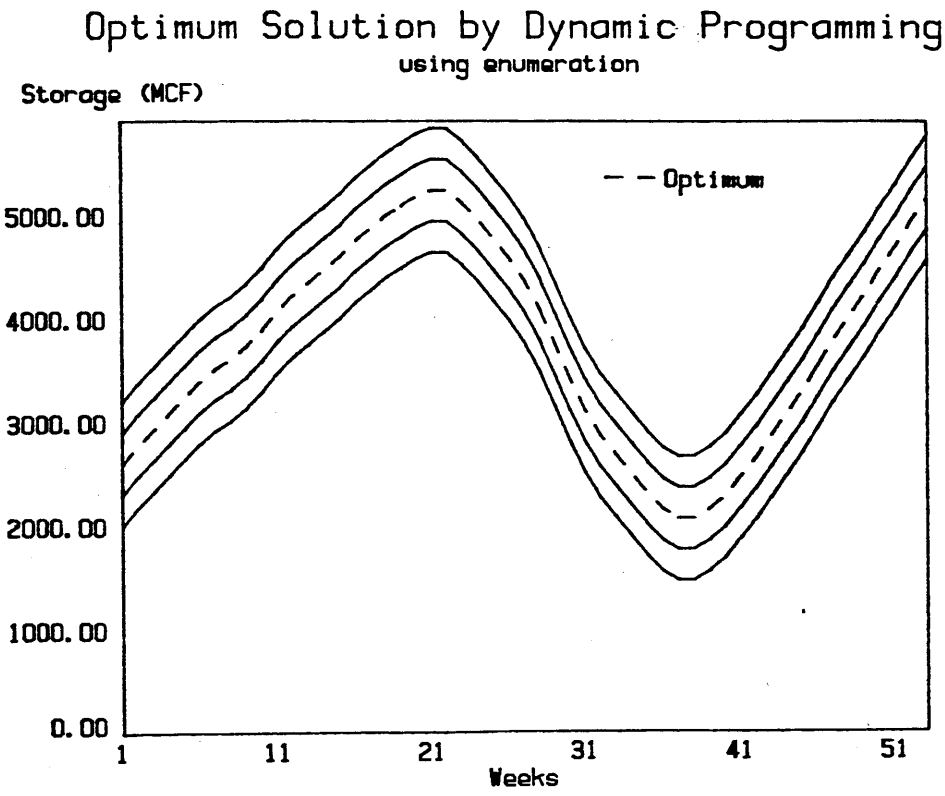


Fig. 4-3: Result of Dynamic Programming using Enumeration

The reduced grid method [40] shrinks the dimensionality of the system, allowing swifter computation. The method lost vital detail as it made use an aggregation of the PDF of the inflow. The reduced resolution of the lumped distribution led to unstable operation of the dynamic programming procedure and resulted in obviously wrong results.

4.5.2. Accelerated Optimal Solution with Adequate Resolution

It was noted from the results shown in Figure 4-3 that the optimum strategy has the same basic form when spillage and the reservoir emptying are not considered. The problem can therefore be separated into a two stage process.

The first stage considers the feasibility penalty and the

generation benefits. These are independent of current storage and need only be calculated once for each different inflow PDF. The optimisation can be performed by using Δx_k as a control variable and Sx_k as a state variable, where Sx_k is defined as,

$$Sx_k = \sum_{j=1}^{k} \Delta x_j$$

By using the addition constraint $Sx_{52} = 0$ then all strategies must finish at the same storage that they started at, ensuring that the solution is stable from one year to the next. This is not the case with the basic enumerated solution of 4.5.1 which must be run over a number of years to ensure annual stability. The constraint $Sx_{52} = 0$ also allows a reduction of the solution search space by a factor of 2.

The reduced search space, in conjunction with the reduced computation required at each node, will vastly reduce the total computation time of the optimisation. The program executed in approximately 9 minutes and resulted in a series of target changes in storage, Δx.

Due to the fact that this optimisation is performed independent of actual storage, the results obtained were a series of relative changes in storage and are not fixed to the actual storage of the reservoir. The second stage of the process is to fix one point of the Δx strategy to the actual storage. All remaining points of the strategy will then also relate to actual storages. The fixed point was obtained by linear search methods which minimised the penalties of equation 4.5.3 . These methods resulted in the highest point of the Δx strategy being fixed very close to the maximum storage of the reservoir. The highest point

normally occurred during the summer so this outcome was to be expected. The results for this accelerated solution method are given in section 4.7.

4.6. Strategic to Tactical Conversion

The optimisation so far has generated only weekly targets. The weekly optimum storage can be interpolated to give target storages for each day. However, this would take no account of the reduced demand for power during the weekend. The use of pre-calculated daily target levels are not practical since the required changes in level over a day are close to the measuring accuracy of the gauges. For example, in the case of Loch Ericht the maximum possible change in the reservoir level per day is 0.2 ft, and the reading error from the level gauge is 0.1 ft.

A more practical method of using the weekly strategic storage plan would be to consider projected target levels nominally to be achieved over one week. If the desired target level and the expected inflow for that period of the year are available, then it is possible to obtain a expected outflow for the week ahead. The projected outflow can then be allocated to the days of the week by assuming that the demand for power at the weekend is 70% of the demand during week days. The daily outflow can easily be calculated by simply dividing up the weekly outflow. This outflow can then be used directly to give the hours for which the turbine should be on or, as discussed in the next chapter, it can be used to calculated the expected storage at the end of the day, which is used as input to the program which optimises operation from day to day.

4.7. Results of the Accelerated Optimisation Procedure

In order to compare the optimisation results to the current strategies, a theoretical model of Loch Ericht reservoir was used. The model used the actual daily inflow data obtained from the NSHEB for the year 1981. Using the method discussed in section 4.6 and the model equations of chapter 3, the operation of Rannoch Station was simulated for one year. The results consisted of the units generated, the water used to generate these units, the spillage from the dam and the financial benefit of the strategy. The financial benefit of the units which were generated was based upon an approximate thermal cost taken from the average demand for power in Scotland during the period 1974-1975.

Three storage strategies were compared. They were :

(i) The actual strategy employed by the NSHEB.
(see section 4.7.1)

(ii) The most likely strategy in light of the most probable inflows.
(see section 4.7.2)

(iii) The strategy attempted to achieve a target demand.
(see section 4.7.3)

4.7.1. The NSHEB Strategy

The NSHEB strategy was simple. It requires that the level of Loch Ericht be at 1172.9 ft on the 1st of January, and for it to be reduced linearly to 1166.0 ft by the 30th of September. Thereafter it is to rise linearly back to 1172.9 ft by the 1st of January.

4.7.2. Most Likely Strategy

In the second strategy, the values of c_b, c_u, c_o were set to 0.0 in equation 4.5.2 . Therefore the optimisation procedure was only concerned with the feasibility penalty, which results in a strategy which supplies its own generation targets and achieves them in the most likely way.

4.7.3. Meeting a Generation Target

The final strategy tried to achieve the generation targets given by the curve of figure 4-4. The feasibility penalty forced a strategy which was the most likely to be achieved in practice.

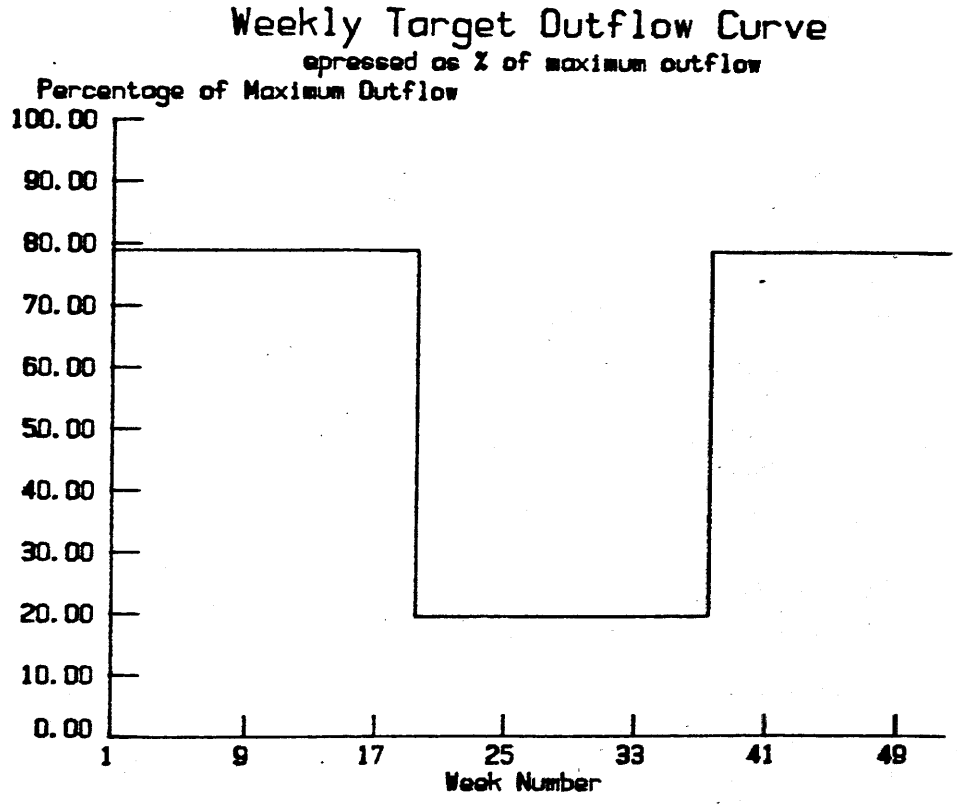


Fig. 4-4: Weekly Generation or Outflow Targets

4.8. Discussion of the Results of Strategy Tests

The results of each of these strategies can be seen by the plots of power output and actual storage which is shown in figures 4-5 to 4-7 for the actual inflow of 1983. The numeric data derived from the simulation is given in Tables 4-1 to 4-3 for the actual inflow for the years 1981 - 1983 .

	Spillage Mcf	Generation Munits	Water Used Mcf
Strategy i	0.00	214.8	19932.1
Strategy ii	0.00	213.6	19846.9
Strategy iii	0.00	216.8	19878.3

Table 4-1: Simulation Results for Actual 1981 Inflow

	Spillage Mcf	Generation Munits	Water Used Mcf
Strategy i	0.00	239.7	22246.8
Strategy ii	0.00	238.8	22161.5
Strategy iii	0.00	241.9	22192.7

Table 4-2: Simulation Results for Actual 1982 Inflow

	Spillage Mcf	Generation Munits	Water Used Mcf
Strategy i	0.00	220.3	20468.3
Strategy ii	0.00	219.7	20383.1
Strategy iii	0.00	222.2	20414.2

Table 4-3: Simulation Results for Actual 1983 Inflow

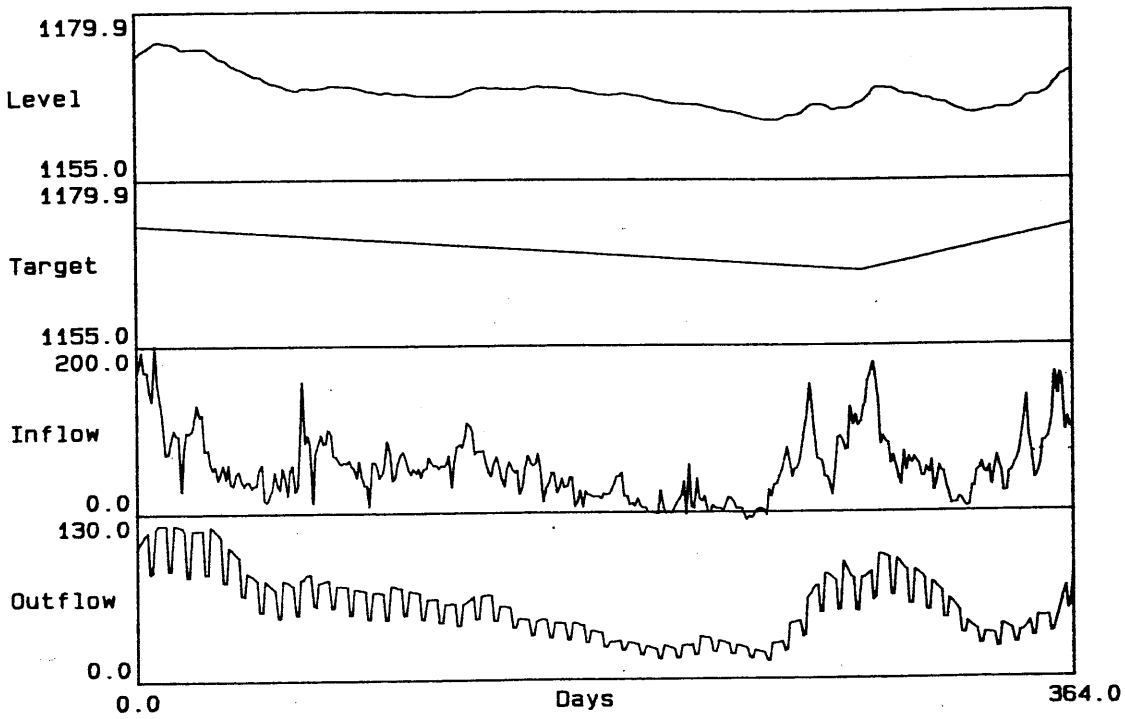


Fig. 4-5: Simulation Results showing Actual Level, Target Level Achieved, Inflow and Outflow for Strategy i using 1983 Inflow Data.

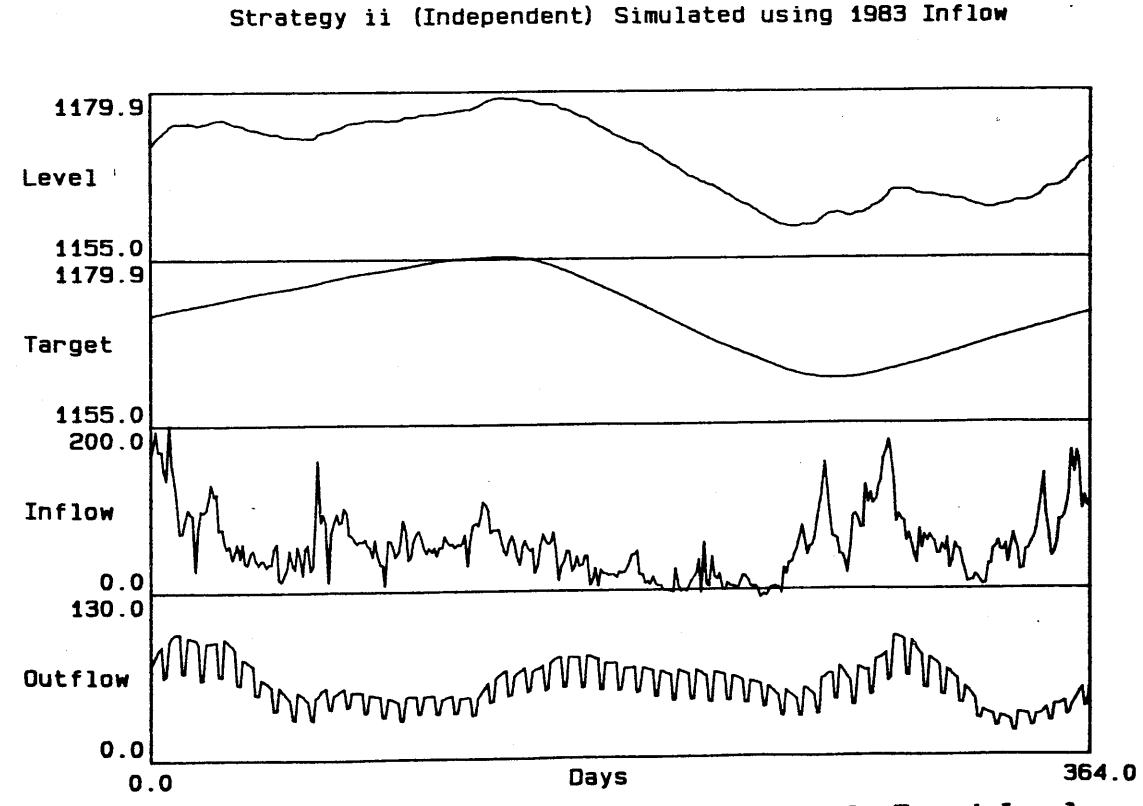


Fig. 4-6: Simulation Results showing Actual Level, Target Level Achieved, Inflow and Outflow for Strategy ii using 1983 Inflow Data.

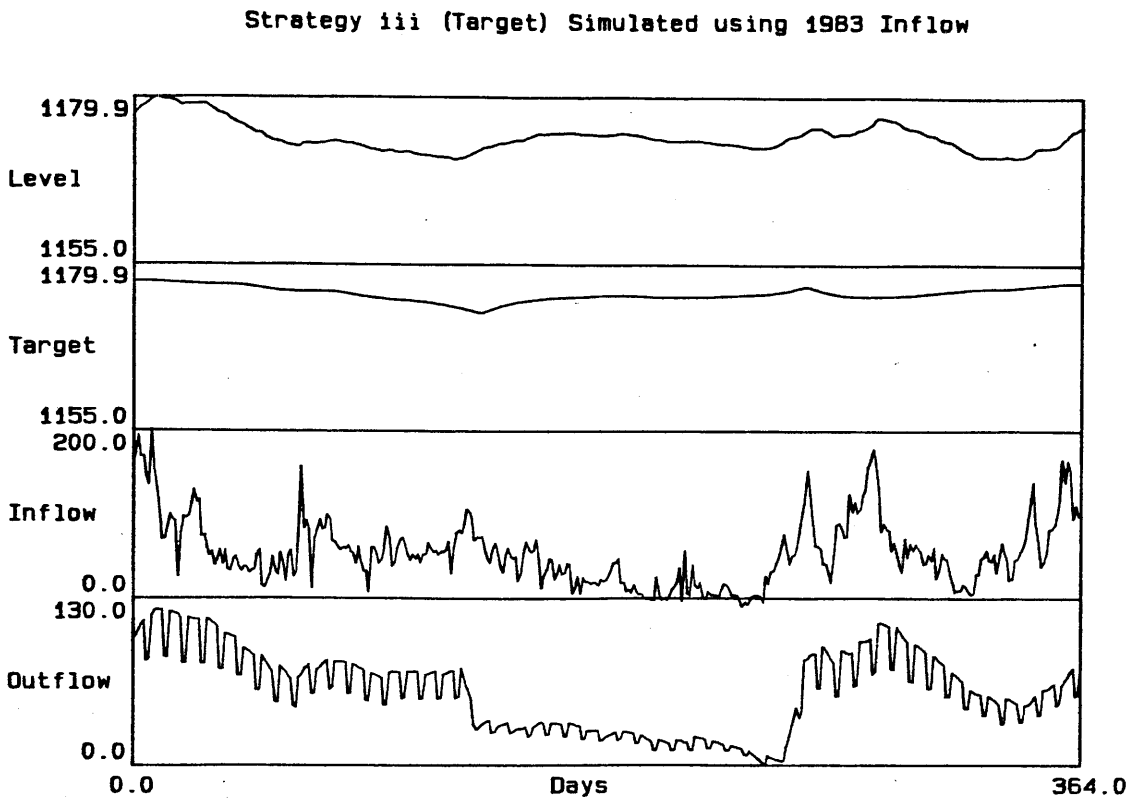


Fig. 4-7: Simulation Results showing Actual Level, Target Level Achieved, Inflow and Outflow for Strategy iii using 1983 Inflow Data.

4.8.1. Conclusion

From Tables 4-1 to 4-3, it can be seen that the predominant factor on the amount of generation produced from year to year is the amount of water available from inflow. The strategies which maintain the highest average head generate the most power within each year.

Strategy iii, meeting a generation target, produces on average about 1% more power that the normal NSHEB strategy. Although this could, in theory, be extremely valuable in terms of the revenue it could raise for the hydro plant, the benefits may not be reflected in the main system. Firstly, the figures take no account of transmission losses. Secondly, as can be seen in Figure 4-7, the

outflow, or generation, tend to follow the pattern of the inflow. Since the inflow may be considered to be a random variable then the generation would also be random. The long term planning of other thermal plant could not be done optimally in light of this random generation.

Strategy ii, the most feasible strategy, results in approximately 0.5% less power being generated than the NSHEB strategy. However, as can be seen from Figure 4-6, the power output is more consistent and less dependent on the inflow than either of the other strategies.

The strategies are basically a trade-off between high head and inflow dependence. The NSHEB strategy is a compromise between the two. The speed and flexibility of this program will allow other strategies to be evaluated which could include additional constraints for planning maintaince.

The methods discussed in this chapter could easily be applied to any major reservoir, including the other reservoirs of the Tummel Valley Scheme.

5. INFLOW PREDICTION

5.1. Introduction

As can be seen from the work of Chapter 4, large storage reservoirs, such as Loch Ericht, can compensate for unexpected natural inflows over periods of a week or more. The amount of water available for generation from these stations is primarily a function of power demand. The Tummel Valley scheme has a number of smaller reservoirs which must respond more rapidly to natural inflow just to avoid spillage. The natural inflow to these stations provides a major contribution to the water available for generation from day to day. One method of improving the operation of such stations is to predict the natural inflow to the reservoir due to rainfall and thus enable optimum use to be made of the water which will be available.

This Chapter describes attempts to model adaptively the water catchment of a hydro power station with a view to providing on-line predictions of water inflow into the headpond. Using limited data which can be conveniently recorded on-site, various adaptive modelling techniques have been applied, with their resultant inflow predictions being compared to the recorded events. Currently, a technique based on the selection of a low order linear model from a set of models shows the most promise.

5.2. Gaur Catchment

A number of adaptive modelling techniques were applied to the catchment of Gaur hydro power station with a view to providing on-line prediction of water inflow into its headpond - Loch Eigheach.

Modelling inflow is particularly relevant at Gaur. Extended periods of wet weather can lead to peaks of inflow which can exceed the generator throughput. Hence a forecast of inflow could be used to select an optimum generation strategy which would minimise water loss due to spillage and still avoid the alternative vice of maintaining a low level in the headpond.

The mismatch of catchment (92.3 sq. miles) and pondage (124.0 Mcf) at Gaur lead to marked changes in water level at the dam over short periods (several hours). The problem of avoiding spillage is further complicated by the maximum throughput of the turbine being less than the peak inflow. This apparent anomaly in design is due to the station having been planned to operate in series with other stations upstream which were proposed but never constructed.

At present, the control strategy of the station is to operate Loch Eigheach at a sufficiently low level to accommodate peak inflow without substantial spillage. Selection of the appropriate level being based on operator experience and seasonal trends in the weather. Unfortunately, a drop in head of 3 ft in 90 ft reduces the power output from the turbine by approximately 3%. So, operation at unnecessarily low levels may cause loss of energy in excess of any saving from decreased spillage.

The daily statistics for one year's operation of Gaur were used to establish the energy generated and spill created by the operator. This was then compared with other operating strategies based on a forward knowledge of inflow. Not surprisingly substantial increases in generation and decreases in spillage could be achieved in such ideal circumstances.

Consequently it seemed desirable to predict inflow into Loch Eigheach, and hence permit an optimum Loch level to be computed which would balance the probability of spillage against head loss[64]. It was hoped that this could be achieved by modelling the catchment area. Measured and forecast rainfall would be supplied as inputs to this model, the output being the desired inflow predictions.

5.3. The Catchment Area

The catchment area of Gaur Power Station comprises mainly of the peat bogs of Rannoch Moor, which leads to a markedly time varying response to rain. During periods of drought, the moor stores any rainfall, only releasing it slowly, whereas after an extended period of wet weather any additional precipitation quickly finds its way into Loch Eigheach. It is during such wet periods that the peak inflow can exceed the maximum throughput of the turbine. In winter, snow, the forming of snow fields, and snow melting complicate the mechanisms even further[65]. The transient response to a rain pulse is of the order of 1-8 hours, but during periods of drought such rain pulses may never appear as inflow pulses.

Clearly an adaptive model is required to follow the changing characteristics of the moor, and inflow predictions will require to be updated at least on an hourly basis. It is also important that an attempt is made to model the catchment and predict the inflow using the limited data which can be conveniently monitored on-site.

5.4. Data Collection

Gaur Power Station is normally unmanned and is operated remotely from Group Control, Tummel Bridge. Loch level and generated power are telemetered back to the control room but rainfall is only measured manually - normally once per weekday. Hence to provide adequate data for the study an automatic tipping bucket rain gauge and a microcomputer based data logger were installed at Gaur in Spring 1985.

5.4.1. Data Logging Equipment

The data logging requirements at Gaur meant that custom equipment had to be built, based around a micro-computer. A schematic diagram of the data logging equipment is shown in Figure 5-1 on the next page.

A pressure sensor placed in Loch Eigheach, below the inlet pipe for the power station, supplies the station communication equipment with a current which represents the reservoir level. Similarly, voltage and current transformers supply a current which represents the turbine output power. These currents were passed through a resistance to give voltages which could be logged. These voltages, representing the reservoir level and turbine power output, were taken from the stations communication equipment via opto-isolation amplifiers in order that the logging equipment could in no way interfere with the stations operation. A 12-bit analogue to digital converter (ADC) allowed the computer to log these signals.

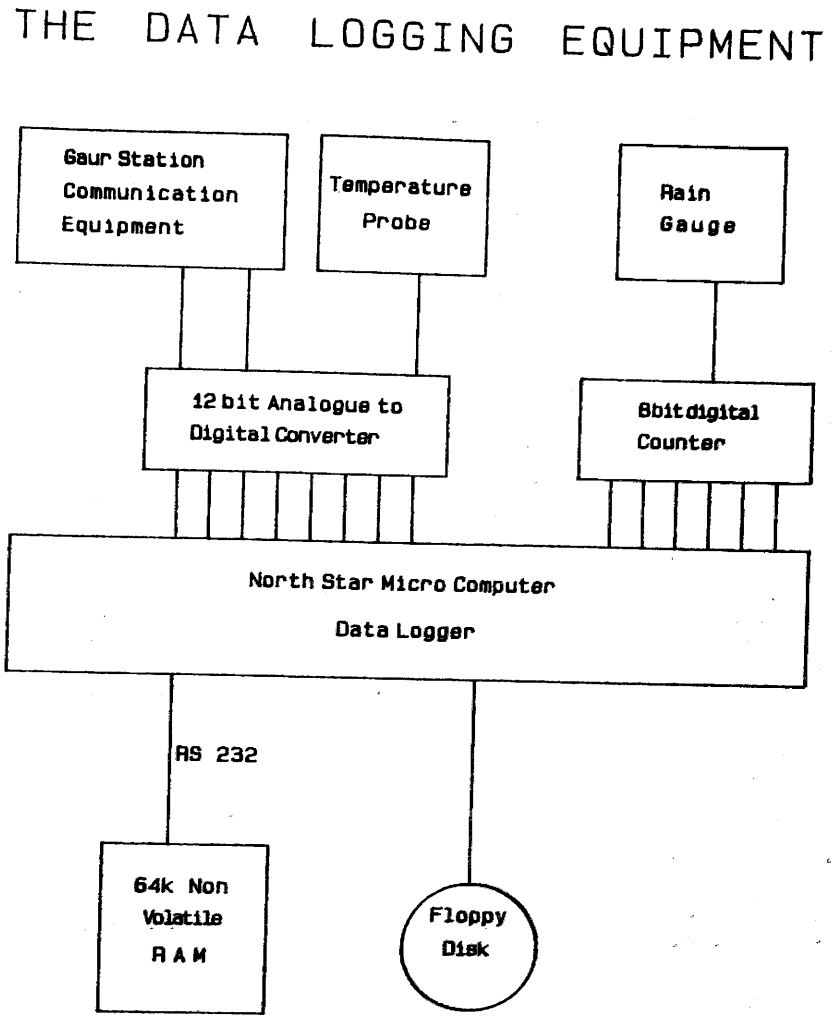


Fig. 5-1: Data Logging Equipment Installed At Gaur

A tipping bucket type rain gauge was installed near the station. The tipping bucket causes two contacts to close for each 0.2mm of rain which falls. This pulse was then passed through an opto-isolation amplifier to isolate this outdoor installation and protect the logging computer. A 8-bit counter summed the pulses and the count was read into the computer via a parallel port.

A temperature probe with a calibration amplifier were installed before the onset of winter in 1985. The ADC was again used to give the main computer access to this data.

The data logging computer was a standard North Star micro computer which logged the data to its floppy disks. A remote power station is a harsh environment for any computing system to operate reliably. For example, the station's power supplies are electrically very noisy, the station itself is very dirty (compared to a laboratory), and the temperature within the station varies greatly depending if the turbine is generating. The most vulnerable part of the logging equipment to the environment was the floppy disks and drives. Therefore, as a back-up storage media, a box was designed and built which contained 64k of non-volatile RAM (NVR). A single chip micro-processor was programmed to supervise the NVR and communicate with the host logging computer through a RS232 serial interface. The data was transferred between the NVR and the logging computer in blocks and a checksum was used to ensure the validity of the data. By communicating with the NVR serially, the NVR was compatable with most computer system, which allowed easy transfer of logged data.

The logging equipment has proved itself very robust. The equipment has at present been logging at Gaur for over 18 months without failure.

5.4.2. Data Processing

The signal representing the reservoir level could be measured to such an accuracy that the effects of surface waves could be seen. The positioning of the pressure sensor was such that the

level reading was artificially reduced if the turbine was on and fluctuated wildly when the turbine was turned on or off. The logging equipment stored the reservoir level to disk every half hour. However, in order to reduce the effects of the surface waves and turbine noise, the level was logged every second and an average value stored.

The power out of the turbine is logged every half hour to floppy disk. The flows through the turbine, when generating, are such that a significant amount of water can be released in a few minutes. The logging computer, therefore, reads the power out each minute and stores the minute that the turbine changed state between generating and off. Thus allowing the flow through the turbine to be calculated to the nearest minute of operation.

The rain pulse count is read via a parallel port and logged each half hour, along with the temperature reading. The storage capacity of the floppy disks was such that the equipment only had to be visited once every two months to change the disks.

The rainfall measurement at the power station is used as the input to the model, while the output from the model is the inflow into Loch Eigheach which can be computed from the measurements of loch level and the generated power. Clearly the rainfall will vary over the catchment area, and ideally rainfall measurements should be taken from a representative sample of points throughout the catchment area, however it was one of the constraints of this study that only data which could be conveniently monitored be used. Remote rain gauges spread over Rannoch moor did not fall into this category. The introduction of Radar weather, as installed in England and Wales, would vastly improve the accuracy

of the measured rainfall over the catchment[66].

Figure 5-2 displays one month of data as logged on-site. At the start of the period there has been prolonged rainfall and the resultant inflow pulse can be clearly seen. During the remainder of the month there are smaller inflow pulses corresponding to extended rainfall periods, but shorter rainbursts have little apparent effect. We can also see that the loch level has been maintained at a rather low level during most of the month, probably a conservative operator reaction to the intense period of rain at the end of the previous month.

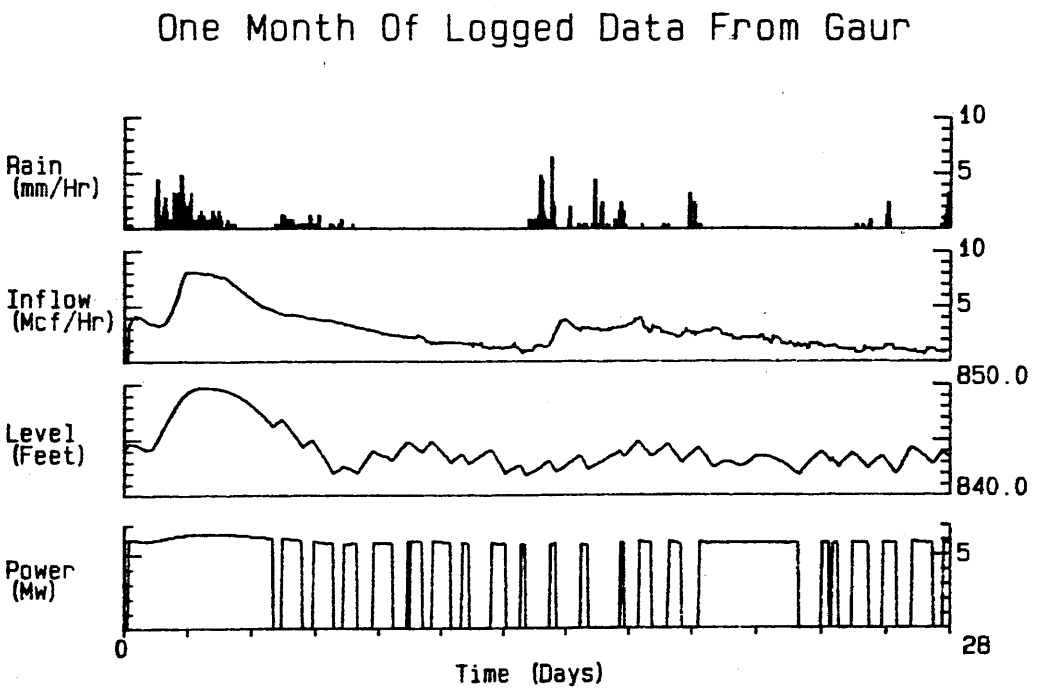


Fig. 5-2: Source Data Logged From Site

Using this collected data, off-line attempts have been made to model the catchment using various adaptive techniques, and the

results evaluated by comparing the predicted inflow with the measured values. Once a sound predictor has been established, it could operate on-line, the computation being carried out in the computer used to log the data, with rain forecasts also being entered to improve the prediction.

5.5. Modelling the Catchment

5.5.1. Physical Models

It has been shown that the land surface processes of a catchment can be accurately modelled[3,67,68,69,70]. These models try to simulate the physical processes which take place within the catchment. Precipitation may fall on a catchment in the form of rain or snow. If it is in the form of snow then it will accumulate on the snow-pack until the energy balance is such that the snow starts to melt. All forms of precipitation will be intercepted by vegetation. Once the volume of the interception storage is exceeded, the excess will fall to the ground surface. If the soil moisture conditions create an infiltration capacity in excess of this throughfall then all the precipitation falling to the ground surface will be absorbed by the soil, adding to the soil moisture storage. Where the infiltration capacity is less than the incoming precipitation, the excess water will form surface flow. Evaporation will take place from each of the possible catchment storages, but will be restricted from the soil moisture where vegetation will have to draw moisture from the soil and transpire it into the atmosphere. Water from the surface flow and the soil moisture storage then flows into the channel network of the catchment and into the main storage reservoir.

This type of catchment modelling is very difficult to estimate parameters for, and requires observations covering an extended period, from a dense network of stations, in order that reasonable estimates can be obtained. Such high quality data could not be practicably obtained at Gaur, hence full physical modelling could not be attempted. Therefore, it was decided to attempt to fit simple empirical models to the limited data set. These models used adaptive methods to follow the time varying parameters of the catchment.

5.5.2. Decision Based Models

One adaptive approach investigated used a decision based model. This attempted to induce a set of rules which related output (inflow) to input (rainfall). As each new situation arises, the rule base is updated.

The inflow to a headpond can be classified into a number of discrete levels and then for each of the classes there will exist a number of attributes which will describe the current state of the catchment area. From the data set of all these attributes rules can be formulated which will differentiate between the classes of inflow.

In the present problem the inflow was divided up into twenty evenly spaced classes of 10 Mcf. Six attributes were attached to each class. The attributes were: the two previous inflow classes and, the rainfall averages of the previous 3, 6, 12, and 24 hours.

A computerised induction routine is used to seek out rules which will successfully pin point inflow classes from their

attributes. The routine used was based upon that of Quinlan[71]. The induction system was given a training sub-set of 700 half hour samples. This produced a decision tree of 668 rules with 72 misclassifications. The misclassifications were due to noisy data which caused similar attributes to produce differing classes. When the next sub-set of 700 samples was examined with respect to the existing rules the result was: 362 data samples which were unclassified and, 223 data samples which were wrongly classified. The rules of the existing decision tree were then modified and supplemented using these data samples. This resulted in the formulation of 623 new or modified rules. As more data sub-sets were included, the number of rules grew by approximately 85% of the sample set size. Typically, when used for prediction on unknown data samples, the model classified over 70% of the data samples wrongly.

Thus, it can be seen that although the decision model worked very well on the data from which the rules were formed, each new data set needed more. The failure is due, primarily, to noise corrupting the classes and attributes and, secondly,to the processes in the catchment being so complex to define that no simple set of rules could be found to describe it.

5.5.3. On-Line Parameter Estimation

Keisel[72] suggested that hydrological data can be represented by a time series model. Loucks et al[73] were satisfied with the performance of auto-regressive moving average (ARMA) models[74] for seasonal or annual average inflows but showed that, on shorter timescales the ARMA parameters were not stationary. To cope with these cases we proposed to implement a simple linear model and

allow its parameters to adapt by using the on-line estimation technique of Peterka[75]. The estimator uses a model of the form:

$$I_t = \sum_{i=1}^{n} a_i * I_{t-i} + \sum_{j=1}^{m} b_j * Ra_{t-j-d} + e_t \qquad \text{—Eq. 5.5.1}$$

Where I_t is the inflow at time t,

a_i are the AR parameters,

Ra_t is the rainfall at time t,

d is the delay factor for the rainfall, and,

e_t is the error between the model and the actual inflow.

Parameters a and b are re-estimated each time a new sample of data becomes available. A technique of multivariate regression minimises the error $\sum e^2$ over the sample set. The error is weighted to make the most recent sample more important. The weighting factor is exponential which results in only the last, k say, samples being considered by the estimator.

It was hoped that the self tuning would allow the model to track the variations of the parameters with rainfall. The self tuning was not a success, the model appears to be ill-conditioned. A major failure is the model's inability to cope with periods of no rainfall. The estimator becomes unstable in situations of low excitation. The ill-conditioning was confirmed by calculating the condition number of the estimating matrix. The value of this number must be less than 100 for meaningful results. In the present case the condition number had a value of 10^7. The self tuning technique seems to be inapplicable to the present system.

5.5.4. Selection From Fixed Linear Models

The two sophisticated techniques have both failed for the same reasons viz: The catchment changes so dramatically from rain event to rain event that short data sets do not contain enough defining information about the system for the 'automatic' techniques to intercept. It becomes clear therefore, that as much of the model as is possible must be specified in advance.

A selection system was constructed which chooses a predefined sub model using some 'expert' like decision based on the current data. For simplicity, the sub models were limited to 1st and 2nd order linear types with transport delay. The forms of the sub models are shown below:

$$\text{First Order} \quad I(s)= \frac{G_r}{(1+sT_r)} Ra(s)e^{-sd} \qquad \text{—Eq. 5.5.2}$$

$$\text{Second Order} \quad I(s)= \frac{G_r}{(1+sT_r)^2} Ra(s)e^{-sd} \qquad \text{—Eq. 5.5.3}$$

where G_r and T_r are the gain and time constant of the model.

In order to facilitate the adaption between models, they were expressed as z-transform equations:

$$\text{First Order} \quad I_t=a_1*I_{t-1}+b_1*Ra_{t-d} \qquad \text{—Eq 5.5.4}$$

$$\text{Second Order} \quad I_t=a_1*I_{t-1}+a_2*I_{t-2}+b_1*Ra_{t-d-1} \qquad \text{—Eq 5.5.5}$$

The system of prediction used three sub models, each sub model fitting a particular rainfall/inflow pattern. The sub models were

set up by fitting them to an ensemble of similar patterns of rainfall collected from the data. In operation,the decision tree picks a sub model which begins the prediction. The system is given the rainfall data 24 hours in advance to mimic a weather forecast. The decision tree can change sub models during the prediction if conditions change sufficiently. The data set shown in Figure 5-3 was used in the evaluation of the model.

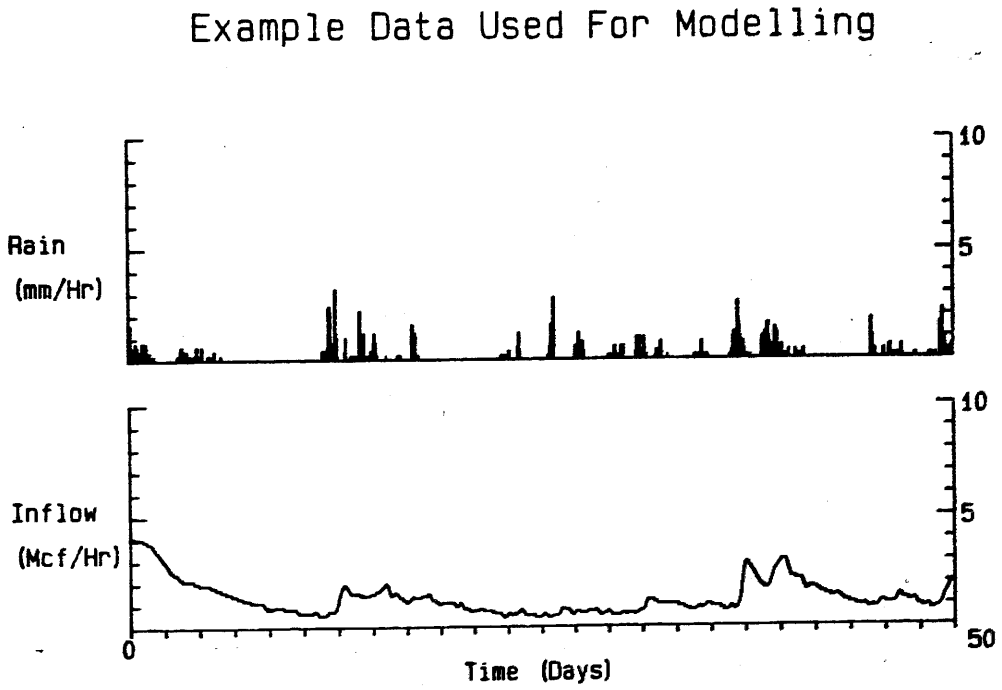


Fig. 5-3: Data Used To Evaluate the Models

The system was evaluated using plots of the errors accumulated after 6,12,and 24 hours. As a comparison, the errors were also compared with those from the simplest method of prediction, i.e., assuming the current value persists. The error between the actual inflow and that predicted at 24 hours using a persistence model is shown in figure 5-4.

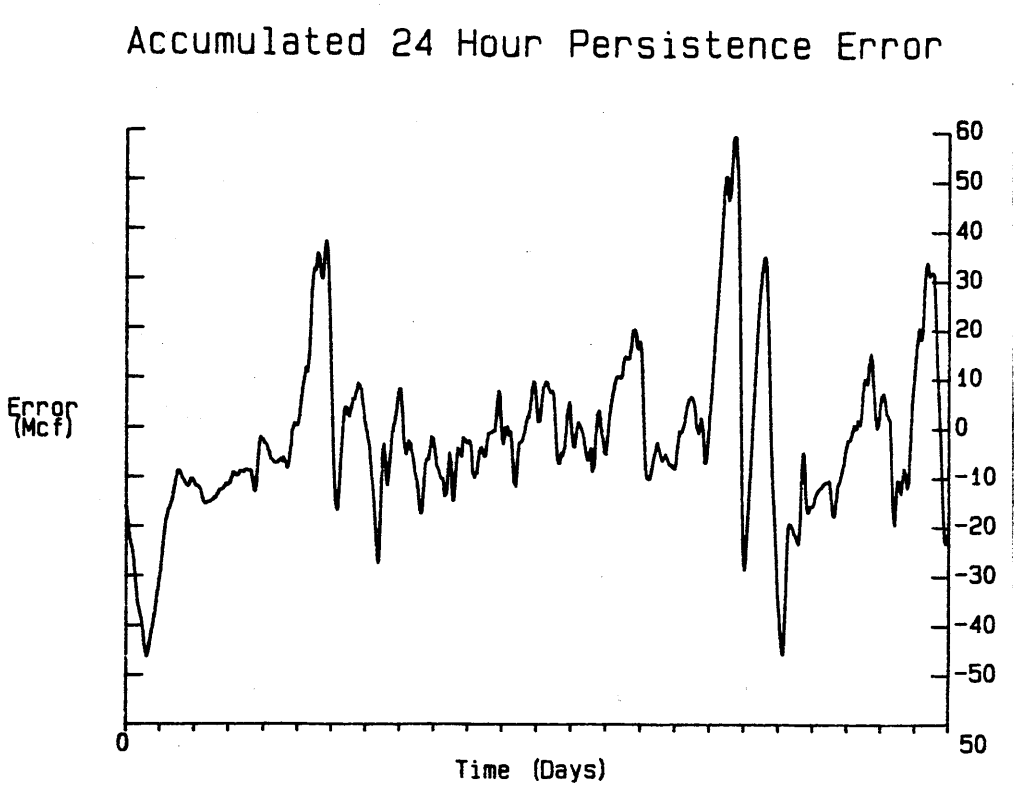


Fig. 5-4: Persistence Error Accumulated Over 24 Hours

The fitting of the sub models to individual events was very successful. Periods of little or no rain and the recession of inflow at the end of a rain event were best fitted by first order models. Second order models, with short time constants, were able to model storm events and periods of steady rainfall. However, the second order models were very sensitive to initial conditions, in particular to the rate of change of inflow. Thus they tended to magnify any error in the rate of change of inflow introduced by the previous model. First order models, on the other hand, do not have this sensitivity. Thus the final system contained only first order models. Some results from this final system are presented in figure 5-5 to 5-7. which shows the error between the actual inflow and that predicted at 6,12 and 24 hours. The errors

increase rapidly over the second 12 hour period. But even in the worst 24 hour prediction with an error of nearly 45 Mcf the error in level at the reservoir is approximately 3 ft. The model after 24 hours is also visibly better than persistence showing that it is indeed tracking some of the variations of the catchment.

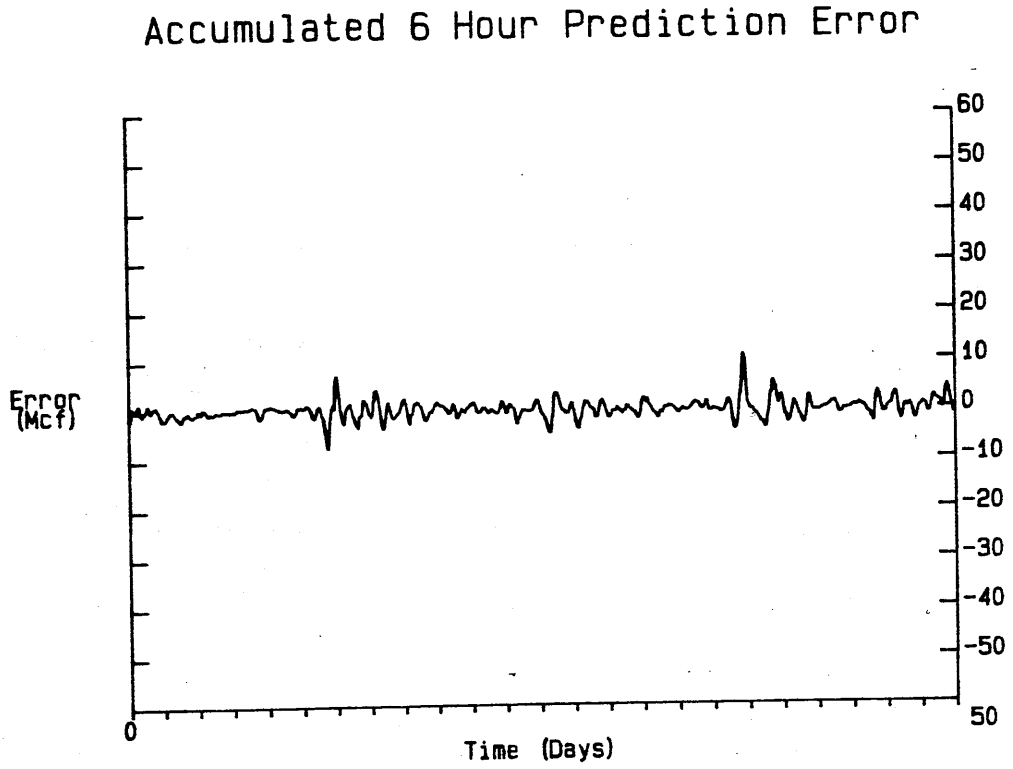


Fig. 5-5: Selective Model Error Accumulated Over 6 Hours

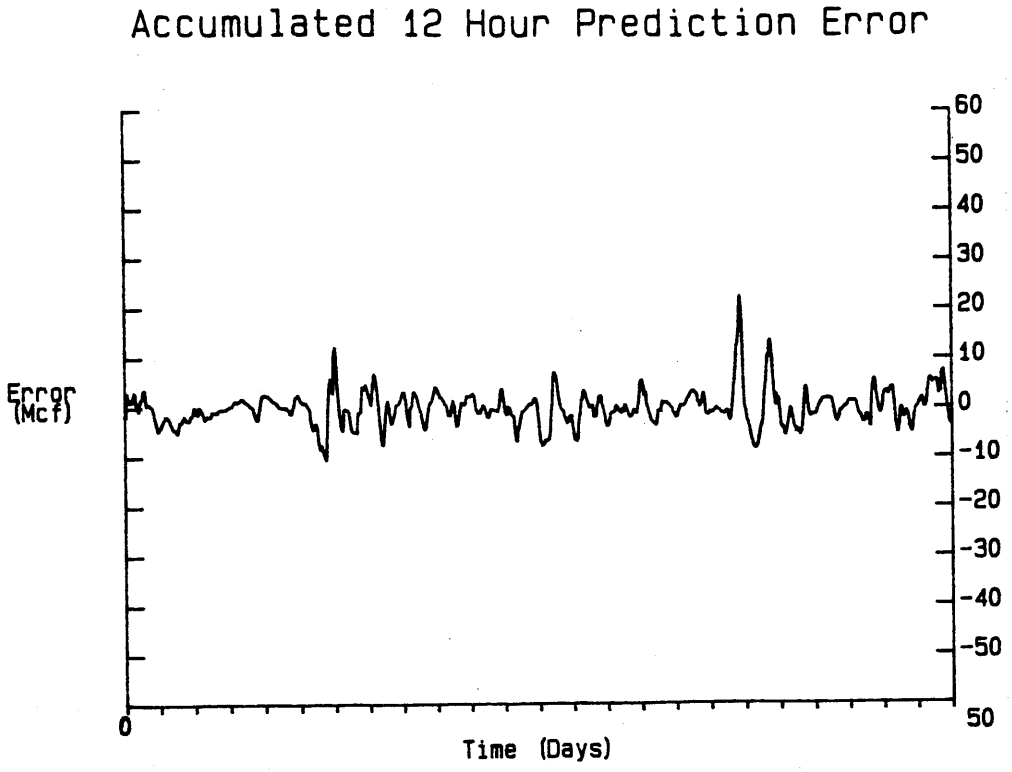


Fig. 5-6: Selective Model Error Accumulated Over 12 Hours

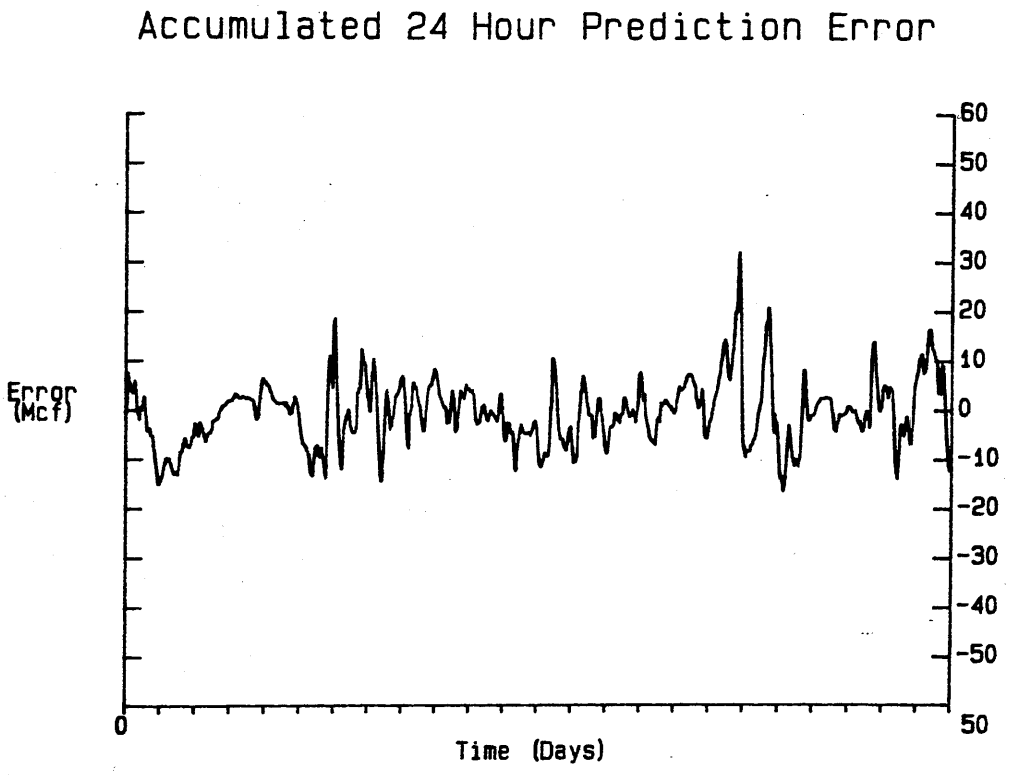


Fig. 5-7: Selective Model Error Accumulated Over 24 Hours

5.6. Results

In our case, with restricted data, the mechanism of the catchment appears to be unsuited to 'expert' or 'automatic' model finding methods. Both the induction of rules in an expert system and the adaption of parameters by self tuning fail when trying to forecast runoff : either because, the data collected from Gaur is too restricted in quantity and quality or, because the catchment itself varies its response too much in time.

The simplified empirical system worked well. The selection mechanism allowing the simple sub models to follow the changes in the state of the catchment. Typically, the errors after 24 hours amount to approximately 1 ft in water level at the dam. Such a margin is better than can be maintained at present.

The system of prediction outlined above need not be regarded as unique to the catchment at Gaur. The method can be applied to any station within the Tummel Valley scheme and may be particularly useful in the case of the run-of-river station at Pitlochry.

6. DECOUPLING THE VALLEY BY OPTIMISING THE DAILY OUTFLOW

6.1. Introduction

Operating constraints in the Tummel Valley scheme arise from the coupling of each reservoir to its neighbour. The magnitude of the flows between the major reservoirs of the valley result in an unmeasureable coupling effect from hour to hour. However, the accumulation of the flows over a day or more produces a highly significant coupling constraint on the operation of the scheme. In this chapter a method will be presented for effectively decoupling the interactions between the reservoirs. Optimum daily outflows are calculated for each station. This takes account of the coupling effects between reservoirs leaving a simpler problem for hourly scheduling (Chapter 7).

6.2. Data Input to the Daily Decoupling Problem

The object of the optimisation is to obtain daily outflows which will take the current system state to some target system state. The system state may be defined by the levels, or storages, of each reservoir and the outflows of each station within the Tummel Valley scheme.

The most important input to the daily decoupling problem is the current system state. Data from each reservoir and station is tele-metered to Errochty Group Control Room. Readings can then be taken from the instrumentation and entered into the optimisation system. Any future system could easily automatically log this data.

The relative benefits of any set of daily outflows are

obtained by comparing the expected system state at the end of the next 24 hour period to some target state. This is only possible if some prediction of the total inflow over the next 24 hour period is available. Chapter 5 describes methods of producing an hour by hour prediction of inflow, which can be integrated to give a 24 hour prediction.

Chapter 4 discusses the long term, or strategic, operating policy for the operation of the reservoirs within the valley. It is from this strategic plan that the daily target state is obtained.

The daily decoupling problem cannot ignore the needs of the national grid for power. The national grid can present its power requirement in the form of an hourly demand curve. The curve must then be translated from an hourly demand to a representation of the daily requirement for power.

6.3. Converting Hourly Power Demands to Daily Power Demands

In a system which uses merit order scheduling, the hourly power demand can be represented by a merit order cost. The most expensive plant used in any hour has the highest merit order cost. It is this cost per unit which is represented in the hourly demand curve, D_k. A typical demand curve is shown in figure 6-1 . At night, the cost per unit is low, as large nuclear stations can easily cope with the load. However, compare this to 16:30 hrs when the tea-time peak might require inefficient small gas turbines to be started for a few hours.

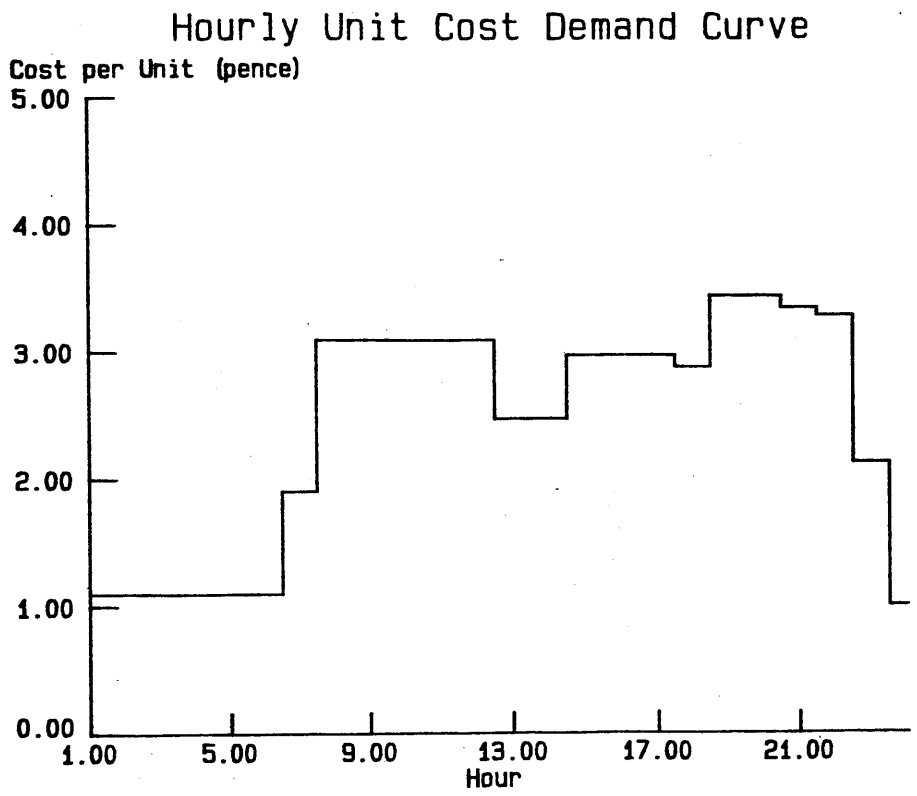


Fig. 6-1: Hourly Demand Curve of Merit Order Costs per unit

The benefits of power generation must, therefore, be calculated based upon the cost per unit of the hour at which the generation is produced. In considering lumped daily outflows the distribution of generation over the day is not known. The daily decoupling problem must, therefore, assume that the hourly scheduling procedure will distribute the daily outflow in such a way as to allocate generation to the highest cost per unit first. For example, if water is available for one hour's generation then it will be allocated to the hour with the highest cost per unit. If water is available for a second hour's generation then it will be allocated to the hour with the second highest cost per unit, and so on. By sorting the hourly demand curve, D_k (Figure 6-1), into descending order, a sorted demand curve, DS_k, is obtained which represents the expected price of each successive hours generation.

This sorted demand curve, DS_k, is shown in Figure 6-2.

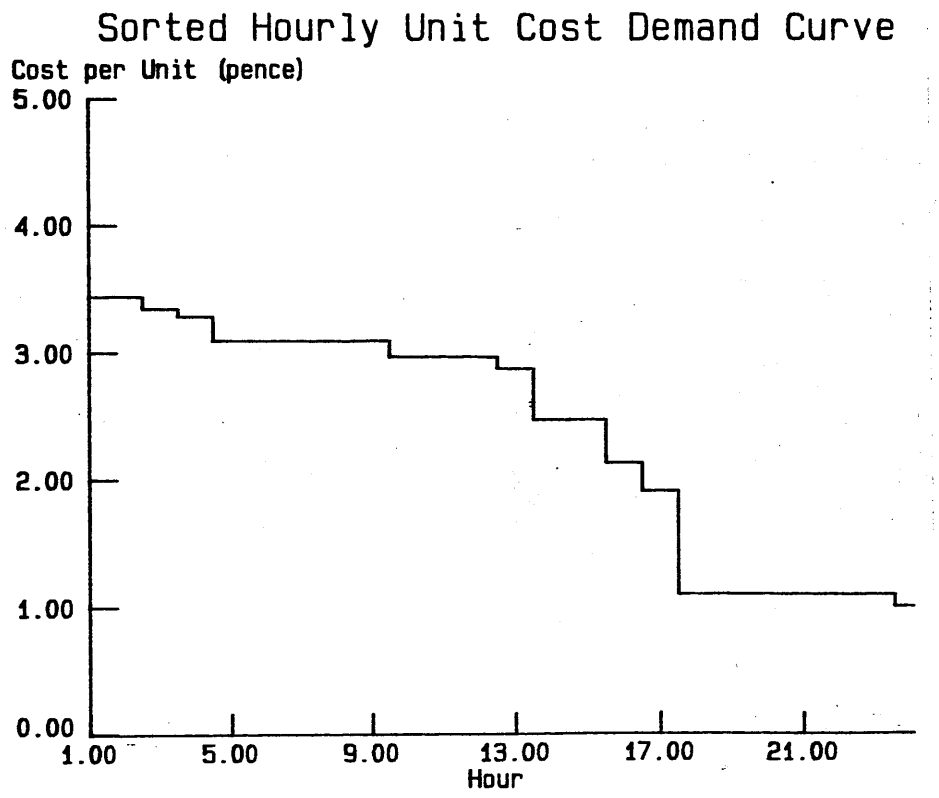


Fig. 6-2: Sorted Demand Curve of Merit Order Costs Per Unit

The daily decoupling problem can then easily assess the potential benefits of daily outflow by integrating the DS_k curve of sorted generation prices. The integration is performed from zero hours to the number of hours of generation that the daily outflow will allow the plant to operate.

6.4. Daily Decoupling Problem Formulation

In the calculation of the optimum daily outflow, although the number of variables is small, the complex coupling of the reservoirs would lead to a high dimensionality if enumeration methods were used. The nature of the benefit function, which for the Tummel Valley scheme is continuous and differentiable, makes the problem ideal for search methods. Thus, all the work reported here has used non-linear programming.

The aim of the optimisation is to maximise the benefits from the system. Benefits are assessed by summing various returns and penalties. The generation of power brings positive returns. Other penalties must be introduced to represent losses due to spillage, and the more abstract cost of failing to meet a target storage. For this study, the cost is expressed as,

$$\text{Benefit} = \sum_{j=1}^{8} BP_j - PL_j - PX_j \qquad \text{—Eq. 6.4.1}$$

where BP_j is the financial return from the power generated by the jth station,

PL_j is the penalty incurred by the jth reservoir in failing to meet its target storage, and,

PX_j is the penalty attached to the current storage level which penalises spillage and forces the reservoir not to run empty.

The limitations placed upon the instantaneous outflow of a turbine (Eq. 3.1.2) do not apply to daily outflow. The set of possible instantaneous outflows combine to give a continuous region for the daily outflow. The limitations placed on the daily outflow are simply,

$$0.0 < od_j < \text{Maximum Daily Flow}$$
$$od_j \text{ must be such that } x_j > 0.0 \qquad \text{—Eq. 6.4.2}$$

6.4.1. Derivation Of Returns and Penalties

In all optimisation methods costings are relative. The costing value, therefore, need not be defined in absolute terms

but merely in relation to other costs. However, to be useful, any results must be understandable by an operator. It is important, especially during the initial testing of an optimisation system, that the operator can follow the method with which the optimisation system produces its results and compare this to his own decision processes. The most important cost in any power system is the cost of one Unit of generation. Here, the costing functions are all related to the price of the units generated.

6.4.2. Return from Power Generated

It has been shown in section 6.3. that the hourly demand price curve can be represented by a sorted curve to give a daily demand curve. The benefit from the units generated may then be calculated by,

$$BP_j = Nu_j * \sum_{k=1}^{Nh_j} DS_k \qquad \text{—Eq. 6.4.5}$$

where Nh_j is the number of hours that the jth station will generate for during that day,

Nu_j is the number of unit that the jth station can generate in one hour from one set, and,

DS_k is the sorted demand price curve.

The number of hours of generation from a station is given by,

$$Nh_j = \frac{od_j}{Fh_j * Ns_j} \qquad \text{—Eq. 6.4.3}$$

where Fh_j is the hourly flow through one set (at optimum power) for station j,

Ns_j is the number of sets available in station j, and,

od_j is the daily flow through the jth station.

The number of units produced by one set of a station in one hour is given by,

$$Nu_j = \frac{Fh_j}{WCF_j} = Popt_j * 1000 \qquad \text{—Eq. 6.4.4}$$

where Nu_j is the number of units produced by one set,

WCF_j is the water conversion factor, and,

$Popt_j$ is the optimum power of the jth station in Mw.

Table 2-4 tabulates the values of $Popt_j$ for each of the stations of the scheme.

6.4.3. Reservoir Storage Penalties : Spillage or Empty

If a reservoir level falls below its minimum operating level then that reservoir may be said to be empty. It is possible that a reservoir may become empty at some point in its operation due to some unforeseen circumstances. This is represented by a fixed penalty in the costing function which results in an effective constraint on the feasible region of the solution.

Spillage is water which is lost. It could have produced power. Hence, the system is penalised by the value of the lost power. If the power had not been lost then it is feasible that the power could have been produced at one of the peak times in the demand. The lost units are, therefore, valued at the maximum possible demand rate, DS_1. The spillage may be calculated using Equation 3.1.7 .

The storage penalties are defined as,

$$PX_j = sp_j * SCF_j * DS_1 \qquad \text{If } x_k > \{x_j\}_{max}$$

$$PX_j = c_e \qquad \text{If } x_j < 0.0$$

—Eq. 6.4.6

where SCF_j is defined as the spill conversion factor which calculates the number of units lost due to spill,

sp_j is the spillage calculated from equation 3.1.7,

DS_1 is the maximum value of the demand price curve, and,

c_e is fixed penalty which constrains reservoir not to empty. A large value of c_e was used to force a non-empty solution.

The spillage from different reservoirs has varying penalties because the spillage takes different routes by-passing the station.

The water which spills from Loch Seilich results in a loss of power from Cuaich, Rannoch, Tummel, Clunie and Pitlochry stations because the water flows out of the valley. The cost of the spill is defined as,

$$SCF_1 = \frac{1}{WCF_1} + \frac{1}{WCF_3} + \frac{1}{WCF_5} + \frac{1}{WCF_7} + \frac{1}{WCF_8} \qquad \text{—Eq. 6.4.7}$$

This is the most expensive water lost.

The water lost from Loch Garry flows into Loch Errochty. The power lost by missing Loch Ericht, Rannoch and Tummel stations is offset by the power gained by the water being available for Errochty station. Thus,

$$SCF_2 = \frac{1}{WCF_2} + \frac{1}{WCF_3} + \frac{1}{WCF_5} - \frac{1}{WCF_6}$$

The spillage for Loch Ericht, Loch Eigheach, Loch Rannoch, Loch Tummel and Loch Faskally simply by passes one station and is defined as,

$$SCF_j = \frac{1}{WCF_j} \qquad \text{for } j = 3,4,5,7 \qquad \text{—Eq. 6.4.9}$$

Due to the physical constraints placed upon spillage from Loch Ericht (see chapter 2) the penalty is modified to,

$$SCF_3 = 4 * \frac{1}{WCF_3} \qquad \text{—Eq. 6.4.10}$$

Water lost at the intake from the River Garry to Loch Errochty is considered to be spilled from Loch Errochty and flows down the River Garry to Loch Faskally. Power is, therefore, lost from Errochty and Clunie stations. This gives,

$$SCF_5 = \frac{1}{WCF_5} + \frac{1}{WCF_7} \qquad \text{—Eq. 6.4.11}$$

Table 6-1 lists the derived cost of spilling.

6.4.4. Penalties Associated with Failing to Achieve a Target Storage

The penalty associated with not achieving a target level must be related to two main factors : the current demand for power and the ability of the reservoir to recover to its long term strategy. In representing these considerations a level penalty may be defined as,

$$PL_j = cx_j * (Tx_j - x_j) \qquad \text{If } Tx_j > x_j$$

$$PL_j = cx_j * (x_j - Tx_j) \qquad \text{If } Tx_j < x_j$$

—Eq. 6.4.12

where cx_j is defined as the penalty of failing to achieve the target storage by one Mcf, and,

Tx is the target storage.

The penalty cx_j must be such that daily decoupling procedure is not penalised for using more water at times of high demand. An initial value of cx_j can be taken from the demand cost curve, and is defined as,

$$cx_j = \frac{500 * DS_1}{WCF_j} \qquad \text{—Eq. 6.4.13}$$

The penalties for failing to achieve a target storage must be greater for the smaller reservoirs, as larger reservoirs are able to compensate for deviations from the target storages in later periods. The value was modified to compensate for the reservoir size in the initial laboratory tests, and later during site trials at Errochty Group Control Room. The final value of cx_j is listed in table 6-1.

Station	SCF (kUnits/Mcf)	cx (kUnits/Mcf)
Cuaich	18.6	5.0
Loch Ericht	4.5	10.0
Rannoch	38.0	0.8
Gaur	1.8	5.0
Tummel	3.3	10.0
Errochty	14.2	0.4
Clunie	3.2	10.0
Pitlochry	0.9	10.0

Table 6-1: Tabulated Value of the Constants within the costing function

6.5. Solution Method of the Daily Decoupling Problem

The method of steepest descent [26] was used to optimise the system. The method is simple and fast if a good initial solution is chosen[26]. The previous day forms a good initial solution since the variations from day to day under normal operating conditions are small. The method of steepest descent uses the gradient of the costing function to adjust the solution in order to get the maximum improvement in the costing function. These gradients were derived analytically from the cost function. The form of the penalty functions are such that the costing function only has one maximum, therefore the search will not stop at a false peak. The partial derivatives of the benefit function with respect to the outflow solution were used to specify the size and direction of the change or step to be made to the outflow solution. The step size was reduced as the optimisation method

approached the maximum of the benefit function. The size of the step was used as the criterion for terminating the optimisation. The constraints discussed earlier (Eq. 6.4.2) were implemented by explicitly restricting the values of the solution outflows to lie within the feasible region. The algorithm executed in under 1 minute on an IBM PC AT with an 80287 maths coprocessor fitted. This normally involved only up sixteen interations in the search space before terminating at an optimum.

6.6. Results of the Daily Decoupling Procedure

The testing of the program was done in two stages. In the laboratory, past records of daily operation were used to find values for each of the constants in the costing function. The situations represented by the daily operational reports are not completely realistic as target storages were not available. The second stage of testing was to use the program at Errochty Group Control Room on situations set up in consultation with the generation engineers. The penalties associated with failing to achieve a target storage were modified in view of these discussions with the generation engineers.

The object of the tests run on the program were twofold : firstly, to test the program's response to normal operation and its response to the demand curve, and, secondly, to ensure that the program takes account of the daily interactions of the scheme. A number of realistic situations were set as problems for the program in consultation with the Generation Engineers at Errochty Control Centre.

6.6.1. Solution 1 - Normal Operation

The period chosen to illustrate the normal operation of the Tummel Valley scheme was late spring/early summer. During this period the main reservoirs of the scheme would be near-full from winter runoff and snow melt. The runoff would be dropping off towards its summer mean. The releases from the reservoirs would be carefully controlled to ensure that water was available to maintain flows during the summer months. The demand curve of Figure 6-1 was used. The complete situation is described by the data given in Table 6-2.

Station	Initial Storage (Mcf)	Target Storage (Mcf)	Daily Inflow (Mcf)	NSHEB Solution Outflow (Mcf)	Program Solution Outflow (Mcf)
Cuaich	50.04	45.00	10.00	16.0	14.8
Loch Ericht	99.96	110.98	20.00	0.0	8.0
Rannoch	4499.59	4488.28	50.00	44.0	65.4
Gaur	75.00	75.00	45.00	45.8	44.6
Tummel	818.48	818.48	25.00	132.6	135.6
Errochty	699.91	640.00	35.00	40.9	89.0
Clunie	189.21	189.21	12.00	189.7	238.6
Pitlochry	27.60	27.60	70.00	259.7	310.0

Table 6-2: Daily Decoupling Problem : Results for Solution 1 - Normal Operation

The major differences between the NSHEB solution and the computer optimisation are in the outflows of two stations, Rannoch and Errochty. The reduced flow at Rannoch is reflected at all downstream stations. The difference at Rannoch is caused by the Generation Engineers using the discrete table of planning periods

(Table 2-6). A set is on or off for whole blocks of time within each period, whereas the computer program operates on a continuous range of outflows. The Generation Engineers plan the operation of Errochty based upon the operation which was required from the station by Central Control Room in Pitlochry the previous day. In this hypothetical situation the previous day's operation is not known, thus the operation of Errochty is simply an estimate of the normal requirements for the time of year.

6.6.2. Solution 2 - Response to Demand

The situation which was used to test the response to an increased demand was the same as solution 1. An increased demand curve, as shown in Figure 6-3, was used in the optimisation.

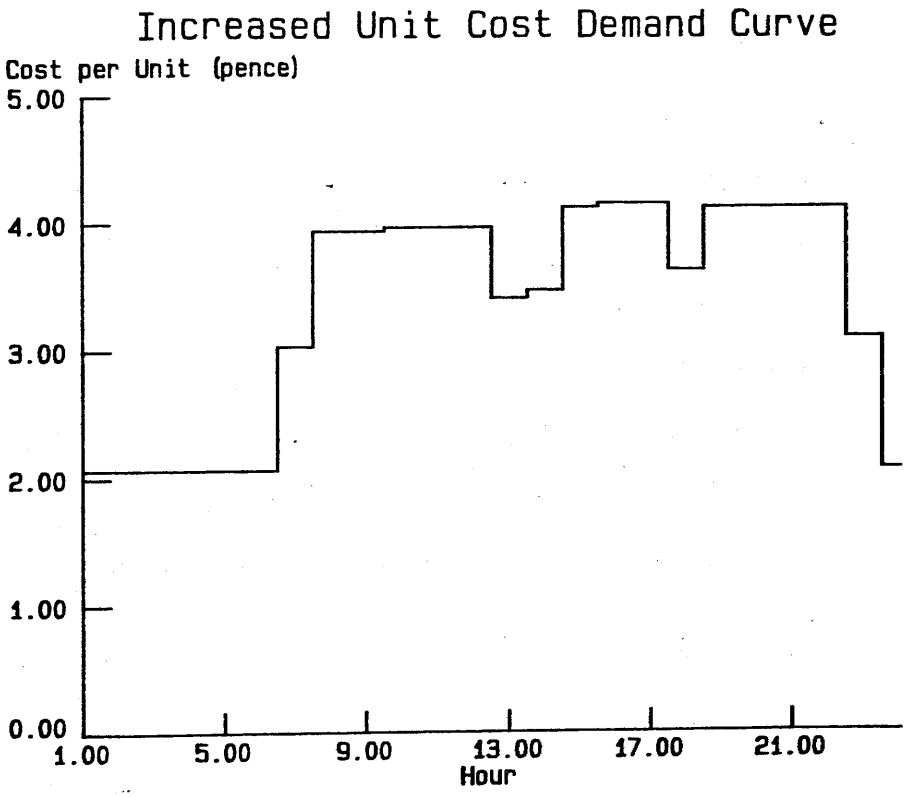


Fig. 6-3: Increased Hourly Demand Curve of Merit Order Costs per unit

The results of the optimisation are listed in Table 6-3.

Station	Initial Storage (Mcf)	Target Storage (Mcf)	Daily Inflow (Mcf)	NSHEB Solution Outflow (Mcf)	Program Solution Outflow (Mcf)
Cuaich	50.04	45.00	10.00	16.0	15.4
Loch Ericht	99.96	110.98	20.00	10.6	9.1
Rannoch	4499.59	4488.28	50.00	70.5	75.6
Gaur	75.00	75.00	45.00	56.3	45.0
Tummel	818.48	818.48	25.00	163.2	146.3
Errochty	699.91	640.00	35.00	109.2	101.4
Clunie	189.21	189.21	12.00	290.6	260.3
Pitlochry	27.60	27.60	70.00	360.6	330.8

Table 6-3: Daily Decoupling Problem : Results for Solution 2 - Response to Demand

The Generation Engineers, to meet an increased demand, would increase the outflow of firstly, Errochty, and then Rannoch, as the storages behind these stations can most easily recover to the target storages. The excess water would be taken out of the valley, increasing the outflow from most of the other stations. The computer optimisation slightly increases the output from all stations, in addition to increases in Rannoch and Errochty.

6.6.3. Solution 3 - Spillage at Rannoch

At the end of winter, the rainfall combines with snow melt to give large runoff in the high reservoirs. A situation was set up in which Loch An-T-Seilich and Loch Garry would spill unless Cuaich and Loch Ericht stations operated for 24 hours. At the same time, Loch Ericht was set up so that the natural inflow, combined with the outflow from Cuaich and Loch Ericht Stations,

would cause spillage at Loch Ericht dam. The other reservoirs of the scheme were set up so as not to interact with the complex Ericht situation. The data for this problem is shown in Table 6-4.

Station	Initial Storage (Mcf)	Target Storage (Mcf)	Daily Inflow (Mcf)	NSHEB Solution Outflow (Mcf)	Program Solution Outflow (Mcf)
Cuaich	94.98	95.0	35.01	0.0	16.02
Loch Ericht	299.04	299.00	20.01	0.0	0.00
Rannoch	5709.28	5709.28	90.00	105.8	105.84
Gaur	75.00	75.00	40.00	45.8	39.88
Tummel	613.86	613.86	30.00	163.2	175.95
Errochty	243.70	243.70	77.00	54.6	77.00
Clunie	126.14	126.14	0.00	233.7	252.84
Pitlochry	34.50	34.50	70.00	303.5	322.84

Table 6-4: Daily Decoupling Problem : Results for Solution 3 - Spillage at Rannoch

In both solutions to this problem Rannoch station generates for 24 hours. Since the spillage from Loch Garry flows to Loch Errochty and is more acceptable than spillage for Loch Ericht, Loch Ericht station remains completely off. The Generation Engineers would keep Cuaich station completely off due to the uncertainty of the inflow to Loch Ericht and their desire to reduce the storage in Loch Ericht for the inflow of the following day. The computer optimisation, however, computes that it is possible to reduce spillage from Loch An-T-Seilich by passing some water through Cuaich station and still not spill at Loch Ericht.

6.6.4. Solution 4 - Spillage at Clunie and Tummel

During the sailing season, Loch Rannoch and Loch Tummel are held high to facilitate the launch of boats. Thus, at the end of summer, a sudden heavy rain storm could cause spillage at these reservoirs. The other reservoirs of the scheme would be nearly empty and running on reduced flows. This is the situation depicted by the data of Table 6-5.

Station	Initial Storage (Mcf)	Target Storage (Mcf)	Daily Inflow (Mcf)	NSHEB Solution Outflow (Mcf)	Program Solution Outflow (Mcf)
Cuaich	20.04	20.00	5.00	0.0	5.0
Loch Ericht	29.96	30.00	5.00	0.0	5.1
Rannoch	1198.38	1198.90	100.00	0.0	60.9
Gaur	45.00	45.00	90.00	45.8	43.7
Tummel	1435.00	1435.00	100.00	163.2	207.1
Errochty	400.00	399.91	80.00	54.6	61.9
Clunie	320.00	319.76	40.00	303.5	313.7
Pitlochry	27.60	27.60	80.00	383.5	395.1

Table 6-5: Daily Decoupling Problem : Results for Solution 4 - Spillage at Clunie and Tummel

The NSHEB solution in this case is close to the computer optimisation solution. Gaur, Rannoch and Errochty stations are off completely, whereas the computer attempts to remove some flow in order to achieve target storages. Tummel, Clunie and Pitlochry are generating throughout the day. Spillage is avoided at all stations.

6.6.5. Solution 5 - Empty at Clunie

At the end of an exceptionally dry summer, the storages of Loch Rannoch and Loch Tummel may have to be totally depleted in order to maintain the minimum statutory flows of the valley. A reserve for this sort of situation is normally held in Loch Ericht. The program must maintain the minimum flows within the valley. The data and results of the situation are shown in Table 6-6.

Station	Initial Storage (Mcf)	Target Storage (Mcf)	Daily Inflow (Mcf)	NSHEB Solution Outflow (Mcf)	Program Solution Outflow (Mcf)
Cuaich	20.04	20.00	5.00	0.0	4.8
Loch Ericht	29.96	30.00	5.00	0.0	5.1
Rannoch	499.70	500.00	5.00	70.5	20.5
Gaur	10.05	10.00	8.00	14.1	16.8
Tummel	0.00	0.00	4.00	71.2	40.2
Errochty	9.99	10.00	5.00	0.0	14.7
Clunie	0.00	0.00	2.00	50.6	56.4
Pitlochry	4.97	5.00	20.00	65.0	79.2

Table 6-6: Daily Decoupling Problem : Results for Solution 5 - Empty at Clunie

The results show that the optimisation does in fact use the reserve of Loch Ericht, augmented by small additional flows from other stations to maintain the minimum flows within the valley. Although not in complete agreement with the Generation Engineers, the optimisation solution was considered by the engineers to be a feasible one.

6.6.6. Discussion and Conclusion

It can be seen from the results of section 6.6. that the daily decoupling procedure does effectively decouple the reservoirs of the valley. The results are, in general, in agreement with the suggested operation of the Generation Engineers of Errochty Group Control room. The engineers, however, stated three main criticisms of the computer solution. The computer assumed that the inflow data, which it was given, was correct and thus made no allowance for inflow prediction error in its solution. In the operation of Loch Ericht station, the optimisation tended to ignore the unreliability of the station by starting and stopping the turbine too often. Finally, the optimisation specified outflows which would result in stations generating partly through the night in order to meet target storages. In general, the output from stations is not changed during the night as this causes problems in meeting the normally steady, low night demand.

The decoupled daily outflows may then be allocated to an hourly schedule by the method presented in the next chapter.

7. HOURLY ALLOCATION OF DAILY WATER RESOURCES

7.1. Introduction

The available daily outflow, calculated by the methods described in the previous chapter, must be allocated in hourly blocks to the sets of each station. This chapter describes the methods and assumptions used to schedule the flows according to the hourly demand price curve described in section 6.3.

The scheduling ignores the backward coupling between the reservoirs. It is assumed that the daily decoupling procedure accounts for this type of interaction. The exception is the interaction between Clunie and Pitlochry stations. The area of Loch Faskally is very small in comparison to the outflow from Clunie station. This, combined with the operational constraints which are placed upon the operation of Pitlochry station, make the hourly scheduling of Clunie and Pitlochry a highly coupled problem which can not be simplified.

7.2. Aims of the Hourly Scheduling

The purpose of the scheduling routine is to allocate the water resources which the daily decoupling program has decided to make available. The water is used to produce power at the most financially beneficial time, which is decided by the hourly demand price curve. The costing function which is used in the optimisation is similar to the one used for the daily interface, and for the jth station may be defined as,

$$\text{Benefit} = BP_j - PX_j - PC_j. \qquad \text{--Eq. 7.2.1}$$

where BP_j is the financial benefit received for the power generated by the jth station,

PX_j is the penalty associated with the reservoir storage, and,

PC_j is the potential benefit of the power which could have been produced by the water lost by changing the state of the set of the jth station.

The system is constrained by the model given for the reservoirs in Equation 3.1.3. Faster computation can be achieved by considering that if a set is to be on for a period then it will operate only at its optimum power. The assumption is valid because at present all sets within the valley use the load limiter of their governor to hold their output constant. The flows between reservoirs can, therefore, be calculated exactly without reference to frequency via the droop of the governor. The three sets of Errochty station are allowed to vary their output with frequency, but are closely monitored by the Generation Engineers at the Errochty Group Control Centre.

7.2.1. Benefits of the Units Generated

The financial return for the units generated within the scheme is dependent upon the time of day at which the units are produced. The series D_k, k=1..24, represents the demand price curve over the day. The number of units produced by one set of a station operating at optimum power for one hour is given by,

$$Nu_j = Popt_j * 1000 \qquad \text{If set is on}$$

$$Nu_j = 0 \qquad \text{If set is off}$$

—Eq. 7.2.2

where Nu_j is the number of unit produced by one set, and,

$Popt_j$ is the optimum power of the jth station in Mw.

Table 2-4 lists the values of $Popt_j$ for each station.

If $\{D_k\}_{max}$ is taken to be the maximum return for one unit, then BP_j may be defined as,

$$BP_j = \sum_{k=1}^{24} D_k * Nu_j * \{D_k\}_{max} \qquad \text{—Eq. 7.2.3}$$

where D_k is the hourly demand price curve.

7.2.2. Penalties on Reservoir Storage

The spillage costing is identical to that used in the previous chapter (Section 5.4.3).

7.2.3. Penalties Incurred by Changing the State of a Station

Typically, it takes 5-10 minutes to synchronise and load a set in the Tummel Valley scheme. During this time, water is being used to generate power at a less than maximum efficiency. It is the benefit from the extra units, which could have been generated if the same water was used at maximum efficiency, that is used as the penalty for starting, or stopping, a turbine.

Within the Tummel Valley scheme, the number of units lost in this way is of the order of 120-800 units for each set started, or

stopped. For some stations in the scheme, the number of units lost have been adjusted to reflect some special operating constraint. The units lost by starting, or stopping, the set at Loch Ericht station has been artificially increased to reflect the unreliability of this station. Similarly the number of units lost by starting, or stopping, the sets of Errochty station has been reduced to reflect the peak lopping nature of this station's operation. The penalty associated with changing the state of a station is given by,

$$PC_j = \sum_{k=1}^{24} c_c * \Delta C_k * \{D_k\}_{max} \qquad \text{—Eq. 7.2.4}$$

where c_c is the number of the units lost when starting, or stopping,

ΔC is the change in the number of sets in operation, and,

$\{D_k\}_{max}$ is the maximum cost per unit.

The number of units lost, c_c is tabulated in Table 7-1.

Station	Units lost
Cuaich	112.5
Loch Ericht	440.0
Rannoch	700.0
Gaur	262.5
Tummel	800.0
Errochty	100.0
Clunie	800.0
Pitlochry	---

Table 7-1: The number of units lost by starting, or stopping one set for each station.

This penalty dictates the response of the station to the hourly demand curve. If c_c is low then the station will start and stop sets often in order to maximise the benefit of the generation. If, however, c_c is large then optimisation will try to maximise the benefit from blocks of power.

7.3. Method of solution

The method, which is used to optimise the penalty function of equation 7.2.1, uses dynamic programming[40] to find the optimal operational policy for each station. The problem is simplified by the application of some simple rules. Each set when operating must operate at its optimal power. The station must only release a given amount of water in a day. The amount is determined by the daily decoupling procedure described in chapter 6.

An illustration of the method is best done by considering the case of one reservoir and one station, with two sets. It is possible to construct a two dimensional matrix of the possible storages throughout each period of the day. At the start of each period there are only three possible actions which can be taken : both turbines off, one turbine on, or both turbines on. By selecting each of these possible actions in turn it is possible to cost each possible path through the matrix of storages. This is illustrated in Figure 7-1. The optimal solution is then found by backtracking from the final action to the initial starting point. The solution time is obviously dependent upon the size of the matrix which has to be searched. The maximum time taken to find a solution was approximately one minute on an IBM PC-AT with 80287 maths coprocessor.

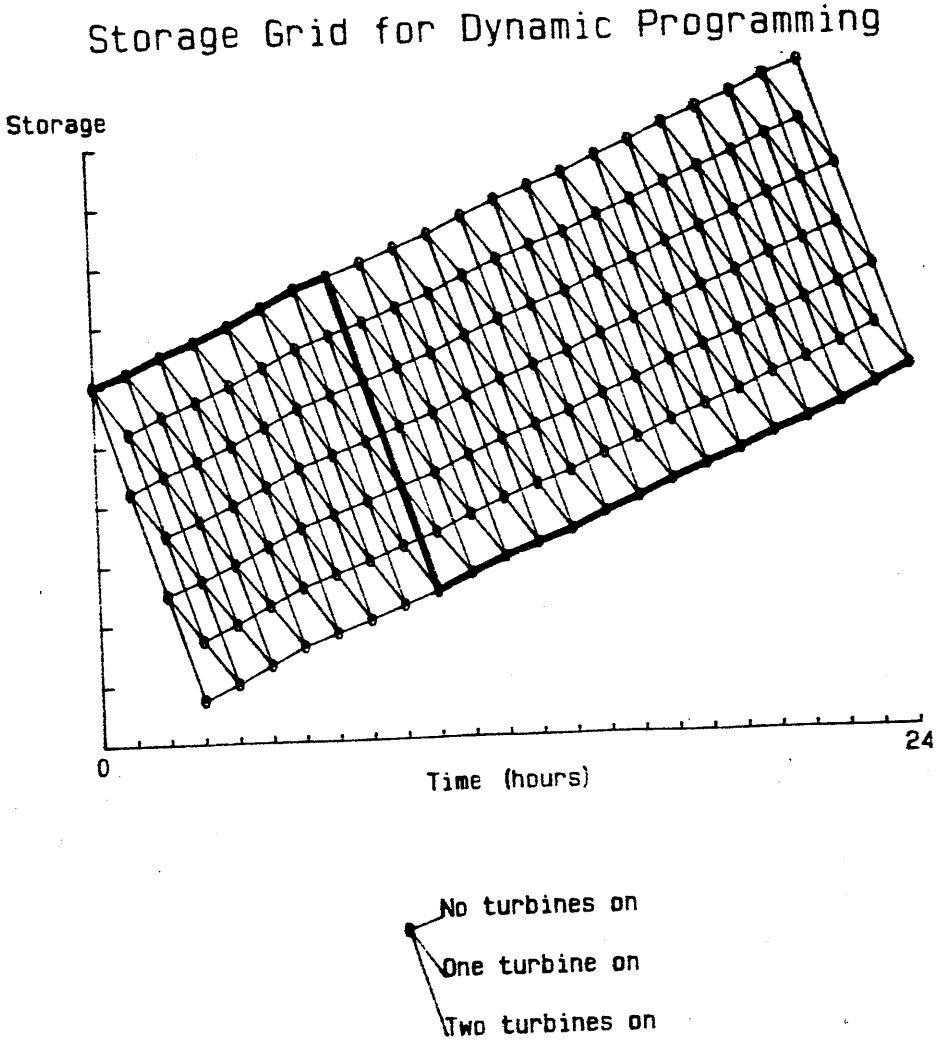


Fig. 7-1: Optimal operation of one station with two sets, using Dynamic Programming.

7.4. Clunie - Pitlochry Interaction

The purpose of Pitlochry station, the last station of the scheme, is to smooth any irregularities in the flow of the River Tummel caused by the outflow from Clunie station. Loch Faskally is a small storage which forces Pitlochry to operate as a run-of-river station. The amount of possible flow into and out of Loch Faskally via Clunie and Pitlochry stations couple the two station operation from hour to hour. The constraint placed upon the operation of Pitlochry, namely the it is only allowed to make two

load changes in 24 hours, further complicates operational planning for the stations.

Each of the two stations cannot be scheduled in isolation from the other, but to schedule them together is impossible due to the dimensionality of the problem. The solution is to use an iterative method to consider each station in turn.

7.4.1. Successive Approximations

The method of iteratively considering each part of a coupled problem as an isolated problem is known as successive approximations[46]. Clunie station is scheduled in isolation. Pitlochry station is then scheduled using the known outflow from Clunie. The two stations are then scheduled iteratively until a stable solution is found for both.

The scheduling of Clunie station when Pitlochry station has been planned is done simply by extending the cost function of equation 7.2.1 to also consider the storage of Loch Faskally. The benefit function therefore becomes,

$$\text{Benefit} = BP_7 + PX_7 + PX_8 + PC_7. \qquad \text{—Eq. 7.4.1}$$

The variables are all defined as previously in section 7.2.

7.4.2. Pitlochry Scheduling

Pitlochry station is constrained to make only two changes in load per day. At 6:00 am the station changes from its night load to its day load, and at midnight it changes from its day load to its night load. It is only possible to deviate from these

statutory constraints if some unforeseen circumstance arises, such as, a sudden change in Loch Faskally inflow which will result in the reservoir spilling or emptying if the load is not changed. The limitation on the number of load changes does not simplify the problem.

Three possible situations may arise depending at what time of day the scheduling routine is called. These are :

(i) Start time is 6:00 or midnight, giving a full day to be scheduled.

(ii) Start time is between 6:00 and midnight, thus day load must be held until midnight.

(iii) Start time is between midnight and 6:00, thus night load must be held until 6:00.

In each of the three cases, only two loads have to be found : Day load and night load. The dynamic programming formulation described previously may still be used to solve the problem. The cost functions must be calculated by simulating the response of the reservoir between possible load change times.

In the case of situations (ii) and (iii) arising the reservoir must be simulated for its fixed load. The simulation will indicate if a situation will arise which will allow the load to be changed. Given that a situation has occurred then the program must then calculate the minimum change in load which will correct for the exception.

7.5. Evaluation of the Hourly Scheduling Results

The results of the hourly scheduling procedure were analysed in three separate areas of performance. The general scheduling

was compared to three actual days of operation of the Tummel Valley scheme. The performance of the Pitlochry-Clunie procedure was examined under conditions of near spillage from Loch Tummel and Loch Faskally. Finally, the response of the system to crisis situations was evaluated.

7.5.1. Comparison with Actual Operation

Three days of actual operation were chosen from different times of the year. The actual daily outflows and a realistic demand curve were given to the scheduler. The results from the scheduler were then compared to the actual operation that was implemented by the Generation Engineers.

7.5.1.1. Daily Operation for Monday 2nd of July 1984

The operation of Monday 2nd of July was that typical of a dry day in summer. There was minimal natural runoff from the catchments within the valley and only the minimum flows are being passed down the Valley. The actual daily outflows used on this day are given in Table 7-2.

Station	Daily Outflow
Cuaich	6.0
Loch Ericht	0.0
Rannoch	41.0
Gaur	0.0
Tummel	55.0
Errochty	17.0
Clunie	72.0
Pitlochry	67.0

Table 7-2: Daily Outflow for the 2nd of July 1984

Figure 7-2, on the next page, shows the actual schedule used by the Generation Engineers on this day. The demand curve and the total generation for the group are shown on page 105, in Figures 7-3 and 7-4 respectively. The results obtained from the optimising procedure are shown on page 106. Figure 7-5 shows the hourly schedule and Figure 7-6 shows the total generation from the group for the optimiser results.

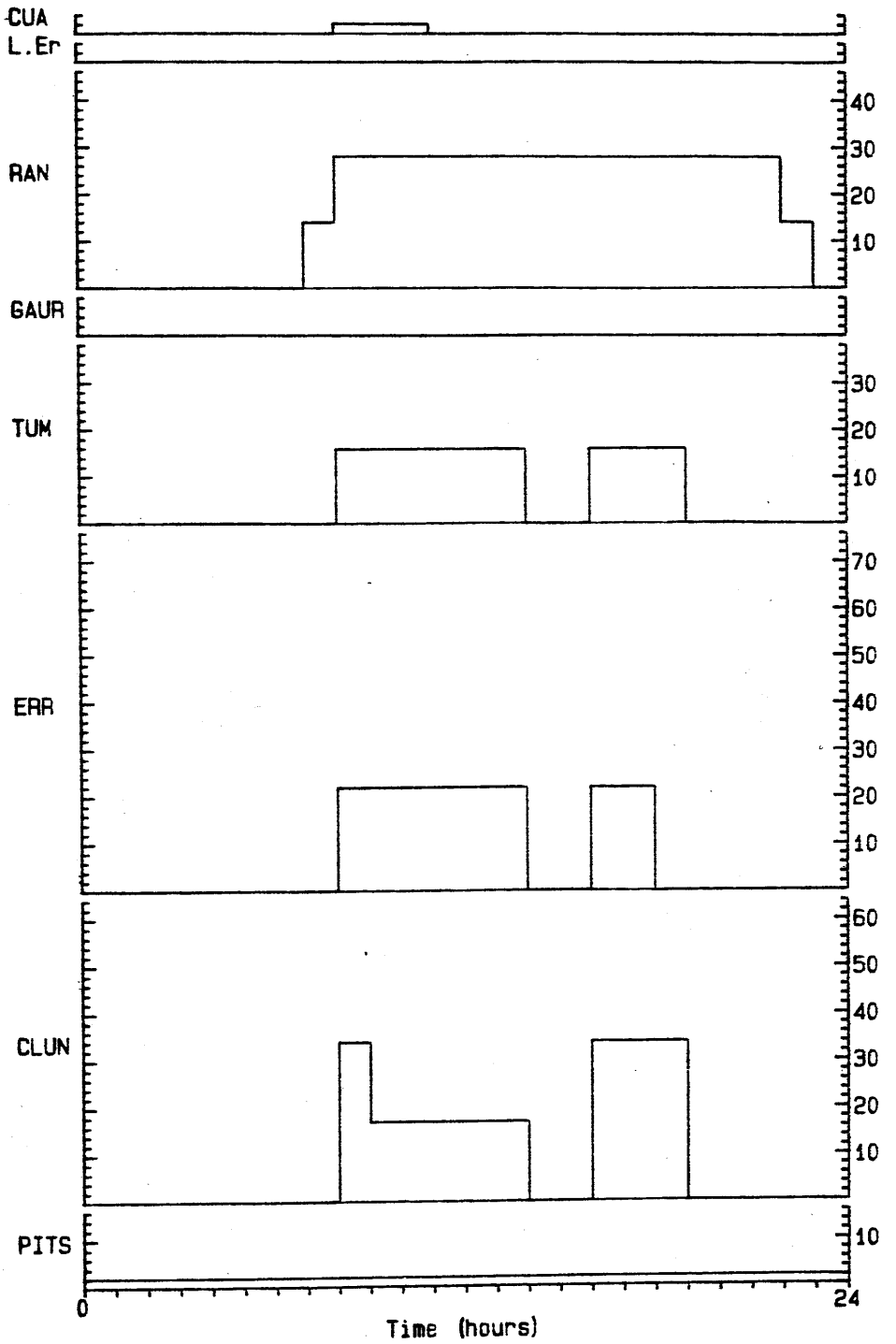


Fig. 7-2: The actual power schedule for the Tummel Valley, as used on 2nd of July 1984.

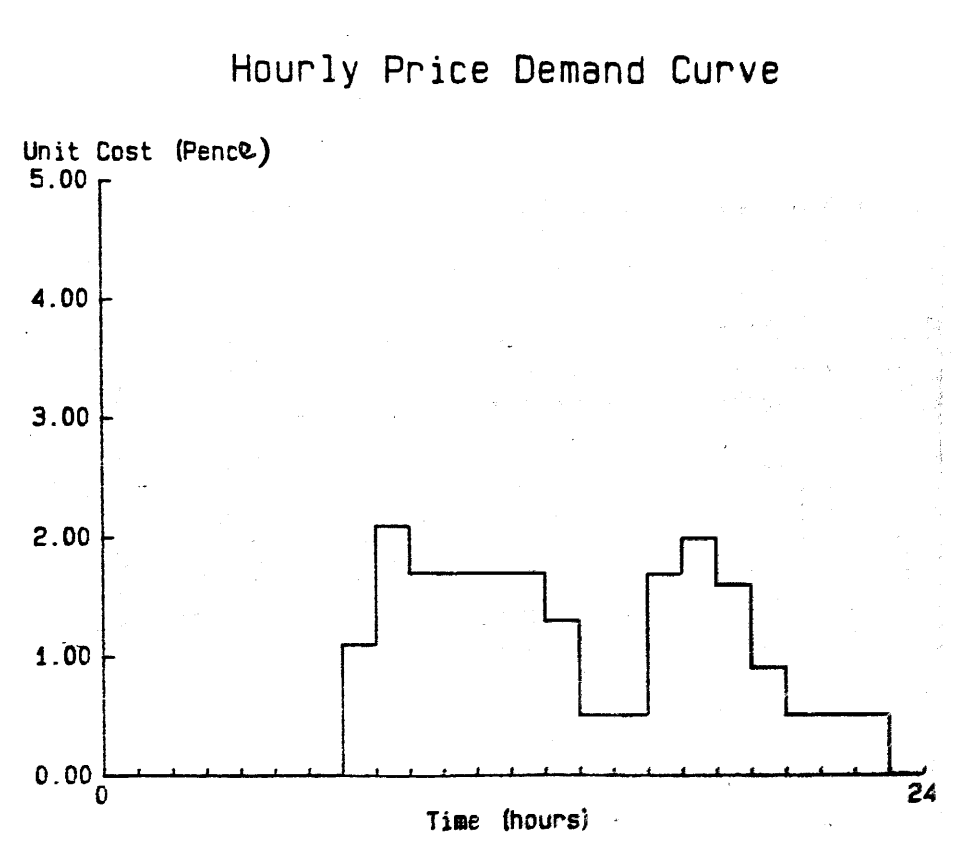


Fig. 7-3: Demand Curve used for the 2nd July 1984

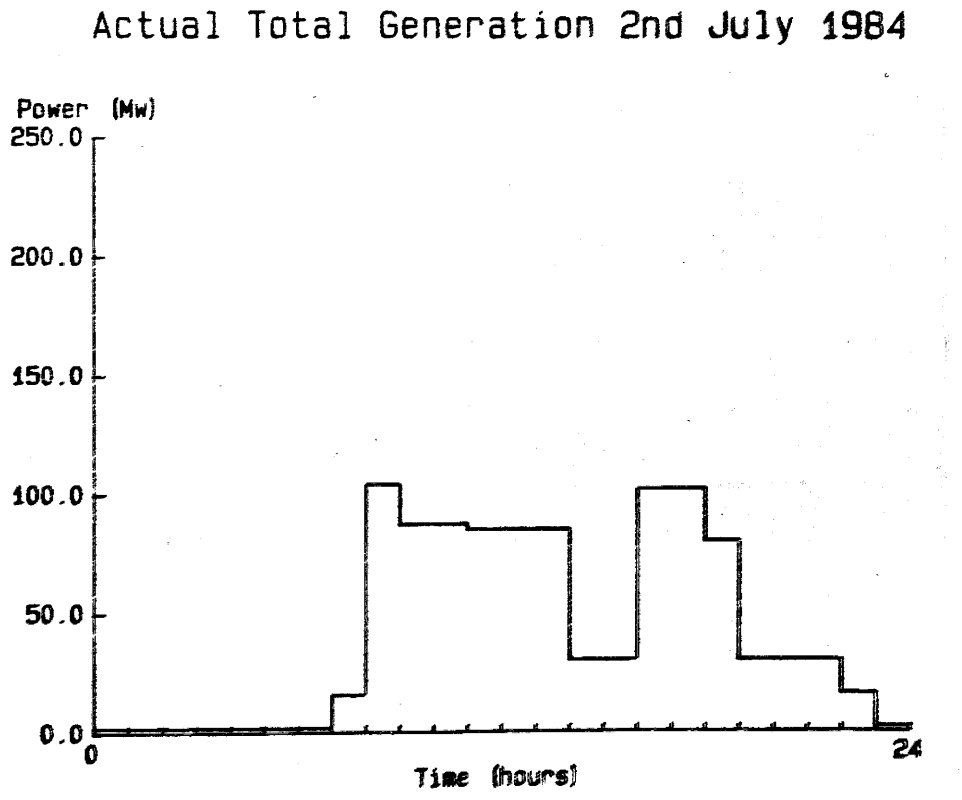


Fig. 7-4: Total Actual Group Generation on the 2nd July 1984.

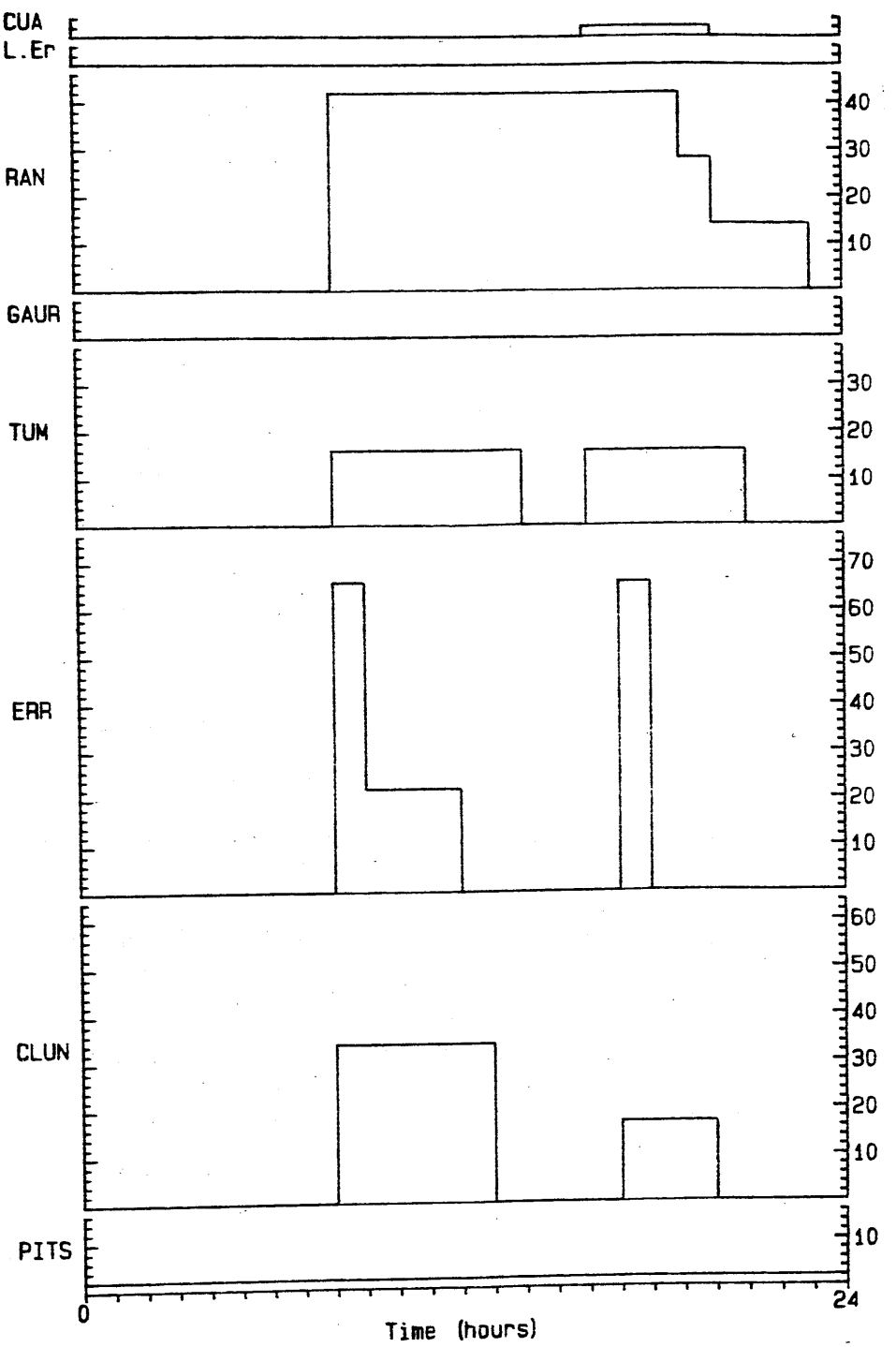


Fig. 7-5: Schedule produced by Computer Optimisation for 2nd July 1984.

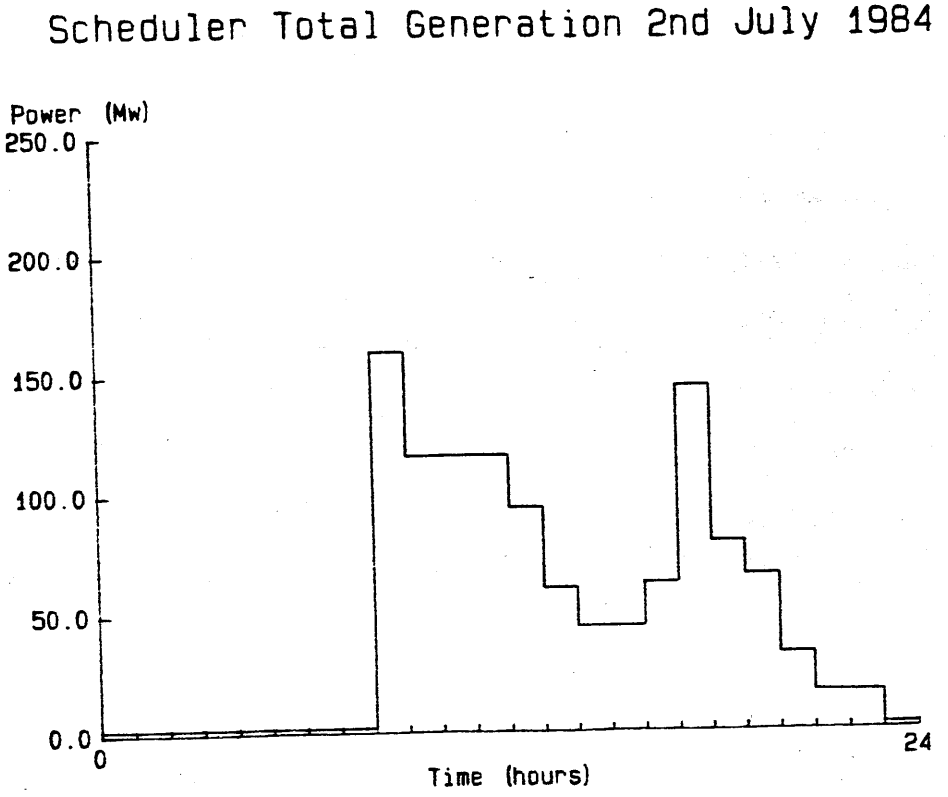


Fig. 7-6: Total Group Power for Optimiser Results for 2nd July 1984 .

By comparing the results shown in Figure 7-2 and 7-5, it can be seen that the schedule produced by the optimisation is very similar to that implemented by the Generation Engineers.

The operation of Errochty station is dictated by the Central Control Room (CCR) in Pitlochry. The program cannot simulate these requirements and uses as much of the allocated water of the station at the time indicated by the demand curve to be the most costly generation period.

7.5.1.2. Daily Operation for Tuesday 27th of December 1983

The operation of the Tummel Valley scheme on the 27th of December corresponded to a cold winter's day. The majority of the precipitation within the valley will be falling as snow onto frozen ground. Thus, the natural runoff from each catchment will be low. The demand curve for this day is quite unusual. The majority of the generation was required later in the day, probably to supply consumer's heating systems. The daily outflows used by each station are given in Table 7-3.

Station	Daily Outflow
Cuaich	26.0
Loch Ericht	19.0
Rannoch	40.0
Gaur	59.0
Tummel	145.0
Errochty	52.0
Clunie	230.0
Pitlochry	291.0

Table 7-3: Daily Outflow for the 27th of December 1983

Figure 7-7, on the next page, shows the actual schedule used by the Generation Engineers on this day. The demand curve and the total generation for the group are shown on page 110, in Figures 7-8 and 7-9 respectively. The results obtained from the optimising procedure are shown on page 111. Figure 7-10 shows the hourly schedule and Figure 7-11 shows the total generation from the group for the optimiser results.

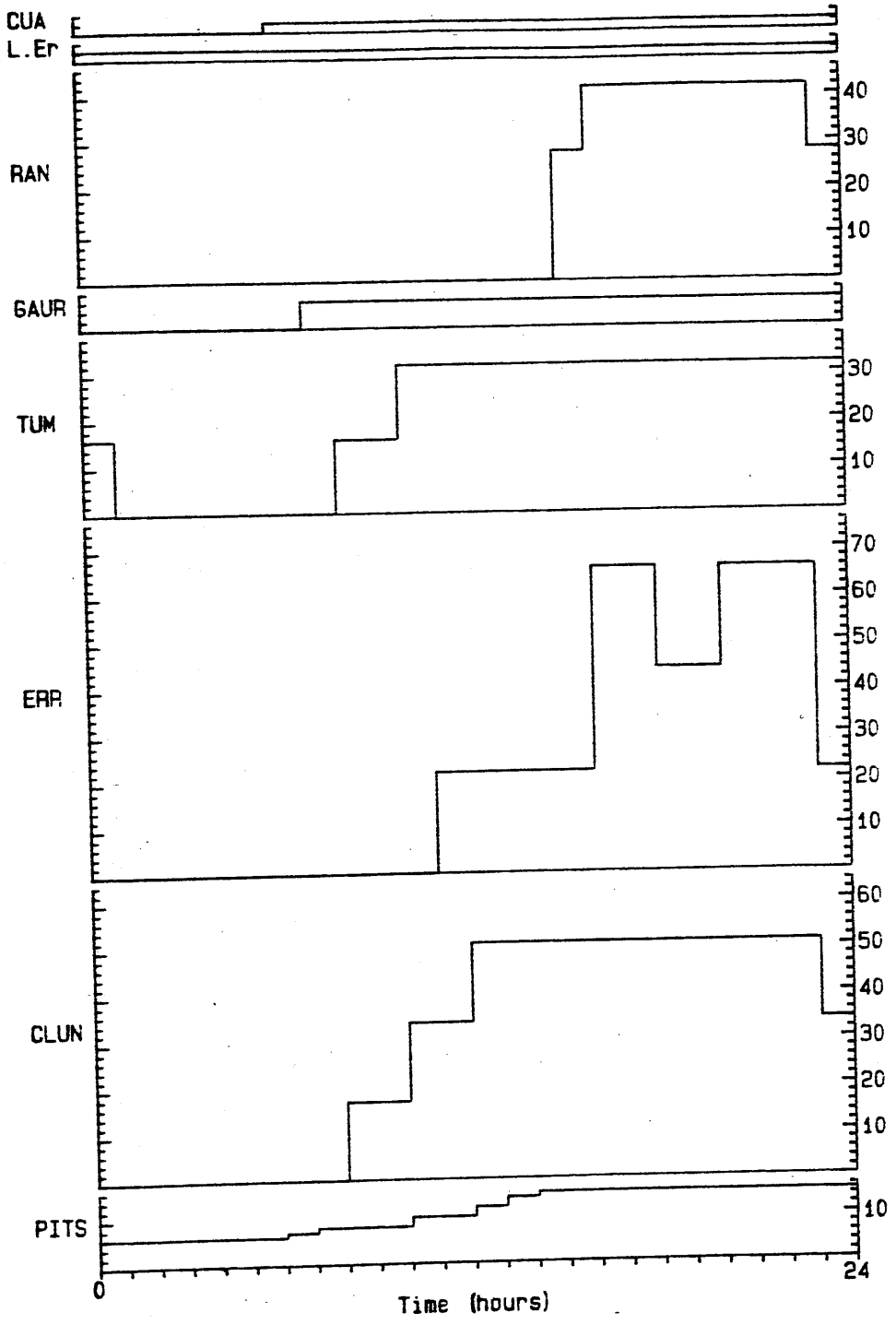


Fig. 7-7: The actual power schedule for the Tummel Valley, as used on the 27th of December 1983.

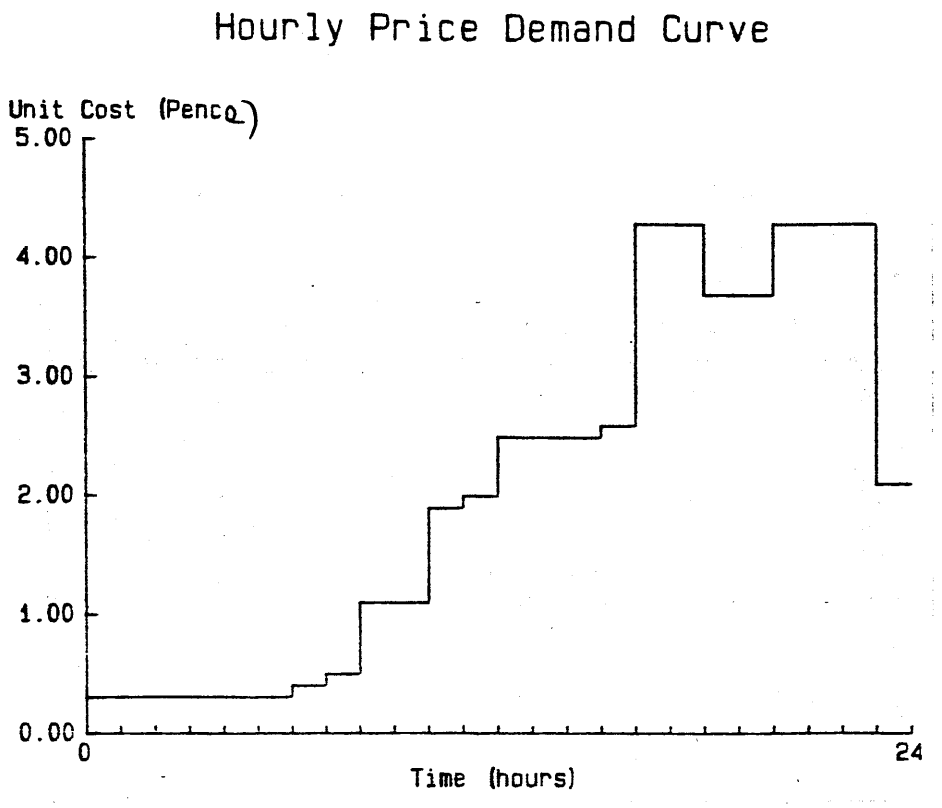


Fig. 7-8: Demand Curve used for the 27th of December 1983.

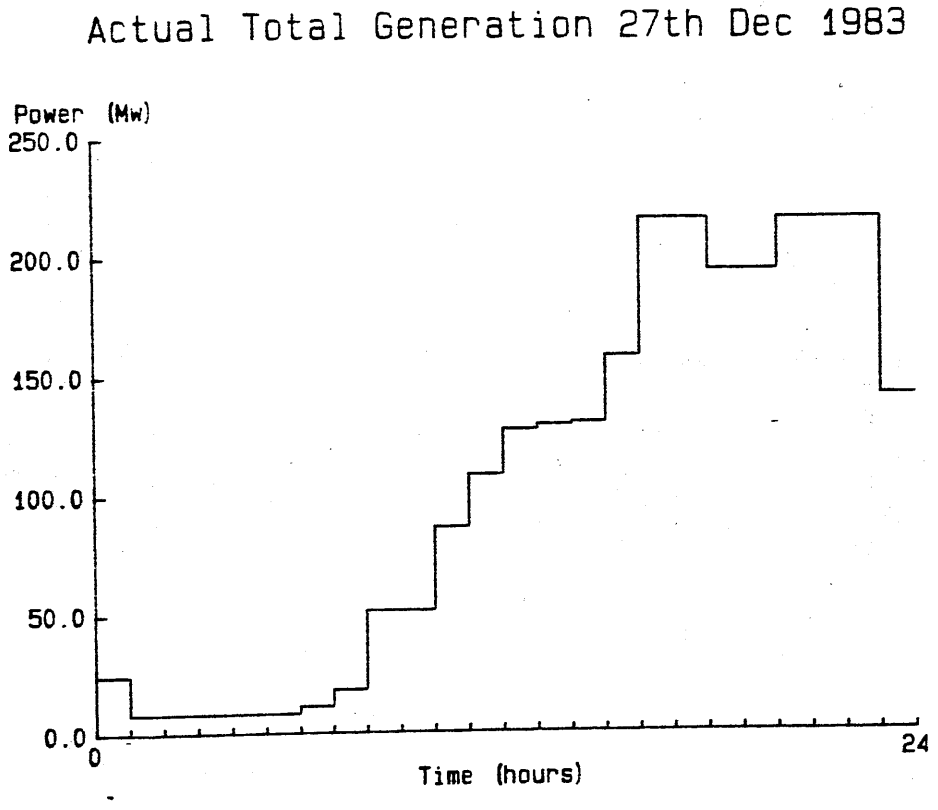


Fig. 7-9: Total Actual Group Generation on the 27th of December 1983.

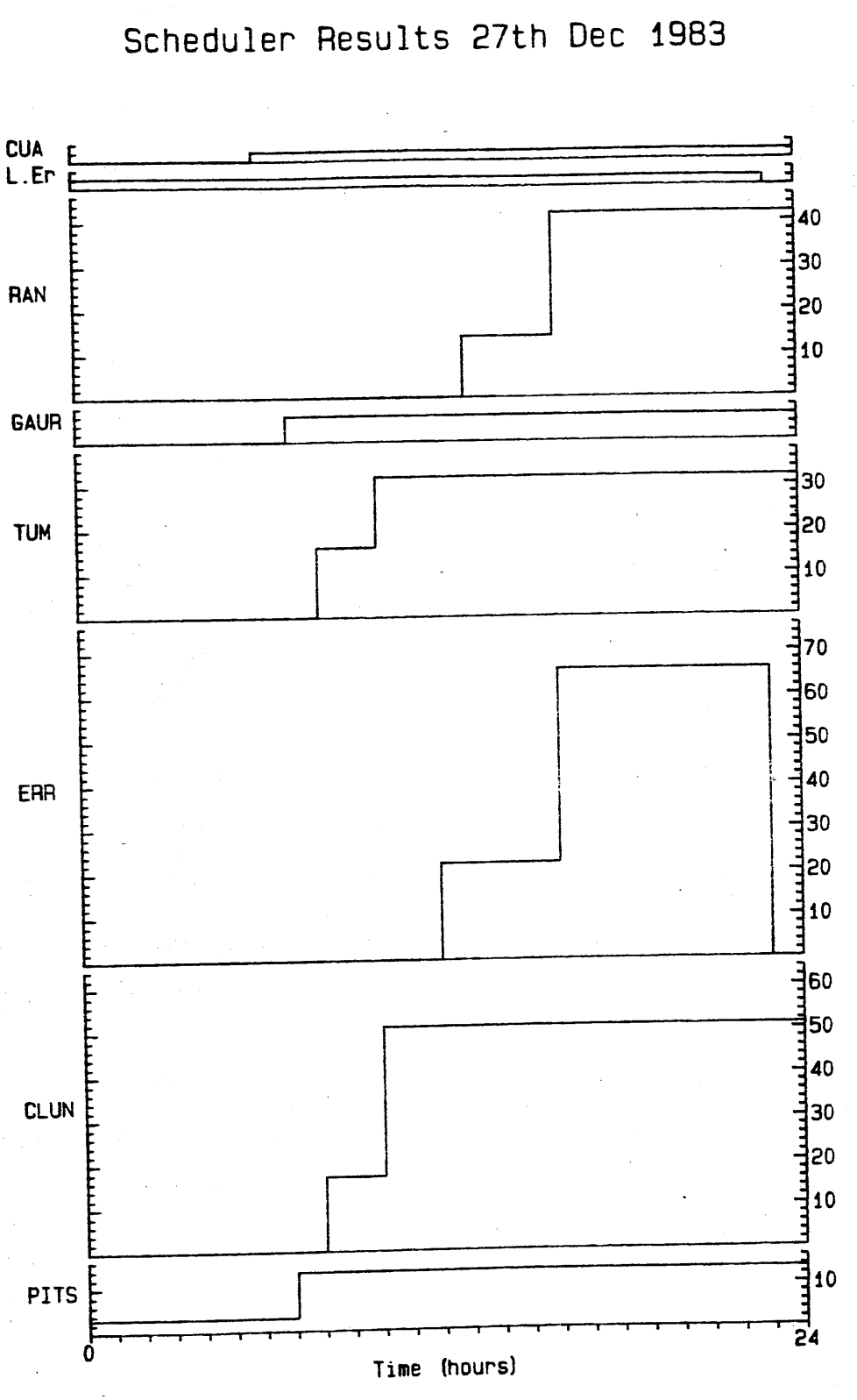


Fig. 7-10: Schedule produced by Computer Optimisation for the 27th of December 1983.

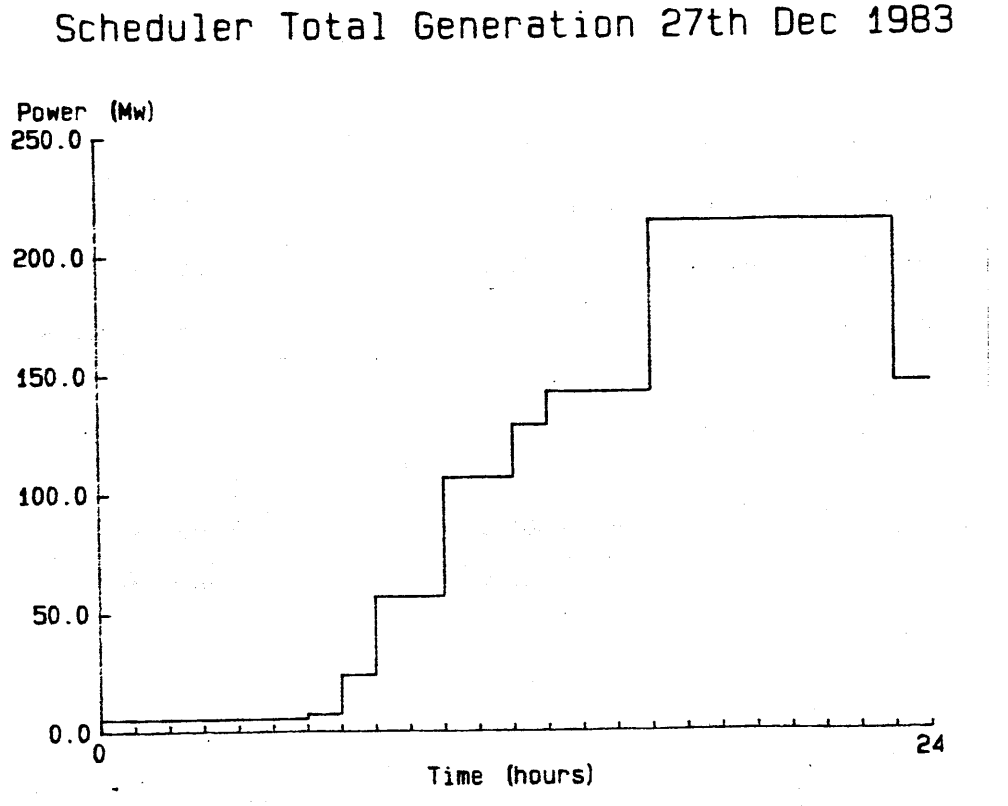


Fig. 7-11: Total Group Power for Optimiser Results for the 27th of December 1983.

The schedule produced by the optimiser, Figure 7-10, was identical to that implemented by the Generation Engineers, Figure 7-11, with the exception of the operation of Clunie and Pitlochry stations. The optimiser managed to follow the unusual demand curve very well.

The optimisation made use of a prediction of the inflow to Loch Faskally. The computer was able, therefore, to plan ahead to produce a better schedule for both Clunie and Pitlochry stations. The Generation Engineers ,however, had to adjust their planning for both stations in view of some unexpected inflow.

7.5.1.3. Daily Operation for Thursday 1st of April 1982

The natural inflow for a typical day in early spring is artifically increased by snow melt. Thus, the operation on the 1st of April had a number of stations generating for 24 hours. The daily outflow which was used on this day is given in Table 7-4.

Station	Daily Outflow
Cuaich	0.0
Loch Ericht	19.0
Rannoch	105.0
Gaur	31.0
Tummel	144.0
Errochty	43.0
Clunie	154.0
Pitlochry	183.0

Table 7-4: Daily Outflow for the 1st of April 1982

Figure 7-12, on the next page, shows the actual schedule used by the Generation Engineers on this day. The demand curve and the total generation for the group are shown on page 115, in Figures 7-13 and 7-14 respectively. The results obtained from the optimising procedure are shown on page 116. Figure 7-15 shows the hourly schedule and Figure 7-16 shows the total generation from the group for the optimiser results.

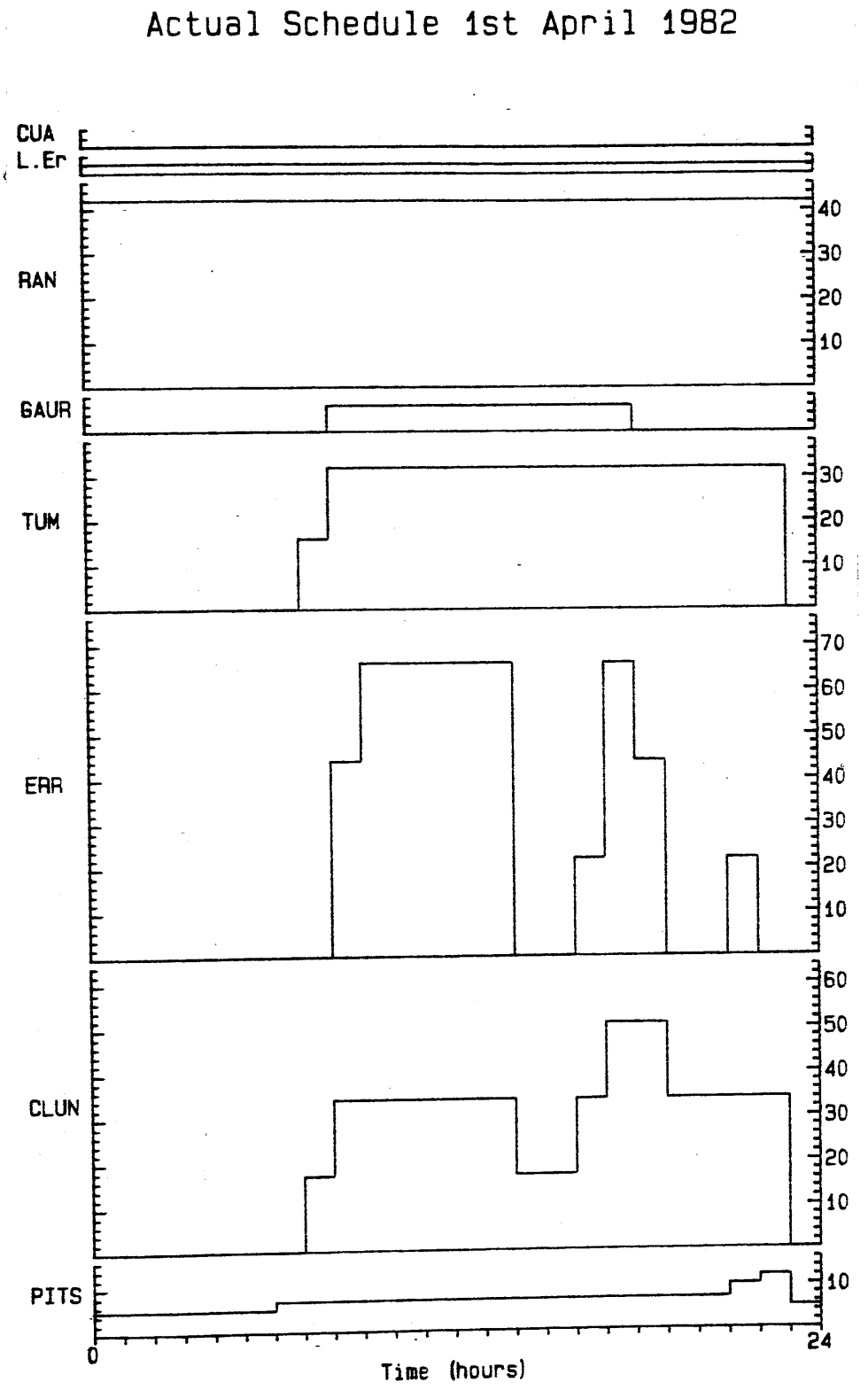


Fig. 7-12: The actual power schedule for the Tummel Valley, as used on the 1st of April 1982.

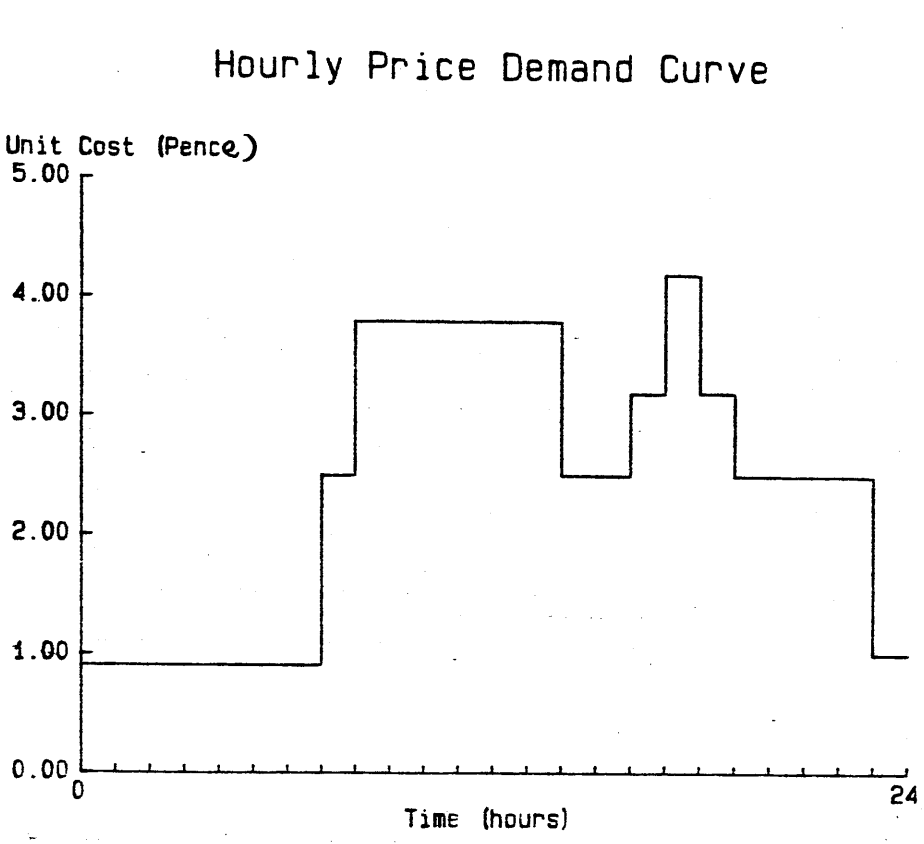


Fig. 7-13: Demand Curve used for the 1st April 1982

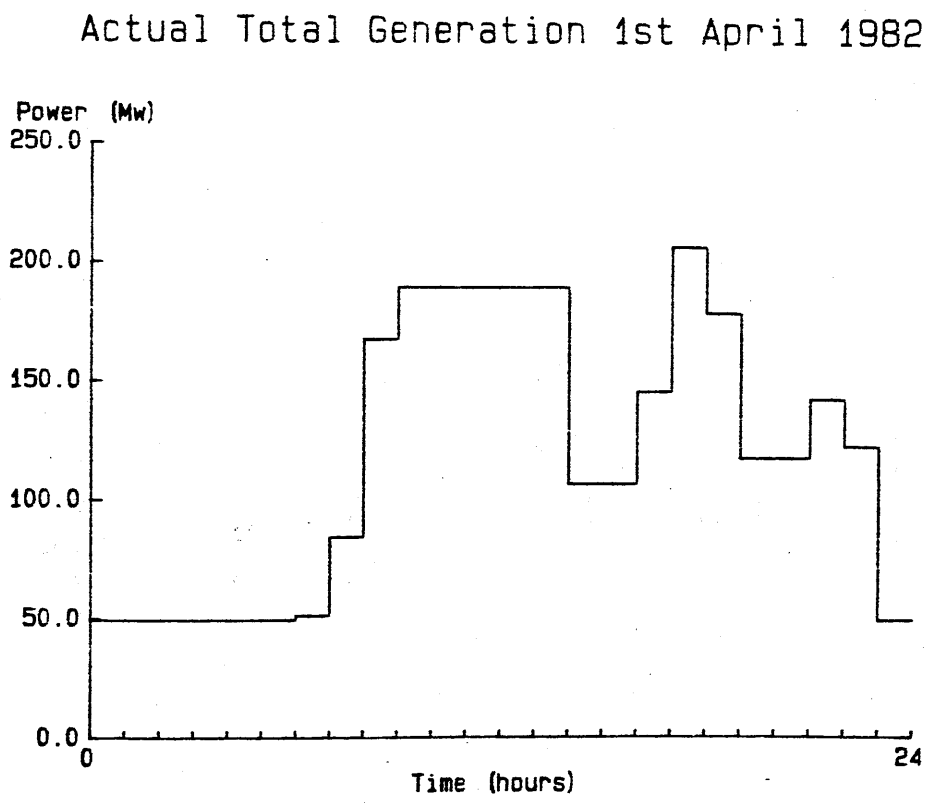


Fig. 7-14: Total Actual Group Generation on the 1st April 1982.

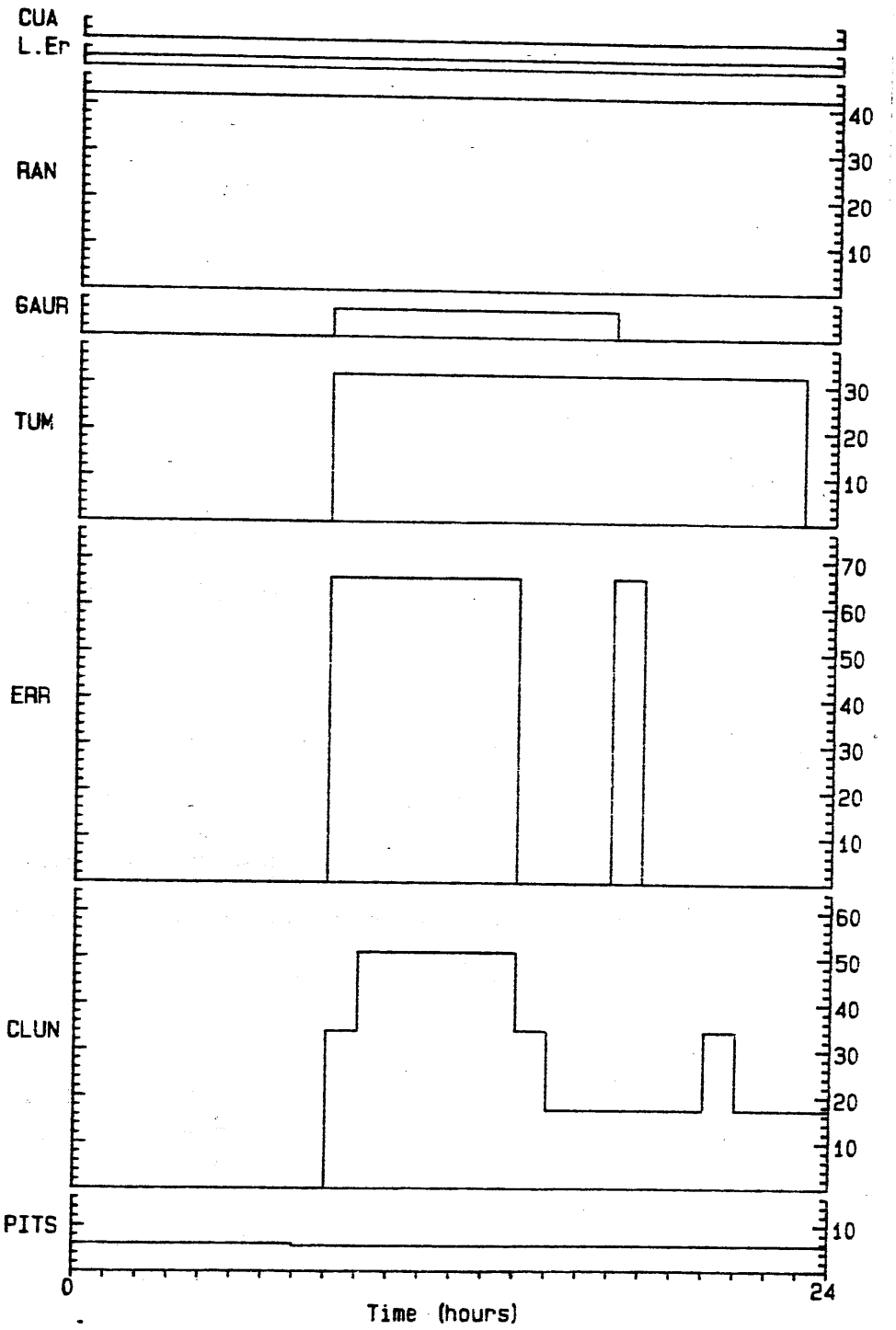


Fig. 7-15: Schedule produced by Computer Optimisation for the 1st of April 1982.

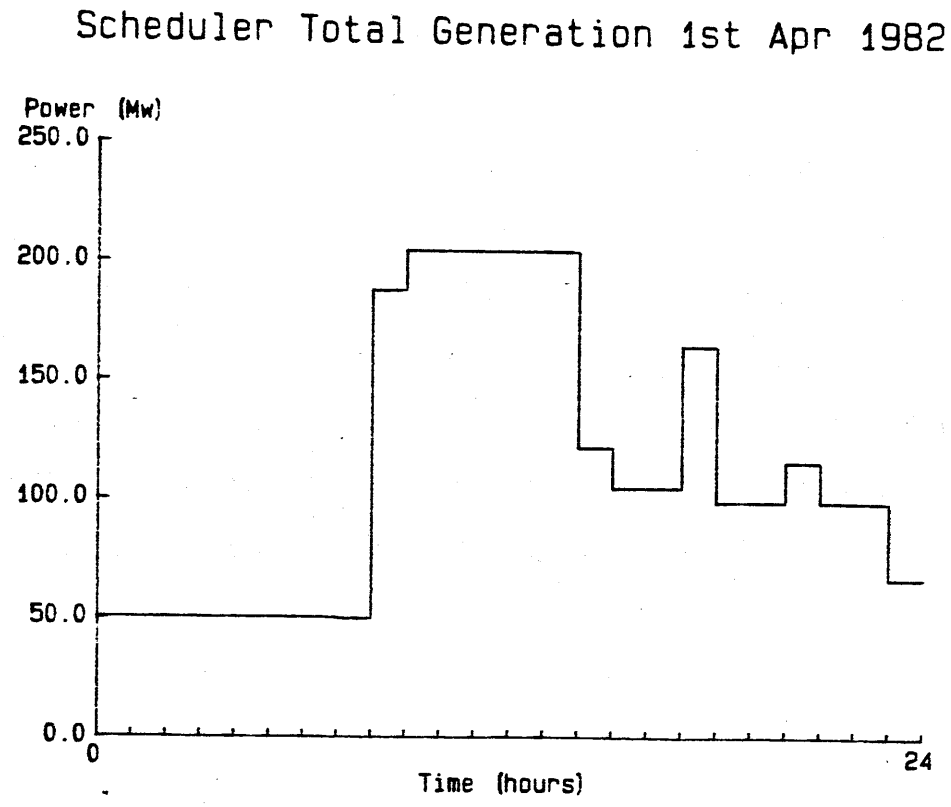


Fig. 7-16: Total Group Power for Optimiser Results for the 1st of April 1982.

As can seen from Figures 7-12 and 7-15, the two schedules are near identical. The more controlled scheduling of Clunie and Pitlochry, as can be seen from Figure 7-15, resulted in a different operation from the actual implementation by the Generation Engineers.

7.5.2. Study of Clunie-Pitlochry Scheduling Under Adverse Conditions

As can be seen from the results of the previous section, the methods used to schedule Clunie and Pitlochry stations worked very well under normal operating conditions. The unique operating constraints placed upon Pitlochry station require that the scheduling method be tested under adverse conditions.

Consider the operational schedule for the 1st of April 1982,

it is feasible that a sudden thunderstorm could pass over the valley at around 10:00 hrs. This storm would increase snow melt and result in a pulse of inflow from each catchment. With its generation already fixed, Pitlochry station may not be able to release enough water to stop Loch Faskally spilling due to the combined inflow from the storm pulse and Clunie station. In an automatic system, the pulse of inflow would be detected and the valley rescheduled. The results of this reschedule are shown for Clunie and Pitlochry stations in Figure 7-17.

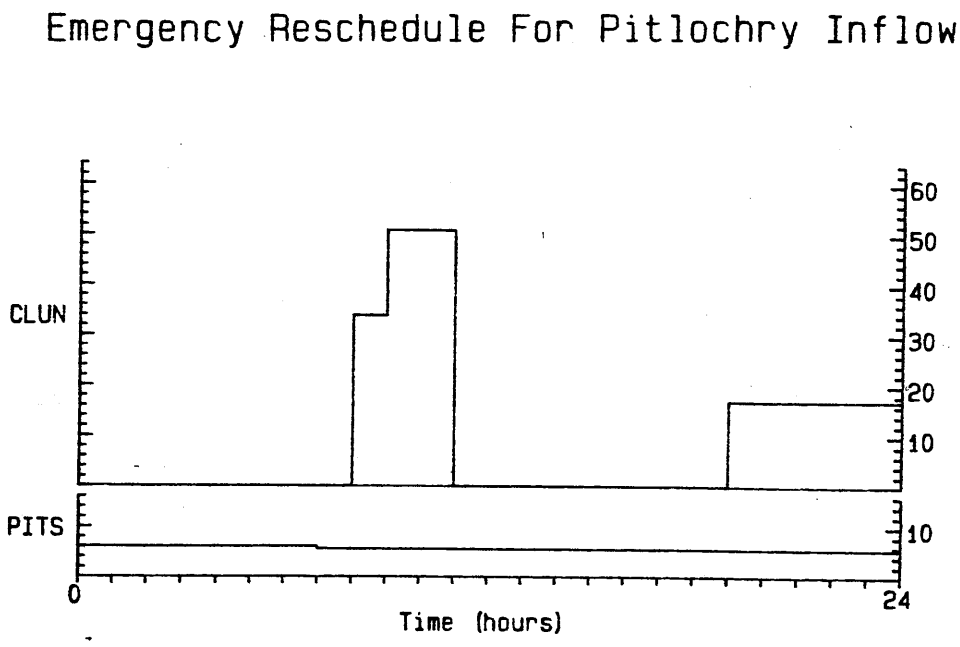


Fig. 7-17: Reschedule Results for Clunie and Pitlochry Stations for Storm inflow pulse

The computer reschedules the operation of Clunie Station so that the station does not generate until later that day or during the next day. This is exactly the operational change that the Generation Engineers would make.

It is possible that Loch Tummel could be near full. If that

were the case, the combined effect of the inflow pulse and holding off the station's generation would result in Loch Tummel spilling. In this situation, the optimiser results are shown in Figure 7-18.

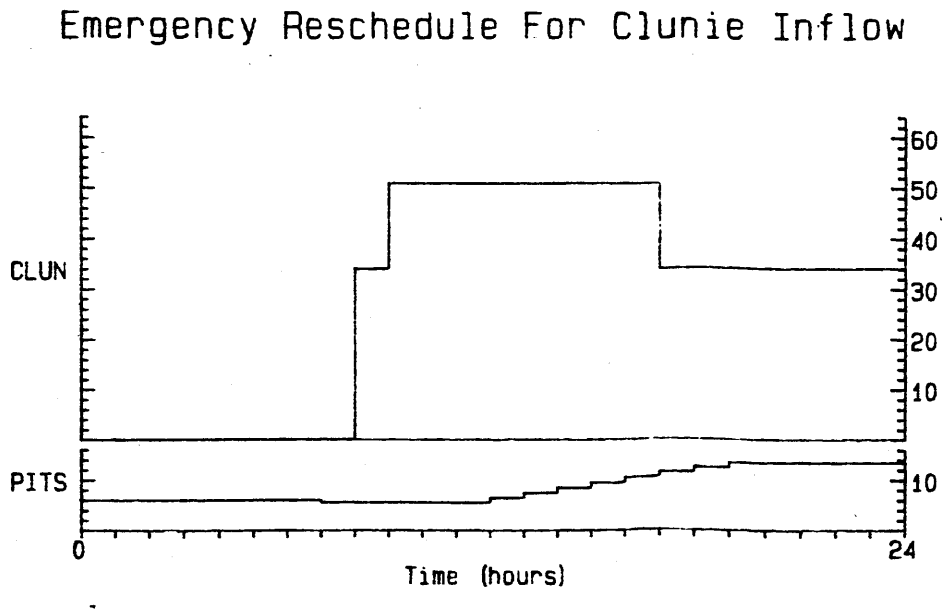


Fig. 7-18: Reschedule with Loch Tummel Near Full

The operation of Clunie station has not been changed. Pitlochry station violates its constraints and increases its generation by the minimum amount necessary to avoid spillage. This, again, is the operational change that the Generation Engineers would make. However, the engineers maintain that if Loch Tummel was full then Tummel and Errochty stations would not have been used and Clunie station would have been generating overnight.

7.5.3. Evaluating the Emergency Scheduling of the Optimiser

Consider the operation of the 27th of December 1983, if at around 7:00 hrs a large thermal set has, for some reason, to stop generating, then the national grid would require a large increase in generation. Hydro is one of the fastest types of plant to

plant to respond to such an increase in power demand. The demand curve for the scheme on the 27th of December was modified to be as shown in Figure 7-19.

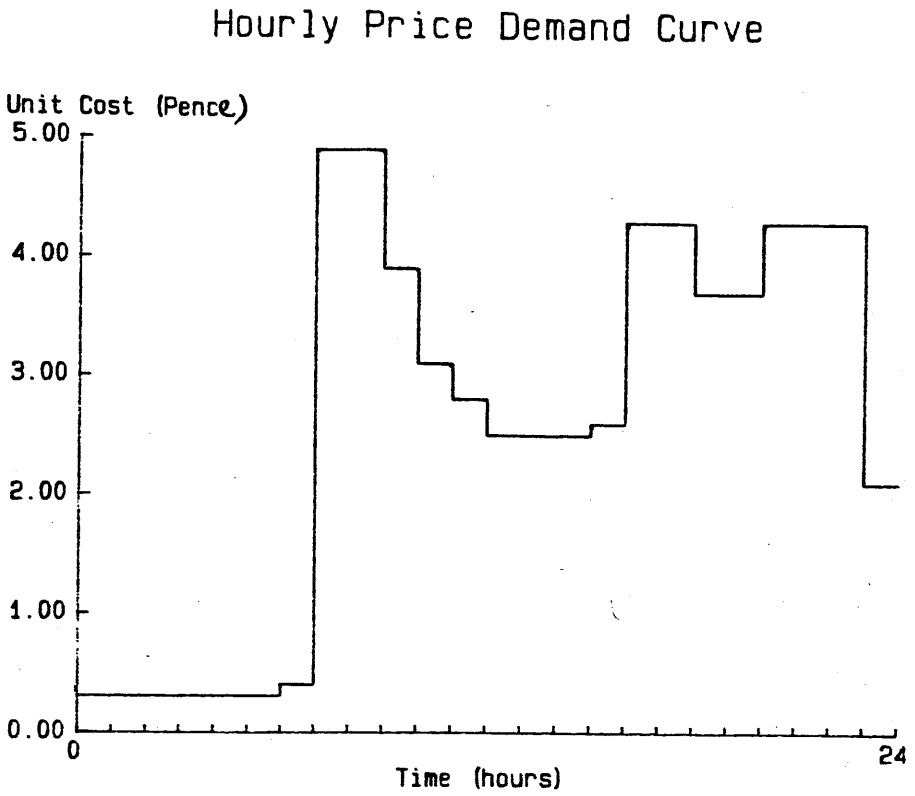


Fig. 7-19: Emergency Modified Demand Curve

The resulting schedule is shown in Figure 7-20, on the Next page.

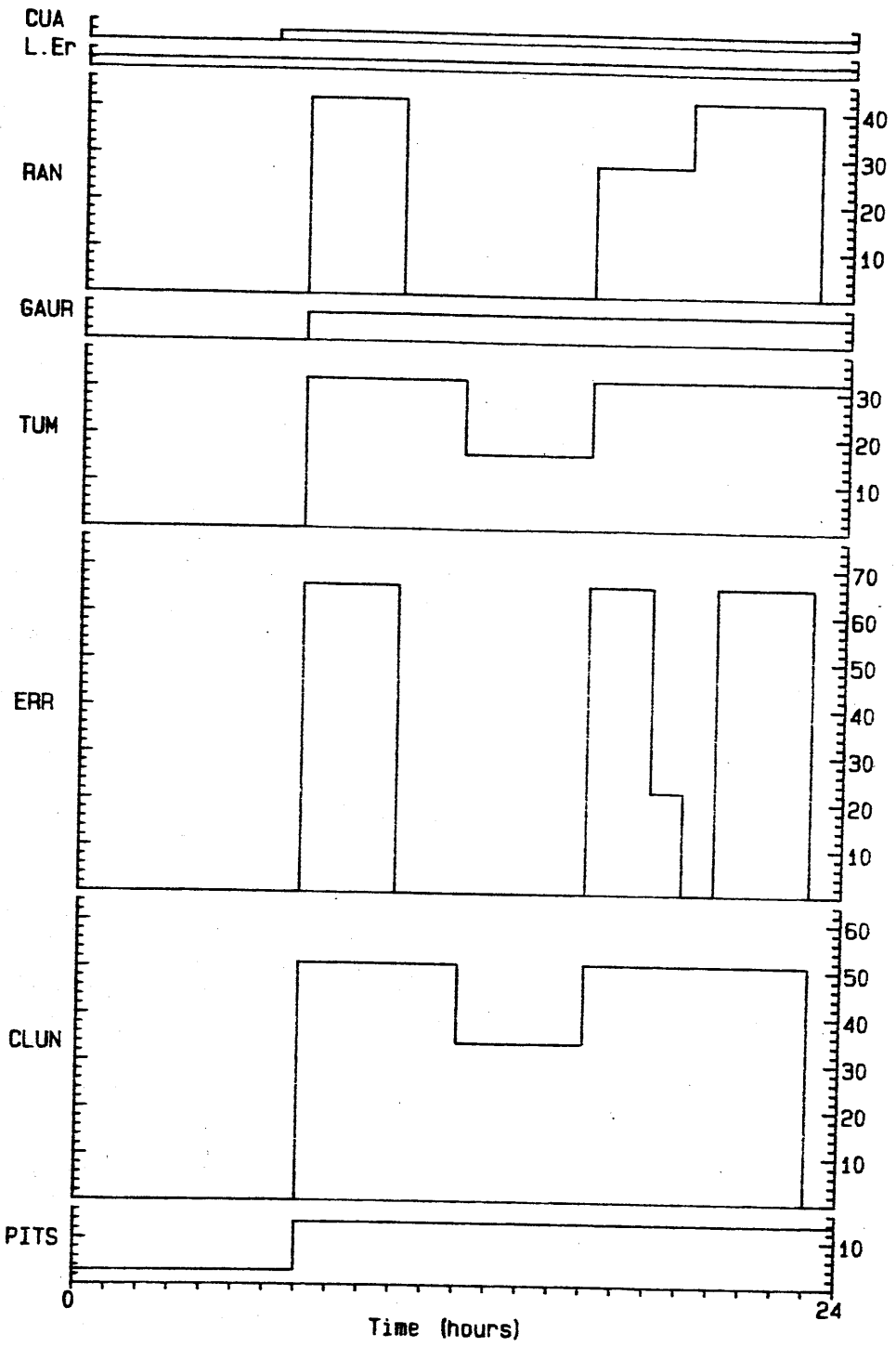


Fig. 7-20: Resulting Emergency Schedule for Modified Demand

The optimiser uses all available sets for generation at the time of the emergency. The effects of these changes are compensated for later that day or during the next day. The operation of Clunie station is adjusted as much it possibly can so as not to violate the constraints of Loch Faskally and Pitlochry station. The Generation Engineers would also have used all available generation in this emergency. The Generation Engineers would not have varied the operation of Clunie station, as the manual calculations required before varying its operation would have taken too long.

7.6. Discussion of the Hourly Scheduling System

The optimiser performed equally as well as the Generation Engineers in the normal operation of the scheme. Small differences between the computer schedules and the actual schedules are due to the intervention of the Central Control Room (CCR) in the actual schedules. The computer optimisation did not interact with CCR to consider the needs of the national grid. The speed of response of the system and the possibility of automatic rescheduling when unexpected inflows are detected allows the optimisation system to produce better operational schedules for Clunie and Pitlochry stations. In emergencies, the optimiser responds faster with more accurate solutions than the Generation Engineers.

At times when the Tummel Valley scheme must operated under highly constrained conditions, such as at times of spillage, the computer program gives better results. The program, by considering all options, is capable of operating the scheme in regions which are closer to the boundary of the feasible operating area. While operating in this region produces better solutions,

the dangers of violating constraints due to unexpected events cannot be ignored. Any error in the inflow prediction could easily result in spillage.

8. SET POINT CONTROL FOR TURBINE DISPATCH

8.1. Introduction

The majority of hydro-turbines, at present in operation in Scotland, are governed by mechanical Temporary Droop (TD) governors. The TD governor is characterised by a dominant time constant of the order of 3 minutes. This long time constant is necessary to ensure that the turbine remains stable during the possible frequency variations associated with supplying an isolated load. Loading the turbine is carried out by varying the frequency set point of the governor. Hence, any load changes must be implemented by the governor, resulting in a slow loading time for the turbine. The loading procedure is further complicated by the non-linear response of the governor. For large variations in frequency set point the governor characteristics may saturate, dramatically reducing the dominant time constant. The frequency set point of the governor is supplied by a speeder motor, whose maximum output is limited. The speeder motor is incremented by a pulse input which ramps it up or down in response to a Raise or Lower command. Control, such as this, which requires slow operator intervention, renders it impossible to achieve a desired load simultaneously and accurately on a number of machines.

The loading problems associated with the TD governor, and speeder motor, have been overcome by using modern micro-processor[76] or electronic governors which allow a power loading signal to be injected after the governor dynamics. It would not be cost effective, however, to replace existing TD governors with modern governors before the end of their planned operational life-time. The performance of existing machines can be greatly

enhanced by the addition of a micro-processor based set point controller which can control the turbine via the existing speeder motor and TD governor.

This micro-processor based set point controller must be able to load the turbine in response to remote computer instructions, either, at maximum speed, or, at some controlled rate. In order to enable the set point controller to be used in conjunction with the existing governor equipment, the controller must be able to take account of the non-linear changes in the dominant time constant of TD governors and be immune to the effects of noise on the measured power signal.

Such a set point controller has been developed by Dr H. Davie, of the department of Electrical Engineering at Glasgow University. The controller was developed outwith the scope of this project and only its basic operating concepts are reported here. In order to use this controller to dispatch the hourly schedule in the Tummel Valley scheme, the controller had to be fully tested on plant models and real plant. Although not actually tested on stations within the Tummel Valley scheme, the controller was tested upon similar stations within Scotland. This chapter details the development of the plant models and the results from site tests of the controller.

8.2. Modelling a Hydro Turbine with a Temporary Droop Governor

The department of Electrical Engineering at Glasgow University has been investigating techniques of speed governing of hydro turbines with experimental results from Sloy Power station[4,77,78]. For this work, a model of the TD governor,

operating only in its non-saturation region was developed. In order to test load control, the governor's saturation characteristics were also required. These characteristics were determined experimentally at Sloy power station. Similarly, it was necessary to develop a model of the existing speeder motor. A simple model of this was also determined experimentally at Sloy power station. A simplified block diagram of the system is shown in Figure 8-1.

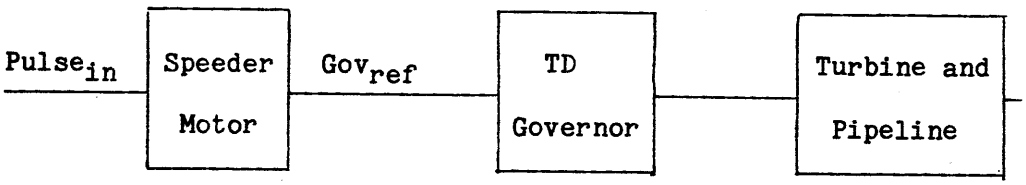


Fig. 8-1: TD Governed Hydro Generating System

8.2.1. The Speeder Motor

The dynamics of the Speeder Motor, as observed on site, can be represented by the equation,

$$Gov_{ref} = \frac{K_s}{s\ (\ 1 + sT_s\ )} * Pulse_{in} \qquad \text{—Eq. 8.2.1}$$

where K_s is the gain of the motor,

T_s is the speeder motor time constant, and,

s is the laplace operator.

The input, $Pulse_{in}$, can only be one of the set [-1, 0, 1] which corresponds to [lower, nothing, raise]. The output, Gov_{ref}, is restricted in value to -0.05 to +0.05 which corresponds to a

desired power set point of -1.66 to 1.66 per unit (p.u.).

8.2.2. The Temporary Droop Governor

The mechanical TD governor has a complicated transfer function. A representation of the TD governor model is given in Figure 8-2.

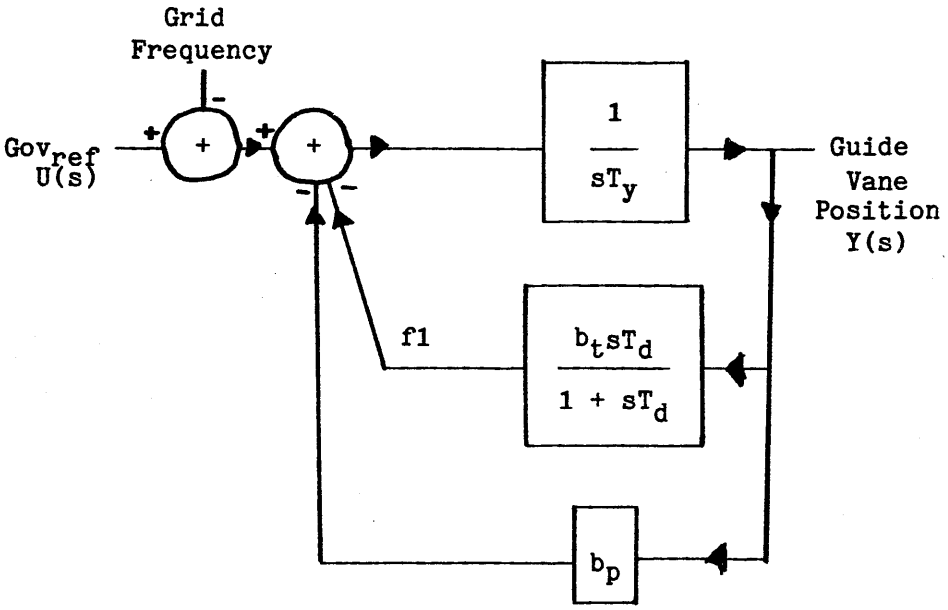


Fig. 8-2: Dynamics of the TD Governor

The dynamics of the temporary droop governor are composed of mechanical and hydraulic systems. The $1/sT_y$ term corresponds to the integrating action of the main hydraulic servo which operates the guide vanes. T_y is the constant of integration of the main servo. The term b_p, the permanent droop, is the gain of the governor and is normally set to give a 100% change in output power for a 3% variation in set point or grid frequency. The feedback loop, marked F1 in Figure 8-2, represents the mechanics of the temporary droop action. This feedback effectively damps the operation of the governor, reducing the gain during transients,

and thus maintaining the stability of the turbine under isolated load conditions. The governor operation depends on the grid frequency. For loading purposes, any variation in frequency set point would be vastly greater than the variations of grid frequency. Thus, during any loading period, the grid frequency can be assumed to be constant. This gives the TD governor a transfer function of,

$$\frac{Y(s)}{U(s)} = \frac{\frac{1}{b_p}(1 + sT_d)}{\frac{T_yT_d}{b_p}s^2 + \frac{1}{b_p}(T_y + T_p(b_t + b_p))s + 1} \qquad \text{—Eq. 8.2.2}$$

Substituting the parameters as measured for Sloy power station[76], as,

$$b_p = 0.03$$
$$T_d = 16.0$$
$$T_y = 0.3$$
$$b_t = 0.25$$

gives,

$$\frac{Y(s)}{U(s)} = \frac{\frac{1}{0.03}(1 + s16.0)}{(1 + s160.0)(1 + s)} \qquad \text{—Eq. 8.2.3}$$

This model was further modified after site tests showed the feedback loop marked f1 in Figure 8-2 saturated for large changes in desired set point. An internal mechanical dash pot is responsible for this behaviour. For small variations of set point within the saturation region, the transfer function of the TD

governor becomes non-linear. By including this effect into Equation 8.2.2, the resulting transfer function becomes,

$$Y(s) = \frac{\frac{U(s)}{b_p}}{\left(1 + s\frac{T_y}{b_p}\right)} - \frac{\frac{sat}{sb_p}}{\left(1 + s\frac{T_y}{b_p}\right)} \quad \text{—Eq. 8.2.4}$$

where, sat is the saturation value of the F1 feedback loop.

This non-linear transfer function may only be fully evaluated using simulation techniques. Using the Sloy values for the parameters, the simulations showed that the effect of the saturation was to reduce the dominant time constant of the governor to approximately 16% of its unsaturated value. Thus allowing turbines to be loaded in 16% of the previously expected time.

8.2.3. Turbine and Pipeline Dynamics

The pipeline, which supplies the turbine, can be thought of as a simple inductor[78]. The pipeline is thus modelled as,

$$1 - H_t = T_w * \frac{dQ}{dt} \quad \text{—Eq. 8.2.6}$$

where H_t is the effective head across the turbine,

T_w is the water time constant, and,

Q is the flow through the turbine.

With little loss of accuracy, the turbine can be assumed to have fixed losses, and is modelled by the non-linear equations,

$$Q = G * (H_t)^2 \qquad \text{--Eq. 8.2.7}$$

$$P = Q * H_t \qquad \text{--Eq. 8.2.8}$$

and,

$$Pout = P - losses + Noise \qquad \text{--Eq. 8.2.9}$$

where G is the Guide Vane position,

P is the power into the turbine, and

Pout is the measured power out of the turbine.

The complete turbine and governor model was implemented using a simulation package developed within the Department. The power output of the turbine was taken to be the power input to the turbine minus fixed friction losses, and a random noise of 0.05 p.u. was added to the power output to simulate noise measured at site.

8.3. Overview of the Development and Operation of the Set Point Controller

An attempt was made to design a load controller for a hydro-turbine equipped with a mechanical temporary droop governor. The intention was to monitor the output power from the turbine and compare it with the desired power. The difference, the feedback error signal, would then be processed by the dynamics of the load controller to provide a three-state output which would raise, maintain or lower the frequency reference signal to the mechanical governor. The power measurement could be simply achieved by inserting a resistor in the current loop of the Station power metering, and reading the voltage via an analog to digital converter into the microcomputer used as the load controller. The automatic raise/lower controls were implemented

by inserting relays in parallel with the operator's raise/lower switches. An interlock box was used to ensure that in the event of computer failure there was no chance of both the raise and lower controls being operated simultaneously.

Initial trials were carried out at Sloy Power station and the following major problems were noted:-

(1) The non-linearity in the governor which resulted in its dominant time constant varying by almost a factor of 6 depending on the size and rate of loading signal applied at the frequency reference point.

(2) The three-state nature of the control signal - raise/maintain/lower.

(3) Noise on the the power measurement and noise on the system frequency which is also effectively injected at the frequency reference point.

Initial tests showed that a controller based on proportional plus derivative action showed some promise but had a number of defects. Although able to execute a large load step in a short time, as it approached its final value it caused the servo motor to hunt backwards and forwards in a limit-cycle-like condition. This was believed to be due to the change in the governor from its short-time constant mode to its long time constant mode. A set of parameters could be selected which would cope with one, but not both of these modes. In addition noise was a considerable problem, and increasing integration of the power signal in an attempt to reduce this effect led to instability.

After many trials against simulations, a novel controller was developed to overcome thse defects. Essentially the controller

also operates in a non-linear manner, with its dynamics changing with the size and rate of load change. In addition the effects of noise are greatly reduced by integrating the control signal and employing hysteresis and deadband on the control action. Instability is avoided by introducing a measure of feedforward control which estimates the control action and discharges the integrator appropriately.

Although this controller was originally designed to permit large load steps to be performed in an optimally short time, its dynamics should be able to cope with any size or rate of step. Tests have shown this to be so, and hence it can also be used to permit loading at operator-requested ramp rates - often a desirable feature in turbine loading.

8.4. Testing the Controller on Site

The controller was tested at three hydro power stations : Loch Sloy on the 5-6th of January 1984, Fasnakyle on the 17-18th of January 1984 and Torr Achilty on the 19-20th of January 1984.

Loch Sloy station, on the banks of Loch Lomond, has four 32Mw sets with a working gross head of about 850 ft. The turbines are vertical francis turbines and are governed by an English Electric Temporary Droop governor. Fasnakyle, part of the Affric-Beauly group near Inverness, has three 22Mw sets operating under a gross head of 500 ft. The sets are similar to those at Sloy. Torr Achilty is a run-of-river station which is part of the Conon group, again near Inverness. The station has two 8Mw sets, with a 48 ft head. The two Kaplin turbines are governed by Boving Temporary Droop governors.

The format of the site work was similar at each station. After connecting data logging equipment, tests were run to establish the parameters of each generating set. The set was then modelled on a microcomputer and the controller parameters adjusted to successfully control the computer model. The controller was then attached to the actual equipment and the real response to the controller's action logged.

8.4.1. Parameter Estimation On Site

The procedure followed to estimate the parameters of each station was identical. It was known that the governors at Sloy and Fasnakyle would have saturation characteristics. The governor at Torr Achilty, however, was unknown and actually, from the site tests, showed no saturation.

Before the turbine was actually started, the speeder motor was repeatedly pulsed using an increasing length of pulse. In the turbine room, the number of turns with which the speeder motor responded to each pulse was manually recorded. From a graph of the number of turns against the length of pulse, it was possible to make accurate estimates of both G_S and T_S. It was found, however, that the value of T_S was non critical to the simulation.

With the turbine running the speeder motor was again pulsed with increasing lengths of pulse. The turbine output was allowed to settle to its steady state value. Using this value, the frequency set point out of the speeder motor, and the current grid frequency, it was possible to calculate an average value for the permanent droop, b_p. By data logging the dynamic response of the turbine to each increasing pulse, a large amount information on

each turbine was gathered. A simulation was set up initially using the Sloy parameters already reported for the parameters, T_d, T_y, and b_t. The parameters and the saturation value were then adjusted to match the responses already logged.

The set point controller was then tested and adjusted using the final simulation until its correct operation was ensured.

For safety reasons, the turbine was then run up and down a few times manually. The controller input was connected to the signal representing the power out of the turbine and the operator attempted to follow the controller's suggested raise and lower output. The correct operation of the controller was thus established before the controller was connected to control the actual turbine. This data was also logged and used to compare with the computer simulation.

8.5. Final Model Parameters After Site Tests

For each station that was modelled Table 8-1 shows the estimated model parameters. Sat is the point at which saturation occurred in feedback loop F1.

	G_s	T_s	b_p	T_d	T_y	b_t	Sat
Sloy	0.0021	1.00	0.03	16.0	0.3	0.25	0.016
Fasnakyle	0.0009	1.00	0.03	20.0	0.6	0.4	0.016
Torr Achilty	0.0012	1.00	0.04	6.0	0.25	0.2	1.000

Table 8-1: Model Parameter for Loch Sloy, Fasnakyle and Torr Achilty Stations

The value of Sat=1.00 for Torr Achilty indicates that no

saturation was observed at that station.

8.6. Response to the Set Point Controller From Simulation and Site Visits.

The primary objective of the site tests was to show the capability of the set point controller to dispatch the hourly loading schedule derived from the optimisation. The speed of response of some stations is limited by the physical constraints of the station. For example, Tummel station is supplied with water by an open aqueduct. If the station is loaded in a fast step, a shock wave of water travels back along the aqueduct. This wave would spill over the sides of the aqueduct, eroding the aqueducts foundations. Some stations must, therefore, be loaded in a controlled ramp. This type of loading was tried on all three test stations.

The evaluation of the improved model of TD governed turbines was the second objective of the work. This was achieved by simulating each station. The simulation can easily be compared to the actual results.

The following graphs show the simulated and actual responses of one turbine at each of the test stations. Each station was tested for a step change in desired output power, and a controlled ramp loading of the set.

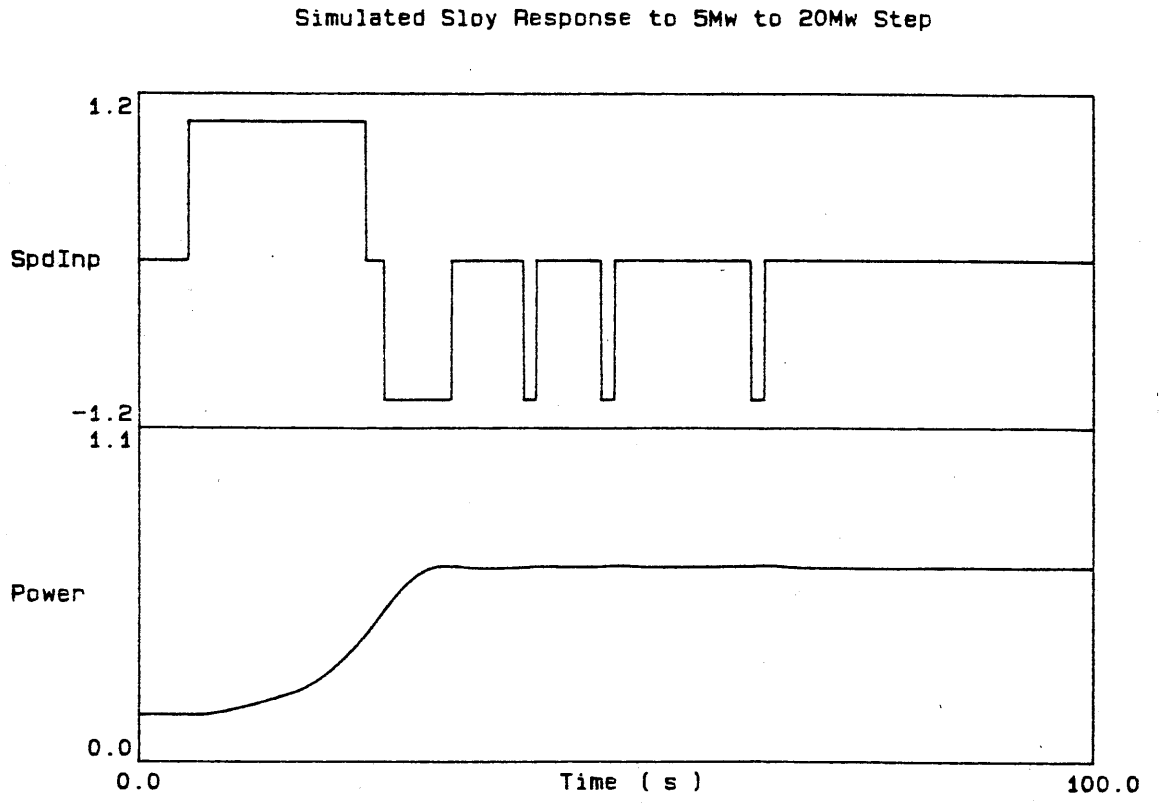


Fig. 8-3: Simulation of Loch Sloy 5-20Mw step.

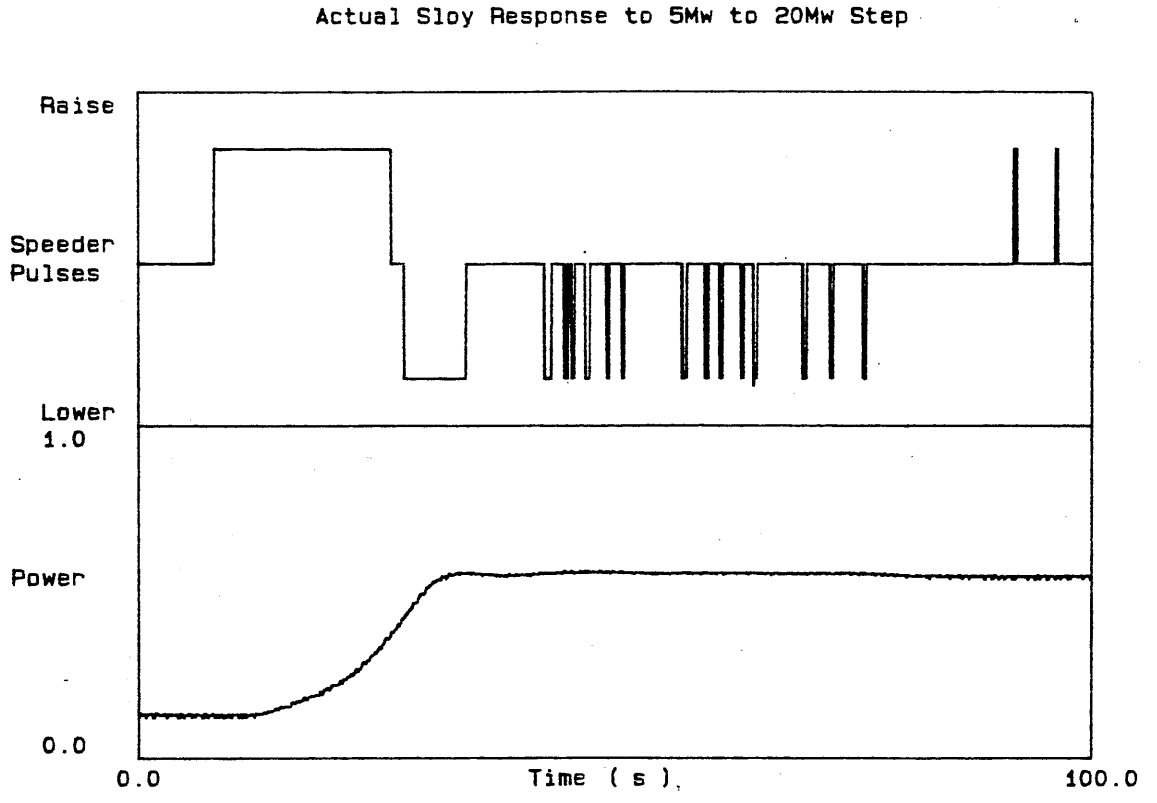


Fig. 8-4: Actual Response of No.1 Set at Loch Sloy to 5-20Mw Step

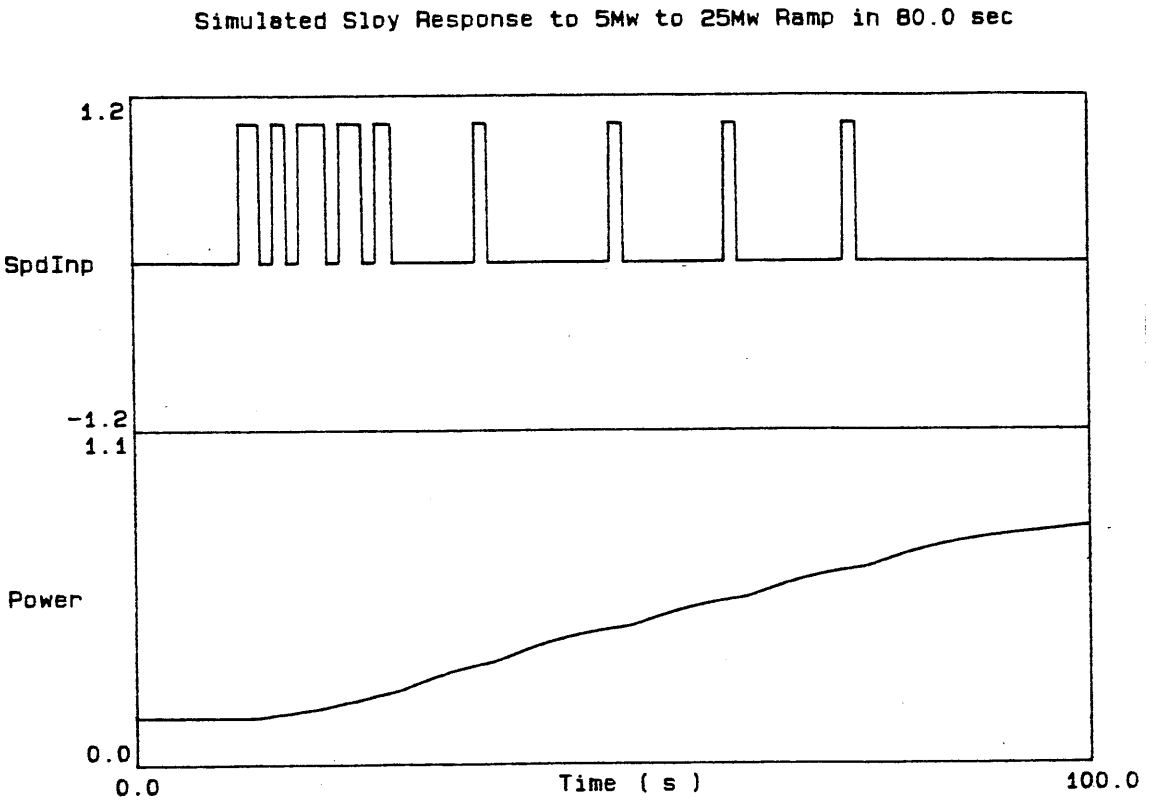


Fig. 8-5: Simulated Response of Loch Sloy to a 5-25Mw Ramp in 80sec.

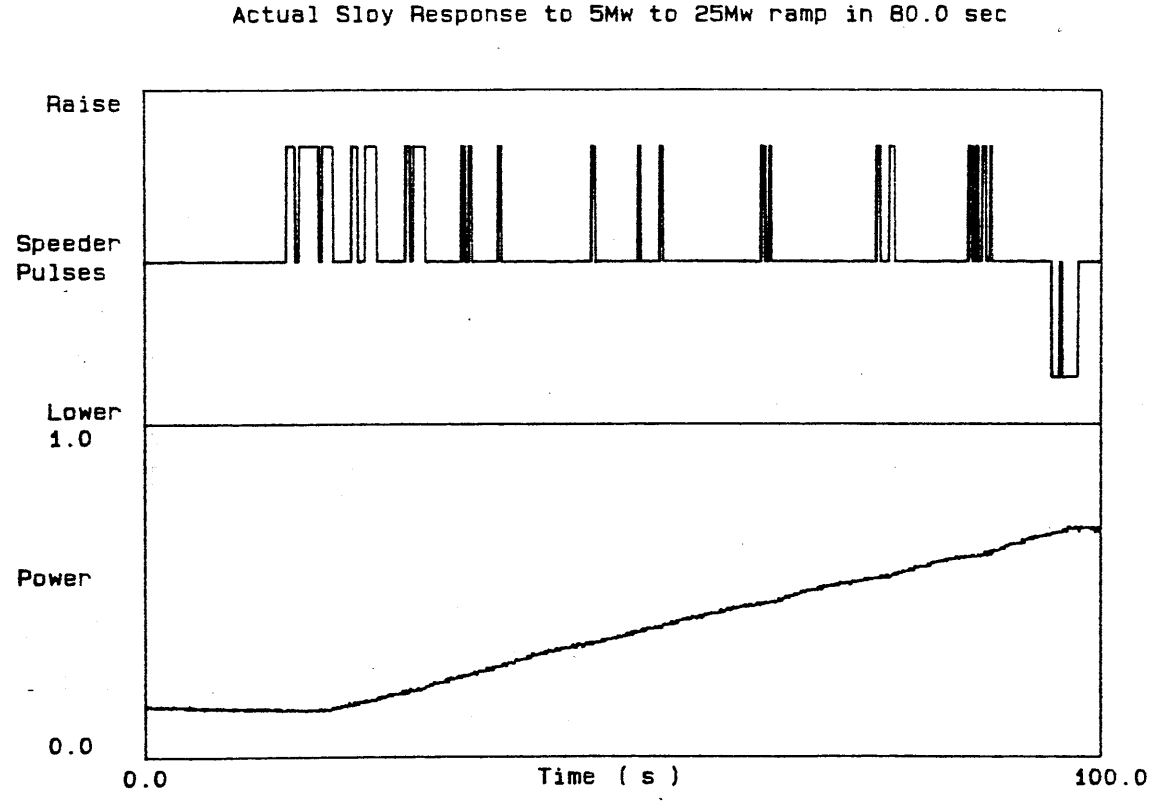


Fig. 8-6: Actual Response of No.1 set at Loch Sloy to a 5-25Mw ramp

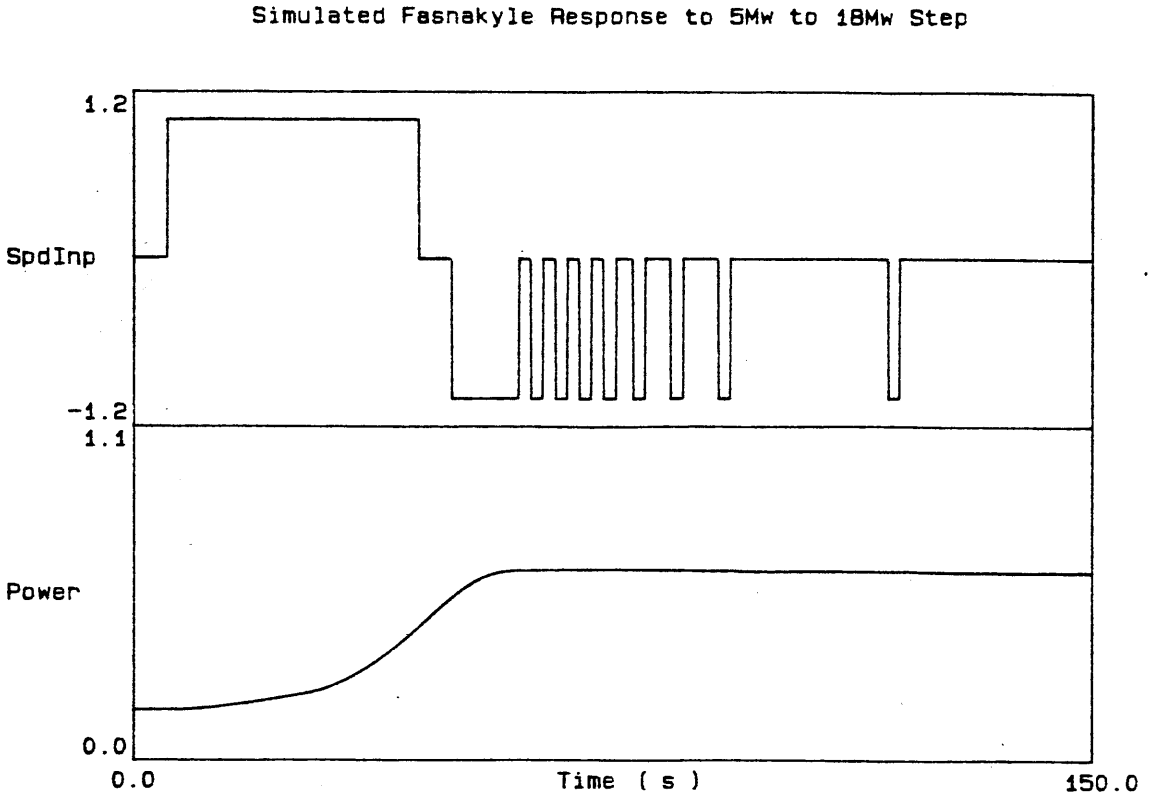


Fig. 8-7: Simulation of Fasnakyle 5-18Mw step.

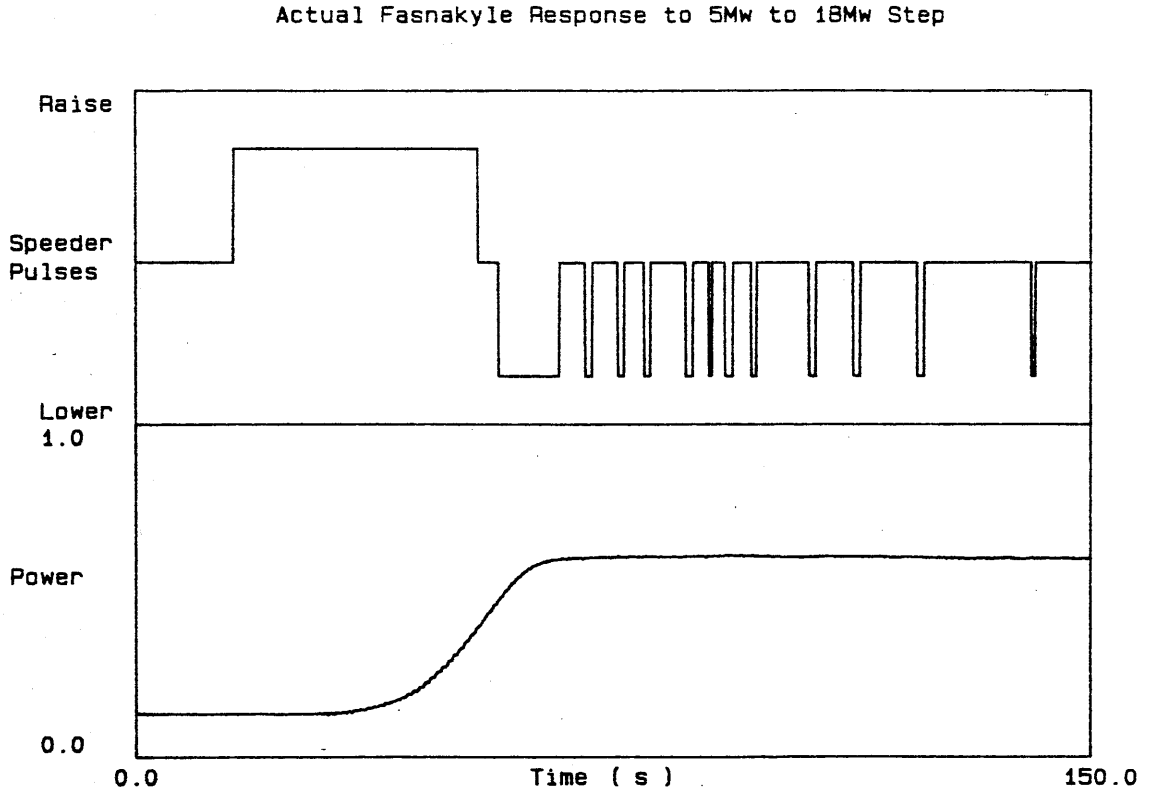


Fig. 8-8: Actual Response of No.1 Set at Fasnakyle to 5-18Mw Step

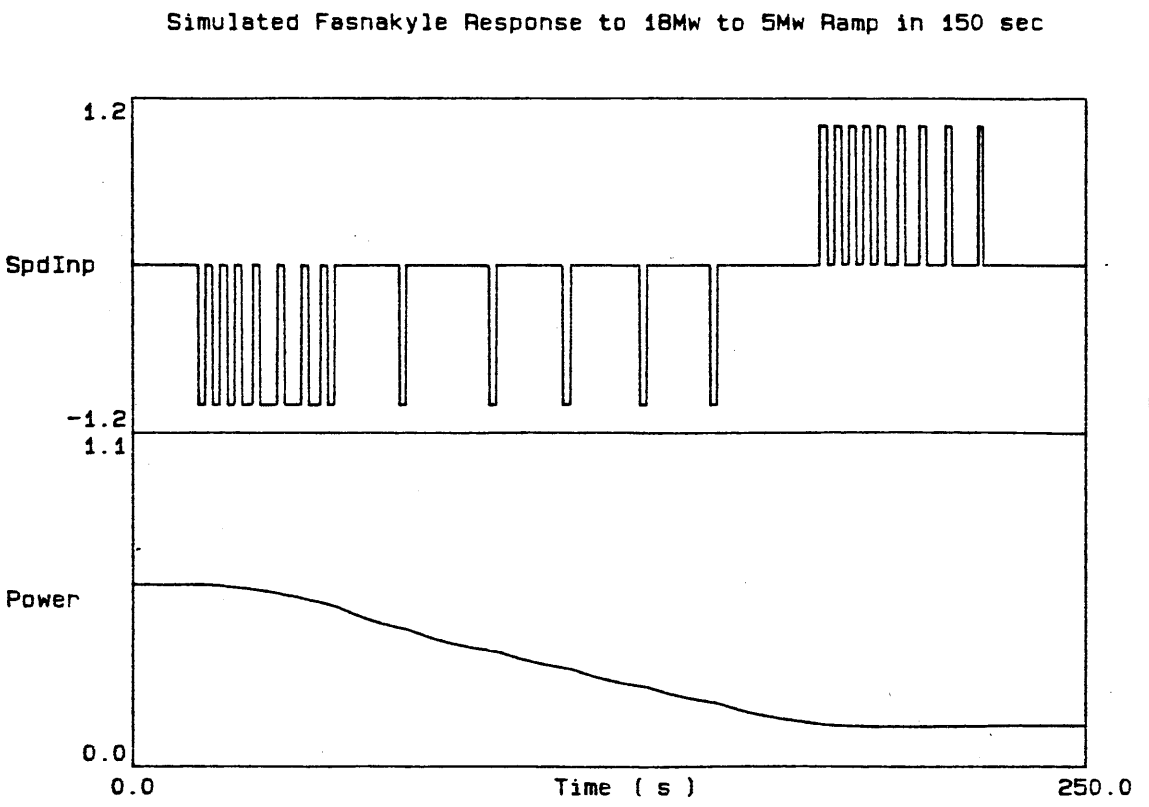


Fig. 8-9: Simulated Response of Fasnakyle to a 18-5Mw Ramp in 150sec.

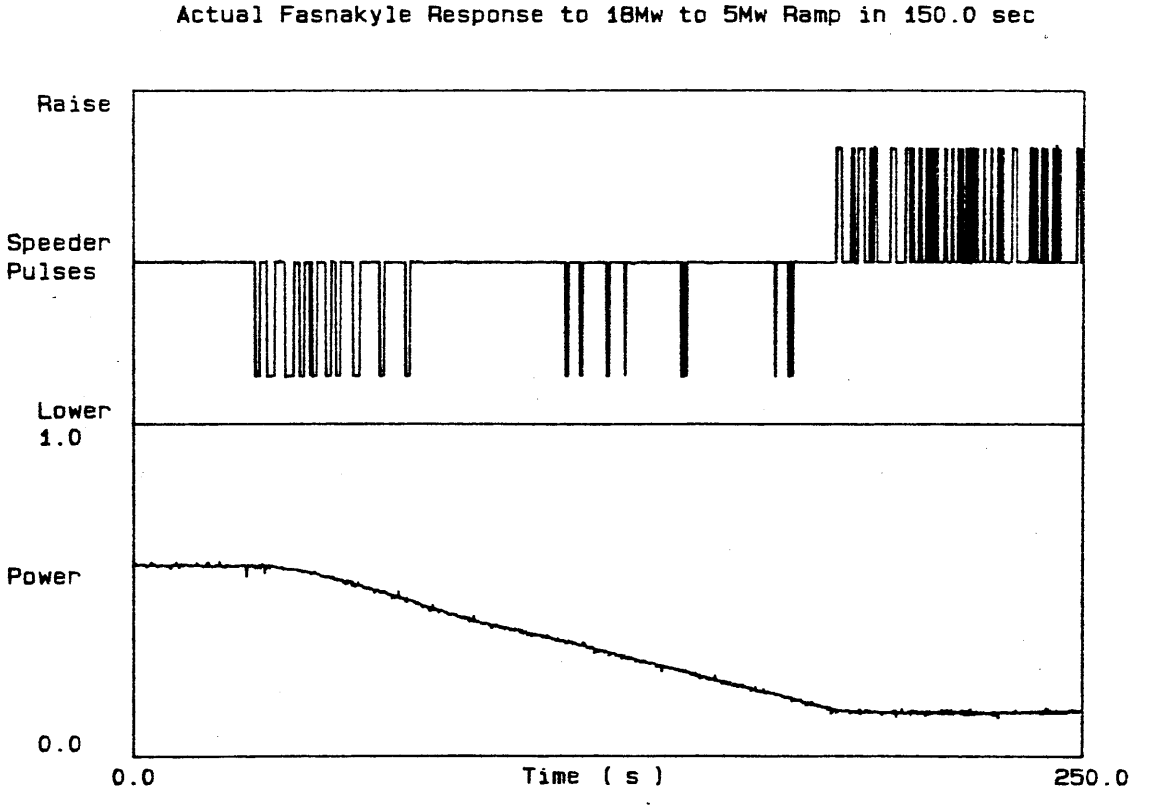


Fig. 8-10: Actual Response of No.1 set at Fasnakyle to a 18-5Mw ramp

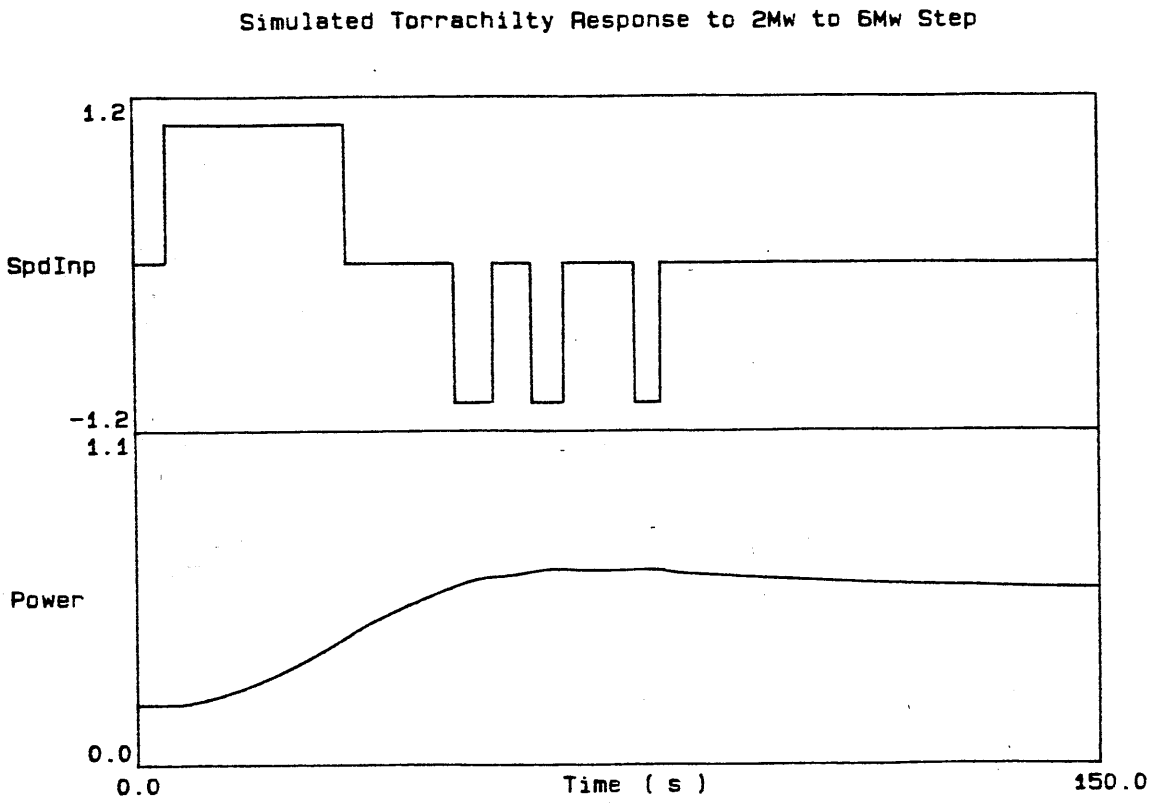


Fig. 8-11: Simulation of Torr Achilty 2-6Mw step.

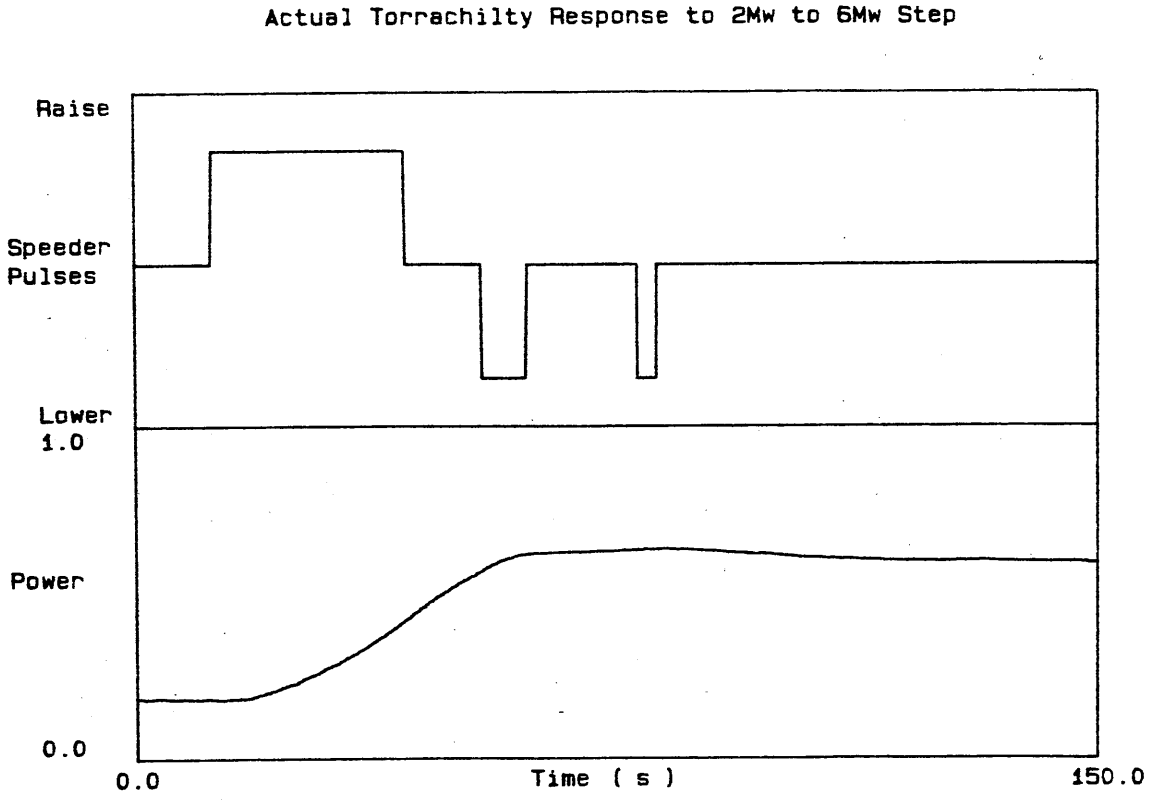


Fig. 8-12: Actual Response of No.2 Set at Torr Achilty to 2-6Mw Step

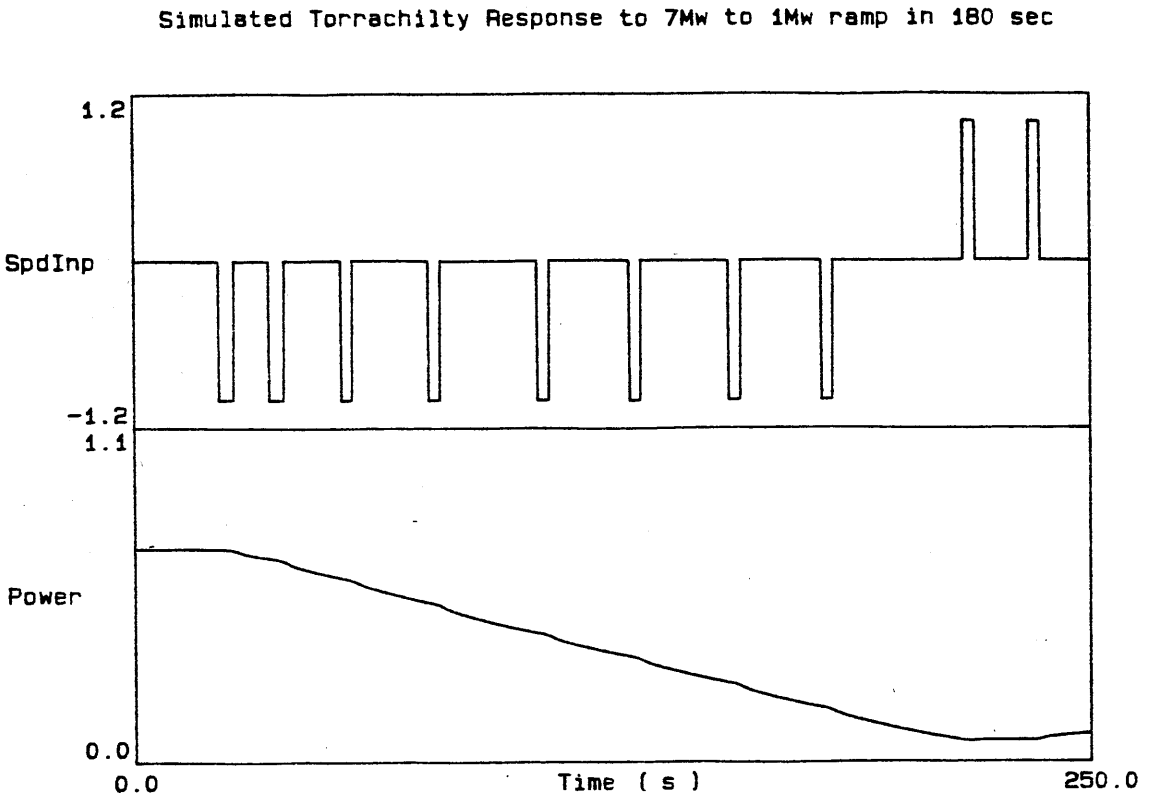


Fig. 8-13: Simulated Response of Torr Achilty to a 7-1Mw Ramp in 180sec.

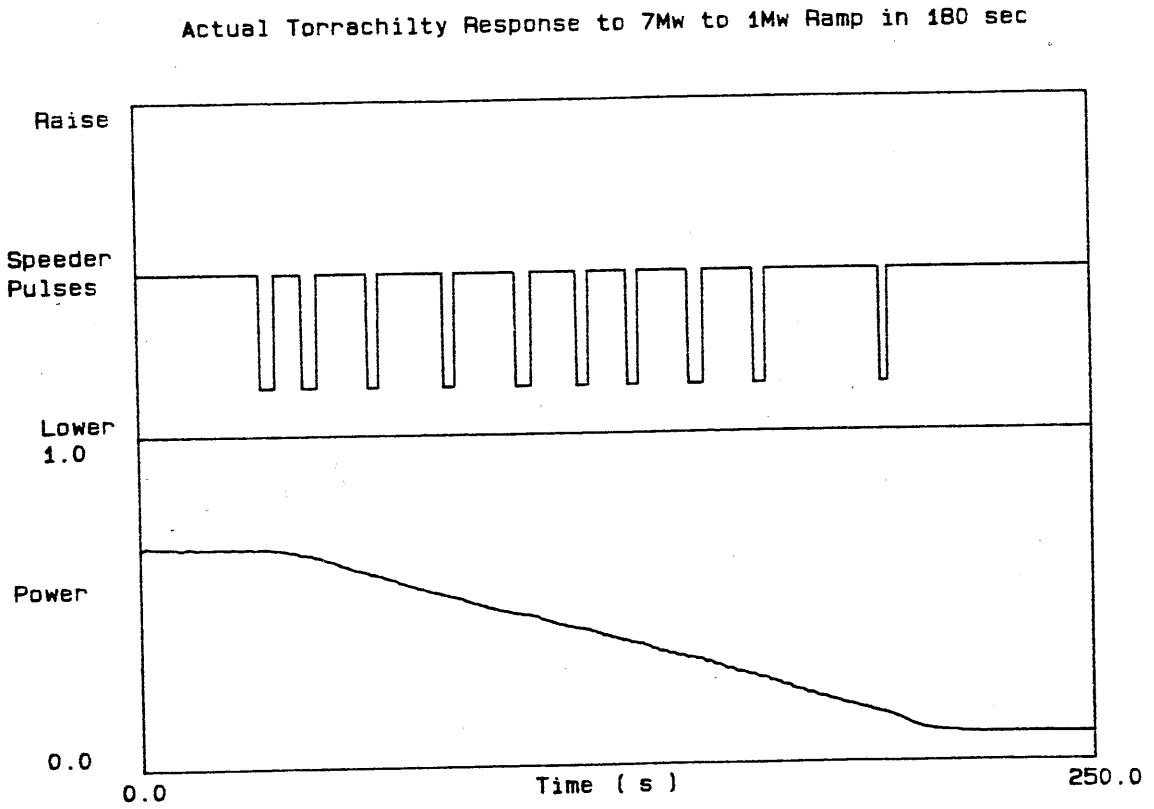


Fig. 8-14: Actual Response of No.2 set at Torr Achilty to a 7-1Mw ramp

8.7. Discussion and Conclusion

As can be seen by comparing the simulated turbine response to the actual responses, the model of a TD governed set was able to accurately predict the operation of a variety of different sets. The simulated behaviour of the controller was not identical to the actual controller action. This was due to two factors : The parameters of the simulated model of the governor were not completely correct, and the effects of noise. The simulation assumed that the grid frequency was constant. However, this is not the case, particularly during the ramp loading. The grid frequency variations introduced noise which the model cannot simulate.

The set point controller showed itself to be more than capable of dispatching the hourly load schedule produced by the optimisation, in both step changes and in controlled ramp loading.

9. DISCUSSION, FURTHER WORK AND CONCLUSION

9.1. Discussion of the Results of the Project

It has been demonstrated that intelligent scheduling of the highly constrained Tummel Valley scheme is possible by computer in real time. This has been made possible by decomposing the problem in time. The time scales derived for each of the sub-problems were based on a knowledge of the operation of the Tummel Valley scheme. Scheduling in this way increases the flexibility and efficiency of the optimisation process.

The scheduling was considered on three separate time scales :
(a) Strategic water management, which considers the annual planning of the long term operation of large reservoirs.
(b) Daily decoupling of the scheme, which decides the water available for generation from each station, taking into account all the operational constraints of the valley.
(c) Hourly scheduling of the available water, which allocates the available water according to a projected demand curve.
Each of these programs is a useful independent planning tool and combine to produce a complete optimisation system.

The strategic water management can be undertaken at any time, without reference to the hourly scheduling. The results suggest that tighter control of the storage of the reservoir will yield useful savings. Tighter control does, however, require that an accurate projected demand is known. Thus, more sophisticated methods must be used in long term generation planning. The fast computation time of the method allows engineers to interactively plan annual generation targets.

It is not possible to completely simulate the complex relationship between the needs of the national grid and the operation of the Tummel Valley scheme. The number of external influences in the assessment of financial benefit are vast. For example, the system would have to consider capital depreciation on plant, interest rates on loans, the current price of oil and coal, and, the more subjective costs such as environmental costs. The assessment of the benefits of the daily decoupling and the hourly scheduling procedures has heavily relied upon the expert knowledge of the Generation Engineers. The emphasis placed on each of the various benefits by the engineers could easily be incorrectly biased. The solution to this is the development of a more realistic method of representing economic costs within power systems.

The representation of the contribution of a sub-system to the generation required by the national grid must be a compromise. The national grid has an exact target to meet and, therefore, must know the contribution it can expect from each group. Similarly, each group must be given enough flexibility to maximise the benefit from its available resources. The generation demand for the Tummel Valley scheme was represented by a curve of the highest merit order cost of generation within the national grid. It can be seen from the results of both the daily decoupling and the hourly scheduling that this demand curve effectively controls the response of the optimisation system. For planning purposes, it would be easier to specify an exact generation target for all sub-systems of the national grid. Each sub-system would then use its available generation to match this exact target. However, in a

highly constrained sub-system, such as the Tummel Valley scheme, this type of forced generation would frequently violate constraints and lead to a sub-optimum solution. The Tummel Valley scheme could easily be combined with other unconstrained generation plant to form a larger group. An exact generation target could then be given to the larger group, with the unconstrained plant compensating for the limitations which must be placed on the generation of the Tummel Valley Scheme.

In smaller reservoirs, the water available for generation is closely tied to the immediate inflow. Better use can be made of this water if the amount of inflow is known in advance. It has been shown that it is feasible to produce useful inflow predictions based upon very limited hydrological and meteorological data collection. The use of inflow predictions can substantially increase the units generated and reduce the water spilled. The catchment area of Gaur is unique in its soil composition. The application of the methods developed here to other catchments would require some additional work. The introduction of RADAR assisted weather forecasting in Scotland would give a more accurate assessment of rainfall magnitude and distribution within a catchment. This more accurate assessment of rainfall characteristics, combined with the longer time horizons for weather predictions available from improved meteorlogical modelling, can only improve the accuracy of the inflow predictions.

A set point controller was studied as a method of dispatching the schedule of the optimiser. This set point controller can economically be added to existing Temporary Droop governed hydro turbines. The controller was extensively tested both on an

improved computer simulation and on three actual stations.

9.2. Further Work

Although the original aims of the project have been achieved, a number of avenues have opened which merit further study.

Recently, Expert Systems have been introduced to the field of operational research and control problems. Compilers for PROLOG, a rule database language, are now available which include the facilities for real number computation and real time interfacing. The use of expert systems in the optimal control of the Tummel Valley scheme could merit further study. The constraints of a system can easily be represented as a database of expert rules. These rules can be used in conjunction with the calculation of flows and storages to derive optimal schedules. The improvements in computational power of micro-processors may also allow this type of system to operate in real time. One of the main advantages of this type of system is that the rule base could be derived and maintained on-site by the Generation Engineers themselves.

The methods of inflow prediction could be extended to include snow melt data, which unfortunately was not recorded during the course of this project. The models have yet to be used in conjunction with data from a catchment area other than Gaur. Thus other logged data could be obtained from another catchment and the method previously described applied to it.

A fuller evaluation of the optimisation results could be achieved if a system were to be installed at Errochty Group

Control room. The system itself could then continually monitor and compare its operation to that of the Generation Engineers. The methods of optimisation which have been devised for the Tummel Valley scheme could easily be applied to some other group of interacting hydro power stations.

The set point controller could be expanded to include the other control functions required in a power station. There is at present a real need for some inexpensive, reliable station monitoring equipment in most power stations. The same equipment could be used for set point control, station communication, alarm monitoring and other control functions.

9.3. Conclusion

A prototype optimal control system has been demonstrated for the Tummel Valley scheme. The novel decomposition of the problem has resulted in fast computation times and the development of three separate planning tools for strategic water management, daily decoupling and hourly scheduling. The system could be installed as an automatic control system, or as an advisory system to the Generation Engineers. Either system would allow complete assessment of the operation of the valley.

The direct economic benefits of inflow prediction have been demonstrated.

The use of a set point controller has been demonstrated for fast or controlled load changes. This type of controller could easily allow stations to operate entirely automatic.

REFERENCES

1. NORTH OF SCOTLAND HYDRO ELECTRIC BOARD
   The Tummel Valley
   G.E.Findly & Co Dundee Scotland October 1981

2. TENEKETZIS D. SRIDHAR B.
   Application of dynamic team theory to the scheduling of reservoirs in the presence of random inflows
   IFAC 8th Triennial World Congress Kyoto Japan 1981 pp1435-1447

3. FLEMING G.
   Computer simulation techniques in Hydrology.
   Published 1975 by American Elsevier, USA

4. GRANT N.F.
   A microprocessor based controller applied to a hydro turbine.
   Ph.D. Thesis University of Glasgow 1980

5. ALLEY W.T.
   Hydro electric plant capability curves.
   IEEE Trans. PAS 96 No 3 May 1977 pp999-1004

6. NORTH OF SCOTLAND HYDRO ELECTRIC BOARD
   Annual Report 1985

7. THE ELECTRICITY COUNCIL
   Handbook of electricity suuply statistics.
   The Electricity Council. 1983

8. MASIELLO R.D.
   Computers in Power : A welcome Invader.
   IEEE SPECTRUM Feb 1985 p51

9. IKURA G. GROSS G.
   Efficient Large-scale Hydro system Scheduling with forced spill conditions.
   IEEE TRANS. PAS Vol 103 no 12 Dec 1984 pp3502-3514

10. LOWERY P.G.
Generating unit commitment by Dynamic programming.
IEEE TRANS. PAS Vol 85 no 5 May 1966 pp422 – 429

11. GAGNON C.R. BOLTON J.F.
Optimal Hydro scheduling at the Bonneville power administation.
IEEE TRANS. PAS Vol 97 no 3 May/June 1978 pp772 – 785

12. HICKS T. GAGNON J. JACOBY L. KOWALIK M.
Large scale, non-linear optimisation of energy capability for the pacific northwest hydro electric system.
IEEE Trans. PAS 93 1974 pp1604 – 1608

13. LAUER G. BERTSEKAS D. SANDELL P. POSBERGH J.
Solution of large scale optimal unit commitment problems.
IEEE Trans. PAS 101 No 1 Jan 1983 pp79 – 87

14. DAHLIN E.B. SHEN W.C.
Application of Dynamic programming to optimisation of HydroElectric/Steam power system operation.
PROC. IEE Vol 112 no 12 pp2255 – 2267

15. BELLMAN R.E. DREFUS S.E.
Applied Dynamic Programming.
PRINCETON UNIVERSITY PRESS Princeton, New Jersey 1962

16. VAN DER WAL J.
Stochastic Dynamic Programming.
Published 1981 by the Mathematical Centre, Amsterdam

17. STOREY C.
Dynamic optimisation.
System modelling and optimisation Ed. P. Nash
IEE Control Engineering Series pp36 – 46
Peter Peregrinus Ltd. 1981

18. LARSON R.E.
State Incremental Dynamic Programming
AMMERICAN ELSEVIER PUBLISHING COMPANY New York 1968

19. KAUFMANN A. CRUON R.
Dynamic Programming.
Mathematics in Science and Engineering Vol 37.
ACADEMIC PRESS New York 1967

20. CAMBELL L.H.
Introduction to Linear Programming.
Systems modelling and optimisation Ed P. Nash
IEE Control Series 16 pp77 – 91
Peter Peregrinus Ltd. 1981

21. CAMBELL L.H.
Decomposition in Linear Programiing.
System modelling and optimisation. Ed P. Nash
IEE Control Engineering Series 16 pp97 – 106
Peter Peregrinus Ltd. 1981

22. BAJPAI A. MUSTOE K. WALKER J.
Advanced Engineering Mathematics.
Published 1977 by J. Wiley & sons, Chichester

23. DREYFUS S.E. LAW A.M.
The Art and Theory of Dynamic Programming.
Mathematics in Science and Engineering Vol 130.
ACADEMIC PRESS New York 1977

24. ORCHARD-HAYS W.
Advanced Linear Programming Computing Techniques.
McGRAW-HILL BOOK COMPANY New York 1968

25. HIMMELBLAN D. M.
Applied Non-Linear Programming.
Published 1972 by McGraw-Hill

26. TOINT P.
Mathematical Programming.
System modelling and optimisation. Ed. P. Nash
IEE Control Engineering series 16 pp1-16
Peter Peregrinus Ltd. 1981

27. MAURRAS J. F. MACHADO J.
The Short-term management of HydroElectric Reserves.
MATHEMATICAL PROGRAMMING STUDY Vol 9 1978 pp176-184

28. ARVANITIDIS N.V. ROSING J.
Composite representation of a multireservoir Hydro electric power system.
IEEE TRANS. PAS Vol 89 no 2 Feb 1970 pp319-334

29. TYREN L.
Short range optimisation of a hydro-thermal system by a gradient method combined with linear programming.
3rd Power System computational Conference Rome 1969

30. HABIBOLLAHZADEH K.
Optimisation of hydro electric power systems.
Analytical techniques for energy planning. Proc. of the 1st symposium 1983 pp109-121

31. SAHA T.N. KHAPARDE S.A.
An application of a direct method to the optimal scheduling of a Hydrothermal system.
IEEE TRANS. PAS Vol 97 no 3 Mar 1978 pp997-1012

32. RAMAMOORTY M. RAO J.G.
Load Scheduling of Hydroelectric/Thermal generating systems using nonlinear programming techniques.
PROC IEE Vol 117 no 4 April 1970 pp794-798

33. MERLIN A. SANDRIN P.
A new method for unit commitment at ELECTRICITE DE FRANCE.
IEEE TRANS. PAS Vol 102 no 5 May 1983 pp1218-1234

34. BERTSEKAS B.LAUER F.SANDELL I.POSBERGH K.
Optimal short-term scheduling of large-scale power systems.
IEEE TRANS. AUTOMATIC CONTROL AC-28 no 1 Jan 1983

35. DILLON EDWIN KOCHS TAUD
Integer programming approach to the problem of optimal unit commitment with probalistic reserve determination.
IEEE TRANS. PAS Vol 97 no 6 Jun 1978 pp2154 - 2169

36. TAHA H.A.
Integer Programming : Theory, Applications, and Computations
ACADEMIC PRESS New York 1975

37. TURGEON A.
Optimal Unit commitment.
IEEE TRANS. AUTOMATIC CONTROL AC-22 no 2 pp223-227

38. WAKAMORI A.MASUI K.MORITA U. SUGIYAMA F.
Layered network model approach to optimal daily Hydro scheduling.
IEEE TRANS. PAS Vol 101 no 9 Sept 1982 pp3310 - 3325

39. HARBOE R.C. MOBASHERI F. YEH WN-G
Optimal policy for reservoir operation.
Proc ASCE J. of Hydr. DIV HY II 1970 pp2297 - 2314

40. LARSON R.E. KECKLER W.G.
Applications of Dynamic programming to the control of water resource systems.
AUTOMATICA VoL 5 1969 pp15-26

41. NASH P.
Optimal operation of canal reservoirs.
System modelling and optimisation. Ed P. Nash
IEE Control Engineers series 16 pp141 - 157
Peter Peregrinus Ltd. 1981

42. VIRAMONTES F.A. HAMILTON H.B.
Optimal long range Hydro scheduling in the integrated power system.
IEEE TRANS. PAS Vol 97 no 1 Jan 1978 pp292 - 304

43. PANG C.K. CHEN H.C.
Optimal short-term Thermal unit commitment.
IEEE TRANS. PAS Vol 95 no 4 Jul 1976 pp1336 - 1345

44. TURGEON A.
Optimal short-term Hydro scheduling from the principle of progressive optimality.
WATER RESOURCES RESEARCH Vol 17 no 3 1981 pp481 - 500

45. NANDA J. BIJWE R.R.
Optimal hydro thermal scheduling with cascaded plant using progressive optimallity.
IEEE Trans PAS Vol 100 No 4 Apr 1981 pp2093 - 2113

46. TURGEON A.
Optimal operation of multireservoir power systems with stochastic inflows.
Water Resources Research Vol 16 No 2 pp275-283 Apr 1980

47. TROTT W.J YEH W.W.G.
Optimisation of multiple reservoir systems.
PROC ASCE JOURN. OF THE HYDRAULICS DIV.
HY10 Oct 1973 pp1865 - 1879

48. LARSON R.E. KORSAK A.J.
A Dynamic programming successive approximation technique with convergence proofs.
AUTOMATICA Vol 6 1970 pp245-252

49. PEREIRA M.V.F. PINTO L.M.V.G.
Application of decomposition techniques to the mid- and short-term scheduling of Hydrothermal systems.
IEEE TRANS. PAS Vol 102 no 11 Nov 1983 pp3611 - 3625

50. LE L. DAY F. COOPER J. GIBBONS M.
A global optimisation method for scheduling Thermal generation, Hydro generation, and economic purchases.
IEEE TRANS. PAS Vol 102 no 7 Jul 1983 pp1986 – 1994

51. HAPP H.H. JOHNSON R.C. WRIGHT W.J.
Large scale Hydro-Thermal unit commitment - method and results.
IEEE TRANS. PAS Vol 89 1970 pp1373 – 1388

52. SOARES S. LYRA B. TAVARES T.
Optimal generation scheduling of hydro thermal power systems
IEEE Trans PAS 99 No 3 May 1980 pp1107 – 1119

53. SHAW J.J. GENDRON R.F. BERTSEKAS D.P.
Optimal scheduling of large Hydrothermal systems.
IEEE TRANS. PAS Vol 104 no 2 Feb 1985 pp286 – 295

54. LEKANE T.M.
Short term scheduling of multi reservoir hydro electric power systems.
Proc. 7th Power system computational conference Switz 1981 pp375-382

55. DONDI P.H. SCHAFFER G.
Simulation and optimisation of a series of hydro stations
Water Power and Dam Construction Nov. 1983 pp20-23

56. AGARWAL S.K.
Optimal scheduling of hydrothermal systems.
Proc. IEE vol 119 No 2 Jan 1972 pp169

57. BRADLEY RUSSELL C.
An optimal policy for operating a multi purpose reservoir.
Operational Research Vol 20 No 6 1972 pp1181-1189

58. ASKEW A.J.
Optimum reservoir operating policies and the imposition of a reliability constraint.
WATER RESOURCES RESEARCH Vol 10 no 1 Feb 1974 pp51-76

59. NETO S.PEREIRA M. KELMAN F.
A risk constrained stochastic Dynamic programming approach to the operational planning of Hydrothermal
IEEE TRANS. PAS Vol 104 no 2 Feb 1985 pp273-281

60. SACHDEVA S.S.
Biblograrphy on optimal reservoir draw down for the Hydroelectric-Thermal power system operation.
IEEE TRANS. PAS Vol 101 no 6 June 1982 pp1487-1499

61. HALLIBURTON T.S. SIRISEN H.R.
Developement of a stochastic optimisation for multireservoir scheduling.
IEEE TRANS. AUTOMATIC CONTROL AC-29 no 1 pp82-84

62. WOOD WOLLENBERG
Power Generation : Operation and Control.
J. Wiley & Sons, New York 1984

63. MEYER P.J.
Introductory Probability and Statistical Applications
ADDISON-WESLEY Pub. Co. London 1980

64. CRAWFORD N.H.
Reducing spill at Hydro projects.
WATER POWER AND DAM CONSTRUCTION Vol 36 Nov 1984 pp51-53

65. FERGUSON R.I.
Magnitude and modelling of snow melt runoff in the Cairngorm mountains Scotland.
Hydrological Science Journal No 29.1.3 1984 pp49-62

66. METEORLOGICAL OFFICE
Radar Weather Research
Meteorlogical Office London Road Bracknell Berkshire 1983

67. GRAY D.M.
Handbook of the principles of Hydrology.
Published 1973 by Macmiilan

68. LINSEY R.K. KOHLER M.A. PAULHUS J.L.
Hydrology for Engineers.
McGRAW-HILL BOOK COMPANY New York 1958

69. RAUDKIVI A.J.
Hydrology an advanced introduction.
Published 1979 by Pergamon

70. RODDA J.C. (ED)
Facets of Hydrology.
JOHN WILEY & SONS Chichester, England 1985

71. QUINLAN J.R.
Discovering rules by induction from large collections of examples.
Expert system in the micro-electronic age. ED. D. Michie
Edinburgh University Press 1979.

72. KISIEL C.C.
Time series analysis of Hydrological data
ADVANCES IN HYDROSCIENCE Vol 5 1969

73. LOUCKS D.P. STEDINGER J.R. HAITH D.A.
Water resources systems planning and analysis.
Published 1981 by Prentice-Hall, USA

74. BOX J. JENKINS J.
Time series analysis and forcasting.
Published 1976 by Holden-Day, San Fansisco

75. PETERKA V.
A square root filter for real time multivariate regression.
KYBERNETIKA Vol 11 no 1 1975

76. FINDLAY D.G.E.
Microprocessor governors for hydro turbine generators.
Ph.D. Thesis University of Glasgow 1980

77. THOMPSON E.C.
A digital simulation of a boiler and turbine in conjunction with a model power system.
Ph.D. Thesis University of Glasgow 1976

78. AITKEN K.H.
Developement of a simulation language in conjunction with hydro turbine modelling
Ph.D. Thesis University of Glasgow 1982

OPTIMAL CONTROL OF A GROUP OF INTERACTING HYDRO POWER STATIONS

PROGRAM LISTINGS

by

J. Clink, B.Sc.

December 1986

INDEX

1. STRATEGIC WATER MANAGEMENT

DP2.PAS This program executes the standard stochastic dynamic programming optimisation on Loch Ericht.

DX.PAS This program performs the modified optimisation of Loch Ericht.

ERICSIM.PAS This program simulates the operation of a target storage plan for actual inflow data.

2. INFLOW PREDICTION

CONVERT.PAS Converts CPM format data to MSDOS files
PROC87.PAS Calculates inflow from logged data
FILT87.PAS Filters Inflow to reduce noise
PLOT87.PAS Plot data on HP plotter

PLOTPAP.PAS Simulate selective models and plot

3. DAILY DECOUPLING

TVDAILY.PAS Performs non-linear programming on daily problem

4. HOURLY SCHEDULING

TVHOURLY.PAS Uses Dynamic Programming to schedule the power output of each station hourly.

5. SET POINT CONTROLLER

RTSIM.PAS Real time simulation of TD governed Hydro Turbine
SLOYSIM.PAS Simulation of set point control at Sloy
FASKSIM.PAS Simulation of set point control at Fasnakyle
TORRSIM.PAS Simulation of set point control at Torr Ackilty

DP2.PAS

This program uses standard stochastic dynamic programming to optimise the strategic storage of Loch Ericht. The program takes in excess of 26 hour for one iteration.

The function Ecost (Line 31-65) calculates the expected cost of going from one node to another.

The main procedure Optimise (Line 67-119) uses this expected cost to find the optimum strategy for the year.

```
  1: PROGRAM LevelOptimisation ;
  2: {$F+}
  3: CONST
  4:   Ks : Real = 10.0 ;
  5:   Kf : Real = 100.00 ;
  6:   Step : Real = 30.87 ;
  7:   MaxLevel : Real = 5910.0 ;
  8:   MaxOutFlow : Real = 740.88 ;
  9: TYPE
 10:   Str80 = STRING[80];
 11:   CharSet = SET OF Char ;
 12:   FileName = STRING[20] ;
 13:   Node = RECORD
 14:            Previous : Integer ;
 15:            Cost : Real ;
 16:          END;
 17:   Dist = ARRAY[0..63] OF Real ;
 18:
 19: VAR
 20:   Nodes : ARRAY[0..200,0..52] OF ^Node ;
 21:   Current : Node ;
 22:   P : Dist ;
 23:   PFile : FILE OF Dist ;
 24:   StartFile:Text;
 25:   Result:FILE OF Node;
 26:
 27:   L,Wk,Next,Start,Stop : Integer ;
 28:   Tcost,DesiredStore,DesiredLevel : Real ;
 29:
 30:
 31:   FUNCTION Ecost(Cnode, Nnode, Time : Integer ):Real ;
 32:
 33:   VAR
 34:     Pfail, Espill, Cost, Clevel, Nlevel,
 35:     Inflow, Outflow, Spill : Real ;
 36:     I : Integer ;
 37:
 38:   BEGIN
 39:     GotoXY(1,1);Write(Cnode:4,Nnode:4,Time:4);
 40:     Pfail := 0.0 ;
 41:     Espill := 0.0 ;
 42:     Cost := 0.0 ;
 43:     FOR I := 0 TO 63 DO
 44:       BEGIN
 45:         Inflow := (I+0.5)*Step ;
 46:         Clevel := Cnode * Step ;
 47:         Nlevel := Nnode * Step ;
 48:         OutFlow := Clevel + Inflow - Nlevel ;
 49:         IF OutFlow < 0 THEN
 50:           BEGIN
 51:             OutFlow := 0 ;
 52:             Pfail := Pfail + P[I] ;
 53:           END;
 54:         IF OutFlow > MaxOutFlow THEN
 55:           BEGIN
 56:             OutFlow := MaxOutFlow ;
 57:             Pfail := Pfail + P[I] ;
 58:           END;
 59:         Spill := Clevel + Inflow - OutFlow - MaxLevel ;
 60:         IF Spill > 0 THEN
```

```
61:          Espill := Espill + P[I] * Spill ;
62:        END;
63:      Cost := Cost + Kf * Pfail + Ks * Espill ;
64:      Ecost := Cost ;
65:    END ;
66:
67:    PROCEDURE Optimise ;
68:    BEGIN
69:      Assign(Pfile,'Dist.p');
70:      Reset(Pfile);
71:      Assign(StartFile,'Start.dat');
72:      Reset(StartFile);
73:      FOR L := 0 TO 200 DO
74:        BEGIN
75:          FOR Wk:=1 TO 52 DO Nodes[L,Wk]^.Previous:=-1;
76:          WITH Nodes[L,0]^ DO
77:            Readln(StartFile,Cost,Previous);
78:        END;
79:      Close(StartFile);
80:      FOR Wk := 0 TO 51 DO
81:        BEGIN
82:          Read(Pfile,P);
83:        FOR L := 0 TO 200 DO
84:          BEGIN
85:            Current := Nodes[L,Wk]^ ;
86:            IF Current.Previous <> -1 THEN
87:              BEGIN
88:                Start := L - 20 ;
89:                Stop := L + 20 ;
90:                IF Start < 0 THEN Start := 0 ;
91:                IF Stop > 200 THEN Stop := 200 ;
92:                FOR Next := Start TO Stop DO
93:                  BEGIN
94:                    Tcost := Current.cost + Ecost(L,Next,Wk) ;
95:                    IF (Tcost < Nodes[Next,Wk+1]^.Cost) OR
96:                       (Nodes[Next,Wk+1]^.Previous = -1) THEN
97:                      WITH Nodes[Next,Wk+1]^ DO
98:                        BEGIN
99:                          Cost := Tcost ;
100:                         Previous := L ;
101:                       END;
102:                 END;
103:             END;
104:         END;
105:       END;
106:     Assign(StartFile,'Start.dat');
107:     ReWrite(StartFile);
108:     FOR L:=0 TO 200 DO
109:       WITH Nodes[L,52]^ DO
110:         WriteLn(StartFile,Cost:15:5,' ',Previous:5);
111:     Close(startFile);
112:     Assign(Result,'Results.dat');
113:     Rewrite(Result);
114:     FOR Wk:=0 TO 52 DO
115:       FOR L:=0 TO 200 DO
116:         Write(Result,Nodes[l,wk]^);
117:     Close(Pfile);
118:     Close(Result);
119:   END;
120:
```

```
121:
122: BEGIN
123:   (*****  Create node tree onto heap  ************************************)
124:
125:   FOR L := 0 TO 200 DO
126:     FOR Wk := 0 TO 52 DO
127:       New(Nodes[L,Wk]);
128:   ClrScr;
129:   GotoXY(1,12);
130:   Writeln('                        THIS MACHINE IS IN USE !!!!');
131:   Writeln('                  The experiment is expected to run overnight.');
132:   Writeln('                              Jim Clink');
133:   Optimise;
134: END.
```

DX.PAS

This program uses a modified stochastic dynamic programming procedure to optimise the strategic changes in storage of Loch Ericht.

The function Ecost (Line 43-88) calculates the expected cost of going from one node to another.

The main procedure Optimise (Line 728-804) uses this expected cost to find the optimum strategy of storage changes for the year.

The other sections of the program simply allow data to be set up and saved.

```
 1: PROGRAM LevelOptimisation ;
 2: {$R+}
 3: CONST
 4:   Ko : Real = 20.00 ;
 5:   Ku : Real = 20.00 ;
 6:   Kb : Real = 100.00 ;
 7:   Kf : Real = 100.00 ;
 8:   Step : Real = 30.87 ;
 9:   MaxLevel : Real = 5910.0 ;
10:   MaxOutFlow : Real = 740.80 ;
11:   DataSetSize = 856 ;
12:
13: TYPE
14:   Str80 = STRING[80];
15:   CharSet = SET OF Char ;
16:   DataSet = ARRAY[1..DataSetSize] OF Byte ;
17:   DataRecord = ARRAY[1..1000] OF Byte ;
18:   FileName = STRING[20] ;
19:   Node = RECORD
20:            Previous : Integer ;
21:            Cost : Real ;
22:          END;
23:   Dist = ARRAY[0..63] OF Real ;
24:
25: VAR
26:   Nodes : ARRAY[0..200,0..52] OF ^Node ;
27:   Current : Node ;
28:   DesiredOutflow : ARRAY[0..52] OF Real ;
29:   Solution : ARRAY[0..52] OF Real ;
30:   ExpectedInflow, ExpectedOutFlow : ARRAY[0..51] OF Real ;
31:   Sol : ARRAY[0..52] OF Integer;
32:   Prob,P : Dist ;
33:   Preal : FILE OF Dist ;
34:   DataFile : FILE OF DataRecord ;
35:   Data : DataSet ABSOLUTE DesiredOutflow ;
36:   DataInFileName,
37:   DataOutFileName : FileName ;
38:   Ec : ARRAY[-10..10] OF Real;
39:   L,WK,Next,Start,Stop : Integer ;
40:   Tcost,DesiredStore,DesiredLevel : Real ;
41:   Pr : Boolean ;
42:
43:   FUNCTION Ecost(Time,Cx : Integer ):Real ;
44:
45:   VAR
46:     Pfail, Espill, Cost, Dx ,
47:     Inflow, Outflow, Spill : Real ;
48:     I,J : Integer ;
49:
50:   BEGIN
51:     Pfail := 0.0 ;
52:     Espill := 0.0 ;
53:     Cost := 0.0 ;
54:     Dx:=Cx*Step;
55:     IF Pr THEN
56:      BEGIN
57:     Write(Cx:4,time:4);
58:     END;
59:     FOR I := 0 TO 63 DO
60:       BEGIN
```

```
        Inflow := Step*(I+0.5) ;
        OutFlow := Inflow - Dx ;
        IF OutFlow < 0 THEN
          BEGIN
            OutFlow := 0 ;
            Pfail := Pfail + P[I] ;
          END
        ELSE
          IF OutFlow > MaxOutFlow THEN
            BEGIN
              OutFlow := MaxOutFlow ;
              Pfail := Pfail + P[I] ;
            END
          ELSE
        IF OutFlow < (DesiredOutFlow[Time]-(4*Step)) THEN
          Cost := Cost + P[I] * Ku * (OutFlow)
          ELSE
        IF OutFlow > (DesiredOutFlow[Time]+(4*Step)) THEN
          Cost := Cost + P[I] * Ko * (OutFlow)
          ELSE
          Cost := Cost + P[I] * Kb * (OutFlow);
{        If Pr THen Writeln('In ',Inflow:8:1,' Out ',outflow:8:1,' P= ',P[i]:10:6,' Cost ',Cost:8:1);
 }     END;
    Cost := Cost - Kf * Pfail ;
    IF Pr THEN Writeln('Cost ',cost:8:1,' Pfail ',pfail:10:6,Espill:10:2);

    Ecost := Cost ;
 END ;


PROCEDURE Enhanced;
VAR
  regs:RECORD CASE integer OF
        1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
        2:(al,ah,bl,bh,cl,ch,dl,dh:byte);
       END;
BEGIN
    WITH regs DO
      ax:=$0010;
    intr($10,regs);
    WITH regs DO
     BEGIN
      ah:=$11;
      al:=$22;
      bl:=2;
     END;
    intr($10,regs);
END;

  PROCEDURE eplot(x,y,colour:integer);
 VAR
  i,off:integer;
 BEGIN
    port[$3ce]:=5;
    port[$3cf]:=2;
    off:=(y*80)+(x DIV 8);
    port[$3ce]:=8;
    port[$3cf]:=1 SHL (7-(x MOD 8));
    i:=mem[$A000:off];
    mem[$a000:off]:=colour;
```

```
121:     port[$3ce]:=5;
122:     port[$3cf]:=0;
123:     port[$3ce]:=8;
124:     port[$3cf]:=$ff;
125:  END;
126:
127:  PROCEDURE Edraw(x1,y1,x2,y2,colour:integer);
128:  {this procedure rewritten by david loomes}
129:  LABEL 999; {exit point if zero length line}
130:  VAR  x,y,deltax,deltay,dx,dy,run : integer;
131:
132:  PROCEDURE dp(x,y:INTEGER);
133:  VAR
134:   i,off:integer;
135:  BEGIN
136:     off:=(y*80)+(x DIV 8);
137:     port[$3ce]:=8;
138:     port[$3cf]:=1 SHL (7-(x MOD 8));
139:     i:=mem[$A000:off];
140:     mem[$a000:off]:=colour;
141:  END;
142:
143:     BEGIN
144:       port[$3ce]:=5;
145:       port[$3cf]:=2;
146:       IF (x1=x2) AND (y1=y2)
147:       THEN  GOTO 999;
148:
149:       deltay := abs(y1-y2);
150:       deltax := abs(x1-x2);
151:       IF y1>y2 THEN dy:=-1 ELSE dy:=1;
152:       IF x1>x2 THEN dx:=-1 ELSE dx:=1;
153:       x := x1;
154:       y := y1;
155:       IF deltax > deltay (increment along x axis)
156:       THEN  BEGIN
157:             run := deltax SHR 1;
158:             REPEAT
159:              dp(x,y);
160:              x := x+dx;
161:              run := run - deltay;
162:              IF run < 0
163:              THEN  BEGIN
164:                    run := run +deltax;
165:                    y := y+dy
166:                    END;
167:             UNTIL x=x2
168:             END
169:       ELSE  BEGIN
170:             run := deltay SHR 1;
171:             REPEAT
172:              dp(x,y);
173:              y := y+dy;
174:              run := run - deltax;
175:              IF run < 0
176:              THEN  BEGIN
177:                    run := run +deltay;
178:                    x := x+dx
179:                    END;
180:             UNTIL y=y2
```

```
181:              END;
182: 999:  dp(x2,y2);
183:  port[$3ce]:=5;
184:  port[$3cf]:=0;
185:  port[$3ce]:=8;
186:  port[$3cf]:=$ff;
187:  END;
188:
189:
190:   PROCEDURE LoadData;
191:
192:   (*****************************************************************************)
193:   (*  This procedure loads start data from file.                               *)
194:   (*****************************************************************************)
195:
196:   VAR
197:     PP: RECORD CASE Integer OF
198:           1: (P1000: DataRecord);
199:           2: (Parmz: DataSet);
200:         END;
201:
202:   BEGIN
203:      Assign(DataFile,DataInFileName);
204:
205:     {$I-}
206:
207:     Reset(DataFile);
208:
209:     {$I+}
210:
211:     IF IOResult <> 0 THEN
212:       BEGIN
213:         WriteLn(DataInFileName, ' not found !');
214:         Delay(500);
215:       END
216:     ELSE
217:       BEGIN
218:         Read(DataFile,PP.P1000);
219:         Data:=PP.Parmz;
220:         Close(DataFile);
221:       END;
222:
223:   (***** END procedure loaddata  ********************************************)
224:
225:   END;
226:
227:
228:   PROCEDURE SaveData;
229:
230:   (*****************************************************************************)
231:   (*  This procedure saves start data to file.                                 *)
232:   (*****************************************************************************)
233:
234:   VAR
235:     Command : CHAR;
236:     PP: RECORD CASE Integer OF
237:           1: (P1000: DataRecord);
238:           2: (Parmz: DataSet);
239:         END;
240:
```

```
241:  BEGIN
242:    Assign(DataFile,DataOutFileName);
243:
244:    {$I-}
245:
246:    Reset(DataFile);
247:
248:    {$I+}
249:
250:    IF IOResult <> 0 THEN
251:      BEGIN
252:        WriteLn('New File ',DataOutFileName,' !');
253:        Delay(500);
254:        Command := 'Y';
255:      END
256:    ELSE
257:      BEGIN
258:        Close(DataFile);
259:        WriteLn(' Overwrite (DESTROY) old ', DataOutFileName);
260:        Read(Kbd,Command);
261:        Command := UpCase(Command);
262:      END;
263:    IF Command = 'Y' THEN
264:      BEGIN
265:        Assign(DataFile, DataOutFileName);
266:
267:        {$I-}
268:
269:        Rewrite(DataFile);
270:
271:        {$I+}
272:
273:        IF IOResult = 0 THEN
274:          BEGIN
275:            PP.Parmz := Data;
276:            Write(DataFile,PP.P1000);
277:            Close(DataFile);
278:          END
279:        ELSE
280:          WriteLn(' Unable to open file ', DataOutFileName);
281:      END;
282:    Delay(500);
283:
284:  (***** END procedure Savedata  ******************************************)
285:
286:  END;
287:
288:
289:  PROCEDURE Title;
290:
291:  BEGIN
292:    ClrScr;
293:    NormVideo;
294:    WriteLn('          Loch Ericht Level Optimisation Program Ver. 1.0   ');
295:    WriteLn;
296:    LowVideo;
297:  END;
298:
299:
300:  FUNCTION UpcaseStr(S : Str80) : Str80;
```

```
301:
302:  (*****************************************************************************)
303:  (*  This Function returns a string which contains the upper case string  *)
304:  (*  of the parameter.                                                        *)
305:  (*****************************************************************************)
306:
307:  VAR
308:    P : Integer;
309:  BEGIN
310:    FOR P := 1 TO Length(S) DO
311:      S[P] := UpCase(S[P]);
312:    UpcaseStr := S;
313:  END;
314:
315:
316:  FUNCTION ConstStr(C : Char; N : Integer) : Str80;
317:
318:  (*****************************************************************************)
319:  (*  ConstStr returns a string with N characters of value C                  *)
320:  (*****************************************************************************)
321:
322:  VAR
323:    S : STRING[80];
324:  BEGIN
325:    IF N < 0 THEN
326:      N := 0;
327:    S[0] := Chr(N);
328:    FillChar(S[1], N, C);
329:    ConstStr := S;
330:  END;
331:
332:
333:  PROCEDURE Beep;
334:
335:  (*****************************************************************************)
336:  (*  Beep sounds the terminal bell or beeper                                  *)
337:  (*****************************************************************************)
338:
339:  BEGIN
340:    Write(^G);
341:  END;
342:
343:
344:  PROCEDURE InputStr(VAR Ssss;
345:                     L, X, Y : Integer;
346:                     Term : CharSet;
347:                     VAR TC : Char;
348:                     Ins : Boolean);
349:
350:  (*****************************************************************************)
351:  (*  This procedure allow the reading and editing of any string.             *)
352:  (*****************************************************************************)
353:
354:  CONST
355:    UnderScore = '_';
356:  VAR
357:    S : STRING[255] ABSOLUTE Ssss;
358:    P : Integer;
359:    Ch : Char;
360:  BEGIN
```

```
361:    GoToXY(X, Y); Write(S, ConstStr(UnderScore, L-Length(S)));
362:    P := 0;
363:    REPEAT
364:      GoToXY(X+P, Y); Read(Kbd, Ch);
365:      CASE Ch OF
366:
367:        (*****  The Following is only valid for the IBM PC  ****************)
368:
369:        #27 : IF KeyPressed THEN
370:                BEGIN
371:                  Read(Kbd, Ch);
372:                  CASE Ch OF
373:                    #75 : IF P > 0 THEN
374:                            P := P-1
375:                          ELSE Beep;
376:                    #77 : IF P < Length(S) THEN
377:                            P := P+1
378:                          ELSE Beep;
379:                    #71 : P := 0;
380:                    #79 : P := Length(S);
381:                    #83 : IF P < Length(S) THEN
382:                            BEGIN
383:                              Delete(S, P+1, 1);
384:                              Write(Copy(S, P+1, L), UnderScore);
385:                            END;
386:                    #72 : Ch := ^F;
387:                    #80 : Ch := ^A;
388:                    #73 : Ch := ^E;
389:                    #81 : Ch := ^X;
390:                    #82 : Ch := ^V;
391:                  ELSE Beep;
392:                  END;
393:                END;
394:        #32..#126 : IF (Ins) OR (P = Length(S)) THEN
395:                      BEGIN
396:                        IF P < L THEN
397:                          BEGIN
398:                            IF Length(S) = L THEN
399:                              Delete(S, L, 1);
400:                            P := P+1;
401:                            Insert(Ch, S, P);
402:                            Write(Copy(S, P, L));
403:                          END
404:                        ELSE Beep;
405:                      END
406:                    ELSE
407:                      BEGIN
408:                        IF P < L THEN P := P+1;
409:                        S[P] := Ch;
410:                        Write(Copy(S, P, L));
411:                      END;
412:        ^S : IF P > 0 THEN
413:               P := P-1
414:             ELSE Beep;
415:        ^D : IF P < Length(S) THEN
416:               P := P+1
417:             ELSE Beep;
418:        ^G : IF P < Length(S) THEN
419:               BEGIN
420:                 Delete(S, P+1, 1);
```

```
421:                Write(Copy(S, P+1, L), UnderScore);
422:              END;
423:        ^H, #127 : IF P ) 0 THEN
424:                     BEGIN
425:                       Delete(S, P, 1);
426:                       Write(^H, Copy(S, P, L), UnderScore);
427:                       P := P-1;
428:                     END
429:                   ELSE Beep;
430:        ^Y : BEGIN
431:               Write(ConstStr(UnderScore, Length(S)-P));
432:               Delete(S, P+1, L);
433:             END;
434:       ELSE
435:         IF NOT(Ch IN Term) THEN Beep;
436:       END;                    {of case}
437:     UNTIL Ch IN Term;
438:     P := Length(S);
439:     GoToXY(X+P, Y);
440:     Write(''':L-P);
441:     TC := Ch;
442:
443:   (*****  END procedure InputStr  ******************************************)
444:
445:   END;
446:
447:   PROCEDURE InputReal(VAR A:Real;
448:                           L,X,Y:INTEGER;
449:                           Term:CharSet;
450:                           VAR TC:CHAR;
451:                           Ins:Boolean);
452:   VAR
453:    Rstr : STRING[10];
454:    Temp : Real;
455:    Test : INTEGER;
456:
457:   BEGIN
458:     Str(A:10:2,Rstr);
459:     InputStr(Rstr,L,X,Y,Term,TC,Ins);
460:     Test:=1;
461:     WHILE Test<=Length(Rstr) DO
462:       IF Rstr[Test]=' ' THEN
463:         Delete(Rstr,Test,1)
464:       ELSE
465:         Test:=Test+1;
466:     Val(Rstr,Temp,Test);
467:     IF Test=0 THEN
468:       BEGIN
469:         A:=Temp;
470:       END;
471:     GotoXY(X,Y);
472:     Write(a:10:2);
473:   END;
474:
475:   PROCEDURE InputInt(VAR A:INTEGER;
476:                           L,X,Y:INTEGER;
477:                           Term:CharSet;
478:                           VAR TC:CHAR;
479:                           Ins:Boolean);
480:   VAR
```

```
481:    Rstr : STRING[10];
482:    Temp : INTEGER;
483:    Test : INTEGER;
484:
485:  BEGIN
486:    Str(A:10,Rstr);
487:    InputStr(Rstr,L,X,Y,Term,TC,Ins);
488:    Test:=1;
489:    WHILE Test<=Length(Rstr) DO
490:      IF Rstr[Test]=' ' THEN
491:        Delete(Rstr,Test,1)
492:      ELSE
493:        Test:=Test+1;
494:    Val(Rstr,Temp,Test);
495:    IF Test=0 THEN
496:      BEGIN
497:        A:=Temp;
498:      END;
499:    GotoXY(X,Y);
500:    Write(a:10);
501:  END;
502:
503:
504:  PROCEDURE Mouse(VAR M1, M2, M3, M4 : Integer);
505:
506:  (*****************************************************************************)
507:  (*  This procedure gets the position and status of the mouse.           *)
508:  (*****************************************************************************)
509:
510:  VAR
511:    Regs : RECORD
512:             Ax, Bx, Cx, Dx, Bp, Si, Di, Ds, Es, Flags : Integer;
513:           END;
514:
515:  BEGIN
516:    WITH Regs DO
517:      BEGIN
518:        Ax := M1;
519:        Bx := M2;
520:        Cx := M3;
521:        Dx := M4;
522:      END;
523:    Intr(51, Regs);
524:    WITH Regs DO
525:      BEGIN
526:        M1 := Ax;
527:        M2 := Bx;
528:        M3 := Cx;
529:        M4 := Dx;
530:      END;
531:
532:  (***** END procedure mouse  ************************************************)
533:
534:  END;
535:
536:
537:  {$R-}
538:
539:  PROCEDURE GetData(T : Str80; VAR Data; No : Integer; Lx, Rx, By, Ty : Real);
540:
```

```
541:  (*****************************************************************************)
542:  (*  This procedure uses the mouse to input curves.                            *)
543:  (*****************************************************************************)
544:
545:  VAR
546:    M1, M2, M3, M4, Py, Oy, Y, Ny, Ox, X : Integer;
547:    Quit : Boolean;
548:    Ch : Char;
549:    Sx, I, Mx : Integer;
550:    Sy : Real;
551:    D : ARRAY[1..2] OF Real ABSOLUTE Data;
552:
553:  BEGIN
554:    M1 := 0;
555:    M2 := 0;
556:    M3 := 0;
557:    M4 := 0;
558:    Mouse(M1, M2, M3, M4);
559:
560:    (***** Test IF mouse installed  ***************************************)
561:
562:    IF M1 = 0 THEN
563:      BEGIN
564:        ClrScr;
565:        WriteLn(' mouse not installed !!');
566:        Delay(500);
567:      END
568:    ELSE
569:      BEGIN
570:        Sx := Trunc(540/No);
571:        IF Sx > 0 THEN
572:          BEGIN
573:            Mx := Sx*No+41;
574:            Enhanced;
575:            Write(T);
576:            Edraw(39, 19, Mx, 19, 1);
577:            Edraw(Mx, 19, Mx, 181, 1);
578:            Edraw(Mx, 181, 39, 181, 1);
579:            Edraw(39, 181, 39, 19, 1);
580:            Sy := 160/(Ty-Ey);
581:            FOR I := 0 TO No DO
582:              Edraw(((I*Sx)+40), 181, ((I*Sx)+40), 185, 1);
583:            FOR I := 0 TO 10 DO
584:              BEGIN
585:                Edraw(35, ((I*16)+20), 39, ((I*16)+20), 1);
586:                Edraw(Mx, ((I*16)+20), Mx+4, ((I*16)+20), 1);
587:              END;
588:            FOR I := 1 TO No-1 DO
589:              BEGIN
590:                Oy := 180-Trunc((D[I]-By)*Sy);
591:                Y := 180-Trunc((D[I+1]-By)*Sy);
592:                Edraw((((I-1)*Sx)+40), Oy, ((I*Sx)+40), Oy, 1);
593:                Edraw(((I*Sx)+40), Oy, ((I*Sx)+40), Y, 1);
594:              END;
595:            Edraw(Mx-Sx, Y, Mx, Y, 1);
596:            M1 := 1;
597:            Mouse(M1, M2, M3, M4);
598:            Quit := False;
599:            M1 := 3;
600:            REPEAT
```

```
601:              Mouse(M1, M2, M3, M4);
602:              IF M2 <> 0 THEN
603:                BEGIN
604:                  BEGIN
605:                    I := (((M3-40) DIV Sx)+1);
606:                    IF I < 1 THEN I := 1;
607:                    IF I > No THEN I := No;
608:                    Y := 180-Trunc((D[I]-By)*Sy);
609:                    GoToXY(50,1); Write((I/No*(Rx-Lx)+Lx):5:2, ' ', D[I]:5:2);
610:                    X := (I*Sx)+40;
611:                    Ox := 2;
612:                    Mouse(Ox, M2, M3, M4);
613:                    IF I > 1 THEN
614:                      BEGIN
615:                        Oy := 180-Trunc((D[I-1]-By)*Sy);
616:                        Edraw(X-Sx, Oy, X-Sx, Y, 0);
617:                      END
618:                    ELSE
619:                      Oy := -1;
620:                    IF I < No THEN
621:                      BEGIN
622:                        Ny := 180-Trunc((D[I+1]-By)*Sy);
623:                        Edraw(X, Ny, X, Y, 0);
624:                      END
625:                    ELSE
626:                      Ny := -1;
627:                    Edraw(X, Y, X-Sx, Y, 0);
628:                    Py := Y;
629:                    REPEAT
630:                      Mouse(M1, M2, M3, M4);
631:                      Y := M4;
632:                      IF Y < 20 THEN Y := 20;
633:                      IF Y > 180 THEN Y := 180;
634:                      D[I] := (180-Y)/Sy+By;
635:                      GoToXY(50,1); Write((I/No*(Rx-Lx)+Lx):5:2, ' ', D[I]:5:2);
636:                      Edraw(X, Py, X-Sx, Py, 0);
637:                      Edraw(X, Y, X-Sx, Y, 1);
638:                      Py := Y;
639:                    UNTIL M2 = 0;
640:                    Delay(200);
641:                    IF Oy <> -1 THEN Edraw(X-Sx, Oy, X-Sx, Y, 1);
642:                    IF Ny <> -1 THEN Edraw(X, Ny, X, Y, 1);
643:                    Ox := 1;
644:                    Mouse(Ox, M2, M3, M4);
645:                  END;
646:                END;
647:              IF KeyPressed THEN
648:                BEGIN
649:                  Read(Kbd, Ch);
650:                  IF Ch = #27 THEN
651:                    Quit := True;
652:                END;
653:            UNTIL Quit;
654:          END;
655:      END;
656:    TextMode;
657:  END;
658:
659:  {$R+}
660:
```

```
661:
662:   PROCEDURE Say(S : Str80);
663:
664:   VAR
665:     I : Integer;
666:
667:   BEGIN
668:     I := 1;
669:     WHILE I <= Length(S) DO
670:       BEGIN
671:         IF S[I] <> '%' THEN
672:           Write(S[I])
673:         ELSE
674:           IF S[I+1] = '%' THEN
675:             BEGIN
676:               WriteLn;
677:               I := I+1;
678:             END
679:           ELSE
680:             IF S[I+1] = '!' THEN
681:               BEGIN
682:                 NormVideo;            { If your screen doesn't have high/low video, }
683:                 Write(Upcase(S[I+2])); { replace these 3 lines with:                 }
684:                 LowVideo;             {       Write(S[I+2],')');                    }
685:                 I := I+2;
686:               END
687:             ELSE Write('%');
688:         I := I+1;
689:       END;                    { While I<=Length(S) }
690:   END;                        { Say }
691:
692: PROCEDURE filedump;
693:
694: {This procedure is used in the program pwrplot to format a file for
695: the DRGRAPH package using input data calculated in the main program.
696: It may obviously be modified to suit other programs as required.
697: }
698:
699: TYPE
700:      filename = STRING[13];
701:
702: VAR
703:     i,j:integer;
704:     outputfile:text;
705:     name:filename;
706:     outputvar:real;
707:
708:
709: BEGIN
710:      {initialise the file}
711:      clrscr;
712:      writeln;
713:      writeln('Input name of file to be created ');
714:      readln(name);
715:      assign(outputfile,name);
716:      rewrite(outputfile);
717:
718:               FOR j := 0 TO 51 DO
719:               BEGIN
720:                    writeln(outputfile,Solution[j]:10:5,ExpectedInflow[J]:10:5);
```

```
                END;   {curve element for loop}
      writeln(Outputfile,solution[0]:10:5,ExpectedInflow[0]:10:5);
      close(outputfile);

END; {procedure filedmp}


   PROCEDURE Optimise ;
   VAR
     I,J,K,MinPtr : Integer;
     Clevel,Nlevel,Min : Real ;

   BEGIN
     Enhanced;
     Assign(Preal,'b:Dist.p');
     Reset(Preal);
     FOR WK := 1 TO 52 DO
       FOR L := 0 TO 200 DO
         Nodes[L,WK]^.Previous := -1 ;
     WITH Nodes[100,0]^ DO
       BEGIN
         Cost := 0.0 ;
         Previous := 0 ;
       END;

     (***** Start Of Dynamic Programming Optimisation  *****)

     FOR Wk := 0 TO 51 DO
       BEGIN
         Read(Preal,Prob);
         ExpectedInflow[Wk]:=0.0;
         FOR J := 0 TO 63 DO
           BEGIN
             P[J]:=Prob[J];
             ExpectedInflow[Wk]:=ExpectedInflow[Wk]+P[J]*(J+0.5)*Step;
           END;
         IF Wk>42 THEN I:=(Wk-42)*10 ELSE I:=100-10*Wk;
         IF I<0 THEN I:=0;
         IF Wk>42 THEN J:=200-(Wk-42)*10 ELSE J:=100+10*Wk;
         IF J>200 THEN J:=200;
         FOR L:=-10 TO 10 DO Ec[L]:=Ecost(Wk,L);
         FOR L := I TO J DO
           BEGIN
             IF (L MOD 5)=0
             THEN Edraw(19+(Nodes[L,Wk]^.Previous*3),(Wk-1)*6,19+L*3,Wk*6,15);
             Gotoxy(1,1);Write(Wk:4,L:4);
             Start:=L-10;
             IF Start<0 THEN start:=0;
             Stop:=L+10;
             IF Stop>200 THEN Stop:=200;
             Current := Nodes[L,Wk]^ ;
             IF Current.Previous <> -1 THEN
             FOR Next := Start TO Stop DO
               BEGIN
                 Tcost := Current.cost + Ec[Next-L] ;
                 IF (Tcost > Nodes[Next,Wk+1]^.Cost) OR
                    (Nodes[Next,Wk+1]^.Previous = -1) THEN
                 WITH Nodes[Next,Wk+1]^ DO
                   BEGIN
                     Cost := Tcost ;
```

```
781:                    Previous := L ;
782:                  END;
783:              END;
784:          END;
785:      END;
786:   L:=100;
787:   Sol[52]:=L;
788:   Solution[52]:= (Sol[52] - 100) * Step ;
789:   L:=Nodes[L,52]^.Previous ;
790:   FOR Wk :=51 DOWNTO 1 DO
791:     BEGIN
792:       Edraw(19+(Nodes[L,Wk]^.Previous*3),(Wk)*6,19+L*3,(wk+1)*6,13);
793:       Sol[Wk]:=L;
794:       Solution[Wk]:=(Sol[Wk]-100)*Step;
795:       L:=Nodes[L,Wk]^.previous ;
796:     END;
797:   Sol[0]:=100;
798:   Solution[0]:=(Sol[0]-100)*Step;
799:   FOR L:=0 TO 51 DO
800:     ExpectedOutFlow[L]:=ExpectedInflow[L]+Solution[L]-Solution[L+1];
801:
802:   Close(Preal);
803:   REPEAT UNTIL KeyPressed;
804: END;
805:
806:   PROCEDURE MainMenu;
807:
808:   VAR
809:     Command : Char;
810:     ReDraw : Boolean;
811:
812:   BEGIN
813:     ReDraw := True;
814:     REPEAT
815:       IF ReDraw THEN
816:         BEGIN
817:           Title;
818:           Say('%2Main menu');
819:           Say('%2%2%!Input File Name : ');
820:           Write(DataInFileName);
821:           Say('%2%!Output Results : ');
822:           Say('%2%2%!Desired Outflow');
823:           Say('%2%!GO');
824:           Say('%2%!Results');
825:           Say('%2%2%!Quit%2%2');
826:         END;                    ( If ReDraw )
827:       Read(Kbd, Command);
828:       Command := UpCase(Command);
829:       ReDraw := True;
830:       CASE Command OF
831:         'I':BEGIN
832:               Write('Enter Data Input File Name ? ');
833:               ReadLn(DataInFileName);
834:               LoadData;
835:               DataOutFileName:=DataInFileName;
836:             END;
837:         'O':BEGIN
838:               FileDump;
839:             END;
840:             END;
```

```
        'O':Optimise;
        'D':GetData('Desired OutFlow',DesiredOutFlow[0],52,1,52,0,MaxOutFlow);
        'F':GetData('Results',SolutionI[],52,1,52,0,maxLevel);
      END;                        { Case Command }
    UNTIL (Command = 'Q');
    WriteLn;
  END;                            { MainMenu }

BEGIN
  (*****  Create node tree onto heap  ***************************************)

  FOR L := 0 TO 200 DO
    FOR Wk := 0 TO 52 DO
      New(Nodes[L,Wk]);
  Pr := False;
  DataInFileName := 'Test.dat';
  DataOutFileName:=DataInFileName;
  FOR L:=0 TO 51 DO
    BEGIN
      DesiredOutFlow[L]:=MaxOutFlow/2.0;
    END;
  MainMenu;
  IF DataOutFileName <> '' THEN
    SaveData;
END.
```

```
 1: PROGRAM ERichtSimulation;
 2:
 3: CONST
 4:   Black=0;
 5:   Green=2;
 6:   Red=4;
 7:   Brown=6;
 8:   Area=226.1;
 9:   MinLevel=1155.0;
10:   Maxflow=120.96;
11:   MinHead=495.0;
12:   MaxStore=5710.0;
13:   Eff=0.9;
14:   Fac=84.653333;
15:
16: TYPE
17:   String10=STRING[10];
18:
19: VAR
20:   MaxTarget,BaseTarget:Real;
21:   FinTim,Time,StopTime,NextPrnTim,TimeScale:Real;
22:   ClockLow,ClockHigh,I,J,NoToPlot,LastX,ThisX:Integer;
23:   PlotMin,PlotMax,YScale,PlotVar:ARRAY[1..5] OF Real;
24:   TopY,LastY,ThisY:ARRAY[1..5] OF Integer;
25:   PlotLabel:ARRAY[1..5] OF STRING[8];
26:   Heading,ParameterLine:STRING[60];
27:
28:   Target, Expected : ARRAY[0..400] OF Real;
29:   Inflow1, Inflow2, Inflow3, TarExp : Text ;
30:   BaseStore, Store, TotalSpill, Spill, TotalInflow, Inflow,
31:   Energy, TotalEnergy, Outflow, TotalOutflow : Real;
32:
33:   T, W :Integer;
34:
35: PROCEDURE Enhanced;
36: VAR
37:   regs:RECORD CASE integer OF
38:         1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
39:         2:(al,ah,bl,bh,cl,ch,dl,dh:byte);
40:        END;
41: BEGIN
42:     WITH regs DO
43:       ax:=$0010;
44:     intr($10,regs);
45:     WITH regs DO
46:      BEGIN
47:       ah:=$11;
48:       al:=$22;
49:       bl:=2;
50:      END;
51:     intr($10,regs);
52: END;
53:
54:   PROCEDURE eplot(x,y,colour:integer);
55:  VAR
56:   i,off:integer;
57:  BEGIN
58:     port[$3ce]:=5;
59:     port[$3cf]:=2;
60:     off:=(y*80)+(x DIV 8);
```

```
61:     port[$3ce]:=8;
62:     port[$3cf]:=1 SHL (7-(x MOD 8));
63:     i:=mem[$A000:off];
64:     mem[$a000:off]:=colour;
65:     port[$3ce]:=5;
66:     port[$3cf]:=0;
67:     port[$3ce]:=8;
68:     port[$3cf]:=$ff;
69:  END;
70:
71:   PROCEDURE Edraw(x1,y1,x2,y2,colour:integer);
72:   {this procedure rewritten by david loomes}
73:   LABEL 999; {exit point if zero length line}
74:   VAR  x,y,deltax,deltay,dx,dy,run : integer;
75:
76:  PROCEDURE dp(x,y:INTEGER);
77:  VAR
78:   i,off:integer;
79:  BEGIN
80:     off:=(y*80)+(x DIV 8);
81:     port[$3ce]:=8;
82:     port[$3cf]:=1 SHL (7-(x MOD 8));
83:     i:=mem[$A000:off];
84:     mem[$a000:off]:=colour;
85:  END;
86:
87:     BEGIN
88:       port[$3ce]:=5;
89:       port[$3cf]:=2;
90:       IF (x1=x2) AND (y1=y2)
91:       THEN  GOTO 999;
92:
93:       deltay := abs(y1-y2);
94:       deltax := abs(x1-x2);
95:       IF y1>y2 THEN dy:=-1 ELSE dy:=1;
96:       IF x1>x2 THEN dx:=-1 ELSE dx:=1;
97:       x := x1;
98:       y := y1;
99:       IF deltax > deltay {increment along x axis}
100:      THEN  BEGIN
101:            run := deltax SHR 1;
102:            REPEAT
103:             dp(x,y);
104:             x := x+dx;
105:             run := run - deltay;
106:             IF run < 0
107:             THEN  BEGIN
108:                   run := run +deltax;
109:                   y := y+dy
110:                   END;
111:             UNTIL x=x2
112:             END
113:      ELSE  BEGIN
114:            run := deltay SHR 1;
115:            REPEAT
116:             dp(x,y);
117:             y := y+dy;
118:             run := run - deltax;
119:             IF run < 0
120:             THEN  BEGIN
```

```
                    run := run +deltay;
                    x := x+dx
                    END;
              UNTIL y=y2
              END;
999:  dp(x2,y2);
 port[$3ce]:=5;
 port[$3cf]:=0;
 port[$3ce]:=8;
 port[$3cf]:=$ff;
 END;

PROCEDURE ModelDialog;
VAR
  Diff: Real;
  I,J : Integer;
  DataFileName : STRING[30];

  BEGIN
  Assign(Inflow1,'AR183.DAT');
  Reset(Inflow1);
  Assign(Inflow2,'AR283.DAT');
  Reset(Inflow2);
  Assign(Inflow3,'AR3583.DAT');
  Reset(Inflow3);
  Write(' Enter Target Level file name ? ');
  Readln(DataFileName);
  Assign(TarExp,DataFileName);
  Reset(TarExp);
  MaxTarget:=0.0;
  FOR I:=0 TO 52 DO
    BEGIN
      J:=7*I;
      Readln(TarExp,Target[J],Expected[J]);
      IF Target[J]>MaxTarget THEN MaxTarget:=Target[J];
    END;
  FOR I:=0 TO 51 DO
    BEGIN
     Diff:=(Target[(I+1)*7]-Target[I*7])/7.0;
     FOR J:=1 TO 6 DO
       BEGIN
        Expected[(I*7)+J]:=Expected[I*7];
        Target[(I*7)+J]:=Target[I*7]+(Diff*J);
       END;
    END;
  FOR I:=0 TO 36
  DO BEGIN
     J:=I+(7*52);
     Target[J]:=Target[I];
     Expected[J]:=Expected[I];
     END;
  TotalSpill:=0.0;
  TotalOutflow:=0.0;
  TotalEnergy:=0.0;
  BaseTarget:=MaxStore-MaxTarget-26.0;
  Writeln(' Optimum base Storage - ',BaseTarget:10:3);
  Write( ' Enter storage base point ? ');
  Readln(Store);
  BaseStore:=Store;
END;
```

```
181:
182:
183: PROCEDURE Model;
184:   BEGIN
185:   Readln(Inflow1,Inflow);
186:   TotalInflow:=Inflow;
187:   Readln(Inflow2,Inflow);
188:   TotalInflow:=TotalInflow+Inflow;
189:   Readln(Inflow3,Inflow);
190:   TotalInflow:=TotalInflow+Inflow;
191:   T:=Trunc(Time);
192:   W:=T MOD 7;
193: {  Outflow:=Expected[T]+Store-Target[T+7]-BaseStore;}
194:   Outflow:=Expected[T]+(Store+3*(Target[T]+BaseStore))/4.0-Target[T+7]-BaseStore;
195:   IF (W=5) OR (W=6)
196:   THEN Outflow:=Outflow*0.7/6.4
197:   ELSE Outflow:=Outflow/6.4;
198:   IF Outflow<0.0
199:   THEN Outflow:=0.0;
200:   IF OUtflow>Maxflow
201:   THEN Outflow:=MaxFlow;
202:   Store:=Store+TotalInflow-Outflow;
203:   IF Store>MaxStore
204:   THEN BEGIN
205:        Spill:=Store-MaxStore;
206:        Store:=MaxStore;
207:        END
208:   ELSE Spill:=0.0;
209:   Energy:=(MinHead+Store/Area)*Outflow*Eff*fac/3.6E6;
210:
211:   TotalOutflow:=TotalOutflow+Outflow;
212:   TotalSpill:=TotalSpill+Spill;
213:   TotalEnergy:=TotalEnergy+Energy;
214:
215:   PlotVar[1]:=Store/Area+MinLevel;
216:   PlotVar[2]:=(Target[T]+BaseStore)/Area+MinLevel;
217:   PlotVar[3]:=TotalInflow;
218:   PlotVar[4]:=OutFlow;
219:   PlotVar[5]:=Spill;
220:   END;
221:
222: PROCEDURE DataForPlot;
223:   BEGIN
224:   NoToPlot:=4;
225:   PlotLabel[1]:='Level';
226:   PlotMax[1]:=1179.5;
227:   PlotMin[1]:=1155.0;
228:   PlotLabel[2]:='Target';
229:   PlotMax[2]:=1179.9;
230:   PlotMin[2]:=1155.0;
231:   PlotLabel[3]:='Inflow';
232:   PlotMax[3]:=200.0;
233:   PlotMin[3]:=0.0;
234:   PlotLabel[4]:='Outflow';
235:   PlotMax[4]:=130.0;
236:   PlotMin[4]:=0.0;
237:   PlotLabel[5]:='Spill';
238:   PlotMax[5]:=200.0;
239:   PlotMin[5]:=0.0;
240:   END;
```

```
241:
242: PROCEDURE ControlDialog;
243:   BEGIN
244:     FinTim:=7*52;
245:   END;
246:
247: PROCEDURE PlotAxes;
248:   VAR
249:     I:Integer;
250:   BEGIN
251:   EDraw(0,0,0,319,Brown);
252:   EDraw(639,0,639,319,Brown);
253:   EDraw(0,0,639,0,Brown);
254:   FOR I:=1 TO NoToPlot DO
255:     BEGIN
256:     EDraw(0,TopY[I],639,TopY[I],Brown);
257:     END;
258:   END;
259:
260: PROCEDURE StartPlot;
261:   VAR
262:     I:Integer;
263:   BEGIN
264:   TimeScale:=640.0/FinTim;
265:   FOR I:=1 TO NoToPlot DO
266:     BEGIN
267:     YScale[I]:=320.0/(NoToPlot*(PlotMax[I]-PlotMin[I]));
268:     LastX:=0;
269:     TopY[I]:=Round(I*320.0/NoToPlot)-1;
270:     LastY[I]:=TopY[I]-Round(YScale[I]*(PlotVar[I]-PlotMin[I]));
271:     END;
272:   Enhanced;
273:   PlotAxes;
274:   END;
275:
276: PROCEDURE MultiPlot;
277:   VAR
278:     I:Integer;
279:   BEGIN
280:   ThisX:=Trunc(TimeScale*Time);
281:   IF ThisX>LastX THEN
282:     BEGIN
283:     FOR I:=1 TO NoToPlot DO
284:       BEGIN
285:       ThisY[I]:=TopY[I]-Round(YScale[I]*(PlotVar[I]-PlotMin[I]));
286:       EDraw(LastX,LastY[I],ThisX,ThisY[I],Green);
287:       LastY[I]:=ThisY[I];
288:       END;
289:     LastX:=ThisX;
290:     END;
291:   END;
292:
293: VAR
294:   PlotrTimeScale:Real;
295:   PlotrStoreX:ARRAY[0..1200] OF Integer;
296:   PlotrStoreY:ARRAY[1..5,0..1200] OF Real;
297:   PlotrBtmY:ARRAY[1..5] OF Integer;
298:   PlotrYScale:ARRAY[1..5] OF Real;
299:   PlotrStorePtr,PlotrStoreLength:Integer;
300:
```

```
301: PROCEDURE StartPlotter;
302:   VAR
303:     I:Integer;
304:   BEGIN
305:   PlotrTimeScale:=1200.0/FinTim;
306:   PlotrStorePtr:=0;
307:   FOR I:=1 TO NoToPlot DO
308:     BEGIN
309:     PlotrYScale[I]:=5000.0/(NoToPlot*(PlotMax[I]-PlotMin[I]));
310:     PlotrBtmY[I]:=1379+Round((NoToPlot-I)*5000.0/NoToPlot);
311:     PlotrStoreY[I,PlotrStorePtr]:=PlotVar[I];
312:     END;
313:   PlotrStoreX[PlotrStorePtr]:=0;
314:   END;
315:
316: PROCEDURE PlotterStore;
317:   VAR
318:     I,ThisPlotrX:Integer;
319:   BEGIN
320:   ThisPlotrX:=Trunc(PlotrTimeScale*Time);
321:   IF ThisPlotrX>PlotrStoreX[PlotrStorePtr] THEN
322:     BEGIN
323:     PlotrStorePtr:=PlotrStorePtr+1;
324:     PlotrStoreX[PlotrStorePtr]:=ThisPlotrX;
325:     FOR I:=1 TO NoToPlot DO
326:       BEGIN
327:       PlotrStoreY[I,PlotrStorePtr]:=PlotVar[I];
328:       END;
329:     END;
330:   END;
331:
332: PROCEDURE UsrOut(Ch:Char);
333:   VAR
334:     B:Byte;
335:   BEGIN
336:   REPEAT B:=Port[$3FD] AND $21 UNTIL (B=$20) OR (B=$21);
337:   IF B=$21 THEN
338:     BEGIN
339:     B:=Port[$3F8] AND $7F;
340:     IF B=$13 THEN
341:       BEGIN
342:       REPEAT
343:         REPEAT B:=Port[$3FD] AND 1 UNTIL B<>0;
344:         B:=Port[$3F8] AND $7F;
345:         UNTIL B=$11;
346:       END;
347:     END;
348:   Port[$3F8]:=Ord(Ch);
349:   END;
350:
351: PROCEDURE PlotterInit;
352:   BEGIN
353:   UsrOutPtr:=Ofs(UsrOut);
354:   Write(Usr,'IN;',Chr(27),'.I20;;17:',Chr(27),'.N;19:,PA;');
355:   END;
356:
357: PROCEDURE PlotterAxes;
358:   VAR
359:     I:Integer;
360:   BEGIN
```

```
361:   Write(Usr,'PU,1650,1379,PD,1650,6379,8850,6379,8850,1379;');
362:   FOR I:=1 TO NoToPlot DO
363:     BEGIN
364:     Write(Usr,'PU,1650,',PlotrBtmY[I],',PD,8850,',PlotrBtmY[I],';');
365:     END;
366:   END;
367:
368: PROCEDURE PlotterLabels;
369:   VAR
370:     TextX,PlotrLabelY:Integer;
371:   BEGIN
372:   Write(Usr,'SP;');
373:   FOR I:=1 TO NoToPlot DO
374:     BEGIN
375:     PlotrLabelY:=PlotrBtmY[I]+Round(5000/NoToPlot)-150;
376:     Write(Usr,'PU,950,',PlotrLabelY,'LB',PlotMax[I]:6:1,Chr(3));
377:     PlotrLabelY:=PlotrBtmY[I]+Round(2500/NoToPlot)-50;
378:     Write(Usr,'PU,725,',PlotrLabelY,'LB',PlotLabel[I],Chr(3));
379:     PlotrLabelY:=PlotrBtmY[I]+50;
380:     Write(Usr,'PU,950,',PlotrLabelY,'LB',PlotMin[I]:6:1,Chr(3));
381:     END;
382:   Write(Usr,'PU,1500,1179,LB',0.0:3:1,Chr(3));
383:   Write(Usr,'PU,4856,1179,LB','Days',Chr(3));
384:   Write(Usr,'PU,8550,1179,LB',FinTim:6:1,Chr(3));
385:   TextX:=5250-Round(Length(Heading)*112.5/2);
386:   Write(Usr,'PU,',TextX,',6895,LB',Heading,Chr(3));
387:   END;
388:
389: PROCEDURE Plotter;
390:   VAR
391:     I,J,PlotrX,PlotrY:Integer;
392:   BEGIN
393:   WriteLn(' Heading for Plot ? - Maximum length 60 characters ');
394:   WriteLn('.....................................................');
395:   ReadLn(Heading);
396:   WriteLn(' Plotter ready ? - Press any key ');
397:   REPEAT UNTIL KeyPressed;
398:   PlotterInit;
399:   Write(Usr,'SP1;');
400:   PlotterAxes;
401:   PlotterLabels;
402:   Write(Usr,'PU,SP2;');
403:   FOR I:=1 TO NoToPlot DO
404:     BEGIN
405:     PlotrX:=1650;
406:     PlotrY:=PlotrBtmY[I]+Round(PlotrYScale[I]*(PlotrStoreY[I,0]-PlotMin[I]));
407:     Write(Usr,'PU',PlotrX,',',PlotrY,'PD;');
408:     FOR J:=1 TO PlotrStoreLength DO
409:       BEGIN
410:       PlotrX:=1650+6*PlotrStoreX[J];
411:       PlotrY:=PlotrBtmY[I]+Round(PlotrYScale[I]*(PlotrStoreY[I,J]-PlotMin[I]));
412:       Write(Usr,'PA,',PlotrX,',',PlotrY,';');
413:       END;
414:     END;
415:   Write(Usr,'PU,1650,1379;');
416:   END;
417:
418: CONST
419:   ClockPeriod=1;
420:
```

```
421: VAR
422:   PlotReqd:Char;
423:
424: BEGIN
425:
426: ModelDialog;
427: ControlDialog;
428: DataForPlot;
429: StopTime:=FinTim+ClockPeriod*0.1;
430: ClockLow:=0;
431: ClockHigh:=0;
432: Time:=0.0;
433: NextPrnTim:=0.0;
434: WHILE Time<=StopTime DO
435:   BEGIN
436:   Model;
437:   IF Time=0.0 THEN
438:     BEGIN
439:     StartPlot;
440:     StartPlotter;
441:     END
442:   ELSE
443:     BEGIN
444:     MultiPlot;
445:     PlotterStore;
446:     END;
447:   ClockLow:=ClockLow+1;
448:   IF ClockLow=10000 THEN
449:     BEGIN
450:     ClockHigh:=ClockHigh+1;
451:     ClockLow:=0;
452:     END;
453:   Time:=(10000.0*ClockHigh+ClockLow)*ClockPeriod;
454:   IF KeyPressed THEN Exit;
455:   END;
456: PlotrStoreLength:=PlotrStorePtr;
457: REPEAT UNTIL KeyPressed;
458: TextMode(BW80);
459: WriteLn(' Do you wish hard copy ? ');
460: ReadLn(PlotReqd);
461: IF (PlotReqd='Y') OR (PlotReqd='y') THEN
462:   BEGIN
463:   Plotter;
464:   WriteLn(Lst,Heading);
465:   writeln(lst,' spill      Mcf ',TotalSpill:10:5);
466:   writeln(lst,' energy     MJ  ',TotalEnergy:10:5);
467:   writeln(lst,' water used Mcf ',TotalOutflow:10:5);
468:   writeln(lst);
469:   END;
470: writeln(' spill      Mcf ',TotalSpill:10:5);
471: writeln(' energy     MJ  ',TotalEnergy:10:5);
472: writeln(' water used Mcf ',TotalOutflow:10:5);
473: writeln;
474:
475: close(Inflow1);
476: close(Inflow2);
477: close(Inflow3);
478:
479: END.
```

CONVERT.PAS
PROC87.PAS
FILT87.PAS
PLOT.PAS

These programs process the raw data logged at Gaur Power Station. They convert the data from the original CPM binary format to MSDOS 8087 binary format. The inflow is calculated from the reservoir level and the turbine output. The inflow is filtered to remove the effects of noise on the level reading. Finally, the data may be plotted.

INDEX

1. STRATEGIC WATER MANAGEMENT

DP2.PAS This program executes the standard stochastic dynamic programming optimisation on Loch Ericht.

DX.PAS This program performs the modified optimisation of Loch Ericht.

ERICSIM.PAS This program simulates the operation of a target storage plan for actual inflow data.

2. INFLOW PREDICTION

CONVERT.PAS Converts CPM format data to MSDOS files
PROC87.PAS Calculates inflow from logged data
FILT87.PAS Filters Inflow to reduce noise
PLOT87.PAS Plot data on HP plotter

PLOTPAP.PAS Simulate selective models and plot

3. DAILY DECOUPLING

TVDAILY.PAS Performs non-linear programming on daily problem

4. HOURLY SCHEDULING

TVHOURLY.PAS Uses Dynamic Programming to schedule the power output of each station hourly.

5. SET POINT CONTROLLER

RTSIM.PAS Real time simulation of TD governed Hydro Turbine
SLOYSIM.PAS Simulation of set point control at Sloy
FASKSIM.PAS Simulation of set point control at Fasnakyle
TORRSIM.PAS Simulation of set point control at Torr Ackilty

DP2.PAS

This program uses standard stochastic dynamic programming to optimise the strategic storage of Loch Ericht. The program takes in excess of 26 hour for one iteration.

The function Ecost (Line 31-65) calculates the expected cost of going from one node to another.

The main procedure Optimise (Line 67-119) uses this expected cost to find the optimum strategy for the year.

```
61:         Espill := Espill + P[I] * Spill ;
62:       END;
63:     Cost := Cost + Kf * Pfail + Ks * Espill ;
64:     Ecost := Cost ;
65:   END ;
66:
67:   PROCEDURE Optimise ;
68:   BEGIN
69:     Assign(Pfile,'Dist.p');
70:     Reset(Pfile);
71:     Assign(StartFile,'Start.dat');
72:     Reset(StartFile);
73:     FOR L := 0 TO 200 DO
74:       BEGIN
75:         FOR Wk:=1 TO 52 DO Nodes[L,Wk]^.Previous:=-1;
76:         WITH Nodes[L,0]^ DO
77:           Readln(StartFile,Cost,Previous);
78:       END;
79:     Close(StartFile);
80:     FOR Wk := 0 TO 51 DO
81:       BEGIN
82:         Read(Pfile,P);
83:       FOR L := 0 TO 200 DO
84:         BEGIN
85:           Current := Nodes[L,Wk]^ ;
86:           IF Current.Previous <> -1 THEN
87:             BEGIN
88:               Start := L - 20 ;
89:               Stop := L + 20 ;
90:               IF Start < 0 THEN Start := 0 ;
91:               IF Stop > 200 THEN Stop := 200 ;
92:               FOR Next := Start TO Stop DO
93:                 BEGIN
94:                   Tcost := Current.cost + Ecost(L,Next,Wk) ;
95:                   IF (Tcost < Nodes[Next,Wk+1]^.Cost) OR
96:                      (Nodes[Next,Wk+1]^.Previous = -1) THEN
97:                     WITH Nodes[Next,Wk+1]^ DO
98:                       BEGIN
99:                         Cost := Tcost ;
100:                        Previous := L ;
101:                      END;
102:                END;
103:            END;
104:        END;
105:      END;
106:    Assign(StartFile,'Start.dat');
107:    ReWrite(StartFile);
108:    FOR L:=0 TO 200 DO
109:      WITH Nodes[L,52]^ DO
110:        WriteLn(StartFile,Cost:15:5,' ',Previous:5);
111:    Close(startFile);
112:    Assign(Result,'Results.dat');
113:    Rewrite(Result);
114:    FOR Wk:=0 TO 52 DO
115:      FOR L:=0 TO 200 DO
116:        Write(Result,Nodes[l,wk]^);
117:    Close(Pfile);
118:    Close(Result);
119:  END;
120:
```

```
121:
122: BEGIN
123:   (*****  Create node tree onto heap  ***********************************)
124:
125:   FOR L := 0 TO 200 DO
126:     FOR Wk := 0 TO 52 DO
127:       New(Nodes[L,Wk]);
128:   ClrScr;
129:   GotoXY(1,12);
130:   Writeln('                        THIS MACHINE IS IN USE !!!!');
131:   Writeln('                The experiment is expected to run overnight.');
132:   Writeln('                             Jim Clink');
133:   Optimise;
134: END.
```

```
 1: PROGRAM LevelOptimisation ;
 2: {$R+}
 3: CONST
 4:   Ko : Real = 20.00 ;
 5:   Ku : Real = 20.00 ;
 6:   Kb : Real = 100.00 ;
 7:   Kf : Real = 100.00 ;
 8:   Step : Real = 30.87 ;
 9:   MaxLevel : Real = 5910.0 ;
10:   MaxOutFlow : Real = 740.88 ;
11:   DataSetSize = 850 ;
12:
13: TYPE
14:   Str80 = STRING[80];
15:   CharSet = SET OF Char ;
16:   DataSet = ARRAY[1..DataSetSize] OF Byte ;
17:   DataRecord = ARRAY[1..1000] OF Byte ;
18:   FileName = STRING[20] ;
19:   Node = RECORD
20:            Previous : Integer ;
21:            Cost : Real ;
22:          END;
23:   Dist = ARRAY[0..63] OF Real ;
24:
25: VAR
26:   Nodes : ARRAY[0..200,0..52] OF ^Node ;
27:   Current : Node ;
28:   DesiredOutflow : ARRAY[0..52] OF Real ;
29:   Solution : ARRAY[0..52] OF Real ;
30:   ExpectedInflow, ExpectedOutFlow : ARRAY[0..51] OF Real ;
31:   Sol : ARRAY[0..52] OF Integer;
32:   Prob,P : Dist ;
33:   Preal : FILE OF Dist ;
34:   DataFile : FILE OF DataRecord ;
35:   Data : DataSet ABSOLUTE DesiredOutflow ;
36:   DataInFileName,
37:   DataOutFileName : FileName ;
38:   Ec : ARRAY[-10..10] OF Real;
39:   L,Wk,Next,Start,Stop : Integer ;
40:   Tcost,DesiredStore,DesiredLevel : Real ;
41:   Pr : Boolean ;
42:
43:   FUNCTION Ecost(Time,Cx : Integer ):Real ;
44:
45:   VAR
46:     Pfail, Espill, Cost, Dx ,
47:     Inflow, Outflow, Spill : Real ;
48:     I,J : Integer ;
49:
50:   BEGIN
51:     Pfail := 0.0 ;
52:     Espill := 0.0 ;
53:     Cost := 0.0 ;
54:     Dx:=Cx*Step;
55:     IF Pr THEN
56:      BEGIN
57:     Write(Cx:4,time:4);
58:     END;
59:     FOR I := 0 TO 63 DO
60:       BEGIN
```

```
        Inflow := Step*(I+0.5) ;
        Outflow := Inflow - Dx ;
        IF OutFlow < 0 THEN
          BEGIN
            OutFlow := 0 ;
            Pfail := Pfail + P[I] ;
          END
        ELSE
          IF OutFlow > MaxOutFlow THEN
            BEGIN
              OutFlow := MaxOutFlow ;
              Pfail := Pfail + P[I] ;
            END
          ELSE
        IF OutFlow < (DesiredOutFlow[Time]-(4*Step)) THEN
          Cost := Cost + P[I] * Ku * (OutFlow)
          ELSE
        IF OutFlow > (DesiredOutFlow[Time]+(4*Step)) THEN
          Cost := Cost + P[I] * Ko * (OutFlow)
          ELSE
          Cost := Cost + P[I] * Kb * (OutFlow);
{        If Pr THen Writeln('In ',Inflow:8:1,' Out ',outflow:8:1,' P= ',P[i]:10:6,' Cost ',Cost:8:1);
}     END;
    Cost := Cost - Kf * Pfail ;
    IF Pr THEN Writeln('Cost ',cost:8:1,' Pfail ',pfail:10:6,Espill:10:2);

    Ecost := Cost ;
  END ;


PROCEDURE Enhanced;
VAR
  regs:RECORD CASE integer OF
        1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
        2:(al,ah,bl,bh,cl,ch,dl,dh:byte);
       END;
BEGIN
    WITH regs DO
      ax:=$0010;
    intr($10,regs);
    WITH regs DO
     BEGIN
      ah:=$11;
      al:=$22;
      bl:=2;
     END;
    intr($10,regs);
END;

  PROCEDURE eplot(x,y,colour:integer);
  VAR
   i,off:integer;
  BEGIN
    port[$3ce]:=5;
    port[$3cf]:=2;
    off:=(y*80)+(x DIV 8);
    port[$3ce]:=8;
    port[$3cf]:=1 SHL (7-(x MOD 8));
    i:=mem[$A000:off];
    mem[$a000:off]:=colour;
```

```
121:    port[$3ce]:=5;
122:    port[$3cf]:=0;
123:    port[$3ce]:=8;
124:    port[$3cf]:=$ff;
125:  END;
126:
127:   PROCEDURE Edraw(x1,y1,x2,y2,colour:integer);
128:   {this procedure rewritten by david loomes}
129:   LABEL 999; {exit point if zero length line}
130:   VAR  x,y,deltax,deltay,dx,dy,run : integer;
131:
132:  PROCEDURE dp(x,y:INTEGER);
133:  VAR
134:   i,off:integer;
135:  BEGIN
136:    off:=(y*80)+(x DIV 8);
137:    port[$3ce]:=8;
138:    port[$3cf]:=1 SHL (7-(x MOD 8));
139:    i:=mem[$A000:off];
140:    mem[$a000:off]:=colour;
141:  END;
142:
143:    BEGIN
144:      port[$3ce]:=5;
145:      port[$3cf]:=2;
146:      IF (x1=x2) AND (y1=y2)
147:      THEN  GOTO 999;
148:
149:      deltay := abs(y1-y2);
150:      deltax := abs(x1-x2);
151:      IF y1)y2 THEN dy:=-1 ELSE dy:=1;
152:      IF x1)x2 THEN dx:=-1 ELSE dx:=1;
153:      x := x1;
154:      y := y1;
155:      IF deltax ) deltay (increment along x axis)
156:      THEN  BEGIN
157:            run := deltax SHR 1;
158:            REPEAT
159:             dp(x,y);
160:             x := x+dx;
161:             run := run - deltay;
162:             IF run ( 0
163:             THEN  BEGIN
164:                   run := run +deltax;
165:                   y := y+dy
166:                   END;
167:             UNTIL x=x2
168:             END
169:      ELSE  BEGIN
170:            run := deltay SHR 1;
171:            REPEAT
172:             dp(x,y);
173:             y := y+dy;
174:             run := run - deltax;
175:             IF run ( 0
176:             THEN  BEGIN
177:                   run := run +deltay;
178:                   x := x+dx
179:                   END;
180:             UNTIL y=y2
```

```
181:              END;
182: 999:  dp(x2,y2);
183:  port[$3ce]:=5;
184:  port[$3cf]:=0;
185:  port[$3ce]:=8;
186:  port[$3cf]:=$ff;
187:  END;
188:
189:
190:   PROCEDURE LoadData;
191:
192:   (*****************************************************************************)
193:   (*  This procedure loads start data from file.                               *)
194:   (*****************************************************************************)
195:
196:   VAR
197:     PP: RECORD CASE Integer OF
198:           1: (P1000: DataRecord);
199:           2: (Parmz: DataSet);
200:         END;
201:
202:   BEGIN
203:      Assign(DataFile,DataInFileName);
204:
205:     {$I-}
206:
207:     Reset(DataFile);
208:
209:     {$I+}
210:
211:     IF IOResult <> 0 THEN
212:       BEGIN
213:         WriteLn(DataInFileName, ' not found !');
214:         Delay(500);
215:       END
216:     ELSE
217:       BEGIN
218:         Read(DataFile,PP.P1000);
219:         Data:=PP.Parmz;
220:         Close(DataFile);
221:       END;
222:
223:   (***** END procedure loaddata  ***********************************************)
224:
225:   END;
226:
227:
228:   PROCEDURE SaveData;
229:
230:   (*****************************************************************************)
231:   (*  This procedure saves start data to file.                                 *)
232:   (*****************************************************************************)
233:
234:   VAR
235:     Command : CHAR;
236:     PP: RECORD CASE Integer OF
237:           1: (P1000: DataRecord);
238:           2: (Parmz: DataSet);
239:         END;
240:
```

```
241:  BEGIN
242:    Assign(DataFile,DataOutFileName);
243:
244:    {$I-}
245:
246:    Reset(DataFile);
247:
248:    {$I+}
249:
250:    IF IOResult <> 0 THEN
251:      BEGIN
252:        WriteLn('New File ',DataOutFileName,' !');
253:        Delay(500);
254:        Command := 'Y';
255:      END
256:    ELSE
257:      BEGIN
258:        Close(DataFile);
259:        WriteLn(' Overwrite (DESTROY) old ', DataOutFileName);
260:        Read(Kbd,Command);
261:        Command := UpCase(Command);
262:      END;
263:    IF Command = 'Y' THEN
264:      BEGIN
265:        Assign(DataFile, DataOutFileName);
266:
267:        {$I-}
268:
269:        Rewrite(DataFile);
270:
271:        {$I+}
272:
273:        IF IOResult = 0 THEN
274:          BEGIN
275:            PP.Parmz := Data;
276:            Write(DataFile,PP.P1000);
277:            Close(DataFile);
278:          END
279:        ELSE
280:          WriteLn(' Unable to open file ', DataOutFileName);
281:      END;
282:    Delay(500);
283:
284:  (***** END procedure Savedata  ********************************************)
285:
286:  END;
287:
288:
289:  PROCEDURE Title;
290:
291:  BEGIN
292:    ClrScr;
293:    NormVideo;
294:    WriteLn('          Loch Ericht Level Optimisation Program Ver. 1.0   ');
295:    WriteLn;
296:    LowVideo;
297:  END;
298:
299:
300:  FUNCTION UpcaseStr(S : Str80) : Str80;
```

```
301:
302:  (*****************************************************************************)
303:  (*  This Function returns a string which contains the upper case string  *)
304:  (*  of the parameter.                                                        *)
305:  (*****************************************************************************)
306:
307:  VAR
308:    P : Integer;
309:  BEGIN
310:    FOR P := 1 TO Length(S) DO
311:      S[P] := UpCase(S[P]);
312:    UpcaseStr := S;
313:  END;
314:
315:
316:  FUNCTION ConstStr(C : Char; N : Integer) : Str80;
317:
318:  (*****************************************************************************)
319:  (*  ConstStr returns a string with N characters of value C                  *)
320:  (*****************************************************************************)
321:
322:  VAR
323:    S : STRING[80];
324:  BEGIN
325:    IF N < 0 THEN
326:      N := 0;
327:    S[0] := Chr(N);
328:    FillChar(S[1], N, C);
329:    ConstStr := S;
330:  END;
331:
332:
333:  PROCEDURE Beep;
334:
335:  (*****************************************************************************)
336:  (*  Beep sounds the terminal bell or beeper                                  *)
337:  (*****************************************************************************)
338:
339:  BEGIN
340:    Write(^G);
341:  END;
342:
343:
344:  PROCEDURE InputStr(VAR Ssss;
345:                     L, X, Y : Integer;
346:                     Term : CharSet;
347:                     VAR TC : Char;
348:                     Ins : Boolean);
349:
350:  (*****************************************************************************)
351:  (*  This procedure allow the reading and editing of any string.              *)
352:  (*****************************************************************************)
353:
354:  CONST
355:    UnderScore = '_';
356:  VAR
357:    S : STRING[255] ABSOLUTE Ssss;
358:    P : Integer;
359:    Ch : Char;
360:  BEGIN
```

```
361:     GoToXY(X, Y); Write(S, ConstStr(UnderScore, L-Length(S)));
362:     P := 0;
363:     REPEAT
364:       GoToXY(X+P, Y); Read(Kbd, Ch);
365:       CASE Ch OF
366:
367:         (*****  The Following is only valid for the IBM PC  *****************)
368:
369:         #27 : IF KeyPressed THEN
370:                 BEGIN
371:                   Read(Kbd, Ch);
372:                   CASE Ch OF
373:                     #75 : IF P > 0 THEN
374:                             P := P-1
375:                           ELSE Beep;
376:                     #77 : IF P < Length(S) THEN
377:                             P := P+1
378:                           ELSE Beep;
379:                     #71 : P := 0;
380:                     #79 : P := Length(S);
381:                     #83 : IF P < Length(S) THEN
382:                             BEGIN
383:                               Delete(S, P+1, 1);
384:                               Write(Copy(S, P+1, L), UnderScore);
385:                             END;
386:                     #72 : Ch := ^F;
387:                     #80 : Ch := ^A;
388:                     #73 : Ch := ^E;
389:                     #81 : Ch := ^X;
390:                     #82 : Ch := ^V;
391:                   ELSE Beep;
392:                   END;
393:                 END;
394:         #32..#126 : IF (Ins) OR (P = Length(S)) THEN
395:                       BEGIN
396:                         IF P < L THEN
397:                           BEGIN
398:                             IF Length(S) = L THEN
399:                               Delete(S, L, 1);
400:                             P := P+1;
401:                             Insert(Ch, S, P);
402:                             Write(Copy(S, P, L));
403:                           END
404:                         ELSE Beep;
405:                       END
406:                     ELSE
407:                       BEGIN
408:                         IF P < L THEN P := P+1;
409:                         S[P] := Ch;
410:                         Write(Copy(S, P, L));
411:                       END;
412:         ^S : IF P > 0 THEN
413:                P := P-1
414:              ELSE Beep;
415:         ^D : IF P < Length(S) THEN
416:                P := P+1
417:              ELSE Beep;
418:         ^G : IF P < Length(S) THEN
419:                BEGIN
420:                  Delete(S, P+1, 1);
```

```
421:                Write(Copy(S, P+1, L), UnderScore);
422:              END;
423:       ^H, #127 : IF P > 0 THEN
424:                    BEGIN
425:                      Delete(S, P, 1);
426:                      Write(^H, Copy(S, P, L), UnderScore);
427:                      P := P-1;
428:                    END
429:                  ELSE Beep;
430:       ^Y : BEGIN
431:              Write(ConstStr(UnderScore, Length(S)-P));
432:              Delete(S, P+1, L);
433:            END;
434:     ELSE
435:       IF NOT(Ch IN Term) THEN Beep;
436:     END;                    {of case}
437:   UNTIL Ch IN Term;
438:   P := Length(S);
439:   GoToXY(X+P, Y);
440:   Write(''':L-P);
441:   TC := Ch;
442:
443: (***** END procedure InputStr ***********************************************)
444:
445: END;
446:
447: PROCEDURE InputReal(VAR A:Real;
448:                         L,X,Y:INTEGER;
449:                         Term:CharSet;
450:                         VAR TC:CHAR;
451:                         Ins:Boolean);
452: VAR
453:  Rstr : STRING[10];
454:  Temp : Real;
455:  Test : INTEGER;
456:
457: BEGIN
458:   Str(A:10:2,Rstr);
459:   InputStr(Rstr,L,X,Y,Term,TC,Ins);
460:   Test:=1;
461:   WHILE Test<=Length(Rstr) DO
462:     IF Rstr[Test]=' ' THEN
463:       Delete(Rstr,Test,1)
464:     ELSE
465:       Test:=Test+1;
466:   Val(Rstr,Temp,Test);
467:   IF Test=0 THEN
468:     BEGIN
469:       A:=Temp;
470:     END;
471:   GotoXY(X,Y);
472:   Write(a:10:2);
473: END;
474:
475: PROCEDURE InputInt(VAR A:INTEGER;
476:                         L,X,Y:INTEGER;
477:                         Term:CharSet;
478:                         VAR TC:CHAR;
479:                         Ins:Boolean);
480: VAR
```

```
481:    Rstr : STRING[10];
482:    Temp : INTEGER;
483:    Test : INTEGER;
484:
485:  BEGIN
486:    Str(A:10,Rstr);
487:    InputStr(Rstr,L,X,Y,Term,TC,Ins);
488:    Test:=1;
489:    WHILE Test<=Length(Rstr) DO
490:      IF Rstr[Test]=' ' THEN
491:        Delete(Rstr,Test,1)
492:      ELSE
493:        Test:=Test+1;
494:    Val(Rstr,Temp,Test);
495:    IF Test=0 THEN
496:      BEGIN
497:        A:=Temp;
498:      END;
499:    GotoXY(X,Y);
500:    Write(a:10);
501:  END;
502:
503:
504:  PROCEDURE Mouse(VAR M1, M2, M3, M4 : Integer);
505:
506:  (*************************************************************************)
507:  (*  This procedure gets the position and status of the mouse.            *)
508:  (*************************************************************************)
509:
510:  VAR
511:    Regs : RECORD
512:             Ax, Bx, Cx, Dx, Bp, Si, Di, Ds, Es, Flags : Integer;
513:           END;
514:
515:  BEGIN
516:    WITH Regs DO
517:      BEGIN
518:        Ax := M1;
519:        Bx := M2;
520:        Cx := M3;
521:        Dx := M4;
522:      END;
523:    Intr(51, Regs);
524:    WITH Regs DO
525:      BEGIN
526:        M1 := Ax;
527:        M2 := Bx;
528:        M3 := Cx;
529:        M4 := Dx;
530:      END;
531:
532:  (***** END procedure mouse  ********************************************)
533:
534:  END;
535:
536:
537:  {$R-}
538:
539:  PROCEDURE GetData(T : Str80; VAR Data; No : Integer; Lx, Rx, By, Ty : Real);
540:
```

```
541:  (*****************************************************************************)
542:  (*  This procedure uses the mouse to input curves.                            *)
543:  (*****************************************************************************)
544:
545:  VAR
546:    M1, M2, M3, M4, Py, Oy, Y, Ny, Ox, X : Integer;
547:    Quit : Boolean;
548:    Ch : Char;
549:    Sx, I, Mx : Integer;
550:    Sy : Real;
551:    D : ARRAY[1..2] OF Real ABSOLUTE Data;
552:
553:  BEGIN
554:    M1 := 0;
555:    M2 := 0;
556:    M3 := 0;
557:    M4 := 0;
558:    Mouse(M1, M2, M3, M4);
559:
560:    (*****  Test IF mouse installed  ***************************************)
561:
562:    IF M1 = 0 THEN
563:      BEGIN
564:        ClrScr;
565:        WriteLn(' mouse not installed !!');
566:        Delay(500);
567:      END
568:    ELSE
569:      BEGIN
570:        Sx := Trunc(540/No);
571:        IF Sx > 0 THEN
572:          BEGIN
573:            Mx := Sx*No+41;
574:            Enhanced;
575:            Write(T);
576:            Edraw(39, 19, Mx, 19, 1);
577:            Edraw(Mx, 19, Mx, 181, 1);
578:            Edraw(Mx, 181, 39, 181, 1);
579:            Edraw(39, 181, 39, 19, 1);
580:            Sy := 160/(Ty-By);
581:            FOR I := 0 TO No DO
582:              Edraw(((I*Sx)+40), 181, ((I*Sx)+40), 185, 1);
583:            FOR I := 0 TO 10 DO
584:              BEGIN
585:                Edraw(35, ((I*16)+20), 39, ((I*16)+20), 1);
586:                Edraw(Mx, ((I*16)+20), Mx+4, ((I*16)+20), 1);
587:              END;
588:            FOR I := 1 TO No-1 DO
589:              BEGIN
590:                Oy := 180-Trunc((D[I]-By)*Sy);
591:                Y := 180-Trunc((D[I+1]-By)*Sy);
592:                Edraw((((I-1)*Sx)+40), Oy, ((I*Sx)+40), Oy, 1);
593:                Edraw(((I*Sx)+40), Oy, ((I*Sx)+40), Y, 1);
594:              END;
595:            Edraw(Mx-Sx, Y, Mx, Y, 1);
596:            M1 := 1;
597:            Mouse(M1, M2, M3, M4);
598:            Quit := False;
599:            M1 := 3;
600:            REPEAT
```

```
601:              Mouse(M1, M2, M3, M4);
602:              IF M2 <> 0 THEN
603:                BEGIN
604:                  BEGIN
605:                    I := (((M3-40) DIV Sx)+1);
606:                    IF I < 1 THEN I := 1;
607:                    IF I > No THEN I := No;
608:                    Y := 180-Trunc((DII-By)*Sy);
609:                    GoToXY(50,1); Write((I/No*(Rx-Lx)+Lx):5:2, ' ', DII:5:2);
610:                    X := (I*Sx)+40;
611:                    Ox := 2;
612:                    Mouse(Ox, M2, M3, M4);
613:                    IF I > 1 THEN
614:                      BEGIN
615:                        Oy := 180-Trunc((DII-1]-By)*Sy);
616:                        Edraw(X-Sx, Oy, X-Sx, Y, 0);
617:                      END
618:                    ELSE
619:                      Oy := -1;
620:                    IF I < No THEN
621:                      BEGIN
622:                        Ny := 180-Trunc((DII+1]-By)*Sy);
623:                        Edraw(X, Ny, X, Y, 0);
624:                      END
625:                    ELSE
626:                      Ny := -1;
627:                    Edraw(X, Y, X-Sx, Y, 0);
628:                    Py := Y;
629:                    REPEAT
630:                      Mouse(M1, M2, M3, M4);
631:                      Y := M4;
632:                      IF Y < 20 THEN Y := 20;
633:                      IF Y > 180 THEN Y := 180;
634:                      DII := (180-Y)/Sy+By;
635:                      GoToXY(50,1); Write((I/No*(Rx-Lx)+Lx):5:2, ' ', DII:5:2);
636:                      Edraw(X, Py, X-Sx, Py, 0);
637:                      Edraw(X, Y, X-Sx, Y, 1);
638:                      Py := Y;
639:                    UNTIL M2 = 0;
640:                    Delay(200);
641:                    IF Oy <> -1 THEN Edraw(X-Sx, Oy, X-Sx, Y, 1);
642:                    IF Ny <> -1 THEN Edraw(X, Ny, X, Y, 1);
643:                    Ox := 1;
644:                    Mouse(Ox, M2, M3, M4);
645:                  END;
646:                END;
647:              IF KeyPressed THEN
648:                BEGIN
649:                  Read(Kbd, Ch);
650:                  IF Ch = #27 THEN
651:                    Quit := True;
652:                END;
653:            UNTIL Quit;
654:          END;
655:      END;
656:    TextMode;
657:  END;
658:
659:  {$R+}
660:
```

```
661:
662:  PROCEDURE Say(S : Str80);
663:
664:  VAR
665:    I : Integer;
666:
667:  BEGIN
668:    I := 1;
669:    WHILE I <= Length(S) DO
670:      BEGIN
671:        IF S[I] <> '%' THEN
672:          Write(S[I])
673:        ELSE
674:          IF S[I+1] = '@' THEN
675:            BEGIN
676:              WriteLn;
677:              I := I+1;
678:            END
679:          ELSE
680:            IF S[I+1] = '!' THEN
681:              BEGIN
682:                NormVideo;            { If your screen doesn't have high/low video, }
683:                Write(Upcase(S[I+2])); { replace these 3 lines with:                 }
684:                LowVideo;             {       Write(S[I+2],')');                    }
685:                I := I+2;
686:              END
687:            ELSE Write('%');
688:        I := I+1;
689:      END;                    { While I<=Length(S) }
690:  END;                        { Say }
691:
692: PROCEDURE filedump;
693:
694: {This procedure is used in the program pwrplot to format a file for
695: the DRGRAPH package using input data calculated in the main program.
696: It may obviously be modified to suit other programs as required.
697: }
698:
699: TYPE
700:     filename = STRING[13];
701:
702: VAR
703:     i,j:integer;
704:     outputfile:text;
705:     name:filename;
706:     outputvar:real;
707:
708:
709: BEGIN
710:     {initialise the file}
711:     clrscr;
712:     writeln;
713:     writeln('Input name of file to be created ');
714:     readln(name);
715:     assign(outputfile,name);
716:     rewrite(outputfile);
717:
718:             FOR j := 0 TO 51 DO
719:             BEGIN
720:                 writeln(outputfile,Solution[j]:10:5,ExpectedInflow[J]:10:5);
```

```
721:              END;   (curve element for loop)
722:      writeln(Outputfile,solution[0]:10:5,ExpectedInflow[0]:10:5);
723:      close(outputfile);
724:
725: END; (procedure filedmp)
726:
727:
728:   PROCEDURE Optimise ;
729:   VAR
730:     I,J,K,MinPtr : Integer;
731:     Clevel,Nlevel,Min : Real ;
732:
733:   BEGIN
734:     Enhanced;
735:     Assign(Preal,'b:Dist.p');
736:     Reset(Preal);
737:     FOR Wk := 1 TO 52 DO
738:       FOR L := 0 TO 200 DO
739:         Nodes[L,Wk]^.Previous := -1 ;
740:     WITH Nodes[100,0]^ DO
741:       BEGIN
742:         Cost := 0.0 ;
743:         Previous := 0 ;
744:       END;
745:
746:     (***** Start Of Dynamic Programming Optimisation  *****)
747:
748:     FOR Wk := 0 TO 51 DO
749:       BEGIN
750:         Read(Preal,Prob);
751:         ExpectedInflow[Wk]:=0.0;
752:         FOR J := 0 TO 63 DO
753:           BEGIN
754:             P[J]:=Prob[J];
755:             ExpectedInflow[Wk]:=ExpectedInflow[Wk]+P[J]*(J+0.5)*Step;
756:           END;
757:         IF Wk>42 THEN I:=(Wk-42)*10 ELSE I:=100-10*Wk;
758:         IF I<0 THEN I:=0;
759:         IF Wk>42 THEN J:=200-(Wk-42)*10 ELSE J:=100+10*Wk;
760:         IF J>200 THEN J:=200;
761:         FOR L:=-10 TO 10 DO Ec[L]:=Ecost(Wk,L);
762:         FOR L := I TO J DO
763:           BEGIN
764:             IF (L MOD 5)=0
765:             THEN Edraw(19+(Nodes[L,Wk]^.Previous*3),(Wk-1)*6,19+L*3,Wk*6,15);
766:             Goto(1,3);Write(Wk:4,L:4);
767:             Start:=L-10;
768:             IF Start<0 THEN start:=0;
769:             Stop:=L+10;
770:             IF Stop>200 THEN Stop:=200;
771:             Current := Nodes[L,Wk]^ ;
772:             IF Current.Previous <> -1 THEN
773:             FOR Next := Start TO Stop DO
774:               BEGIN
775:                 Tcost := Current.cost + Ec[Next-L] ;
776:                 IF (Tcost > Nodes[Next,Wk+1]^.Cost) OR
777:                    (Nodes[Next,Wk+1]^.Previous = -1) THEN
778:                 WITH Nodes[Next,Wk+1]^ DO
779:                   BEGIN
780:                     Cost := Tcost ;
```

```
781:                    Previous := L ;
782:                  END;
783:              END;
784:          END;
785:      END;
786:   L:=100;
787:   Sol[52]:=L;
788:   Solution[52]:= (Sol[52] - 100) * Step ;
789:   L:=Nodes[L,52]^.Previous ;
790:   FOR WK :=51 DOWNTO 1 DO
791:     BEGIN
792:       Edraw(19+(Nodes[L,Wk]^.Previous*3),(Wk)*6,19+L*3,(wk+1)*6,13);
793:       Sol[Wk]:=L;
794:       Solution[Wk]:=(Sol[Wk]-100)*Step;
795:       L:=Nodes[L,WK]^.previous ;
796:     END;
797:   Sol[0]:=100;
798:   Solution[0]:=(Sol[0]-100)*Step;
799:   FOR L:=0 TO 51 DO
800:     ExpectedOutFlow[L]:=ExpectedInflow[L]+Solution[L]-Solution[L+1];
801:
802:   Close(Preal);
803:   REPEAT UNTIL KeyPressed;
804: END;
805:
806:   PROCEDURE MainMenu;
807:
808:   VAR
809:     Command : Char;
810:     ReDraw : Boolean;
811:
812:   BEGIN
813:     ReDraw := True;
814:     REPEAT
815:       IF ReDraw THEN
816:         BEGIN
817:           Title;
818:           Say('%@Main menu');
819:           Say('%@%@%!Input File Name : ');
820:           Write(DataInFileName);
821:           Say('%@%!Output Results : ');
822:           Say('%@%@%!Desired Outflow');
823:           Say('%@%!GO');
824:           Say('%@%!Results');
825:           Say('%@%@%!Quit(%@%@)');
826:         END;                    { If ReDraw }
827:       Read(Kbd, Command);
828:       Command := UpCase(Command);
829:       ReDraw := True;
830:       CASE Command OF
831:         'I':BEGIN
832:               Write('Enter Data Input File Name ? ');
833:               ReadLn(DataInFileName);
834:               LoadData;
835:               DataOutFileName:=DataInFileName;
836:             END;
837:         'O':BEGIN
838:               FileDump;
839:             END;
840:             END;
```

```
841:         'G':Optimise;
842:         'D':GetData('Desired Outflow ',DesiredOutFlow[0],52,1,52,0,MaxOutFlow);
843:         'R':GetData('Results ',Solution[1],52,1,52,0,maxLevel);
844:       END;                        { Case Command }
845:     UNTIL (Command = 'Q');
846:     WriteLn;
847:   END;                            { MainMenu }
848:
849: BEGIN
850:   (*****  Create node tree onto heap  ************************************)
851:
852:   FOR L := 0 TO 200 DO
853:     FOR WK := 0 TO 52 DO
854:       New(Nodes[L,WK]);
855:   Pr := False;
856:   DataInFileName := 'Test.dat';
857:   DataOutFileName:=DataInFileName;
858:   FOR L:=0 TO 51 DO
859:     BEGIN
860:       DesiredOutFlow[L]:=MaxOutFlow/2.0;
861:     END;
862:   MainMenu;
863:   IF DataOutFileName <> '' THEN
864:     SaveData;
865: END.
```

```
 1: PROGRAM ERichtSimulation;
 2:
 3: CONST
 4:   Black=0;
 5:   Green=2;
 6:   Red=4;
 7:   Brown=6;
 8:   Area=226.1;
 9:   MinLevel=1155.0;
10:   Maxflow=120.96;
11:   MinHead=495.0;
12:   MaxStore=5710.0;
13:   Eff=0.9;
14:   Fac=84.653333;
15:
16: TYPE
17:   String10=STRING[10];
18:
19: VAR
20:   MaxTarget,BaseTarget:Real;
21:   FinTim,Time,StopTime,NextPrnTim,TimeScale:Real;
22:   ClockLow,ClockHigh,I,J,NoToPlot,LastX,ThisX:Integer;
23:   PlotMin,PlotMax,YScale,PlotVar:ARRAY[1..5] OF Real;
24:   TopY,LastY,ThisY:ARRAY[1..5] OF Integer;
25:   PlotLabel:ARRAY[1..5] OF STRING[8];
26:   Heading,ParameterLine:STRING[60];
27:
28:   Target, Expected : ARRAY[0..400] OF Real;
29:   Inflow1, Inflow2, Inflow3, TarExp : Text ;
30:   BaseStore, Store, TotalSpill, Spill, TotalInflow, Inflow,
31:   Energy, TotalEnergy, Outflow, TotalOutflow : Real;
32:
33:   T, W :Integer;
34:
35: PROCEDURE Enhanced;
36: VAR
37:   regs:RECORD CASE integer OF
38:         1:(ax,bx,cx,dx,bp,si,di,ds,es,flags:integer);
39:         2:(al,ah,bl,bh,cl,ch,dl,dh:byte);
40:        END;
41: BEGIN
42:     WITH regs DO
43:       ax:=$0010;
44:     intr($10,regs);
45:     WITH regs DO
46:      BEGIN
47:       ah:=$11;
48:       al:=$22;
49:       bl:=2;
50:      END;
51:     intr($10,regs);
52: END;
53:
54:   PROCEDURE eplot(x,y,colour:integer);
55:  VAR
56:   i,off:integer;
57:  BEGIN
58:     port[$3ce]:=5;
59:     port[$3cf]:=2;
60:     off:=(y*80)+(x DIV 8);
```

```
    port[$3ce]:=8;
    port[$3cf]:=1 SHL (7-(x MOD 8));
    i:=mem[$A000:off];
    mem[$a000:off]:=colour;
    port[$3ce]:=5;
    port[$3cf]:=0;
    port[$3ce]:=8;
    port[$3cf]:=$ff;
END;

 PROCEDURE Edraw(x1,y1,x2,y2,colour:integer);
 {this procedure rewritten by david loomes}
 LABEL 999; {exit point if zero length line}
 VAR  x,y,deltax,deltay,dx,dy,run : integer;

PROCEDURE dp(x,y:INTEGER);
VAR
 i,off:integer;
BEGIN
    off:=(y*80)+(x DIV 8);
    port[$3ce]:=8;
    port[$3cf]:=1 SHL (7-(x MOD 8));
    i:=mem[$A000:off];
    mem[$a000:off]:=colour;
END;

    BEGIN
      port[$3ce]:=5;
      port[$3cf]:=2;
      IF (x1=x2) AND (y1=y2)
      THEN  GOTO 999;

      deltay := abs(y1-y2);
      deltax := abs(x1-x2);
      IF y1>y2 THEN dy:=-1 ELSE dy:=1;
      IF x1>x2 THEN dx:=-1 ELSE dx:=1;
      x := x1;
      y := y1;
      IF deltax > deltay {increment along x axis}
      THEN  BEGIN
            run := deltax SHR 1;
            REPEAT
             dp(x,y);
             x := x+dx;
             run := run - deltay;
             IF run < 0
             THEN  BEGIN
                   run := run +deltax;
                   y := y+dy
                   END;
             UNTIL x=x2
             END
      ELSE  BEGIN
            run := deltay SHR 1;
            REPEAT
             dp(x,y);
             y := y+dy;
             run := run - deltax;
             IF run < 0
             THEN  BEGIN
```

```
                    run := run +deltay;
                    x := x+dx
                    END;
               UNTIL y=y2
               END;
999:  dp(x2,y2);
 port[$3ce]:=5;
 port[$3cf]:=0;
 port[$3ce]:=8;
 port[$3cf]:=$ff;
 END;

PROCEDURE ModelDialog;
VAR
  Diff: Real;
  I,J : Integer;
  DataFileName : STRING[30];

  BEGIN
  Assign(Inflow1,'AR183.DAT');
  Reset(Inflow1);
  Assign(Inflow2,'AR283.DAT');
  Reset(Inflow2);
  Assign(Inflow3,'AR3583.DAT');
  Reset(Inflow3);
  Write(' Enter Target Level file name ? ');
  Readln(DataFileName);
  Assign(TarExp,DataFileName);
  Reset(TarExp);
  MaxTarget:=0.0;
  FOR I:=0 TO 52 DO
    BEGIN
      J:=7*I;
      Readln(TarExp,Target[J],Expected[J]);
      IF Target[J]>MaxTarget THEN MaxTarget:=Target[J];
    END;
  FOR I:=0 TO 51 DO
    BEGIN
     Diff:=(Target[(I+1)*7]-Target[I*7])/7.0;
     FOR J:=1 TO 6 DO
       BEGIN
        Expected[(I*7)+J]:=Expected[I*7];
        Target[(I*7)+J]:=Target[I*7]+(Diff*J);
       END;
    END;
  FOR I:=0 TO 66
  DO BEGIN
     J:=I+(7*52);
     Target[J]:=Target[I];
     Expected[J]:=Expected[I];
     END;
  TotalSpill:=0.0;
  TotalOutflow:=0.0;
  TotalEnergy:=0.0;
  BaseTarget:=MaxStore-MaxTarget-26.0;
  Writeln(' Optimum base Storage - ',BaseTarget:10:3);
  Write( ' Enter storage base point ? ');
  Readln(Store);
  BaseStore:=Store;
END;
```

```
181:
182:
183: PROCEDURE Model;
184:   BEGIN
185:   Readln(Inflow1,Inflow);
186:   TotalInflow:=Inflow;
187:   Readln(Inflow2,Inflow);
188:   TotalInflow:=TotalInflow+Inflow;
189:   Readln(Inflow3,Inflow);
190:   TotalInflow:=TotalInflow+Inflow;
191:   T:=Trunc(Time);
192:   W:=T MOD 7;
193: {  Outflow:=Expected[T]+Store-Target[T+7]-BaseStore;}
194:   Outflow:=Expected[T]+(Store+3*(Target[T]+BaseStore))/4.0-Target[T+7]-BaseStore;
195:   IF (W=5) OR (W=6)
196:   THEN Outflow:=Outflow*0.7/6.4
197:   ELSE Outflow:=Outflow/6.4;
198:   IF Outflow(0.0
199:   THEN Outflow:=0.0;
200:   IF OUtflow)Maxflow
201:   THEN Outflow:=MaxFlow;
202:   Store:=Store+TotalInflow-Outflow;
203:   IF Store)MaxStore
204:   THEN BEGIN
205:        Spill:=Store-MaxStore;
206:        Store:=MaxStore;
207:        END
208:   ELSE Spill:=0.0;
209:   Energy:=(MinHead+Store/Area)*Outflow*Eff*fac/3.6E6;
210:
211:   TotalOutflow:=TotalOutflow+Outflow;
212:   TotalSpill:=TotalSpill+Spill;
213:   TotalEnergy:=TotalEnergy+Energy;
214:
215:   PlotVar[1]:=Store/Area+MinLevel;
216:   PlotVar[2]:=(Target[T]+BaseStore)/Area+MinLevel;
217:   PlotVar[3]:=TotalInflow;
218:   PlotVar[4]:=OutFlow;
219:   PlotVar[5]:=Spill;
220:   END;
221:
222: PROCEDURE DataForPlot;
223:   BEGIN
224:   NoToPlot:=4;
225:   PlotLabel[1]:='Level';
226:   PlotMax[1]:=1179.9;
227:   PlotMin[1]:=1155.0;
228:   PlotLabel[2]:='Target';
229:   PlotMax[2]:=1179.9;
230:   PlotMin[2]:=1155.0;
231:   PlotLabel[3]:='Inflow';
232:   PlotMax[3]:=200.0;
233:   PlotMin[3]:=0.0;
234:   PlotLabel[4]:='Outflow';
235:   PlotMax[4]:=130.0;
236:   PlotMin[4]:=0.0;
237:   PlotLabel[5]:='Spill';
238:   PlotMax[5]:=200.0;
239:   PlotMin[5]:=0.0;
240:   END;
```

```
241:
242: PROCEDURE ControlDialog;
243:   BEGIN
244:     FinTim:=7*52;
245:   END;
246:
247: PROCEDURE PlotAxes;
248:   VAR
249:     I:Integer;
250:   BEGIN
251:   EDraw(0,0,0,319,Brown);
252:   EDraw(639,0,639,319,Brown);
253:   EDraw(0,0,639,0,Brown);
254:   FOR I:=1 TO NoToPlot DO
255:     BEGIN
256:     EDraw(0,TopY[I],639,TopY[I],Brown);
257:     END;
258:   END;
259:
260: PROCEDURE StartPlot;
261:   VAR
262:     I:Integer;
263:   BEGIN
264:   TimeScale:=640.0/FinTim;
265:   FOR I:=1 TO NoToPlot DO
266:     BEGIN
267:     YScale[I]:=320.0/(NoToPlot*(PlotMax[I]-PlotMin[I]));
268:     LastX:=0;
269:     TopY[I]:=Round(I*320.0/NoToPlot)-1;
270:     LastY[I]:=TopY[I]-Round(YScale[I]*(PlotVar[I]-PlotMin[I]));
271:     END;
272:   Enhanced;
273:   PlotAxes;
274:   END;
275:
276: PROCEDURE MultiPlot;
277:   VAR
278:     I:Integer;
279:   BEGIN
280:   ThisX:=Trunc(TimeScale*Time);
281:   IF ThisX>LastX THEN
282:     BEGIN
283:     FOR I:=1 TO NoToPlot DO
284:       BEGIN
285:       ThisY[I]:=TopY[I]-Round(YScale[I]*(PlotVar[I]-PlotMin[I]));
286:       EDraw(LastX,LastY[I],ThisX,ThisY[I],Green);
287:       LastY[I]:=ThisY[I];
288:       END;
289:     LastX:=ThisX;
290:     END;
291:   END;
292:
293: VAR
294:   PlotrTimeScale:Real;
295:   PlotrStoreX:ARRAY[0..1200] OF Integer;
296:   PlotrStoreY:ARRAY[1..5,0..1200] OF Real;
297:   PlotrBtmY:ARRAY[1..5] OF Integer;
298:   PlotrYScale:ARRAY[1..5] OF Real;
299:   PlotrStorePtr,PlotrStoreLength:Integer;
300:
```

```
301: PROCEDURE StartPlotter;
302:   VAR
303:     I:Integer;
304:   BEGIN
305:   PlotrTimeScale:=1200.0/FinTim;
306:   PlotrStorePtr:=0;
307:   FOR I:=1 TO NoToPlot DO
308:     BEGIN
309:     PlotrYScale[I]:=5000.0/(NoToPlot*(PlotMax[I]-PlotMin[I]));
310:     PlotrBtmY[I]:=1379+Round((NoToPlot-I)*5000.0/NoToPlot);
311:     PlotrStoreY[I,PlotrStorePtr]:=PlotVar[I];
312:     END;
313:   PlotrStoreX[PlotrStorePtr]:=0;
314:   END;
315:
316: PROCEDURE PlotterStore;
317:   VAR
318:     I,ThisPlotrX:Integer;
319:   BEGIN
320:   ThisPlotrX:=Trunc(PlotrTimeScale*Time);
321:   IF ThisPlotrX>PlotrStoreX[PlotrStorePtr] THEN
322:     BEGIN
323:     PlotrStorePtr:=PlotrStorePtr+1;
324:     PlotrStoreX[PlotrStorePtr]:=ThisPlotrX;
325:     FOR I:=1 TO NoToPlot DO
326:       BEGIN
327:       PlotrStoreY[I,PlotrStorePtr]:=PlotVar[I];
328:       END;
329:     END;
330:   END;
331:
332: PROCEDURE UsrOut(Ch:Char);
333:   VAR
334:     B:Byte;
335:   BEGIN
336:   REPEAT B:=Port[$3FD] AND $21 UNTIL (B=$20) OR (B=$21);
337:   IF B=$21 THEN
338:     BEGIN
339:     B:=Port[$3F8] AND $7F;
340:     IF B=$13 THEN
341:       BEGIN
342:       REPEAT
343:         REPEAT B:=Port[$3FD] AND 1 UNTIL B<>0;
344:         B:=Port[$3F8] AND $7F;
345:         UNTIL B=$11;
346:       END;
347:     END;
348:   Port[$3F8]:=Ord(Ch);
349:   END;
350:
351: PROCEDURE PlotterInit;
352:   BEGIN
353:   UsrOutPtr:=Ofs(UsrOut);
354:   Write(Usr,'IN;',Chr(27),'.I20;;17:',Chr(27),'.N;19:,PA;');
355:   END;
356:
357: PROCEDURE PlotterAxes;
358:   VAR
359:     I:Integer;
360:   BEGIN
```

```
361:   Write(Usr,'PU,1650,1379,PD,1650,6379,8850,6379,8850,1379;');
362:   FOR I:=1 TO NoToPlot DO
363:     BEGIN
364:     Write(Usr,'PU,1650,',PlotrBtmY[I],',PD,8850,',PlotrBtmY[I],';');
365:     END;
366:   END;
367:
368: PROCEDURE PlotterLabels;
369:   VAR
370:     TextX,PlotrLabelY:Integer;
371:   BEGIN
372:   Write(Usr,'SR;');
373:   FOR I:=1 TO NoToPlot DO
374:     BEGIN
375:     PlotrLabelY:=PlotrBtmY[I]+Round(5000/NoToPlot)-150;
376:     Write(Usr,'PU,950,',PlotrLabelY,'LB',PlotMax[I]:6:1,Chr(3));
377:     PlotrLabelY:=PlotrBtmY[I]+Round(2500/NoToPlot)-50;
378:     Write(Usr,'PU,725,',PlotrLabelY,'LB',PlotLabel[I],Chr(3));
379:     PlotrLabelY:=PlotrBtmY[I]+50;
380:     Write(Usr,'PU,950,',PlotrLabelY,'LB',PlotMin[I]:6:1,Chr(3));
381:     END;
382:   Write(Usr,'PU,1500,1179,LB',0.0:3:1,Chr(3));
383:   Write(Usr,'PU,4856,1179,LB','Days',Chr(3));
384:   Write(Usr,'PU,8550,1179,LB',FinTim:6:1,Chr(3));
385:   TextX:=5250-Round(Length(Heading)*112.5/2);
386:   Write(Usr,'PU,',TextX,',6895,LB',Heading,Chr(3));
387:   END;
388:
389: PROCEDURE Plotter;
390:   VAR
391:     I,J,PlotrX,PlotrY:Integer;
392:   BEGIN
393:   WriteLn(' Heading for Plot ? - Maximum length 60 characters ');
394:   WriteLn('').....................................................('');
395:   ReadLn(Heading);
396:   WriteLn(' Plotter ready ? - Press any key ');
397:   REPEAT UNTIL KeyPressed;
398:   PlotterInit;
399:   Write(Usr,'SP1;');
400:   PlotterAxes;
401:   PlotterLabels;
402:   Write(Usr,'PU,SP2;');
403:   FOR I:=1 TO NoToPlot DO
404:     BEGIN
405:     PlotrX:=1650;
406:     PlotrY:=PlotrBtmY[I]+Round(PlotrYScale[I]*(PlotrStoreY[I,0]-PlotMin[I]));
407:     Write(Usr,'PU',PlotrX,',',PlotrY,'PD;');
408:     FOR J:=1 TO PlotrStoreLength DO
409:       BEGIN
410:       PlotrX:=1650+6*PlotrStoreX[J];
411:       PlotrY:=PlotrBtmY[I]+Round(PlotrYScale[I]*(PlotrStoreY[I,J]-PlotMin[I]));
412:       Write(Usr,'PA,',PlotrX,',',PlotrY,';');
413:       END;
414:     END;
415:   Write(Usr,'PU,1650,1379;');
416:   END;
417:
418: CONST
419:   ClockPeriod=1;
420:
```

```
421: VAR
422:   PlotReqd:Char;
423:
424: BEGIN
425:
426: ModelDialog;
427: ControlDialog;
428: DataForPlot;
429: StopTime:=FinTim+ClockPeriod*0.1;
430: ClockLow:=0;
431: ClockHigh:=0;
432: Time:=0.0;
433: NextPrnTim:=0.0;
434: WHILE Time<=StopTime DO
435:   BEGIN
436:   Model;
437:   IF Time=0.0 THEN
438:     BEGIN
439:     StartPlot;
440:     StartPlotter;
441:     END
442:   ELSE
443:     BEGIN
444:     MultiPlot;
445:     PlotterStore;
446:     END;
447:   ClockLow:=ClockLow+1;
448:   IF ClockLow=10000 THEN
449:     BEGIN
450:     ClockHigh:=ClockHigh+1;
451:     ClockLow:=0;
452:     END;
453:   Time:=(10000.0*ClockHigh+ClockLow)*ClockPeriod;
454:   IF KeyPressed THEN Exit;
455:   END;
456: PlotrStoreLength:=PlotrStorePtr;
457: REPEAT UNTIL KeyPressed;
458: TextMode(BW80);
459: WriteLn(' Do you wish hard copy ? ');
460: ReadLn(PlotReqd);
461: IF (PlotReqd='Y') OR (PlotReqd='y') THEN
462:   BEGIN
463:   Plotter;
464:   WriteLn(Lst,Heading);
465:   writeln(lst,' spill      Mcf ',TotalSpill:10:5);
466:   writeln(lst,' energy     MJ  ',TotalEnergy:10:5);
467:   writeln(lst,' water used Mcf ',TotalOutflow:10:5);
468:   writeln(lst);
469:   END;
470: writeln(' spill      Mcf ',TotalSpill:10:5);
471: writeln(' energy     MJ  ',TotalEnergy:10:5);
472: writeln(' water used Mcf ',TotalOutflow:10:5);
473: writeln;
474:
475: close(Inflow1);
476: close(Inflow2);
477: close(Inflow3);
478:
479: END.
```

TVDAILY.PAS

This program performs the daily decoupling of the reservoirs within the valley, using non-linear programming methods. Procedure CostandPD (Line 830-918) calculates the cost of the current solution and partial derivatives of the cost with respect to the daily outflow. The procedure DailySchedule (Line 815-1131) uses the partial dervatives to adjust the current solution.

```
PROGRAM DailyDecouplingProblem;

  {$R+}

CONST
  DCon : Real = 256.0;
  Limit : Real = 50.0;
  DataSetSize = 2000;
  MaxStation = 8;
  MaxTime = 24;

  (*****************************************************************************
   *****  CONSTANTS WHICH DEFINE THE SIZE OF THE PROBLEM  *****************
   *****************************************************************************)

TYPE
  Str80 = STRING[80];
  Str15 = STRING[15];
  Str10 = STRING[10];
  CharSet = SET OF Char;
  Vector = ARRAY[1..MaxStation] OF Real;
  Point = ARRAY[1..MaxStation] OF Integer;
  BUFFER = STRING[80];
  Names = ARRAY[1..MaxStation] OF STRING[10];
  Flow = ARRAY[1..MaxTime] OF Real;
  FileName = STRING[20];
  DataSet = ARRAY[1..DataSetSize] OF Byte;
  DataRecord = ARRAY[1..1880] OF Byte;



VAR
   *****************************************************************************
  (*****  CONSTANTS FOR THE OPTIMISATION PROBLEM  *************************
   *****************************************************************************

   *****  LOCH INFORMATION: STORAGES IN MCF  ******************************
   *****                      LEVELS IN FEET  ******************************

  LochName : Names;
  MaxStore : Vector;
  MinStore : Vector;
  Area : Vector;
  MinLevel : Vector;

   *****  DEFINITION OF [illegible]  *****  ***********************

  Destination : Point;
  SpillDestination : Point;
  SpillDelay : Vector;

   *****  STATION INFORMATION: FLOW IN MCF PER SET PER HOUR  **************

  StationName : Names;
  FlowPerUnit : Vector;
  Popt : Vector;
  MaxTurbine : Point;
  MinTurbine : Point;

  (*****  COST FUNCTIONS FOR OPTIMISATION  *********************************
```

```
61:
62:   CP : Vector;
63:   CL1 : Vector;
64:   CS : Vector;
65:   CL2 : Vector;
66:   CE : Vector;
67:
68:   (***** DAWN AND DUSK DEFINITIONS FOR STATION EIGHT FLOW CONTROL  *********)
69:
70:   Dawn : Integer;
71:   Dusk : Integer;
72:
73:   (*****  STEP SIZE FOR DAILY NLP PROBLEM  ********************************)
74:
75:   StepSize : Real;
76:
77:   (*****  BASE TIME FOR START OF OPTIMISATION  ****************************)
78:
79:   BaseTime : Integer;
80:
81:   (*****************************************************************************)
82:   (*****  MAIN SYSTEM VARIABLE DEFINITION  *********************************)
83:   (*****************************************************************************)
84:
85:   Demand : Flow;
86:   Spillage, Storage, Inflow : Vector;
87:   InitialLevel, DesiredLevel : Vector;
88:   InitialStore, DesiredStore : Vector;
89:   Solution : Vector;
90:
91:   Buff : ARRAY[1..500] OF Real;
92:
93:   Data : DataSet ABSOLUTE LochName;
94:
95:   DataInFileName : FileName;
96:   DataOutFileName : FileName;
97:   I, J : Integer;
98:
99:
100:
101:  PROCEDURE LoadData;
102:
103:    (*****************************************************************************)
104:    (*  THIS PROCEDURE LOADS START DATA FROM FILE.                          *)
105:    (*****************************************************************************)
106:
107:  VAR
108:    DataFile : FILE OF DataRecord;
109:    PP : RECORD CASE Integer OF
110:      1 : (P1000 : DataRecord);
111:      2 : (Parmz : DataSet);
112:        END;
113:
114:  BEGIN
115:    Assign(DataFile, DataInFileName);
116:
117:    {$I-}
118:
119:    Reset(DataFile);
120:
```

```
121:    {$I+}
122:
123:    IF IOResult <> 0 THEN
124:      BEGIN
125:        WriteLn(DataInFileName, ' not found !');
126:        Delay(500);
127:      END
128:    ELSE
129:      BEGIN
130:        Read(DataFile, PP.P1000);
131:        Data := PP.Parmz;
132:        Close(DataFile);
133:      END;
134:    SpillDestination[2] := 6;
135:    (***** END PROCEDURE LOADDATA  ********************************************)
136:
137:  END;
138:
139:
140:  PROCEDURE SaveData;
141:
142:    (*****************************************************************************)
143:    (*  THIS PROCEDURE SAVES START DATA TO FILE.                                 *)
144:    (*****************************************************************************)
145:
146:  VAR
147:    DataFile : FILE OF DataRecord;
148:    Command : Char;
149:    PP : RECORD CASE Integer OF
150:      1 : (P1000 : DataRecord);
151:      2 : (Parmz : DataSet);
152:         END;
153:
154:  BEGIN
155:    Assign(DataFile, DataOutFileName);
156:
157:    {$I-}
158:
159:    Reset(DataFile);
160:
161:    {$I+}
162:
163:    IF IOResult <> 0 THEN
164:      BEGIN
165:        WriteLn('New File ', DataOutFileName, ' !');
166:        Delay(500);
167:        Command := 'Y';
168:      END
169:    ELSE
170:      BEGIN
171:        Close(DataFile);
172:        WriteLn(' Overwrite (DESTROY) old ', DataOutFileName);
173:        Read(Kbd, Command);
174:        Command := UpCase(Command);
175:      END;
176:    IF Command = 'Y' THEN
177:      BEGIN
178:        Assign(DataFile, DataOutFileName);
179:
180:        {$I-}
```

```
181:
182:        Rewrite(DataFile);
183:
184:        {$I+}
185:
186:        IF IOResult = 0 THEN
187:          BEGIN
188:            PP.Parmz := Data;
189:            Write(DataFile, PP.P1000);
190:            Close(DataFile);
191:          END
192:        ELSE
193:          WriteLn(' Unable to open file ', DataOutFileName);
194:      END;
195:    Delay(500);
196:
197:    (***** END PROCEDURE SAVEDATA  ***********************************************)
198:
199:  END;
200:
201:
202:  PROCEDURE Title;
203:
204:  BEGIN
205:    ClrScr;
206:    NormVideo;
207:    WriteLn('       Tummel Valley Daily Decoupling Program Ver. 3.0   ');
208:    WriteLn;
209:    LowVideo;
210:  END;
211:
212:
213:  FUNCTION UpcaseStr(S : Str80) : Str80;
214:
215:    (*****************************************************************************)
216:    (*  THIS FUNCTION RETURNS A STRING WHICH CONTAINS THE UPPER CASE STRING  *)
217:    (*  OF THE PARAMETER.                                                    *)
218:    (*****************************************************************************)
219:
220:  VAR
221:    P : Integer;
222:  BEGIN
223:    FOR P := 1 TO Length(S) DO
224:      S[P] := UpCase(S[P]);
225:    UpcaseStr := S;
226:  END;
227:
228:
229:  FUNCTION ConstStr(C : Char; N : Integer) : Str80;
230:
231:    (*****************************************************************************)
232:    (*  CONSTSTR RETURNS A STRING WITH N CHARACTERS OF VALUE C               *)
233:    (*****************************************************************************)
234:
235:  VAR
236:    S : STRING[80];
237:  BEGIN
238:    IF N < 0 THEN
239:      N := 0;
240:    S[0] := Chr(N);
```

```
241:    FillChar(S[1], N, C);
242:    ConstStr := S;
243:  END;
244:
245:
246:  PROCEDURE Beep;
247:
248:    (*************************************************************************)
249:    (*  BEEP SOUNDS THE TERMINAL BELL OR BEEPER                              *)
250:    (*************************************************************************)
251:
252:  BEGIN
253:    Write(^G);
254:  END;
255:
256:
257:  PROCEDURE InputStr(VAR Ssss;
258:                     L, X, Y : Integer;
259:                     Term : CharSet;
260:                     VAR TC : Char;
261:                     Ins : Boolean);
262:
263:    (*************************************************************************)
264:    (*  THIS PROCEDURE ALLOW THE READING AND EDITING OF ANY STRING.          *)
265:    (*************************************************************************)
266:
267:  CONST
268:    UnderScore = '_';
269:  VAR
270:    S : STRING[255] ABSOLUTE Ssss;
271:    P : Integer;
272:    Ch : Char;
273:  BEGIN
274:    GoToXY(X, Y); Write(S, ConstStr(UnderScore, L-Length(S)));
275:    P := 0;
276:    REPEAT
277:      GoToXY(X+P, Y); Read(Kbd, Ch);
278:      CASE Ch OF
279:
280:        (*****  THE FOLLOWING IS ONLY VALID FOR THE IBM PC  ****************)
281:
282:        #27 : IF KeyPressed THEN
283:                BEGIN
284:                  Read(Kbd, Ch);
285:                  CASE Ch OF
286:                    #75 : IF P > 0 THEN
287:                            P := P-1
288:                          ELSE Beep;
289:                    #77 : IF P < Length(S) THEN
290:                            P := P+1
291:                          ELSE Beep;
292:                    #71 : P := 0;
293:                    #79 : P := Length(S);
294:                    #83 : IF P < Length(S) THEN
295:                            BEGIN
296:                              Delete(S, P+1, 1);
297:                              Write(Copy(S, P+1, L), UnderScore);
298:                            END;
299:                    #72 : Ch := ^F;
300:                    #80 : Ch := ^A;
```

```
                    #73 : Ch := ^E;
                    #81 : Ch := ^X;
                    #82 : Ch := ^V;
                  ELSE Beep;
                  END;
                END;
      #32..#126 : IF (Ins) OR (P = Length(S)) THEN
                    BEGIN
                      IF P < L THEN
                        BEGIN
                          IF Length(S) = L THEN
                            Delete(S, L, 1);
                          P := P+1;
                          Insert(Ch, S, P);
                          Write(Copy(S, P, L));
                        END
                      ELSE Beep;
                    END
                  ELSE
                    BEGIN
                      IF P < L THEN P := P+1;
                      S[P] := Ch;
                      Write(Copy(S, P, L));
                    END;
      ^S : IF P > 0 THEN
             P := P-1
           ELSE Beep;
      ^D : IF P < Length(S) THEN
             P := P+1
           ELSE Beep;
      ^G : IF P < Length(S) THEN
             BEGIN
               Delete(S, P+1, 1);
               Write(Copy(S, P+1, L), UnderScore);
             END;
      ^H, #127 : IF P > 0 THEN
                   BEGIN
                     Delete(S, P, 1);
                     Write(^H, Copy(S, P, L), UnderScore);
                     P := P-1;
                   END
                 ELSE Beep;
      ^Y : BEGIN
             Write(ConstStr(UnderScore, Length(S)-P));
             Delete(S, P+1, L);
           END;
    ELSE
      IF NOT(Ch IN Term) THEN Beep;
    END;                    {OF CASE}
  UNTIL Ch IN Term;
  P := Length(S);
  GoToXY(X+P, Y);
  Write(''':L-P);
  TC := Ch;

  (*****  END PROCEDURE INPUTSTR  ****************************************)

END;

PROCEDURE InputReal(VAR A : Real;
```

```
                    L, X, Y : Integer;
                    Term : CharSet;
                    VAR TC : Char;
                    Ins : Boolean);
VAR
  Rstr : STRING[10];
  Temp : Real;
  Test : Integer;

BEGIN
  Str(A:L:2, Rstr);
  InputStr(Rstr, L, X, Y, Term, TC, Ins);
  Test := 1;
  WHILE Test <= Length(Rstr) DO
    IF Rstr[Test] = ' ' THEN
      Delete(Rstr, Test, 1)
    ELSE
      Test := Test+1;
  Val(Rstr, Temp, Test);
  IF Test = 0 THEN
    BEGIN
      A := Temp;
    END;
  GoToXY(X, Y);
  Write(A:L:2);
END;

PROCEDURE InputInt(VAR A : Integer;
                   L, X, Y : Integer;
                   Term : CharSet;
                   VAR TC : Char;
                   Ins : Boolean);
VAR
  Rstr : STRING[10];
  Temp : Integer;
  Test : Integer;

BEGIN
  Str(A:8, Rstr);
  InputStr(Rstr, L, X, Y, Term, TC, Ins);
  Test := 1;
  WHILE Test <= Length(Rstr) DO
    IF Rstr[Test] = ' ' THEN
      Delete(Rstr, Test, 1)
    ELSE
      Test := Test+1;
  Val(Rstr, Temp, Test);
  IF Test = 0 THEN
    BEGIN
      A := Temp;
    END;
  GoToXY(X, Y);
  Write(A:8);
END;


PROCEDURE Mouse(VAR M1, M2, M3, M4 : Integer);

  (*************************************************************************)
  (*  THIS PROCEDURE GETS THE POSITION AND STATUS OF THE MOUSE.            *)
```

```
421:    (*****************************************************************************)
422:
423:  VAR
424:    Regs : RECORD
425:             Ax, Bx, Cx, Dx, Bp, Si, Di, Ds, Es, Flags : Integer;
426:           END;
427:
428:  BEGIN
429:    WITH Regs DO
430:      BEGIN
431:        Ax := M1;
432:        Bx := M2;
433:        Cx := M3;
434:        Dx := M4;
435:      END;
436:    Intr(51, Regs);
437:    WITH Regs DO
438:      BEGIN
439:        M1 := Ax;
440:        M2 := Bx;
441:        M3 := Cx;
442:        M4 := Dx;
443:      END;
444:
445:    (***** END PROCEDURE MOUSE  ************************************************)
446:
447:  END;
448:
449:
450:  {$R-}
451:
452:  PROCEDURE GetData(T : Str80; VAR Data; No : Integer; Lx, Rx, By, Ty : Real);
453:
454:    (*****************************************************************************)
455:    (*  THIS PROCEDURE USES THE MOUSE TO INPUT CURVES.                           *)
456:    (*****************************************************************************)
457:
458:  VAR
459:    M1, M2, M3, M4, Py, Oy, Y, Ny, Ox, X : Integer;
460:    Quit : Boolean;
461:    Ch : Char;
462:    Sx, I, My : Integer;
463:    Sy : Real;
464:    D : ARRAY[1..2] OF Real ABSOLUTE Data;
465:
466:  BEGIN
467:    M1 := 0;
468:    M2 := 0;
469:    M3 := 0;
470:    M4 := 0;
471:    Mouse(M1, M2, M3, M4);
472:
473:    (*****  TEST IF MOUSE INSTALLED  ********************************************)
474:
475:    IF M1 = 0 THEN
476:      BEGIN
477:        ClrScr;
478:        WriteLn(' mouse not installed !!');
479:        Delay(500);
480:      END
```

```
481:     ELSE
482:       BEGIN
483:         Sx := Trunc(540/No);
484:         IF Sx > 0 THEN
485:           BEGIN
486:             Mx := Sx*No+41;
487:             HiRes;
488:             Write(T);
489:             Draw(39, 19, Mx, 19, 1);
490:             Draw(Mx, 19, Mx, 181, 1);
491:             Draw(Mx, 181, 39, 181, 1);
492:             Draw(39, 181, 39, 19, 1);
493:             Sy := 160/(Ty-By);
494:             FOR I := 0 TO No DO
495:               Draw(((I*Sx)+40), 181, ((I*Sx)+40), 185, 1);
496:             FOR I := 0 TO 10 DO
497:               BEGIN
498:                 Draw(35, ((I*16)+20), 39, ((I*16)+20), 1);
499:                 Draw(Mx, ((I*16)+20), Mx+4, ((I*16)+20), 1);
500:               END;
501:             FOR I := 1 TO No-1 DO
502:               BEGIN
503:                 Oy := 180-Trunc((D[I]-By)*Sy);
504:                 Y := 180-Trunc((D[I+1]-By)*Sy);
505:                 Draw(((I-1)*Sx)+40), Oy, ((I*Sx)+40), Oy, 1);
506:                 Draw(((I*Sx)+40), Oy, ((I*Sx)+40), Y, 1);
507:               END;
508:             Draw(Mx-Sx, Y, Mx, Y, 1);
509:             M1 := 1;
510:             Mouse(M1, M2, M3, M4);
511:             Quit := False;
512:             M1 := 3;
513:             REPEAT
514:               Mouse(M1, M2, M3, M4);
515:               IF M2 <> 0 THEN
516:                 BEGIN
517:                   BEGIN
518:                     I := (((M3-40) DIV Sx)+1);
519:                     IF I < 1 THEN I := 1;
520:                     IF I > No THEN I := No;
521:                     Y := 180-Trunc((D[I]-By)*Sy);
522:                     GoToXY(50, 1); Write((I/No*(Rx-Lx)+Lx):5:2, ' ', D[I]:5:2);
523:                     X := (I*Sx)+40;
524:                     Ox := 2;
525:                     Mouse(Ox, M2, M3, M4);
526:                     IF I > 1 THEN
527:                       BEGIN
528:                         Oy := 180-Trunc((D[I-1]-By)*Sy);
529:                         Draw(X-Sx, Oy, X-Sx, Y, 0);
530:                       END
531:                     ELSE
532:                       Oy := -1;
533:                     IF I < No THEN
534:                       BEGIN
535:                         Ny := 180-Trunc((D[I+1]-By)*Sy);
536:                         Draw(X, Ny, X, Y, 0);
537:                       END
538:                     ELSE
539:                       Ny := -1;
540:                     Draw(X, Y, X-Sx, Y, 0);
```

```
                    Py := Y;
                    REPEAT
                      Mouse(M1, M2, M3, M4);
                      Y := M4;
                      IF Y < 20 THEN Y := 20;
                      IF Y > 180 THEN Y := 180;
                      D[I] := (180-Y)/Sy+By;
                      GoToXY(50, 1); Write((I/No*(Rx-Lx)+Lx):5:2, ' ', D[I]:5:2);
                      Draw(X, Py, X-Sx, Py, 0);
                      Draw(X, Y, X-Sx, Y, 1);
                      Py := Y;
                    UNTIL M2 = 0;
                    Delay(200);
                    IF Oy <> -1 THEN Draw(X-Sx, Oy, X-Sx, Y, 1);
                    IF Ny <> -1 THEN Draw(X, Ny, X, Y, 1);
                    Ox := 1;
                    Mouse(Ox, M2, M3, M4);
                  END;
                END;
              IF KeyPressed THEN
                BEGIN
                  Read(Kbd, Ch);
                  IF Ch = #27 THEN
                    Quit := True;'
                END;
            UNTIL Quit;
          END;
      END;
    TextMode;
END;

{$R+}


PROCEDURE Say(S : Str80);

VAR
  I : Integer;

BEGIN
  I := 1;
  WHILE I <= Length(S) DO
    BEGIN
      IF S[I] <> '%' THEN
        Write(S[I])
      ELSE
        IF S[I+1] = '@' THEN
          BEGIN
            WriteLn;
            I := I+1;
          END
        ELSE
          IF S[I+1] = '!' THEN
            BEGIN
              NormVideo;    { IF YOUR SCREEN DOESN'T HAVE HIGH/LOW VIDEO, }
              Write(UpCase(S[I+2])); { REPLACE THESE 3 LINES WITH:          }
              LowVideo;     {       WRITE(S[I+2],')');                    }
              I := I+2;
            END
          ELSE Write('%');
```

```
      I := I+1;
    END;                        { WHILE I<=LENGTH(S) }
  END;                          { SAY }

PROCEDURE InitialScreen;

VAR
  I, L : Integer;
  TC : Char;
BEGIN
  Title;
  WriteLn('                          Initial Functions');
  WriteLn('IMMMMMMMMMMMMMMMMMMMMMQMMMMMMMMQMMMMMMMMQMMMMMMMMQMMMMMMMMQMMMMMMMMQMMMMMMMM;');
  WriteLn(':  Station              3Initial 3Initial 3 Final  3 Final  3 Inflow 3Outflow :');
  WriteLn(':                       3 Level  3Storage 3 Level  3Storage 3        3        :');
  FOR I := 1 TO 8 DO
    BEGIN
      WriteLn('GDDDDDDDDDDDDDDDDDDDDDEDDDDDDDDEDDDDDDDDEDDDDDDDDEDDDDDDDDEDDDDDDDDEDDDDDDDD6');
      WriteLn(':', StationName[I], '':21-Length(StationName[I]), '3', InitialLevel[I]:8:2, '3',
      InitialStore[I]:8:2, '3', DesiredLevel[I]:8:2, '3', DesiredStore[I]:8:2, '3',
      Inflow[I]:8:2, '3', Solution[I]:8:2, ':');
    END;
  WriteLn('HMMMMMMMMMMMMMMMMMMMMMOMMMMMMMMOMMMMMMMMOMMMMMMMMOMMMMMMMMOMMMMMMMMOMMMMMMMM<');
  I := 1;
  L := 1;
  REPEAT
    CASE I OF
      1 : BEGIN
            InputReal(InitialLevel[L], 8, 24, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
            InitialStore[L] := (InitialLevel[L]-MinLevel[L])*Area[L];
            IF InitialStore[L] < 0.0 THEN
              BEGIN
                InitialStore[L] := 0.0;
                InitialLevel[L] := MinLevel[L];
              END;
            IF InitialStore[L] > (MaxStore[L]+200.0) THEN
              BEGIN
                InitialStore[L] := MaxStore[L];
                InitialLevel[L] := MinLevel[L]+(MaxStore[L]/Area[L]);
              END;
            GoToXY(24, (6+L*2)); Write(InitialLevel[L]:8:2);
            GoToXY(33, (6+L*2)); Write(InitialStore[L]:8:2);
          END;
      2 : BEGIN
            InputReal(InitialStore[L], 8, 33, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
            InitialLevel[L] := MinLevel[L]+(InitialStore[L]/Area[L]);
            IF InitialStore[L] < 0.0 THEN
              BEGIN
                InitialStore[L] := 0.0;
                InitialLevel[L] := MinLevel[L];
              END;
            IF InitialStore[L] > (MaxStore[L]+200.0) THEN
              BEGIN
                InitialStore[L] := MaxStore[L];
                InitialLevel[L] := MinLevel[L]+(MaxStore[L]/Area[L]);
              END;
            GoToXY(24, (6+L*2)); Write(InitialLevel[L]:8:2);
            GoToXY(33, (6+L*2)); Write(InitialStore[L]:8:2);
          END;
      3 : BEGIN
```

```
              InputReal(DesiredLevel[L], 8, 42, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
              DesiredStore[L] := (DesiredLevel[L]-MinLevel[L])*Area[L];
              IF DesiredStore[L] < 0.0 THEN
                BEGIN
                  DesiredStore[L] := 0.0;
                  DesiredLevel[L] := MinLevel[L];
                END;
              IF DesiredStore[L] > (MaxStore[L]+200.0) THEN
                BEGIN
                  DesiredStore[L] := MaxStore[L];
                  DesiredLevel[L] := MinLevel[L]+(MaxStore[L]/Area[L]);
                END;
              GoToXY(42, (6+L*2)); Write(DesiredLevel[L]:8:2);
              GoToXY(51, (6+L*2)); Write(DesiredStore[L]:8:2);
            END;
        4 : BEGIN
              InputReal(DesiredStore[L], 8, 51, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
              DesiredLevel[L] := MinLevel[L]+(DesiredStore[L]/Area[L]);
              IF DesiredStore[L] < 0.0 THEN
                BEGIN
                  DesiredStore[L] := 0.0;
                  DesiredLevel[L] := MinLevel[L];
                END;
              IF DesiredStore[L] > (MaxStore[L]+200.0) THEN
                BEGIN
                  DesiredStore[L] := MaxStore[L];
                  DesiredLevel[L] := MinLevel[L]+(MaxStore[L]/Area[L]);
                END;
              GoToXY(42, (6+L*2)); Write(DesiredLevel[L]:8:2);
              GoToXY(51, (6+L*2)); Write(DesiredStore[L]:8:2);
            END;
        5 : InputReal(Inflow[L], 8, 60, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
      END;
      IF TC = ^F THEN
        IF L > 1 THEN L := L-1 ELSE L := 8;
      IF TC = ^A THEN
        IF L < 8 THEN L := L+1 ELSE L := 1;
      IF TC = ^M THEN
        IF I < 5 THEN I := I+1 ELSE I := 1;
    UNTIL TC = ^Z;
  END;



  PROCEDURE CostScreen;

  VAR
    I, L : Integer;
    TC : Char;
  BEGIN
    Title;
    WriteLn('                              Cost Functions');
    WriteLn('IMMMMMMMMMMMMMMMMMMMMMQMMMMMMMMMMQMMMMMMMMMMQMMMMMMMMMMQMMMMMMMMMM;');
    WriteLn(':  Station             3  Spill    3  Empty    3  Level1   3  Level2   :');
    WriteLn(':                      3           3           3           3           :');
    FOR I := 1 TO 8 DO
      BEGIN
        WriteLn('GDDDDDDDDDDDDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDDD6');
        WriteLn(':', StationName[I], '':21-Length(StationName[I]), '3',
        CS[I]:10:2, '3', CE[I]:10:2, '3', CL1[I]:10:2, '3', CL2[I]:10:2, ':');
```

```
721:      END;
722:    WriteLn('H[illegible]');
723:    I := 1;
724:    L := 1;
725:    REPEAT
726:      CASE I OF
727:        1 : InputReal(CS[L], 10, 24, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
728:        2 : InputReal(CE[L], 10, 35, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
729:        3 : InputReal(CL1[L], 10, 46, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
730:        4 : InputReal(CL2[L], 10, 57, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
731:      END;
732:      IF TC = ^F THEN
733:        IF L > 1 THEN L := L-1 ELSE L := 8;
734:      IF TC = ^A THEN
735:        IF L < 8 THEN L := L+1 ELSE L := 1;
736:      IF TC = ^M THEN
737:        IF I < 4 THEN I := I+1 ELSE I := 1;
738:    UNTIL TC = ^Z;
739:  END;
740:
741:  PROCEDURE LochDataScreen;
742:
743:  VAR
744:    I, L : Integer;
745:    TC : Char;
746:  BEGIN
747:    Title;
748:    WriteLn('                              Loch Data     ');
749:    WriteLn('H[illegible]');
750:    WriteLn(':  Loch              3  Max    3  Min    3  Min    3  Area   :');
751:    WriteLn(':                    3  Store  3  Store  3  Level  3         :');
752:    FOR I := 1 TO 8 DO
753:      BEGIN
754:        WriteLn('GDDDDDDDDDDDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDDD6');
755:        WriteLn(':', LochName[I], '':21-Length(LochName[I]), '3', MaxStore[I]:10:2, '3', MinStore[I]:10:2,
756:        MinLevel[I]:10:2, '3', Area[I]:10:2, ':');
757:      END;
758:    WriteLn('H[illegible]');
759:    I := 1;
760:    L := 1;
761:    REPEAT
762:      CASE I OF
763:        1 : InputStr(LochName[L], 20, 2, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
764:        2 : InputReal(MaxStore[L], 10, 24, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
765:        3 : InputReal(MinStore[L], 10, 35, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
766:        4 : InputReal(MinLevel[L], 10, 46, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
767:        5 : InputReal(Area[L], 10, 57, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
768:      END;
769:      IF TC = ^F THEN
770:        IF L > 1 THEN L := L-1 ELSE L := 8;
771:      IF TC = ^A THEN
772:        IF L < 8 THEN L := L+1 ELSE L := 1;
773:      IF TC = ^M THEN
774:        IF I < 5 THEN I := I+1 ELSE I := 1;
775:    UNTIL TC = ^Z;
776:  END;
777:
778:  PROCEDURE StationDataScreen;
779:
780:  VAR
```

```
    I, L : Integer;
    TC : Char;
  BEGIN
    Title;
    WriteLn('                              Station Data  ');
    WriteLn('IMMMMMMMMMMMMMMMMMMMMMQMMMMMMMMMMQMMMMMMMMMMQMMMMMMMMMMQMMMMMMMMMM;');
    WriteLn(':  Station            3   Opt    3   Max    3   Min    3 Turbine  :');
    WriteLn(':                     3  Power   3 Turbine  3 Turbine  3  Flow    :');
    FOR I := 1 TO 8 DO
      BEGIN
        WriteLn('GDDDDDDDDDDDDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDDD6');
        WriteLn(':', StationName[I], '':21-Length(StationName[I]), '3', Popt[I]:10:2, '3', MaxTurbine[I]:1
        MinTurbine[I]:10, '3', FlowPerUnit[I]:10:2, ':');
      END;
    WriteLn('HMMMMMMMMMMMMMMMMMMMMMOMMMMMMMMMMOMMMMMMMMMMOMMMMMMMMMMOMMMMMMMMMM<');
    I := 1;
    L := 1;
    REPEAT
      CASE I OF
        1 : InputStr(StationName[L], 20, 2, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
        2 : InputReal(Popt[L], 10, 24, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
        3 : InputInt(MaxTurbine[L], 10, 35, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
        4 : InputInt(MinTurbine[L], 10, 46, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
        5 : InputReal(FlowPerUnit[L], 10, 57, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
      END;
      IF TC = ^F THEN
        IF L > 1 THEN L := L-1 ELSE L := 8;
      IF TC = ^A THEN
        IF L < 8 THEN L := L+1 ELSE L := 1;
      IF TC = ^M THEN
        IF I < 5 THEN I := I+1 ELSE I := 1;
    UNTIL TC = ^Z;
  END;

  PROCEDURE DailySchedule;

  VAR
    TempDemand, TR, Cost, PrevCost, DemandMax,
    TotalPd, RequiredOutFlow, DemandMin : Real;
    PartialDerivatives, MaxOutFlow, MinOutFlow,
    DesSolution, ActualStore, HourlyFlow, Step : Vector;
    IntegralDemand : ARRAY[0..MaxTime] OF Real;
    SortDemand : ARRAY[1..25] OF Real;
    NoHours : Point;
    DemandMaxPtr, DemandMinPtr,
    TimeCount, Time, J, I, Station, Dam : Integer;
    Ch : Char;


    PROCEDURE CostAndPd(VAR Cost : Real;
                        VAR PD : Vector;
                        VAR Solution : Vector);

    VAR
      IntPow : Real;
      Dam, Station : Integer;
      NH1, NH2 : Integer;
    BEGIN
      FOR Dam := 1 TO MaxStation DO
        BEGIN
```

```
841:          IF Solution[Dam] > MaxOutFlow[Dam] THEN
842:            Solution[Dam] := MaxOutFlow[Dam];
843:          IF Solution[Dam] < (MinOutFlow[Dam]+1.5*HourlyFlow[Dam]) THEN
844:            Solution[Dam] := MinOutFlow[Dam];
845:        END;
846:      FOR Dam := 1 TO MaxStation DO
847:        BEGIN
848:          NoHours[Dam] := Trunc(Solution[Dam]/HourlyFlow[Dam]+0.5);
849:          ActualStore[Dam] := InitialStore[Dam]+Inflow[Dam]
850:          -Solution[Dam];
851:          FOR Station := 1 TO Dam DO
852:            BEGIN
853:              IF SpillDestination[Station] = Dam THEN
854:                IF Spillage[Station] > 0.0 THEN
855:                  ActualStore[Dam] := ActualStore[Dam]
856:                  +Spillage[Station];
857:              IF Destination[Station] = Dam THEN
858:                ActualStore[Dam] := ActualStore[Dam]+Solution[Station];
859:            END;
860:          IF ActualStore[Dam] > MaxStore[Dam] THEN
861:            BEGIN
862:              Spillage[Dam] := ActualStore[Dam]-MaxStore[Dam];
863:              ActualStore[Dam] := MaxStore[Dam];
864:            END
865:          ELSE
866:            Spillage[Dam] := 0.0;
867:        END;
868:      Cost := 0.0;
869:      FOR Dam := 1 TO MaxStation DO
870:        BEGIN
871:          NH1 := NoHours[Dam] DIV MaxTurbine[Dam];
872:          NH2 := NoHours[Dam] MOD MaxTurbine[Dam];
873:          Cost := Cost-IntegralDemand[NH1]*MaxTurbine[Dam]*Popt[Dam]*1000-
874:          SortDemand[NH1+1]*NH2*Popt[Dam]*1000;
875:          IF ActualStore[Dam] >= DesiredStore[Dam] THEN
876:            Cost := Cost+CL2[Dam]*(ActualStore[Dam]-DesiredStore[Dam]);
877:          IF ActualStore[Dam] < DesiredStore[Dam] THEN
878:            Cost := Cost+CL1[Dam]*(DesiredStore[Dam]-ActualStore[Dam]);
879:          IF Spillage[Dam] > 0.0 THEN
880:            Cost := Cost+CS[Dam]*Spillage[Dam]*5.0;
881:          IF ActualStore[Dam] < MinStore[Dam] THEN
882:            Cost := Cost-CE[Dam]*(MinStore[Dam]-ActualStore[Dam]);
883:        END;
884:      FOR Dam := 1 TO MaxStation DO
885:        BEGIN
886:          NH1 := (NoHours[Dam] DIV MaxTurbine[Dam])+1;
887:          PD[Dam] := -SortDemand[NH1]*Popt[Dam]*1000/HourlyFlow[Dam];
888:          IF ActualStore[Dam] > DesiredStore[Dam] THEN
889:            PD[Dam] := PD[Dam]-CL2[Dam]
890:          ELSE
891:            IF ActualStore[Dam] < DesiredStore[Dam] THEN
892:              PD[Dam] := PD[Dam]+CL1[Dam];
893:          IF Spillage[Dam] > 0.0 THEN
894:            PD[Dam] := PD[Dam]-CS[Dam]*5.0;
895:          IF (ActualStore[Dam] < MinStore[Dam]) THEN
896:            PD[Dam] := PD[Dam]+CE[Dam];
897:          IF Destination[Dam] <> 0 THEN
898:            BEGIN
899:              IF ActualStore[Destination[Dam]] < DesiredStore[Destination[Dam]] THEN
900:                PD[Dam] := PD[Dam]-CL1[Dam]/4.0
```

```
901:              ELSE
902:                IF ActualStore[Destination[Dam]] > DesiredStore[Destination[Dam]] THEN
903:                  PD[Dam] := PD[Dam]+CL2[Dam]/4.0;
904:              IF ActualStore[Destination[Dam]] < MinStore[Destination[Dam]] THEN
905:                BEGIN
906:                  PD[Dam] := PD[Dam]-CE[Destination[Dam]];
907:                  IF (ActualStore[Dam] < MinStore[Dam]) THEN
908:                    PD[Dam] := PD[Dam]-CE[Dam];
909:                END;
910:              IF Spillage[Destination[Dam]] > 0.0 THEN
911:                PD[Dam] := PD[Dam]+CS[Destination[Dam]]*5.0;
912:              IF ((Solution[Dam] = MinOutFlow[Dam]) AND (PD[Dam] > 0.0)) THEN
913:                PD[Dam] := 0.0;
914:              IF ((Solution[Dam] = MaxOutFlow[Dam]) AND (PD[Dam] < 0.0)) THEN
915:                PD[Dam] := 0.0;
916:            END;
917:        END;
918:    END;
919:
920:    PROCEDURE filedmp;
921:
922:      {This procedure is used in the program pwrplot to format a file for
923:      the DRGRAPH package using input data calculated in the main program.
924:      It may obviously be modified to suit other programs as required.
925:      }
926:
927:    TYPE
928:      FileName = STRING[13];
929:
930:    VAR
931:      I, J : Integer;
932:      outputfile : Text;
933:      name : FileName;
934:      outputvar : Real;
935:
936:
937:    BEGIN
938:      {INITIALISE THE FILE}
939:      WriteLn;
940:      WriteLn('Input name of file to be created for demand (*.dif).');
941:      ReadLn(name);
942:      Assign(outputfile, name);
943:      Rewrite(outputfile);
944:
945:      {MAIN PROCESSING STARTS HERE}
946:      Write(outputfile, 'TABLE');
947:      WriteLn(outputfile);
948:      Write(outputfile, '0,1');
949:      WriteLn(outputfile);
950:      Write(outputfile, '""');
951:      WriteLn(outputfile);
952:      Write(outputfile, 'DATA');
953:      WriteLn(outputfile);
954:      Write(outputfile, '0,0');
955:      WriteLn(outputfile);
956:      Write(outputfile, '""');
957:      WriteLn(outputfile);
958:      {INITIALISES SDF FILE}
959:
960:      Write(outputfile, '-1,0');
```

```
961:      WriteLn(outputfile);
962:      Write(outputfile, 'BOT');
963:      WriteLn(outputfile);
964:      FOR J := 1 TO 24 DO
965:        BEGIN
966:          outputvar := Demand[J];
967:          Write(outputfile, '0,', outputvar:10:5);
968:          WriteLn(outputfile);
969:          Write(outputfile, 'V');
970:          WriteLn(outputfile);
971:        END;                    {CURVE ELEMENT FOR LOOP}
972:      Write(outputfile, '-1,0');
973:      WriteLn(outputfile);
974:      Write(outputfile, 'BOT');
975:      WriteLn(outputfile);
976:      FOR J := 1 TO 24 DO
977:        BEGIN
978:          outputvar := SortDemand[J]+2.5;
979:          Write(outputfile, '0,', outputvar:10:5);
980:          WriteLn(outputfile);
981:          Write(outputfile, 'V');
982:          WriteLn(outputfile);
983:        END;                    {CURVE ELEMENT FOR LOOP}
984:      Write(outputfile, '-1,0');
985:      WriteLn(outputfile);
986:      Close(outputfile);
987:      WriteLn('Input name of file to be created for results (*.dat).');
988:      ReadLn(name);
989:      Assign(outputfile, name);
990:      Rewrite(outputfile);
991:      FOR I := 1 TO 8 DO
992:        WriteLn(outputfile, InitialStore[I]:10:2, ' ', DesiredStore[I]:10:2, ' ',
993:        Inflow[I]:10:2, ' ', Solution[I]:10:5);
994:      Close(outputfile);
995:
996:    END;                        {PROCEDURE FILEDMP}
997:
998:    PROCEDURE Display;
999:
1000:   VAR
1001:     Station : Integer;
1002:
1003:   BEGIN
1004:     WriteLn(I:4, ' ', Cost:15:5, ' ', TotalPd:15:5);
1005:     FOR Station := 1 TO MaxStation DO
1006:       WriteLn(Station:2, Solution[Station]:10:3, ' ',
1007:       PartialDerivatives[Station]:15:5, ' ', ActualStore[Station]:10:3, ' ',
1008:       Spillage[Station]:10:3, ' ', Step[Station]:10:3);
1009:     WriteLn;
1010:   END;
1011:
1012: BEGIN
1013:   ClrScr;
1014:   WriteLn(' Non Linear Programming Daily Solution  Start ...');
1015:   WriteLn;
1016:
1017:   { FIND THE TOTAL NATURAL INFLOW TO EACH DAM }
1018:
1019:   FOR Station := 1 TO MaxStation DO
1020:     BEGIN
```

```
      HourlyFlow[Station] := FlowPerUnit[Station]*Post[Station]/1000;
      MaxOutFlow[Station] := HourlyFlow[Station]*24*MaxTurbine[Station];
      MinOutFlow[Station] := HourlyFlow[Station]*24*MinTurbine[Station];
      Step[Station] := 0.0;
      Spillage[Station] := 0.0;
    END;
  { SORT THEN DEMAND COST CURVE INTO DESCENDING ORDER }

  FOR Time := 1 TO 24 DO SortDemand[Time] := Demand[Time]-2.5;
  SortDemand[25] := 0.0;
  FOR Time := 1 TO (MaxTime DIV 2) DO
    BEGIN
      DemandMaxPtr := 0;
      DemandMinPtr := 0;
      DemandMax := -99.9E15;
      DemandMin := 99.9E15;
      FOR TimeCount := Time TO (MaxTime+1-Time) DO
        BEGIN
          IF SortDemand[TimeCount] > DemandMax THEN
            BEGIN
              DemandMax := SortDemand[TimeCount];
              DemandMaxPtr := TimeCount;
            END;
          IF SortDemand[TimeCount] < DemandMin THEN
            BEGIN
              DemandMin := SortDemand[TimeCount];
              DemandMinPtr := TimeCount;
            END;
        END;
      IF DemandMaxPtr = DemandMinPtr THEN
        Time := MaxTime DIV 2
      ELSE
        BEGIN
          TempDemand := SortDemand[DemandMaxPtr];
          SortDemand[DemandMaxPtr] := SortDemand[Time];
          SortDemand[Time] := TempDemand;
          IF DemandMinPtr = Time THEN
            DemandMinPtr := DemandMaxPtr;
          TempDemand := SortDemand[DemandMinPtr];
          SortDemand[DemandMinPtr] := SortDemand[MaxTime+1-Time];
          SortDemand[MaxTime+1-Time] := TempDemand;
        END;
    END;

  { INTEGRATE THEN DEMAND COST CURVE }
  IntegralDemand[0] := 0.0;
  IntegralDemand[1] := SortDemand[1];
  FOR Time := 2 TO MaxTime DO
    BEGIN
      IntegralDemand[Time] := IntegralDemand[Time-1]+SortDemand[Time];
    END;
  { GET AN INITIAL FEASIBLE SOLUTION TO THE PROBLEM }

  FOR Station := 1 TO MaxStation DO
    BEGIN

      { NATURAL INFLOWS }

      RequiredOutFlow := InitialStore[Station]-DesiredStore[Station]
      +InFlow[Station];
```

```
1081:        { INFLOW FROM OTHER STATIONS }
1082:
1083:        FOR Dam := 1 TO Station DO
1084:          IF Destination[Dam] = Station THEN
1085:            RequiredOutFlow := RequiredOutFlow+Solution[Dam];
1086:        Solution[Station] := RequiredOutFlow;
1087:        IF Solution[Station] > MaxOutFlow[Station] THEN
1088:          Solution[Station] := MaxOutFlow[Station];
1089:        IF Solution[Station] < MinOutFlow[Station] THEN
1090:          Solution[Station] := MinOutFlow[Station];
1091:      END;
1092:    I := 1;
1093:    J := 1;
1094:    Cost := 9.99E20;
1095:    REPEAT
1096:      FOR Station := 1 TO MaxStation DO
1097:        Solution[Station] := Solution[Station]+Step[Station];
1098:      PrevCost := Cost;
1099:      CostAndPd(Cost, PartialDerivatives, Solution);
1100:      TotalPd := 0.0;
1101:      FOR Station := 1 TO MaxStation DO
1102:        BEGIN
1103:          TotalPd := TotalPd+Sqr(PartialDerivatives[Station]);
1104:        END;
1105:      TotalPd := Sqrt(TotalPd/(DCon/J));
1106:      FOR Station := 1 TO MaxStation DO
1107:        IF TotalPd > Limit THEN
1108:          BEGIN
1109:            TR := PartialDerivatives[Station]/TotalPd;
1110:            IF ((TR > 0.1) AND
1111:            (Solution[Station] > MinOutFlow[Station])) THEN
1112:              Step[Station] := -TR
1113:            ELSE
1114:              IF ((TR < -0.1) AND
1115:              (Solution[Station] < (MaxOutFlow[Station]))) THEN
1116:                Step[Station] := -TR
1117:              ELSE
1118:                Step[Station] := 0;
1119:          END
1120:        ELSE
1121:          Step[Station] := 0;
1122:      Display;
1123:      I := I+1;
1124:      IF Cost >= PrevCost THEN J := J+Trunc(DCon/4.00);
1125:    UNTIL (TotalPd < Limit) OR (I >= 16);
1126:    WriteLn(' Finished ');
1127:    Write(' Do you wish data dumped to file ? ');
1128:    ReadLn(Ch);
1129:    IF Ch IN ['Y', 'y'] THEN
1130:      filedmp;
1131:  END;
1132:
1133:  PROCEDURE MainMenu;
1134:
1135:  VAR
1136:    Command : Char;
1137:    ReDraw : Boolean;
1138:
1139:  BEGIN
1140:    ReDraw := True;
```

```
1141:     REPEAT
            IF ReDraw THEN
              BEGIN
1144:           Title;
                Say('%2Main menu');
                Say('%2%2%2Input File Name : ');
                Write(DataInFileName);
                Say('%2%2Output File Name : ');
                Write(DataOutFileName);
                Say('%2%2%2Loch Data Menu');
1151:           Say('%2%2Station Data Menu');
                Say('%2%2Demand');
                Say('%2%2Cost Function Menu');
1154:           Say('%2%2Results and Initialisation ');
                Say('%2%2%2Execute ');
                Say('%2%2%2Quit%2%2');
              END;                    { IF REDRAW }
1158:       Read(Kbd, Command);
            Command := UpCase(Command);
            ReDraw := True;
1161:       CASE Command OF
              'I' : BEGIN
                      Write('Enter Data Input File Name ? ');
1164:                 ReadLn(DataInFileName);
                      LoadData;
                      DataOutFileName := DataInFileName;
                    END;
              'O' : BEGIN
                      Write('Enter Data Output File Name ? ');
                      ReadLn(DataOutFileName);
1171:                 IF DataOutFileName <> '' THEN SaveData;
                    END;
              'L' : LochDataScreen;
1174:         'S' : StationDataScreen;
              'D' : GetData('Daily Demand Curve', Demand[1], 24, 0.0, 24.0, 0.0, 5.0);
              'C' : CostScreen;
              'R' : InitialScreen;
1179:         'E' : DailySchedule;
            END;                      { CASE COMMAND }
          UNTIL (Command = 'Q');
1181:     WriteLn;
        END;                          { MAINMENU }

1184: BEGIN
        (*****  CREATE NODE TREE ONTO HEAP  ************************************)

1187:   DataInFileName := 'Default.bd';
1188:   DataOutFileName := DataInFileName;
        LoadData;
        MainMenu;
1191:   IF DataOutFileName <> '' THEN
          SaveData;
1193: END.
```

```
 1: PROGRAM HuorlySchedulingProblem;
 2:
 3:   {$R+}
 4:
 5: CONST
 6:   DataSetSize = 7000;
 7:   MaxStation = 8;
 8:   MaxTime = 24;
 9:
10:   (*****************************************************************************)
11:   (*****  CONSTANTS WHICH DEFINE THE SIZE OF THE PROBLEM  ******************)
12:   (*****************************************************************************)
13:
14: TYPE
15:   Str80 = STRING[80];
16:   Str15 = STRING[15];
17:   Str10 = STRING[10];
18:   CharSet = SET OF Char;
19:   Vector = ARRAY[1..MaxStation] OF Real;
20:   Point = ARRAY[1..MaxStation] OF Integer;
21:   Buffer = STRING[80];
22:   Names = ARRAY[1..MaxStation] OF STRING[20];
23:   Flow = ARRAY[1..MaxTime] OF Real;
24:   FileName = STRING[20];
25:   Node = RECORD
26:            Cost,
27:            Store1,
28:            Store2,
29:            Spill,
30:            Empty : Real;
31:            U, Prevnode : Integer;
32:          END;
33:   DataSet = ARRAY[1..DataSetSize] OF Byte;
34:   DataRecord = ARRAY[1..DataSetSize] OF Byte;
35:
36:
37:
38: VAR
39:   (*****************************************************************************)
40:   (*****  CONSTANTS FOR THE OPTIMISATION PROBLEM ***************************)
41:   (*****************************************************************************)
42:
43:   (*****  LOCH INFORMATION: STORAGES IN MCF  *******************************)
44:   (*****                    LEVELS IN FEET   *******************************)
45:
46:   LochName : Names;
47:   MaxStore : Vector;
48:   MinStore : Vector;
49:   Area : Vector;
50:   MinLevel : Vector;
51:
52:   (*****  DEFINITION OF VALLEY INTERACTIONS  *******************************)
53:
54:   Destination : Point;
55:   SpillDestination : Point;
56:   SpillDelay : Vector;
57:
58:   (*****  STATION INFORMATION: FLOW IN MCF PER SET PER HOUR  ***************)
59:
60:   StationName : Names;
```

```
  FlowPerUnit : Vector;
  Optpower : Vector;
  MaxTurbine : Point;
  MinTurbine : Point;

  (*****  COST FUNCTIONS FOR OPTIMISATION  ***********************************)

  CP : Vector;
  CO : Vector;
  CS : Vector;
  CL2 : Vector;
  CE : Vector;

  (***** DAWN AND DUSK DEFINITIONS FOR STATION EIGHT FLOW CONTROL  *********)

  Dawn : Integer;
  Dusk : Integer;

  (*****  STEP SIZE FOR DAILY NLP PROBLEM  ***********************************)

  StepSize : Real;

  (*****  BASE TIME FOR START OF OPTIMISATION  *******************************)

  BaseTime : Integer;

  (*****************************************************************************)
  (*****  MAIN SYSTEM VARIABLE DEFINITION  ***********************************)
  (*****************************************************************************)

  Demand : Flow;
  A1, A2, A3 : Vector;
  InitialLevel, DesiredLevel : Vector;
  InitialStore, DesiredStore : Vector;
  Solution : Vector;

  Spillage, Storage, Inflow : ARRAY[1..8] OF Flow;
  InitialTurbine : Point;
  HourlySolution : ARRAY[1..8, 0..24] OF Integer;
  Buf : ARRAY[1..100] OF Real;
  Data : DataSet ABSOLUTE LochName;

  DataInFileName : FileName;
  DataOutFileName : FileName;
  I, J : Integer;

  NodePtr : ARRAY[0..MaxTime, 0..50] OF ^Node;


PROCEDURE LoadData;

  (*****************************************************************************)
  (*  THIS PROCEDURE LOADS START DATA FROM FILE.                               *)
  (*****************************************************************************)

VAR
  DataFile : FILE OF DataRecord;
  PP : RECORD CASE Integer OF
    1 : (P1000 : DataRecord);
    2 : (Parmz : DataSet);
```

```
121:         END;
122:
123:   BEGIN
124:     Assign(DataFile, DataInFileName);
125:
126:     {$I-}
127:
128:     Reset(DataFile);
129:
130:     {$I+}
131:
132:     IF IOResult <> 0 THEN
133:       BEGIN
134:         WriteLn(DataInFileName, ' not found !');
135:         Delay(500);
136:       END
137:     ELSE
138:       BEGIN
139:         Read(DataFile, PP.P1000);
140:         Data := PP.Parmz;
141:         Close(DataFile);
142:       END;
143:     SpillDestination[2] := 6;
144:     (***** END PROCEDURE LOADDATA  ********************************************)
145:
146:   END;
147:
148:
149:   PROCEDURE SaveData;
150:
151:     (*************************************************************************)
152:     (*  THIS PROCEDURE SAVES START DATA TO FILE.                             *)
153:     (*************************************************************************)
154:
155:   VAR
156:     DataFile : FILE OF DataRecord;
157:     Command : Char;
158:     PP : RECORD CASE Integer OF
159:       1 : (P1000 : DataRecord);
160:       2 : (Parmz : DataSet);
161:          END;
162:
163:   BEGIN
164:     Assign(DataFile, DataOutFileName);
165:
166:     {$I-}
167:
168:     Reset(DataFile);
169:
170:     {$I+}
171:
172:     IF IOResult <> 0 THEN
173:       BEGIN
174:         WriteLn('New File ', DataOutFileName, ' !');
175:         Delay(500);
176:         Command := 'Y';
177:       END
178:     ELSE
179:       BEGIN
180:         Close(DataFile);
```

```
181:        WriteLn(' Overwrite (DESTROY) old ', DataOutFileName);
182:        Read(Kbd, Command);
183:        Command := UpCase(Command);
184:      END;
185:    IF Command = 'Y' THEN
186:      BEGIN
187:        Assign(DataFile, DataOutFileName);
188:
189:        {$I-}
190:
191:        Rewrite(DataFile);
192:
193:        {$I+}
194:
195:        IF IOResult = 0 THEN
196:          BEGIN
197:            PP.Parmz := Data;
198:            Write(DataFile, PP.P1000);
199:            Close(DataFile);
200:          END
201:        ELSE
202:          WriteLn(' Unable to open file ', DataOutFileName);
203:      END;
204:    Delay(500);
205:
206:    (***** END PROCEDURE SAVEDATA  ********************************************)
207:
208:  END;
209:
210:  PROCEDURE Title;
211:
212:  BEGIN
213:    ClrScr;
214:    NormVideo;
215:    WriteLn('          Tummel Valley Optimisation Program Ver. 3.1   ');
216:    WriteLn;
217:    LowVideo;
218:  END;
219:
220:
221:  FUNCTION UpcaseStr(S : Str80) : Str80;
222:
223:    (*************************************************************************)
224:    (*  THIS FUNCTION RETURNS A STRING WHICH CONTAINS THE UPPER CASE STRING  *)
225:    (*  OF THE PARAMETER.                                                    *)
226:    (*************************************************************************)
227:
228:  VAR
229:    P : Integer;
230:  BEGIN
231:    FOR P := 1 TO Length(S) DO
232:      S[P] := UpCase(S[P]);
233:    UpcaseStr := S;
234:  END;
235:
236:
237:  FUNCTION ConstStr(C : Char; N : Integer) : Str80;
238:
239:    (*************************************************************************)
240:    (*  CONSTSTR RETURNS A STRING WITH N CHARACTERS OF VALUE C               *)
```

```
241:    (*****************************************************************************)
242:
243:  VAR
244:    S : STRING[80];
245:  BEGIN
246:    IF N < 0 THEN
247:      N := 0;
248:    S[0] := Chr(N);
249:    FillChar(S[1], N, C);
250:    ConstStr := S;
251:  END;
252:
253:
254:  PROCEDURE Beep;
255:
256:    (*****************************************************************************)
257:    (*  BEEP SOUNDS THE TERMINAL BELL OR BEEPER                                  *)
258:    (*****************************************************************************)
259:
260:  BEGIN
261:    Write(^G);
262:  END;
263:
264:
265:  PROCEDURE InputStr(VAR Ssss;
266:                     L, X, Y : Integer;
267:                     Term : CharSet;
268:                     VAR TC : Char;
269:                     Ins : Boolean);
270:
271:    (*****************************************************************************)
272:    (*  THIS PROCEDURE ALLOW THE READING AND EDITING OF ANY STRING.              *)
273:    (*****************************************************************************)
274:
275:  CONST
276:    UnderScore = '_';
277:  VAR
278:    S : STRING[255] ABSOLUTE Ssss;
279:    P : Integer;
280:    Ch : Char;
281:  BEGIN
282:    GoToXY(X, Y); Write(S, ConstStr(UnderScore, L-Length(S)));
283:    P := 0;
284:    REPEAT
285:      GoToXY(X+P, Y); Read(Kbd, Ch);
286:      CASE Ch OF
287:
288:        (*****  THE FOLLOWING IS ONLY VALID FOR THE IBM PC  ****************)
289:
290:        #27 : IF KeyPressed THEN
291:                BEGIN
292:                  Read(Kbd, Ch);
293:                  CASE Ch OF
294:                    #75 : IF P > 0 THEN
295:                            P := P-1
296:                          ELSE Beep;
297:                    #77 : IF P < Length(S) THEN
298:                            P := P+1
299:                          ELSE Beep;
300:                    #71 : P := 0;
```

```
301:                  #79 : P := Length(S);
302:                  #83 : IF P < Length(S) THEN
303:                          BEGIN
304:                            Delete(S, P+1, 1);
305:                            Write(Copy(S, P+1, L), UnderScore);
306:                          END;
307:                  #72 : Ch := ^F;
308:                  #80 : Ch := ^A;
309:                  #73 : Ch := ^E;
310:                  #81 : Ch := ^X;
311:                  #82 : Ch := ^V;
312:                ELSE Beep;
313:                END;
314:              END;
315:       #32..#126 : IF (Ins) OR (P = Length(S)) THEN
316:                     BEGIN
317:                       IF P < L THEN
318:                         BEGIN
319:                           IF Length(S) = L THEN
320:                             Delete(S, L, 1);
321:                           P := P+1;
322:                           Insert(Ch, S, P);
323:                           Write(Copy(S, P, L));
324:                         END
325:                       ELSE Beep;
326:                     END
327:                   ELSE
328:                     BEGIN
329:                       IF P < L THEN P := P+1;
330:                       S[P] := Ch;
331:                       Write(Copy(S, P, L));
332:                     END;
333:       ^S : IF P > 0 THEN
334:              P := P-1
335:            ELSE Beep;
336:       ^D : IF P < Length(S) THEN
337:              P := P+1
338:            ELSE Beep;
339:       ^G : IF P < Length(S) THEN
340:              BEGIN
341:                Delete(S, P+1, 1);
342:                Write(Copy(S, P+1, L), UnderScore);
343:              END;
344:       ^H, #127 : IF P > 0 THEN
345:                    BEGIN
346:                      Delete(S, P, 1);
347:                      Write(^H, Copy(S, P, L), UnderScore);
348:                      P := P-1;
349:                    END
350:                  ELSE Beep;
351:       ^Y : BEGIN
352:              Write(ConstStr(UnderScore, Length(S)-P));
353:              Delete(S, P+1, L);
354:            END;
355:     ELSE
356:       IF NOT(Ch IN Term) THEN Beep;
357:     END;                      {OF CASE}
358:   UNTIL Ch IN Term;
359:   P := Length(S);
360:   GoToXY(X+P, Y);
```

```
361:    Write('':L-P);
362:    TC := Ch;
363:
364:    (***** END PROCEDURE INPUTSTR ***************************************)
365:
366:  END;
367:
368:  PROCEDURE InputReal(VAR A : Real;
369:                      L, X, Y : Integer;
370:                      Term : CharSet;
371:                      VAR TC : Char;
372:                      Ins : Boolean);
373:  VAR
374:    Rstr : STRING[10];
375:    Temp : Real;
376:    Test : Integer;
377:
378:  BEGIN
379:    Str(A:10:2, Rstr);
380:    InputStr(Rstr, L, X, Y, Term, TC, Ins);
381:    Test := 1;
382:    WHILE Test <= Length(Rstr) DO
383:      IF Rstr[Test] = ' ' THEN
384:        Delete(Rstr, Test, 1)
385:      ELSE
386:        Test := Test+1;
387:    Val(Rstr, Temp, Test);
388:    IF Test = 0 THEN
389:      BEGIN
390:        A := Temp;
391:      END;
392:    GoToXY(X, Y);
393:    Write(A:10:2);
394:  END;
395:
396:  PROCEDURE InputInt(VAR A : Integer;
397:                     L, X, Y : Integer;
398:                     Term : CharSet;
399:                     VAR TC : Char;
400:                     Ins : Boolean);
401:  VAR
402:    Rstr : STRING[10];
403:    Temp : Integer;
404:    Test : Integer;
405:
406:  BEGIN
407:    Str(A:10, Rstr);
408:    InputStr(Rstr, L, X, Y, Term, TC, Ins);
409:    Test := 1;
410:    WHILE Test <= Length(Rstr) DO
411:      IF Rstr[Test] = ' ' THEN
412:        Delete(Rstr, Test, 1)
413:      ELSE
414:        Test := Test+1;
415:    Val(Rstr, Temp, Test);
416:    IF Test = 0 THEN
417:      BEGIN
418:        A := Temp;
419:      END;
420:    GoToXY(X, Y);
```

```
421:     Write(A:10);
422:   END;
423:
424:
425:   PROCEDURE Mouse(VAR M1, M2, M3, M4 : Integer);
426:
427:     (*************************************************************************)
428:     (*  THIS PROCEDURE GETS THE POSITION AND STATUS OF THE MOUSE.           *)
429:     (*************************************************************************)
430:
431:   VAR
432:     Regs : RECORD
433:              Ax, Bx, Cx, Dx, Bp, Si, Di, Ds, Es, Flags : Integer;
434:            END;
435:
436:   BEGIN
437:     WITH Regs DO
438:       BEGIN
439:         Ax := M1;
440:         Bx := M2;
441:         Cx := M3;
442:         Dx := M4;
443:       END;
444:     Intr(51, Regs);
445:     WITH Regs DO
446:       BEGIN
447:         M1 := Ax;
448:         M2 := Bx;
449:         M3 := Cx;
450:         M4 := Dx;
451:       END;
452:
453:     (***** END PROCEDURE MOUSE  ***************************************************)
454:
455:   END;
456:
457:
458:   {$R-}
459:
460:   PROCEDURE GetData(T : Str80; VAR Data; No : Integer; Lx, Rx, By, Ty : Real);
461:
462:     (*************************************************************************)
463:     (*  THIS PROCEDURE USES THE MOUSE TO INPUT CURVES.                       *)
464:     (*************************************************************************)
465:
466:   VAR
467:     M1, M2, M3, M4, Py, Oy, Y, Ny, Ox, X : Integer;
468:     Quit : Boolean;
469:     Ch : Char;
470:     Sx, I, Mx : Integer;
471:     Sy : Real;
472:     D : ARRAY[1..2] OF Real ABSOLUTE Data;
473:
474:   BEGIN
475:     M1 := 0;
476:     M2 := 0;
477:     M3 := 0;
478:     M4 := 0;
479:     Mouse(M1, M2, M3, M4);
480:
```

```
481:    (*****  TEST IF MOUSE INSTALLED  ***************************************)
482:
483:    IF M1 = 0 THEN
484:      BEGIN
485:        ClrScr;
486:        WriteLn(' mouse not installed !!');
487:        Delay(500);
488:      END
489:    ELSE
490:      BEGIN
491:        Sx := Trunc(540/No);
492:        IF Sx > 0 THEN
493:          BEGIN
494:            Mx := Sx*No+41;
495:            HiRes;
496:            Write(T);
497:            Draw(39, 19, Mx, 19, 1);
498:            Draw(Mx, 19, Mx, 181, 1);
499:            Draw(Mx, 181, 39, 181, 1);
500:            Draw(39, 181, 39, 19, 1);
501:            Sy := 160/(Ty-By);
502:            FOR I := 0 TO No DO
503:              Draw(((I*Sx)+40), 181, ((I*Sx)+40), 185, 1);
504:            FOR I := 0 TO 10 DO
505:              BEGIN
506:                Draw(35, ((I*16)+20), 39, ((I*16)+20), 1);
507:                Draw(Mx, ((I*16)+20), Mx+4, ((I*16)+20), 1);
508:              END;
509:            FOR I := 1 TO No-1 DO
510:              BEGIN
511:                Oy := 180-Trunc((D[I]-By)*Sy);
512:                Y := 180-Trunc((D[I+1]-By)*Sy);
513:                Draw((((I-1)*Sx)+40), Oy, ((I*Sx)+40), Oy, 1);
514:                Draw(((I*Sx)+40), Oy, ((I*Sx)+40), Y, 1);
515:              END;
516:            Draw(Mx-Sx, Y, Mx, Y, 1);
517:            M1 := 1;
518:            Mouse(M1, M2, M3, M4);
519:            Quit := False;
520:            M1 := 3;
521:            REPEAT
522:              Mouse(M1, M2, M3, M4);
523:              IF M2 <> 0 THEN
524:                BEGIN
525:                  BEGIN
526:                    I := ((M3-40) DIV Sx)+1;
527:                    IF I < 1 THEN I := 1;
528:                    IF I > No THEN I := No;
529:                    Y := 180-Trunc((D[I]-By)*Sy);
530:                    GoToXY(50, 1); Write((I/No*(Rx-Lx)+Lx):5:2, ' ', D[I]:5:2);
531:                    X := (I*Sx)+40;
532:                    Ox := 2;
533:                    Mouse(Ox, M2, M3, M4);
534:                    IF I > 1 THEN
535:                      BEGIN
536:                        Oy := 180-Trunc((D[I-1]-By)*Sy);
537:                        Draw(X-Sx, Oy, X-Sx, Y, 0);
538:                      END
539:                    ELSE
540:                      Oy := -1;
```

```
541:                    IF I < No THEN
542:                      BEGIN
543:                        Ny := 180-Trunc((D[I+1]-By)*Sy);
544:                        Draw(X, Ny, X, Y, 0);
545:                      END
546:                    ELSE
547:                      Ny := -1;
548:                    Draw(X, Y, X-Sx, Y, 0);
549:                    Py := Y;
550:                    REPEAT
551:                      Mouse(M1, M2, M3, M4);
552:                      Y := M4;
553:                      IF Y < 20 THEN Y := 20;
554:                      IF Y > 180 THEN Y := 180;
555:                      D[I] := (180-Y)/Sy+By;
556:                      GoToXY(50, 1); Write((I/No*(Rx-Lx)+Lx):5:2, ' ', D[I]:5:2);
557:                      Draw(X, Py, X-Sx, Py, 0);
558:                      Draw(X, Y, X-Sx, Y, 1);
559:                      Py := Y;
560:                    UNTIL M2 = 0;
561:                    Delay(200);
562:                    IF Oy <> -1 THEN Draw(X-Sx, Oy, X-Sx, Y, 1);
563:                    IF Ny <> -1 THEN Draw(X, Ny, X, Y, 1);
564:                    Ox := I;
565:                    Mouse(Ox, M2, M3, M4);
566:                  END;
567:                END;
568:              IF KeyPressed THEN
569:                BEGIN
570:                  Read(Kbd, Ch);
571:                  IF Ch = #27 THEN
572:                    Quit := True;
573:                END;
574:            UNTIL Quit;
575:          END;
576:      END;
577:    TextMode;
578:  END;
579:
580:  {$R+}
581:
582:
583:  PROCEDURE Say(S : Str80);
584:
585:  VAR
586:    I : Integer;
587:
588:  BEGIN
589:    I := 1;
590:    WHILE I <= Length(S) DO
591:      BEGIN
592:        IF S[I] <> '%' THEN
593:          Write(S[I])
594:        ELSE
595:          IF S[I+1] = '%' THEN
596:            BEGIN
597:              WriteLn;
598:              I := I+1;
599:            END
600:          ELSE
```

```
601:            IF S[I+1] = '!' THEN
602:              BEGIN
603:                NormVideo;    { IF YOUR SCREEN DOESN'T HAVE HIGH/LOW VIDEO, }
604:                Write(UpCase(S[I+2])); { REPLACE THESE 3 LINES WITH:                }
605:                LowVideo;     {       WRITE(S[I+2],')');                    }
606:                I := I+2;
607:              END
608:            ELSE Write('%');
609:          I := I+1;
610:        END;                    { WHILE I<=LENGTH(S) }
611:    END;                        { SAY }
612:
613:    {$I HOURLY.INC}
614)   {$R+}
615)
616)   PROCEDURE HourlySchedule(St, Fn : Integer);
617)
618)
619)     (*****************************************************************************)
620)     (*  THIS PROCEDURE USES DYNAMIC PROGRAMMING TO SCHEDULE THE FIRST        *)
621)     (*  SIX STATIONS OF THE TUMMEL VALLEY. THEN FLOW OF WATER DOWN THEN      *)
622)     (*  VALLEY IS CONSIDERED AND SPILLAGE FROM EACH DAM IS ALSO CONSIDERED.  *)
623)     (*  THIS PROCEDURE USES THE RESULTS FROM THEN NON-LINEAR PROGRAMMING     *)
624)     (*  PROCEDURE TO RESTRICT THE SEARCH SPACE FOR SPEED.                    *)
625)     (*****************************************************************************)
626)
627)   VAR
628)     Time, Station, Dam, Turbine, I,
629)     Tstart, Tstop : Integer;
630)     CurrentCost, CurrentSpill, CurrentEmpty,
631)     NextStore, CurrentStore : Real;
632)     CurrentTurbine : Integer;
633)
634)     CurrentNodePtr, NextNodePtr : ^Node;
635)
636)     HourlyFlow : Vector;
637)     HoursOn : Point;
638)
639)     StationInflow : Flow;
640)
641)   BEGIN
642)
643)     { FIND THE TOTAL NATURAL INFLOW TO EACH DAM }
644)
645)     FOR Station := 1 TO MaxStation DO
646)       BEGIN
647)         HourlyFlow[Station] := FlowPerUnit[Station]*Optpower[Station]/1000;
648)         HoursOn[Station] := Trunc(Solution[Station]/HourlyFlow[Station]);
649)       END;
650)
651)
652)
653)
654)     (***** FOR ALL STATIONS  ************************************************)
655)
656)     FOR Station := St TO Fn DO
657)
658)       (***** TEST FOR ALL SETS OFF ALL THE TIME  ***************************)
659)       (***** NO POINT IN OPTIMISATION WITH NO OPTIONS  **********************)
660)
```

```
661)      IF HoursOn[Station] = 0 THEN
662)        FOR Time := 1 TO MaxTime DO
663)
664)          (*****  SET RESULT TO ALL OFF  ************************************)
665)
666)          HourlySolution[Station, Time] := 0
667)      ELSE
668)
669)        (***** TEST FOR ALL SETS ON ALL THE TIME  ***************************)
670)        (***** NO POINT IN OPTIMISATION WITH NO OPTIONS  *********************)
671)
672)        IF HoursOn[Station] = (MaxTurbine[Station]*MaxTime) THEN
673)          FOR Time := 1 TO MaxTime DO
674)
675)            (***** SET RESULT TO ALL ON ************************************)
676)
677)            HourlySolution[Station, Time] := MaxTurbine[Station]
678)        ELSE
679)
680)          (*****  ACTUAL OPTIMISATION  ************************************)
681)
682)          BEGIN
683)
684)            Write(StationName[Station], ' Scheduling ...');
685)
686)            (*****  GET NATURAL INFLOW FOR CURRENT STATION  *****************)
687)
688)            StationInflow := Inflow[Station];
689)
690)            (***** IF APPLICABLE ADD ON OTHER STATION OUTPUT AND SPILL  ****)
691)
692)            FOR Dam := 1 TO Station DO
693)              BEGIN
694)                IF Destination[Dam] = Station THEN
695)                  FOR Time := 1 TO MaxTime DO
696)
697)                    (***** ADD OTHER STATIONS OUTPUT  ***********************)
698)
699)                    StationInflow[Time] := StationInflow[Time]+HourlyFlow[Dam]*HourlySolution[Dam, Time];
700)                IF SpillDestination[Dam] = Station THEN
701)                  FOR Time := 1 TO MaxTime DO
702)
703)                    (***** ADD OTHER LOCH SPILLAGE  *************************)
704)
705)                    StationInflow[Time] := StationInflow[Time]+Spillage[Dam, Time];
706)              END;
707)
708)            (***** SET ALL NODES TO BE USED TO NOT REACHED  *****************)
709)
710)            FOR Time := 0 TO MaxTime DO
711)              FOR I := 0 TO HoursOn[Station] DO
712)                WITH NodePtr[Time, I]^ DO
713)                  Prevnode := -1;
714)
715)            (***** INITIALISE FIRST NODE TO INITIAL DATA  *******************)
716)
717)            WITH NodePtr[0, 0]^ DO
718)              BEGIN
719)                Cost := 0.0;
720)                Prevnode := 0;
```

```
721)            U := InitialTurbine[Station];
722)            Store1 := InitialStore[Station];
723)            Spill := 0.0;
724)            Empty := 0.0;
725)          END;
726)
727)        (***** FOR ALL TIME  ********************************************)
728)
729)        FOR Time := 0 TO 23 DO
730)          BEGIN
731)
732)            (*****  FIND LOWER LIMIT FOR POSSIBLE CURRENT NODES  *******)
733)
734)            Tstart := HoursOn[Station]-((MaxTime-Time)*MaxTurbine[Station]);
735)            IF Tstart < 0 THEN Tstart := 0;
736)
737)            (*****  FIND UPPER LIMIT FOR POSSIBLE CURRENT NODES  *******)
738)
739)            Tstop := Time*MaxTurbine[Station];
740)            IF Tstop > HoursOn[Station] THEN Tstop := HoursOn[Station];
741)
742)            (*****  FOR EACH NODE IN RESTRICTED SET  ********************)
743)            (*****  SET IS RESTRICTED BY FINAL DESIRED LEVEL  ***********)
744)
745)            FOR I := Tstart TO Tstop DO
746)              BEGIN
747)
748)                (*****  GET CURRENT NODE POINTER  ***********************)
749)
750)                CurrentNodePtr := NodePtr[Time, I];
751)
752)                (*****  FOR EACH POSSIBLE SET OUTPUT  *******************)
753)
754)                FOR Turbine := 0 TO MaxTurbine[Station] DO
755)                  BEGIN
756)
757)                    (*****  DISPLAY SEARCH PROGRESS IN GRAPHICAL FORM  *)
758)
759)
760)                    (*****  GET NEXT NODE POINTER  **********************)
761)
762)                    NextNodePtr := NodePtr[Time+1, I+Turbine];
763)
764)                    (*****  GET DATA FROM CURRENT NODE TO TEMP VARS  ***)
765)
766)                    WITH CurrentNodePtr^ DO
767)                      BEGIN
768)                        CurrentSpill := 0.0;
769)                        CurrentEmpty := 0.0;
770)                        CurrentCost := Cost;
771)                        CurrentStore := Store1;
772)                        CurrentTurbine := U;
773)                      END;
774)
775)                    (*****  ADD COST FOR CHANGE IN OUTPUT  **************)
776)
777)                    CurrentCost := CurrentCost+CO[Station]*5.0*
778)                    Abs(Turbine*1.0-CurrentTurbine);
779)
780)                    (*****  ADD COST FOR POWER GENERATED  ***************)
```

```
781)
782)      CurrentCost := CurrentCost-Optpower[Station]*1000*
783)      Demand[((Time+BaseTime) MOD MaxTime)+1]*
784)      Turbine;
785)
786)      (*****  CALCULATE LOCH STORAGE FOR NEXT NODE  ******)
787)
788)      NextStore := CurrentStore+StationInflow[Time+1]-
789)      HourlyFlow[Station]*Turbine;
790)
791)      (*****  TEST FOR SPILL AND CALCULATE  ***************)
792)
793)      IF NextStore > MaxStore[Station] THEN
794)        BEGIN
795)          CurrentSpill := NextStore-MaxStore[Station];
796)          NextStore := MaxStore[Station];
797)        END
798)      ELSE
799)        CurrentSpill := 0.0;
800)
801)      (*****  TEST FOR EMPTY AND CALCULATE  ***************)
802)
803)      IF NextStore < MinStore[Station] THEN
804)        BEGIN
805)          CurrentEmpty := MinStore[Station]-NextStore;
806)          NextStore := MinStore[Station];
807)        END
808)      ELSE
809)        CurrentEmpty := 0.0;
810)
811)      (*****  ADD COST OF SPILL OR EMPTY  *****************)
812)
813)      CurrentCost := CurrentCost+CS[Station]*CurrentSpill+
814)      CE[Station]*CurrentEmpty;
815)
816)      WITH NextNodePtr^ DO
817)        BEGIN
818)
819)          (*****  NEXT NODE IS TO BE UPDATED IF COST IS  *)
820)          (*****  LESS OR IF THE NEXT NODE HAS NOT BEEN  *)
821)          (*****  RREACHED BEFORE  ************************)
822)
823)          IF (CurrentCost < Cost) OR (Prevnode = -1) THEN
824)            BEGIN
825)
826)              (*****  UPDATE NEXT NODE  *******************)
827)
828)              Cost := CurrentCost;
829)              Store1 := NextStore;
830)              Prevnode := I;
831)              Spill := CurrentSpill;
832)              Empty := CurrentEmpty;
833)              U := Turbine;
834)            END;
835)        END;
836)
837)    (*****  END FOR EACH POSSIBLE TURBINE  ***************)
838)
839)  END;
840)
```

```
841)                  (*****  END FOR EACH CURRENT NODE  ************************)
842)
843)                END;
844)
845)              (*****  END FOR ALL TIME  ************************************)
846)
847)            END;
848)
849)          (*****  GET THE LAST POINT OF THE SOLUTION SET  *****************)
850)          I := HoursOn[Station];
851)          WITH NodePtr[24, I]^ DO
852)            BEGIN
853)              Storage[Station, 24] := Store1;
854)              Spillage[Station, 24] := Spill;
855)              I := Prevnode;
856)              HourlySolution[Station, 24] := U;
857)            END;
858)
859)          (*****  TRACE SOLUTION SET THROUGH NODE DOWN TIME  **************)
860)
861)          FOR Time := 23 DOWNTO 1 DO
862)            WITH NodePtr[Time, I]^ DO
863)              BEGIN
864)                Storage[Station, Time] := Store1;
865)                Spillage[Station, Time] := Spill;
866)                HourlySolution[Station, Time] := U;
867)
868)                I := Prevnode;
869)              END;
870)
871)          WriteLn(' Finished !');
872)          (*****  END ACTUAL OPTIMISATION  AND FOR EACH STATION  ***********)
873)
874)        END;
875)
876)
877)    (*****  END PROCEDURE SCHEDULE  ***************************************)
878)
879)  END;
880)
881:  {$I PITLOCH3.INC}
882)  {$R+}
883)
884)
885)  PROCEDURE PitlochrySchedule(VAR Same : Boolean);
886)
887)    (*****************************************************************************)
888)    (*  THIS PROCEDURE USES DYNAMIC PROGRAMMING TO SCHEDULE PITLOCHRY WITH  *)
889)    (*  LOCH FASKALLY STORAGE AS A STATE VARIABLE.                          *)
890)    (*****************************************************************************)
891)
892)
893)  VAR
894)    Time, Station, Turbine, I, J, Inc, Rate, T : Integer;
895)
896)    dt, Sp : Real;
897)
898)    Init : Node;
899)
```

```
900)     HourlyFlow : Vector;
901)
902)     StationInflow : Flow;
903)
904)     TempSolution : ARRAY[0..25] OF Integer;
905)
906)     PROCEDURE Simulate(Start, Finish : Integer; VAR Result : Node);
907)
908)     VAR
909)       T : Integer;
910)
911)     BEGIN
912)       WITH Result DO
913)         FOR T := Start TO Finish DO
914)           BEGIN
915)             Store1 := Store1-TempSolution[T]*HourlyFlow[8]+
916)             StationInflow[T];
917)             IF Store1 > MaxStore[8] THEN
918)               BEGIN
919)                 Spill := Spill+Store1-MaxStore[8];
920)                 Store1 := MaxStore[8];
921)               END;
922)             IF Store1 < 0.0 THEN
923)               BEGIN
924)                 Empty := Empty-Store1;
925)                 Store1 := 0.0;
926)               END;
927)           END;
928)     END;                          (* END PROCEDURE SIMULATE *)
929)
930)     PROCEDURE Opt(Str, Fin, StartNode, StartTime : Integer; Day : Boolean);
931)
932)     VAR
933)       Turbine, T, Next : Integer;
934)       NextNode : Node;
935)
936)     BEGIN
937)       FOR Turbine := MinTurbine[8] TO MaxTurbine[8] DO
938)         BEGIN
939)           FOR T := Str TO Fin DO
940)             TempSolution[T] := Turbine;
941)           NextNode := NodePtr[StartTime, StartNode]^;
942)           Simulate(Str, Fin, NextNode);
943)           WITH NextNode DO
944)             BEGIN
945)               Cost := Cost+CS[8]*Spill;
946)               Cost := Cost+CE[8]*Empty;
947)               IF Day THEN
948)                 Cost := Cost-4.0*(Fin-Str)*Optpower[8]*1000*Turbine;
949)               Next := Trunc(Store1/0.55+0.5);
950)             END;
951)           WITH NodePtr[StartTime+1, Next]^ DO
952)             IF (NextNode.Cost < Cost) OR (Prevnode = -1) THEN
953)               BEGIN
954)                 NodePtr[StartTime+1, Next]^ := NextNode;
955)                 Prevnode := StartNode;
956)                 U := Turbine;
957)               END;
958)         END;
959)     END;
```

```
960)
961)
962)    PROCEDURE Optimise(Start, Break, Finish : Integer;
963)                       Day : Boolean;
964)                       Init : Node);
965)
966)    VAR
967)      StartNode, Time, I, J, Next : Integer;
968)
969)    BEGIN
970)      StartNode := Trunc(Init.Store1/0.55+0.5);
971)
972)      (*****  INITIALISE NODE AS NOT REACHED  ********************************)
973)
974)      FOR Time := 1 TO 2 DO
975)        FOR I := 0 TO 100 DO
976)          WITH NodePtr[Time, I]^ DO
977)            Prevnode := -1;
978)
979)      (***** INITIALISE START NODE TO INITIAL DATA  ************************)
980)
981)      NodePtr[0, StartNode]^ := Init;
982)
983)      (***** FIRST OPTIMISATION  ***************************************)
984)
985)      Opt(Start, Break, StartNode, 0, Day);
986)      (***** SECOND OPTIMISATION  **************************************)
987)
988)      FOR I := 0 TO 100 DO
989)        BEGIN
990)          IF NodePtr[1, I]^.Prevnode <> -1 THEN
991)            Opt(Break+1, Finish, I, 1, NOT Day);
992)        END;
993)      Next := Trunc(DesiredStore[0]/0.55+0.5);
994)      I := Next+1;
995)      J := Next-1;
996)      REPEAT
997)        IF I > 0 THEN I := I-1;
998)        IF J < 100 THEN J := J+1;
999)      UNTIL (NodePtr[2, I]^.Prevnode <> -1) OR
1000)     (NodePtr[2, J]^.Prevnode <> -1);
1001)     IF (NodePtr[2, I]^.Prevnode <> -1) THEN Next := I;
1002)     IF (NodePtr[2, J]^.Prevnode <> -1) THEN Next := J;
1003)
1004)     WITH NodePtr[2, Next]^ DO
1005)       BEGIN
1006)         FOR I := Finish DOWNTO (Break+1) DO
1007)           TempSolution[I] := U;
1008)         Next := Prevnode;
1009)       END;
1010)
1011)     WITH NodePtr[1, Next]^ DO
1012)       BEGIN
1013)         FOR I := Break DOWNTO Start DO
1014)           TempSolution[I] := U;
1015)       END;
1016)
1017)   END;                    (***** END PROCEDURE OPTIMISE ********************************)
1018)
1019)   BEGIN
```

```
1020)
1021)    { FIND THE TOTAL NATURAL INFLOW TO EACH DAM }
1022)
1023)    FOR Station := 6 TO MaxStation DO
1024)      BEGIN
1025)        HourlyFlow[Station] := FlowPerUnit[Station]*Optpower[Station]/1000;
1026)      END;
1027)
1028)    (*****  SET STATION TO BE PITLOCHRY  ***********************************)
1029)
1030)    Write(' Pitlochry Schedule  Start ...');
1031)    Station := 8;
1032)
1033)    (*****  GET NATURAL INFLOW  ********************************************)
1034)
1035)    StationInflow := Inflow[Station];
1036)
1037)    (***** ADD OUTFLOW FROM 7 AND SPILL FROM 7 AND 6  **********************)
1038)
1039)    FOR Time := 1 TO MaxTime DO
1040)      StationInflow[Time] := StationInflow[Time]+HourlyFlow[7]*HourlySolution[7, Time]+
1041)      Spillage[7, Time]+Spillage[6, Time];
1042)
1043)    WITH Init DO
1044)      BEGIN
1045)        Cost := 0.0;
1046)        Prevnode := 0;
1047)        U := InitialTurbine[Station];
1048)        Store1 := InitialStore[Station];
1049)        Spill := 0.0;
1050)        Empty := 0.0;
1051)      END;
1052)
1053)    IF (BaseTime = 7) THEN
1054)      Optimise(1, 18, 24, True, Init)
1055)    ELSE
1056)      IF (BaseTime >= 24) OR (BaseTime <= 0) THEN
1057)        Optimise(1, 7, 24, False, Init)
1058)      ELSE
1059)        IF (BaseTime >= 1) AND (BaseTime <= 5) THEN
1060)          BEGIN
1061)            FOR Time := 1 TO (6-BaseTime) DO
1062)              TempSolution[Time] := Init.U;
1063)            Simulate(1, (6-BaseTime), Init);
1064)            WITH Init DO
1065)              BEGIN
1066)                IF Spill > 0.0 THEN
1067)                  BEGIN
1068)                    IF NOT Same THEN
1069)                      BEGIN
1070)                        WriteLn(' Emergency Spillage Correction !');
1071)                        T := 6-BaseTime;
1072)                        Rate := 0;
1073)                        REPEAT
1074)                          Rate := Rate+1;
1075)                          dt := (MaxTurbine[8]-Init.U)/Rate;
1076)                          IF dt > T*1.0 THEN dt := T;
1077)                          Sp := Rate*dt*(T-(0.5*dt));
1078)                        UNTIL (Sp > Spill) OR (Rate > MaxTurbine[8]);
1079)                        REPEAT
```

```
1080)                    dt := dt-1.0;
                         Sp := Rate*dt*(T-(0.5*T));
                       UNTIL (Sp < Spill) OR (dt < 0.0);
1083)                  IF dt < 0.0 THEN
                         dt := 0.0
                       ELSE
                         dt := dt+1.0;
                       TempSolution[1] := Init.U+Rate;
                       FOR T := 2 TO (6-BaseTime) DO
                         IF T*1.0 < dt THEN
1090)                      BEGIN
                             TempSolution[T] := TempSolution[T-1]+Rate;
1093)                        IF TempSolution[T] > MaxTurbine[8] THEN
                               TempSolution[T] := MaxTurbine[8];
                           END
                         ELSE
1096)                      TempSolution[T] := TempSolution[T-1];
                       Simulate(1, (6-BaseTime), Init);
                     END;
1099)              END;
1100)            IF Empty > 0.0 THEN
                   BEGIN
                     IF NOT Same THEN
1103)                BEGIN
                       WriteLn(' Emergency Empty Correction !');
                       T := 6-BaseTime;
1106)                  Rate := 0;
                       REPEAT
                         Rate := Rate-1;
                         dt := (MinTurbine[8]-Init.U)/Rate;
1110)                    IF dt > T*1.0 THEN dt := T;
                         Sp := Rate*dt*(T-(0.5*dt));
                       UNTIL (Sp > Empty) OR (Rate < MinTurbine[8]);
1113)                  REPEAT
                         dt := dt-1.0;
                         Sp := Rate*dt*(T-(0.5*T));
1116)                  UNTIL (Sp < Empty) OR (dt < 0.0);
                       IF dt < 0.0 THEN
                         dt := 0.0
                       ELSE
1120)                    dt := dt+1.0;
                       TempSolution[1] := Init.U+Rate;
                       FOR T := 2 TO (6-BaseTime) DO
1123)                    IF T*1.0 < dt THEN
                           BEGIN
                             TempSolution[T] := TempSolution[T-1]+Rate;
1126)                        IF TempSolution[T] > MinTurbine[8] THEN
                               TempSolution[T] := MinTurbine[8];
                           END
                         ELSE
1130)                      TempSolution[T] := TempSolution[T-1];
                       Simulate(1, (6-BaseTime), Init);
                     END;
1133)              END;
                 END;
               Optimise((7-BaseTime), (24-BaseTime), 24, False, Init);
1136)        END
1137)      ELSE
1138)        IF (BaseTime >= 8) AND (BaseTime <= 23) THEN
1139)          BEGIN
```

```
1140)          FOR Time := 1 TO (24-BaseTime) DO
1141)            TempSolution[Time] := Init.U;
1142)          Simulate(1, (24-BaseTime), Init);
1143)          WITH Init DO
1144)            BEGIN
1145)              IF Spill > 0.0 THEN
1146)                BEGIN
1147)                  IF NOT Same THEN
1148)                    BEGIN
1149)                      WriteLn(' Emergency Spillage Correction !');
1150)                      T := 24-BaseTime;
1151)                      Rate := 0;
1152)                      REPEAT
1153)                        Rate := Rate+1;
1154)                        dt := (MaxTurbine[8]-Init.U)/Rate;
1155)                        IF dt > T*1.0 THEN dt := T;
1156)                        Sp := Rate*dt*(T-(0.5*dt));
1157)                      UNTIL (Sp > Spill) OR (Rate > MaxTurbine[8]);
1158)                      REPEAT
1159)                        dt := dt-1.0;
1160)                        Sp := Rate*dt*(T-(0.5*T));
1161)                      UNTIL (Sp < Spill) OR (dt < 0.0);
1162)                      IF dt < 0.0 THEN
1163)                        dt := 0.0
1164)                      ELSE
1165)                        dt := dt+1.0;
1166)                      TempSolution[1] := Init.U+Rate;
1167)                      FOR T := 2 TO (24-BaseTime) DO
1168)                        IF T*1.0 < dt THEN
1169)                          BEGIN
1170)                            TempSolution[T] := TempSolution[T-1]+Rate;
1171)                            IF TempSolution[T] > MaxTurbine[8] THEN
1172)                              TempSolution[T] := MaxTurbine[8];
1173)                          END
1174)                        ELSE
1175)                          TempSolution[T] := TempSolution[T-1];
1176)                      Simulate(1, (24-BaseTime), Init);
1177)                    END;
1178)                END;
1179)              IF Empty > 0.0 THEN
1180)                BEGIN
1181)                  IF NOT Same THEN
1182)                    BEGIN
1183)                      WriteLn(' Emergency Empty Correction !');
1184)                      T := 24-BaseTime;
1185)                      Rate := 0;
1186)                      REPEAT
1187)                        Rate := Rate-1;
1188)                        dt := (MinTurbine[8]-Init.U)/Rate;
1189)                        IF dt > T*1.0 THEN dt := T;
1190)                        Sp := Rate*dt*(T-(0.5*dt));
1191)                      UNTIL (Sp > Empty) OR (Rate < MinTurbine[8]);
1192)                      REPEAT
1193)                        dt := dt-1.0;
1194)                        Sp := Rate*dt*(T-(0.5*T));
1195)                      UNTIL (Sp < Empty) OR (dt < 0.0);
1196)                      IF dt < 0.0 THEN
1197)                        dt := 0.0
1198)                      ELSE
1199)                        dt := dt+1.0;
```

```
                        TempSolution[1] := Init.U+Rate;
                        FOR T := 2 TO (24-BaseTime) DO
                          IF T*1.0 < dt THEN
                            BEGIN
                              TempSolution[T] := TempSolution[T-1]+Rate;
                              IF TempSolution[T] > MinTurbine[8] THEN
                                TempSolution[T] := MinTurbine[8];
                            END
                          ELSE
                            TempSolution[T] := TempSolution[T-1];
                        Simulate(1, (24-BaseTime), Init);
                      END;
                  END;
              END;
            Optimise((25-BaseTime), (31-BaseTime), 24, False, Init);
          END;
    WITH Init DO
      BEGIN
        Cost := 0.0;
        Prevnode := 0;
        U := InitialTurbine[Station];
        Store1 := InitialStore[Station];
        Spill := 0.0;
        Empty := 0.0;
      END;

    WITH Init DO
      FOR Time := 1 TO 24 DO
        BEGIN
          Store1 := Store1-TempSolution[Time]*HourlyFlow[8]+StationInflow[Time];
          IF Store1 > MaxStore[8] THEN
            BEGIN
              Spillage[8, Time] := Store1-MaxStore[8];
              Store1 := MaxStore[8];
            END
          ELSE
            Spillage[8, Time] := 0.0;
          IF Store1 < 0.0 THEN
            BEGIN
              Empty := -Store1;
              Store1 := 0.0;
            END
          ELSE
            Empty := 0.0;
          IF HourlySolution[8, Time] <> TempSolution[Time] THEN Same := False;
          HourlySolution[8, Time] := TempSolution[Time];
          Storage[8, Time] := Store1;
        END;
    WriteLn(' Finished !');
  END;




  PROCEDURE PitlochrySchedule1;

    (*************************************************************************)
    (*  THIS PROCEDURE USES DYNAMIC PROGRAMMING TO SCHEDULE PITLOCHRY WITH  *)
    (*  LOCH FASKALLY STORAGE AS A STATE VARIABLE.                          *)
```

```
(*****************************************************************************)



VAR
  Same : Boolean;

BEGIN
  ClrScr;
  Same := True;
  PitlochryScheduleSame);
END;

{$I CLUNIE.INC}
{$R+}

PROCEDURE ClunieSchedule(VAR Same : Boolean);

  (*****************************************************************************)
  (*  THIS PROCEDURE USES DYNAMIC PROGRAMMING TO SCHEDULE CLUNIE.              *)
  (*  IT TAKES ACCOUNT OF THE INTER-ACTION WITH THE RUN-OF-RIVER               *)
  (*  STATION AT PITLOCHRY BY OPTIMISING WITH RESPECT TO NOT ONLY ITS          *)
  (*  OWN LEVEL BUT ALSO THAT OF LOCH FASKNALLY.                               *)
  (*****************************************************************************)

VAR
  Time, Station, Dam, Turbine, I,
  Tstart, Tstop : Integer;
  CurrentCost, CurrentSpill1, CurrentSpill2, CurrentEmpty1,
  CurrentEmpty2, NextStore1, NextStore2, CurrentStore1,
  CurrentStore2 : Real;
  CurrentTurbine : Integer;
  StationInflow : Flow;
  HoursOn : Integer;
  HourlyFlow : Vector;
  CurrentNodePtr, NextNodePtr : ^Node;
BEGIN
  FOR I := 1 TO 8 DO
    BEGIN
      HourlyFlow[I] := FlowPerUnit[I]*Optpower[I]/1000;
    END;
  HoursOn := Trunc(Solution[7]/HourlyFlow[7]);
  Station := 7;
  Write(' Clunie Schedule Start ...');
  (***** TEST FOR ALL SETS OFF ALL THE TIME  ***************************)
  (***** NO POINT IN OPTIMISATION WITH NO OPTIONS  **********************)

  IF HoursOn = 0 THEN
    FOR Time := 1 TO MaxTime DO

      (*****  SET RESULT TO ALL OFF  ********************************)

      HourlySolution[Station, Time] := 0
  ELSE

    (***** TEST FOR ALL SETS ON ALL THE TIME  ***************************)
    (***** NO POINT IN OPTIMISATION WITH NO OPTIONS  *********************)

    IF HoursOn = (MaxTurbine[Station]*MaxTime) THEN
      FOR Time := 1 TO MaxTime DO
```

```
1319)
1320)       (***** SET RESULT TO ALL ON ************************************)
1321)
1322)       HourlySolution[Station, Time] := MaxTurbine[Station]
1323)     ELSE
1324)
1325)       (*****  ACTUAL OPTIMISATION  ************************************)
1326)
1327)       BEGIN
1328)
1329)         (*****  GET NATURAL INFLOW FOR CURRENT STATION  ****************)
1330)
1331)         StationInflow := Inflow[Station];
1332)
1333)         (***** IF APPLICABLE ADD ON OTHER STATION OUTPUT AND SPILL  ****)
1334)
1335)         FOR Dam := 1 TO Station DO
1336)           BEGIN
1337)             IF Destination[Dam] = Station THEN
1338)               FOR Time := 1 TO MaxTime DO
1339)
1340)                 (***** ADD OTHER STATIONS OUTPUT  ***********************)
1341)
1342)                 StationInflow[Time] := StationInflow[Time]+HourlyFlow[Dam]*HourlySolution[Dam, Time];
1343)             IF Spill[Destination[Dam] = Station THEN
1344)               FOR Time := 1 TO MaxTime DO
1345)
1346)                 (***** ADD OTHER LOCH SPILLAGE  *************************)
1347)
1348)                 StationInflow[Time] := StationInflow[Time]+Spillage[Dam, Time];
1349)           END;
1350)
1351)         (***** SET ALL NODES TO BE USED TO NOT REACHED  ****************)
1352)
1353)         FOR Time := 0 TO MaxTime DO
1354)           FOR I := 0 TO HoursOn DO
1355)             WITH NodePtr[Time, I]^ DO
1356)               Prevnode := -1;
1357)
1358)         (***** INITIALISE FIRST NODE TO INITIAL DATA  ******************)
1359)
1360)         WITH NodePtr[0, 0]^ DO
1361)           BEGIN
1362)             Cost := 0.0;
1363)             Prevnode := 0;
1364)             U := InitialTurbine[Station];
1365)             Store1 := InitialStore[Station];
1366)             Store2 := InitialStore[0];
1367)             Spill := 0.0;
1368)             Empty := 0.0;
1369)           END;
1370)
1371)         (***** FOR ALL TIME  ********************************************)
1372)
1373)         FOR Time := 0 TO 23 DO
1374)           BEGIN
1375)
1376)             (*****  FIND LOWER LIMIT FOR POSSIBLE CURRENT NODES  *******)
1377)
1378)             Tstart := HoursOn-((MaxTime-Time)*MaxTurbine[Station]);
```

```
IF Tstart < 0 THEN Tstart := 0;

(*****  FIND UPPER LIMIT FOR POSSIBLE CURRENT NODES  *******)

Tstop := Time*MaxTurbine[Station];
IF Tstop > HoursOn THEN Tstop := HoursOn;

(*****  FOR EACH NODE IN RESTRICTED SET  ******************)
(*****  SET IS RESTRICTED BY FINAL DESIRED LEVEL  **********)

FOR I := Tstart TO Tstop DO
  BEGIN

    (*****  GET CURRENT NODE POINTER  ***********************)

    CurrentNodePtr := NodePtr[Time, I];

    (*****  FOR EACH POSSIBLE SET OUTPUT  ******************)

    FOR Turbine := 0 TO MaxTurbine[Station] DO
      BEGIN

        (*****  DISPLAY SEARCH PROGRESS IN GRAPHICAL FORM  *)


        (*****  GET NEXT NODE POINTER  **********************)

        NextNodePtr := NodePtr[Time+1, I+Turbine];

        (*****  GET DATA FROM CURRENT NODE TO TEMP VARS  ***)

        WITH CurrentNodePtr^ DO
          BEGIN
            CurrentSpill1 := 0.0;
            CurrentEmpty1 := 0.0;
            CurrentCost := Cost;
            CurrentStore1 := Store1;
            CurrentStore2 := Store2;
            CurrentTurbine := U;
          END;

        (*****  ADD COST FOR CHANGE IN OUTPUT  **************)

        CurrentCost := CurrentCost+COI[Station]*5.0*
        Abs(Turbine*1.0-CurrentTurbine);

        (*****  ADD COST FOR POWER GENERATED  ***************)

        CurrentCost := CurrentCost-Optpower[Station]*1000*
        Demand[((Time+BaseTime) MOD MaxTime)+1]*
        Turbine;

        (*****  CALCULATE LOCH STORAGE FOR NEXT NODE  ******)
        NextStore1 := CurrentStore1+StationInflow[Time+1]-
        HourlyFlow[Station]*Turbine;
        NextStore2 := CurrentStore2+HourlyFlow[Station]*Turbine-
        HourlySolution[8, Time+1]*HourlyFlow[8]+
        Spillage[7, Time+1]+Spillage[6, Time+1]
        +Inflow[8, Time+1];
        (*****  TEST FOR SPILL AND CALCULATE  ***************)
```

```
1439)
1440)
1441)
1442)
1443)
1444)
1445)
1446)
1447)
1448)
1449)
1450)
1451)
1452)
1453)
1454)
1455)
1456)
1457)
1458)
1459)
1460)
1461)
1462)
1463)
1464)
1465)
1466)
1467)
1468)
1469)
1470)
1471)
1472)
1473)
1474)
1475)
1476)
1477)
1478)
1479)
1480)
1481)
1482)
1483)
1484)
1485)
1486)
1487)
1488)
1489)
1490)
1491)
1492)
1493)
1494)
1495)
1496)
1497)
1498)

  IF NextStore1 > MaxStore[Station] THEN
    BEGIN
      CurrentSpill1 := NextStore1-MaxStore[Station];
      NextStore1 := MaxStore[Station];
    END
  ELSE
    CurrentSpill1 := 0.0;

  (***** TEST FOR EMPTY AND CALCULATE ***************)

  IF NextStore1 < MinStore[Station] THEN
    BEGIN
      CurrentEmpty1 := MinStore[Station]-NextStore1;
      NextStore1 := MinStore[Station];
    END
  ELSE
    CurrentEmpty1 := 0.0;

  (***** TEST FOR SPILL AND CALCULATE ***************)

  IF NextStore2 > MaxStore[8] THEN
    BEGIN
      CurrentSpill2 := NextStore2-MaxStore[8];
      NextStore2 := MaxStore[8];
    END
  ELSE
    CurrentSpill2 := 0.0;

  (***** TEST FOR EMPTY AND CALCULATE ***************)

  IF NextStore2 < MinStore[8] THEN
    BEGIN
      CurrentEmpty2 := MinStore[8]-NextStore2;
      NextStore2 := MinStore[8];
    END
  ELSE
    CurrentEmpty2 := 0.0;
  (***** ADD COST OF SPILL OR EMPTY ******************)

  CurrentCost := CurrentCost+CS[Station]*CurrentSpill1+
  CE[Station]*CurrentEmpty1+
  500.0*CS[8]*CurrentSpill2+
  500.0*CE[8]*CurrentEmpty2;

  WITH NextNodePtr^ DO
    BEGIN

      (***** NEXT NODE IS TO BE UPDATED IF COST IS  *)
      (***** LESS OR IF THE NEXT NODE HAS NOT BEEN  *)
      (***** RREACHED BEFORE  ************************)

      IF (CurrentCost < Cost) OR (Prevnode = -1) THEN
        BEGIN

          (***** UPDATE NEXT NODE  ******************)

          Cost := CurrentCost;
          Store1 := NextStore1;
          Store2 := NextStore2;
```

```
                    Prevnode := I;
                    Spill := CurrentSpill1;
                    Empty := CurrentEmpty1;
                    U := Turbine;
                  END;
                END;

              (*****  END FOR EACH POSSIBLE TURBINE  ***************)

            END;

          (*****  END FOR EACH CURRENT NODE  ***********************)

        END;

      (*****  END FOR ALL TIME  **********************************)

    END;

    (*****  GET THE LAST POINT OF THE SOLUTION SET  ****************)

    Same := True;

    I := HoursOn;
    WITH NodePtr[24, I]^ DO
      BEGIN
        Storage[Station, 24] := Store1;
        Spillage[Station, 24] := Spill;
        I := Prevnode;
        IF HourlySolution[Station, 24] <> U THEN Same := False;
       HourlySolution[Station, 24] := U;
      END;

    (*****  TRACE SOLUTION SET THROUGH NODE DOWN TIME  *************)

    FOR Time := 23 DOWNTO 1 DO
      WITH NodePtr[Time, I]^ DO
        BEGIN
          Storage[Station, Time] := Store1;
          Spillage[Station, Time] := Spill;
          IF HourlySolution[Station, Time] <> U THEN Same := False;
          HourlySolution[Station, Time] := U;
          (*****  DISPLAY SOLUTION IN GRAPHICAL FORM  *************)

          I := Prevnode;
        END;

   WriteLn(' Finished !');


   (*****  END ACTUAL OPTIMISATION  AND FOR EACH STATION  **********)

  END;

 (*****  END PROCEDURE SCHEDULE  ************************************)

END;

{$I PLOT.INC}
PROCEDURE UsrOut(Ch : Char);
```

```
1558)  VAR
1559)    B : Byte;
1560)  BEGIN
1561)    REPEAT B := Port[$3FD] AND $21 UNTIL (B = $20) OR (B = $21);
1562)    IF B = $21 THEN
1563)      BEGIN
1564)        B := Port[$3F8] AND $7F;
1565)        IF B = $11 THEN
1566)          BEGIN
1567)            REPEAT
1568)              REPEAT B := Port[$3FD] AND 1 UNTIL B <> 0;
1569)              B := Port[$3F8] AND $7F;
1570)            UNTIL B = $11;
1571)          END;
1572)      END;
1573)    Port[$3F8] := Ord(Ch);
1574)  END;
1575)
1576)  PROCEDURE Output;
1577)  CONST
1578)    Scale : ARRAY[1..8] OF Real = (4.0, 4.0, 46.0, 8.0, 38.0, 76.0, 64.0, 16.0);
1579)    Base1 : ARRAY[1..8] OF Integer = (11876, 11150, 9341, 8964, 7457, 4518, 2831, 1353);
1580)    EOT = #3;
1581)  VAR
1582)    Top,
1583)    I, X, Y, J, K : Integer;
1584)    Ch : Char;
1585)    Mtit, Pax, Stit : STRING[80];
1586)    Sax : ARRAY[1..8] OF STRING[15];
1587)    Total : ARRAY[1..24] OF Real;
1588)  BEGIN
1589)    ClrScr;
1590)    Write(' Press enter when plotter ready ');
1591)    ReadLn(Ch);
1592)    UserOutPtr := Ofs(UserOut);
1593)    Write(User, 'IN;', Chr(27), '.128;;17:', Chr(27), '.N;19:,PA;');
1594)    Write(User, 'P80;RO90;IP;IW;VS40;PU;LT;SP1;');
1595)    Write(User, 'S10,10,0.31;CS0;SS;PA2818,1353;PD;PA8281,1353;PU;');
1596)    Write(User, 'PA1764,1152;LB0', EOT, 'PA8173,1152;LB24', EOT);
1597)    FOR I := 0 TO 24 DO
1598)      BEGIN
1599)        X := Trunc(6263.0/24.0*I)+2818;
1600)        Write(User, 'PA', X, ',1353;PD;PA', X, ',1303;PU;');
1601)      END;
1602)
1603)    Mtit := 'Tunnel Valley Hourly Scheduling Results';
1604)    Write(' Enter main title ? ');
1605)    ReadLn(Mtit);
1606)    Pax := 'Time (hours)';
1607)    Stit := '';
1608)    Sax[1] := 'CUM';
1609)    Sax[2] := 'L.Er';
1610)    Sax[3] := 'RAN';
1611)    Sax[4] := 'GAUP';
1612)    Sax[5] := 'TUM';
1613)    Sax[6] := 'ERR';
1614)    Sax[7] := 'CLUN';
1615)    Sax[8] := 'PITS';
1616)    IF Pax <> '' THEN
1617)      BEGIN
```

```
1618)         X := ((6262-Trunc(Length(Pax)*1.8*40.38/0.67)) DIV 2)+2018;
1619)         Write(Usr, 'PA', X, ',953;LB', Pax, EOT);
1620)       END;
1621)     IF Stit <> '' THEN
1622)       BEGIN
1623)         X := ((6262-Trunc(Length(Stit)*1.8*40.38/0.67)) DIV 2)+2018;
1624)         Write(Usr, 'PA', X, ',12006;LB', Stit, EOT);
1625)       END;
1626)     IF Mtit <> '' THEN
1627)       BEGIN
1628)         X := ((6262-Trunc(Length(Mtit)*2.8*40.38/0.67)) DIV 2)+2018;
1629)         Write(Usr, 'SI0.28,0.47;PA', X, ',12330;LB', Mtit, EOT);
1630)         Write(Usr, 'SI0.18,0.31;');
1631)       END;
1632)     Top := 11602;
1633)     FOR I := 1 TO 24 DO Total[I] := 0.0;
1634)     FOR I := 1 TO 8 DO
1635)       BEGIN
1636)         Write(Usr, 'IP2018,', Base1[I], ',8281,', Top, ';');
1637)         WriteLn(Base1[I]:7, Top:7, ' ', Scale[I]:5:3);
1638)         Top := Base1[I];
1639)         Write(Usr, 'SC0,24,0,', (Scale[I]+2.0):4:1, ';');
1640)         J := Trunc(Scale[I]/2.0)-1;
1641)         FOR K := 0 TO J DO
1642)           BEGIN
1643)             Write(Usr, 'PA0,', (K*2), ';PD;PA0,', ((K+1)*2), ';');
1644)             IF ((K+1) MOD 5) = 0 THEN
1645)               Write(Usr, 'PA0.4,', ((K+1)*2), ';PU;')
1646)             ELSE
1647)               Write(Usr, 'PA0.2,', ((K+1)*2), ';PU;');
1648)           END;
1649)         Write(Usr, 'PA0,0;PD;PA24.0,0;PU;');
1650)         FOR K := 0 TO J DO
1651)           BEGIN
1652)             Write(Usr, 'PA24.0,', (K*2), ';PD;PA24.0,', ((K+1)*2), ';');
1653)             IF ((K+1) MOD 5) = 0 THEN
1654)               BEGIN
1655)                 Write(Usr, 'PA23.6,', ((K+1)*2), ';PU;');
1656)                 Write(Usr, 'PA24.2,', (K*2), ';LB', (K+1)*2, EOT);
1657)               END
1658)             ELSE
1659)               Write(Usr, 'PA23.8,', ((K+1)*2), ';PU;');
1660)           END;
1661)         Write(Usr, 'PA-2.0,', (Scale[I]/2.0):4:1, ';LB', Sax[I], EOT, ';');
1662)         Write(Usr, 'PA0,', (HourlySolution[I, 1]*Optpower[I]):4:1, ';PD;');
1663)         FOR K := 1 TO 24 DO
1664)           BEGIN
1665)             Write(Usr, 'PA', (K-1), ',', (HourlySolution[I, K]*Optpower[I]):4:1, ';');
1666)             Write(Usr, 'PA', K, ',', (HourlySolution[I, K]*Optpower[I]):4:1, ';');
1667)             Total[K] := Total[K]+(HourlySolution[I, K]*Optpower[I]);
1668)           END;
1669)         Write(Usr, 'PU;');
1670)       END;
1671)     Write(' Press enter when plotter ready !');
1672)     ReadLn(Ch);
1673)     Write(Usr, 'SC;PU;LT;SP1;SI0.18,0.31;CS0;SS;PA2018,1353;PD;PA8281,1353;');
1674)     Write(Usr, 'PU;PA2018,1353;PD;PA2018,5801;PU;');
1675)     Write(Usr, 'PA1964,1152;LB0', EOT, 'PA8173,1152;LB24', EOT);
1676)     FOR I := 0 TO 24 DO
1677)       BEGIN
```

```
678)        X := Trunc(6263.0/24.0*I)+2018;
679)        Write(Usr, 'PA', X, ',1353;PD;PA', X, ',1403;PU;');
680)      END;
681)    FOR I := 0 TO 5 DO
682)      BEGIN
683)        X := Trunc(4443.0/5.0*I)+1353;
684)        Write(Usr, 'PA2018,', X, ';PD;PA2100,', X, ',;PU;');
685)        Write(Usr, 'PA1418,', X-50, ';LB', (I*50.0):5:1, EOT);
686)      END;
687)    Mtit := 'Total Power For Group';
688)    Write('Enter total power title ? ');
689)    ReadLn(Mtit);
690)    Pax := 'Time (hours)';
691)    Sax[1] := 'Power (Mw)';
692)    Stit := '';
693)    IF Sax[1] <> '' THEN Write(Usr, 'PA1470,5958;LB', Sax[1], EOT);
694)    IF Pax <> '' THEN
695)      BEGIN
696)        X := ((6262-Trunc(Length(Pax)*1.8*40.38/0.67)) DIV 2)+2018;
697)        Write(Usr, 'PA', X, ',953;LB', Pax, EOT);
698)      END;
699)    IF Stit <> '' THEN
700)      BEGIN
701)        X := ((6262-Trunc(Length(Stit)*1.8*40.38/0.67)) DIV 2)+2018;
702)        Write(Usr, 'PA', X, ',6240;LB', Stit, EOT);
703)      END;
704)    IF Mtit <> '' THEN
705)      BEGIN
706)        X := ((6262-Trunc(Length(Mtit)*2.8*40.38/0.67)) DIV 2)+2018;
707)        Write(Usr, 'SI0.28,0.47;PA', X, ',6554;LB', Mtit, EOT);
708)      END;
709)    Write(Usr, 'IP2018,1353,8281,5801;SC0,24,0,250;');
710)    Write(Usr, 'PA0,', (Total[1]):4:1, ';PD;');
711)    FOR K := 1 TO 24 DO
712)      BEGIN
713)        Write(Usr, 'PA', (K-1), ',', (Total[K]):4:1, ';');
714)        Write(Usr, 'PA', K, ',', (Total[K]):4:1, ';');
715)      END;
716)    Write(Usr, 'PU;');
717)    Write(Usr, 'SC;PU;LT;SP1;SI0.18,0.31;CS0;SS;PA2018,7353;PD;PA8281,7353;');
718)    Write(Usr, 'PU;PA2018,7353;PD;PA2018,11801;PU;');
719)    Write(Usr, 'PA1964,7152;LB0', EOT, 'PA8173,7152;LB24', EOT);
720)    FOR I := 0 TO 24 DO
721)      BEGIN
722)        X := Trunc(6263.0/24.0*I)+2018;
723)        Write(Usr, 'PA', X, ',7353;PD;PA', X, ',7403;PU;');
724)      END;
725)    FOR I := 0 TO 5 DO
726)      BEGIN
727)        X := Trunc(4443.0/5.0*I)+7353;
728)        Write(Usr, 'PA2018,', X, ';PD;PA2100,', X, ',;PU;');
729)        Write(Usr, 'PA1518,', X-50, ';LB', (I*1.0):4:2, EOT);
730)      END;
731)    Mtit := 'Hourly Price Demand Curve';
732)    Pax := 'Time (hours)';
733)    Sax[1] := 'Unit Cost (Pence)';
734)    Stit := '';
735)    IF Sax[1] <> '' THEN Write(Usr, 'PA1470,11958;LB', Sax[1], EOT);
736)    IF Pax <> '' THEN
737)      BEGIN
```

```
1738)       X := ((6262-Trunc(Length(Pax)*1.8*40.38/0.67)) DIV 2)+2018;
1739)       Write(Usr, 'PA', X, ',6953;LB', Pax, EOT);
1740)     END;
1741)   IF Stit <> '' THEN
1742)     BEGIN
1743)       X := ((6262-Trunc(Length(Stit)*1.8*40.38/0.67)) DIV 2)+2018;
1744)       Write(Usr, 'PA', X, ',12240;LB', Stit, EOT);
1745)     END;
1746)   IF Mtit <> '' THEN
1747)     BEGIN
1748)       X := ((6262-Trunc(Length(Mtit)*2.8*40.38/0.67)) DIV 2)+2018;
1749)       Write(Usr, 'SI0.28,0.47;PA', X, ',12554;LB', Mtit, EOT);
1750)     END;
1751)   Write(Usr, 'IP2018,7353,8281,11801;SC0,24,0,5;');
1752)   Write(Usr, 'PA0,', (Demand[1]):4:1, ';PD;');
1753)   FOR K := 1 TO 24 DO
1754)     BEGIN
1755)       Write(Usr, 'PA', (K-1), ',', (Demand[K]):4:1, ';');
1756)       Write(Usr, 'PA', K, ',', (Demand[K]):4:1, ';');
1757)     END;
1758)   Write(Usr, 'PU;');
1759) END;
1760:
1761:
1762: PROCEDURE InflowMenu;
1763:
1764: VAR
1765:   Command : Char;
1766:   ReDraw : Boolean;
1767:
1768: BEGIN
1769:   ReDraw := True;
1770:   REPEAT
1771:     IF ReDraw THEN
1772:       BEGIN
1773:         Title;
1774:         Say('%2inflow menu');
1775:         Say('%2%2Loch %!Seilich Inflow%2');
1776:         Say('Loch %!Garry Inflow%2');
1777:         Say('%!Loch Ericht Inflow%2');
1778:         Say('Loch %!Eigheach Inflow%2');
1779:         Say('Loch %!Rannoch Inflow%2');
1780:         Say('Loch Err%!ochty Inflow%2');
1781:         Say('Loch T%!ummel Inflow%2');
1782:         Say('Loch %!Faskally Inflow%2');
1783:         Say('%2%!Quit%2%2');
1784:       END;                  { IF REDRAW }
1785:     Read(Kbd, Command);
1786:     Command := UpCase(Command);
1787:     ReDraw := True;
1788:     CASE Command OF
1789:       'S' : GetData('Loch Seilch Inflow Curve ', Inflow[1, 1], 24, 0.0, 24.0, 0.0, 10.0);
1790:       'G' : GetData('Loch Garry Inflow Curve ', Inflow[2, 1], 24, 0.0, 24.0, 0.0, 10.0);
1791:       'L' : GetData('Loch Ericht Inflow Curve ', Inflow[3, 1], 24, 0.0, 24.0, 0.0, 10.0);
1792:       'E' : GetData('Loch Eigheach Inflow Curve ', Inflow[4, 1], 24, 0.0, 24.0, 0.0, 10.0);
1793:       'R' : GetData('Loch Rannoch Inflow Curve ', Inflow[5, 1], 24, 0.0, 24.0, 0.0, 10.0);
1794:       'O' : GetData('Loch Errochty Inflow Curve ', Inflow[6, 1], 24, 0.0, 24.0, 0.0, 10.0);
1795:       'U' : GetData('Loch Tummel Inflow Curve ', Inflow[7, 1], 24, 0.0, 24.0, 0.0, 10.0);
1796:       'F' : GetData('Loch Faskally Inflow Curve ', Inflow[8, 1], 24, 0.0, 24.0, 0.0, 10.0);
1797:     END;                    { CASE COMMAND }
```

```
1797:    UNTIL (Command = 'Q');
1798:    WriteLn;
1799:  END;                          { INFLOWMENU }
1800:
1801:  PROCEDURE StorageMenu;
1802:
1803:  VAR
1804:    Command : Char;
1805:    ReDraw : Boolean;
1806:
1807:  BEGIN
1808:    ReDraw := True;
1809:    REPEAT
1810:      IF ReDraw THEN
1811:        BEGIN
1812:          Title;
1813:          Say('%2Storage menu');
1814:          Say('%2%2Loch %!Seilich Storage%2');
1815:          Say('Loch %!Garry Storage%2');
1816:          Say('%!Loch Ericht Storage%2');
1817:          Say('Loch %!Eigheach Storage%2');
1818:          Say('Loch %!Rannoch Storage%2');
1819:          Say('Loch Err%!ochty Storage%2');
1820:          Say('Loch T%!ummel Storage%2');
1821:          Say('Loch %!Faskally Storage%2');
1822:          Say('%2%!Quit%2%2');
1823:        END;                    { IF REDRAW }
1824:      Read(Kbd, Command);
1825:      Command := UpCase(Command);
1826:      ReDraw := True;
1827:      CASE Command OF
1828:        'S' : GetData('Loch Seilch Storage Curve ', Storage[1, 1], 24, 0.0, 24.0, MinStore[1], MaxStore[1]);
1829:        'G' : GetData('Loch Garry Storage Curve ', Storage[2, 1], 24, 0.0, 24.0, MinStore[2], MaxStore[2]);
1830:        'L' : GetData('Loch Ericht Storage Curve ', Storage[3, 1], 24, 0.0, 24.0, MinStore[3], MaxStore[3]);
1831:        'E' : GetData('Loch Eigheach Storage Curve ', Storage[4, 1], 24, 0.0, 24.0, MinStore[4], MaxStore[4]);
1832:        'R' : GetData('Loch Rannoch Storage Curve ', Storage[5, 1], 24, 0.0, 24.0, MinStore[5], MaxStore[5]);
1833:        'O' : GetData('Loch Errochty Storage Curve ', Storage[6, 1], 24, 0.0, 24.0, MinStore[6], MaxStore[6]);
1834:        'U' : GetData('Loch Tummel Storage Curve ', Storage[7, 1], 24, 0.0, 24.0, MinStore[7], MaxStore[7]);
1835:        'F' : GetData('Loch Faskally Storage Curve ', Storage[8, 1], 24, 0.0, 24.0, MinStore[8], MaxStore[8]);
1836:      END;                      { CASE COMMAND }
1837:    UNTIL (Command = 'Q');
1838:    WriteLn;
1839:  END;                          { STORAGEMENU }
1840:
1841:
1842:  PROCEDURE SpillageMenu;
1843:
1844:  VAR
1845:    Command : Char;
1846:    ReDraw : Boolean;
1847:
1848:  BEGIN
1849:    ReDraw := True;
1850:    REPEAT
1851:      IF ReDraw THEN
1852:        BEGIN
1853:          Title;
1854:          Say('%2Spillage menu');
1855:          Say('%2%2Loch %!Seilich Spillage%2');
1856:          Say('Loch %!Garry Spillage%2');
```

```
1857:          Say('%!Loch Ericht Spillage%3');
1858:          Say('Loch %!Eigheach Spillage%3');
1859:          Say('Loch %!Rannoch Spillage%3');
1860:          Say('Loch Err%!ochty Spillage%3');
1861:          Say('Loch T%!ummel Spillage%3');
1862:          Say('Loch %!Faskally Spillage%3');
1863:          Say('%3%!Quit%3%3)');
1864:        END;                    { IF REDRAW }
1865:      Read(Kbd, Command);
1866:      Command := UpCase(Command);
1867:      ReDraw := True;
1868:      CASE Command OF
1869:        'S' : GetData('Loch Seilch Spillage Curve ', Spillage[1, 1], 24, 0.0, 24.0, 0.0, 20.0);
1870:        'G' : GetData('Loch Garry Spillage Curve ', Spillage[2, 1], 24, 0.0, 24.0, 0.0, 20.0);
1871:        'L' : GetData('Loch Ericht Spillage Curve ', Spillage[3, 1], 24, 0.0, 24.0, 0.0, 20.0);
1872:        'E' : GetData('Loch Eigheach Spillage Curve ', Spillage[4, 1], 24, 0.0, 24.0, 0.0, 20.0);
1873:        'R' : GetData('Loch Rannoch Spillage Curve ', Spillage[5, 1], 24, 0.0, 24.0, 0.0, 20.0);
1874:        'O' : GetData('Loch Errochty Spillage Curve ', Spillage[6, 1], 24, 0.0, 24.0, 0.0, 20.0);
1875:        'U' : GetData('Loch Tummel Spillage Curve ', Spillage[7, 1], 24, 0.0, 24.0, 0.0, 20.0);
1876:        'F' : GetData('Loch Faskally Spillage Curve ', Spillage[8, 1], 24, 0.0, 24.0, 0.0, 20.0);
1877:      END;                      { CASE COMMAND }
1878:    UNTIL (Command = 'Q');
1879:    WriteLn;
1880:  END;                          { SPILLAGEMENU }
1881:
1882:  PROCEDURE InitialScreen;
1883:
1884:  VAR
1885:    I, L : Integer;
1886:    TC : Char;
1887:  BEGIN
1888:    Title;
1889:    WriteLn('                              Initial Functions');
1890:    WriteLn('I[illegible];');
1891:    WriteLn(':  Station           3 Initial  3 Desired  3 Initial  :');
1892:    WriteLn(':                    3 Storage  3 Outflow  3 Turbine  :');
1893:    FOR I := 1 TO 8 DO
1894:      BEGIN
1895:        WriteLn('GDDDDDDDDDDDDDDDDDDDDDDEDDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDDDD6');
1896:        WriteLn(':', StationName[I], '':21-Length(StationName[I]), '3',
1897:        InitialStore[I]:10:2, '3',
1898:        Solution[I]:10:2, '3',
1899:        InitialTurbine[I]:10, ':');
1900:      END;
1901:    WriteLn('H[illegible]');
1902:    I := 1;
1903:    L := 1;
1904:    REPEAT
1905:      CASE I OF
1906:        1 : InputReal(InitialStore[L], 10, 24, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
1907:        2 : InputReal(Solution[L], 10, 35, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
1908:        3 : InputInt(InitialTurbine[L], 10, 46, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
1909:      END;
1910:      IF TC = ^F THEN
1911:        IF L > 1 THEN L := L-1 ELSE L := 8;
1912:      IF TC = ^A THEN
1913:        IF L < 8 THEN L := L+1 ELSE L := 1;
1914:      IF TC = ^M THEN
1915:        IF I < 3 THEN I := I+1 ELSE I := 1;
1916:    UNTIL TC = ^Z;
```

```
1917:     DesiredStore[8] := InitialStore[8];
1918:   END;
1919:   PROCEDURE TimeMenu;
1920:
1921:   VAR
1922:     Command : Char;
1923:     ReDraw : Boolean;
1924:
1925:   BEGIN
1926:     ReDraw := True;
1927:     REPEAT
1928:       IF ReDraw THEN
1929:         BEGIN
1930:           Title;
1931:           Say('%2Set Initial Data');
1932:           Say('%2%2%!Base time : ');
1933:           Write(BaseTime:3);
1934:           Say('%2D%!Awn : ');
1935:           Write(Dawn:3);
1936:           Say('%2D%!Usk : ');
1937:           Write(Dusk:3);
1938:           Say('%2%!Demand ');
1939:           Say('%2%!Initial Data ');
1940:           Say('%2%2%!Quit%2%2');
1941:         END;                    { IF REDRAW }
1942:       Read(Kbd, Command);
1943:       Command := UpCase(Command);
1944:       ReDraw := True;
1945:       CASE Command OF
1946:         'B' : BEGIN
1947:                 Write('Enter new BaseTime ? ');
1948:                 ReadLn(BaseTime);
1949:               END;
1950:         'A' : BEGIN
1951:                 Write('Enter new Dawn Time ? ');
1952:                 ReadLn(Dawn);
1953:               END;
1954:         'U' : BEGIN
1955:                 Write('Enter new Dusk Time ? ');
1956:                 ReadLn(Dusk);
1957:               END;
1958:         'I' : InitialScreen;
1959:         'D' : GetData('Daily Demand Curve', Demand[1], 24, 0.0, 24.0, 0.0, 5.0);
1960:       END;                      { CASE COMMAND }
1961:     UNTIL (Command = 'Q');
1962:     WriteLn;
1963:   END;                          { TIMEMENU }
1964:
1965:
1966:
1967:
1968:   PROCEDURE HourlyScreen;
1969:
1970:     (*************************************************************************)
1971:     (*  THIS PROCEDURE DISPLAYS THE OPTIMISATION RESULTS                     *)
1972:     (*************************************************************************)
1973:
1974:   VAR
1975:     I, J : Integer;
1976:     Ch : Char;
```

```
1977:
1978:  BEGIN
1979:    Title;
1980:    FOR I := 1 TO 7 DO
1981:      BEGIN
1982:        HourlySolution[I, 0] := InitialTurbine[I];
1983:        Write(StationName[I], ' ':(20-Length(StationName[I])));
1984:        FOR J := 0 TO 24 DO
1985:          Write(HourlySolution[I, J]:1);
1986:        WriteLn('     ', StationName[I], ' ':(20-Length(StationName[I])));
1987:      END;
1988:    HourlySolution[8, 0] := InitialTurbine[8];
1989:    Write(' ':20);
1990:    FOR J := 0 TO 24 DO
1991:      IF HourlySolution[8, J] < 10 THEN
1992:        Write(' ')
1993:      ELSE
1994:        Write((HourlySolution[8, J] DIV 10):1);
1995:    WriteLn;
1996:    Write(StationName[8], ' ':(20-Length(StationName[8])));
1997:    FOR J := 0 TO 24 DO
1998:      Write((HourlySolution[8, J] MOD 10):1);
1999:    WriteLn('     ', StationName[8], ' ':(20-Length(StationName[8])));
2000:    I := 1;
2001:    J := 0;
2002:    REPEAT
2003:      GoToXY(21+J, 2+I);
2004:      Read(Kbd, Ch);
2005:      CASE Ch OF
2006:        '0'..'9' : BEGIN
2007:                     Write(Ch);
2008:                     IF I < 8 THEN
2009:                       HourlySolution[I, J] := Ord(Ch)-48
2010:                     ELSE
2011:                       IF I = 8 THEN
2012:                         HourlySolution[I, J] := (Ord(Ch)-48)*10
2013:                         +(HourlySolution[I, J] MOD 10)
2014:                       ELSE
2015:                         HourlySolution[I-1, J] := (Ord(Ch)-48)+
2016:                         (HourlySolution[I-1, J] DIV 10)*10;
2017:                     IF J < 24 THEN J := J+1;
2018:                   END;
2019:        #27 : IF KeyPressed THEN
2020:                BEGIN
2021:                  Read(Kbd, Ch);
2022:                  CASE Ch OF
2023:                    #75 : BEGIN
2024:                            J := J-1;
2025:                            IF J < 0 THEN J := 24;
2026:                          END;
2027:                    #77 : BEGIN
2028:                            J := J+1;
2029:                            IF J > 24 THEN J := 0;
2030:                          END;
2031:                    #80 : BEGIN
2032:                            I := I+1;
2033:                            IF I > 9 THEN I := 1;
2034:                          END;
2035:                    #72 : BEGIN
2036:                            I := I-1;
```

```
2037:                         IF I < 1 THEN I := 9;
2038:                       END;
2039:
2040:                   END;
2041:                 END;
2042:       END;
2043:     UNTIL Ch = ^Z;
2044:     FOR J := 1 TO 8 DO
2045:       InitialTurbine[J] := HourlySolution[J, 0];
2046:   END;
2047:
2048:
2049:   PROCEDURE ResultsMenu;
2050:
2051:   VAR
2052:     Command : Char;
2053:     ReDraw : Boolean;
2054:
2055:   BEGIN
2056:     ReDraw := True;
2057:     REPEAT
2058:       IF ReDraw THEN
2059:         BEGIN
2060:           Title;
2061:           Say('%2Results menu');
2062:           Say('%2S%!Pillage ');
2063:           Say('%2S%!Torage');
2064:           Say('%2%!Hourly Solution ');
2065:           Say('%2%!Output to plotter');
2066:           Say('%2%2%!Quit%2%2');
2067:         END;                    { IF REDRAW }
2068:       Read(Kbd, Command);
2069:       Command := UpCase(Command);
2070:       ReDraw := True;
2071:       CASE Command OF
2072:         'I' : InitialScreen;
2073:         'P' : SpillageMenu;
2074:         'T' : StorageMenu;
2075:         'H' : HourlyScreen;
2076:         'O' : Output;
2077:       END;                      { CASE COMMAND }
2078:     UNTIL (Command = 'Q');
2079:     WriteLn;
2080:   END;                          { RESULTSMENU }
2081:
2082:   PROCEDURE CostScreen;
2083:
2084:   VAR
2085:     I, L : Integer;
2086:     TC : Char;
2087:   BEGIN
2088:     Title;
2089:     WriteLn('                          Cost Functions');
2090:     WriteLn('I[redacted];');
2091:     WriteLn(':  Station           3  On-Off  3  Spill   3  Empty   :');
2092:     WriteLn(':                    3         3         3          :');
2093:     FOR I := 1 TO 8 DO
2094:       BEGIN
2095:         WriteLn('GDDDDDDDDDDDDDDDDDDDDDEDDDDDDDDDDDEDDDDDDDDDDEDDDDDDDDD6');
2096:         WriteLn(':', StationName[I], '':21-Length(StationName[I]), '3', CO[I]:10:2, '3',
```

```
97:        CS[I]:10:2, '3', CE[I]:10:2, ':');
98:      END;
99:    WriteLn('H[illegible]');
100:   I := 1;
101:   L := 1;
102:   REPEAT
103:     CASE I OF
104:       1 : InputReal(CO[L], 10, 24, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
105:       2 : InputReal(CS[L], 10, 35, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
106:       3 : InputReal(CE[L], 10, 46, (6+L*2), [^M, ^A, ^F, ^Z], TC, True);
107:     END;
108:     IF TC = ^F THEN
109:       IF L > 1 THEN L := L-1 ELSE L := 8;
110:     IF TC = ^A THEN
111:       IF L < 8 THEN L := L+1 ELSE L := 1;
112:     IF TC = ^M THEN
113:       IF I < 3 THEN I := I+1 ELSE I := 1;
114:   UNTIL TC = ^Z;
115: END;
116:
117:
118:
119: PROCEDURE ExecuteMenu;
120:
121: VAR
122:   Command : Char;
123:   ReDraw : Boolean;
124:   I, St, Fn : Integer;
125:   Same : Boolean;
126:
127: BEGIN
128:   ReDraw := True;
129:   St := 1;
130:   Fn := 7;
131:   REPEAT
132:     IF ReDraw THEN
133:       BEGIN
134:         Title;
135:         Say('%2Execute menu');
136:         Say('%3%!Start Station Hourly Schedule : ');
137:         Write(St);
138:         Say('%3%!Finish Station Hourly Schedule : ');
139:         Write(Fn);
140:         Say('%3%3%!Hourly Schedule');
141:         Say('%3%!Fitiochry Schedule');
142:         Say('%3%3%!Quit%3%3)');
143:       END;                    { IF REDRAW }
144:     Read(Kbd, Command);
145:     Command := UpCase(Command);
146:     ReDraw := True;
147:     CASE Command OF
148:       'S' : BEGIN
149:               Write('Enter Hourly Start Station Number ? ');
150:               ReadLn(St);
151:             END;
152:       'F' : BEGIN
153:               Write('Enter Hourly Finish Station Number ? ');
154:               ReadLn(Fn);
155:             END;
156:       'H' : HourlySchedule(St, Fn);
```

```
157:        'P' : BEGIN
158:                PitlochrySchedule1;
159:                Same := False;
160:                I := 1;
161:                REPEAT
162:                  I := I+1;
163:                  ClunieSchedule(Same);
164:                  IF NOT Same THEN
165:                    PitlochrySchedule(Same);
166:                UNTIL Same OR (I > 5);
167:              END;
168:      END;                    { CASE COMMAND }
169:    UNTIL (Command = 'Q');
170:    WriteLn;
171:  END;                        { EXECUTEMENU }
172:
173:  PROCEDURE MainMenu;
174:
175:  VAR
176:    Command : Char;
177:    ReDraw : Boolean;
178:
179:  BEGIN
180:    ReDraw := True;
181:    REPEAT
182:      IF ReDraw THEN
183:        BEGIN
184:          Title;
185:          Say('%2Main menu');
186:          Say('%2%3%1Input File Name : ');
187:          Write(DataInFileName);
188:          Say('%3%1Output File Name : ');
189:          Write(DataOutFileName);
190:          Say('%3%1Cost Function Menu');
191:          Say('%2%1Get Inflow Menu');
192:          Say('%3%1Set Initial Data');
193:          Say('%3%1Results');
194:          Say('%2%3%1Execute Menu');
195:          Say('%3%2%1Quit%2%2)');
196:        END;                  { IF REDRAW }
197:      Read(Kbd, Command);
198:      Command := UpCase(Command);
199:      ReDraw := True;
200:      CASE Command OF
201:        'I' : BEGIN
202:                Write('Enter Data Input File Name ? ');
203:                ReadLn(DataInFileName);
204:                LoadData;
205:                DataOutFileName := DataInFileName;
206:              END;
207:        'O' : BEGIN
208:                Write('Enter Data Output File Name ? ');
209:                ReadLn(DataOutFileName);
210:              END;
211:        'C' : CostScreen;
212:        'G' : InflowMenu;
213:        'S' : TimeMenu;
214:        'R' : ResultsMenu;
215:        'E' : ExecuteMenu;
216:      END;                    { CASE COMMAND }
```

```
2217:     UNTIL (Command = '0');
2218:     WriteLn;
2219:   END;                          { MAINMENU }
2220:
2221: BEGIN
2222:   FillChar(LochName, 7000, #0);
2223:   (*****  CREATE NODE TREE ONTO HEAP  ************************************)
2224:
2225:   FOR J := 0 TO MaxTime DO
2226:     FOR I := 0 TO 50 DO
2227:       New(NodePtr[J, I]);
2228:
2229:   DataInFileName := 'Default.bd';
2230:   DataOutFileName := DataInFileName;
2231:   LoadData;
2232:   MainMenu;
2233:   IF DataOutFileName <> '' THEN
2234:     SaveData;
2235: END.
```

RTSIM.PAS
SLOYSIM.PAS
FASKSIM.PAS
TORRSIM.PAS

Programs simulating Temporary Droop governed turbines using a set point controller. The main model of the simulations is contained in Procedure Model (Line 253-315).

```
 1: PROGRAM HydroSetPoint;
 2:   {Simulation of Hydro Temporary Droop governor
 3:   with set point controller                    }
 4:
 5: TYPE
 6:   String10 = STRING[10];
 7:
 8: VAR
 9:   FinTim, Time, StopTime, NextPrnTim, TimeScale : Real;
10:   ClockLow, ClockHigh, I, J, NoToPlot, LastX, ThisX : Integer;
11:   PlotMin, PlotMax, YScale, PlotVar : ARRAY[1..5] OF Real;
12:   TopY, LastY, ThisY : ARRAY[1..5] OF Integer;
13:   PlotLabel : ARRAY[1..5] OF STRING[8];
14:   Heading, ParameterLine : STRING[60];
15:
16:
17:
18: CONST
19:   NoDelays = 1;
20:   StoreLength = 100;
21: VAR
22:   Store : ARRAY[1..NoDelays, 1..StoreLength] OF Real;
23:   StrPeriod, LastStoreTime : ARRAY[1..NoDelays] OF Real;
24:   NoStrCells, StrPtr1, StrPtr2, StrPtr3 : ARRAY[1..NoDelays] OF Integer;
25:   Interpolate : ARRAY[1..NoDelays] OF Boolean;
26:   FUNCTION Delay(I : Integer; Val, Td, StartVal, H : Real) : Real;
27:   VAR
28:     TimeInPeriod, Prop : Real;
29:     J : Integer;
30:   BEGIN
31:     IF Time = 0.0 THEN
32:       BEGIN
33:         NoStrCells[I] := Round(Td/H);
34:         IF NoStrCells[I] > StoreLength THEN
35:           BEGIN
36:             Interpolate[I] := True;
37:             LastStoreTime[I] := 0.0;
38:             StrPeriod[I] := Td/StoreLength;
39:             NoStrCells[I] := StoreLength;
40:             IF StrPeriod[I] < 10.0*H THEN
41:               BEGIN
42:                 StrPeriod[I] := 10.0*H;
43:                 NoStrCells[I] := Round(Td/StrPeriod[I]);
44:               END;
45:           END
46:         ELSE Interpolate[I] := False;
47:         FOR J := 1 TO NoStrCells[I] DO
48:           BEGIN
49:             Store[I, J] := StartVal;
50:           END;
51:         StrPtr1[I] := 0;
52:         StrPtr2[I] := 1;
53:         StrPtr3[I] := 2;
54:       END;
55:     IF Interpolate[I] THEN
56:       BEGIN
57:         TimeInPeriod := Time-LastStoreTime[I];
58:         Prop := TimeInPeriod/StrPeriod[I];
59:         Delay := Store[I, StrPtr2[I]]+(Store[I, StrPtr3[I]]-Store[I, StrPtr2[I]])*Prop;
60:         IF TimeInPeriod > StrPeriod[I] THEN
```

```
61:           BEGIN
62:             LastStoreTime[I] := Time;
63:             StrPtr1[I] := StrPtr1[I]+1;
64:             IF StrPtr1[I] > NoStrCells[I] THEN StrPtr1[I] := 1;
65:             StrPtr2[I] := StrPtr2[I]+1;
66:             IF StrPtr2[I] > NoStrCells[I] THEN StrPtr2[I] := 1;
67:             StrPtr3[I] := StrPtr3[I]+1;
68:             IF StrPtr3[I] > NoStrCells[I] THEN StrPtr3[I] := 1;
69:             Store[I, StrPtr1[I]] := Val;
70:           END;
71:       END
72:     ELSE
73:       BEGIN
74:         StrPtr1[I] := StrPtr1[I]+1;
75:         IF StrPtr1[I] > NoStrCells[I] THEN StrPtr1[I] := 1;
76:         StrPtr2[I] := StrPtr2[I]+1;
77:         IF StrPtr2[I] > NoStrCells[I] THEN StrPtr2[I] := 1;
78:         Store[I, StrPtr1[I]] := Val;
79:         Delay := Store[I, StrPtr2[I]];
80:       END;
81:   END;
82:
83:   FUNCTION Strip(VAR PaddedString : String10) : String10;
84:   VAR
85:     I : Integer;
86:   BEGIN
87:     I := 1;
88:     WHILE (Copy(PaddedString, I, 1) = ' ') AND (I <= Length(PaddedString)) DO
89:       BEGIN
90:         Delete(PaddedString, 1, 1);
91:         I := I+1;
92:       END;
93:     Strip := PaddedString;
94:   END;
95:
96: VAR
97:   Inp : Byte;
98:   Count, TStore : Integer;
99:
100:  PROCEDURE InitTimer(ST : Integer);
101:  VAR
102:    Hb, Lb : Byte;
103:  BEGIN
104:    Count := ST;
105:    Port[$43] := $34;
106:    Port[$40] := 0;
107:    Port[$40] := 0;
108:    Port[$43] := 0;
109:    Lb := Port[$40];
110:    Hb := Port[$40];
111:    TStore := Hb*256+Lb;
112:  END;
113:
114:  FUNCTION TimerWait : Boolean;
115:  VAR
116:    Hb, Lb : Byte;
117:    Temp : Integer;
118:    Ok : Boolean;
119:  BEGIN
120:    Ok := False;
```

```
121:     Port[$43] := 0;
122:     Lb := Port[$40];
123:     Hb := Port[$40];
124:     Temp := Hb*256+Lb;
125:     IF (TStore-Temp) > Count THEN
126:       Ok := False
127:     ELSE
128:       REPEAT
129:         Port[$43] := 0;
130:         Lb := Port[$40];
131:         Hb := Port[$40];
132:         Temp := Hb*256+Lb;
133:         IF (TStore-Temp) > Count THEN
134:           Ok := True;
135:       UNTIL Ok;
136:     TStore := TStore-Count;
137:     TimerWait := Ok;
138:   END;
139:
140:
141:   FUNCTION Select(Reply : Char) : Boolean;
142:   BEGIN
143:     IF (Reply = 'Y') OR (Reply = 'y') THEN Select := True ELSE Select := False;
144:   END;
145:
146: VAR
147:   X, Da, Temp : ARRAY[1..20] OF Real;
148:   SPDINP, GOVIN, GOVOUT, ERRSIG, FEDBAK, FLOW, DERPOW,
149:   HEAD, WATPOW, POWOUT, PONOIS, TY, BT, BP, Td, RTIM, DPOW, TPOW,
150:   RLU, RLD, SATFE, TS, RAMP, SATIN, CKD, DE, CONIN, H, Con,
151:   POWETEP, StepTime,
152:   PER, GH, GHV, GHT, PPOW, SPOW, RPOW, TW, DLPOW, ANOISE : Real;
153:   LSIZE, N, LCOUNT, MCOUNT, KONLEV, LINC, LAST, KONBOT : Integer;
154:
155:   { LOADER3 - LOAD CONTROLLER AS USED AS ON-SITE
156:   AT SLOY, FASNAKYLE AND TORR-ACHILTY FOR STEP CHANGES
157:   MODEL OF TD GOVERNOR WITH SATURATION
158:   AND TURBINE
159:   AND POWER TRANSDUCER
160:   AND NOISE
161:   27TH JUNE 84    }
162:
163:   PROCEDURE Init;
164:
165:
166:     { INITIALISATION FOR SIMULATION
167:     SIMULATION OF THE NON LINEAR TD GOV
168:     SPDINP - INPUT TO SPEEDER MOTOR
169:     GOVIN  - INPUT TO GOVERNOR
170:     GOVOUT - OUTPUT OF GOVERNOR
171:     ERRSIG - ERROR SIGNAL
172:     FEDBAK - FEED BACK SIGNAL
173:     FLOW - WATER FLOW
174:     HEAD - HEAD AT TURBINE
175:     WATPOW - WATER POWER
176:     POWOUT - MEASURED POWER }
177:
179:   BEGIN
179:
180:     N := 6;
```

```
181:     H := 0.02;
182:
183:     { SET UP DEFAULT VALUES
184:     GOVERNOR PARAMETERS - SLOY VALUES }
185:
186:     TY := 0.3;
187:     BT := 0.25;
188:     BP := 0.03;
189:     Td := 16.0;
190:     SATFB := 0.015;
191:     RLU := 0.05;
192:     RLD := -0.25;
193:
194:     { SPEEDER MOTOR CONSTANTS - SLOY VALUES }
195:
196:     TS := 0.05;
197:     RTIM := 24.0;
198:     SATIN := 0.05;
199:
200:     { CONTROLLER CONSTANTS - AS TESTED AT SLOY }
201:
202:     CKD := 60.0;
203:     DB := 0.025;
204:     PER := 0.1;
205:     LCOUNT := 0;
206:     CONIN := 0.0;
207:     MCOUNT := 40;
208:     KONLEV := 30;
209:     LINC := 2;
210:     LAST := 0;
211:     KONBOT := 17;
212:
213:     { TURBINE PARAMETERS - SLOY VALUE }
214:
215:     TW := 1.1;
216:
217:     { POWER TRANSDUCER PARAMETERS }
218:
219:     DLPOW := 1.0;
220:     ANOISE := 0.001;
221:
222:     { CONTROLLER PERIOD CALCULATIONS }
223:
224:     GH := PER;
225:     GHV := GH;
226:     GHT := GH-0.5*H;
227:
228:     { INPUT DATA }
229:     REPEAT
230:       Write(' Enter Start Power, required power and ramp time ? ');
231:       ReadLn(SPOW, RPOW, StepTime);
232:     UNTIL (SPOW > 0.01) AND (RPOW < 1.0);
233:     PPOW := SPOW;
234:     TPOW := SPOW;
235:     IF StepTime > 0.0 THEN
236:       POWSTEP := (RPOW-SPOW)/StepTime*PER
237:     ELSE
238:       POWSTEP := (RPOW-SPOW);
239:     RAMP := SATIN/RTIM;
240:
```

```
241:    { INITIAL CONDITIONS }
242:
243:    X[1] := 0.0;
244:    X[2] := SPOW*BP;
245:    X[3] := SPOW;
246:    X[4] := 0.0;
247:    X[5] := SPOW;
248:    X[6] := SPOW;
249:    InitTimer(23900);
250:  END;
251:
252:
253:  PROCEDURE Model;
254:  VAR
255:    PO1 : Real;
256:    II : Integer;
257:  BEGIN
258:    IF (Time = 0.0) THEN Init;
259:    GOVOUT := X[3];
260:    ERRSIG := (X[2]-X[4]-(BP*X[3]));
261:    FLOW := X[5];
262:    POWOUT := X[6];
263:    HEAD := FLOW*FLOW/(GOVOUT*GOVOUT);
264:    WATPOW := HEAD*FLOW;
265:    PONOIS := POWOUT+ANOISE*(Random(1)-0.5);
266:    PO1 := PONOIS;
267:    GOVIN := X[2];
268:    Inp := Port[$3BD];
269:    Inp := Inp AND $30;
270:    IF Inp = 16 THEN
271:      CONIN := 1
272:    ELSE
273:      IF Inp = 32 THEN
274:        CONIN := -1
275:      ELSE
276:        CONIN := 0.0;
277:    SPDINP := CONIN+(1E-6*(Random(1)-0.5));
278:    IF (X[2] < -SATIN) THEN X[2] := -SATIN;
279:    IF (X[2] > SATIN) THEN X[2] := SATIN;
280:    IF (X[4] < -SATFB) THEN X[4] := -SATFB;
281:    IF (X[4] > SATFB) THEN X[4] := SATFB;
282:
283:    { MODEL OF TD GOVERNOR WITH LEFT DASHPOT COMING OUT }
284:
285:    Dx[1] := ((SPDINP*RAMP)-X[1])/TS;
286:    Dx[2] := X[1];
287:    Dx[3] := (X[2]-X[4]-(BP*X[3]))/TY;
288:    IF (Dx[3] < RLD) THEN Dx[3] := RLD;
289:    IF (Dx[3] > RLU) THEN Dx[3] := RLU;
290:    Dx[4] := (BT*Dx[3])-(X[4]/Td);
291:
292:    { TURBINE MODEL }
293:
294:    Dx[5] := (1.0-HEAD)/TW;
295:
296:    { POWER TRANSDUCER MODEL }
297:
298:    Dx[6] := (WATPOW-POWOUT)/DLPOW;
299:
300:    FOR II := 1 TO N DO
```

```
301:      BEGIN
302:        Temp[II] := H*Dx[II];
303:        X[II] := X[II]+Temp[II];
304:      END;
305:    PlotVar[1] := SPDINP;
306:    PlotVar[2] := PONOIS;
307:    PlotVar[3] := X[1];
308:    PlotVar[4] := X[2];
309:    IF NOT TimerWait THEN
310:      BEGIN
311:        ClrScr;
312:        WriteLn('Timer OverFlow!', TStore);
313:        Halt;
314:      END;
315:  END;
316:
317:  PROCEDURE DataForPlot;
318:  BEGIN
319:    NoToPlot := 4;
320:    PlotLabel[1] := 'SpdInp';
321:    PlotMin[1] := -1.2;
322:    PlotMax[1] := 1.2;
323:    PlotLabel[2] := 'Power';
324:    PlotMin[2] := 0.0;
325:    PlotMax[2] := 1.1;
326:    PlotLabel[3] := 'Lcount';
327:    PlotMin[3] := -0.01;
328:    PlotMax[3] := 0.01;
329:    PlotLabel[4] := 'Power';
330:    PlotMin[4] := 0.0;
331:    PlotMax[4] := 0.05;
332:  END;
333:
334:  PROCEDURE ModelDialog;
335:  BEGIN
336:  END;
337:
338:
339:
340: CONST
341:   Black = 0;
342:   Green = 2;
343:   Red = 4;
344:   Brown = 6;
345:
346:   PROCEDURE Enhanced;
347:   VAR
348:     regs : RECORD CASE Integer OF
349:       1 : (ax, bx, cx, Dx, BP, si, di, ds, es, flags : Integer);
350:       2 : (al, ah, bl, bh, cl, ch, dl, dh : Byte);
351:           END;
352:   BEGIN
353:     WITH regs DO
354:       ax := $0010;
355:     Intr($10, regs);
356:     WITH regs DO
357:       BEGIN
358:         ah := $11;
359:         al := $22;
360:         bl := 2;
```

```
361:      END;
362:    Intr($10, regs);
363:  END;
364:
365:  PROCEDURE eplot(X, y, colour : Integer);
366:  VAR
367:    I, off : Integer;
368:  BEGIN
369:    Port[$3ce] := 5;
370:    Port[$3cf] := 2;
371:    off := (y*80)+(X DIV 8);
372:    Port[$3ce] := 8;
373:    Port[$3cf] := 1 SHL(7-(X MOD 8));
374:    I := Mem[$A000:off];
375:    Mem[$A000:off] := colour;
376:    Port[$3ce] := 5;
377:    Port[$3cf] := 0;
378:    Port[$3ce] := 8;
379:    Port[$3cf] := $ff;
380:  END;
381:
382:  PROCEDURE Edraw(x1, y1, x2, y2, colour : Integer);
383:    (THIS PROCEDURE REWRITTEN BY DAVID LOOMES)
384:  LABEL 999;                    (EXIT POINT IF ZERO LENGTH LINE)
385:  VAR X, y, deltax, deltay, Dx, dy, run : Integer;
386:
387:    PROCEDURE dp(X, y : Integer);
388:    VAR
389:      I, off : Integer;
390:    BEGIN
391:      off := (y*80)+(X DIV 8);
392:      Port[$3ce] := 8;
393:      Port[$3cf] := 1 SHL(7-(X MOD 8));
394:      I := Mem[$A000:off];
395:      Mem[$A000:off] := colour;
396:    END;
397:
398:  BEGIN
399:    Port[$3ce] := 5;
400:    Port[$3cf] := 2;
401:    IF (x1 = x2) AND (y1 = y2)
402:    THEN GOTO 999;
403:
404:    deltay := Abs(y1-y2);
405:    deltax := Abs(x1-x2);
406:    IF y1 > y2 THEN dy := -1 ELSE dy := 1;
407:    IF x1 > x2 THEN Dx := -1 ELSE Dx := 1;
408:    X := x1;
409:    y := y1;
410:    IF deltax > deltay          (INCREMENT ALONG X AXIS)
411:    THEN BEGIN
412:      run := deltax SHR 1;
413:      REPEAT
414:        dp(X, y);
415:        X := X+Dx;
416:        run := run-deltay;
417:        IF run < 0
418:        THEN BEGIN
419:          run := run+deltax;
420:          y := y+dy
```

```
421:        END;
422:      UNTIL X = x2
423:    END
424:    ELSE BEGIN
425:      run := deltay SHR 1;
426:      REPEAT
427:        dp(X, y);
428:        y := y+dy;
429:        run := run-deltax;
430:        IF run < 0
431:        THEN BEGIN
432:          run := run+deltay;
433:          X := X+Dx
434:        END;
435:      UNTIL y = y2
436:    END;
437: 999: dp(x2, y2);
438:    Port[$3ce] := 5;
439:    Port[$3cf] := 0;
440:    Port[$3ce] := 8;
441:    Port[$3cf] := $ff;
442:  END;
443:
444:  PROCEDURE ControlDialog;
445:  BEGIN
446:    WriteLn(' Length of simulation run ? ');
447:    ReadLn(FinTim);
448:  END;
449:
450:
451:  PROCEDURE PlotAxes;
452:  VAR
453:    I : Integer;
454:  BEGIN
455:    Edraw(0, 0, 0, 319, Brown);
456:    Edraw(639, 0, 639, 319, Brown);
457:    Edraw(0, 0, 639, 0, Brown);
458:    FOR I := 1 TO NoToPlot DO
459:      BEGIN
460:        Edraw(0, TopY[I], 639, TopY[I], Brown);
461:      END;
462:  END;
463:
464:  PROCEDURE StartPlot;
465:  VAR
466:    I : Integer;
467:  BEGIN
468:    TimeScale := 640.0/FinTim;
469:    FOR I := 1 TO NoToPlot DO
470:      BEGIN
471:        YScale[I] := 320.0/(NoToPlot*(PlotMax[I]-PlotMin[I]));
472:        LastX := 0;
473:        TopY[I] := Round(I*320.0/NoToPlot)-1;
474:        LastY[I] := TopY[I]-Round(YScale[I]*(PlotVar[I]-PlotMin[I]));
475:      END;
476:    Enhanced;
477:    PlotAxes;
478:  END;
479:
480:  PROCEDURE MultiPlot;
```

```
481:   VAR
482:     I : Integer;
483:   BEGIN
484:     ThisX := Trunc(TimeScale*Time);
485:     IF ThisX > LastX THEN
486:       BEGIN
487:         FOR I := 1 TO NoToPlot DO
488:           BEGIN
489:             ThisY[I] := TopY[I]-Round(YScale[I]*(PlotVar[I]-PlotMin[I]));
490:             Edraw(LastX, LastY[I], ThisX, ThisY[I], Green);
491:             LastY[I] := ThisY[I];
492:           END;
493:         LastX := ThisX;
494:       END;
495:   END;
496:
497: VAR
498:   PlotrTimeScale : Real;
499:   PlotrStoreX : ARRAY[0..1200] OF Integer;
500:   PlotrStoreY : ARRAY[1..5, 0..1200] OF Real;
501:   PlotrBtmY : ARRAY[1..5] OF Integer;
502:   PlotrYScale : ARRAY[1..5] OF Real;
503:   PlotrStorePtr, PlotrStoreLength : Integer;
504:
505:   PROCEDURE StartPlotter;
506:   VAR
507:     I : Integer;
508:   BEGIN
509:     PlotrTimeScale := 1200.0/FinTim;
510:     PlotrStorePtr := 0;
511:     FOR I := 1 TO NoToPlot DO
512:       BEGIN
513:         PlotrYScale[I] := 5000.0/(NoToPlot*(PlotMax[I]-PlotMin[I]));
514:         PlotrBtmY[I] := 1379+Round((NoToPlot-I)*5000.0/NoToPlot);
515:         PlotrStoreY[I, PlotrStorePtr] := PlotVar[I];
516:       END;
517:     PlotrStoreX[PlotrStorePtr] := 0;
518:   END;
519:
520:   PROCEDURE PlotterStore;
521:   VAR
522:     I, ThisPlotrX : Integer;
523:   BEGIN
524:     ThisPlotrX := Trunc(PlotrTimeScale*Time);
525:     IF ThisPlotrX > PlotrStoreX[PlotrStorePtr] THEN
526:       BEGIN
527:         PlotrStorePtr := PlotrStorePtr+1;
528:         PlotrStoreX[PlotrStorePtr] := ThisPlotrX;
529:         FOR I := 1 TO NoToPlot DO
530:           BEGIN
531:             PlotrStoreY[I, PlotrStorePtr] := PlotVar[I];
532:           END;
533:       END;
534:   END;
535:
536:   PROCEDURE UsrOut(ch : Char);
537:   VAR
538:     B : Byte;
539:   BEGIN
540:     REPEAT B := Port[$3FD] AND $21 UNTIL (B = $20) OR (B = $21);
```

```
541:    IF B = $21 THEN
542:      BEGIN
543:        B := Port[$3F8] AND $7F;
544:        IF B = $13 THEN
545:          BEGIN
546:            REPEAT
547:              REPEAT B := Port[$3FD] AND 1 UNTIL B <> 0;
548:              B := Port[$3F8] AND $7F;
549:            UNTIL B = $11;
550:          END;
551:      END;
552:    Port[$3F8] := Ord(ch);
553:  END;
554:
555:  PROCEDURE PlotterInit;
556:  BEGIN
557:    UsrOutPtr := Ofs(UsrOut);
558:    Write(Usr, 'IN;', Chr(27), '.I20;;17:', Chr(27), '.N;19:,PA;');
559:  END;
560:
561:  PROCEDURE PlotterAxes;
562:  VAR
563:    I : Integer;
564:  BEGIN
565:    Write(Usr, 'PU,1650,1379,PD,1650,6379,8850,6379,8850,1379;');
566:    FOR I := 1 TO NoToPlot DO
567:      BEGIN
568:        Write(Usr, 'PU,1650,', PlotrBtmY[I], ',PD,8850,', PlotrBtmY[I], ';');
569:      END;
570:  END;
571:
572:  PROCEDURE PlotterLabels;
573:  VAR
574:    TextX, PlotrLabelY : Integer;
575:  BEGIN
576:    Write(Usr, 'SR;');
577:    FOR I := 1 TO NoToPlot DO
578:      BEGIN
579:        PlotrLabelY := PlotrBtmY[I]+Round(5000/NoToPlot)-150;
580:        Write(Usr, 'PU,950,', PlotrLabelY, 'LB', PlotMax[I]:6:1, Chr(3));
581:        PlotrLabelY := PlotrBtmY[I]+Round(2500/NoToPlot)-50;
582:        Write(Usr, 'PU,725,', PlotrLabelY, 'LB', PlotLabel[I], Chr(3));
583:        PlotrLabelY := PlotrBtmY[I]+50;
584:        Write(Usr, 'PU,950,', PlotrLabelY, 'LB', PlotMin[I]:6:1, Chr(3));
585:      END;
586:    Write(Usr, 'PU,1500,1179,LB', 0.0:3:1, Chr(3));
587:    Write(Usr, 'PU,4856,1179,LB', 'Time (sec)', Chr(3));
588:    Write(Usr, 'PU,8550,1179,LB', FinTim:6:1, Chr(3));
589:    TextX := 5250-Round(Length(Heading)*112.5/2);
590:    Write(Usr, 'PU,', TextX, ',6895,LB', Heading, Chr(3));
591:  END;
592:
593:  PROCEDURE Plotter;
594:  VAR
595:    I, J, PlotrX, PlotrY : Integer;
596:  BEGIN
597:    WriteLn(' Heading for Plot ? - Maximum length 60 characters ');
598:    WriteLn('.....................................................');
599:    ReadLn(Heading);
600:    WriteLn(' Plotter ready ? - Press any key ');
```

```
601:     REPEAT UNTIL KeyPressed;
602:     PlotterInit;
603:     Write(Usr, 'SP1;');
604:     PlotterAxes;
605:     PlotterLabels;
606:     Write(Usr, 'PU,SP2;');
607:     FOR I := 1 TO NoToPlot DO
608:       BEGIN
609:         PlotrX := 1650;
610:         PlotrY := PlotrBtmY[I]+Round(PlotrYScale[I]*(PlotrStoreY[I, 0]-PlotMin[I]));
611:         Write(Usr, 'PU', PlotrX, ',', PlotrY, 'PD;');
612:         FOR J := 1 TO PlotrStoreLength DO
613:           BEGIN
614:             PlotrX := 1650+6*PlotrStoreX[J];
615:             PlotrY := PlotrBtmY[I]+Round(PlotrYScale[I]*(PlotrStoreY[I, J]-PlotMin[I]));
616:             Write(Usr, 'PA,', PlotrX, ',', PlotrY, ';');
617:           END;
618:       END;
619:     Write(Usr, 'PU,1650,1379;');
620:   END;
621:
622: CONST
623:   ClockPeriod = 0.02;
624:
625: VAR
626:   PlotReqd : Char;
627:
628: BEGIN
629:
630:   ModelDialog;
631:   ControlDialog;
632:   DataForPlot;
633:   StopTime := FinTim+ClockPeriod;
634:   ClockLow := 0;
635:   ClockHigh := 0;
636:   Time := 0.0;
637:   NextPrnTim := 0.0;
638:   WHILE Time <= StopTime DO
639:     BEGIN
640:       Model;
641:       IF Time = 0.0 THEN
642:         BEGIN
643:           StartPlot;
644:           StartPlotter;
645:         END
646:       ELSE
647:         BEGIN
648:           MultiPlot;
649:           PlotterStore;
650:         END;
651:       (If Time>=NextPrnTim Then
652:       Begin
653:       WriteLn(Time:8:4,u:8:4,y:8:4);
654:       NextPrnTim:=NextPrnTim+PrntInt;
655:       End;)
656:       ClockLow := ClockLow+1;
657:       IF ClockLow = 10000 THEN
658:         BEGIN
659:           ClockHigh := ClockHigh+1;
660:           ClockLow := 0;
```

```
661:         END;
662:       Time := (10000.0*ClockHigh+ClockLow)*ClockPeriod;
663:       IF KeyPressed THEN Time := StopTime+10.0;
664:     END;
665:   PlotrStoreLength := PlotrStorePtr;
666:   REPEAT UNTIL KeyPressed;
667:   TextMode(BW80);
668:   WriteLn(' Do you wish hard copy ? ');
669:   ReadLn(PlotReqd);
670:   IF (PlotReqd = 'Y') OR (PlotReqd = 'y') THEN Plotter;
671: END.
```