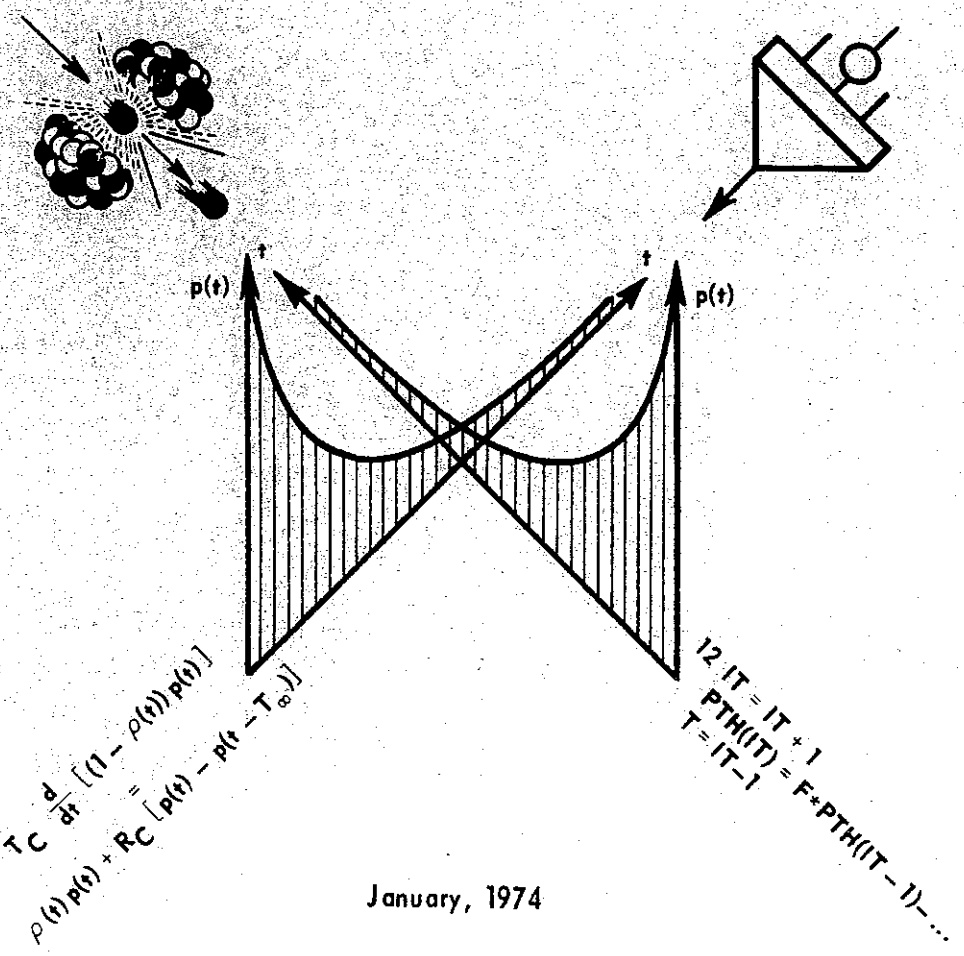
AUSTRALIAN ATOMIC ENERGY COMMISSION
RESEARCH ESTABLISHMENT
LUCAS HEIGHTS

A MATHEMATICIAN'S COMPUTER STUDY OF THE REACTOR MOATA

SUMMER SCHOOL

Lectures by

J. M. BARRY
B. E. CLANCY
C. P. GILBERT
D. B. McCULLOCH
J. P. POLLARD
P. L. SANGER



January, 1974

AUSTRALIAN ATOMIC ENERGY COMMISSION
RESEARCH ESTABLISHMENT
LUCAS HEIGHTS

A MATHEMATICIAN'S COMPUTER STUDY OF THE REACTOR MOATA

Lectures by

J. M. BARRY
B. E. CLANCY
C. P. GILBERT
D. B. McCULLOCH
J. P. POLLARD
P. L. SANGER

ABSTRACT

These notes collect together lectures on analysis of time dependent (kinetics) experiments on the reactor MOATA. The student will be introduced to scientific problem solving through the kinetics study and he will use mathematics and computers in his analysis in much the same way as a research scientist (although on a somewhat reduced scale).

CONTENTS

CHAPTER 1

PHYSICS OF REACTOR KINETICS

Lecture by

D.B. McCULLOCH

CONTENTS

1.1 NEUTRON CHAIN REACTIONS

1.1.1 Fission

Because of its zero electrical charge, the fundamental particle, the NEUTRON is very favourably placed to interact with atomic nuclei, even in the case of very heavy ones (high atomic weight, A) with large electrical charge (Z).

Some elements high in the Periodic Table, particularly uranium, are capable of interacting with a neutron in such a way that the nucleus splits (or 'fissions') into two more or less equal parts (fission products), with the liberation of a number of further neutrons, and a significant quantity of energy.

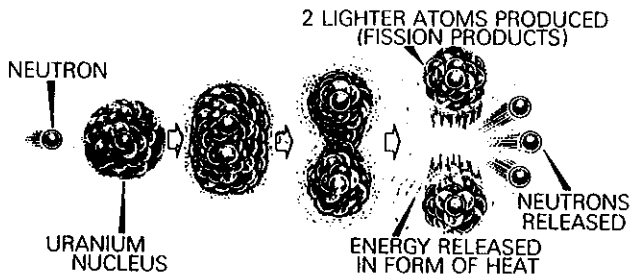


FIGURE 1 Fission of Uranium

The energy release appears because the mass of the two resulting fission product nuclei and the liberated neutrons are together slightly less than the mass of the original neutron plus target nucleus. The energy equivalent, E, of this mass difference m is given by Einstein's relationship

$$E = mc^2$$

and mostly takes the form of kinetic energy of the fission fragments and neutrons, subsequently appearing as heat as these particles are slowed down in the bulk fissioning material. Some of the energy appears also as gamma rays.

The neutrons liberated in fission arise because stable nuclear configurations for elements at the high end of the periodic table favour a higher neutron to proton ratio than is generally required for elements lower down. Hence the neutron surplus when fission product nuclei are formed.

In addition to the neutrons which are 'boiled off' at the instant of fission, some of the fission product nuclei formed still have too many neutrons to be stable and subsequently emit them by a radioactive decay process with half-lives ranging from a few tenths to a few tens of seconds. These are known as 'delayed neutrons' and amount to the order of one per cent

of the number released directly ('prompt' neutrons) in the fission process. As we shall see later, they have an extremely important part to play in the dynamic behaviour and the control of nuclear fission reactors.

The energy released in a single fission is about 200 million electron volts. A fission rate of 3×10^{10} per sec therefore releases energy at approximately 1 joule per sec, i.e. a power of 1 watt. This may sound very little, but should be looked at in the light that complete fissioning of 1 gram of a heavy element would release approximately 1 megawatt day of energy, i.e. sufficient to run 1000 - 1 kilowatt electric radiators continuously for 24 hours! Compare this with the energy release for any energy producing chemical process, e.g. burning of coal and estimate the equivalent quantities of fuel material required.

In order to induce fission, a neutron must first be absorbed into the target nucleus. Usually, the probability of absorption for slowly moving neutrons is greater than that for fast neutrons. However, energy considerations inside the compound nucleus formed when the neutron is absorbed may favour other processes than fission, unless the neutron brings with it at least a certain minimum, or 'fission threshold' energy.

An element is identified by the number of protons in, and hence the electric charge of, its nucleus. In some elements, these protons may be associated with different numbers of neutrons, giving rise to species of the same element having different atomic weights. These are known as <u>isotopes</u>.

Of the naturally occuring heavy elements, only the light isotope of uranium $^{235}U_{92}$ undergoes fission with low energy neutrons. This isotope is present to about 0.7 per cent by weight in natural uranium. The abundant uranium isotope $^{238}U_{92}$ ($\sim$ 99.3%), and also $^{232}Th_{90}$ undergo fission only with energetic (fast) neutrons ($E_n \simeq 1$ MeV, $v_n \simeq 10^4$ km s^{-1}).

Uranium can be artificially processed to yield some fractions which contain higher and some which contain lower than natural proportions of the ^{235}U isotope. The former material is favourable to fission reactions, is known as 'enriched uranium', and is widely used as a fuel in nuclear power reactors.

It is because fission is induced by neutrons, and is accompanied by the release of further neutrons, that the possibility of a continuing chain fission reaction exists. That naturally occurring fission chain reactions have not consumed all naturally occurring uranium is because processes other than fission compete with fission for the neutrons released by fission. We shall now examine briefly some of these other processes.

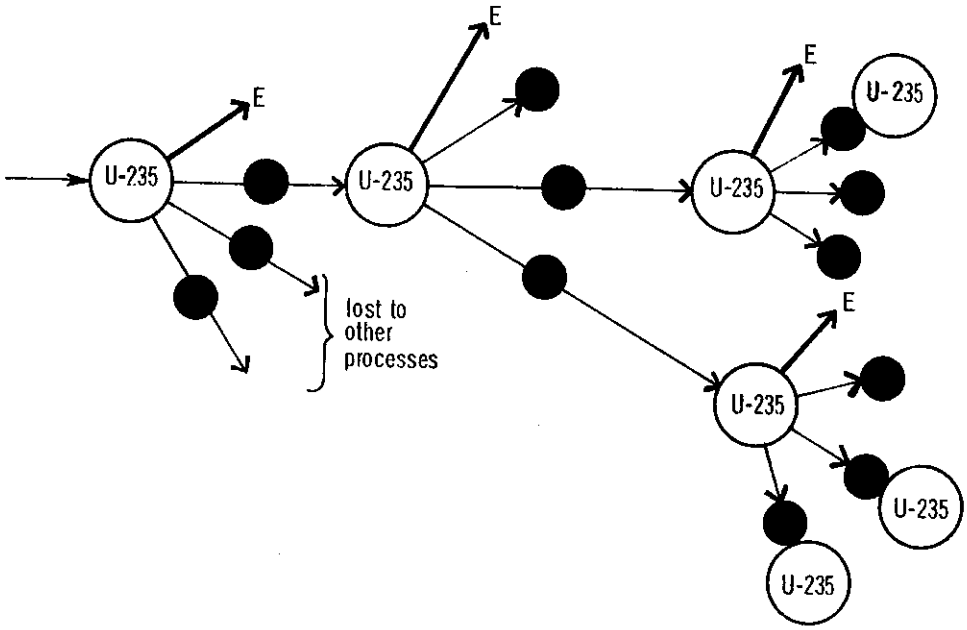


FIGURE 2 Neutron Fission Chain Reaction

1.1.2 Some Other Neutron Interactions with Matter

Elastic Scattering

In this type of interaction, both neutron and interacting nucleus behave rather like hard spheres, or billiard balls. Energy and momentum are exchanged essentially as given by the laws of classical mechanics, depending on the mass of the target nucleus and the angle of impact. Successive collisions of this type in moderating materials (light atoms of low absorption cross-section) are used in thermal neutron reactors to slow neutrons down from the energies at which they are born in fission (max ~ 10 MeV, average ~ 2 MeV) until they approach thermal equilibrium with the molecules of the reactor materials (~ 0.025 eV at room temperature).

Absorption Processes

Inelastic scattering

This process occurs mostly at fairly high neutron energies in interactions with heavier nuclei. It involves absorption into the nucleus of a neutron with energy E1, and its re-emission at a lower energy E2, accompanied by a gamma-ray, which carries off the balance of the energy. This process is very effective in transferring neutrons at fission energies to below the threshold (~ 0.8 MeV) where they would be capable of causing fission in ^{238}U.

Capture

This covers a variety of processes in which a neutron is captured to form either a stable nucleus or one which decays by emission of charged

particles and/or γ-rays to give a new product nuclide. The decay may be essentially instantaneous, as for example

$$^{10}B_5(n,\alpha)^{7}Li_3$$

which is extensively used in neutron detectors, or may take place with any half-life, e.g.

$$^{23}Na_{11} + n \rightarrow {}^{24}Na_{11} \xrightarrow[\tau_{\frac{1}{2}} = 15h]{\beta^-,\ \gamma} {}^{24}Mg_{12}$$

Capture reactions are extensively used in nuclear reactors (a) in the form of absorbing 'control rods' for direct trimming of the fission reaction rate, or for shut-down, and (b) as fillings or coatings of detectors to monitor the neutron flux level.

1.1.3 Neutron Detectors and Reactor Instrumentation

The neutron flux level is of great importance in a reactor, since it determines the rate of fission, and so the power produced in the system.

Because neutrons carry no charge, they are not detected directly, but their presence is deduced from the ionisation produced by the secondary charged particles or γ-rays arising from their interaction with some detecting nucleus.

Detectors fall into two types which are now described.

Activation Detectors

Here a material, usually in the form of foil or wire, giving rise to a radioactive decay species of suitable extended half-life is irradiated. Subsequently, it is removed from the reactor and its radioactivity measured. This is an indication of the total neutron dose (that is, time integrated neutron flux) received by the detector at its location in the reactor.

Instantaneous Indicating Detectors

These are designed to monitor the neutron flux level continuously and make available a read-out of its value at any time. They may be designed to operate in either

(i) Pulse mode, or

(ii) Current mode.

In either case, the ionisation caused by the radiation arising from neutron interaction in the detector material is measured as an electrical output.

In the pulse mode of operation, the discrete packet of electrical charge arising from each detector event is separately processed and counted. The rate of arrival of the charge pulses is then proportional to the neutron flux at the detector position.

In the current mode of operation, the detector acts as a smoother of the

electrical pulses from the ionising events to produce an ionisation current which can be measured with a sensitive direct current monitoring instrument.

Both pulse and ionisation current-type equipment are employed extensively in reactor control and safety instrumentation. Pulse equipment is generally preferable for low power operation and current type at high power.

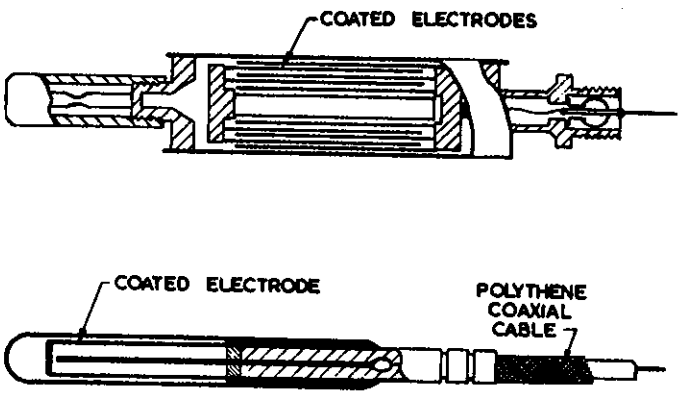


FIGURE 3 *Typical Pulsed-Type Fission Chambers (Schematic)*

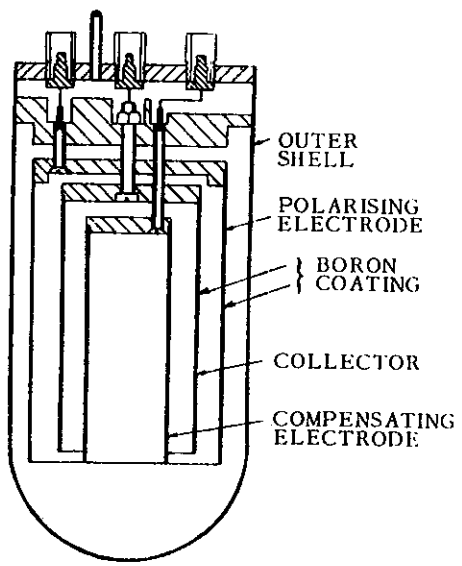


FIGURE 4 *γ-Compensated Current Type Ionisation Chamber (Schematic)*

The MOATA reactor has a pulse channel using a fission detector in which the charged fission product fragments arising from fission in a ^{235}U coating causes ionisation in a gas filling. The ionising events are detected as discrete pulses. The reactor also uses a number of current ionisation chambers, using the $^{10}B_5(n,\alpha)^7Li_3$ reaction discussed in Section 2.2.2. Gas ionisation caused by the resulting α and $^7Li_3$ particles is amplified and registered as an electrical current.

1.1.4 The Fission Chain Reaction

Consider an assembly made up of a number of materials, one of which, fuel, is capable of undergoing fission by neutron interaction. The other materials may include impurities associated with the fuel, cladding material to cover the fuel and contain the products formed in fission, a coolant material to remove fission heat, a moderator material to scatter and so reduce loss of neutrons from the assembly, and to reduce them from the energies at which they are released in fission (fission neutrons) towards thermal equilibrium with the moderator atoms (thermal neutrons) and structural materials forming part of the engineering integrity of the assembly. Such an assembly

is potentially a nuclear chain reactor.

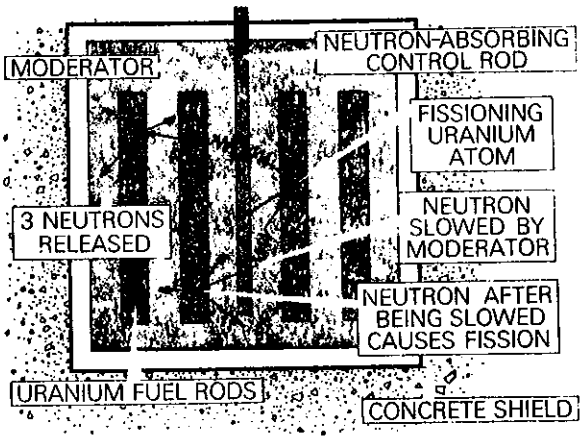


FIGURE 5 Simplified Arrangement of Reactor Showing Chain Reaction

Consider now a fission event within the assembly. On average, $\bar{\nu}$ (approximately 2.5 for ^{235}U) fission neutrons and approximately 200 MeV of energy are released. Of these $\bar{\nu}$ neutrons $\bar{\nu}(1-\beta)$ are released instantaneously and $\beta\bar{\nu}$ over an extended interval following radioactive decay half-lives from ~ 0.1 to ~ 55 sec. β is called the delayed neutron fraction.

The $\bar{\nu}$ neutrons released from an average fission will begin to have collisions with the nuclei of the materials making up the assembly. At each collision there is a probability of

(i) Elastic scattering with or without reduction in neutron energy, although on average there will be a steady reduction,

(ii) Absorption with re-emission (inelastic scattering),

(iii) Absorption with formation of a new nuclide and loss of the neutron to the system (capture), or

(iv) Absorption followed by fission.

In continuous competition with all these processes is the probability of being scattered out of and lost to the system (leakage).

The probabilities of each of the processes above are dependent on the energy of the neutron and the type of nucleus with which it collides, and are called cross sections for the various processes involved.

It is clear from the above description that if the competing processes in the assembly are just balanced so that of the $\bar{\nu}$ neutrons released on average in fission, exactly 1 is on average left to cause a further fission, then the neutron population and hence the fission rate or power of the assembly will

be constant. This state is called critical.

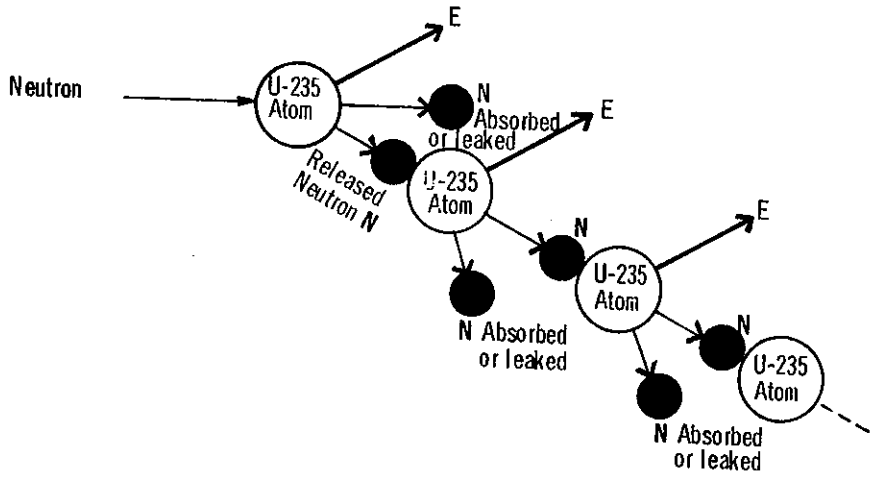


FIGURE 6 Chain Reaction - Critical State

Suppose now that we make some changes to the just critical system, for example, by removing a piece of absorbing material. Now the balance of competing processes is changed and instead of exactly 1 neutron per fission being available to cause a further fission, there will be, say, k where k is a little larger than unity. Consequently, if n_0 neutrons are initially present in the reactor the next generation will have $n_0 k$, the following are $n_0 k^2$, the next $n_0 k^3$, etc. That is, the reactor neutron population and hence the fission rate and power level will increase so that in the r^{th} generation, $n_r = n\ k^{(r-1)}$. By substituting $t = (r-1)\ell$, where ℓ is the mean lifetime of a neutron generation, then the neutron population, n, after time t, is given by

$$n = n_0 k^{t/\ell}$$

whence taking logarithms (to the base e = 2.718282) we obtain the result

$$n = n_0 e^{(\ell n\ k)t/\ell}$$

$$(\ell n\ k = \log_e k) \quad .$$

Since k normally differs only very slightly from unity, then to a close approximation

$$\ell n\ k = \ell n[1 + (k-1)] \simeq k-1$$

and then

$$n = n_0 e^{\frac{k-1}{\ell} \cdot t} \quad ,$$

showing that the neutron population or power increases exponentially. (k-1) is sometimes called the reactivity of the system and denoted by the symbol, ρ.

This state of the reactor, where the power is diverging exponentially, is called 'super-critical'. The increase may be arrested and the power maintained

at a new constant level simply by making the change necessary to restore k to unity when the required level is reached.

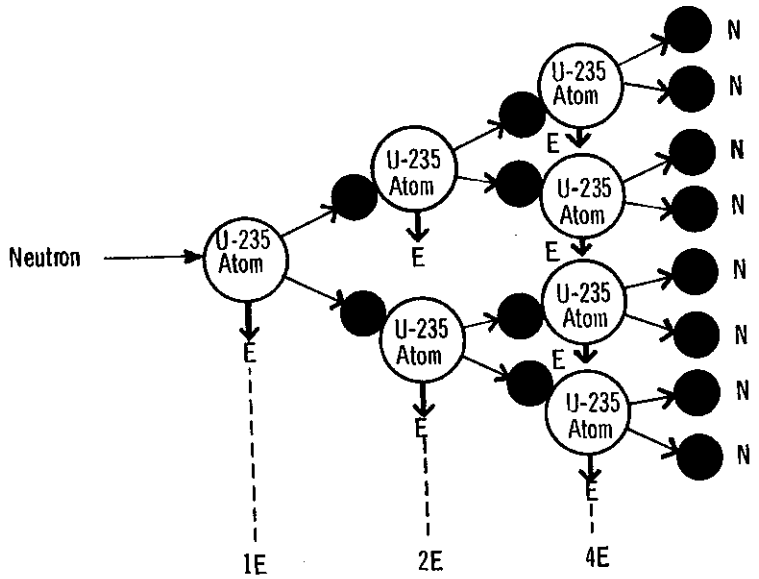


FIGURE 7 Chain Reaction - Supercritical State

By a similar argument to that just outlined above for $k > 1$, if a change is made to the reactor so that k is reduced below unity, the exponent in the neutron population equations becomes negative and we would expect the neutron population to die away exponentially to zero. A reactor in a state for which $k < 1$ is said to be subcritical. In actual reactors, the sub-critical behaviour of the neutron population is very strongly modified by the neutron source which is always present by design or by inherent existence in the reactor materials. With such a source, say of S neutrons per sec, then the neutron population in the sub-critical system does not die away to zero, but by following successive generation arguments similar to those above, can be shown to reach a steady level given by

$$n \propto S(1 + k + k^2 + k^3 + \ldots) = \frac{S}{1-k}$$

$1/(1-k)$ is now said to be the multiplication factor or multiplication of the sub-critical reactor.

The behaviour deduced in the preceding two paragraphs for disturbances from the critical state was simplified by neglecting the fact that some of the $\bar{\nu}$ neutrons arising from fission are emitted from fission product fragments at quite long times after the fission occurs. The effect of these delayed

neutrons is to slow down very markedly the response of the reactor power to a change in the criticality constant, k. This is very fortunate, since with typical neutron generation life-times ranging from ~ 0.1 μs (10^{-7} seconds) for small fast neutron reactors to ~ 1 ms (10^{-3} seconds) for a large thermal neutron reactor, the formulae just derived show that power increase rates for even small changes in reactivity could be very rapid indeed, and would give rise to severe control problems.

In qualitative terms, a just critical reactor operating at steady power level with neutron density n_0 can be regarded as having its unity criticality constant made up of a prompt contribution (1-β) plus a delayed part, β. The fission products giving rise to the delayed neutrons are called the delayed neutron precursors and are being produced at a constant rate just equal to their rate of depletion by radioactive decay.

If now the criticality constant is increased to 1 + δk, the 1-β part increases to (1-β) (1+δk) and the reactor responds immediately to this as given in Section 1.1.4 paragraph 6. The rate of production of delayed neutron precursors increases at once corresponding to the new power level, but the rate at which the additional delayed neutrons are released is governed by the radioactive half-lives of the precursors as well as by the term β(1+δk). The net result is that reactor response to a change in k consists of an initial rapid change due to the prompt neutrons, followed by a slower one governed by the delayed neutrons. This is indicated qualitatively in Figures 8 and 9 for increase and decrease in k respectively.

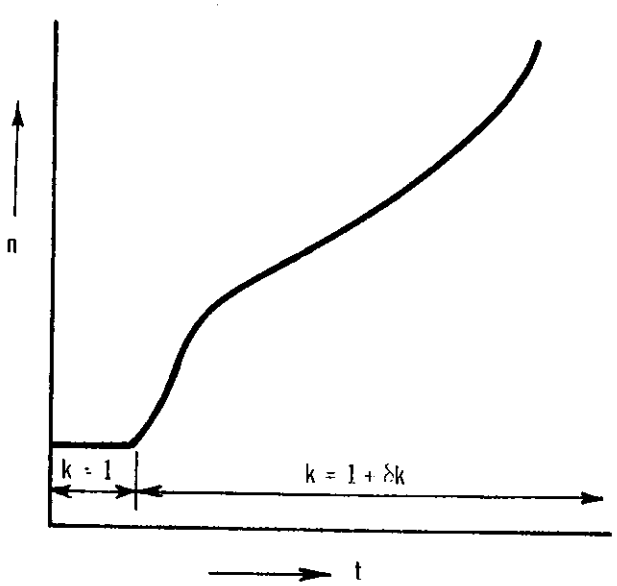


FIGURE 8 *Reactor Power Response to Increase in k*

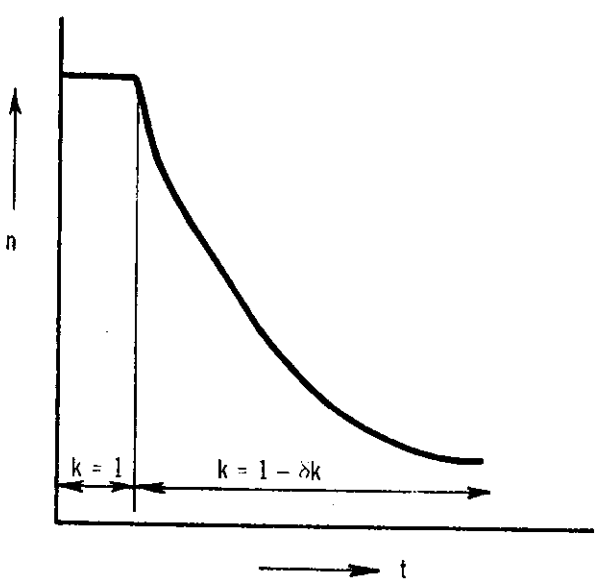


FIGURE 9 *Reactor Power Response to Decrease in k*

It follows easily that when k is returned to unity to level off at a new desired power, the neutron population does not respond instantaneously, but continues to change according to its 'memory' of the preceeding delayed neutron precursor concentrations. Anticipation by the operator based on his experience is therefore necessary if a new power level is to be approached smoothly and without 'over shoot'.

The further investigation of reactor kinetic behaviour requires the use of the reactor kinetic equations which will be discussed in Chapter 2. These should be easier to understand with the picture of the physical principles involved which you now have. It should, however, be borne in mind that only a very simple model has been presented. For example, many of the competing processes for neutron rates depend on temperature in different ways, so modifying the reactor response as the power (and therefore temperature) rises or falls following a change in k. This will, however, not be of great significance in the experimental you will be doing on the MOATA reactor.

1.1.5 The MOATA Reactor

MOATA* is an Argonaut type reactor, based on a design by the Argonne National Laboratory in the United States. Argonaut was intended originally to be an inexpensive, flexible and safe reactor, suitable for use in universities and similar institutions, the type of core having been the subject of kinetic and transient tests on water-moderated systems. These tests showed this type of core to have strongly self-limiting properties with regard to energy release and power levels reached in reactivity excursions, provided that certain reasonable restrictions on core fuel content are observed. It is therefore an inherently very docile reactor system.

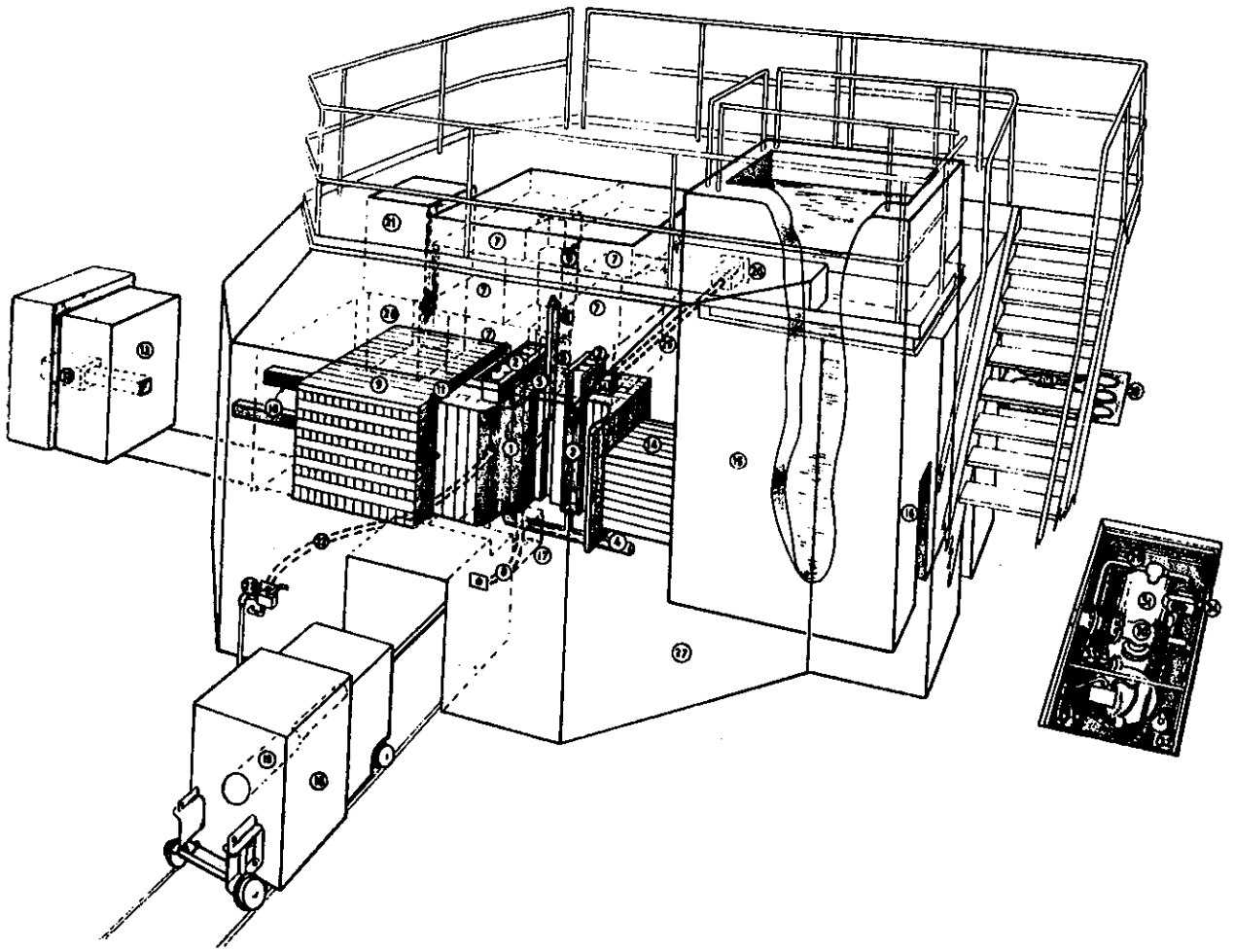
The MOATA version of the Argonaut genus can operate at a maximum heat output of 100 kW, when the peak thermal neutron flux is approximately 1.5×10^{12} n cm^{-2} sec^{-1}. The reactor is designed to be used for a variety of irradiation experiments and as a source of neutron beams.

MOATA Core

The core of the reactor consists of a 1.3 m cube of graphite into which are set two aluminium core tanks. Six fuel elements are located in each core tank, each fuel element consisting of twelve fuel plates. These fuel plates are of sandwich construction, with a core of uranium-aluminium alloy sheet, clad with pure aluminium. The uranium content is enriched in the

---

* Aboriginal word of North Queensland Ngerikudi tribe, meaning 'firesticks' or 'gentle heat'.



KEY TO DRAWING

1. GRAPHITE CORE.
2. CORE TANKS.
3. FUEL ELEMENT.
4. MODERATOR/COOLANT INLET.
5. REMOVABLE STRINGERS.
6. CENTRAL STRINGER PLUGS.
7. TOP CLOSURES.
8. NEUTRON SOURCE POSITIONER.
9. THERMAL COLUMN.
10. THERMAL COLUMN STRINGERS.
11. LEAD GAMMA CURTAIN.
12. THERMAL COLUMN DOOR.
13. THERMAL COLUMN DOOR PLUG.
14. SHIELD TANK DUCT.
15. SHIELD TANK.
16. SHIELD TANK OUTER DOOR.
17. HORIZONTAL RADIATION CAVITY.
18. RADIATION CAVITY DOOR.
19. RADIATION CAVITY DOOR PLUG.
20. VERTICAL RADIATION CAVITY.
21. RADIATION CAVITY CLOSURE.
22. PNEUMATIC TUBE.
23. PNEUMATIC TUBE AIR SUPPLY.
24. CONTROL ROD DRUM HOUSING.
25. CONTROL ROD DRIVE SHAFT.
26. CONTROL ROD DRIVE MECHANISM.
27. BIOLOGICAL SHIELD.
28. FUEL & EXPERIMENT STORAGE PIT.
29. PROCESS PIT.
30. DUMP VALVE.
31. DUMP TANK.
32. ION EXCHANGE COLUMN.
33. FLOW RATE REGULATOR.
34. MOTORISED VALVE.

FIGURE 10 The Research Reactor MOATA

Demineralised light water circulates between the fuel plates when the reactor is operating and acts as neutron moderator as well as removing heat generated in the core. Water enters each core tank through a large diameter pipe at its base and leaves at the top.

High-purity nuclear grade graphite acts as an internal neutron reflector between the two core tanks and as an external reflector around the outer surfaces of the tanks. Inserted into the reflector is a one curie, plutonium-beryllium neutron source, which keeps neutron flux measuring instruments on scale when the reactor is shut down. This source can be withdrawn into the biological shield by remote control while the reactor is operating.

Many of the experimental uses of the reactor involve the irradiation of subcritical assemblies, and for this purpose the principal facilities available are cavities into which the assemblies are placed. Six cavities, adjacent to the core, can be provided. Four are situated on the vertical faces of the core, one is situated vertically in the biological shield and one may be formed by removal of the internal reflector graphite from between the core tanks. The four horizontal cavities are filled with graphite at present and thus form thermal columns, regions of relatively pure thermal neutron flux, but the graphite in them is readily removable.

Graphite regions of the reactor contain stringers, which may be removed to provide small irradiation volumes where calibration is carried out of materials used as neutron flux detectors. Access to these stringers and to the cavities, is obtained through plugs or doors in the biological shield, and this can only be done when the reactor is shut down. Rapid access to a high neutron flux region is available by means of the pneumatic tube or 'rabbit', a device similar to the tubes once used in some department stores for sending money to the cashier.

Specimens to be irradiated in the rabbit are inserted in an aluminium tube on the outer face of the biological shield and blown direct to the core by compressed air. Removal is also by compressed air.

Biological Shield

The biological shield is required to protect personnel from intense neutron and gamma radiation associated with the fuel during operation and from high-level, fission-product radiation present even when the reactor is shut down. The MOATA shield consists of layers of heavy concrete immediately surrounding the core, and in the outer faces of the shield. This concrete is made from ilmenite sand (iron and titanium oxides) and steel punchings. The remainder of the shield volume is filled with ordinary concrete. Total

shield thickness is about 2.3 m in any direction from the core.

This shield design is a compromise between two requirements. The core gamma rays require a dense material, such as heavy concrete, in order to be attenuated in a short distance. Nevertheless, neutrons escaping from the core activate any iron in their path, and de-excitation of the iron isotopes produced gives rise to further high energy gamma rays, which in turn require adequate thickness of shielding. Thus any iron, such as that in the heavy concrete, must either be sufficiently shielded on the outer side to remove the iron capture-gammas, or be shielded sufficiently on the core side to absorb and scatter neutrons before they reach the iron.

Plugs and doors in the shield, some of which are rail-mounted, are made either of solid steel, or of steel frames filled with medium density concrete.

The whole-body radiation dose at the shield surface at full power is nowhere greater than a few milliröntgens per hour.

Control System

The control system may be considered in two parts, firstly, the devices used to vary reactor conditions, and secondly, the instruments used to measure reactor conditions. In the first category are the control absorbers or rods, consisting of pieces of neutron-absorbing material arranged to move in a region adjacent to the core tanks. The neutron absorber is boral, a boron carbide-aluminium complex, in the form of sheet, which is welded to stainless-steel spring strip. The spring is wound on a drum mounted on top of the reflector graphite, and the drum is rotated by a shaft passing through the biological shield to a recess on the outer face where the motor and drive mechanisms are housed. A magnetic clutch is interposed between motor and shaft; de-energising of the clutch allows the spring and gravity forces to insert the absorber rapidly to the position of maximum effectiveness. Details of a control rod and its drive mechanism are shown in Figure 12.

MOATA has four such control rods, two being used as safety rods. The latter are fully inserted when the reactor is shut down, and fully withdrawn when it is operating, providing a substantial margin for shutting down whenever necessary. Two rods are used for coarse and fine adjustments when taking the reactor to a critical state. These are, respectively, the shim rod and the regulating rod.

Normal and emergency methods for shutting down the reactor are the same, namely, breaking of magnetic clutch currents and thus fully inserting all rods; by this means rods are inserted in less than half a second. At the same time, the water in the core tanks is dumped and the reactor becomes completely shut

thermally-fissile isotope uranium-235 to 90 per cent. Plates are assembled with spacers and bolts into an element as shown in Figure 11. The total core loading is approximately 3 kg ^{235}U.

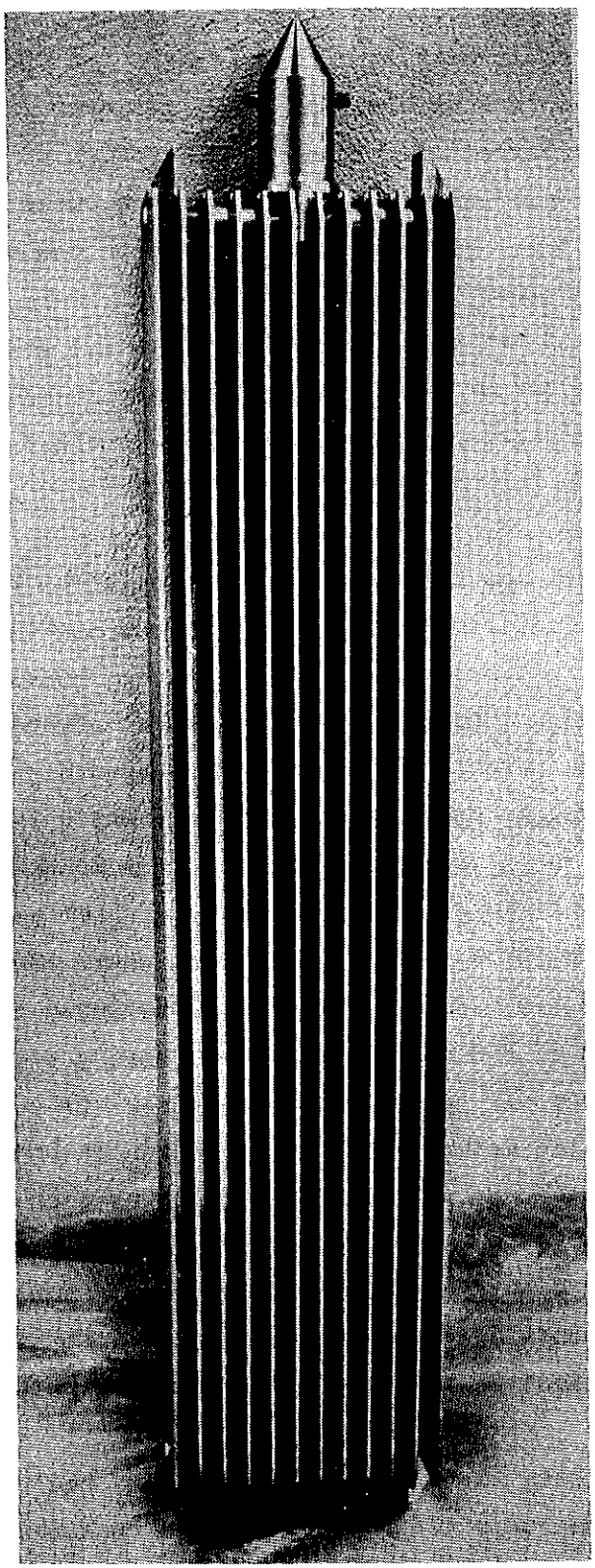


FIGURE 11 A MOATA Fuel Element

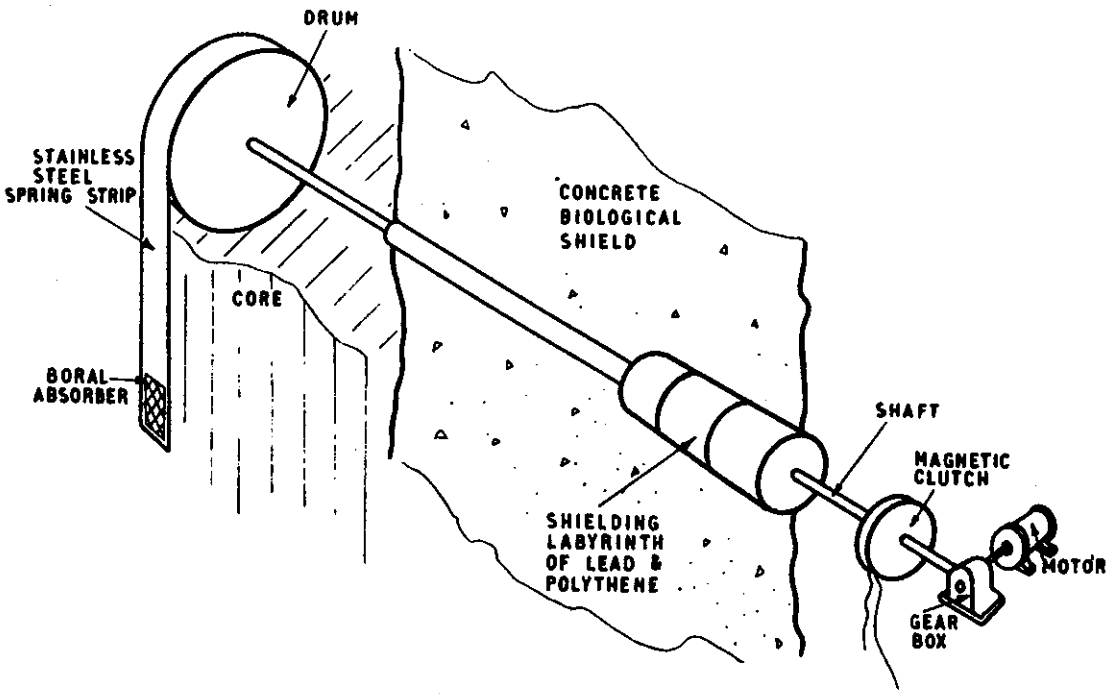


FIGURE 12 Diagram of the MOATA Control Rod Drive Mechanism (Schematic)

down within a few seconds.

Measurement of reactor conditions, the second category of control mechanism, is accomplished by five neutron-detecting chambers arranged on top of the core.

A fission-chamber, with a coating of ^{235}U, is sufficiently sensitive to provide measureable signals when the reactor is shut down. A pulse output is taken from this chamber, and is displayed on the control console as a count-rate and as a rate of change of count rate, or period.

The neutron flux level in MOATA from shut-down to full power varies over about 8 decades. Because this is much smaller than the range (~ 11 or 12 decades) in a full commercial power producing reactor, high range current type detectors remain quite effective in MOATA from full power down to very low flux levels.

Two similar current-type detectors are used for this wide-range reactor control. They are boron lined γ-compensated ionisation chambers from which mean current outputs are taken for display, one on a logarithmic scale over the full flux range, together with associated flux period information, the other on a switchable-range linear pico-ammeter, allowing accurate control and resetting of reactor power.

Two further entirely separate channels are used as high-flux trip

instruments through a safety amplifier, which supplies current for all magnetic clutches. By interrupting the clutch currents, the amplifier shuts down the reactor automatically on reaching a level 25 per cent above rated maximum power.

One problem concerning neutron detecting instruments in reactors is their sensitivity to gamma radiation. In the shut down state fission products in the fuel emit high intensity gamma radiation, which could produce a signal in an ionisation chamber much greater than that due to the source neutrons. The fission chamber does not suffer from this trouble because the pulses due to fission fragments are so much larger than those due to γ-rays, but the log and linear current chambers are sensitive to gamma rays. This is overcome by using compensated chambers, where current due to gamma rays only is cancelled out through use of two similar ionisation volumes in one instrument, but only one of the volumes being neutron sensitive.

It is a principle of safe reactor operation that two independent channels must be operative at any time, and provide information about both flux (or power) and period (or rate of change of flux). Automatic trip circuits are incorporated in the electronic units, which may shut the reactor down at a chosen flux or period limit. The channels provided ensure that flux and period trips are available at any power level from the shut down state to peak power.

An automatic power controller keeps the reactor at a selected power level by appropriate movement of the regulating rod. This instrument relieves the operator of the need to make frequent adjustments to control absorbers, for instance, to compensate for core temperature changes. It also enables reactor power and hence neutron flux at any given point, to be kept more accurately constant over a long period of time than could be attained manually.

A significant contribution to safety of operation is made by the start-up sequence. This is an assembly of switches, arranged so that the operation involved in starting-up the reactor may only be carried out in a prescribed order, no operation of a particular item being possible until the preceding one has been completed satisfactorily. Lights indicate the sequence, and the stage reached at any time.

All electronic instruments and controls are housed in a small control console, which may be operated by a single operator.

Process System

The process system consists of those units associated with circulation and cooling of the light water moderator. Circulation through core tanks and

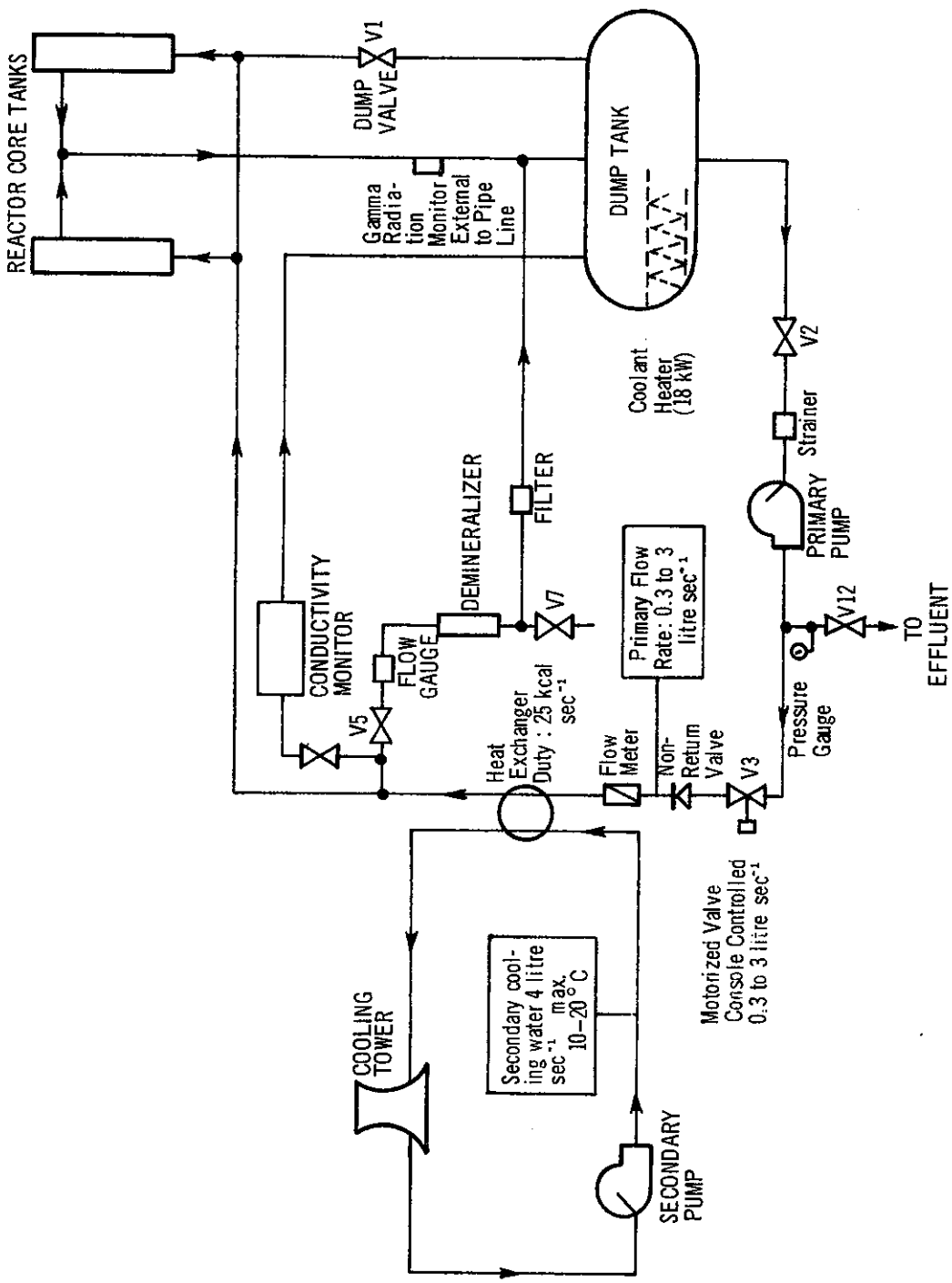


FIGURE 13 MOATA Process System

heat exchanger is maintained by a pump, but water may only rise into the core tanks on closure of the dump valve. The latter operation forms part of the start-up sequence, and is carried out before raising any control absorber.

The dump valve opens rapidly on shut-down, allowing all water in the core tanks to drain within a few seconds into the dump tank, which is the normal storage vessel for the total charge of demineralised light water.

A by-pass loop in the process system provides a filter and mixed-bed ion-exchange column for the water. The former maintains concentration of possible radioactive contaminants at a low level, and the latter keeps pH and conductivity values within a range which minimises corrosive attack on aluminium components in the water circuit.

The heat exchanger is of the shell and tube type cooled on the secondary side by water passing through and rejecting heat to an air cooling tower external to the building. The flow is controllable to permit adjustment of reactor operating temperatures.

The process system also includes instrumentation to monitor primary and secondary coolant temperatures and flow, together with pressure switches which monitor water levels in the core tanks, detect interruptions to water circulation, and so on.

Reactor Calibration and Operation

Control characteristics may be described in terms of reactivity, ρ, which was shown in Section 1.1.4 to determine the rate at which changes in neutron population occur. The available reactivity depends on a number of factors, including the quantity of fuel in the core in excess of the critical loading. The absorptive properties of the control absorbers may be expressed in terms of reactivity, and safe operation of a reactor requires available 'negative' reactivity in control absorbers to be considerably greater than the 'positive' reactivity contributed by the fuel excess.

Rate of change of reactor flux obeys an exponential law, the characteristic time or period being dependent on the excess reactivity present. In order to keep the period longer than a certain safe minimum time, excess reactivity of the reactor is limited to a maximum of 0.6 per cent. This limitation brings the reactor into the 'eversafe' category, a sufficiently safe state to permit it to be sited in an open bay of a building without the need for sealed shell containment.

At the initial start-up of MOATA, the period obtained with all control absorbers fully withdrawn gave an immediate indication of excess reactivity acquired from the quantity of fuel loaded. The amount of control absorber

insertion needed to restore the critical state gave a measure of the corresponding reactivity worth for that portion of the absorber. By adding further small quantities of fuel, which in turn required greater control absorber insertion to maintain a critical state, it was possible to calibrate most of the length of control absorbers in terms of reactivity. Both regulating rod and shim-rod were calibrated in this way, further calibration being obtained by balancing one rod against the other (e.g. withdrawal of one against insertion of the other).

It was not possible to load sufficient fuel to calibrate the shim rod in its fully-inserted state, so these measurements were made with the reactor subcritical, reactivity worths of rod position changes being related to changes in the subcritical flux level.

Safety rods, which cannot be set in intermediate positions, were calibrated by a 'rod-drop' method. In this technique, the current from an ionisation chamber in the core was followed on a fast pen recorder from the moment current was interrupted to the rod magnetic clutch. The resulting transient signal enabled calculation of the total reactivity worth of the rod and also provided information on time elapsed between breaking of clutch current and actual fall of the rod. Portions of the shim rod were also calibrated by the drop method and thus this rod was calibrated by three different methods. Results were found to be in good agreement and Figure 14 shows typical rod calibration curves.

With the reactivity corresponding to every position of shim and regulating rods determined, it becomes possible to measure reactivity worths of other core components. Each component has neutron scattering and absorption properties, so its addition or removal must influence the fission cycle and its reactivity effect is measured simply by obtaining critical settings of the control absorbers, both with and without that particular component in place. The corresponding difference in reactivity of the rods then gives the reactivity of the component.

Neutron flux has been mentioned a number of times in these notes and is a quantity which is often measured. Ionisation detectors, such as ion chambers with coatings of fissile material or proportional counters filled with boron trifluoride gas enable the shape of the reactor flux distribution to be determined quite rapidly and with good accuracy, but do not lend themselves to precise determination of the absolute value of the flux level. For this purpose, activation detectors, such as indium or gold foils are irradiated under standard conditions at a number of standard locations. Their induced

radioactivity is then subsequently analysed by absolute counting using 4π-β and β-γ coincidence techniques and allow the actual neutron flux at the irradiation positions to be deduced.

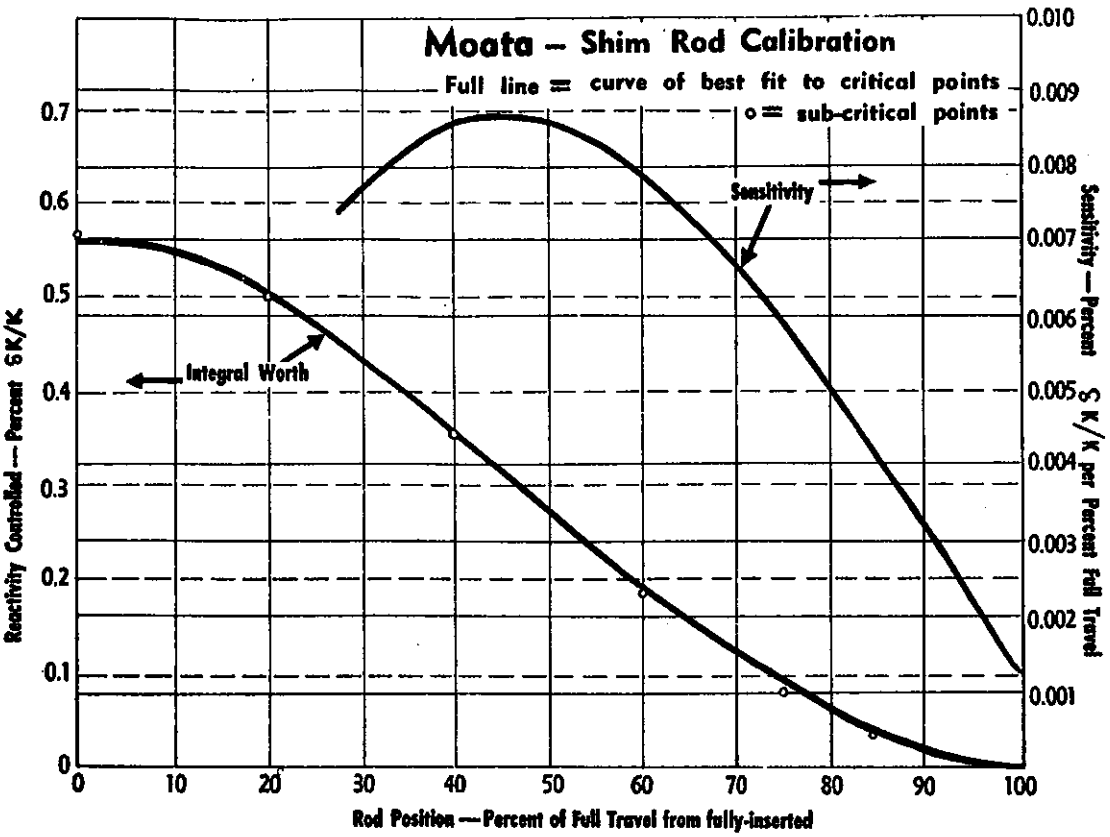


FIGURE 14 Typical Rod Calibration Curves

Another aspect of flux measurements is determination of neutron energy spectrum over the reactor. Neutrons are born in the fission process with energies ranging up to about 10 million electron-volts (MeV), and as they collide with moderating material they lose energy. Eventually they become thermalised, taking up the thermal motion of the moderator, energies then being about 0.025 eV.

By activating detectors having selective responses to neutrons of different energies it is possible to build up information on the neutron energy spectrum and its variation over the reactor. This is of importance for many experimental applications of the reactor.

Power calibration of MOATA is a little indirect. Although a high proportion of the energy released in fission ultimately appears in the form of heat, the large thermal capacity of the core, combined with relatively low power produced, means that it takes a long time to reach thermal equilibrium.

Hence measurement of heat added to the circulating coolant is not a reliable guide to total power produced. In any case, reactor users are usually more interested in the neutron flux than thermal power. Thermal power depends on the total fission rate over the core at a given flux level, and thus determination of neutron flux over the fuel region, relative to that at a standard position is the quantity of primary concern. By irradiating gold and copper wires fastened to nylon cords between the fuel plates the flux variations in the fuel can be determined. This enables the integrated core fission rate and hence thermal power to be determined relative to the readings of the reactor installed instrumentation.

The reactor operates routinely at all power levels up to 100 kW as required on a daily basis by a wide range of experiments and users. Some typical uses are

(1) Extraction of external neutron beams for nuclear physics measurements, such as cross-section determinations.

(2) Production of radioisotopes for a variety of applications embracing experimental, industrial and medical uses.

(3) Neutron irradiations of materials for analysis by activation techniques, investigation of radiation effects on chemical reaction rates, etc.

(4) Neutron radiography as a technique for some metallurgical examinatins, etc.

1.2 ACKNOWLEDGEMENT

The material of Section 1.1.5 is taken largely (with some updating) from the article 'MOATA Reactor' by A.P. Marks, which was published in the October 1962 issue of the AAEC journal 'Atomic Energy in Australia'.

1.3 SUGGESTED FURTHER READING

(1) Any modern text-book available to you on nuclear physics at an introductory level. Concentrate particularly on chapters dealing with interactions of neutrons, γ-rays and low energy charged particles with matter.

(2) Price, W.J. Nuclear Radiation Detection (McGraw Hill)

(3) Murray, R.L. Nuclear Reactor Physics (Prentice-Hall)

(4) Murray, R.L. Introduction to Nuclear Engineering (Prentice-Hall)

(5) Glasstone, S. Principles of Nuclear Reactor Engineering, Chapters I-VI (Van Nostrand), or

Glasstone, S. and Sesonke, A. Nuclear Reactor Engineering, Chapters I-V (Van Nostrand Reinhold Co).

CHAPTER 2

MATHEMATICS OF REACTOR KINETICS

Lecture by

J.P. POLLARD

CONTENTS

2.1 INTRODUCTION

The detailed study of the time behaviour (kinetics) of a reactor requires an understanding of the reactions of neutrons with material constituting the reactor (Chapter 1). In particular neutron production by fission occurs both by

(1) prompt neutrons, which are essentially emitted instantaneously and

(2) delayed neutrons, which are emitted some seconds later.

Although accurate kinetics calculations require the solution of somewhat involved mathematical equations (taking several hours of computer time) we will use an approximate description (taking several seconds of computer time). Considering the severity of the approximations embodied in our description perhaps surprisingly we will be able to analyse kinetics experiments on the reactor MOATA to within almost tolerable accuracy.

During normal operation of a reactor the power is kept at a constant level, p_0. If for some reason the power is to be raised to a new operating level then a control rod is moved out of the reactor and the 'reactivity' (related to the extent to which neutrons multiply by fission reactions) is thereby increased from the steady state value of $\rho = 0$ to some other normally time, t, dependent value $\rho(t)$ 'dollars'. Figure 1 is a simplified view of the reactor and Figures 2 and 3 depicts possible power responses.

When the power is near the level required, the control rod is restored to near the original position, corresponding to $\rho(t) = 0$, and the power levels off at the new power - again to a constant value. Some fine adjustment of the control rod up ($\rho(t) > 0$) or down ($\rho(t) < 0$) may be necessary to achieve the required power level. As we will see the response is not instantaneous (due to delayed neutrons) so that the actual fine adjustment to be made is not immediately clear. In order for us to obtain a 'feel' of the time responses involved (in much the same way as the person operating a reactor) we will use an analogue computer (Chapter 6) to simulate the site reactor MOATA. Also we will measure ρ using experiments to be carried out on MOATA (Chapter 1).

Vital to the safe running of a reactor is the ability to shut it down as quickly as possible (that is to 'scram' the reactor). This is achieved by dropping safety rods into the reactor (Figure 1) as we ourselves will do with MOATA. The type of power response is depicted in Figure 3.

An approximate mathematical description for the variation of power with time, p(t), following the variation of control or safety rods introducing $\rho(t)$ 'dollars' of reactivity is given by the delay differential equation

$$T_c \frac{d}{dt}\left[(1-\rho(t))\ p(t)\right] = \rho(t)\ p(t) + R_c\left[p(t) - p(t-T_\infty)\right] \quad , \qquad \ldots(1)$$

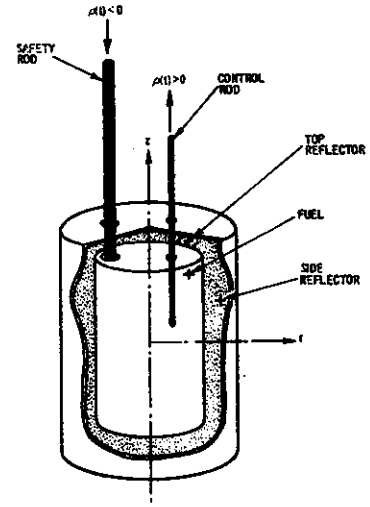


FIGURE 1 A Simplified View of a Reactor

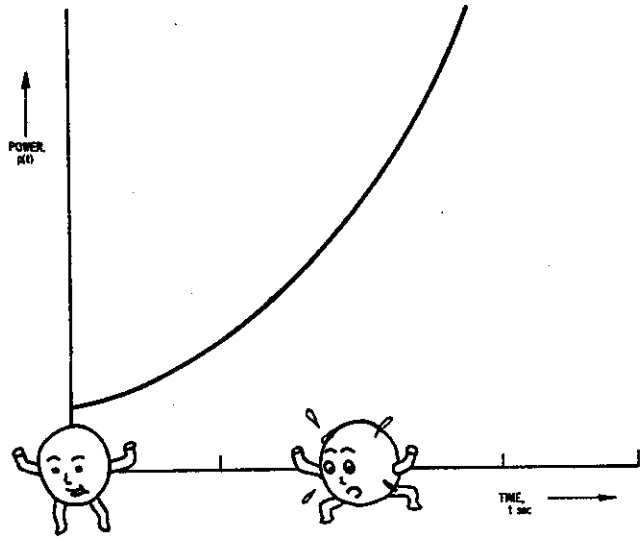


FIGURE 2 A Possible Power Response of a Reactor ($p(t) > 0$)

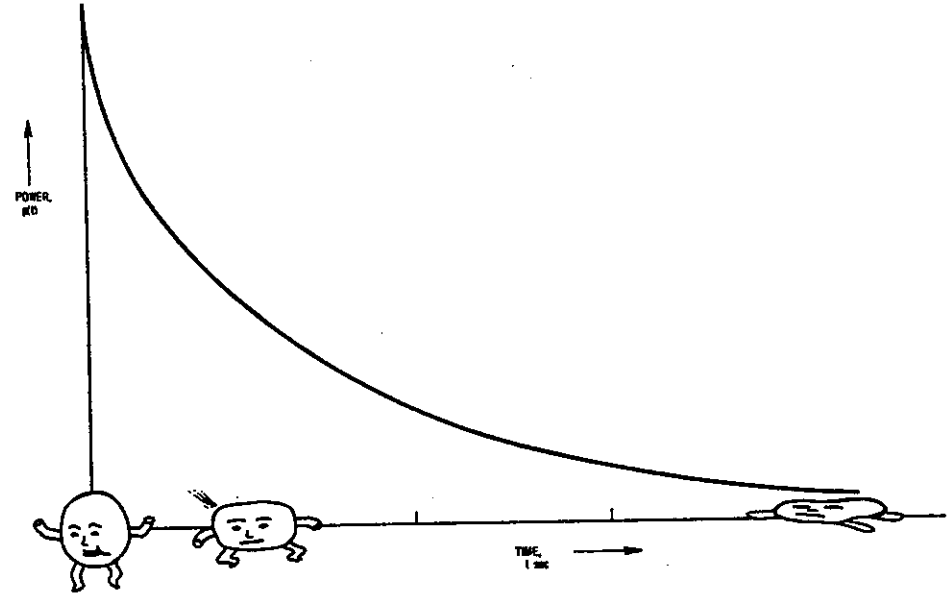


FIGURE 3 A Quick Reactor Shutdown (SCRAM, $p(t) \ll 0$)

where T_c, T_∞ and R_c are reactor constants, which for MOATA are given as

$$\left.\begin{aligned} R_c &= (1-c)\left(\frac{T_c}{T_\infty} - 1\right) \\ T_c &= 4 + 9/c \text{ sec} \\ \text{(hence } T_\infty &= 4 \text{ sec)} \end{aligned}\right\} \quad \ldots(2)$$

and $c = 0.29$ (for a so called 2 delayed group approximation). ...(3)

Our experiments on MOATA are particularly concerned with sudden changes of reactivity $\rho(t)$ from 0 to some other steady value at time $t = t_0$. (For example, a safety rod is suddenly dropped into MOATA.) We have

$$\left.\begin{aligned} &\rho(t) = 0 \ , \ t < t_0 \\ \text{and} \quad &\rho(t) = \rho \ , \ t \geqslant t_0 \end{aligned}\right\} \quad \ldots(4)$$

where ρ is the constant value of the adjusted reactivity. Our kinetics Equation (1) then becomes

$$\left.\begin{aligned} (1-\rho)\ T_c \frac{dp(t)}{dt} &= \rho\ p(t) + R_c\left[p(t) - p(t-T_\infty)\right] \ , \ t \geqslant t_0 \ , \\ p(t) &= p_0 \ , \ t < t_0 \\ p(t_0) &= p_0/(1-\rho) \ , \end{aligned}\right\} \quad \ldots(5)$$

which we will solve (using computers) and compare with actual kinetics experiments to be carried out on MOATA. From a kinetic experiment we will calculate a value for ρ that according to our description (5) fits the experiment in an overall time sense. We will then compare results of theory with experiment for $p(t)$ at the individual times $t = 0,1,2,\ldots,500$ sec encountered during the course of the experiment. Before we do however, we will need some more maths...

2.2 STEADY STATE

In order to understand some aspects of our kinetics Equation (1) we note that if $\rho(t) = 0$ then the equation becomes

$$T_c \frac{dp(t)}{dt} = R_c \left[p(t) - p(t-T_\alpha)\right] \ ,$$

which we may verify has the solution

$$p(t) = p_0 \text{ (a constant)} \quad \ldots(6)$$

(with then

$$\frac{dp(t)}{dt} = 0) \ .$$

The power thus remains steady. (In practice we find that ρ tends to undergo small random variations of small magnitude and consequently the power only remains approximately steady.)

2.3 APPROXIMATIONS TO OUR DESCRIPTION

The constants for MOATA were written in terms of a parameter c, as

variation of c corresponds to taking different approximations in a more detailed study of the actual kinetics process. As stated previously the choice

$$c = 0.29 \quad (T_c = 35 \text{ sec})$$

corresponds to what is called a 2 delayed group approximation. Other choices are appropriate to other approximations although the more detailed kinetics study suggests the above choice should be the best! The practical limits to c are

$$\left.\begin{aligned} &0.117 \leqslant c \leqslant 1 \\ &\text{and correspondingly} \\ &81 \geqslant T_c \geqslant 13 \text{ sec} \end{aligned}\right\} \quad \ldots(7)$$

If we take c = 1 (a so called 1 delayed group approximation) we find that Equation (5) readily yields an analytic solution as the delayed term vanishes since

$$R_1 = 0 \ .$$

The equation hence becomes the differential equation

$$\left.\begin{aligned} &(1-\rho)\ T_1 \frac{dp}{dt} = \rho\ p(t) \ , \ t \geqslant t_0 \ , \\ &\text{with initial condition} \\ &p(t_0) = p_0/(1-\rho) \ . \end{aligned}\right\} \quad \ldots(8)$$

We may readily verify (by differentiation) that the solution is in fact

$$p(t) = \frac{p_0}{1-\rho} e^{\rho(t-t_0)/(1-\rho)\ T_1)} \ , \ t \geqslant t_0 \ , \quad \ldots(9)$$

where e^x is the exponential function with the following properties

$$\left.\begin{aligned} &e = 2.718282 \\ &e^{x+y} = e^x\ e^y \\ &\frac{de^x}{dx} = e^x \end{aligned}\right\} \quad \ldots(10)$$

and which is readily available with our computer (Chapter 4) using a FORTRAN function EXP(X). Given t_0, p_0 and ρ we may then (using our computer) readily produce a table for p(t) evaluated at intervals of a second for comparison with experimental results. We will however mainly be concerned with solving Equation (5) as it stands (with c = 0.29).

2.4 ANALYSIS OF THE MOATA SAFETY ROD DROP EXPERIMENT

The safety rod drop experiment consists of the sudden insertion of a large negative amount of 'reactivity' ($\rho \simeq -1$ \$) which is normally inserted to shut down the reactor (p(t) = 0). On account of delayed neutrons the reactor power does not instantaneously cut off, as we will see from our calculations, but

certainly $p(t) \to 0$ as $t \to \infty$. Integrating Equation (5) over the interval $t = t_0$ to $t = \infty$ (in practice 400 sec will do) we have

$$(1-\rho)\ T_c \int_{t_0}^{\infty} \frac{dp(t)}{dt}\, dt = \rho \int_{t_0}^{\infty} p(t)\, dt + R_c\left[\int_{t_0}^{\infty} p(t)\, dt - \int_{t_0}^{\infty} p(t-T_\infty)\, dt\right]$$

Simplifying the above using the following properties of the integrals

$$\int_{t_0}^{\infty} \frac{dp(t)}{dt}\, dt = \Big[p(t)\Big]_{t_0}^{\infty} = p(\infty) - p(t_0) = -p_0/(1-\rho)$$

$$\int_{t_0}^{\infty} p(t-T_\infty)\, dt = \int_{t_0-T_\infty}^{\infty} p(u)\, du = \int_{t_0-T_\infty}^{t_0} p(u)\, du + \int_{t_0}^{\infty} p(u)\, du$$

$$= \int_{t_0-T_\infty}^{t_0} p_0\, du + \int_{t_0}^{\infty} p(u)\, du = p_0 T_\infty + \int_{t_0}^{\infty} p(u)\, du$$

we have

$$(1-\rho)\ T_c\left[-p_0/(1-\rho)\right] = \rho \int_{t_0}^{\infty} p(t)\, dt + (1-c)\ \left(\frac{T_c}{T_\infty} - 1\right)\ \left[-p_0\ T_\infty\right]$$

and finally the working result we require is obtained as

$$\rho = -p_0\ T_1 / \int_{t_0}^{\infty} p(t)\, dt \quad \text{(independent of c)}\ , \qquad \ldots(11)$$

where $T_1 = 13$ sec (Equation (3)). Given an experimental tabulation of $p(t_0)$, $p(t_0 + 1)$, $p(t_0 + 2)$,..., that is the power every second, then we may use a numerical integration procedure, say the trapezoidal rule (Chapter 3), to evaluate the integral required in the above. In addition we find that $p(t)$ is only approximately constant for $t < t_0$ and so we define an average value for p_0 thus

$$p_0 = \frac{1}{t_0} \int_{0}^{t_0} p(t)\, dt\ , \qquad \ldots(12)$$

which proves to be adequate in our analysis of the experiment.

2.5 ANALYSIS OF THE MOATA CONTROL ROD WITHDRAWAL EXPERIMENT

The control rod withdrawal experiment consists of the addition of a small amount of 'reactivity' ($\rho \simeq 0.1$ \$) by the withdrawal of a rod normally used for fine control of the reactor. Unlike the shutting down operation (Section 2.4) which is almost instantaneous (for safety reasons) the withdrawal operation of

the control rod may take several seconds. Nevertheless we will still consider the operation to be instantaneous. With the present experiment $p(t) \to \infty$ as $t \to \infty$ and hence to avoid damage to the reactor that would result from extreme power levels, the power increase is cut back after about 450 sec by reversing the control rod ($\rho(t) < 0$, $t > 450$). The time at which this reversal takes place will be denoted by the t_e ($\simeq 450$ sec) and then Equation (5) holds only for the time interval

$$t_0 \leqslant t \leqslant t_e \ .$$

After a sufficient time following the withdrawal we find that

$$p(t) = be^{t/\tau} \qquad \ldots(13)$$

where b and τ (the asymptotic period) are constants. Substituting Equation (13) into Equation (5) using the standard results of calculus (from Equation (10))

$$\frac{d}{dt}\left(be^{t/\tau}\right) = b\frac{d}{dt}e^{t/\tau} = \frac{b}{\tau}e^{t/\tau}$$

along with the exponential property

$$e^{(t-T_\infty)/\tau} = e^{t/\tau}\, e^{-T_\infty/\tau}$$

we have

$$(1-\rho)\ T_c \frac{b}{\tau} e^{t/\tau} = \rho be^{t/\tau} + R_c\left[be^{t/\tau} - be^{t/\tau}\, e^{-T_\infty/\tau}\right]$$

hence $$(1-\rho)\frac{T_c}{\tau} = \rho + R_c\left(1-e^{-T_\infty/\tau}\right)$$

and $$\rho = \left[\frac{T_c}{\tau} - R_c\left(1-e^{-T_\infty/\tau}\right)\right]\Big/\left(1+\frac{T_c}{\tau}\right) \text{ (dependent on c)} \ , \qquad \ldots(14)$$

which is called the inhour equation. If from the experimental results we can calculate τ, then the inhour Equation (14) will immediately yield the 'reactivity' ρ that produced the asymptotically obtainable exponential power increase of Equation (13).

Pursuing the calculation of τ we consider S = a suitable time step, say 50 sec, and we define

$$I_1 = \int_{t_0}^{t_0+S} p(t)\ dt \ , \quad I_2 = \int_{t_0+S}^{t_0+2S} p(t)\ dt \ , \ldots,$$

$$I_n = \int_{t_0+(n-1)S}^{t_0+nS} p(t)\ dt \quad \text{(provided } t_0 + nS \leqslant t_e\text{)} \ . \qquad \ldots(15)$$

For n sufficiently large Equation (13) holds and

$$I_n = \int_{t_0+(n-1)S}^{t_0+nS} be^{t/\tau}\,dt = b\tau\left[e^{t/\tau}\right]_{t_0+(n-1)S}^{t_0+nS}$$

$$= b\tau\left[e^{(t_0+nS)/\tau} - e^{(t_0+(n-1)S)/\tau}\right] .$$

Replacing n by n-1 in the above gives immediately

$$I_{n-1} = b\tau\left[e^{(t_0+(n-1)S)/\tau} - e^{(t_0+(n-2)S)/\tau}\right]$$

$$= b\tau\left[e^{(t_0+nS)/\tau} - e^{(t_0+(n-1)S/\tau}\right] e^{-S/\tau}$$

$$= I_n\, e^{-S/\tau}$$

therefore $e^{S/\tau} = I_n/I_{n-1}$

and finally

$$\tau_n = S/\ell n\ (I_n/I_{n-1}) \ , \quad (\ell n\ x = \log_e x) \ , \qquad \ldots(16)$$

where we have added a subscript to τ to designate the dependence on n. The calculation procedure amounts to applying numerical integration, say the trapezoidal rule (Chapter 3), to the experimental power tabulation according to Equation (15) for n=1,2,... with subsequent calculation of τ_n from Equation (16). When τ_n is effectively constant for n sufficiently large, n=N, that is given the error indicator

$$f_n = \left|(\tau_n - \tau_{n-1})/\tau_n\right| \qquad \ldots(17)$$

and an index N such that

$$f_N \leqslant f_n \quad \text{for any n,} \qquad \ldots(18)$$

then we take

$$\tau = \tau_N \ . \qquad \ldots(19)$$

The table below gives an idea of the dependence of ρ on c for values of τ that could be encountered in the experiment.

TABLE 1

ρ (DOLLARS) AS A FUNCTION OF c AND τ FROM INHOUR EQUATION (14)

c	40	50	60	70	80	τ sec
0.117	0.135	0.119	0.108	0.099	0.091	Table entries are ρ
0.29	0.187	0.163	0.144	0.130	0.118	
1.0	0.245	0.206	0.178	0.157	0.140	
more exact theory *	0.190	0.164	0.145	0.129	0.117	

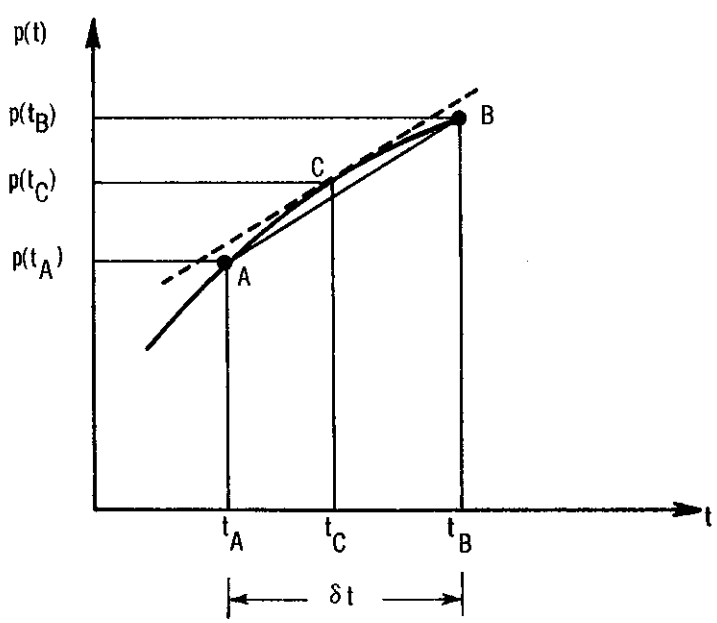
* the more exact theory corresponds to a 6 delayed group approximation.

From Table 1 we certainly see that our 2 delayed group description (c = 0.29) is adequate compared with the more exact theory and that the choice of any other c is likely to introduce serious errors in ρ even though by so doing we will reproduce the experimental value for τ.

2.6 DETAILED KINETICS CALCULATION

In order to solve our kinetics Equation (5) we adopt an entirely numerical approach and we are thereby introduced to the subject of numerical analysis. Many different approaches are possible - here we will pursue a method which introduces ideas as quickly as possible.

Say, by some means or other, we know the solution we seek up to the point A, as sketched, and we wish to extend the solution to the point B corresponding to the given time t_B.



From the sketch we note that the slope of the chord AB = the slope of the curve at some point C between A and B

$$\frac{p(t_B) - p(t_A)}{\delta t} = \frac{dp}{dt}\Big|_C = \left(\frac{dp}{dt} \text{ evaluated at } t = t_C\right) . \qquad \ldots(20)$$

We assume that the slope of the curve at C is an average of the slopes at A and B

$$\frac{dp}{dt}\Big|_C = \frac{1}{2}\left(\frac{dp}{dt}\Big|_A + \frac{dp}{dt}\Big|_B\right) , \qquad \ldots(21)$$

then Equation (20) becomes

$$p(t_B) = p(t_A) + \frac{\delta t}{2}\left(\frac{dp}{dt}\Big|_A + \frac{dp}{dt}\Big|_B\right) . \qquad \ldots(22)$$

Now we know $\frac{dp}{dt}\Big|_A$ from the equation we are seeking to solve Equation (5)

$$\frac{dp}{dt}\Big|_A = \frac{\rho + R_c}{(1-\rho)\ T_c}\, p(t_A) - \frac{R_c}{(1-\rho)\ T_c}\, p(t_A - T_\infty) \qquad \ldots(23)$$

since $p(t_A)$ is known along with $p(t_A - T_\infty)$ (assumed). Similarly

$$\frac{dp}{dt}\Big|_B = \frac{\rho + R_c}{(1-\rho)\ T_c}\, p(t_B) - \frac{R_c}{(1-\rho)\ T_c}\, p(t_B - T_\infty) \qquad \ldots(24)$$

although here $p(t_B)$ is unknown. Nevertheless we substitute the above into Equation (22) and we have

$$p(t_B) = p(t_A) + \frac{\delta t}{2}\left[\frac{\rho + R_c}{(1-\rho)\ T_c}\, p(t_A) - \frac{R_c}{(1-\rho)\ T_c}\, p(t_A - T_\infty)\right]$$

$$+ \frac{\delta t}{2}\left[\frac{\rho + R_c}{(1-\rho)\ T_c}\, p(t_B) - \frac{R_c}{(1-\rho)\ T_c}\, p(t_B - T_\infty)\right] ,$$

$$\text{therefore } \left[1 - \frac{\delta t}{2}\frac{\rho + R_c}{(1-\rho)\ T_c}\right] p(t_B) = \left[1 + \frac{\delta t}{2}\frac{\rho + R_c}{(1-\rho)\ T_c}\right] p(t_A)$$

$$- \frac{\delta t}{2}\frac{R_c}{(1-\rho)\ T_c}\left[p(t_A - T_\infty) + p(t_B - T_\infty)\right] .$$

Hence we obtain the relation we seek

$$p(t_B) = f\, p(t_A) - g\left[p(t_A - T_\infty) + p(t_B - T_\infty)\right] , \qquad \ldots(25)$$

where $$f = \left[1 + \frac{t}{2}\frac{\rho + R_c}{(1-\rho)\ T_c}\right] \Big/ \left[1 - \frac{\delta t}{2}\frac{\rho + R_c}{(1-\rho)\ T_c}\right] \qquad \ldots(26)$$

and $$g = \frac{\delta t}{2}\frac{R_c}{(1-\rho)\ T_c} \Big/ \left[1 - \frac{\delta t}{2}\frac{\rho + R_c}{(1-\rho)\ T_c}\right] \qquad \ldots(27)$$

are constants for a particular calculation if we select a fixed step length (δt). Since $p(t_0)$ is known (given initial condition) we may start with $t_A = t_0$ then Equation (25) yields an approximation for $p(t_0 + \delta t)$. Again knowing $p(t_0 + \delta t)$ (we have just calculated it) we progress the solution to $p(t_0 + 2\delta t)$ using Equation (25). We may repeat this process for $t_0 + 3\delta t$, $t_0 + 4\delta t$,..., until we have available an approximate solution for $p(t)$ covering the time interval of interest in our experiment (to t_e).

It turns out that a step length of $\delta t = 1$ sec is suitable for our study. As for the experiment we use the times

$$t = 0,1,2,\ldots,\ t_0 - 1,\quad t_0,\qquad t_i = i\text{-}1 \text{ sec}$$

where $\quad i = 1,2,3,\ldots,\ t_0,\qquad t_0 + 1,\ t_0 + 2,\ t_0 + 3,\ t_0 + 4,\ldots,\ t_e + 1,$

then $\quad p(t) = p_0, p_0, p_0, \ldots, p_0, \dfrac{p_0}{1-\rho},\qquad p_i = fp_{i-1} - g\left[p_{i-1-T_\infty} + p_{i-T_\infty}\right]$

(initial condition → |prompt jump| → numerical solution)

briefly indicates our computational procedure. Presented in a form to assist later coding in FORTRAN (Chapter 4) the computational procedure amounts to the following steps carried out one after the other...

input t_0, p_0, ρ, c, t_e ←

$T_c = 4 + 9/c$ (equation (2))

$T_\infty = 4$

$R_c = (1-c)\left(\dfrac{T_c}{T_\infty} - 1\right)$ (equation (2))

$f = \left[1 + \dfrac{1}{2}\dfrac{\rho + R_c}{(1-\rho)\ T_c}\right] \Big/ \left[1 - \dfrac{1}{2}\dfrac{\rho + R_c}{(1-\rho)\ T_c}\right]$ (equation (26))

$g = \dfrac{1}{2}\dfrac{R_c}{(1-\rho)\ T_c} \Big/ \left[1 - \dfrac{1}{2}\dfrac{\rho + R_c}{(1-\rho)\ T_c}\right]$ (equation (27))

$p_i,\ (i = 1,2,\ldots,\ t_0) = p_0$ (equation (5))

$p_{t_0+1} = p_0/(1-\rho)$ (equation (5))

$i = t_0 + 2,\ t_0 + 3,\ldots,\ t_e + 1$ ←

$p_i = fp_{i-1} - g\left[p_{i-1-T_\infty} + p_{i-T_\infty}\right]$ (equation (25))

$t_i = i\text{-}1$ →

output t_i, p_i

then do next →

Refinements such as inputting experimental results and plotting comparison results (on the CALCOMP plotter) may be readily inserted in the above as required.

Figures 4 and 5 show results obtained for comparison of experimental and theoretical estimates of p(t) for typical experiments.

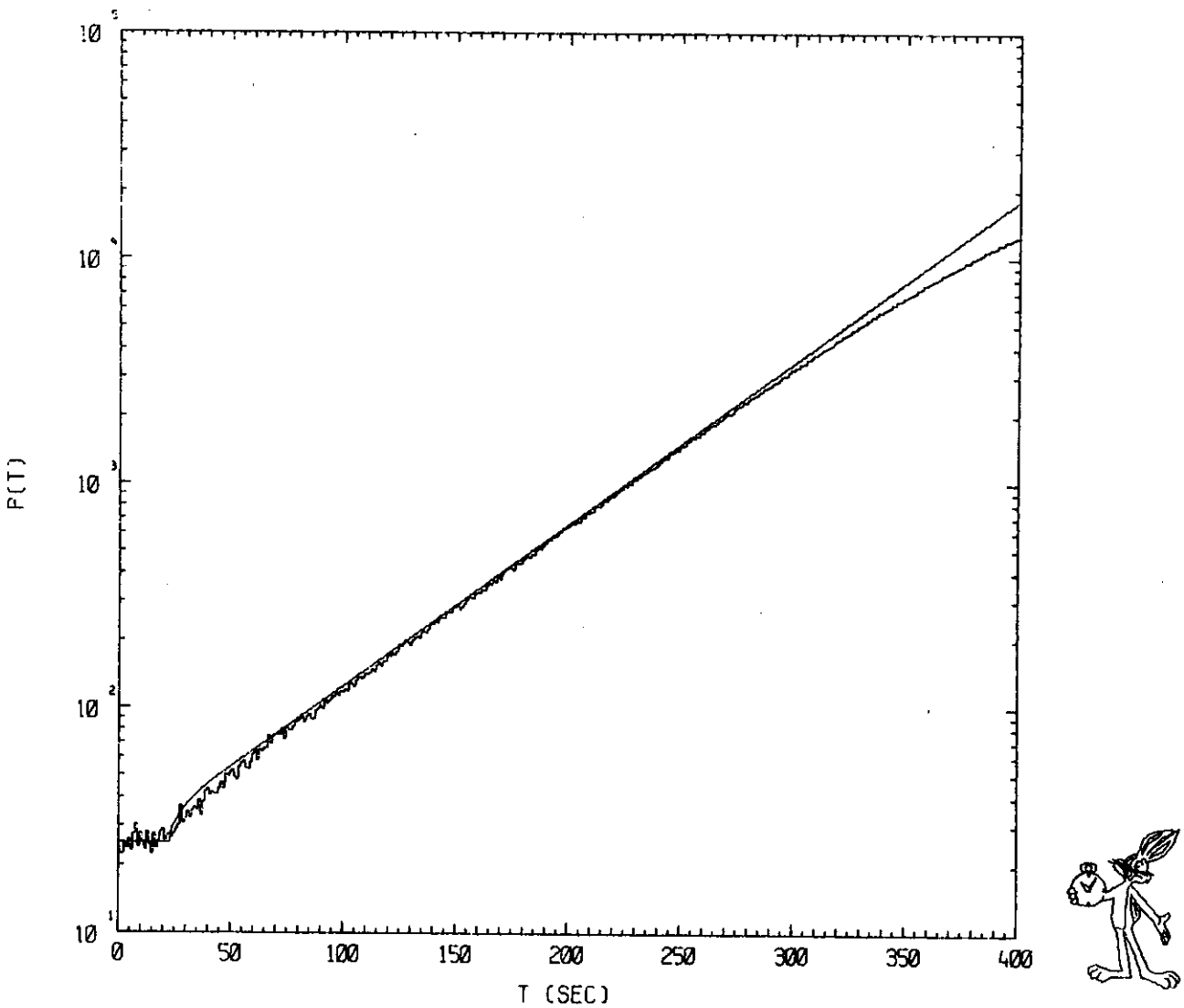


FIGURE 4 MOATA Control Rod Withdrawal with ρ, C and GP = 0.144, 0.290, 0

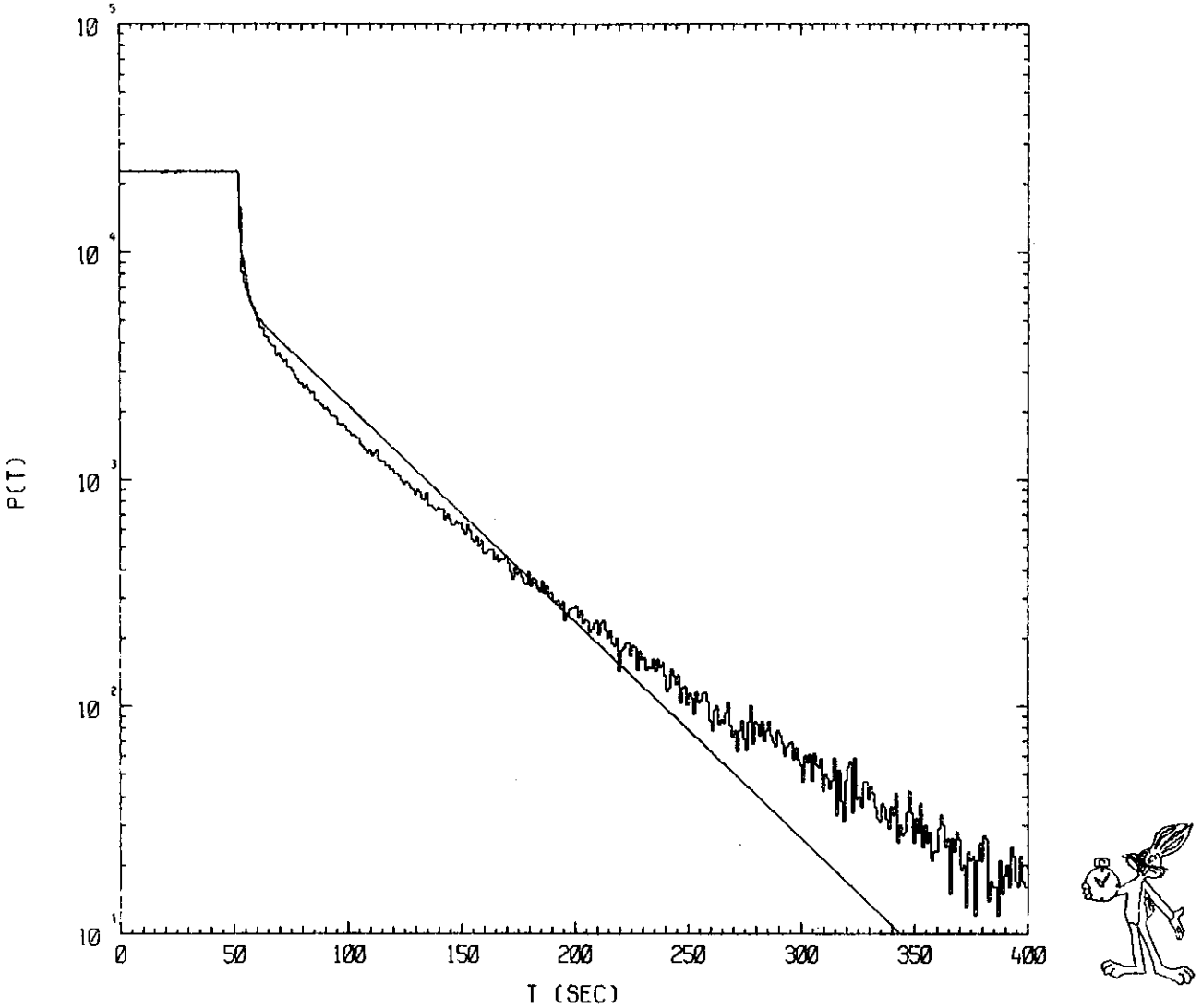


FIGURE 5 MOATA Safety Rod Drop with ρ, C and GP= -1.150, 0.290, 0

CHAPTER 3

NUMERICAL INTEGRATION

Lecture by

B.E. CLANCY

CONTENTS

3.1 INTRODUCTION

The concept of integration is usually introduced - and rightly so - by way of the problem of finding 'the area under a curve'. We are exposed to a diagram such as that in Figure 1 with the curve described by the equation $y = f(x)$ and we learn that the shaded region has area

$$S = \int_a^b f(x)\, dx \quad .$$

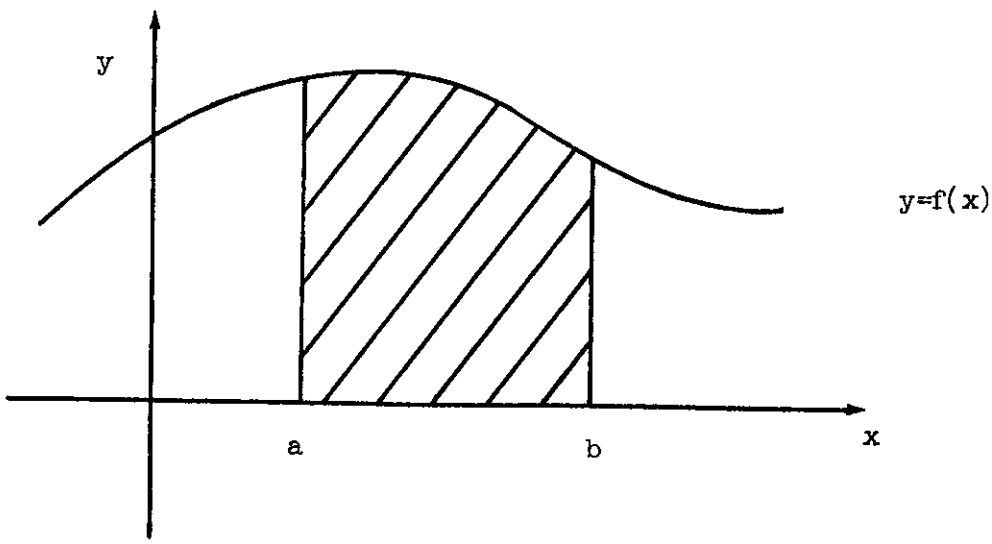


FIGURE 1

There can be no quarrel with this presentation because all integrals can be thought of in these terms.

It was certainly my own experience (and it may be yours too) that this first exposure was followed by a good deal of practice devoted to acquiring skills in tricks for calculating integrals by expressing the answer in terms of formulae and then looking up tables to evaluate the formulae.

To evaluate an integral using a computer it is necessary to learn a new set of tricks - but all of them are based on the idea of an integral being the area under a curve.

3.2 THE TRAPEZOIDAL RULE

The simplest numerical integration procedure to understand - and one of the easiest to implement - is the trapezoidal rule. If, as in Figure 2, we suppose the segment (a,b) of our original figure divided into 3 equal parts, draw verticals from the subdividing points up to our curve $y = f(x)$ and join the points of intersection with short straight lines, it is clear that a reasonable approximation to the shaded area in Figure 1 is obtained by calculating

the areas of the 3 trapeziums and adding them together.

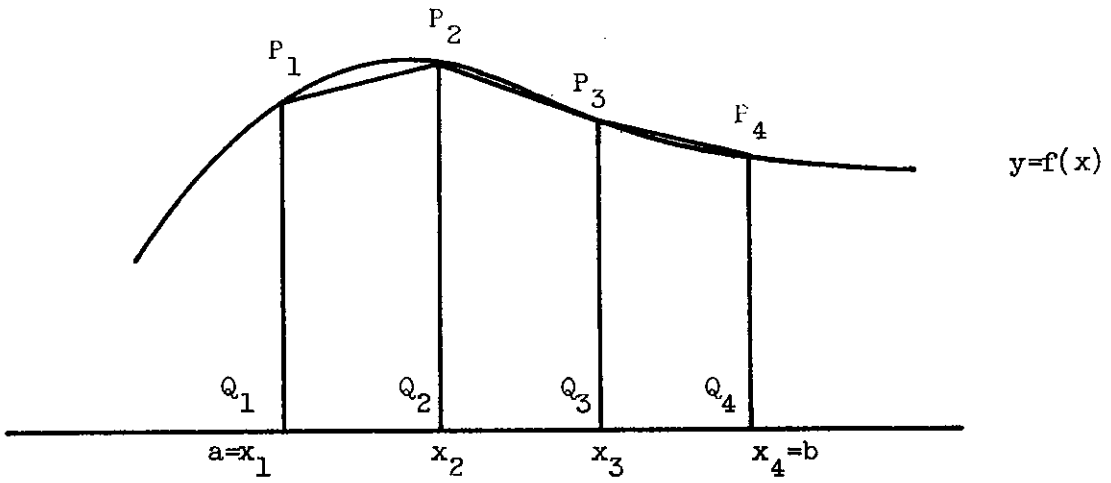


FIGURE 2

Calculation of the individual areas is a straightforward task. The lines $Q_1 P_1$ and $Q_2 P_2$ are of lengths $f(x_1)$ and $f(x_2)$ respectively while the base $Q_1 Q_2$ has length $(b-a)/3$. The area of $Q_1P_1P_2Q_2$ is thus simply

$$\frac{1}{2}\,(f(x_1) + f(x_2)) \;.\; \frac{b-a}{3}$$

and our approximation to the integral is

$$S \approx \frac{b-a}{6}\,[\,f(x_1) + 2f(x_2) + 2f(x_3) + f(x_4)]$$

That this simple expression should approximate the integral follows from the fact that the sequence of straight line segments $P_1\ P_2$, $P_2\ P_3$, $P_3\ P_4$ follows the curve $y = f(x)$ fairly closely. We can improve our approximation by using more than three subdivisions so that the corresponding sequence of straight line segments together approximate the shape of the curve $y = f(x)$ more closely. If we divide the distance (a,b) into N equal subdivision by points $x_1, x_2, x_3, \ldots, x_{N+1}$ with $x_1 = a$ and $x_{N+1} = b$ and with each pair of points a distance $(b-a)/N$ apart, our approximation for the integral becomes

$$S \approx \frac{(b-a)}{2N}\left[f(x_1) + 2\sum_{i=2}^{N} f(x_i) + f(x_{N+1})\right]$$

For a sufficiently accurate answer we must choose N large enough. A typical procedure for finding out whether or not a large enough value of N has been used is to repeat the calculation using 2N subdivisions and to see whether or not the new answer differs significantly from the old.

It may be interesting to see how the trapezoidal rule tends to get

closer to the correct answer as N increases. Let us therefore consider the case where $f(x) = x^2$ and the integral

$$S = \int_0^1 x^2 \, dx = \frac{1}{3} x^3 \Big|_0^1 = \frac{1}{3} \quad .$$

The trapezoidal rule gives, since $f(x_i) = x_i^2 = \left(\frac{i-1}{N}\right)^2$,

$$S \simeq \frac{1}{2N} \left[0 + 2 \sum_{i=2}^{N} \left(\frac{i-1}{N}\right)^2 + 1 \right]$$

$$= \frac{1}{N^3} \left[1^2 + 2^2 + 3^2 + \ldots + (N-1)^2 \right] + \frac{1}{2N} \quad .$$

We can show that

$$1^2 + 2^2 + 3^2 + \ldots n^2 = \frac{1}{6} n \; (n+1 \; (2n+1)$$

so that our approximation is

$$S \approx \frac{1}{N^3} \left[\frac{1}{6} \; (N-1) \; \; (N) \; \; (2N-1) \right] + \frac{1}{2N}$$

$$= \frac{1}{3} + \frac{1}{6N^2} \quad .$$

As N grows larger the 'error' $\frac{1}{6N^2}$ becomes smaller and tends to zero as N tends to infinity. Even with N = 10 the error is only $\frac{1}{2}\%$.

In practice, of course, the summation is carried out numerically rather than analytically as was possible for our example. However, we do have an interesting converse of the trapezoidal rule for the range (0,1) that

$$\sum_{i=2}^{N} f\left(\frac{i-1}{N}\right) \approx N \int_0^1 f(x) \, dx - \frac{1}{2} f(0) - \frac{1}{2} f(1)$$

which gives us a possible way of approximating the analytic sum of a series (provided that we can do the integrals analytically).

The trapezoidal rule is only one of a set of procedures for performing numerical integrations by repeatedly subdividing the baseline (a,b) and approximating the areas of the small strips. In the trapezoidal rule the curve $y = f(x)$ is approximated by small straight line segments. In the second procedure of the set, pairs of strips are used (so that N must be even) and the curve $y = f(x)$ is approximated by a parabola $y = \alpha x^2 + \beta x + \gamma$ with coefficients α, β, γ chosen so that the parabola and the curve $y = f(x)$ intersect at the top corners of the strips. This procedure, called Simpson's rule, leads to an approximation of the form

$$S \approx \frac{b-a}{6M}\{f(x_1) + 4f(x_2) + 2f(x_3) + 4f(x_4) + 2f(x_5) + \\ + 2f(x_{2M-1}) + 4f(x_{2M}) + f(x_{2M+1})\}$$

An approximation using a particular value for N = 2M could be improved by doubling N and M and comparing the new answer with the old.

More sophisticated rules have been constructed in which the true curve is fitted by higher order polynomial expressions but these tend not to be used in computing applications.

3.3 QUADRATURE RULES

Both the trapezoidal rule and Simpson's rule involve the evaluation of the function f(x) at a sequence of points in the interval (a,b), multiplication of the function values by numbers from another sequence and addition of the products to give an approximation to the integral. They are all of the form

$$S \approx \sum_{i=1}^{L} w_i \, f(x_i)$$

with the points x_i equally spaced. This general form is sometimes called a quadrature formula.

If the requirement that the points be equally spaced is abandoned and instead the ordinates x_i, the weights w_i and their number L are chosen so that a function f(x) of a particular type will be integrated exactly the quadrature formula is said to be of a Gaussian type.

For L = 2 any polynomial of order less than or equal to 3 can be integrated exactly over the interval (a,b) if we choose

$$x_1 = \frac{b}{2}\left(1 - \frac{1}{\sqrt{3}}\right) + \frac{a}{2}\left(1 + \frac{1}{\sqrt{3}}\right) \qquad w = \frac{1}{2}(b-a)$$

$$x_2 = \frac{b}{2}\left(1 + \frac{1}{\sqrt{3}}\right) + \frac{a}{2}\left(1 - \frac{1}{\sqrt{3}}\right) \qquad w = \frac{1}{2}(b-a)$$

With a = -1, b = +1 these become

$$x_1 = -\frac{1}{\sqrt{3}} \qquad w_1 = 1$$

$$x_2 = +\frac{1}{\sqrt{3}} \qquad w_2 = 1$$

and any integral of the form

$$\int_{-1}^{1} (p + qx + rx^2 + sx^3)\, dx$$

will be calculated exactly by use of the formula, which is called the Gauss-Legendre formula of order 2.

For L = 3 any polynomial up to fifth order is calculated exactly for the interval (-1, 1) by using

$$x_1 = -\sqrt{\frac{3}{5}} \qquad w_1 = \frac{5}{9}$$

$$x_2 = 0 \qquad w_2 = \frac{8}{9}$$

$$x_3 = +\sqrt{\frac{3}{5}} \qquad w_3 = \frac{5}{9}$$

Reference books on numerical analysis list tables of weights and ordinates for Gauss-Legendre formulae of orders up to 96. In many computer applications 16 would be the maximum order used with which polynomials of order up to 33 would be integrated exactly. For integrating functions which are not polynomials, the Gauss-Legendre formulae are used on the basis that a high order polynomial will approximate most functions quite well. The reference books also give tables for the exact integration of integrals of the form

$$\int_{-1}^{1} \frac{(\text{polynomial in } x)}{\sqrt{(1-x^2)}}\, dx \qquad \text{Gauss-Chebyshev}$$

$$\int_{0}^{\infty} e^{-x}\, (\text{polynomial in } x)\, dx \qquad \text{Gauss-Laguerre}$$

and for a number of others.

3.4 THE MONTE CARLO METHOD

This technique is rarely used for simple integrals of the type we have been examining, but it does have application to multiple integrals like

$$\int_a^b \left[\int_{u(x)}^{v(x)} \left[\int_{p(x,y)}^{q(x,y)} f(x,y,z)\, dz \right] dy \right] dx$$

We will, however, illustrate the method by considering the simple integral

$$\int_0^1 \sqrt{(1-x^2)}\, dx$$

which is just the shaded area in Figure 3.

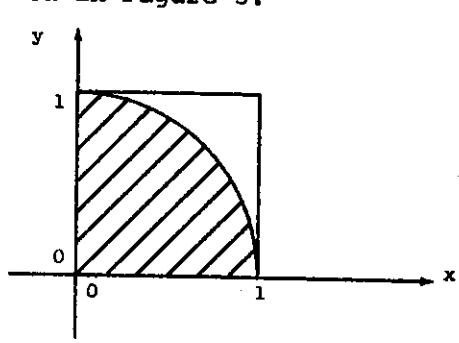


FIGURE 3

The integration techniques we have talked about so far involve the evaluation of the function to be integrated, an evaluation which has to be performed many times. It may happen that this function evaluation is very costly in terms of computer time while it may be very simple to determine on which side of the curve $y = f(x)$ any given point (x,y) lies. This is almost the case for the example we have chosen. To evaluate $\sqrt{(1-x^2)}$ for a given x takes about 3.6×10^{-4} seconds on the AAEC main computer but given x and y to decide whether or not $x^2 + y^2$ is greater than 1.0 takes about 1.5×10^{-4} seconds. This second test determines whether or not the point x,y lies outside $y = \sqrt{(1-x^2)}$.

We can think of the Monte Carlo procedure as a game of dart playing using Figure 3 as a target. If we assume that the dart player lands his darts uniformly over the outlined square in Figure 3, then the ratio of the number of darts landing in the shaded area to the total number of darts will be approximately the ratio of the area of the shaded figure to the area of the square. In the Monte Carlo procedure we could repeatedly choose a pair of random values for x and y within the square and find out whether or not the point (x,y) is within the circle. Our approximation to the integral would be

$$\frac{\text{Number of hits within circle}}{\text{Number of trials}}$$

3.5 AUTOMATIC INTEGRATION ROUTINES

Confronted with the need to evaluate a simple integral

$$\int_a^b f(x)\ dx$$

numerically the casual programmer will often make use of a standard integration subroutine if one is available. Such a standard routine should only need to be 'handed' the limits (a,b), a procedure for evaluating the function f(x) given any x and lastly the accuracy with which the integral is to be determined (ϵ_r). We shall now look briefly at one possible simple procedure based on successive applications of the trapezoidal rule with the interval (a,b) divided into 1,2,4,8,16,... subintervals.

Let us write down the result of using just this a few times: putting $h = b-a$ the simplest answer is

$$S_0 = \tfrac{1}{2} h_0 [f(a) + f(b)] \quad , \qquad \text{where } h_0 = h \ .$$

With one intermediate point we find

$$S_1 = \tfrac{1}{2} h_1 [f(a) + 2f(a+h_1) + f(b)] \quad , \qquad h_1 = \tfrac{1}{2} h_0 ,$$

and with three intermediate points we find

$$S_2 = \tfrac{1}{2} h_2 [fa + 2f(a+h_2) + 2f(a+2h_2) + 2f(a+3h_2) + f(b)] \quad , \qquad h_2 = \tfrac{1}{2} h_1 .$$

We notice that $f(a+2h_2) \equiv f(a+h_1)$, hence we can, after a bit of manipulation, find a connection between S_2 and S_1 viz.

$$S_2 = \tfrac{1}{2} S_2 + h_2 [f(a+h_2) + f(a+3h_2)] \quad .$$

If we were to go on repeating the process described above we would always be able to discover a relationship between S_n and S_{n+1}, the two successive approximations to the integral and this would be

$$S_{n+1} = \tfrac{1}{2} S_n + h_{n+1} \sum$$

where $h_{n+1} = \tfrac{1}{2} h_n$

and $\sum$ is simply the sum of the values of the function f(x) at the new subdivision points

i.e. $\sum = f(a+h_{n+1}) + f(a+3h_{n+1}) + f(a+5h_{n+1}) + \ldots + f(a+\overline{2n+1}\, h_{n+1})$.

A measure of the relative error in the latest approximation would be

$$\varepsilon = |(S_{n+1} - S_n)/S_n|$$

and the process of repeated halvings of the interval would be continued until the corresponding value of ε is as small as required ($\varepsilon \leqslant \varepsilon_r$).

3.6 EXERCISES

When you have completed the exercises set down in the FORTRAN course (Chapter 4) you may care to get more practice by writing a FORTRAN programme to evaluate

$$\int_0^1 \sqrt{(1-x^2)}\, dx$$

using the trapezoidal rule with 100 subdivisions for the interval (0,1).

Then try a programme to do the same integral by the Monte Carlo method using 10,000 trials. To get a uniformly distributed random number in the FORTRAN variable X use the statement

```
X = RAND(0)
```

each time a new random value is wanted in the range (0,1).

CHAPTER 4

PROGRAMMING THE IBM360 IN FORTRAN IV

Lecture by

J.M. BARRY

CONTENTS

4.1 INTRODUCTION

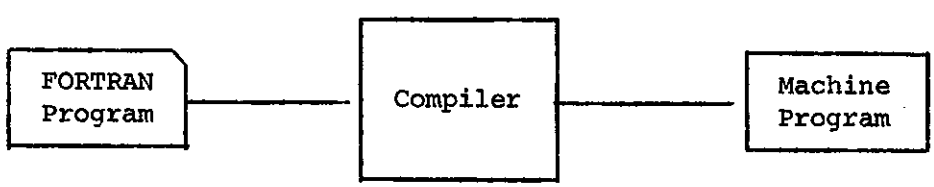
Each digital computer built is capable of obeying a number of basic instructions. These instructions will vary for different computers but they have many attributes in common:

(i) The ability to perform the four arithmetic operations (+, -, x, ÷).

(ii) The ability to perform logical operations (is A ⩾ B).

(iii) The ability to perform 'housekeeping' instructions (e.g. moving numbers from core store to registers where arithmetic and logical operations may be performed on them).

For a programmer to communicate with the computer at this most fundamental level it would be necessary for him to develope programs in the basic machine language of the computer at his disposal. In the early days of computing it was necessary for scientists and mathematicians to concern themselves with the intricasies of binary coding. The long delays and inconvenience of this form of man-machine communication accelerated the growth of programming languages that could be more readily used by the problem solver. Many languages (FORTRAN, ALGOL, PLI, APL, ACL etc.) have been developed for scientific, commercial and other applications. FORTRAN is chosen as the vehicle for problem solving at this summer school due to its world wide acceptability as a scientifically oriented programming language. There are no computers that obey programs written in FORTRAN directly. It is necessary for 'high level' programs in languages such as FORTRAN to be translated into an appropriate set of machine language instructions. This process is known as compilation.



The FORTRAN source statements are translated to a set of machine language instructions by a FORTRAN compiler. The compiler is itself a program (usually supplied by the machine manufacturer) that first checks to ensure the FORTRAN statements obey the 'rules' of the language (syntax analysis), and then supplies a set of machine instructions that will implement what the programmer has specified. When the compilation process is completed the machine instructions generated may be executed. The finer details of this process and the way it is implemented on the IBM360 will not be our concern at this summer school as we are primarily interested in using the computer as a tool for mathematical problem solving.

4.2 OVERVIEW OF FORTRAN PROGRAMMING

Let us first consider the steps involved in solving a sample problem and the FORTRAN program that could be developed to carry them out. When this is done we shall examine the various FORTRAN statements in closer detail.

Problem: If $18,000 is borrowed at a rate of 7% (monthly reducable) and repayments of $200 each month are made, then how many years will it take to repay the loan?

Before we can program a digital computer to solve a problem it is necessary for us to be able to logically detail the steps that are needed to solve the problem, in much the same way we would if we were going to tackle the problem with a desk calculating machine, sliderule, or pen and paper. Some people find it a help to draw a flowchart (Figure 1) showing the steps involved, while others prefer to visualise all the steps in their mind.

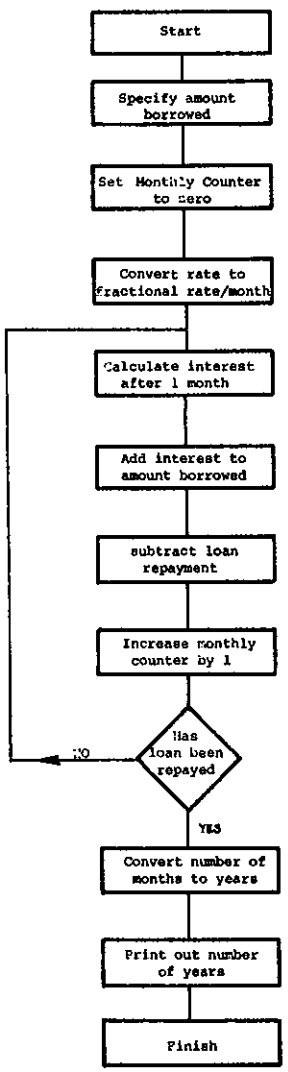


FIGURE 1 Flow Chart for Compound Interest Problem

From this flow chart the following program can be coded. At this point we will not concern ourselves with the formal rules for coding but just look at the end product (Figure 2).

```
1    5 6 7                                                        72 73    80
C       PROGRAM BY J.M. BARRY TO DETERMINE THE NUMBER OF
C       YEARS NECESSARY TO REPAY A LOAN.
C       THE PRINCIPAL BORROWED, INTEREST RATE AND MONTHLY REPAYMENT
C       ARE TO BE READ FROM A PUNCHED DATA CARD.
        READ(1,100)PRINC,RATE,PAYMNT
        MNCNTR=0
        FRATEM=RATE/(100.*12.)
      1 ADPRIN=PRINC*FRATEM
        PRINC=PRINC+ADPRIN
        PRINC=PRINC-PAYMNT
        MNCNTR=MNCNTR+1
        IF(PRINC.GT.0)GØ TØ 1
        YEARS=MNCNTR/12.
        WRITE(3,101)YEARS
  100   FØRMAT(3F10.3)
  101   FØRMAT(' NØ ØF YEARS = ',F10.3)
        STOP
        END
```

The data card necessary for this problem would be

```
         10        20        30
    18000.       7.      200.
```

Figure 2. Sample program for compound interest problem.

4.3 PUNCHING OF CARDS

To assist in the punching of cards, programmers usually use a standard coding sheet representing the 80 columns available on a punched card. Each line of the sheet represents a new card which may contain only one statement.

```
1    5 6 7                                                        72 73    80
C       J.M. BARRY STATEMENT EXAMPLE JAN 1974                    THIS
C       THE ABOVE IS A COMMENT                                   SECTION
        X=A+B                                                    NOT
   50   Y=9.-C                                                   USED
        P R INC = RATE * PRINC/100. + PRINC                      BY
            SUM = A+B+C+D+E+F+G+H+                               FORTRAN
      1     Ø+P+Q+R                                              PROGRAMS
```

Statements can be punched from columns 7 to 72. To assist the programmer to later recall aspects of a program a comment card (denoted by a C in column 1) may be placed anywhere within the punched deck. These are ignored by the FORTRAN compiler. Normally we shall commence our programs with a comment card to assist with the identification of the program.

Columns 1 through 5 can be used if desired to assign a statement label in the form of a number in the range 1 to 99999 (there is no need to choose labels in ascending order).

Blanks may be inserted within a statement to make it more readable, and may be considered as being removed in the compilation process. Should a statement be too long to fit on one card it is continued from column 7 of a subsequent card provided column 6 of this card contains a continuation character (Any character other than a blank or zero will suffice as a continuation character.

The character set available within the FORTRAN system consists of

(i) 26 capital letters A,B,C,...,Z.

(ii) 10 numerals 0,1,2,...,9.

(iii) 10 special characters +,-,*(multiplication),/(division),·,',(,),=,'

(iv) a blank (usually written ∅ if its presence is to be emphasised for punching.

4.4 ARITHMETIC CONSTANTS

We will treat 3 different types of constants sufficient for handling data (numbers) in most scientific problems.

(i) INTEGER (or fixed point) constants.

- a whole number without a decimal point whose absolute value is $\leqslant 2^{31} - 1$ (2147483647).

Valid integer constants 0 -5 +357 7005192

Invalid integer constants 27· 5,132 9812735997

(ii) REAL (or floating point) single precision constants.

- up to 7 decimal digits with a decimal point, with or without an exponent. The absolute magnitude is approximately 10^{-78} to 10^{75}.

Valid real single precision constants +0· 7·91 5·3E+2(=5.3x10^2)
5·3E2(5·3x10^2) -.051E-03 (-·051x10^{-3})

Invalid real single precision constants 1 3,471·2 1·E

(iii) REAL (or floating point) double precision constants.

- similar to (ii) but up to 16 digits are possible with a D exponent being necessary instead of E. Double precision constants will not

be necessary for the summer school problems.

The reason for the careful distinction drawn between the three types of constants is on electronic rather than mathematical grounds. The electronic 'hardware' necessary for INTEGER arithmetic operations is less sophisticated and consequently faster than that used for REAL arithmetic. By performing those operations that require no decimal point in integer mode considerable time savings can be made.

4.5 VARIABLES

A variable is a symbolic name used to identify a data item that will occupy a location (one word) of core storage. The actual address of this location is assigned by the compilation process. If we move a number into a variable it will replace the previous contents of that location.

```
TIME=0.
```

places zero in the location reserved for TIME. When a transfer is made from a location the previous contents remain unaltered.

```
X=TIME
```

This assigns the contents of the location reserved for TIME to that reserved for X without altering the contents of the location associated with TIME.

The '=' operation should be interpreted as the assignment of the result of the right hand side expression to the left hand side location. Consequently an expression such as

```
A=A+1.
```

does not yield any algebraic result but rather is interpreted as increasing the old value associated with A by 1. to give a new result also called A.

Variable names may have up to 6 characters (special characters are not permitted) the first of which must be alphabetic.

```
TIME , X3B , I5 , T
```

Variables like constants take an INTEGER or REAL form. Unless the programmer provides specifications to the contrary all variables commencing with I,J,K,L,M or N are INTEGER variables, while the remainder are single precision REAL variables.

4.6 INPUT AND OUTPUT

One way of assigning values to variables is through the direct use of an arithmetic expression:

```
X=6.3
```

Should one wish to alter the data on which the program is to operate without changing the program itself then a READ statement is needed.

The read statements initiate the reading of data cards (these are

separate from the program cards) and the transfer of the numbers on these cards to the variables in the READ lists.

```
      READ(1,100)PRINC,RATE,PAYMNT
          /    \   \______________/
         /      \  list of variables to be used
        /        \
device 1      number of a
card reader   FORMAT statement
```

Numbers are read from device 1 under the control of an editing (FORMAT) statement. The supplied FORMAT statement (100) will describe the way the punched data card is layed out. In this case

```
  100 FØRMAT(3F10.3)
```

indicates that 3 numbers are punched on the data card satisfying the format code F10.3.

```
          10|       20|      30|                                  80
   25000.       7.        300.
```

F10.3 signifies REAL constants without any exponent (F), 10 columns being kept for each number. Should there be no decimal point punched on the card one will automatically be assumed to exist 3 digits to the left of column 10. When the decimal point is punched the second parameter is ignored.

The output statement WRITE,functions in a similar manner.

```
      WRITE(3,101)YEARS
  101 FØRMAT(' NØ ØF YEARS = ',F10.3)
```

would display on the printer (device 3) output of the form

```
NØ ØF YEARS =     26.314
```

4.7 ARITHMETIC OPERATIONS AND EXPRESSIONS

Five arithmetic operations are available to FORTRAN users:

(i)	addition	+	e.g. A+B
(ii)	subtraction	-	e.g. A-B
(iii)	multiplication	*	e.g. A*B
(iv)	division	/	e.g. A/B
(v)	exponenciation	**	e.g. A**3 (A^3)

Expressions may be enclosed within parentheses as in normal algebra.

$(a+b)\ (c+d)$	(A+B)*(C+D)
$(a+b)^2$	(A+B)**2
$\frac{a}{bc}$	A/(B*C)

Parentheses are necessary to prevent two operations from appearing next to each other (should such a combination be possible)

X*-Y must be coded X*(-Y)

The sequence of operations in expressions is determined from the following

hierarchy and is consistent with normal mathematics.

(i) **

(ii) */ left to right precedence

(iii) +- left to right precedence

Consequently the expression

```
X+(Y/A)-(3.*U)+P*(S**4)/3.
```

could have been correctly abbreviated to

```
X+Y/A-3.*U+P*S**4/3.
```

The integer variables or constants deserve special mention. Division of one integer by another results in the truncation of any fractional remainder.

```
I=9
K=I/2
```

Would result in K taking the value 4. This property can often be exploited to the programers advantage in the testing for even integers

```
K=I-I/2*2
```

would assign 1 to K if I is odd and 0 if I is even.

Expressions should consist of variables or constants all in the same mode i.e. all REAL or all INTEGER. There is one exception to this rule in that the exponent of a REAL variable or constant may be INTEGER. The following are permitted forms of exponenciation:

```
V**2        V**A
(-V)**.45   V**(-I)
V**(-2)     I**3
```

The mode of a variable on the left hand side of an arithmetic assignment need not be the same as that of the expression on the right.

```
A=I+1
```

The compiler will arrange for the right hand side to be evaluated in INTEGER mode and the result converted to the REAL mode before it is stored away. Because of truncation in INTEGER division great care should be exercised in using this type of arithmetic.

4.8 SUPPLIED MATHEMATICAL FUNCTIONS

As there are a number of special mathematical functions or operations that are common to many problems, the FORTRAN compiler provides these as part of the normal system. To calculate the exponential function $y=e^x$ all we need do is code

```
Y=EXP(X)
```

To use a supplied mathematical function it is only necessary to follow the function name by an argument enclosed in parentheses. The result will be

returned as though the function name itself designated a variable in the program. The argument may be a variable, constant or arithmetic expression

```
A=EXP(A-C)+SQRT(15.)
```

A list of frequently required functions follows:

Mathematical Function	Function Name (Argument)
square root,	SQRT(X)
exponential, e^x	EXP(X)
natural logarithm, log x (or ln x)	ALOG(X)
sine of an angle (in radians), sin x	SIN(X)
cosine of an angle (in radians), cos x	CØS(X)
arctangent (result in radians), $\tan^{-1}x$	ATAN(X)
absolute value (real numbers), $\|x\|$	ABS(X)

Functions other than those supplied through the compiler are often necessary so FORTRAN allows a programmer to name and define his own special functions (Section 4.13).

4.9 TRANSFER OF CONTROL

Execution of a program will commence at the first executable statement and proceed through subsequent instructions in order unless a transfer of control statement is encountered. The simplest means of transfer of control is through an 'unconditional GØ TØ' statement.

```
56 READ(1,9)X
   WRITE(3,11)X
   GØ TØ 56
```

This section of program would cause cards to be read and printed with no escape mechanism until the supply of punched data cards was exhausted in which case an error condition would cause the program to fail. Clearly such a statement alone would be of limited use.

There is an extension of this statement known as the 'computed GØ TØ' which gives a little more choice in the statement to which the branch is to be made.

```
GØ TØ (71,56,1,9),I
```

If I=1 control passes to statement 71
If I=2 " " " " 56
If I=3 " " " " 1
If I=4 " " " " 9

For any other value of I control would pass to the next sequential statement in the program.

The most useful form of the transfer of control statement is the

'logical IF' as is demonstrated in our first sample program.

```
IF(PRINC.GT.O.)GØ TØ 1
```

If PRINC is greater than zero then control will pass to the statement labelled 1. The logical IF statement can be considered to be of the form

IF (logical expression) executable statement

The logical expression can take one of two values only, .TRUE. or .FALSE. In a logical IF, the statement appended will be executed only if the logical expression returns a .TRUE. result. When it is .FALSE. the appended statement is ignored and control will pass to the next statement.

```
      IF(A.LT.B)GØ TØ 56
      WRITE(3,11)B
   56 A=B*C
```

If A < B then A will be recalculated as the product of B and C. For A ≥ B the value of B will be printed first.

While the appended statement is frequently a 'GØ TØ' statement, it may be any other executable statement other than another 'logical IF' or a 'DØ' statement (Section 4.10).

```
IF(A.LT.O.)A=-A
```

This would be sufficient to replace A with its absolute value although the coding would be somewhat slower then using the alternative statement

```
A=ABS(A)
```

Logical expressions are most frequently formed by two arithmetic expressions and a relational operator.

A.EQ.B	a is equal to	b	a = b
A.NE.B	a is not equal to	b	a ≠ b
A.GT.B	a is greater than	b	a > b
A.GE.B	a is greater than or equal to	b	a ≥ b
A.LT.B	a is less than	b	a < b
A.LE.B	a is less than or equal to	b	a ≤ b

e.g. IF(A+B.LE.C+SQRT(X**2*Y**2))A=1.

Frequently we wish to carry out more than one logical test at a time. This can be done by combining logical expressions with one of the following logical operators:

.AND.	both expressions must be .TRUE. to return a .TRUE. result
.ØR.	result a .TRUE. if either expression is .TRUE.

```
      IF(A+B.GE.C.AND.A+C.GE.B.AND.B+C.GE.A)WRITE(3,157)A,B,C
  157 FØRMAT(' A,B AND C ARE POSSIBLE SIDES AT A TRIANGLE',3F10.3)
```

The above example will be capable of testing whether 3 values A,B and C are capable of being the lengths of the sides of a triangle. As before if the combined logical expression is ·FALSE· then control will pass to the next statement.

4.10 LOOPS

We frequently find it necessary to repeat a section of code a given number of times. Suppose our problem is to find the sum of the first 20 integers i.e. 1+2+...+20. Ignoring any appeal to mathematical analysis then the following code would be sufficient

```
      :
      ISUM=0
      I=1
    5 ISUM=ISUM+I
      I=I+1
      IF(I·LE·20)GØ TØ 5
      :
```

In this example ISUM is chosen as a variable name to accumulate the sum of the integers (integer variables start with I,J,K,L,M or N). It is first necessary to initialise this to zero and the counter (I) to 1. Two statements are then necessary to increase the counter and to test it to determine whether the loop is complete, and transfer control back if it is not. Because scientific programming is often repetitive in this way FORTRAN supplies a 'DO' statement to allow operations such as the above to be quickly coded as

```
      ISUM=0
      DØ  5 I=1,20
    5 ISUM=ISUM+I
```

The 'DO' statement specifies the last statement in the series of statements to be repeated (5), an INTEGER variable to act as the counter (I), and two INTEGER constants or variables to act as the initial and final values for which the loop is executed.

```
      DØ  2 J=N,M
```

will cause all statements between itself and that with label 2 inclusive to be repeated (M-N+1) times. It is necessary for N and M to have previously been assigned values, either as the left hand side of an arithmetic assignment, or through a READ command. FORTRAN requires that $N \geqslant 1$, while $M \geqslant N$. When a 'DO' loop is completed the 'DO' variable (J in the above example) is regarded as being undefined.

It is at times necessary to nest one 'DO' inside another 'DO' loop. Suppose we have 100 punched data cards with one number on each card, and that our aim is to find the average of each group of 10 and print that average out. The following is a complete program capable of doing this.

```
C     PROGRAM BY J. SMITH
C     TO READ 100 NUMBERS AND
C     FIND AND PRINT THE AVERAGE OF EACH GROUP OF 10
      DØ  1 I=1,10
      SUM=0.
      DØ  2 J=1,10
      READ(1,15)X
    2 SUM=SUM+X
      AVG=SUM/10.
    1 WRITE(3,16)AVG
   15 FØRMAT(F10.3)
   16 FØRMAT(' AVERAGE FØR GROUP ØF 10 = ',F10.3)
      STØP
      END
```

```
      5.32
      8.61    data cards.
       :
       :
```

The loops function so that the inner counter will vary the most rapidly i.e.

```
I 1 1 1 ...  1 2 2 2 ...  2 ... 10 10
J 1 2 3 ... 10 1 2 3 ... 10 ...  9 10
```

The last statement in a 'DO' loop can be any executable statement other than a transfer of control. A dummy statement CONTINUE is provided which does not perform any machine function as a way around this restriction.

```
      DØ 27 I=1,N
       :
       :
      IF(X.GT.27.35)GØ TØ 95
   27 CØNTINUE
       :
       :
   95 X=X+7.
       :
       :
```

4.11 STOP AND END STATEMENTS

The STOP and END statements serve two different purposes.

(i) The END statement provides an indication to the compiler that all

the FORTRAN statements that precede it form a complete and separate program or subprogram in their own right.

(ii) The STOP statement is translated by the FORTRAN compiler as part of the machine program to be executed. When the program is executed and the STOP statement encountered execution of it will cease and the computer will switch to the next job waiting.

4.12 ARRAYS OF VARIABLES

Many mathematical operations require the use of vectors and matrices. FORTRAN supplies a means of handling 1,2,3 or higher dimensional arrays. For the simplest array,(the 1 dimensional vector)the i^{th} element of the vector v (v_i) is represented in FORTRAN as V(I). Elements of an array or vector are capable of being used in FORTRAN in the same way ordinary variables are employed.

```
V(I)=0.        the i^th element of V is set to zero
A=V(I)+C(J)-D(3)
V(I-1)=V(3*I-7)
```

The subscripts used to refer to vector or array elements must be INTEGER and greater than zero. They may be constants, variables or expressions. The FORTRAN compiler reserves one location (word) for non-subscripted variables to be stored in. As subscripted variables take one location for each array element, it is necessary for the programmer to specify to the compiler the maximum number of elements associated with each array. This is done through a non-executable statement, the 'DIMENSIØN' statement that must precede the first use of the array it is defining.

```
      DIMENSIØN V(15)
      DØ  1 I=1,8
    1 V(2*I-1)=0.
```

The 'DIMENSIØN' statement would tell the compiler that V is a vector (1 dimensional) array requiring 15 storage locations. The supplied statements would set all the odd components of V to zero. The next example demonstrates how a vector may be used to calculate the mean and standard deviation of a set of 10 numbers. These numbers are read from 10 cards (i.e. 1 number per card).

$$\bar{X} = \frac{\sum_{i=1}^{10} X_i}{10}$$

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{i=1}^{10} (X_i - \bar{X})^2}{9}}$$

```
C     J. SMITH CALCULATE MEAN AND STANDARD DEVIATION
C     OF 10 NUMBERS
      DIMENSIØN X(10)
      DØ  1 I=1,10
    1 READ(1,53)X(I)
   53 FØRMAT(F10.3)
      SUM=0.
      DØ  2 I=1,10
    2 SUM=SUM+X(I)
      AVG=SUM/10.
      SUMSQ=0.
      DØ  3 I=1,10
    3 SUMSQ=SUMSQ+(X(I)-AVG)**2
      SDEV=SQRT(SUMSQ/9.)
      WRITE(3,541)AVG,SDEV
  541 FØRMAT(' MEAN AND STANDARD DEVIATIØN ',2F10.3)
      STØP
      END
```

Here we use the vector X to store 10 numbers prior to finding the mean and standard deviation. Before employing vectors in a program make sure they are really necessary. In a previous example (Section 9) the mean of a set of numbers was required. There was no need in that case to retain the 10 numbers as the accumalated sum of each number as it was read was sufficient. When the standard deviation is sought, the numbers must be retained at least up to the point at which the mean is determined. When arrays of higher order than the one dimensional vector treated so far are needed, the 'DIMENSIØN' statement informs the compiler of the number of dimensions (i.e. the number of subscripts) and the total storage for the array.

```
     DIMENSIØN A(5,5)
```

This informs the compiler that A is a matrix (2 dimensional array) requiring 25 locations for storage

```
      DIMENSIØN A(5,5),B(5,5),C(5,5)
      :
      :
      DØ  1 I=1,5
      DØ  1 J=1,5
1     C(I,J)=A(I,J)+B(I,J)
      :
      :
```

In this case two matrices A and B are summed and the result stored in a new matrix C.

4.13 SUBPROGRAMS

We have met (Section 4.8) the special mathematical functions supplied through the FORTRAN compiler. The user is able to supply two types of subprograms of his own when necessary:

(i) FUNCTION subprogram.

(ii) SUBROUTINE subprogram.

Need for subprograms arise when:

(i) The same mathematical function or procedure is required at many points in a program.

(ii) In larger programs where it pays to write and test sections of the code independently.

(iii) More than one person is responsible for developing the code.

The FUNCTION subprogram returns a single value as its result and is usually used to perform mathematical operations similar to $\sqrt{\ }$, or function evaluation. The user supplied function is best demonstrated by an example. Suppose we wish to evaluate a cubic polynomial for various values of x:

$$f(x) = 1 + 1.5x + 3.2x^2 + 6x^3,$$

which for speed of computation is best written as

$$f(x) = 1 + x\ (1.5 + x\ (3.2 + 6x)).$$

Then we might use the coding...

```
      :
Y = F(X)+6.
      :
Z = F(X-1.)
      :
STØP
END
```

} Main or calling program

```
FUNCTIØN F(A)
F = 1.+A*(1.5+A*(3.2+6.*A))
RETURN
END
```

} FUNCTION subprogram

In the main program the function is invoked by naming the function and enclosing in parentheses a constant, variable, or expression for which the cubic polynomial is to be evaluated. The FUNCTION subprogram is defined through the use of the 'FUNCTION' statement and the name 'F' by which the function is to be known. An argument list corresponding to that in the main program is also required. The argument names in the function are only dummy and do not have to be the same as those in the main program (all the other

variables and labels are local to the function and are in no way associated with labels or variables in the main program). When a function is invoked by the main program the values X and X-1. are transferred into the location set aside for A. The function must return one value through the assignment of an arithmetic expression to the function name as in

```
F = 1.+A*(1.5+A*(3.2+6.*A))
```

The 'RETURN' is a transfer of control from the function back to the main program from where control was originally passed. The END statement is once again a signal to the compiler that this is the end of a logically independent set of FORTRAN statements.

The SUBROUTINE subprogram is the more powerful version of a subprogram and usually performs more involved operations than those for which the FUNCTION is designed. Typical tasks for which subroutines are used would include finding the roots of equations, multiplication or inversion of matrices, solving sets of linear equations. Unlike the function the subroutine is not restricted to returning one result as part of an arithmetic expression. The SUBROUTINE and the main program communicate through the argument list only. The following code shows the use of a subroutine QUAD to determine real roots of a quadratic equation $ax^2 + bx + c = 0$. The coefficients of the equation to be solved are supplied as arguments to the subroutine, while the subroutine is responsible for returning the two roots and an indication as to whether real roots were possible.

```
  1 READ(1,5)C1,C2,C3
    CALL QUAD(C1,C2,C3,X1,X2,IER)
    IF(IER.EQ.0)WRITE(3,57)X1,X2
    IF(IER.NE.0)WRITE(3,59)
  5 FØRMAT(3F10.3)
 57 FØRMAT(' RØØTS ØF QUADRATIC ARE ',2F10.3)
 59 FØRMAT(' NØ REAL RØØTS EXIST ')
    GØ TØ 1
    END
    SUBRØUTINE QUAD(A,B,C,R1,R2,K)
    DISC=B*B-4.*A*C
    IF(DISC.LT.0) GØ TØ 5
    DISC=SQRT(DISC)
    R1=(-B+DISC)/(2.*A)
    R2=(-B-DISC)/(2.*A)
    K=0
```

```
      RETURN
    5 K=1
      RETURN
      END
```

The subroutine is invoked through a 'CALL' statement by naming the subroutine and supplying a list of variables through which values are to be transferred to and from the subroutine. The main program passes the three coefficients of the quadratic while the subroutine will return the roots in X1 and X2 and an indication (K=1 or 0) to the sign of the discriminant. Once again the code within the subroutine is independent of the calling program.

4.14 ERRORS IN PROGRAMMING

The FORTRAN compiler will inform us in no uncertain terms of any syntactical errors we make in coding a program. Such errors are trivial to detect and correct. The computer is a totally obedient servant, provided we ask it to perform a task in the language it understands, it will obey us without question. Therefore the hardest errors to identify are the ones we make in specifying the logic or steps involved in solving our problem. All programs should be considered guilty of containing bugs until proven innocent ('debugged').

Too often the poor computer is blamed for an error in the program that should have been found and removed by the programmer when he was 'debugging' his code.

garbage in implies garbage out

This adage is certainly true but the programmer and in particular the scientific programmer may find it difficult to recognise the output of a program for what it is. It is advisable to test programs thoroughly before placing any confidence on their output. This is often done by comparing the computed solution with a known mathematical or physical solution. When agreement is satisfactory we may then proceed to use our program for all the cases we are interested in.

Unlike the more commercialy oriented programmer the problems faced by a mathematical programmer are three-fold. As most commercial tasks are well defined, errors in the computer output are directly due to the program or incorrect data on which it operated. The scientific problem solver is solving a mathematical model of some real physical system. When this model was developed many assumptions (and probably simplifications) were made. Just how valid were these and are they the source of errors? Were the errors caused by the type of numerical technique chosen to solve the model? Or were the errors due to the coding of these techniques?

4.15 PRACTICE EXAMPLES

Before you attempt to code any of the reactor problems try these practice examples. The answers to the questions are given in Section 4.6, but don't be too hasty to seek out these answers until you have had a go yourself.

Q1. (a) In the list below, which items are variables or constants?

(b) What is the mode (integer or real) of each variable or constant in the list?

(c) Are any invalid?

List (1) 1., (2) ABC, (3) I4, (4) 14, (5) -0.0001E-10, (6) INKSTAIN, (7) FIVE, (8) 6IX, (9) e, (10) 0, (11) BØS, (12) A*B, (13) 5,312.6, (14) +5.E-03

Q2. Write each of the following algebraic formulae as a FORTRAN statement to calculate y. Use any convenient real names for the variables, which will be assumed to have been assigned values by previous steps of the program.

(1) $y = \frac{1}{2}(b+c)$

(2) $y = (a+b)^2/3$

(3) $y = \left(\frac{1}{a} + \frac{1}{b} + \frac{1}{c}\right)^{-1}$

(4) $y = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!}$ $\quad (n! = 1x2x...x(n-1)xn)$

(5) $y-x = a-\pi y$ $\quad (\pi=3.141592)$

(6) $\sqrt{y} = u$

(7) $x = \frac{1}{y} + \frac{1}{b} + \frac{1}{c}$

What values would be stored in the variable on the left of the following arithmetic statements given that A=3?

(8) I=A

(9) I=A/2

(10) U=A/2

Q3. Write the necessary statements of portion of a program to calculate the variables given by the following expressions. Use any convenient names for the variables. You may assume that variables on the right have been assigned values by previous steps of the program and that the values do not require special consideration in calculating the expressions - for example $a \neq 0$ in (1).

(1) $x = \frac{1}{2a}\left(-b + \sqrt{b^2 - 4ac}\right)$

(2) $s = \sqrt{x^2 + y^2 + z^2}$

(3) $u = \tanh x = \dfrac{\frac{1}{2}(e^{x} - e^{-x})}{\frac{1}{2}(e^{x} + e^{-x})}$

(4) $v = \tan x$

(5) $h = 1 - \dfrac{x^2}{2!} + \dfrac{x^4}{4!}$

(6) $y = 1 - e^{-x} - \dfrac{1}{5}e^{-2x} - \dfrac{1}{25}e^{-3x}$

(7) $c = \ln \left|\dfrac{1}{1+a^3}\right|$

(8) $g = \left(\dfrac{\pi}{xy}\right)^{1/2} \sin\left(\dfrac{xy}{\pi}\right)$

(9) $y = \left(e^{ax} + e^{-\sqrt{ax}}\right)/3$

(10) $z = \dfrac{-\tan^{-1}(x/a)}{1 + u/a}$

Q4. Write a program that will

(1) Read the four coefficients of a cubic polynomial $f(x)=a+bx+cx^2+dx^3$ from a punched card in FORMAT (4F10.3)

(2) Read the estimate X_0 of a root of the equation $f(x)=0$ from a second card (FORMAT(F10.3)).

(3) Improve the estimate of the root by the Newton-Raphson method

i.e. $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$

(4) The process can be considered to have converged if

$\dfrac{|x_{n+1} - x_n|}{|x_n|} < 0.001$

(5) Print out the improved estimate of the root.

(6) Allow only 5 iterations. If convergence has'nt been achieved print a message warning of this.

(7) Repeat from (1).

Q5. (For advanced students only)

Read in a set of 10 numbers punched one per card (FORMAT(F10.3)). Write code that will sort these numbers in descending order.

4.16 ANSWERS AND TYPICAL CODING

Q1. (1) real constant, (2) real variable, (3) integer variable, (4) integer constant, (5) real constant, (6) invalid variable name as more than 6

characters, (7) real variable, (8) invalid variable name as first character is not alphabetic, (9) invalid variable name as e is a lower case letter, (10) integer constant, (11) real variable, (12) invalid since an expression is not a variable, (13) invalid as comma is not permitted, (14) real constant.

Q2. (1) `Y = 0.5*(B+C)`

(2) `Y = 0.3333333*(A+B)*(A+B)`

(3) `Y = 1./(1./A+1./B+1./C)`

(4) `Y = 1.+X*(1.+X*(0.5+0.1666667*X))`

(5) `Y = (X+A)/4.141592`

(6) `Y = U*U`

(7) `Y = 1./(X-1./B-1./C)`

(8) `I = 3`

(9) `I = 1`

(10) `U = 1.5`

Q3. (1)
```
X = (-B+SQRT(B*B-4.*A*C))/(2.*A)
```

(2)
```
S = SQRT(X*X+Y*Y+Z*Z)
```

(3)
```
W1 = EXP(X)
W2 = 1./W1
U = (W1-W2)/(W1+W2)
```

(4)
```
V = TAN(X)
```

(5)
```
W1 = X*X
H = 1.-W1*(0.5-0.04166667*W1)
```

(6)
```
W1 = EXP(-X)
Y = 1.-W1*(1.+W1*(0.2+0.04*W1))
```

(7)
```
C = -ALOG(ABS(1.+A*A*A))
```

(8)
```
W1 = X*Y/3.141592
G = SQRT(1./W1)*SIN(W1)
```

(9)
```
W1 = A*X
Y = (EXP(W1)+EXP(-SQRT(W1)))*0.3333333
```

(10)
```
Z = -ATAN(X/A)/(1.+U/A)
```

Q4.
```
C     J. SMITH
C     RØØT ØF CUBIC EQUATION
    9 READ(1,56)A,B,C,D
   56 FØRMAT(4F10.3)
      READ(1,57)XN
   57 FØRMAT(F10.3)
```

```
      DØ  1 I=1,5
      XNP1=XN-(A+XN*(B+XN*(C+D*XN)))/(B+XN(2.*C+XN*3.*D))
      IF(ABS((XNP1-XN)/XN).LT..001)GØ TØ 2
    1 XN=XNP1
      WRITE(3,58)
   58 FØRMAT(' RØØT NØT FØUND WITHIN 5 ITERATIØNS ')
      GØ TØ 9
    2 WRITE(3,59)XNP1
   59 FØRMAT (' RØØT ØF CUBIC = ',F10.3)
      GØ TØ 9
      END
```

Sample data cards:

```
column 10 |       20 |        30|        40 |
       5.6       3.7        2.       -46.
       1.2
```

Q5.

```
C     J. SMITH
C     SØRTING PRØBLEM
      DIMENSIØN X(10)
      DØ  1 I=1,10
    1 READ(1,9)X(I)
    9 FØRMAT(F10.3)
      DØ  2 I=1,9
      N=I+1
      DØ  2 J=N,10
      IF(X(J).LE.X(I))GØ TØ 2
      TEMP=X(J)
      X(J)=X(I)
      X(I)=TEMP
    2 CONTINUE
      .
      .
      .
```

CHAPTER 5

INTERACTIVE COMPUTING

Lecture by

P.L. SANGER

CONTENTS

CONTENTS (Cont'd.)

5.1 INTRODUCTION

A computer user who has access to a powerful computer, and has a convenient method of getting information into it directly, and who can immediately see the results of a complex calculation in a simple, understandable form has a great advantage over another who submits a program to be run on the computer and gets back tables of numbers a few hours later. If the user can experiment with the input and see the effect of each change then he may quickly begin to understand a complicated problem.

This is the principle of man-machine interaction or on-line problem solving. The point is that both man and computer can each do some things much better than the other. A closed loop system allows the special abilities of each to be used to advantage and the combination may be far more powerful than the sum of the two components (it has been described as the 'two plus-two equals five effect').

By comparison, the man who has to wait for his results loses 'thinking momentum' and has to collect his thoughts on the problem after every computer run.

The computer network being set up at the Research Establishment will allow powerful interactive computing facilities to be provided at various locations on site. In the meantime, the ACL-NOVA system allows users to solve small to medium-sized problems in an interactive manner.

To see some of the advantages of interactive computing, the 1 delayed group approximation to the power response of the MOATA reactor (Chapter 2) will be solved using the ACL-NOVA system. Assuming that sufficient FORTRAN has been mastered to solve the MOATA power response equations, the conversational programming language, ACL, will now be presented as an alternative, more flexible method of solution. For more details of the ACL language and the ACL-NOVA system, refer to the publications (Sanger 1971, Bennett and Sanger 1973).

5.2 THE ACL-NOVA SYSTEM

5.2.1 General Discussion

The ACL language (ACL stands for A Conversational Language) was developed at the AAEC. It was implemented on a NOVA computer supporting five teletypewriter terminals and this is referred to as the ACL-NOVA system.

The ACL language consists of immediate statements and stored statements. Stored statements are translated and saved in a user work area from which they may be recalled and executed under user control. Each stored statement has a sequence number in the range 100 to 999 and may also have a statement

number in the range 0 to 99. Stored statements are used to build up a stored program. They are ordered according to their sequence number and may be inserted, modified, or deleted freely by the user. Stored statements may be typed in any order, and any newly entered statement will replace a previously saved statement with the same sequence number.

An immediate statement, which does not have a sequence number, is translated and executed when typed and is then discarded. Immediate statements are used to perform one-time or 'desk calculator' calculations, to control the execution of a stored program and to perform various editing and debugging functions. The basic statement forms used for immediate and stored statements are shown in the ACL-NOVA summary in Appendix 5.1.

5.2.2 Input from a Terminal

The ACL-NOVA system currently supports five teletypewriter terminals. Input to the system is specified by pressing keys on the typewriter-like keyboard. To begin work at a terminal, you press the CTRL key and the G key at the same time, and the system responds by typing ACL-NOVA and spacing a few lines. Once the terminal has been initialised in this way, statements may be stored or immediate statements executed.

Input begins from column 1, which is taken to be the leftmost position of the teletype carriage that results from pressing the Carriage Return (CR) key, and may consist of up to 72 characters. If input is continued past column 72, the whole line is cancelled and must be entered again. In the ACL-NOVA system, then, you can think of input on a 72 column layout. For statements to be stored for later execution, this layout takes the form

```
1  45 78                    ACL                                    72
230    A←4
235 1  B←6
240 23 A+B
```

Immediate statements begin from column 1 and have the layout:

```
1                           ACL                                    72
26+4
A2←23.5
SQR(42.978)
```

The terminals attached to the NOVA computer are operated in what is termed full-duplex mode. This means that a character pressed at the teletype keyboard is sent to the computer, but is not printed at the teletype

unless it is sent back to the terminal (echoed) by the NOVA computer. By making use of this feature, only characters that are valid in syntax are 'echoed' at a terminal; that is, only characters that make sense are typed at the terminal. For example, if you pressed the keys 3++3 at the keyboard the system would only type 3+3 at your terminal. A statement is executed when it is completed by a 'valid' carriage return.

This 'echo-checking' mechanism is a very valuable feature of the ACL-NOVA system and provides interaction even while statements are being entered at the terminal.

5.2.3 Arithmetic

All arithmetic operations are carried out using single precision floating point numbers. For input, the numbers may have free format; that is, they may be integers, may contain a decimal point or may contain an exponent.

Examples are: 327, 1.231, 0.0014, -1.45E+12, 3E2.

5.2.4 Variables

Three types of variables may be used; a simple variable, a singly subscripted variable or a doubly subscripted variable.

A simple variable name must begin with an alphabetic character and may be followed by up to three alphanumeric characters. Singly subscripted variables consist of an alphabetic character, which may be followed by an alphanumeric character, followed by an arithmetic statement or expression (see Section 5.2.5) enclosed in brackets. The arithmetic statement or expression is evaluated and truncated to form an integer in the range 0 to 65,535, the appropriate subscript. Doubly subscripted variables consist of an alphabetic character, which may be followed by an alphanumeric character, followed by two arithmetic statements or expressions which are separated by a comma and are enclosed in brackets. Each of these arithmetic statements or expressions is evaluated and truncated to form an integer in the range 0 to 255, the appropriate subscript.

Examples are:

A,A1B2,ZA,RETN,.... simple variables

AC(1),D1(I+X-3/2E1),X(I←3+N←2),.... singly subscripted variables

B3(7,2),Y(I←I+1,J←J+INT(BX(A←3)←4+Z←3)),.... doubly subscripted variables.

In ACL all variables are treated as real variables, so that there is no distinction between real and integer variables as there is in FORTRAN. In addition, there is no conflict between simple and subscripted variable names in ACL. For example, in ACL one can use A, A(1), A(2,3) as separate variables, while this is not allowed in FORTRAN. The subscript zero is also allowed in ACL.

5.2.5 Arithmetic Statements and Expressions

Arithmetic statements and expressions play an important role in the ACL language and their definitions and mode of evaluation should be understood.

The basic operands in an arithmetic statement or expression are variables and numbers. The relevant operators are +, -, *, /(divide), ↑(power), ←(assignment) and the built-in mathematical functions such as COS, EXP, LOG, SIN and SQR.

Examples:

addition	A+B
subtraction	A-B
multiplication	A*B
division	A/B
exponentiation	A↑B (in FORTRAN this is A**B)
use of functions	EXP(A+B↑3) equivalent to e^{a+b^3}

The mathematical operators have the usual hierarchy with the order of execution as ↑ first, then * or / at the same level and finally + or - at the same level. Brackets may be used to override the mathematical hierarchy. For example:

3+4*6↑2	would give the result 36*4+3=147
3+(4*6)↑2	would give the result 24*24+3=579
((3+4)*6)↑2	would give the result 42*42=1764

Brackets are also necessary to prevent two arithmetic operators from appearing next to each other, for example,

A*-B must be written as A*(-B) .

If a variable is followed by an assignment arrow (←), then during statement execution the expression to the right of the assignment arrow is evaluated and its value given to that variable. The variable name and its value in floating point form are stored in a symbol table in the user work area. Multiple assignments may occur in the one arithmetic statement or expression, and this is an important difference between ACL and FORTRAN. When multiple assignments occur in an expression, assuming first of all that the expression is bracket free, then the rightmost assignment arrow is located, the expression to the right of this is evaluated and the resulting value given to the variable. This process is continued until all the assignments have been carried out.

Examples:

A←3	A is assigned the value 3.

C←6+B←X1←2 X1 is first assigned the value 2, B is next assigned the value 2 and finally C is assigned the value 8.

In more complicated expressions that contain a number of levels of brackets plus multiple assignments, the expression is evaluated by searching first for the rightmost pair of opening and closing brackets. The expression enclosed between these brackets (an expression at zero bracket level) is then evaluated by searching for the rightmost assignment arrow and proceeding as described in the last paragraph. The bracket processing and assignment arrow processing is continued until the whole expression is reduced to zero bracket level and evaluated.

Examples:

Y←(A←2*B)↑(B←SQR(Z←9))+X←4 Z is first assigned the value 9, B is then assigned the value 3, A is assigned the value 6, X is assigned the value 4 and Y is finally assigned the value $6^3+4=220$.

To do this in FORTRAN, the following five statements are required.

```
Z=9
B=SQR(Z)
A=2.*B
X=4.
Y=A**B+X
```

Another important difference between ACL and FORTRAN is that no DIMENSION statement is required in ACL to indicate the size of singly or doubly subscripted arrays. Subscripted variables are added to the symbol table as they are defined, and a user can thus use the variables A(5), A(7), A(9) without needing to say DIMENSION A(9) as required in FORTRAN.

An expression is termed an arithmetic statement if the first term in the expression is a variable followed by an assignment arrow; otherwise it is an arithmetic expression.

The result of executing an arithmetic expression is printed at a terminal in one of two possible formats depending on its magnitude. The number is given in exponent form if it is in the range $|\text{number}| \leqslant 10^{-4}$ or $|\text{number}| \geqslant 10^{3}$. If the number lies in the range $10^{-4} < |\text{number}| < 10^{3}$ it is printed in non-exponent form with seven significant figures, if necessary. Integers are printed without a decimal point.

Examples:

(i) 3↑2+AB1←2.453 is an arithmetic expression and during execution AB1 would be assigned the value 2.453 and the result 11.453 would

be printed at the terminal.

(ii) AX←SQR(C←4*B(1)←1)+3-B←-2 is an arithmetic statement and during execution B(1) would first be assigned the value 1, C would then be assigned the value 4, B would be assigned the value -2 and AX would finally be assigned the value 7, but nothing would be printed at the terminal.

(iii) (Z2←8.332055E-3) is an arithmetic expression and during execution Z2 would be assigned the value 8.332055E-3 and the result 0.008332056 would be printed at the terminal.

(iv) C6 is an arithmetic expression and during execution the value of the variable C6 would be printed at the terminal.

5.2.6 Sequence Numbers and Statement Numbers

Each stored statement must have a sequence number in the range 100 to 999. The statements are ordered according to their sequence number and consequently statements in a stored program may be typed in any order and inserted or deleted quite freely.

A one or two digit statement number in the range 0 to 99 may also be associated with a stored statement. Thus a stored statement can be referred to either by a three digit sequence number or a one or two digit statement number. It is possibly better to use statement numbers for branching within a stored program, since the branch statements will not have to be altered if the sequence numbers of the stored statements are changed.

5.3 STORED STATEMENTS

Statements that have a sequence number associated with them are classed as stored statements, and these are used to build up a stored program that may later be executed. Arithmetic statements and arithmetic expressions may form part of a stored program, and they may also be used as operands in the statements that are described below.

5.3.1 ACCEPT Statement

Data may be read by a program by using the ACCEPT statement which takes the form:

ACCEPT ↓ { variable [,variable [,variable ....]] }

The curly brackets are used to indicate a field that must be present, while the square brackets are used to indicate an optional field. In the case of the ACCEPT statement, then, at least one variable name must be specified to complete the statement.

When an ACCEPT statement is executed, its sequence number is typed on a new line followed by the variable name and assignment arrow. Execution of the

stored program is temporarily suspended until the value of the variable is specified. This procedure is repeated until all of the variables listed in the ACCEPT statement have been read, and program execution then proceeds normally.

For example, execution of the stored statement

```
123 5  8                    ACL                                          72
317 10 ACCEPT B51,A(1,1),X
```

causes the line

```
1  5
317 B51←
```

to be typed at the terminal. When the value of B51 has been given (completed by a valid CR), the line

```
1  5
317 A(1,1)←
```

is then printed. After A(1,1) is given a value, the line

```
1  5
317 X←
```

is printed. On reading X, the ACCEPT statement is fully processed and program execution continues normally with the next stored statement.

5.3.2 GO TO Statements

Stored statements are usually processed sequentially in the order of increasing sequence numbers. However, it is possible to transfer out of the current sequence of statement processing by executing a GO TO statement which takes the form:

GO TO {arith stmt or exprn}

When this statement is executed, control is passed to the stored statement with the sequence number (if 3 digits) or statement number (if 1 or 2 digits) obtained by evaluating the arithmetic statement or expression.

Examples:

```
1  5  8
325   GO TO 211
```

causes control to be passed to the stored statement with the sequence number 211.

```
1  5  8
411   GO TO I←2*2E2-309
```

cause the variable I to be assigned the value 91, and control then to be passed to the stored statement with the statement number 91.

5.3.3 IF Statement

The most important statement in controlling the logical flow of a program is the IF statement which takes the form:

IF({arith stmt or exprn} [.EQ. .NE. .GT. .GE. .LT. .LE.] {arith stmt or exprn})ø [arith stmt or exprn / TYPE stmt / ACCEPT stmt / GO TO stmt / CALL stmt / RETURN stmt / CONTINUE stmt / STOP stmt]

If the logical relation between the two arithmetic statements or expressions enclosed in brackets is obeyed when the statement is executed, then the third statement outside the brackets is executed; otherwise control is passed to the next stored statement.

Example:

```
1    5    8
273     IF(I←1+I.LE.10) F1←0
```

causes the value of the variable I to be increased by one, and if the new value of I is less than or equal to 10 then the variable F1 is set to zero before executing the next stored statement.

There is no DO statement in the ACL language and the IF statement must be used for program looping, that is, if the same set of statements are to be executed a number of times.

5.3.4 TYPE Statement

The results of the calculations carried out by a stored program may be printed at the terminal by using the TYPE statement. The value of an expression may be typed out in a format determined by its magnitude, as discussed in Section 5.2.5, or entirely in exponent format.

For example:

```
1    5    8
512     TYPE A3
```

causes the value of A3 to be typed in a form determined by its magnitude, while the statement

```
1    5    8
512     TYPE "A3
```

causes the value of A3 to be typed in exponent form, regardless of its magnitude.

When variables are separated by commas, they are printed on the same line with a space between each number. For example, if A1 has the value 271 and A(2,1) has the value 6731 in all of the following examples, then

```
1    5    8
473     TYPE A1,A(2,1)
```

causes the line,

```
1    5
271 6.731000E+03
```

to be typed at the terminal.

Literal data (character output or headings) can be printed by enclosing it within apostrophes. For example, the statement

```
1    5    8
671     TYPE 'ANSWER=', A1
```

causes the line

```
1
ANSWER=271
```

to be printed.

To continue output on a new line, a semi-colon should be used as the delimiter, so that

```
1    5    8
423     TYPE A1;A(2,1)
```

causes the lines

```
1
271
6.731000E+03
```

to be typed at the terminal. The semi-colon delimiter can also be used to space a number of lines, for example

```
1    5    8
483     TYPE ;;;
```

causes the teletypewriter to be spaced three lines.

Output may be placed at a particular carriage position by indicating the required position by means of an arithmetic statement or expression enclosed between the < and > symbols. This positional parameter must appear before the required variable or literal data and be separated from it by a comma. For example, the statement

```
1   5   8
821   TYPE <27>, 'A1=',"A1
```

causes the line

```
1                                  27|28
                                   A1=2.710000E+02
```

to be printed.

5.3.5 STOP Statement

Execution of a STOP statement completes the execution of a stored program. The statement takes the form

```
1   5   8
418   STOP
```

and when it is executed the message

```
1   5
418 STOP
```

is printed at the terminal.

5.3.6 END Statement

The END statement is used to indicate that work has finished at a terminal. It should be the last statement executed before leaving the terminal.

5.4 MOATA POWER RESPONSE

You have now been introduced to enough of the ACL language to be able to set up a stored program to calculate the MOATA power response using the 1 delayed group approximation (chosen to simplify the presentation of the ACL language). The 1 delayed group approximation corresponds to setting C=1 in the delay differential equation (Equation (1), Chapter 2) yielding the analytic solution (Equation (9), Chapter 2):

$$p(t) = \frac{P_0}{(1-\rho)} \; e^{\rho(t-t_0)/((1-\rho)\,T_1)} \quad , \text{for } t \geq t_0 \qquad \ldots(1)$$

$$\left.\begin{array}{ll} \text{with} & P(t_0) = \dfrac{p_0}{(1-\rho)} \\ \text{and} & T_1 = 13 \text{ seconds} \end{array}\right\} \quad \ldots(2)$$

Using t_0, p_0 and measurements of p(t) at various time steps, we will use Equation (1) to calculate the value of ρ that gives the 'best fit' to your MOATA experiments. We will begin by taking a certain value of ρ and calculating theoretical values of the power, PT(t), at time steps corresponding to the experimental measurements, PE(t). The sum of the squares, $(PE(t) - PT(t))^2$, will then be calculated and used as our criterion of 'goodness of fit'. The value of ρ will then be increased by $\delta\rho$ and the whole process repeated until we obtain a minimum value of the sum of square of differences (often referred to as $\Sigma\Delta^2$). The corresponding value of ρ is then said to give the least squares 'best fit' to the experimental measurements using the 1 delayed group approximation.

The computational procedure for the above calculations is shown below:

t_0 = variable (depends on experiment)
p_0 = variable (depends on experiment)
t_{end} = (value of t for last measurement)
δt = 10 (every 10th measurement after t_0 will be used)
t = t_0
$p_e(t)$ = variable (depends on experiment) ←
t = $t+\delta t$
if (t) $\leq t_{end}$ read next measurement ←
ρ = variable (0.14 for withdrawal experiment)
$\delta\rho$ = variable (0.01 for withdrawal experiment)
psum = 10^{70} (initially set to a very large value)
sum = 0 ←
t = t_0
a = $1-\rho$
b = $\rho/(13*a)$
a = $p_0*\exp(-b*t_0)/a$
c = $p_e(t)-a*\exp(b*t)$ ←
sum = c*c
t = $t+\delta t$
if (t) $\leq t_{end}$ continue $\Sigma\Delta^2$ calculation ←
if (sum) $\geq$ psum then we have passed minimum and print results
ρ = $\rho+\delta\rho$
psum = sum
go to ←

Note:

The relation, $e^{x+y} = e^x e^y$ ...(3)

has been used to convert Equation (1) to the form:

$$p_t(t)=p_0*\exp(-\rho*t_0/(13*(1-\rho)))*\exp(\rho*t/(13*(1-\rho)))/(1-\rho) \quad \ldots(4)$$

An ACL program based on this computational procedure is shown below:

```
1    5   8
11Ø     ACCEPT TØ,PØ,TEND
12Ø     DT←1Ø
13Ø     T←TØ
14Ø 1   ACCEPT PE(T)
15Ø     IF(T←T+DT.LE.TEND) GO TO 1
16Ø 2   ACCEPT RHO,DRHO
17Ø     TYPE ;'MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)'
18Ø     TYPE <19>,'TØ=',TØ,' PØ=',PØ;;
19Ø     PSUM←1E7Ø
20Ø     TYPE <14>,'RHO',<24>,'SUM OF (PE-PT)**2'
21Ø 3   T←TØ+SUM←Ø
22Ø     B←RHO/(13*A←1-RHO)
23Ø     A←PØ*EXP(-B*TØ)/A
24Ø 4   SUM←SUM+C*C←PE(T)-A*EXP(B*T)
25Ø     IF(T←T+DT.LE.TEND) GO TO 4
26Ø     IF(SUM.GE.PSUM) GO TO 5
27Ø     TYPE <12>,RHO,<27>,SUM
28Ø     RHO←RHO+DRHO
29Ø     PSUM←SUM
3ØØ     GO TO 3
31Ø 5   TYPE ;<13>,'T',<23>,'PE(T)',<4Ø>,'PT(T)'
32Ø     T←TØ
33Ø     B←RHO/(13*A←1-RHO←RHO-DRHO)
34Ø     A←PØ*EXP(-B*TØ)/A
35Ø 6   TYPE <12>,T,<2Ø>,PE(T),<37>,A*EXP(B*T)
36Ø     IF(T←T+DT.LE.TEND) GO TO 6
37Ø     TYPE ;;
38Ø     STOP
```

Note the use of multiple assignments in statements 210,220,240 and 330, and the way in which the IF statement is used for loop control in statements 150,250 and 360 by incrementing T by DT before testing its value against TEND.

Results obtained from running the above program for the control rod withdrawal experiment, using an initial value of $\rho = 0.14$ and $\delta\rho = 0.01$, are shown in Table 5.1. For this run a minimum of $\Sigma\Delta^2$ occurred for $\rho = 0.17$.

To obtain a more accurate value of ρ, the program can be run again using initial values of $\rho = 0.17$ and $\delta\rho = 0.001$ and this gave the results shown in Table 5.2. This time the value $\rho = 0.175$ gave the minimum value $\Sigma\Delta^2$. Note that this run was started at statement 160, using an immediate GO TO statement (see Section 5.5.2), to avoid typing in the experimental data a second time.

Looking at the way the program has been set up to determine the best value of ρ, you can see how easy it is to change the initial value of ρ and how quickly one can get a feel for the effect of $\delta\rho$ on the convergence of the calculation. The ability of the user to interact with the problem by immediately being able to try various combinations of ρ and $\delta\rho$ depending on the outcome of previous values tried is a considerable advantage over the FORTRAN user who must anticipate suitable values of ρ and $\delta\rho$ prior to submitting his run to the central computer.

What value of ρ gives the best fit for your control rod withdrawal measurements? - let's run the program and see. What change in the results, if any, would you expect if you changed DT? - let's modify the program and see. Does the 1 delayed group approximation give a good fit to your control rod drop measurements ? - let's run the program using these measurements and check. Any other aspects? - well let's try them too. With interactive computing you can observe the effect of various modifications to the basic program as ideas come to mind!

5.5 IMMEDIATE STATEMENTS

So far, the ACL statements that you have met are very similar to FORTRAN statements and it is now time we looked at the immediate statements that provide most of the interaction with the user.

Statements that do not have a sequence number associated with them are classed as immediate statements and they are executed as soon as a 'valid' CR is type. Arithmetic statements, arithmetic expressions, TYPE statements and the END statement can be executed as immediate statements, in addition to those mentioned specifically below.

5.5.1 RUN Statement

Perhaps the first thing a user wants to know once he has typed in a stored program is how to execute it. This is done by using the RUN statement which begins execution of the stored program starting at the statement with the lowest sequence number. The statement takes the form

1

RUN

The results in Table 5.1 were obtained by using this statement to begin execution of the stored program designed to obtain the best fit between the 1 delayed group approximation and the MOATA control rod experiments.

5.5.2 GO TO Statement

An immediate GO TO statement can be used to begin execution of a stored program with any stored statement and it takes the form

```
1
GO TO {arith stmt or exprn}
```

The results in Table 5.2 were obtained by restarting the MOATA power response program by executing the statement

```
1
GO TO 2
```

5.5.3 LIST Statement

Stored statements are listed (printed out) at a terminal (in sequence number order) when the LIST statement is executed. A single statement, a small group of statements or the entire stored program may be listed.

Examples:

(i)

```
1
LIST
```

causes the whole program to be listed

(ii)

```
1
LIST 67
```

causes the stored statement with the statement number 67 to be listed.

(iii)

```
1
LIST 117,421
```

causes all the stored statements starting from sequence number 117 and finishing with sequence number 421 to be listed. Some of the listing possibilities are shown in Table 5.3 for the case of the MOATA power response program.

The teletypewriter terminals have a paper tape reader and paper tape punch attached to them. Input from a terminal can come from either the teletype or the paper tape reader. When output is being printed at a terminal, it will also be punched onto paper tape if the paper tape punch is ON.

The paper tape punch is normally OFF.

If you have tested your program and wish to keep a copy of it on paper tape, then you should execute the statement.

```
1
LIST:
```

In this case, to give you time to turn the punch ON, the listing is delayed until a CR is entered as the first character on the next line. When the CR is entered at the terminal, five inches of feeder holes are punched at the start of the tape. Five inches of feeder holes are also punched at the end of the listing.

Listing may be terminated at any time by entering ? from the keyboard

5.5.4 SYMBOLS Statement

Execution of the SYMBOLS statement, which takes the form

```
1
SYMBOLS [:]
```

causes the values of all the variables in the symbol table to be printed out in the form

```
1
A1←1.320000E-20
Z13←21
A2(7)←.01
NA(3,7)←.0823
```

If the colon is present, it indicates that the output is to be punched onto paper tape. Entering ? from the keyboard will again cause listing to be terminated.

The above form of the symbol table listing was chosen so that each line was an immediate arithmetic statement and if the output was punched onto paper tape, it could be later read in to re-initialise the symbol table.

5.5.5 Program Tracing

To assist in debugging a program, special tracing facilities are built into the ACL language.

A statement is said to be traced if any symbol table assignments that occur while the statement is being executed are typed at the terminal. These assignments are typed in the form

```
1                    5
{sequence number} {variable}←{value}
```

or for example,

```
1    5
317 A72←6.13
```

Individual statements may be traced by executing an immediate statement of the form

```
1    6
TRON {arith stmt or exprn}
```

before beginning the execution of a stored program. For example, in the case of the MOATA power response program, if the immediate statements

```
1    6
TRON 210
TRON 220
TRON 230
TRON 280
TRON 290
```

were executed, followed by the execution of the immediate GO TO 2 statement to restart the program then the results shown in Table 5.4(a) would be obtained. When you have finished tracing individual statements you should turn the trace off using statements of the form

```
1    7
TROFF {arith stmt or exprn}
```

as shown, for example in Table 5.4(b).

If you are really having trouble with your program then you can get a full trace of your program by executing the immediate statement

```
1    6
TRON
```

before saying GO TO 2.

Every stored statement that is subsequently executed is listed at the terminal and traced as shown for example in Table 5.5(a). This allows a complete trace of a stored program to be obtained. When you have finished your full trace you should execute the immediate statement

```
1    7
TROFF
```

as shown in Table 5.5(b).

5.5.6 Interrupting Program Execution

Execution of a stored program may be interrupted by the character ? being typed at the terminal as the result of an error condition in the stored program or as the result of certain PAUSE conditions. At this point, stored statements may be inserted, modified or deleted, or immediate statements executed. If the first input character on a line is CR, then execution continues from the point where the program was suspended. Execution of the stored program may recommence at a different point by executing an immediate GO TO statement or it may be restarted by executing a RUN statement.

For example, in Table 5.6 you can see the effect of typing ? while the program was calculating results starting with the initial values $\rho = 0.17$ and $\delta\rho = 0.001$. The message

```
1    5
250 ?
```

was printed to indicate that the next statement to be executed was the one with the sequence number 250. At this stage the value of T and RHO were determined. The character CR was next typed as the first character on the line, and the stored program continued from the point at which it was interrupted.

The program was again interrupted by typing the character ?, statement 240 listed and the values of SUM and T determined. Execution of the immediate GO TO 380 statement caused a STOP statement to be executed and execution of the stored program was terminated with the message

```
1    5
380 STOP
```

being printed at the terminal.

5.5.7 Error Correction

Errors that occur while a statement is being typed may be corrected quite simply. For example, to delete the last two characters that were accepted as input type < 2 CR . This would cause the original line of input minus the last two characters to be typed on a new line and to be syntax checked as though they were the original keyboard input. Thus,

```
1                                   ACL
A2←B+SQR(AZ1+C←3)-X3(4,2)-6 <  2
```

would result in the new line

```
1                          ACL
A2←B+SQR(AZ1+C←3)-X3(4,2)
```

being typed out. A one or two digit number may follow the < symbol.

A statement being entered at the keyboard may also be corrected in edit mode (see next Section) by typing << CR after the last input character.

Finally, if the whole input line is to be deleted, type <<< after the last input character.

5.5.8 EDIT Statement

The EDIT statement is used to modify statements that are part of a stored program in what is described as edit mode. The statement takes the form:

EDIT ⌀ {arith stmt or exprn}

and when executed the stored statement corresponding to the sequence or statement number specified by the arithmetic statement or expression is first printed and the carriage returned to the next line at column 1. At this point the statement is ready to be edited, and to follow the 'edit mode' procedure consider a pointer to each character in the original statement, the OS pointer, where initially OS is one.

To copy a character from the original statement, the SPACE (⌀) key is pressed for each required character and the copied character becomes virtual input from the terminal. If this character is syntactically correct, it is echoed at the terminal and the OS pointer increased by one.

To insert a new character, the required character should be typed and it is echoed if it is correct in syntax. If a space character itself is to be inserted, then the special character ESC should be typed. The OS pointer is not altered in this case.

To jump over a character in the original statement the DEL or RUB OUT character should be typed and the OS pointer is simply increased by one, without motion of the carriage.

Once the OS pointer has reached the end of the original statement, further attempts to press SPACE or RUB OUT have no effect. For example, the stored statement

```
1   5  8
211 72 A3←6
```

can be modified by executing the immediate statement

```
1   6
EDIT 211
```

or

```
1   6
EDIT 72
```

If the characters

```
1
ØØØØØØØØ(2)   Rub Out   Ø 2 Ø
```

are typed, after the statement has first been listed, then the resulting stored statement would be

```
1   5   8
211 72 A(2)←26
```

5.5.9 Special Input Features

When you are typing a statement that is to be part of a stored program, you need a three digit sequence number followed by a space, followed by three spaces if you don't want a statement number and then the stored statement beginning from column 8 completed by a 'valid' CR. However, to make this process a little easier, the system is designed so that the pressing of SPACE at the column 1 position will cause a sequence number to be automatically generated for you and this is typed in columns 1, 2, 3 followed by a space in column 4. The automatically generated sequence number is ten greater than the last sequence number that was typed in. The sequence number 110 is given if there was no previous sequence number.

If no statement number is required, then the SPACE key should again be pressed at the column 5 position, and the teletype carriage is automatically positioned to column 8. The required stored statement can be entered at this point.

If you wish to delete a statement from your program then you must type in the relevant sequence number followed by a space and then type CR. For example

```
1
231 Ø  CR
```

causes statement 231 to be deleted.

5.6 CONCLUSIONS

The syntax checking of input and the powerful editing and error correction facilities provided by the ACL-NOVA system allow a user to set up a stored program very simply and easily.

In no time at all, the user can be producing results and with the interrupt and tracing facilities at his disposal he can also quickly debug his program. By having the opportunity to control the input to a program, the user

can gain valuable insight into his calculations.

However, with such ready access to problem solution using interactive computing one must be careful not to be carried away by the computer. There are times when the only way to discover an error is to THINK - and that is solely our prerogative.

5.7 REFERENCES

Bennett, N.W. and Sanger, P.L. (1973) - The Development of the ACL Language and its implementation ACL-NOVA. August issue of the Australian Computer Journal.

Sanger, P.L. (1971) - ACL-NOVA: A Multi-user Conversational Interpreter for the Nova Computer. AAEC/E221 (Revised July, 1972).

TABLE 5.1

RESULTS FOR MOATA POWER RESPONSE PROGRAM

```
RUN

110 T0=24
110 P0=26
110 TEND=400
140 PE(24)=26.5
140 PE(34)=35.8
140 PE(44)=41.4
140 PE(54)=54.5
140 PE(64)=64.1
140 PE(74)=72.7
140 PE(84)=91.7
140 PE(94)=107
140 PE(104)=126.8
140 PE(114)=145.1
140 PE(124)=178.3
140 PE(134)=205.2
140 PE(144)=249.8
140 PE(154)=290.8
140 PE(164)=351.2
140 PE(174)=424.9
140 PE(184)=486.504
140 PE(194)=584.102
140 PE(204)=686.504
140 PE(214)=796.402
140 PE(224)=952.203
140 PE(234)=1121.504
140 PE(244)=1305.504
140 PE(254)=1564.203
140 PE(264)=1835.504
140 PE(274)=2159.603
140 PE(284)=2520.004
140 PE(294)=2948.004
140 PE(304)=3433.102
140 PE(314)=4028.102
140 PE(324)=4684.305
140 PE(334)=5420.805
140 PE(344)=6291.504
140 PE(354)=7164.203
140 PE(364)=8259.750
140 PE(374)=9327.125
140 PE(384)=10608.313
140 PE(394)=11973.750
160 RHO=0.14
160 DRHO=1/100
```

MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)
T0=24 P0=26

RHO	SUM OF (PE-PT)**2
.14	3.304571E+08
.15	2.325454E+08
.16	1.185625E+08
.17	1.996016E+07

T	PE(T)	PT(T)
24	26.5	31.32528
34	35.8	36.67073
44	41.39999	42.92833
54	54.5	50.25373
64	64.10001	58.82924
74	72.7	68.86805
84	91.7	80.61983
94	107	94.37714
104	126.8	110.4818
114	145.1	129.3348
124	178.3	151.405
134	205.2	177.2412
144	249.7999	207.4862
154	290.8	242.8922
164	351.1999	284.3404
174	424.8999	332.8609
184	486.5039	389.6614
194	584.1021	456.1541
204	686.5039	533.9941
214	796.4018	625.1155
224	952.2031	731.7881
234	1.121504E+03	856.6629
244	1.305504E+03	1.002846E+03
254	1.564203E+03	1.173975E+03
264	1.835505E+03	1.374304E+03
274	2.159602E+03	1.608821E+03
284	2.520004E+03	1.883354E+03
294	2.948003E+03	2.204737E+03
304	3.433103E+03	2.580961E+03
314	4.028104E+03	3.021382E+03
324	4.684305E+03	3.536963E+03
334	5.420805E+03	4.140520E+03
344	6.291504E+03	4.847071E+03
354	7.164206E+03	5.674187E+03
364	8.259750E+03	6.642448E+03
374	9.327124E+03	7.775932E+03
384	1.060831E+04	9.102847E+03
394	1.197375E+04	1.065619E+04

380 STOP

TABLE 5.2

EXECUTION OF STORED PROGRAM USING

IMMEDIATE GO TO STATEMENT

```
GO TO 2

160 RHO=0.17
160 DRHO=1/1000

MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)
                T0=24 P0=26
```

RHO	SUM OF (PE-PT)**2
.17	1.996016E+07
.1710001	1.386325E+07
.1719999	8.981009E+06
.1729999	5.496504E+06
.1739998	3.613998E+06
.1749998	3.559896E+06

T	PE(T)	PT(T)
24	26.5	31.51515
34	35.8	37.1008
44	41.39999	43.67643
54	54.5	51.41748
64	64.10001	60.53062
74	72.7	71.25887
84	91.7	83.88853
94	107	98.75661
104	126.8	116.26
114	145.1	136.8654
124	178.3	161.1232
134	205.2	189.6802
144	249.7999	223.2984
154	290.8	262.8752
164	351.1999	309.4661
174	424.8999	364.3147
184	486.5039	428.8852
194	584.1021	504.8994
204	686.5039	594.386
214	796.4018	699.733
224	952.2031	823.7513
234	1.121504E+03	969.751
244	1.305504E+03	1.141626E+03
254	1.564203E+03	1.343965E+03
264	1.835505E+03	1.582166E+03
274	2.159602E+03	1.862582E+03
284	2.520004E+03	2.192700E+03
294	2.948003E+03	2.581326E+03
304	3.433103E+03	3.038836E+03
314	4.028104E+03	3.577429E+03
324	4.684305E+03	4.211477E+03
334	5.420805E+03	4.957906E+03
344	6.291504E+03	5.836634E+03
354	7.164206E+03	6.871094E+03
364	8.259750E+03	8.088913E+03
374	9.327124E+03	9.522563E+03
384	1.060831E+04	1.121032E+04
394	1.197375E+04	1.319718E+04

```
380 STOP
```

TABLE 5.3

LIST STATEMENT EXAMPLES

```
LIST 170
170     TYPE !'MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)'
LIST 210,250
210 3   T←T0+SUM←0
220     B←RHO/(13*A←1-RHO)
230     A←P0*EXP(-B*T0)/A
240 4   SUM←SUM+C*C←PE(T)-A*EXP(B*T)


LIST 170
170     TYPE !'MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)'

LIST 210,250
210 3   T←T0+SUM←0
220     B←RHO/(13*A←1-RHO)
230     A←P0*EXP(-B*T0)/A
240 4   SUM←SUM+C*C←PE(T)-A*EXP(B*T)
250     IF(T←T+DT.LE.TEND) GO TO 4

LIST
110     ACCEPT T0,P0,TEND
120     DT←10
130     T←T0
140 1   ACCEPT PE(T)
150     IF(T←T+DT.LE.TEND) GO TO 1
160 2   ACCEPT RHO,DRHO
170     TYPE !'MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)'
180     TYPE <19>,'T0=',T0,' P0=',P0!!
190     PSUM←1E70
200     TYPE <14>,'RHO',<24>,'SUM OF (PE-PT)**2'
210 3   T←T0+SUM←0
220     B←RHO/(13*A←1-RHO)
230     A←P0*EXP(-B*T0)/A
240 4   SUM←SUM+C*C←PE(T)-A*EXP(B*T)
250     IF(T←T+DT.LE.TEND) GO TO 4
260     IF(SUM.GE.PSUM) GO TO 5
270     TYPE <12>,RHO,<27>,SUM
280     RHO←RHO+DRHO
290     PSUM←SUM
300     GO TO 3
310 5   TYPE !<13>,'T',<23>,'PE(T)',<40>,'PT(T)'
320     T←T0
330     B←RHO/(13*A←1-RHO←RHO-DRHO)
340     A←P0*EXP(-B*T0)/A
350 6   TYPE <12>,T,<20>,PE(T),<37>,A*EXP(B*T)
360     IF(T←T+DT.LE.TEND) GO TO 6
370     TYPE !!
380     STOP
```

TABLE 5.4

TRACING INDIVIDUAL STATEMENTS

(a)

```
TRON 210
TRON 220
TRON 230
TRON 280
TRON 290
GO TO 2

160 RHO=0.17
160 DRHO=1/1000

MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)
                T0=24 P0=26

              RHO          SUM OF (PE-PT)**2
210 SUM=0
210 T=24
220 A=.83
220 B=.01575533
230 A=21.4623
              .17           1.996816E+07
280 RHO=.1710001
290 PSUM=1.996016E+07
210 SUM=0
210 T=24
220 A=.8289999
220 B=.01586712
230 A=21.43064
              .1710001      1.386325E+07
280 RHO=.1719999
290 PSUM=1.386325E+07
210 SUM=0
210 T=24
220 A=.8280001
220 B=.01597918
230 A=21.39887
              .1719999      8.981009E+06
280 RHO=.1729999
290 PSUM=8.981009E+06
210 SUM=0
210 T=24
220 A=.827
220 B=.01609152
230 A=21.36706
              .1729999      5.496504E+06
280 RHO=.1739998
290 PSUM=5.496504E+06
210 SUM=0
210 T=24
220 A=.826
220 B=.01620413
230 A=21.3352
              .1739998      3.613998E+06
280 RHO=.1749998
290 PSUM=3.613998E+06
210 SUM=0
210 T=24
220 A=.825
220 B=.01631701
230 A=21.30327
              .1749998      3.559896E+06
```

(b)

```
TROFF 210
TROFF 220
TROFF 230
TROFF 280
TROFF 290
GO TO 2

160 RHO=0.17
160 DRHO=1/1000

MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)
                T0=24 P0=26

        RHO          SUM OF (PE-PT)**2
      .17              1.996016E+07
      .1710001         1.386325E+07
      .1719999         8.981009E+06
      .1729999         5.496504E+06
      .1739998         3.613998E+06
      .1749998         3.559896E+06

   T        PE(T)          PT(T)
  24        26.5           31.51515
  34        35.8           37.1008
  44        41.39999       43.67643
  54        54.5           51.41748
  64        64.10001       60.53062
  74        72.7           71.25887
  84        91.7           83.88853
  94        107            98.75661
 104        126.8          116.26
 114        145.1          136.8654
 124        178.3          161.1232
 134        205.2          189.6802
 144        249.7999       223.2984
 154        298.8          262.8752
 164        351.1999       309.4661
 174        424.8999       364.3147
 184        486.5039       428.8852
 194        584.1021       504.8994
 204        686.5039       594.386
 214        796.4018       699.733
 224        952.2031       823.7513
 234        1.121504E+03   969.751
 244        1.305504E+03   1.141626E+03
 254        1.564203E+03   1.343965E+03
 264        1.835505E+03   1.582166E+03
 274        2.159602E+03   1.862582E+03
 284        2.520004E+03   2.192700E+03
 294        2.948003E+03   2.581326E+03
 304        3.433103E+03   3.038836E+03
 314        4.028104E+03   3.577429E+03
 324        4.684305E+03   4.211477E+03
 334        5.420805E+03   4.957906E+03
 344        6.291504E+03   5.836634E+03
 354        7.164206E+03   6.871094E+03
 364        8.259750E+03   8.088913E+03
 374        9.327124E+03   9.522563E+03
 384        1.060831E+04   1.121032E+04
 394        1.197375E+04   1.319718E+04

380 STOP
```

TABLE 5.5

COMPLETE TRACE OF PROGRAM

(a)

```
TRON
GO TO 2

160 2  ACCEPT RHO,DRHO
160 RHO=0.17
160 DRHO=1/1000
170    TYPE :'MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)'

MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)
180    TYPE <19>,'T0=',T0,' P0=',P0;;
                  T0=24 P0=26

190    PSUM=1E70
190 PSUM=1.000000E+70
200    TYPE <14>,'RHO',<24>,'SUM OF (PE-PT)**2'
             RHO        SUM OF (PE-PT)**2
210 3  T=T0+SUM=0
210 SUM=0
210 T=24
220    B=RHO/(13*A-1-RHO)
220 A=.83
220 B=.01575533
230    A=P0*EXP(-B*T0)/A
230 A=21.4623
240 4  SUM=SUM+C*C-PE(T)-A*EXP(B*T)
240 C=-4.825287
240 SUM=23.28339
250    IF(T=T+DT.LE.TEND) GO TO 4
250 T=34
240 4  SUM=SUM+C*C-PE(T)-A*EXP(B*T)
240 C=-.8707273
240 SUM=24.04155
250    IF(T=T+DT.LE.TEND) GO TO 4
250 T=44
240 4  SUM=SUM+C*C-PE(T)-A*EXP(B*T)
240 C=-1.528336
240 SUM=26.37736
250    IF(T=T+DT.LE.TEND) GO TO 4
250 T=54
240 4  SUM=SUM+C*C-PE(T)-A*EXP(B*T)
240 C=4.246262
240 SUM=44.40889
250    IF(T=T+DT.LE.TEND) GO TO 4
250 T=64
240 4  SUM=SUM+C*C-PE(T)-A*EXP(B*T)
240 C=5.270767
240 SUM=72.18908
250    IF(T=T+DT.LE.TEND) GO TO 4
250 T=74
240 4  SUM=SUM+C*C-PE(T)-A*EXP(B*T)
240 C=3.831955
240 SUM=86.87297
250    IF(T=T+DT.LE.TEND) GO TO 4
250 T=84
240 4  SUM=SUM+C*C-PE(T)-A*EXP(B*T)
240 C=11.08017
240 SUM=209.6431
```

(b)

```
TROFF
GO TO 2

160 RHO=0.17
160 DRHO=1/1000

MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)
                T0=24 P0=26

      RHO        SUM OF (PE-PT)**2
 .17             1.996016E+07
 .1710001        1.386325E+07
 .1719999        8.981009E+06
 .1729999        5.496504E+06
 .1739998        3.61399RE+06
 .1749998        3.559896E+06

  T        PE(T)          PT(T)
 24        26.5           31.51515
 34        35.8           37.1008
 44        41.39999       43.67643
 54        54.5           51.4174R
 64        64.10001       60.53062
 74        72.7           71.25KR7
 84        91.7           83.88853
 94        107            98.75661
104        126.8          116.26
114        145.1          136.8654
124        178.3          161.1232
134        205.2          189.6882
144        249.7999       223.2984
154        290.8          262.8752
164        351.1999       309.4661
174        424.8999       364.3147
184        486.5039       428.8852
194        584.1021       504.K994
204        686.5039       594.386
214        796.4018       699.733
224        952.2031       823.7513
234        1.121504E+03   969.751
244        1.305504E+03   1.141626E+03
254        1.564203E+03   1.343965E+03
264        1.835505E+03   1.582166E+03
274        2.159602E+03   1.862582E+03
284        2.520004E+03   2.192700E+03
294        2.948003E+03   2.561326E+03
304        3.433103E+03   3.038836E+03
314        4.028104E+03   3.577429E+03
324        4.684305E+03   4.211477E+03
334        5.420805E+03   4.957906E+03
344        6.291504E+03   5.836634E+03
354        7.164206E+03   6.871094E+03
364        8.259750E+03   8.088913E+03
374        9.327124E+03   9.522563E+03
384        1.064831E+04   1.121032E+04
394        1.197375E+04   1.319718E+04

380 STOP
```

TABLE 5.6

INTERRUPTING EXECUTION OF A STORED PROGRAM

```
GO TO 2

160 RHO←0.17
160 DRHO←1/1000

MOATA REACTOR - 1 DELAYED GROUP APPROXIMATION (C=1)
                T0=24 P0=26

              RHO         SUM OF (PE-PT)**2
             .17           1.996016E+07
250 ?
T
  44
RHO
  .1710001
240 ?
LIST 240
240 4  SUM←SUM+C*C←PE(T)-A*EXP(B*T)
SUM
  3.881973E+04
T
  204
GO TO 380

380 STOP
```

? typed here

CR typed here

? typed here

immediate GO TO statement executed to finish program

APPENDIX 5.1

ACL-NOVA SUMMARY

INTRODUCTION

ACL is a higher-level language specifically designed to suit scientific problems. The language was developed at the Australian Atomic Energy Commission Research Establishment, Lucas Heights, New South Wales, and was implemented on a NOVA computer supporting multiple terminals - this is referred to as the ACL-NOVA system.

The ACL language consists of immediate statements and stored statements. Stored statements are translated and saved in a user work area from which they may be recalled and executed under user control. Each stored statement has a sequence number in the range 100 to 999 and may also have a statement number in the range 0 to 99. Stored statements are used to build up a stored program. They are ordered according to their sequence number and may be inserted, modified, or deleted freely by the user. Stored statements may be typed in any order, and any newly entered statement will replace a previously saved statement with the same sequence number.

An immediate statement, which does not have a sequence number, is translated and executed when typed and is then discarded. Immediate statements are used to perform one-time or 'desk calculator' calculations, to control the execution of a stored program and to perform various editing and debugging functions.

The ACL-NOVA system is a stand-alone system that uses the first 6 K of memory on the NOVA computer. The number of terminals to be supported by the system, the size of work area increments and the available user area must be specified when the system is first loaded into the computer. Work space may be reserved for each terminal or left 'floating' to provide the greatest flexibility for the system. Dynamic allocation of work space allows the 'floating' space to be assigned to users who need extra work area, this work space being returned to the system when the user completes work at a terminal.

INPUT FROM A TERMINAL

To begin work at a terminal, the user presses the CTRL/G combination of keys and the system responds by typing ACL-NOVA and spacing a few lines. Once the terminal has been initialised in this way, statements may be stored or immediate statements executed.

Input begins from column 1, which is taken to be the leftmost position of the teletype carriage that results from pressing the Carriage Return (CR) key, and may consist of up to 72 characters. If input is continued past column 72,

the whole line is cancelled and must be entered again. In the ACL-NOVA system then, one can think of input on a 72 column layout. For statements to be stored for later execution, this layout takes the form

```
1  4 5  7 8                    ACL                                    72
230     A←4
235 1   B←6
240 23  A+B
```

Immediate statements begin from column 1 and have the layout

```
1                               ACL                                    72
26+4
A2←23.5
SQR(42.978)
```

The terminals attached to the NOVA computer are operated in what is termed full-duplex mode. This means that a character pressed at the teletype keyboard is sent to the computer, but is not printed at the teletype unless it is sent back to the terminal (echoed) by the NOVA computer. By making use of this feature, only characters that are valid in syntax are 'echoed' at a terminal; that is, only characters that make sense are typed at the terminal. For example, if one pressed the keys 3++3 at the keyboard, the system would only type 3+3 at your terminal. A statement is executed when it is completed by a 'valid' carriage return.

The full-duplex mode feature is also used to simplify the task of entering stored statements from a terminal. If the user gives a space at the column 1 position the system responds with a sequence number followed by a space. This sequence number is 10 greater than the last sequence number specified. In the same way, if a space is entered at the column 5 position, the system responds with 3 spaces. Thus for a statement to be stored, a new sequence number and a null statement number may be nominated by entering two spaces.

Stored statements are deleted by entering sequence number-space-carriage return.

ARITHMETIC

All arithmetic operations are carried out using single precision floating point numbers. For input, the numbers may have free format, that is, they may be integers, may contain a decimal point or may contain an exponent.

Examples are: 327, 1.231, 0.0014, -1.45E+12, 3E2

VARIABLES

Three types of variables may be used - simple variables, singly subscripted variables or doubly subscripted variables.

A simple variable name must begin with an alphabetic character and may be followed by up to three alphanumeric characters. Subscripted variable names consist of an alphabetic character, which may be followed by an alphanumeric character, followed by the subscript (or subscripts) enclosed in brackets. Singly subscripted variables must have subscripts in the range 0 to 65535. Subscripts for doubly subscripted variables must be in the range 0 to 255.

OPERANDS AND OPERATORS

The basic operands in an arithmetic statement or expression are variables and numbers. The relevant operators are +,-,*,/(divide),↑(power),←(assignment) and the functions ABS, ATN, COS, DPT, EXP, INT, LOG, SIN and SQR. The mathematical operators have the usual hierarchy of ↑ first, * or / next, and then + or - .

ASSIGNMENTS

If a variable is followed by an assignment arrow, then during statement execution the expression to the right of the assignment arrow is evaluated and its value given to that variable. If a number of assignment arrows occur in an expression then they are processed from right to left.

ARITHMETIC STATEMENTS AND EXPRESSIONS

An expression is termed an arithmetic statement if the first term in the expression is a variable followed by an assignment arrow; otherwise it is an arithmetic expression.

The result of executing an arithmetic expression is typed at the terminal in two possible formats depending on its magnitude. The result is given in exponent form if $|\text{result}| \leq 10^{-4}$ or $|\text{result}| \geq 10^{3}$; otherwise it is typed in non-exponent form with seven significant figures.

Examples:

(i) 3↑2+AB1←2.453 is an arithmetic expression and during execution AB1 would be assigned the value 2.453 and the result 11.453 would be printed at the terminal.

(ii) AX←SQR(C←4*B(1)←1) +3-B←-2 is an arithmetic statement and during execution B(1) would first be assigned the value 1, C would then be assigned the value 4, B would be assigned the value -2 and AX would finally be assigned the value 7, but nothing would be printed at the terminal.

SEQUENCE NUMBERS AND STATEMENT NUMBERS

Every statement that is to be stored for later execution must have a sequence number in the range 100 to 999. These statements are stored in the user's work area in an order based upon increasing sequence numbers.

A one or two digit statement number in the range 0 to 99 may also be associated with each stored statement.

ERROR CORRECTION

Errors that occur while a statement is being entered at the keyboard may be corrected quite simply. To delete the last two characters that were accepted as input, for example, type <2). A statement may be corrected in edit mode by typing <<) after the last input character. Finally the whole input line may be deleted by typing <<< after the last input character.

The EDIT statement allows stored statements to be modified in edit mode. Using this statement characters may be copied from, inserted into or deleted from an existing statement or a complete statement copied by changing the sequence number.

STORED PROGRAM EXECUTION

Execution of the RUN statement or the immediate GO TO statement begins stored program execution. Input to a stored program can be read by using the ACCEPT statement, while the IF statement can be used for loop control.

The user can interrupt stored program execution at any time by entering the character? At this point, stored statements may be inserted, modified or deleted, or immediate statements executed. If the user subsequently enters CR at the column 1 position, then execution continues from the point where the program was suspended. Execution of the stored program may recommence at a different point by executing an immediate GO TO statement, or it may be restarted by executing a RUN statement.

Stored program execution may also be suspended if an error condition occurs, or as the result of certain PAUSE conditions. This allows the user to take corrective action, or to turn local or global trace indicators ON before continuing execution of the stored program.

A STOP statement is used to complete stored program execution, while an END statement should always be the last statement executed when a user has completed his work at a terminal.

INPUT FROM PAPER TAPE

Standard programs should be held on paper tape and may be entered from the paper tape reader once a user has commenced work at a terminal. The value of variables to satisfy a series of ACCEPT statements in a program may also be read from paper tape. The LIST: and SYMBOLS: options make it very convenient to use input in this form.

LIST OF ACL STATEMENTS

A. IMMEDIATE STATEMENTS

Comments

Arithmetic statements

Arithmetic expressions

RUN

GO␣TO␣ { arith stmt or exprn}

LIST [:] [␣ arith stmt or exprn [,arith stmt or exprn]]

SYMBOLS [:]

CLEAR [␣ variable [,variable] ...]

SPACE

FTRON

FTROFF

TRON [␣ arith stmt or exprn]

TROFF [␣ arith stmt or exprn]

TYPE [␣ { [: [:] ...] operand [{: [:] ...} operand [: [:] ...] ...] ; [:] ... }]

where the operands take the form, [<arith stmt or exprn>,] { 'characters' / ["] arith stmt or exprn }

PB ␣ {arith stmt or exprn}

PA ␣ {arith stmt or exprn}

STOP

END

SUSPEND [:]

EDIT ␣ {arith stmt or exprn}

System Messages

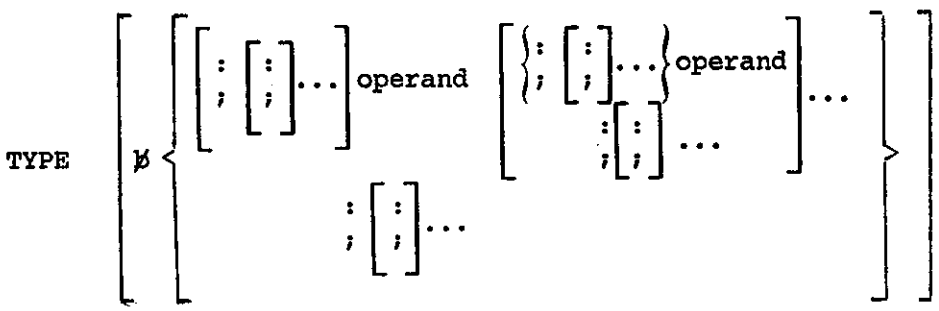
B. STORED STATEMENTS

Comments

Arithmetic statements

Arithmetic expressions

GO␣TO␣ { arith stmt or exprn}



where the operands take the form, [< arith stmt or exprn >,] { 'characters' / ["] arith stmt or exprn }

ACCEPT ␣ { variable [,variable] ...}

CALL ␣ { arith stmt or exprn}

RETURN

CONTINUE

STOP

IF ({ arith stmt or exprn} { .EQ. / .NE. / .GT. / .GE. / .LT. / .LE. } { arith stmt or exprn}) ␣ { arith stmt or exprn / TYPE stmt / ACCEPT stmt / GO TO stmt / CALL stmt / RETURN stmt / CONTINUE stmt / STOP stmt }

PAUSE

END

CHAPTER 6

ANALOGUE AND HYBRID COMPUTERS

Lecture by

C.P. GILBERT

CONTENTS

6.1 ANALOGUES

When examining physical systems such as the reactor problem already introduced, we use mathematical equations to describe their behaviour. In some instances the situation is reversed, and we use carefully chosen physical processes to help us understand, or 'solve' a set of equations. Such processes are called analogues.

Figure 1a shows how addition can be performed using a liquid: if the contents of the smaller containers are emptied into a sufficiently large container the final volume of liquid is the sum of the initial volumes.

$$V = v_1 + v_2 + v_3 + v_4 + v_5 \ .$$

A more important process is integration, and this can be achieved as shown in Figure 1b. The height H of the fluid in a container of base area A, is the integral, with respect to time, of the fluid flow F (volume/sec) as determined by the tap.

$$H = \frac{1}{A}\int_0^t F \, dt \ .$$

These simple analogues would be of no use in practice: they are inaccurate, slow, unsuitable for interconnection, and probably wet. However, there are much better analogue processes available, and the most important of them is described in the following section.

6.2 OPERATIONAL AMPLIFIER CIRCUITS

An operational amplifier has the following properties, illustrated in Figure 2:

(a) Very high voltage amplification, or 'gain' K (10^6 say).

(b) Negative gain (a positive input produces a negative output, and vice versa).

(c) Works at d.c. as well as a.c. for all frequencies up to perhaps 10^6 Hz.

For computing purposes, such amplifiers are used in a feedback circuit which exchanges the high voltage gain for other more desirable properties. The circuit of Figure 3a is arranged so that the voltages v_1, v_2, v_3 and V_0 are all of the order of a few volts. Then, if $K = 10^6$, the amplifier input voltage u is equal to $\frac{-V_0}{K}$, which never exceeds a few microvolts and can usually be neglected. For instance, if the combined effect of all the inputs is positive, u tries to go positive: this causes V_0 to go negative by a very much larger amount, which opposes the rise in u because of R_f. Finally u ends up very slightly positive, causing a negative output V_0.

If we assume that u is zero, the current i is the sum of the input currents:

$$i = \frac{v_1}{R_1} + \frac{v_2}{R_2} + \frac{v_3}{R_3} \quad .$$

This current cannot enter the amplifier, but is drawn through R_f by V_0, and so

$$i = -\frac{V_0}{R_f}$$

Eliminating i,

$$-\frac{V_0}{R_f} = \frac{v_1}{R_1} + \frac{v_2}{R_2} + \frac{v_3}{R_3}$$

leading to

$$V_0 = -\left[v_1 \frac{R_f}{R_1} + v_2 \frac{R_f}{R_2} + v_3 \frac{R_f}{R_3}\right] \quad .$$

Thus the output is minus the sum of the input voltages. The resistance ratios $\frac{R_f}{R_1}$ are normally fixed at convenient values, such as 1 or 10, and variable coefficients are introduced using potentiometers (see Figure 3b). It is a pity that the amplifier gives a reversal in sign, but it is unavoidable, and causes little difficulty.

The complete adder circuit is conventionally drawn as in Figure 3c, the values of the resistance ratios being marked only if they are other than unity. Then

$$V_0 = -\left[0.3\ V_1 + 1.6\ V_2 + V_3\right] \quad .$$

Note that all voltages are measured with respect to earth, although the earth connection itself is omitted.

In the integrator circuit of Figure 4a, a feedback capacitor C is used. Assuming as before that u = 0,

$$i = \frac{v}{R} \quad .$$

Again, the current is constrained to flow into the feedback component, but in this case the voltage is proportional to the integral of the current i with respect to time t, namely

$$V_0 = -\frac{1}{C}\int_0^t i\ dt$$

and substituting for i shows that

$$V_0 = -\frac{1}{CR}\int_0^t v\ dt \quad .$$

A constant value of v causes V_0 to change at a constant rate, a sine input gives a cosine output and so on (Figure 4b). Unless otherwise shown, the time constant CR can be assumed to be unity, and the integrator circuit is conventionally drawn as in Figure 4c, for which

$$V_0 = -0.6 \int_0^t V_1 \, dt$$

or $$-\frac{dV_0}{dt} = 0.6\, V_1 \quad .$$

Accuracies better than 0.1% can be obtained without difficulty for adders and integrators of the type shown.

6.3 ELECTRONIC ANALOGUE COMPUTERS

An electronic analogue computer consists of a number of operational amplifiers which can be used for addition, integration, multiplication and a range of other functions: facilities are provided which permit the interconnection and switching of the computing circuits, and which allow accurate measurements to be made on them. The problem variables in which we are interested (flux, velocity or force, for instance) are all represented in the computer by voltages. These voltages may vary quite slowly, and can then be read on a voltmeter, or they may change so quickly that an oscilloscope is required to observe them.

A medium sized machine might have about 100 amplifiers, including perhaps 30 integrators, and could thus perform 30 integrations at the same time.

Consider the circuit of Figure 5. The extra input on top of the integrator is inverted, and supplies a fixed voltage b to the output as an 'initial condition' before the integration starts, but has no other effect. When switch A is closed, this defines the instant which the computer regards as t = 0; at this time, V = b. We have now made the integrator input equal to V, and so the circuit obeys the equation

$$-\frac{dV}{dt} = V \qquad \text{or} \qquad \frac{dV}{dt} = -V \qquad \ldots(1)$$

(Simple analysis shows that if we let $V = ke^{-t}$, where k is an unknown constant, then differentiating we get

$$\frac{dV}{dt} = -ke^{-t} = -V \quad .$$

This demonstrates that $V = ke^{-t}$ is a solution of Equation (1). Since we have made V = b at t = 0, substitution shows that k = b, and so the solution is $V = be^{-t}$.)

The circuit 'solves' Equation (1) by producing a voltage proportional to

be^{-t} each time switch A is closed. V starts off positive, and, via the integrator, forces itself to get smaller. As it does so, it falls more slowly, reaching zero exponentially.

Switches such as A, and many other controls which the computer needs, are usually omitted from the computing circuit - their presence is assumed.

6.4 PROBLEM SOLUTION

In the previous section we started with a computer circuit and analysed its behaviour. The usual process is the other way round - we are given an equation and have to design a circuit which will solve it, resulting in the process illustrated in Figure 6. The equivalence between the physical system and the analogue circuit is very marked, and examination of the behaviour of the latter, used as a working model or 'simulator', provides valuable insight into the operation of the original system. In some cases simulators behave so much like the original system that they are used to train operators, and the analogue circuit which we will develop can be used in this way.

Our problem has been given as (Chapter 2, Equation (1))

$$T_c \frac{d}{dt}\left[(1-\rho(t))\ p(t)\right] = \rho(t)\ p(t) + R_c\left[p(t) - p(t-T_\infty)\right]$$

where $$R_c = (1-c)\left[\frac{T_c}{T_\infty} - 1\right]$$

$$T_c = (4 + 9/c) \text{ sec.}$$

and so $T_\infty = 4$ sec.

The usual value for c is 0.29, but we may investigate the use of other values.

For convenience we will rewrite the equation using

$$P(t) = \left[(1-\rho(t))\ p(t)\right] \qquad \ldots(2)$$

and dividing through by T_c we get

$$\frac{dP(t)}{dt} = \frac{\rho(t)\ p(t)}{T_c} + \frac{R_c}{T_c}\left[p(t) - p(t-T_\infty)\right] \qquad \ldots(3)$$

For each of the MOATA experiments, $\rho(t)$ only takes on a fixed value ρ, and we will consider this case first.

Suppose that a signal proportional to $\frac{dP(t)}{dt}$ is available at the input to integrator 35 in the circuit of Figure 7a. The output of this integrator is proportional to

$$-P(t) = -\left[(1-\rho)\ p(t)\right]$$

and using potentiometer 68 to divide by the (fixed) value of $(1-\rho)$ gives the reactor power $p(t)$ at the output of adder 38 : $-p(t)$ is available from adder 3, which is simply an inverter.

The $p(t)$ signal from adder 38 is also fed to a unit marked 'DELAY' which inverts the signal and reproduces it after a delay of T_∞ seconds. Combining this with $p(t)$ in adder 65 produces the signal

$$-\left[p(t) - p(t-T_\infty)\right] \quad .$$

If now potentiometer 34 is set to the value $\frac{\rho}{T_c}$ and potentiometer 36 to $\frac{R_c}{T_c}$, the total input to adder 36 is minus the right hand side of Equation (3). It follows that the output of adder 36 is proportional to the left hand side of Equation (3), i.e. $\frac{dP(t)}{dt}$, and can be used as the input to the integrator 35, thus completing the circuit.

To obtain a given solution, the operator would set potentiometer 35 to give the required initial value of $P(t)$, and potentiometers 34 and 68 to the required values of $\frac{\rho}{T_c}$ and $\frac{1}{(1-\rho)}$ respectively. After the circuit was started, the varying voltage at the output of adder 38 ('reactor power') would be plotted on a chart, or displayed on an oscilloscope. By this means, results similar to those of the MOATA experiments could be obtained.

While the circuit of Figure 7a behaves in exactly the same way as the reactor described in Equation (3), it cannot be used as a simulator because an operator would be unable to change potentiometers 34 and 68 at the same time. As we were aware, this circuit is only suitable for fixed values of ρ.

The circuit of Figure 7b overcomes this drawback. It is the same as the circuit of Figure 7a except that adders 3 and 38 have been replaced by multipliers. A single adjustment to potentiometer 2 (equivalent to an adjustment of the MOATA control rods) gives the value of $\rho(t)$ and provides the corresponding value of $\frac{1}{(1-\rho(t))}$ via the non-linear circuit F37. These signals are introduced via the multipliers 3 and 38 respectively. The reactivity can now be changed even while the circuit is operating, i.e. even when $p(t)$ is varying, just as in the reactor, and the skill obtained in practicing with the simulator would be very helpful in learning to operate the reactor itself.

Notice that in both circuits all the computing operations occur <u>simultaneously</u>, (in 'parallel'), not in <u>sequence</u> as in a digital computer, and that the solution arises at exactly the same speed as the reactor power would change. By using smaller capacitors in the integrators, the operation can be made very much faster, and solutions can be obtained in a few

milliseconds. If many integrations are involved, the overall operation can be made far faster than can be achieved by a digital computer. However, these very high speeds cannot be properly utilised except as described in Section 6.5.

Pure analogue computers have a number of shortcomings, and in general their usefulness is limited to the solution of ordinary differential, and straightforward algebraic equations. However, within this field they provide valuable insight as well as useful answers.

6.5 HYBRID COMPUTERS

A recent development, whose full impact has not yet been felt, is the Hybrid Computer. This consists of an analogue computer, a general purpose digital computer, and an interface. The latter provides the facilities required for the two machines to interchange signals, thus permitting them to cooperate effectively (Figure 8).

The analogue computer allows high speed, parallel computation. The digital computer can be programmed to:

(a) Perform the scaling calculations and check the analogue circuit, thus saving a great deal of time.

(b) Act as a very high speed operator, who readjusts the analogue computer before each solution, as determined by the preceding solution. This permits the very high analogue computing speeds mentioned above to be used effectively.

(c) Perform parts of the computation which the analogue computer finds difficult (or impossible in some cases).

The Delays of Figure 7 form a simple example of (c) above. While a delay can be produced by pure analogue means, it is very much simpler to read the information into the store of the digital computer, leave it there for the desired time (T_∞ = 4 sec. in this case) and then transfer it back to the analogue computer again.

Figure 9 illustrates the process. At a given instant the input signal is measured by the Analogue to Digital converter, and the digital value stored. A short time td later another value is measured and stored in the next memory location. This process continues indefinitely. Each time a new value is measured, a value which has been held in the store for T_∞ seconds is returned to analogue form by the Digital to Analogue converter, which in this case also inverts the signal. Once a stored value has been read back, it is no longer of any use and the same storage locations can be used over and over again. Thus only a small part of the digital computer is actually

employed ($\frac{T_\infty}{t_d}$ locations), and it can perform other duties as well.

There are few problems which a hybrid computer cannot profitably undertake, and considerable use is made of the facilities described in items (a) and (b) above. We are at present doing our best to gain experience in the application of item (c) to large problems.

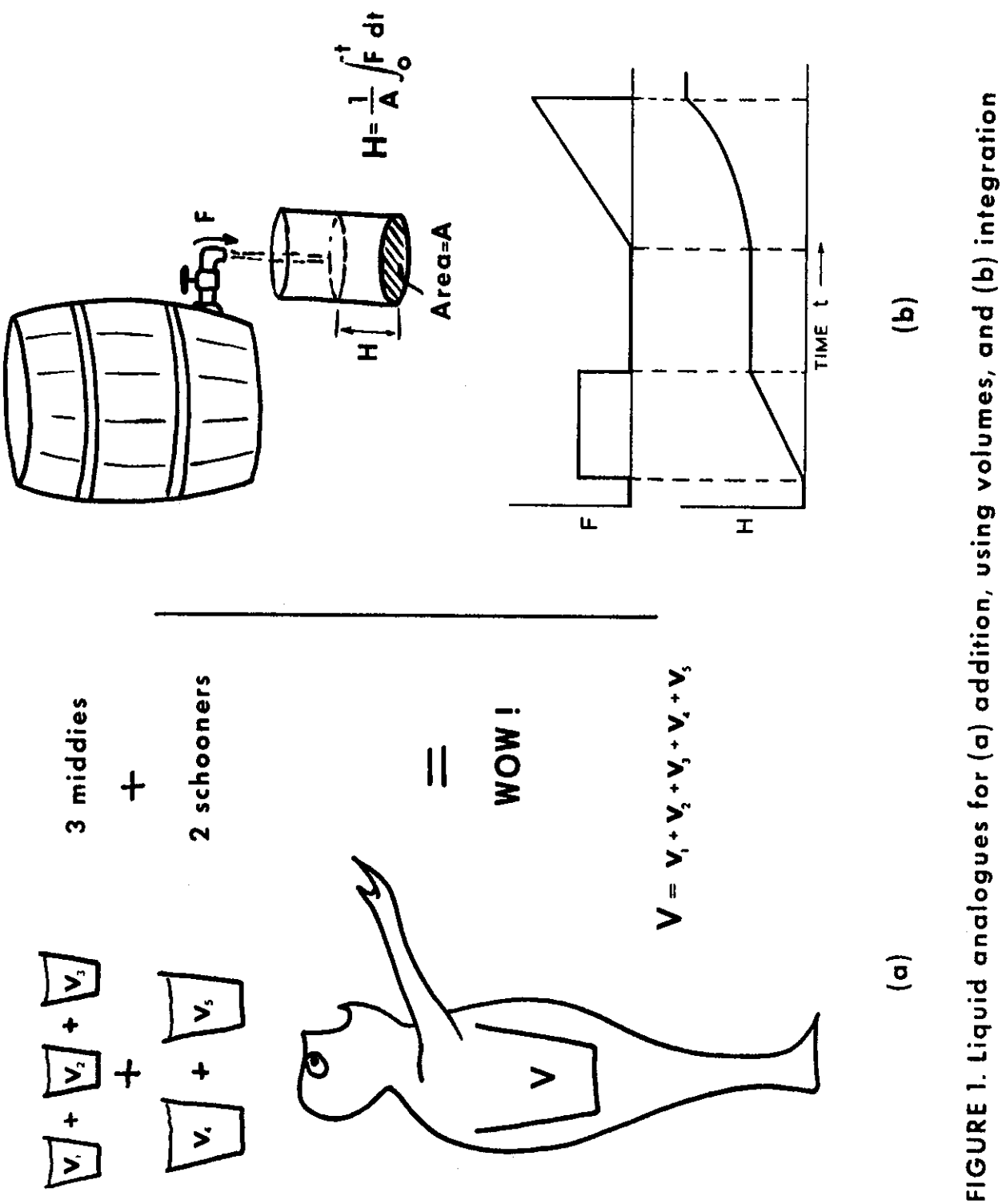


FIGURE 1. Liquid analogues for (a) addition, using volumes, and (b) integration

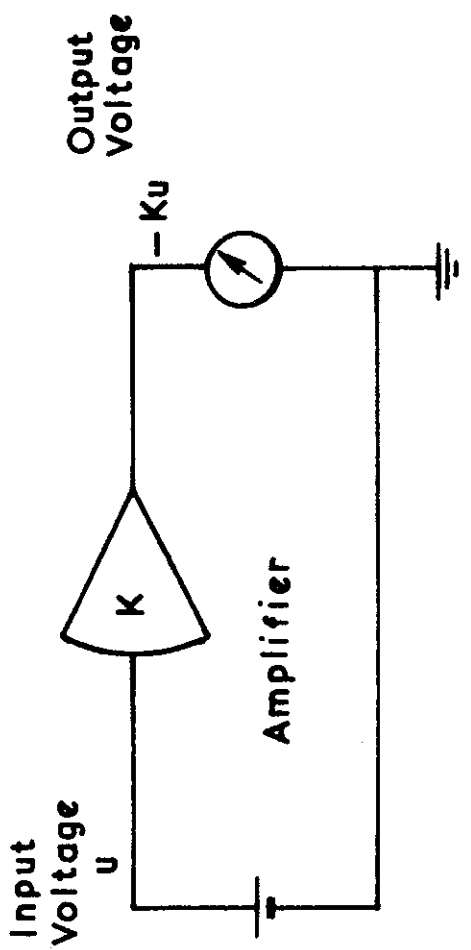
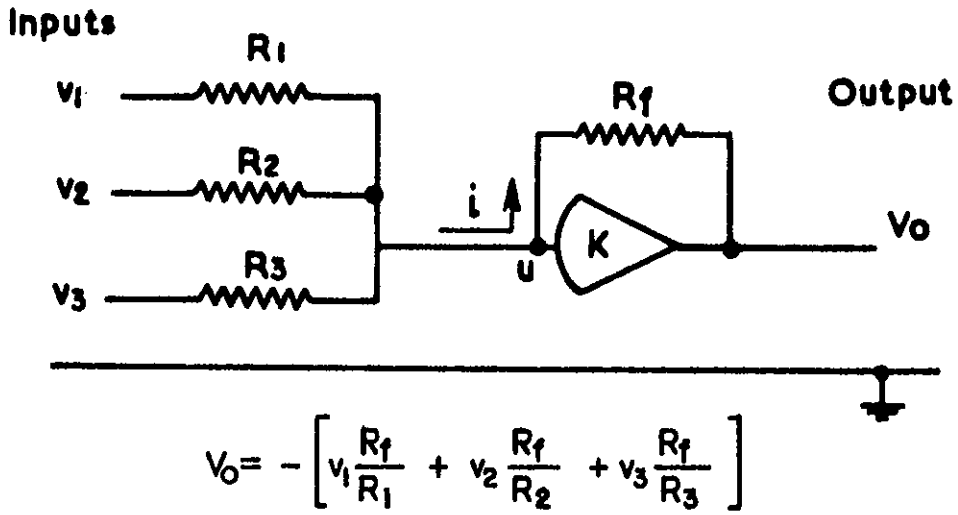
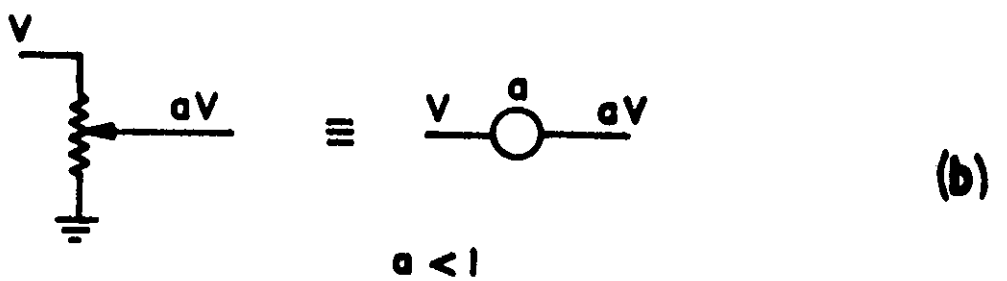


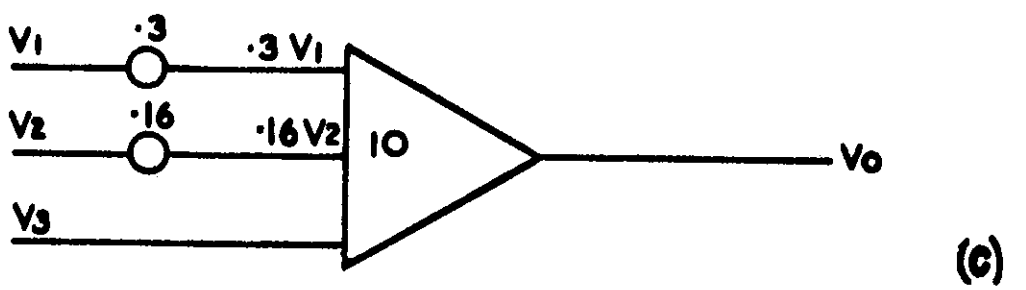
FIGURE 2. A circuit demonstrating the properties of an operational amplifier. The amplifier itself is represented by the triangular symbol with a curved side at the input



$$V_o = -\left[v_1\frac{R_f}{R_1} + v_2\frac{R_f}{R_2} + v_3\frac{R_f}{R_3}\right]$$

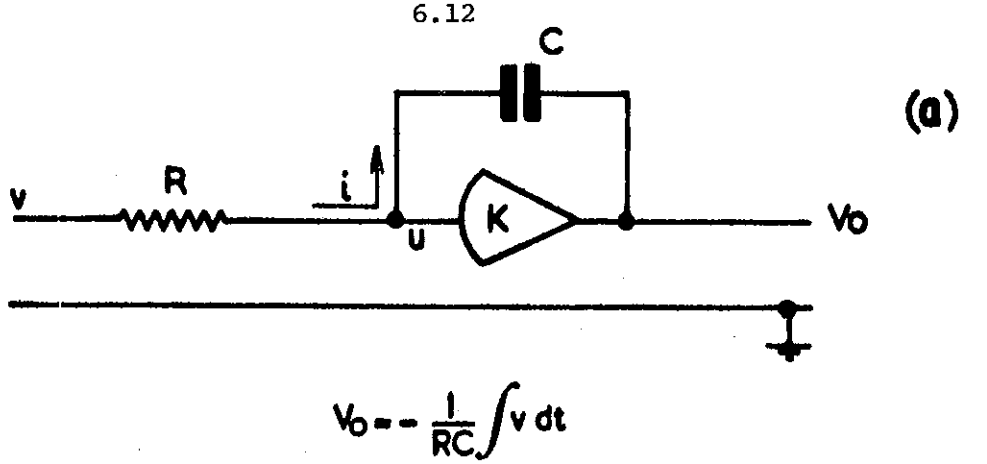


$a < 1$

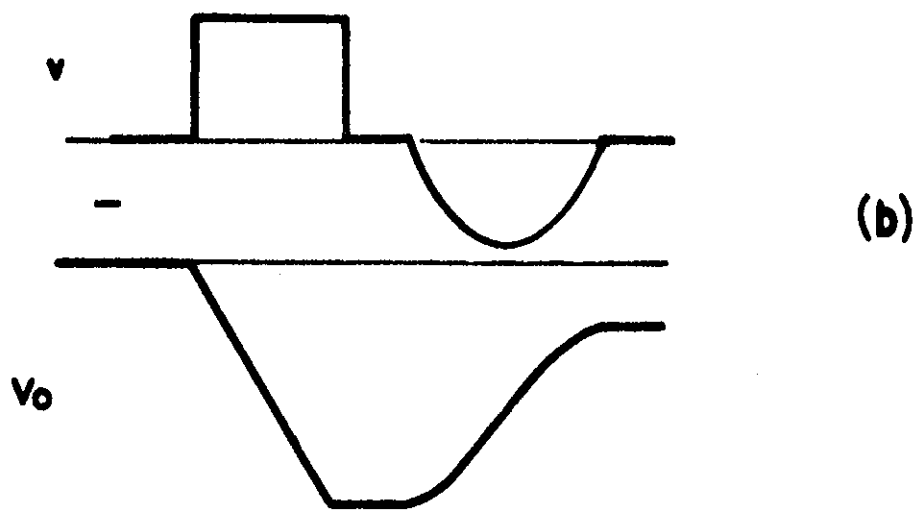


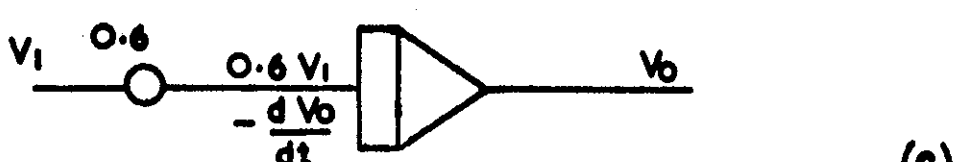
$$V_o = -\left[\cdot 3V_1 + 1\cdot 6V_1 + V_3\right]$$

FIGURE 3. A circuit for addition: (a) Circuit details (b) Potentiometers (c) Circuit using conventional symbols: the whole of the circuit (a) is contained within the triangle



$$V_o = -\frac{1}{RC}\int v\,dt$$





$$V_o = -0{\cdot}6\int V_i\,dt$$

FIGURE 4. A circuit for integration: (a) Circuit details (b) Typical waveforms (c) Circuit using conventional symbols: the whole of circuit (a) is contained within the special triangular symbol. More than one input circuit is permitted

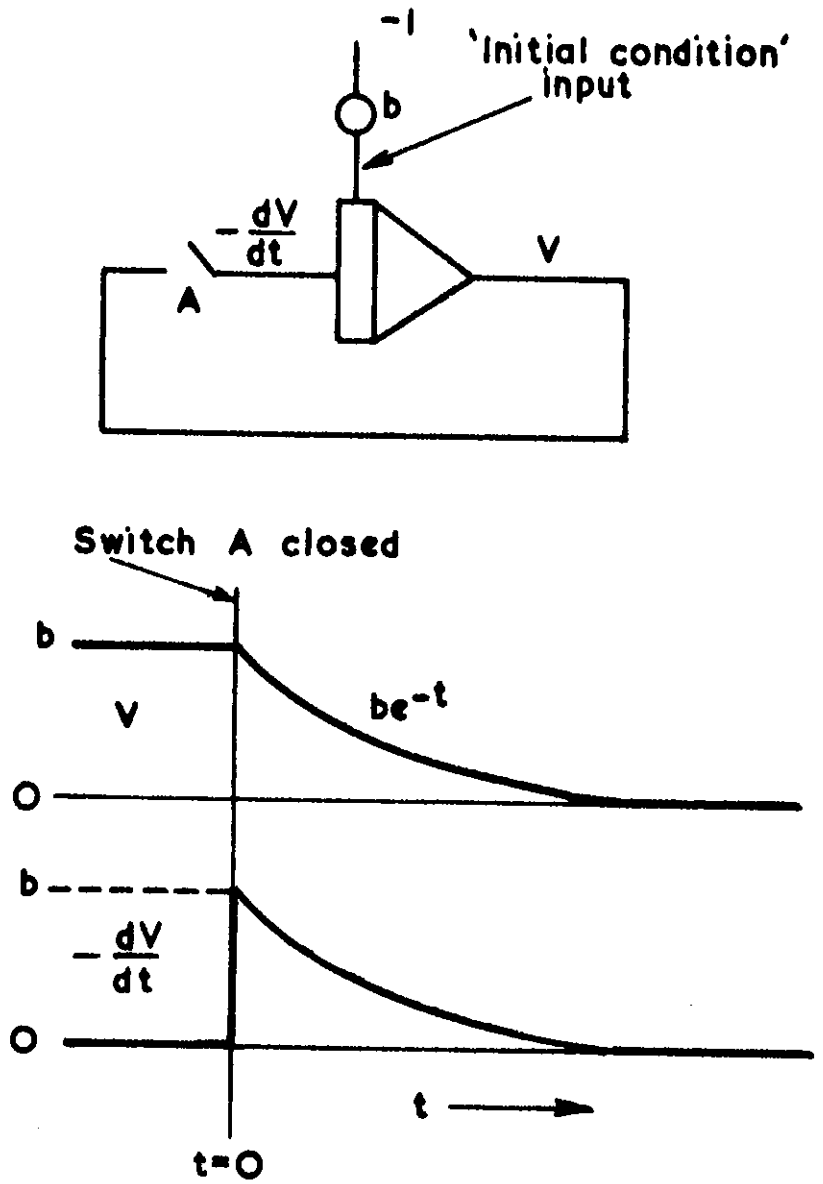


FIGURE 5. A circuit for the solution of $\frac{dV}{dt} = -V$ and typical waveforms. The input via b only provides the starting voltage

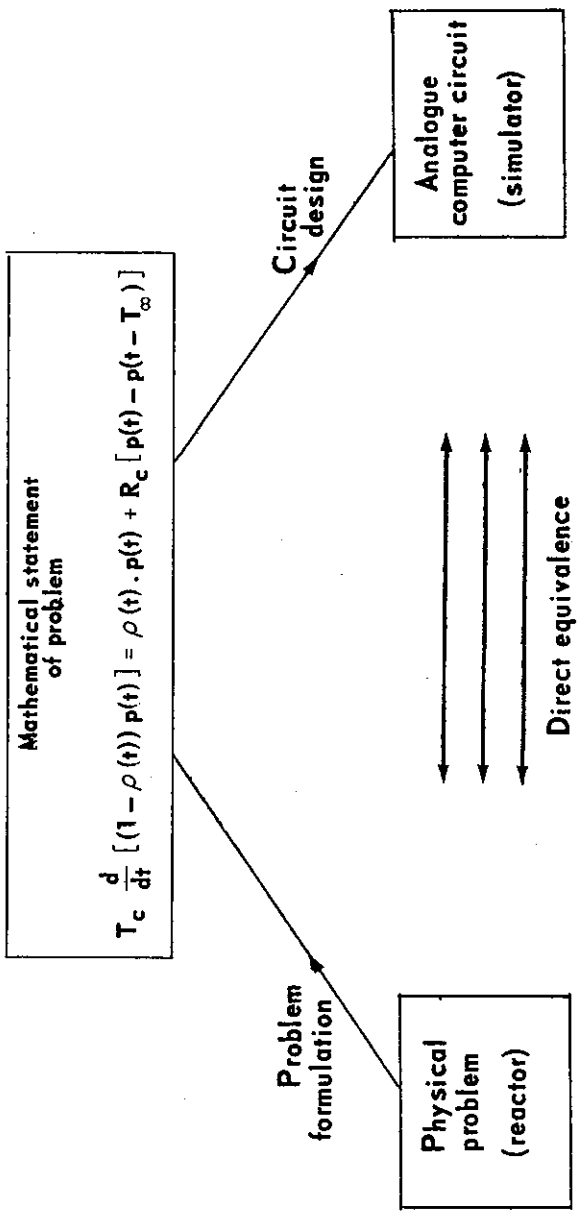


FIGURE 6. Pictorial representation of the analogue method of problem solving

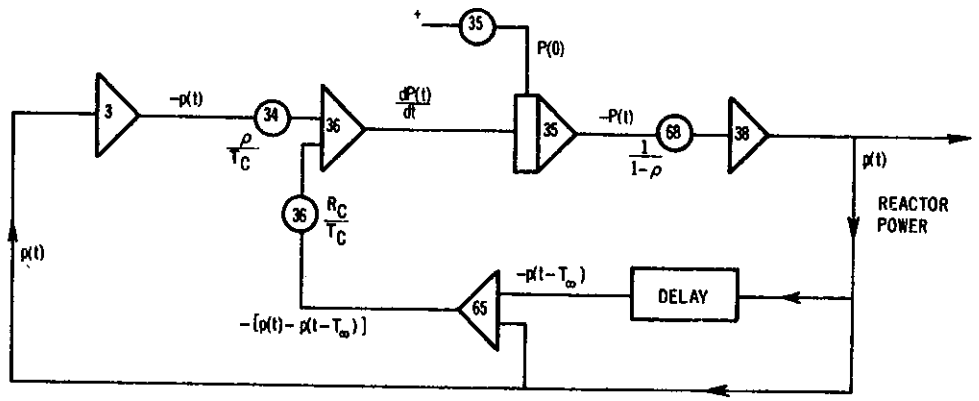


FIGURE 7(a) A CIRCUIT FOR THE SOLUTION OF EQUATION 3 WITH CONSTANT VALUES FOR ρ. REMEMBER THAT EACH AMPLIFIER INVERTS THE SIGNAL AS WELL AS CARRYING OUT ITS EXPECTED FUNCTION.

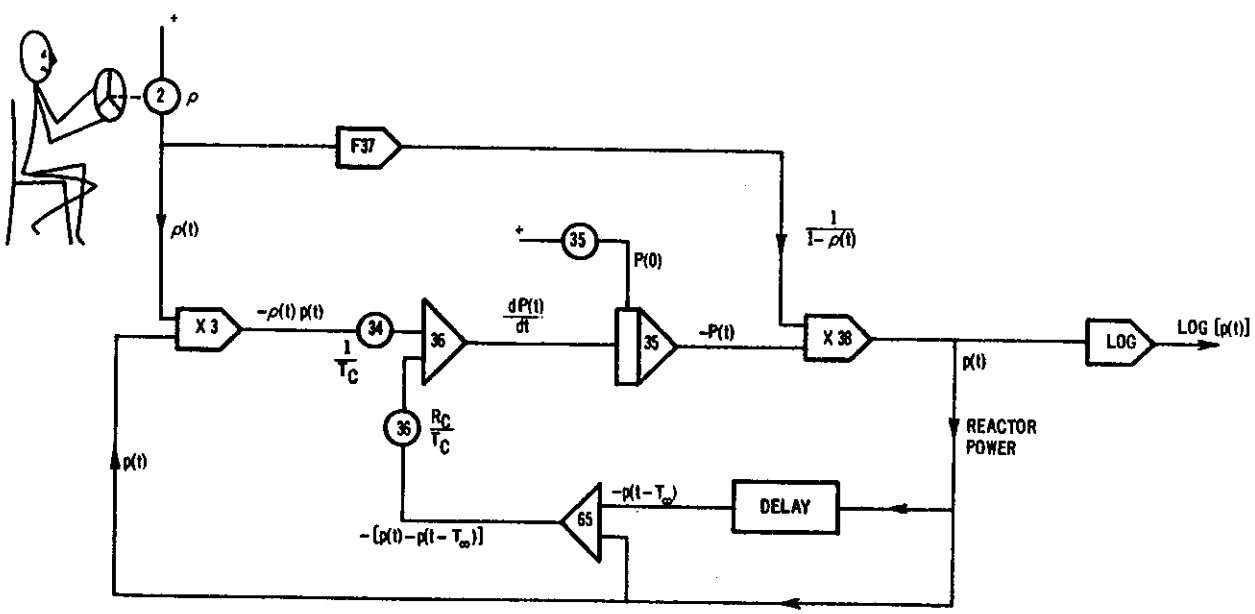


FIGURE 7(b) A CIRCUIT, DERIVED FROM THAT OF FIGURE 7(a) FOR THE SOLUTION OF EQUATION 3 WITH VARYING VALUES OF ρ. THIS CIRCUIT IS A TRUE SIMULATOR, WITH THE OPERATOR VARYING THE REACTIVITY $\rho(t)$ USING POTENTIOMETER 2.

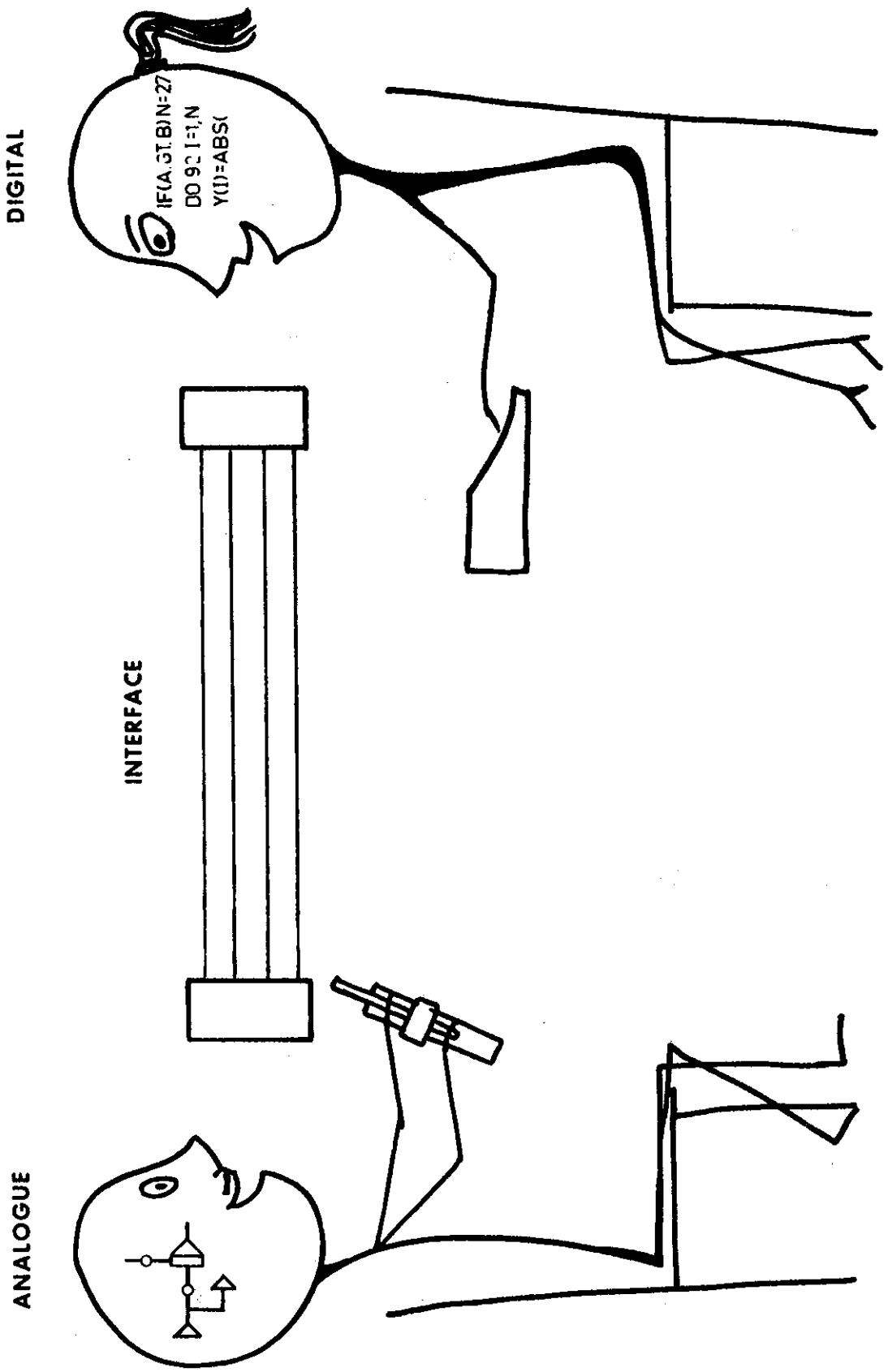


FIGURE 8. A hybrid computer

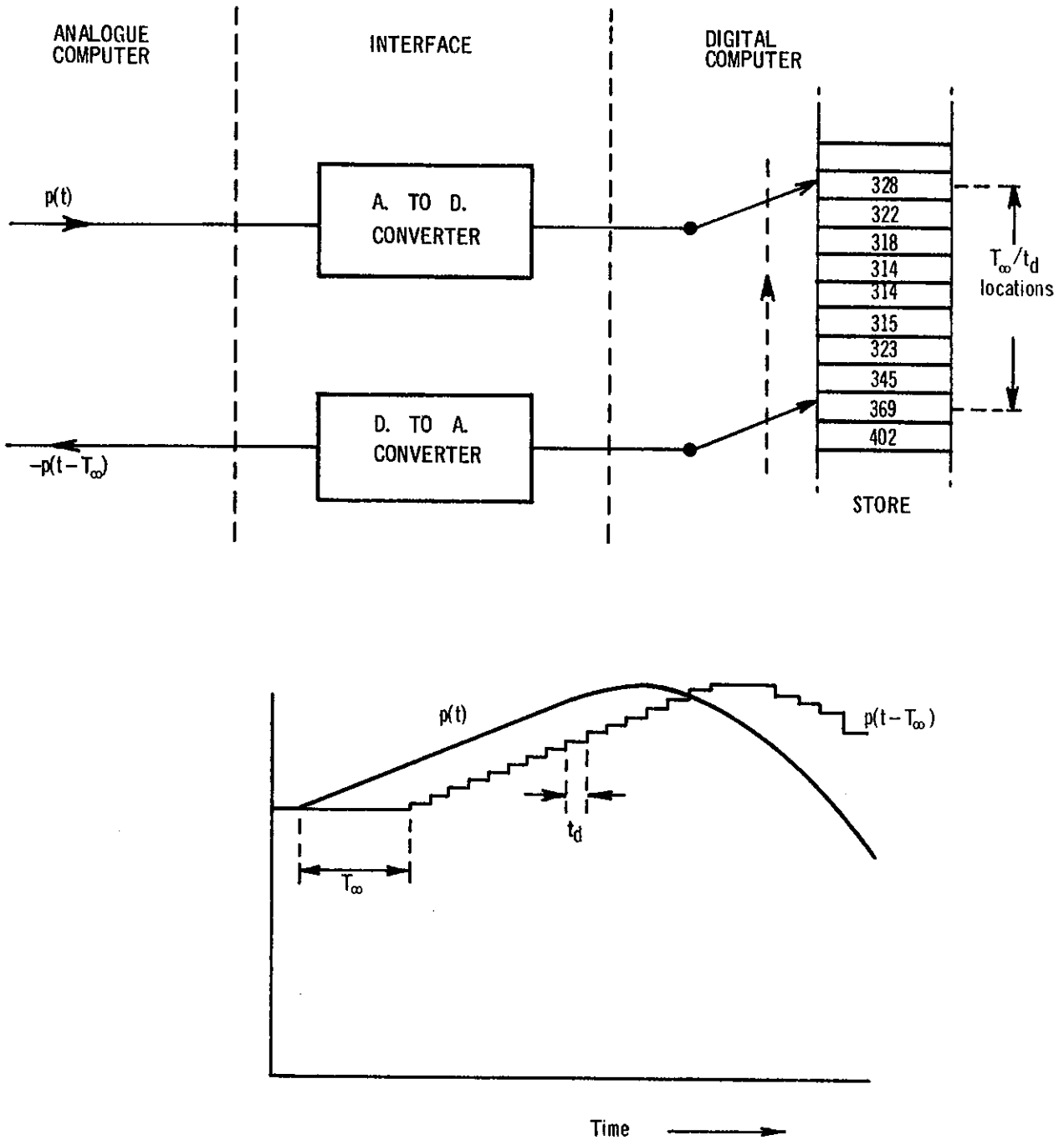


FIGURE 9 A DELAY, PROVIDED BY THE DIGITAL COMPUTER, AND INTRODUCED INTO THE ANALOGUE COMPUTER VIA THE INTERFACE. FOR REASONABLE ACCURACY, $t_d \ll T_\infty$.