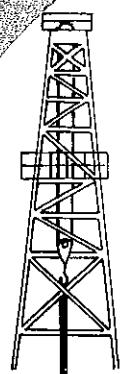
AAEC/S25

Australian Atomic Energy Commission
Lucas Heights Research Laboratories



SUMMER SCHOOL 1982

MATHEMATICAL MODELLING OF A LIMITED RESOURCE

Edited by J.P. Pollard

AUSTRALIAN ATOMIC ENERGY COMMISSION
RESEARCH ESTABLISHMENT
LUCAS HEIGHTS

SUMMER SCHOOL 1982

MATHEMATICAL MODELLING OF A LIMITED RESOURCE

Edited by
J.P. POLLARD

ABSTRACT

These notes are for a Summer School which will introduce mathematically minded year-12 High School students to aspects of resource modelling. The course considers the application of

(i) mathematics of exponential processes, and

(ii) numerical solution of differential equations

to a currently important question - 'When will the world production of crude oil begin to diminish?'

A considerable portion of the Summer School course is devoted to electronic computing, using

- large scientific digital computers,
- mini-computers
- micro-computers, and
- hybrid, analogue-to-digital computers;

these are applied to the resource modelling problem and to some simulation processes.

CONTENTS

National Library of Australia card number and ISBN 0 642 59755 3

Division. She is primarily involved in the computational solution of reactor physics problems.

3. JOHN POLLARD, Dip.Appl.Chem., M.Sc., Ph.D., MACS, is leader of the Scientific Computing Section within Applied Mathematics & Computing Division and his research activities include mathematical problem solving with computers. John is also involved with the use of microcomputers for educational purposes.

4. PAUL MISKELLY, B.E., M.Eng.Sci., is a member of Applied Physics Division and is involved with systems programming on hybrid computing facilities. Paul is also interested in the modelling and solution of time-dependent problems.

5. BRIAN McGREGOR, B.Sc.(Hons), is a member of Nuclear Technology Division. His research interests include reactor shielding and applications involving similar methods.

6. KEN MAHER, B.Sc.(Hons), M.Sc., Ph.D., MAIE, is a member of the Energy Systems Analysis Group (now part of CSIRO Division of Energy Technology) and is involved in building a large model of the Australian energy system.

8. TREVOR GOLLAN, is a member of Applied Mathematics & Computing Division and is involved with systems programming and computer operations.

9. EDITH ROSE, B.Sc., is a member of Applied Physics Division and is involved in the solution of theoretical problems with computational techniques.

10. JERARD BARRY, M.Sc., Ph.D., is a member of Applied Mathematics & Computing Division and is interested in scientific problem solving with numerical mathematics.

11. MELANIA MOORE, is a member of Applied Mathematics & Computing Division and is involved in computer operations.

12. ERIC CLAYTON, M.Sc., is a member of Applied Physics Division and undertakes surface analysis by nuclear techniques.

Staff not in photograph

MALCOLM DAVIDSON, B.Sc.(Hons), Ph.D., is a member of Applied Mathematics & Computing Division. He is particularly interested in obtaining a mathematical understanding of air flow in lungs, and is studying blood flow in major arteries.

PHIL GILBERT, M.Sc.(Elect.Eng.), MIEE, is a member of Applied Physics Division. Phil solves problems in dynamics with a hybrid computer, and is also interested in methods of risk analysis and its application to nuclear power stations.

STEPHEN WONG, B.Sc.(Hons), is a member of Applied Mathematics & Computing Division and is involved with the design, development and implementation of computer application programs.

BOB McLAUGHLIN, B.A., Dip.Ed., is a member of Applied Mathematics & Computing Division and is involved with systems programming and computer applications.

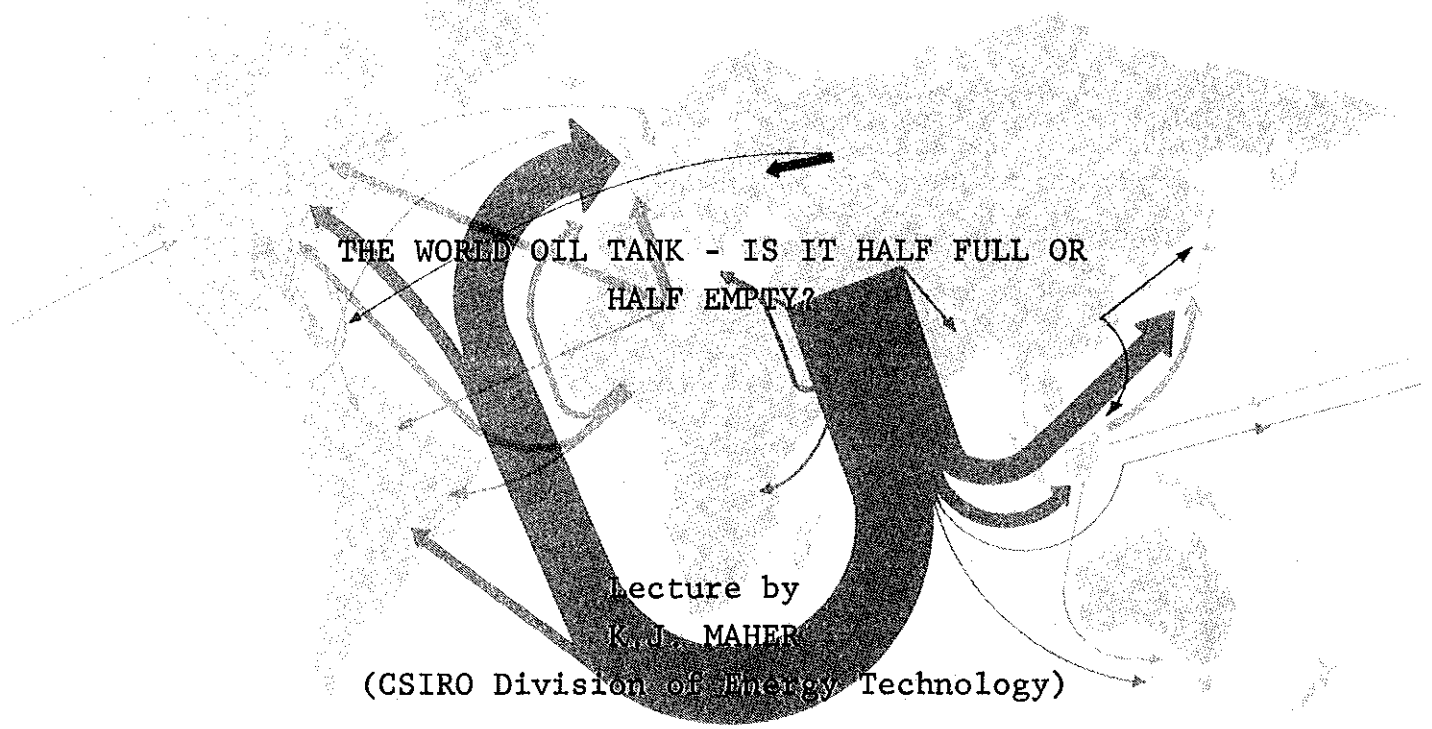
PETER ESSAM, B.Eng., M.Eng.Sci., Mem.ASME, MAIE, is the leader, Energy Systems Analysis Group, Power and Energy Unit, supervising the building of a large model of the Australian energy system.

JAGODA CERGOVSKA, B.Math.(Hons), is a member of Applied Mathematics & Computing Division and is involved with the AAEC Pascal 8000 compiler as well as aspects of data base design.

CAROL FAIRWEATHER is a member of ADP Section and is involved in key punching and key to disk operations.

Main Oil Movements by Sea 1979

CHAPTER 1



THE WORLD OIL TANK - IS IT HALF FULL OR HALF EMPTY?

Lecture by
K.J. MAHER
(CSIRO Division of Energy Technology)

CONTENTS

1.1 INTRODUCTION

We began this Summer School with the British Petroleum Company's film *Energy in Perspective*. We hoped thereby to achieve two objectives. The first was to give you a quick but effective overview of the different kinds of energy available to man and to explain why energy is so important in an industrial society. The second objective was to take you rapidly to the heart of the problem - to confront you with the central issue of what is often called the 'energy crisis'.

The central issue is the availability of oil. As we shall see, if one takes into account natural gas, coal and uranium as well as oil, then the world's energy resources are plentiful over any realistic forward planning period. Therefore the catch phrase 'energy crisis' is a bit misleading. I would prefer to use the phrase 'oil problem'. The crisis, if it cannot be prevented, will come to the world in the form of a catastrophic cut-off of the supply of Middle Eastern oil. There will then follow collapse of the transport systems of the industrialised nations, economic depression, mass hunger and, possibly, oil war. Solutions to the oil problem will prevent such a crisis.

An important question is how much time there is left for the world to switch away from its vulnerable oil supply in favour of other energy forms. We will try to answer this question at the Summer School using mathematics and computers.

What caused the oil problem? For decades the world energy industry has been turning away from its most abundant fossil resource, coal, and has steadily increased its use of a much more limited quantity of energy capital, oil. Why has so much dependence been placed on what now appears to be an uncertain supply?

There were good reasons for doing so. In many situations oil delivered energy to the factory for process heat, to homes for space heat and to electricity generating stations, more cheaply than coal. Where coal was cheaper, it was still often rejected because oil was easier to store and, except for high sulphur varieties, oil burned cleaner and produced less harmful environmental effects. Only natural gas could hold its own as an alternative energy form.

Oil is 'par excellence' *the* transport fuel. It emerges from refineries as motor spirit, diesel fuel and jet fuel. The ready availability of these fluids at low prices fostered an explosive growth, after the Second World War, of the movement of people and goods.

1.2 THE OIL RESOURCE

The energy planners of the 1950s and the 1960s who watched the flight to cheap oil were well aware of the relative abundances of the various fossil fuels. In terms of the broad relativities, the energy reserves picture (figure 1.1) was the same then as it is now.

In figure 1.1 the relative sizes of the blocks are in proportion to the reserves of the various fossil fuels as known in 1981. We use the word 'reserves' to describe the amount of a mineral that is definitely known to exist and which can be economically extracted. This quantity is known because exploratory drilling has proved the size and quality of the mineral deposit. From past experience of finding reserves and from the ever-increasing fund of knowledge of geological structures, it is sometimes possible to make fairly reasonable estimates of another quantity called 'resources'. The resources quantity is the estimated value of the *total* quantity of the mineral originally available in an economically extractable form.

Thus, at any time,

Resource = Amount already discovered and used
+ amount already discovered and still available (reserves)
+ the undiscovered remainder.

Figure 1.2 shows the estimated relative abundances of coal, oil and natural gas resources.

The estimate of the oil resource includes a speculative component - the undiscovered remainder. Therefore it too is speculative. However, over the past fifteen years or so there has emerged something of a consensus that the oil resource created between one million and 500 million years ago is about 2 million million barrels of oil. In fact, that is the value mentioned in the film *'Energy in Perspective'*.

Now the volume of a barrel is about 160 litres. We will work in this unit throughout the Summer School in blatant defiance of the system of metric measurements! The barrel is the volume unit most favoured within the oil industry, probably in nostalgic deference to those pioneering days when oil was carried around the world in barrels, even on little fire traps of ships.

To give you a grasp of the energy content of a barrel of crude oil, I will provide you with a few illustrations. If it could be entirely converted to motor spirit (it can't) it would take a medium size, six-cylinder car (consuming 14 litres/100 km) from Sydney to Brisbane (1100

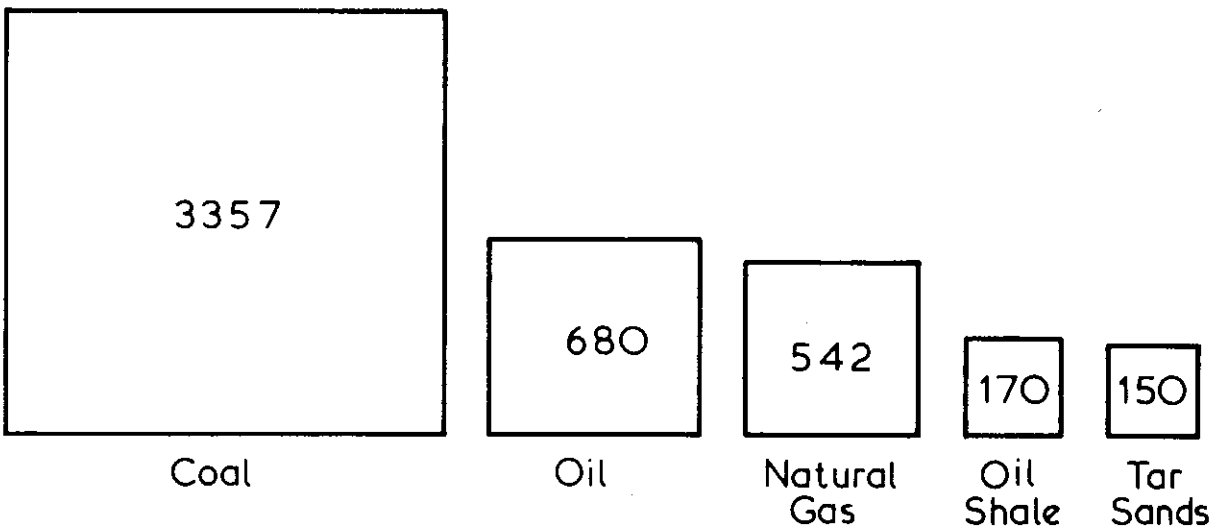


Figure 1.1 World fossil fuel reserves 1981 (10^9 barrels oil equivalent)

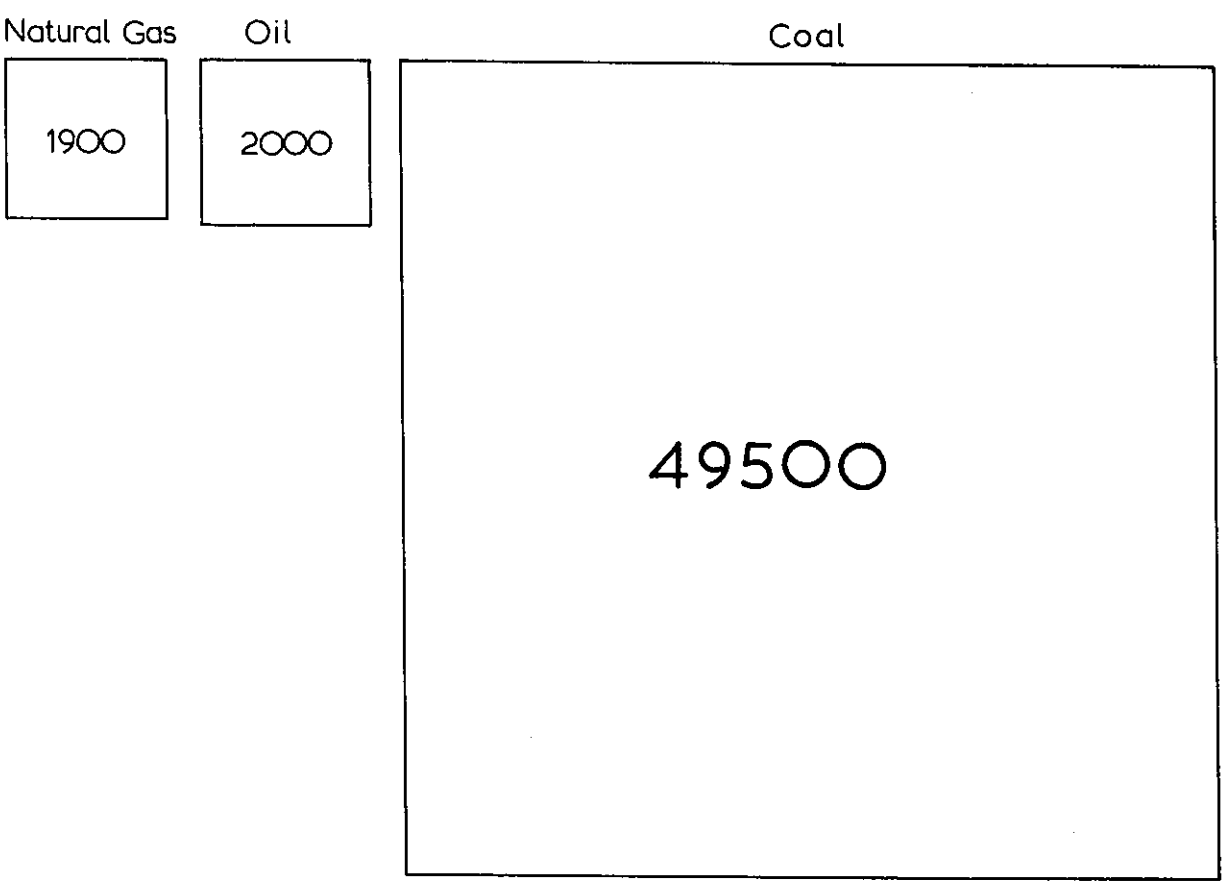


Figure 1.2 World fossil fuel resources (10^9 barrels oil equivalent)

km). If converted into home heating oil, it would warm a Sydney home with an average size oil fire and typical usage pattern for about three winter weeks.

So two million million barrels is a great deal of energy. It is also a cumbersome way of saying things. We will from now on use scientific notation. The oil resource, which is all the naturally occurring and economically recoverable oil available, for all time (*i.e.* time infinity) we express as

$$x_\infty = 2 \times 10^{12} \text{ barrels.}$$

Perhaps we can redeem ourselves a little in the eyes of the metric purists by using the 'giga' prefix for 10^9 and write

$$x_\infty = 2000 \text{ gigabarrels (sometimes written Gbbls).}$$

For reference: in 1981 Australia consumed 0.22 gigabarrels, the United States 5.6 gigabarrels and the world 21.8 gigabarrels.

1.3 THE OIL CONSUMPTION FUNCTION

If you accept that the world oil tank is finite and was originally filled to the brim with x_∞ = 2000 gigabarrels, then several urgent questions should come to mind. For example, how much of the original x_∞ has been consumed by the end of 1982? This amount we shall write as x(1982). In general, the amount used up to time t is x(t). We shall measure time in years.

In figure 1.3, historical values for the cumulative consumption function x(t) are plotted for the time range 1910 to 1982. At first sight the figure is reassuring. Less than a quarter of the world oil tank has been used up in just over 100 years of consumption (1880 to 1982).

Look at the x(t) data more closely. How much was consumed in the past decade? Surprisingly,

$$x(1980) - x(1970) \cong 210 \text{ gigabarrels,}$$

or about 43% of total consumption to date! Clearly, if the consumption rate was to increase beyond the values recorded in the 1970s, the oil resource would be rapidly exhausted. The x(t) curve in the time range after 1982 (*i.e.* the future) would climb steeply towards x_∞ and final exhaustion.

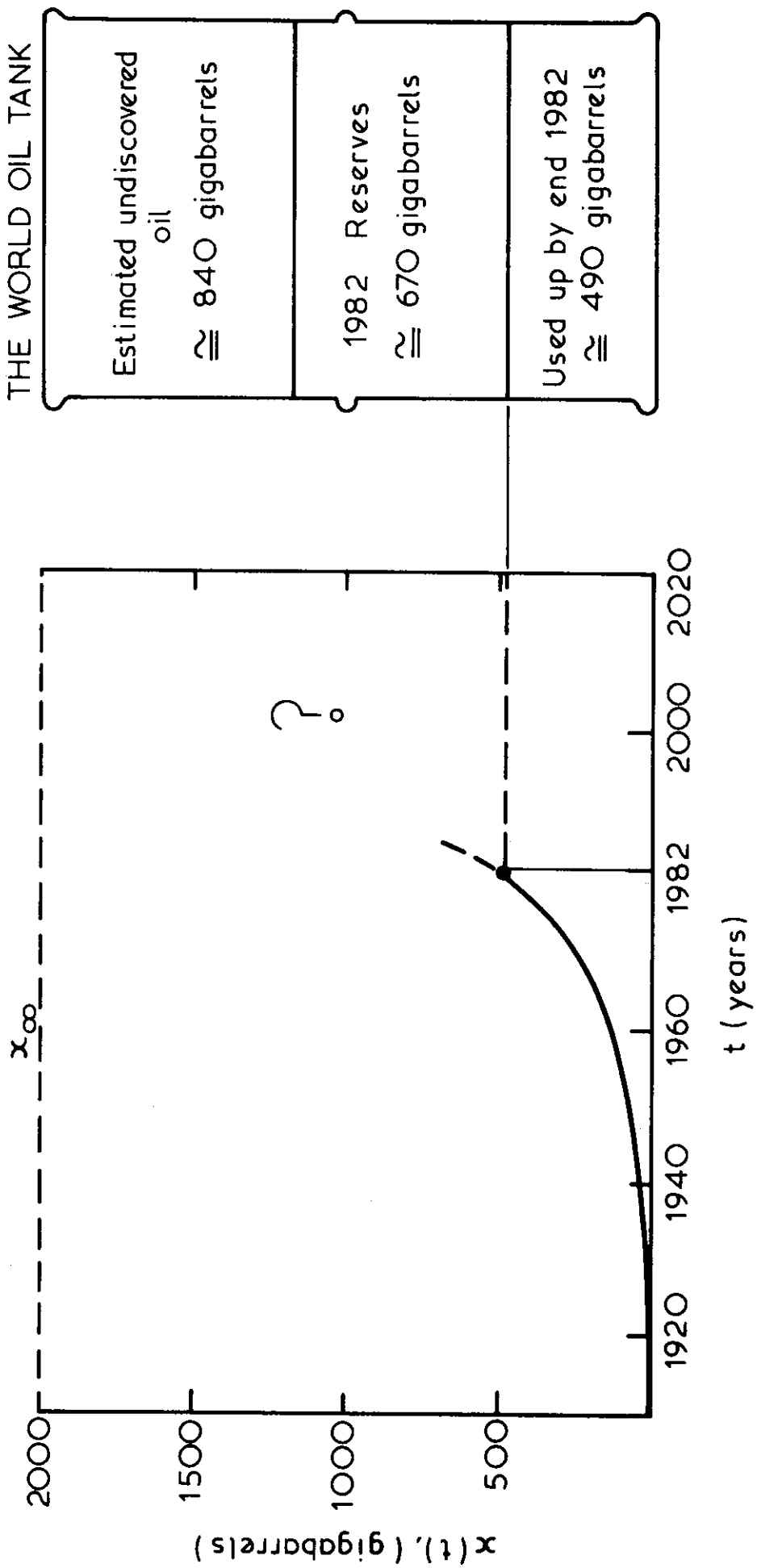


Figure 1.3 Cumulative world oil consumption, historical values

I have introduced a new concept at this point, the consumption rate. The rate is related to the cumulative consumption, x(t), by a simple expression. The average consumption rate over some time period Δt is

$$c = \frac{\text{amount consumed in period } \Delta t}{\Delta t} = \frac{x(t + \Delta t) - x(t)}{\Delta t} \qquad (1.1)$$

and

$$\lim_{\Delta t \to 0} c(t) = \frac{dx}{dt} \qquad (1.2)$$

When $\Delta t = 1$, c is the annual consumption rate.

Until fairly recently, the annual consumption rate has been climbing steadily. In 1970, it was 16.4 gigabarrels/year, in 1975 20.3 gigabarrels/year and in 1979 23.4 gigabarrels/year. However, in the last two years the annual consumption rate has staggered under the double impact of high oil prices and world recession and in 1981 it had fallen to 21.8 gigabarrels/year.

Regardless of whether the consumption rate, c(t), rises or falls, the cumulative consumption, x(t), *always* rises, heading relentlessly towards x_∞ and the total exhaustion of the world's oil. The only uncertainty is how quickly x(t) will approach x_∞ in future years. In figure 1.3, the behaviour of the cumulative consumption curve is covered by a giant question mark. No one can foretell the future. In this Summer School we shall make some *hypotheses* about the future behaviour of x(t) and incorporate these into a differential equation. This equation will be our *model* of oil resource depletion.

The word 'model' implies an abstraction. Reality in this case consists of a myriad of decisions made over time by individual oil consumers, oil producers, governments and other political groupings of people - decisions made for a variety of economic, political and, sometimes, irrational reasons. We can hope, therefore, to capture only the broad features of this collective human behaviour in our model. We cannot expect the precision that often characterises models of physical systems (for example, the equations governing the dynamics of a pendulum). The models of physics and chemistry can be subjected to the repeated tests of experiments, whereas our oil depletion model can ultimately only be tested by the unfolding of time. We should, however, ensure that the model is not contradicted by the 'experimental' data that are available. That is to say, it should provide an adequate description of the past behaviour of x(t).

1.4 THE EXPONENTIAL GROWTH MODEL

From the end of the Second World War until 1973, $x(t)$ evolved in accordance with a very simple rule-of-thumb: the value of $x(t)$ was simply 7 per cent larger than the value $x(t - 1)$ of the previous year. (There were, of course, occasional fluctuations of a few percentage points on either side of this average 7 per cent annual growth rate.) In fact, this simple growth rule is adequate for the description of the evolution of the $x(t)$ curve over the entire period 1910 to 1973 with the exception of the period of the Great Depression and the first years of the Second World War.

Thus,

$$x(t + 1) = (1 + \lambda)\, x(t) \quad ,$$

where $$\lambda = 0.07 \text{ year}^{-1} \quad .$$

That is,

$$x(t + 1) - x(t) = \lambda\, x(t) \quad .$$

Over Δt, a shorter period of growth than one year, we can write

$$x(t + \Delta t) - x(t) = \Delta t\, \lambda\, x(t) \quad .$$

(This is not strictly true but, because λ is not too big, it isn't far wrong.) Therefore,

$$\frac{x(t + \Delta t) - x(t)}{\Delta t} = \lambda\, x(t)$$

and in $\lim_{\Delta t \to 0}$,

$$\frac{dx}{dt} = \lambda\, x(t) \quad . \tag{1.3}$$

Equation (1.3) is a simple example of a differential equation. In order to solve it we need an *initial condition*. The oil consumption data before 1910 are a bit suspect, but we estimate the cumulative consumption up to the end of 1909 to be 4.29 gigabarrels. Our initial condition is, therefore,

$$x(t_o) = x(1909) = 4.29 \text{ gigabarrels} \quad . \tag{1.4}$$

Equation (1.3), taken with the initial condition equation (1.4), is the first version of our oil depletion model. I will not solve the equation here. It exposes a trove of rather beautiful mathematics worthy of an entire lecture in itself. Dr. Pollard will present that lecture later today.

Simply let me say that equation (1.3) governs a particular form of growth known as 'exponential' growth. Money left to compound in a savings account grows in this manner (at least in nominal terms - its real, inflation-corrected value probably shows negative exponential growth). Many biological systems (*e.g.* populations of bacteria, insects *etc.*) expand exponentially until a constraint emerges such as a predator on the population or the exhaustion of food or living space. Growth then slows!

Constrained systems clearly are governed by a more complicated equation than equation (1.3). There must be the involvement of some kind of time-dependent growth coefficient rather than a growth *constant* such as λ. The time-dependent growth coefficient should tend towards zero as the constraint is encountered.

Clearly, although it describes historical behaviour well enough, the first version of our model (equation 1.3) fails to acknowledge the main physical constraint on the world oil system, the finite value of x_∞. As it stands, our model projects steady exponential growth in cumulative consumption, $x(t)$, (and hence in the consumption rate, dx/dt) right up to the day on which the last drop of x_∞ is exhausted (figure 1.4). We have, in equation (1.3), a 'going out with a bang' model of consumption.

World oil production could not match this kind of consumption pattern. We require consumption to equal production. (We neglect the small departures from the equality that result from the movement of oil in and out of stockpiles.)

Total oil production is the summed output of many individual oil fields. Each field reaches a maximum pumping rate which is sustained for a period before the production rate slides in long decay towards zero. New fields make up for the decline of older fields. As $x(t)$ approaches x_∞ the discovery rate of new fields will fall and most producing fields will be in decline. In due course, the annual world oil production rate, which equals dx/dt, must reach a maximum and then fall towards zero as $x(t) \to x_\infty$. Such behaviour is a result of the physical limitations to production from a finite resource.

We must replace our 'going out with a bang' exponential growth model with something reminiscent of Eliot's vision of the future.*

---

* "This is the way the world ends
This is the way the world ends
This is the way the world ends
Not with a bang but a whimper."
T.S. Eliot - *The Hollow Men*

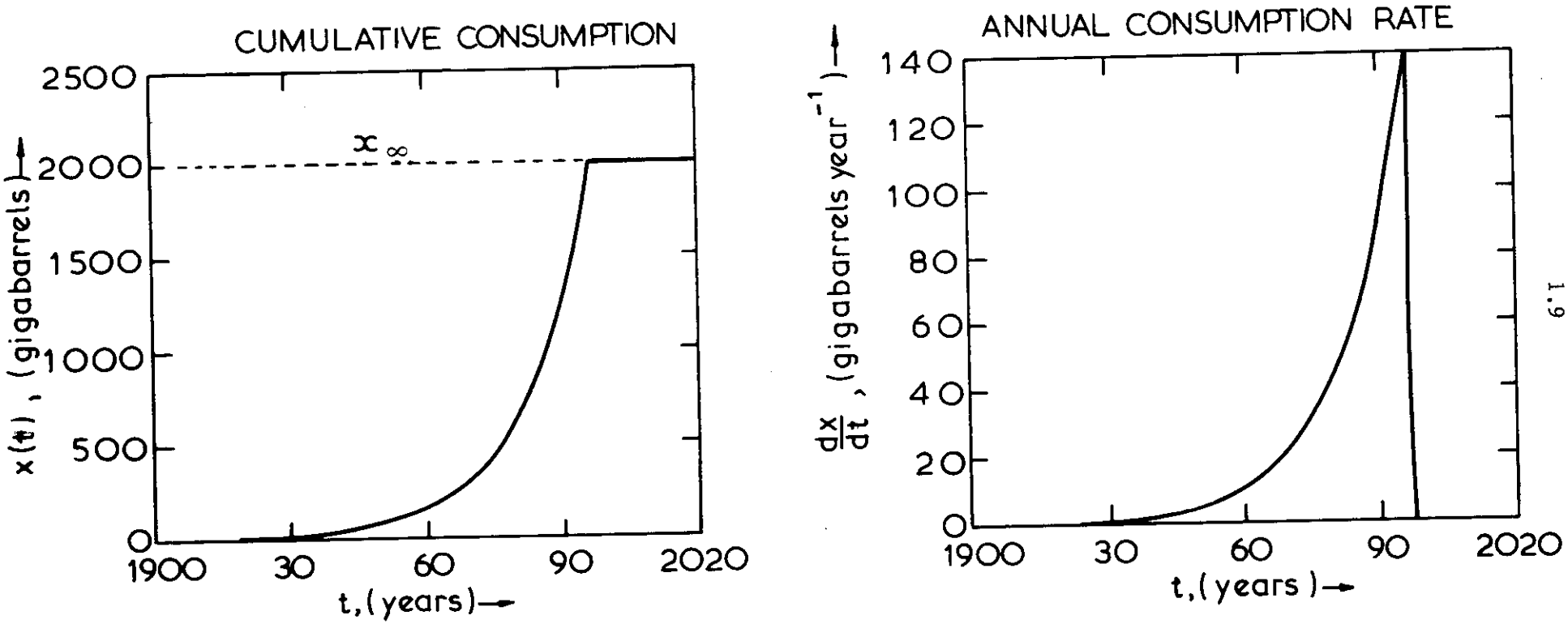


Figure 1.4 Exponential growth of world oil depletion

1.5 THE HUBBERT OR LOGISTIC GROWTH MODEL

We need to introduce into equation (1.3) some kind of time-dependent growth coefficient, $g(t)$, that causes the growth of the cumulative consumption function, $x(t)$, to slow as $x(t)$ approaches the constraining limit, x_∞.

The differential equation then becomes

$$\frac{dx}{dt} = \lambda\, g(t)\, x(t) \quad .$$

We require $g(t) \doteq 1$ for the period $1910 < t < 1970$ (the exponential model worked well enough over this time range); we also require

$$g(t) \to 0 \qquad \text{as} \qquad x(t) \to x_\infty \quad .$$

There is a wide selection of functions $g(t)$ that exhibit these properties. For example, there is the class of functions

$$g_n(t) = 1 - \left(\frac{x(t)}{x_\infty}\right)^n \quad , \quad n = 1,\ 2,\ 3\ \ldots .$$

We will quite arbitrarily select the first member of the class in order to construct the second version of our model:

$$\frac{dx}{dt} = \lambda\left(1 - \frac{x(t)}{x_\infty}\right) x(t) \quad . \qquad (1.5)$$

The initial condition (equation 1.4) is still applicable.

Our new version of the model, equation (1.5), was applied by M.K. Hubbert [1962, 1969] to project possible future paths for $x(t)$ for different estimates of x_∞. The solution of equation (1.5) with initial condition of the form of equation (1.4) has come to be known as the 'Hubbert' curve but sometimes it is called the 'logistic' growth curve. Dr. Clancy will take you through the mathematics for solving this type of equation during the Summer School.

The logistic growth curve for $x_\infty = 2000$ gigabarrels, $\lambda = 0.07$ year^{-1} and the initial condition (equation 1.4), is shown in figure 1.5 together with the associated curve for the annual consumption rate, dx/dt.

Our model now satisfies a few basic criteria. The broad historical behaviour of $x(t)$ is reproduced, but not the fine year-to-year detail. The broad features of finite resource depletion have been captured in that $x(t)$ approaches x_∞ in an asymptotic manner, and dx/dt peaks and then declines smoothly towards zero.

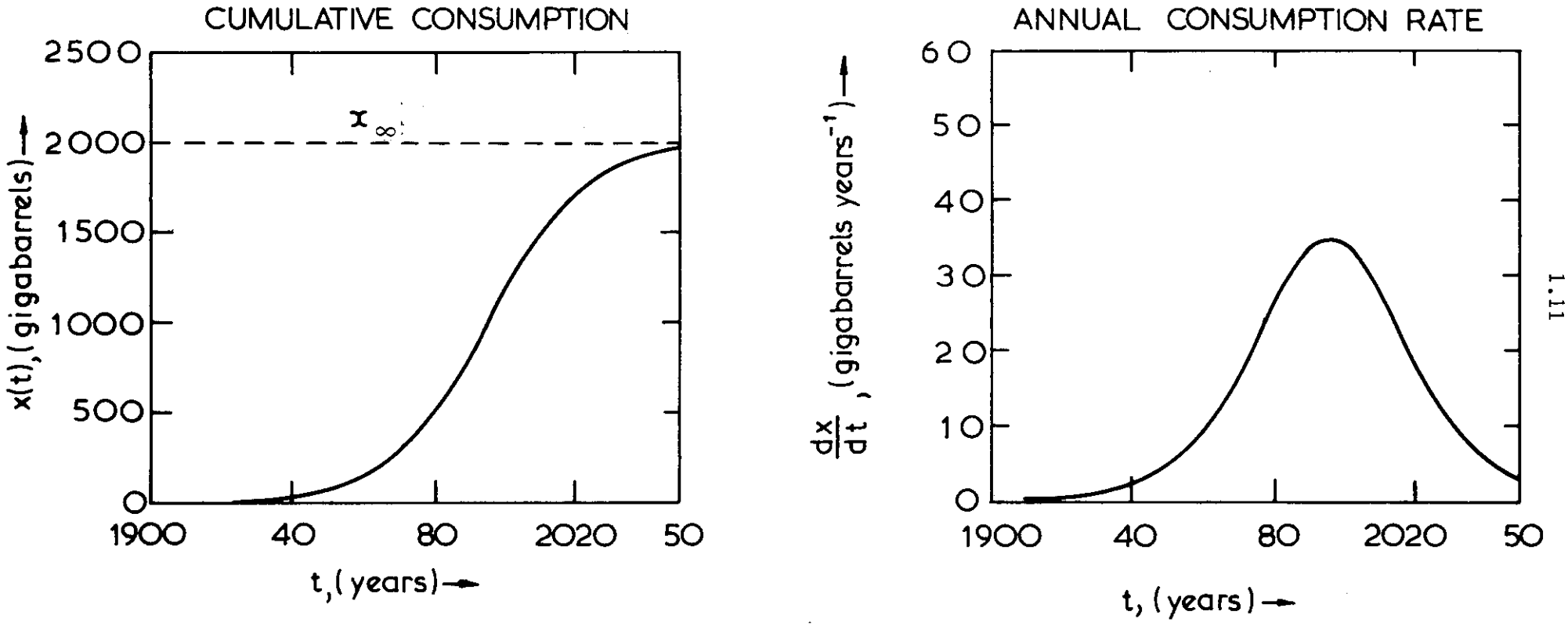


Figure 1.5 Logistic growth model of world oil depletion

1.6 EFFECT OF PRICE ON CONSUMPTION

The logistic growth model is rather mechanistic, depending only on x_∞, λ and $x(t_o)$. There has been, so far, no attempt to include (other than in the growth constant, λ) the behaviour of people, the oil consumers and producers. For example, the model does not take into account a basic economic principle; when the price of a commodity such as oil increases, there is a tendency on the part of consumers to reduce their consumption rate.

You will be aware that the price of oil on the world market has risen in a spectacular manner in recent years. The growth in consumption has been dampened as a result. Many commentators predict further rises in price as the resource is depleted and supply is increasingly limited. We will try to include at least this much economic theory into our model.

Before doing so, we must define our meaning of price. Because of the changing value of money due to inflation, price can be a rather rubbery concept. We should not expect to see an effect on consumption of, say, a doubling over a period in time of the oil price in dollars per barrel if, over the same period, the value of the dollar (as measured by its ability to purchase goods and services) has been halved. We would say then that the real price of oil has not changed over the period.

Economists attempt to overcome this problem by using a deflator index which corrects for changes in the purchasing power of money from year to year. A base year is selected and the deflator index is set to unity for that year. Prices for all years are divided by the appropriate value of the deflator index and are thereby expressed in terms of dollars of the base year.

We shall select 1982 to be our base year. We shall follow the practice of the world oil industry and express oil prices in terms of United States dollars. Our unit of price, therefore, will be constant 1982 US dollars per barrel. The 'official' price set for Middle East oil during 1982 has been about \$34/barrel. Assume that it rises to \$35/barrel in 1983. If, for example, inflation in the United States runs at 6 per cent between 1982 and 1983, then the deflator index for 1983 will be 1.06. The 1983 price of oil, expressed in constant 1982 US dollars will be \$35/1.06 or \$33/barrel.

To build into our model the consumption dampening effect of oil price rises, we need some measure of the responsiveness of the consum-

ption rate to price movements. Economists use a measure called an 'elasticity coefficient', β, for this purpose. It is defined as follows:

$$\beta = \frac{\text{percentage negative change in consumption rate}}{\text{percentage positive change in constant dollar price}} ,$$

all other influential variables being held fixed.

That is,

$$\beta = \frac{-\Delta c/c}{\Delta p/p} ,$$

where p is the price.

$$\text{In} \lim_{\Delta p \to 0} , \qquad \beta = -\frac{p}{c}\frac{dc}{dp} . \tag{1.6}$$

If β is a constant, then it is not difficult to show that a relationship between c and p that satisfies (1.6) is

$$c = \frac{\alpha}{p^{\beta}} ,$$

where α is constant when all parameters other than price and consumption rate are held fixed. (You may like to prove this as an exercise in calculus.) Or, in the notation of our model,

$$\frac{dx}{dt} = \frac{\alpha}{p^{\beta}} . \tag{1.7}$$

1.7 THE PRICE-DEPENDENT MODEL

We now combine equation (1.7) with the earlier version of the model (equation 1.5) to derive a new version of the model which includes price effects:

$$\frac{dx}{dt} = \lambda \left(1 - \frac{x(t)}{x_{\infty}}\right) x(t) \left(\frac{p_o}{p(t)}\right)^{\beta} , \tag{1.8}$$

where the normalisation factor

$$p_o = p(t_o)$$

is the price of oil in the year 1909 (estimated to be \$6.11/barrel in constant 1982 US dollars).

In figure 1.6, p(t) is plotted for the years 1910 to 1982. Using these data, Dr. Pollard has determined that the model (equation 1.8) gives a close approximation to the historical x(t) curve if a value of 0.07 is assumed for the price elasticity coefficient, β. (It is a

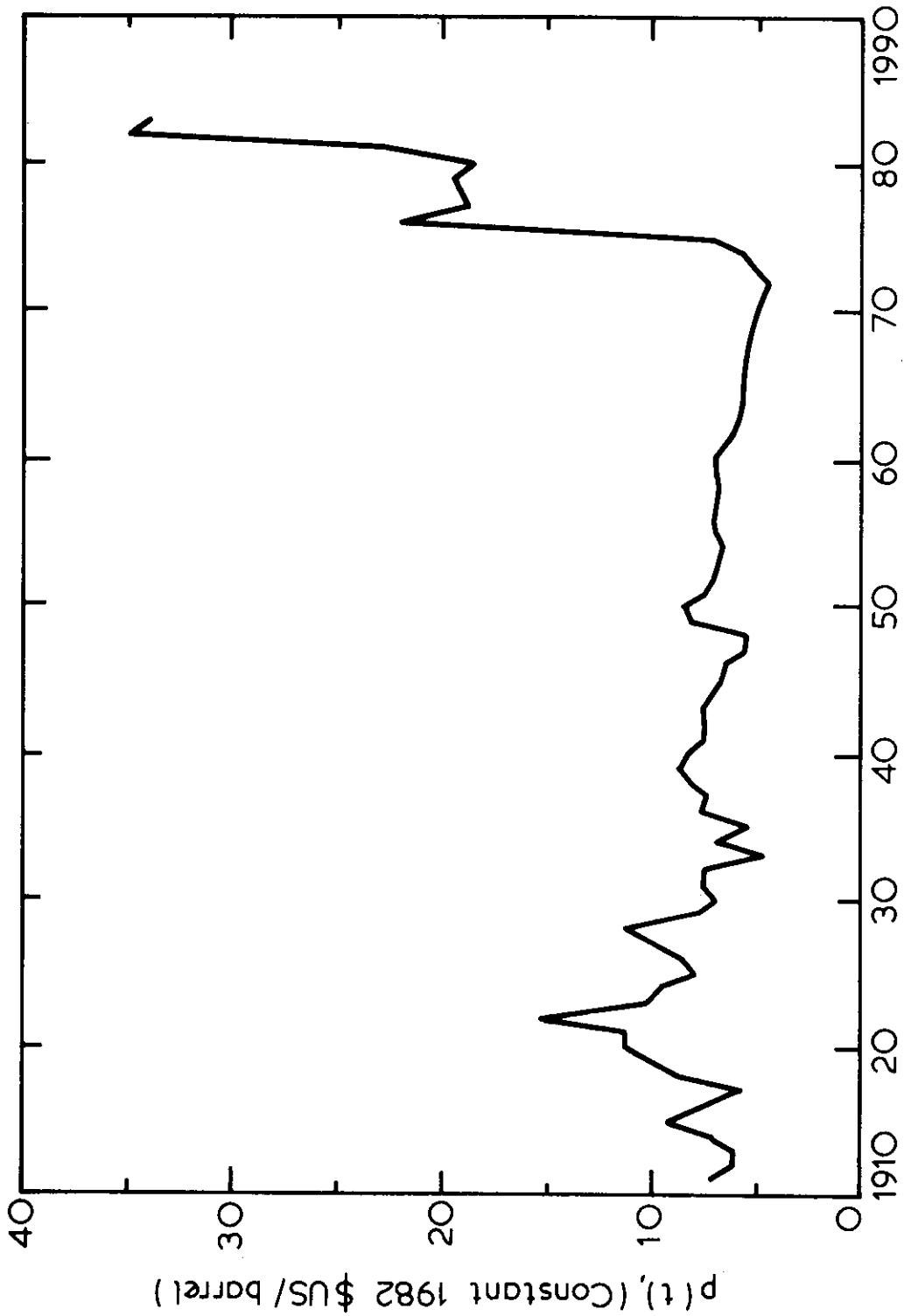


Figure 1.6 The real price of oil, p(t) (1982 $US/barrel)

coincidence that this is the same value as that for λ.)

Although price fluctuated between \$6 to \$15 in the period 1910 to 1930, the period 1930 to 1973 was notable for price stability within a narrow range of \$4.50 to \$9 per barrel. With price confined to this range, the term

$$\left(\frac{p_o}{p(t)}\right)^{0.07}$$

varies only slightly from unity between the values 0.985 and 1.033. Therefore, the price-dependent model operates with a price term close to unity for the years up to 1973. That is, up to 1973 it effectively behaves like the price-independent logistic growth model (equation 1.5).

We might have expected that stable price behaviour would have continued through the 1970s and possibly for another decade beyond. After all, in 1973 considerably less than a quarter of the world oil tank had been consumed (figure 1.3). It would not have been unreasonable in that year to assume that large oil price rises would be deferred until the 1990s. Only then, in the final years of the century, would we expect the required shift from optimism to pessimism - the perception that the world oil tank was not half full but, instead, half empty. Only then would come the steady rise in oil price to reflect its scarcity value.

Instead, in late 1973 and early 1974 the price of oil in constant 1982 US dollars jumped from \$7 to \$22. What happened? Entire books have been written about the events up to that traumatic northern winter of 1973-74 but the following sketchy summary must suffice.

At the end of the 1960s, after 80 years of steady growth, the United States oil production rate faltered and began to fall. The consumption rate of the world's largest economy continued to rise and the US joined Western Europe and Japan as a major importer of oil, critically dependent on supplies from the Persian Gulf, Africa, Venezuela and Indonesia. Political and economic control over the world oil market began to shift significantly away from the western, industrialised oil importing countries towards a group of 13 less developed oil exporting countries. That group* is known as the Organisation of Petroleum Exporting Countries (OPEC) and it controls over 65 per cent of the present world oil reserves.

---

* Saudi Arabia, Iraq, Venezuela, Iran, Kuwait, Qatar, Indonesia, Libya, Abu Dhabi (now United Arab Emirates), Algeria, Nigeria, Ecuador, Gabon.

OPEC was formed in 1960 to provide both a mechanism for consultation between the oil exporting countries and a common front in price negotiations with the major oil companies. The major oil companies not only had power over the transportation and distribution of oil, but also usually operated the production facilities of the large oil fields in OPEC countries. OPEC was formed to counterbalance the companies' power over production levels and prices.

However, it took until 1971 for OPEC to develop real bargaining muscle. The Teheran Agreement of that year provided for a rise in oil prices of 50 per cent over a five-year period. In 1973, the world economy and the oil consumption rate reached overheated levels. World oil production was bumping up against the maximum capacity of developed oil fields. In October of 1973, the Yom Kippur war in the Middle East set fire to the tinder. The Arab members of OPEC denied oil for several months to the United States and the Netherlands for their support of Israel in the war. Although the oil embargo leaked, the confusion on an already overstretched world oil market was sufficient to force the price up by a factor of four.

OPEC was successful during the period 1974 to 1978 in the face of subdued oil demand in containing the price 'slippage' from this new high level. This confounded the confident predictions of many commentators that economic forces would destroy the unity of the oil cartel and prices would tumble as OPEC members tried to undercut each other for a larger share of the market.

In 1979, the unity of OPEC was fractured in an unexpected way. Oil exports from the second largest exporter in the group, Iran, ceased for the first quarter of the year during revolutionary upheaval and resumed at a much lower level than before. A wild scramble for advantage amongst OPEC members led to the second major jump in price from $18.50/barrel in 1978 to about $35/barrel in 1980. Now we are in another period of price 'slippage'.

Both large jumps in price have reduced the consumption rate of world oil, an effect we have tried to model by virtue of the price term in equation (1.8). The term describes the reductions made by consumers and industries when the price rises. It does not estimate the reductions in oil use that occur when an economy goes into recession. Industries then produce below normal levels and, as a result, use less fuel oil. Some consumers lose their jobs and are unable to run cars. Other consumers have less income to spend on petrol. Such 'income' effects are not included in our model. It is a significant omission.

1.8 THE TASK AHEAD

As it stands, equation (1.8) provides sufficient opportunity for you to explore a number of interesting hypothetical variations on the future pattern of world oil consumption. Variation will be generated by your selection from a number of alternative versions of the price function p(t) which we shall label with an index, M.

The first version, M = 0, has p(t) uniformly set to the value p_o. That is, the price term is always unity and the model reduces to the logistic growth equation (1.5). This version will be used to test the computer program.

The second version, M = 1, has p(t) set to a simple mathematical function as indicated in Appendix 1A. Again an analytic form of solution is possible (as demonstrated by Dr. Clancy in Appendix 4A). This version will also be used to test the computer program, particularly the price dependence.

The third version, M = 2, follows the reality of the traumatic price jumps of 1974 and 1980 and then projects a future price function rising smoothly from the high 1982 base of $34/barrel (see Appendix 1A).

The fourth version, M = 3, is rather gloomy, for it assumes some kind of major traumatic disruption to the world oil market in 1990 when the price is assumed to again savagely lurch upwards.

Finally, you may wish to consider the consequences of a hypothetical discovery that the oil resource has been underestimated, by running the M = 2 case with x_∞ = 3000 gigabarrels instead of 2000 gigabarrels.

For easy reference, the model and its data are summarised in Appendix 1B.

1.9 ACKNOWLEDGEMENTS

The author gratefully acknowledges the assistance of Mr. A. Musgrove in the initial preparation of oil price data and Miss Y. Hargreaves and Mr. P. Parker in the presentation of diagrams. Dr. J. Pollard translated a few general comments by the author on prices and the Hubbert model into the concrete form of equation (1.8). The careful tailoring of the mathematics to stretch but not outreach the capabilities of our students is entirely due to his considerable teaching skill.

1.10 REFERENCES

Hubbert, M.K. (1962) - U.S. National Academy of Sciences, National Research Council, Publication 1000-0.

Hubbert, M.K. (1969) - Energy resources, *In Resources and Man,* National Academy of Sciences, National Research Council, W.H. Freeman, San Francisco.

APPENDIX 1A
THE PRICING EXTRAPOLATION PROCESS

Figure 1.6 gives the constant 1982 US dollars price of oil from historical data up to the year 1982. However, our oil model requires an estimate of the price all the way to the year 2050!

We resort to a possibly plausible process for extrapolation based on the following assumptions...

(a) as oil is consumed (as x(t) increases) the price, p(t), will rise, and

(b) as the resource reaches its final stages, the price will saturate (level off).

The simplest formula that captures these assumptions is probably

$$p(t) = p(1982)\ (a\ x(t)/x_\infty - b) \quad , \quad t > 1982 \quad ,$$

and (effectively) this is the one used for the Summer School. The constants a and b are chosen so that

(i) the formula also applies to the year 1982, and

(ii) p(2050) ≈ 150 (estimated using 'think of a big number' principle) for the M = 2 case with appropriate rescaling for the other cases.

Although our formula may prove to be an inexact representation of the price of oil (and shouldn't be used to work out the possible price of a litre of petrol for the year 2000), the price influence is not such a great one in our model and our estimate will probably be adequate for projecting future trends of oil consumption.

Model M = 1 does not use this extrapolation but rather for all time, $t \geqslant 1909$, assumes that

$$p(t) = p(1909)\ \{1 + b\ (t - 1909)\}^{1/\beta} \quad .$$

The constant b is chosen so that condition (ii) above is met, giving

$$b = 0.00178 \quad .$$

It is interesting that using our analytic expression for price gives us

$$p(1982) = \$35$$

which is reasonably close to the actual value of $34/barrel for the year 1982.

APPENDIX 1B

SUMMARY OF MODEL AND DATA

The model equation:

$$\frac{dx}{dt} = \lambda \left(1 - \frac{x(t)}{x_\infty}\right) x(t) \left(\frac{p_o}{p(t)}\right)^{\beta}$$

Initial condition:

$$x(t_o) = 4.29 \text{ gigabarrels}$$

Variables:

x(t) is the cumulative consumption of oil (measured in gigabarrels) up to the end of year t.

p(t) is the price of a barrel of oil expressed in constant 1982 US dollars.

Constants:

λ is a growth constant	= 0.07 $year^{-1}$
x_∞ is an estimate of the total oil resource	= 2000 gigabarrels
t_o is the end of the starting year	= 1909
p_o is the price of oil in year t_o	= 6.11 (constant 1982 US dollars per barrel)
β is the price elasticity of consumption	= 0.07 (dimensionless)

CHAPTER 2

ENERGY IN PERSPECTIVE

A Film Script by

B.P. AUSTRALIA LIMITED

"ENERGY IN PERSPECTIVE"

A SUMMARY OF THE FILM SCRIPT
Made available by BP Australia Limited

SUMMARY This is a commentary on a wide range of energy issues covering the world's dependence on finite reserves of fossil fuels, energy wastage, conservation and the need to develop renewable energy resources.

1 INTRODUCTION

Energy. Without it, there would be no movement, no life, not even light or sound. A dead and empty world.

No wonder man worshipped the sun. Through the process of photosynthesis it creates all our food, and many of our raw materials. Its energy powers the climate, evaporating water from seas, creating winds that carry it over the land, to fall as rain, and so return once more to the sea. A perpetual cycle of motion, which man long since learned to harness. Power, free as air; and constantly being renewed.

But today, wind and water supply less than ten per cent of our energy needs. Over ninety per cent now comes from the planet's capital reserves - energy locked away in the remains of once-living matter. Coal, oil and natural gas - the fossil fuels. Once used, they are gone forever. And so is uranium, our most recently harnessed capital resource. This dependence on finite capital reserves makes our civilisation different from any that has gone before.

In the past, civilisations rose and fell that were totally dependent on the daily income of energy from the sun, on the labour of men and animals, on the harnessing of wind and water.

2 CONSUMPTION/CONSERVATION

It was only a few hundred years ago that a shortage of wood first caused 'stinking sea coals' to be burnt. For the first time, man was digging into his capital. For generations, consumption was negligible. It was not until the middle of the last century that it began to double every 15 years.

Shortly afterwards, oil made its first modest appearance. Discoveries multiplied fast, as its potential came to be realised. And as gas followed towards the end of the century, consumption of all the fossil fuels increased substantially. Even into the 30's when road transport really came into its own, coal still provided most of our energy. But in the 60's although nuclear energy was now on the scene it was oil consumption that rose most dramatically.

Only a decade later came the first ominous signs - the rate of oil consumption began to outstrip the discovery of new reserves. Yet it's expected that in the 80's, demand will be greater still, accounting for more than half the world's total energy needs. This expansion cannot continue indefinitely. From the start, there's only been a finite quantity of oil on this planet - some two million-million barrels.

In the first hundred years we consumed only 15 per cent of it, but in the single decade of the 70's we shall consume another 15 per cent. If we continue this way, consumption will reach a peak around the turn of the century, and then fall at a rate similar to its rise. This kind of pattern is seen as inevitable; the staggering fact is that 80 per cent of total world reserves could well be consumed in less than a human lifetime.

With coal, from deep mines and especially from the vast opencast reserves still undeveloped in Russia and North America, the outlook is brighter. The rate of consumption is lower and world reserves are eleven times those of oil and gas. Production could be increased considerably. But looking as far ahead as the Spanish Armada is behind us - 400 years - its expected that 80 per cent of it will have gone.

ENERGY IN PERSPECTIVE

The challenge of ever-growing demand is nothing new, but as we're forced to venture further into the remote and hostile wastes of our planet, as we probe even deeper into the oceans for the earth's remaining reserves, energy is bound to become more expensive. The investment in North Sea oil is enormous - for a given yield of energy, 25 times what it cost to develop the oilfields of the Middle East only a few years ago.

In North America and elsewhere, there are vast reserves of oil-bearing sands and shales, but to exploit them the unit cost could be even higher than North Sea oil.

The age of cheap energy is over. Over just as we'd begun to get used to it. Frivolous waste is something we could easily avoid, but the problem goes deeper, to the roots of our whole way of thinking. The need is urgent to turn ingenuity from thoughtless over-consumption to a more intelligent use of energy. But first, we need to understand the situation we're in. Ahead of us there are problems as well as exciting.possibilities.

3 PRIORITIES

Coal, oil, natural gas and the metal, uranium. Each is a remarkable compact source of energy, available to us as heat, that can be turned into work. Transformed, for instance, into mechanical

energy. Add a dynamo - and mechanical energy becomes electrical.

But whenever energy is converted from one form to another, some of it goes to waste. However well a system is designed, complete conversion is impossible. This is one of the laws of nature. Apply it to the conversion of coal into electricity, and we begin to see what this can mean. Of the total energy-content of the coal, it takes the equivalent of five percent to mine it, and raise it to the surface and a further 0.2 percent of the balance to transport it, say 50 miles to the power station. The coal is then burnt, with a heat-loss of as much as 70 percent, then there are transmission losses. On average, a further six per cent. So only about a quarter of the coal's potential reached your electric fire, and a light bulb is much worse; one and one-third per cent converted to light, the rest, lost between coalmine and chandelier. It can even be shown that the total heat wasted in America's power stations is enough to warm every home in the country. Small wonder we need such prodigious quantities of fuel.

Power stations can't avoid producing surplus heat, but we could make better use of it, as they do at Bures-Orsay, near Paris.

By combining power-generation with district central-heating, they've raised the overall efficiency to 70 percent - more than double the norm.

The same line of thinking lies behind the drive to insulate the millions of homes we built in the years when energy was cheap. "Save it" is now the motto we're learning to live by. But the future demands more radical measures, and it's vital to get our priorities right.

For transport, the liquid fuels of today will remain essential for the foreseeable future. But otherwise, precious reserves of petroleum should no longer be burned. Instead, they should be used increasingly as feedstock in the creation of essential chemicals, fertilisers and pesticides, plastics, solvents, and rubbers.

We should look on natural gas as the fuel for those industrial processes that depend on its high degree of purity, for use as chemical feedstock and for domestic heating. We cannot afford to squander it on power-generation. Instead, we should continue to rely on coal as the principal fuel for this purpose. Even so, increasing demand for electricity will also call for a marked expansion of nuclear power. Till now, progress has been steady, but slow. Present day reactors are still very inefficient in their use of uranium. The priority is now urgent to develop the breeder-reactor, 150 times as efficient, before the sources of relatively cheap uranium are exhausted. But ultimate success depends on finding safe methods of handling and storing highly toxic wastes.

Faced with such problems and with the eventual limits of our capital resources, we must examine every possibility, including any contribution that could be made by the enduring forms of income-energy.

Hydro-schemes at present supply only a few per cent of the world's electricity, but there's potential for expansion in a number of areas.

There's power too, in the rise and fall of the tides. It's already been harnessed on the south coast of Brittany, and has proved itself economic, at no risk to the environment. There are other places in the world with potential for such a development.

Power from the sea - power from the wind. It, too, has potential, but as yet the technology to harness it efficiently doesn't exist.

4 COST BARRIER

Power, too, from the earth itself. At Lardarello in Italy, they've been tapping a natural source of high-pressure steam since 1904, producing enough electricity to run most of Italy's railways. Again, there are places where the same phenomenon occurs, which could make a valuable, if limited, contribution to energy needs.

Power direct from the sun! There is an experimental centre at Mont Louis, in the Pyrenees. Even with existing technology, a tenth of the Arizona desert could power the entire United States - in theory. In practice, the problems are daunting. The harnessing of solar energy for power is a long-term goal, but certainly one worth pursuing. But the use of solar energy for domestic water heating is already a reality.

To convert sunlight directly into electricity would open up even greater possibilities. The barrier is its cost. But one day the solar-cell could be working for us all.

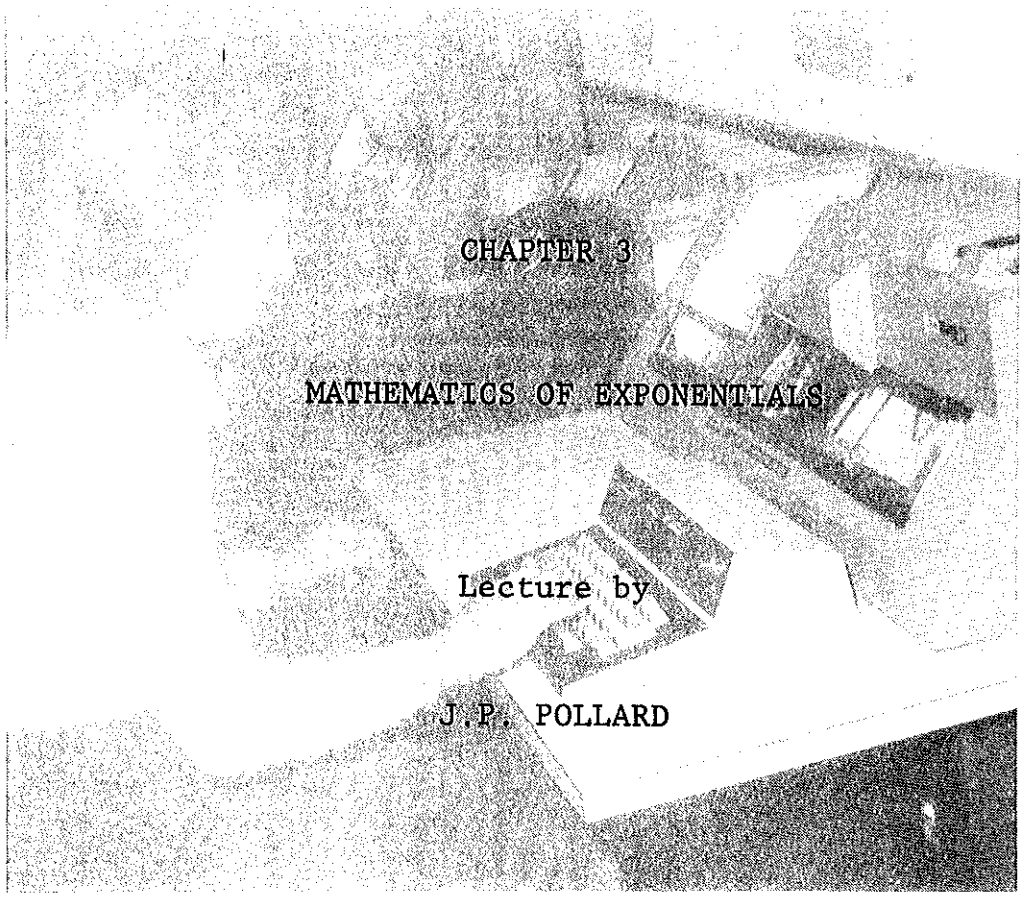
Nuclear fusion - the source of the sun's prodigious energy. To harness this on earth would be the greatest prize of all. It would mean that from a cubic metre of sea-water we could produce energy equal to 200 tons of oil, and solve our energy needs for ever. The prize continues to elude us. No-one knows when, or indeed, if ever, controlled fusion will be achieved.

In laboratories around the world, the means to power the 21st century are being explored. But on the frontiers of science, time is of the essence. For the remainder of this century at least, science must also devote its ingenuity to finding ways of conserving our precious capital reserves - the fossil fuels, and uranium.

5. CONCLUSION

We have to accept that the world of cheap energy is behind us, and a world of limitless energy still beyond our reach. The gap can be bridged, but only by adopting a radically new attitude to existing resources and the way we use them. It begins with the recognition that everything has its energy-demand. A concrete motorway? 270 000 litres of fuel for every single kilometre! Even to keep a family in drinking water takes 40 kilos of coal a year. And nowadays, a steak takes more energy to produce than we get back by eating it.

Energy is the most precious commodity there is. The challenge of our times is to find ways of using more wisely the sources we know, and to harness the new and prolific sources we know to exist - and to do it in a single generation.



CHAPTER 3

MATHEMATICS OF EXPONENTIALS

Lecture by

J.P. POLLARD

CONTENTS

3.1 THE EXPONENTIAL FUNCTION

A power function

$$y(x) = e^x \quad \text{(e raised to the power x)}, \tag{3.1}$$

where x is an integer and e is a constant (base) as yet unspecified, is easy to understand. We have

$y(0) = e^0 = 1$ (most numbers raised to the zeroth power are 1)

$y(1) = e^1 = e$

$y(2) = e^2 = ee$

$y(3) = e^3 = eee.$

The extension to rational mumbers (for x) is reasonably straightforward. For example

$$y(\tfrac{1}{2}) = e^{1/2} = \sqrt{e} \quad \text{and}$$

$$y(\tfrac{3}{2}) = e^{3/2} = (\sqrt{e})^3.$$

The further extension to real numbers is conceptually simple. For example

$$y(1.25) = e^1 \, e^{0.25} \quad \text{(remember the rule about adding exponents)}$$

$$= e \, e^{1/4} \quad \text{and}$$

$$= e \, \sqrt[4]{e}.$$

Admitting negative values is trivial since

$$e^x e^{-x} = 1$$

hence $$y(-x) = e^{-x} = 1/e^x = 1/y(x). \tag{3.2}$$

Taking logarithms (to the base 10) of equation (3.1) yields

$$\log y = x \log e \tag{3.3}$$

hence the 'inverse' function which expresses x as a function of y is

$$x(y) = \log y / \log e.$$

Our original expression given by equation (3.1) is thus like an antilogarithm.

Encouraged by the tangible results obtained so far we will attempt to differentiate our power function. For ease of presentation, we will adopt the standard notation

$$y'(x) = \frac{dy}{dx} \quad \text{then,}$$

since $y(x) = e^x$ (remember 'e' is not yet specified),

$$y'(x) = \lim_{\delta x \to 0}\left[\frac{e^{x+\delta x} - e^x}{\delta x}\right] \quad \text{- from first principles}$$

$$= \lim_{\delta x \to 0}\left[\frac{e^x e^{\delta x} - e^x}{\delta x}\right] \quad \text{- from a property of powers}$$

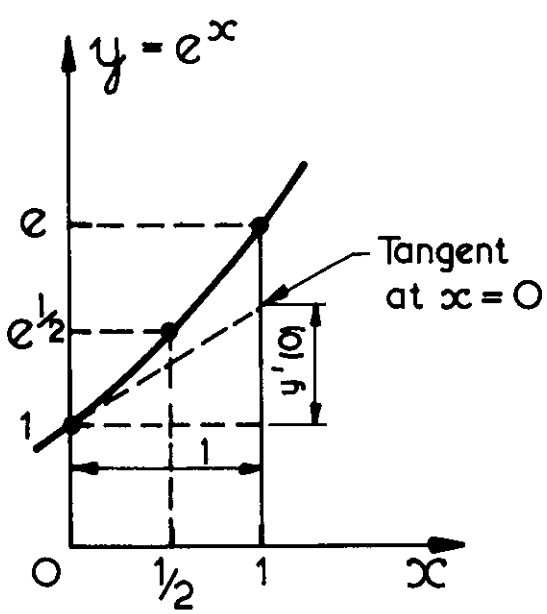
$$= e^x \lim_{\delta x \to 0}\left[\frac{e^{\delta x} - 1}{\delta x}\right] \quad \text{- taking out } e^x \text{ as a common factor.}$$

Now the derivative at x = 0 is simply

$$y'(0) = \lim_{\delta x \to 0}\left[\frac{e^{\delta x} - 1}{\delta x}\right] \quad \text{- from the definition of derivative}$$

hence $y'(x) = y'(0)\, e^x = y'(0)\, y(x)$.

A rough sketch of the function



shows that for e close to 1, $y'(0)$ is close to 0, and that for e large, $y'(0)$ is also large. Let us seek a particular value of e such that

$$y'(0) = 1 \tag{3.4}$$

for then we have the highly desirable result

$$y'(x) = y(x) \tag{3.5}$$

from which our central result is obtained for the so-called exponential function,

$$\frac{dy}{dx} = y(x)$$

$$\text{or} \quad \frac{de^x}{dx} = e^x \ ,$$

i.e. e^x is its own derivative. (3.6)

Of almost equal importance is the integral result obtained from equation (3.6). We have

$$\int_0^x \frac{dy}{dx}\,dx = \int_0^x y(x)dx \quad ;$$

but

$$\int_0^x \frac{dy}{dx}\,dx = \int_0^x dy = [y(x)]_0^x = y(x) - y(0) = y(x) - 1 \ ,$$

hence

$$\int_0^x y(x)dx = y(x)-1 \ ,$$

$$\text{or} \quad \int_0^x e^x dx = e^x-1 \text{ (definite integral)} \ ,$$

$$\text{or} \quad \int e^x dx = e^x \text{ (indefinite integral)} \ ,$$

i.e. e^x is its own integral. (3.7)

Good, you say, but what is the value of e that gives these fantastic properties? On to that now.

3.2 ESTIMATING THE NUMBER e

At present, all we know about the number e is contained in the expression

$$y'(0) = \lim_{\delta x \to 0} \left[\frac{e^{\delta x} - 1}{\delta x} \right] = 1 \quad .$$

Now if we consider the above equation for sufficiently small values of δx, we have

$$e^{\delta x} - 1 \simeq \delta x$$

then $$e^{\delta x} \simeq 1 + \delta x$$

and, taking logs,

$$\delta x \log e \simeq \log(1+\delta x)$$

$$\log e \simeq \frac{1}{\delta x} \log(1+\delta x)$$

$$\simeq \log(1+\delta x)^{\frac{1}{\delta x}}$$

$$e \simeq (1+\delta x)^{\frac{1}{\delta x}} \quad .$$

Instead of the above, let us introduce

$$n = 1/\delta x \quad (n \to \infty \text{ as } \delta x \to 0)$$

then $$e \simeq (1+\frac{1}{n})^n \tag{3.8}$$

and, in fact, $$e = \lim_{n \to \infty} (1+\frac{1}{n})^n \quad . \tag{3.9}$$

Let us tabulate a few estimations of e using the approximation (3.8). We obtain the results

$$(1+\frac{1}{2})^2 = 2.25$$

$$(1+\frac{1}{5})^5 = 2.489$$

$$(1+\frac{1}{10})^{10} = 2.594$$

$$(1+\frac{1}{20})^{20} = 2.653 \quad .$$

This method does not seem to be very practical since we need to go much further to obtain e to 4 figures; nevertheless someone has calculated

$$(1+\frac{1}{10000})^{10000} = 2.7181 \quad .$$

Even this is not entirely correct for the figures stated, since the value to 7 figures is

$$\boxed{e = 2.718282} \quad ; \tag{3.10}$$

which is as highly regarded in the mathematical world as the number

$$\pi = 3.141593 \tag{3.11}$$

Digression

Having met π as a really special number, you may not like to see a competitor enter the field. Well don't worry! π and e are cousins through the strange but beautiful relationship

$$e^{\pi\sqrt{-1}} = -1, \quad \text{(derived by Euler)}. \tag{3.12}$$

3.3 APPROXIMATIONS FOR e^x

We have already established the derivative property of the exponential function $y(x) = e^x$; it is

$$\frac{dy}{dx} = e^x \quad \text{(equation (3.6))}.$$

Let us assume

$$y(x) = e^x = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + \ldots, \tag{3.13}$$

a power series expansion with coefficients a_0, a_1, a_2, ... to be determined, then

$$y(0) = e^0 = 1 = a_0 + 0 + 0 + \ldots$$

$$\therefore \quad a_0 = 1 \quad .$$

Differentiating equation (3.13),

$$\frac{dy}{dx} = e^x = a_1 + 2a_2x + 3a_3x^2 + 4a_4x^3 + \ldots ,$$

hence equating coefficients of like powers of x (they must be the same)

$$a_1 = a_0$$

$$\therefore \quad a_1 = 1$$

$$2a_2 = a_1$$

$$\therefore \quad a_2 = 1/2$$

$$3a_3 = a_2$$

$$\therefore \quad a_3 = 1/(2x3)$$

$$4a_4 = a_3$$

$$\therefore \quad a_4 = 1/(2x3x4)$$

and so on. If we define the factorial function as

$$n! = 1x2x3 \ldots x(n-1)xn \quad \text{(the product of the first n integers)}, \tag{3.14}$$

then we obtain the important result

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots \tag{3.15}$$

which, fortunately, has terms towards the end that get smaller and smaller no matter how big x happens to be since n! for large n is a whopper (see the table).

Pardon my figuring out.

n	n!
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
15	1.3076744×10^{12}
20	2.4329020×10^{18}

```
01 RCLO
02 1
03 +
04 STOO
05 fPAUSE
06 X
07 GTOOO
```

```
10 REM N!
20 M=1
30 FOR N=1 TO 20
40 M=M*N
50 PRINT N;"!=";M
60 NEXT N
70 END
```

Equation (3.15) then gives us a convergent series expansion although many terms may be required for large values of x. The value of e^x for x = 1 is simply obtained by adding up the reciprocals of entries in the above table (to n = 10),

$$e \simeq 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{10!} \tag{3.16}$$

$$= 2.718282 \quad .$$

The additive property of powers may be exploited when x is large. For example, say we require e^3. Rather than calculate the result directly from equation (3.15)

$$e^3 = 1 + 3 + \frac{3^2}{2} + \frac{3^3}{6} + \frac{3^4}{24} + \ldots$$

we would use instead

$$e^3 = eee = 20.08554 \quad .$$

Other approaches are used to calculate e^x on a digital computer but we will not investigate them here except to say that they are usually part of a package of routines made available by the machine manufacturer. For example, in FORTRAN or BASIC we simply code

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

to return the exponential of X in the storage location Y.

3.4 NATURAL LOGARITHMS

The natural logarithm, denoted $\log_e y$ or $\ell n\ y$, inverts the relationship given by the exponential

$$y = e^x ,$$

to give $\ell n\ y = x ,$ (3.16)

just as the ordinary logarithm, denoted log y, inverts the relationship given by the antilog expression

$$y = 10^z ,$$

to give $\log y = z .$

From our equation (3.6) rearrangement gives

$$\frac{dy}{y} = dx$$

and then $\int \frac{dy}{y} = \int dx = x = \ell n\ y ,$

hence the important result

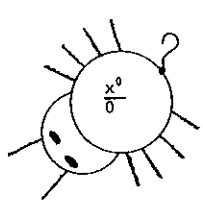
$$\boxed{\int \frac{dy}{y} = \ell n\ y} \quad . \tag{3.17}$$

Have you ever tried to apply the integration formula

$$\int x^n dx = \frac{x^{n+1}}{n+1}$$

when $n = -1$

$$\int x^{-1} dx \ (= \int \frac{dx}{x}) =$$



Now we know better

$$\int x^n dx = \begin{cases} \dfrac{x^{n+1}}{n+1} , & n \neq -1 \\ \ell n\ x , & n = -1 \end{cases} \quad . \tag{3.18}$$

On a computer, we use one of the following expressions to return a natural logarithm:

```
X=ALOG(Y)      - with the FORTRAN language
```

or

```
X=LOG(Y)       - with the BASIC language .
```

3.5 DIFFERENTIAL EQUATIONS

We have already met a differential equation, namely equation (3.6),

$$\frac{dy}{dx} = y \quad , \quad y(0) = 1 \quad ,$$

which has as its solution the exponential function,

$$y = e^{x} \quad .$$

For the Summer School, we need to change to different variables. (Mathematicians do this all the time - it keeps out the uninitiated!) We want to determine the total consumption of oil, x gigabarrels, as a function of time, t years. In the *excessively optimistic model of an unlimited resource* ($x_\infty = \infty$), and in the absence of any pricing constraints, we have the equation

$$\boxed{\frac{dx}{dt} = \lambda x} \quad , \quad \boxed{x(t_0) = x_0} \quad . \qquad (3.19)$$

DE initial condition

Here is one way we can solve the above problem:

$$\frac{dx}{x} = \lambda dt$$

hence $$\int_{x_0}^{x} \frac{dx}{x} = \lambda \int_{t_0}^{t} dt$$

(x is x_0 when t is t_0 - the initial condition),

$$[\ell n\ x]_{x_0}^{x} = \lambda [t]_{t_0}^{t} \quad \text{from equation (3.17)}$$

$$\ell n\ x - \ell n\ x_0 = \lambda(t-t_0) \quad ,$$

$$\ell n(x/x_0) = \lambda(t-t_0) \quad , \text{ remember } \ell n(a/b) = \ell n\ a - \ell n\ b,$$

and finally $$x = x_0 e^{\lambda(t-t_0)} \quad . \qquad (3.20)$$

[Also equations (3.19) and (3.20) show us that with some slight manipulation

$$\frac{de^{\lambda t}}{dt} = \lambda\, e^{\lambda t} \quad .] \qquad (3.21)$$

Of course, we would not be surprised that using the result (3.20) gives us an enormous overestimate of oil consumption to the present time although it turns out that twenty years ago the model produced acceptable estimates and, by some, was accepted as a basis for estimating consumption in the future.

Other, usually more common, exponential processes, for example radioactive decay, follow the equation

$$\boxed{\frac{dx}{dt} = -\lambda x} \quad , \quad \boxed{x(t_0) = x_0} \qquad (3.22)$$

DE initial condition

(Note the negative sign!) In this instance, the solution is just that of equation (3.20) with the sign of λ changed.

We have

$$x = x_0 e^{-\lambda(t-t_0)} \quad . \qquad (3.23)$$

The model here is usually more global (covers a large time span) as the solution dies away, not like our simplistic oil model where the solution

grows and grows and grows

without bound!

NOTES

APPENDIX 3A

SOME TUTORIAL PROBLEMS

Q1. Write down the derivatives of the following functions:
(i) x^2, (ii) x^n, (iii) e^x, (iv) e^{cx}, (v) ℓnx

Q2. Write down the indefinite integrals of the following functions:
(i) x^2, (ii) x^n, (iii) e^x, (iv) e^{cx}, (v) $(x+1)/x$

Q3. Solve the following DEs analytically and hence calculate $y(\frac{1}{2})$.

(i) $\frac{dy}{dx} = 2x$, $y(o)=1$

(ii) $\frac{dy}{dx} = 2y$, $y(o)=1$

Q4. Show that (at least for sufficiently small values of x)

$$1 + 4e^{ax} + e^{2ax} \cong \frac{3}{ax}(e^{2ax}-1)$$

by expanding both sides to terms of order x^3 using the series expansion for e^x given as equation (3.15).

Q5. Taking equality for the approximation of *Q4*, solve for e^x and hence obtain an approximation for e^x - let's call the approximation E(x) which could be used if an EXP function is not available (as is done in Appendix 3B with $a=\frac{1}{4}$).

Q6. Show for the approximation of *Q5* that $1/E(x)=E(-x)$, *cf.* $1/e^x = e^{-x}$.

A1. (i) $2x$, (ii) $n\,x^{n-1}$, (iii) e^{x}, (iv) ce^{cx}, (v) $1/x$

A2. (i) $x^3/3$, (ii) $x^{n+1}/(n+1)$, (iii) e^{x}, (iv) e^{cx}/c, (v) $x+\ell nx$

A3. (i) $y=1+x^2$, $y(\frac{1}{2})=1.25$

(ii) $y=e^{2x}$, $y(\frac{1}{2})=2.718$

A4. Clue: $e^{2ax} = 1 + 2ax + 2a^2x^2 + \frac{4}{3}a^3x^3 + \frac{2}{3}a^4x^4 + 0(x^5)$,

where $+ 0(x^5)$ means 'plus terms of order x^5 and higher'.

A5. $E(x) = [\{ 2(\frac{ax}{3}) + \sqrt{1+3(\frac{ax}{3})^2}\} /(1-\frac{ax}{3})]^{1/a} \cong e^{x}$

A6. Clue: $\frac{1}{b+\sqrt{c}} = \frac{1}{b+\sqrt{c}} \left(\frac{b-\sqrt{c}}{b-\sqrt{c}}\right)$

$$= \frac{b-\sqrt{c}}{b^2-c} \quad .$$

APPENDIX 3B

SOME BASIC FUNCTIONS FOR MICROS IF NOT PROVIDED

(i) $x' = \sqrt{x}$

Input argument variable X

Uses variables X,Y,Z

Output variable X

Calling procedure

GØSUB 31005 - normal

or GØSUB 31010 - if $\sqrt{|x|}$ required

or GØSUB 31015 - if x known to be non-negative

or GØSUB 31020 - above and y ≃ $\sqrt{x}$ is available as a guess

or GØSUB 31025 - above and x known to be non-zero.

BASIC coding based on Newton-Raphson method.

```
31000     REM X=SQR(X)
31005     IF X<0 THEN PRINT "SQR ERR";X
31010     X=ABS(X)
31015     Y=X
31020     IF X=0 THEN RETURN
31025     Z=1E38:GØTØ 31035
31030     Z=Y
31035     Y=(Y+X/Y)/2: IF Y<Z THEN 31030
31040     X=Y:RETURN
```

(ii) $x' = e^x$

Input argument variable X

Uses variables V,W,X,Y,Z

Output variable X

Calling procedure

GØSUB 31105

BASIC coding based on square root method of a previous Summer School (reference on next page)*

```
31100     REM X=EXP(X)
31105     IF X>87 THEN PRINT "EXP ØVER"; X : X=1E38 : RETURN
31110     IF X<-87 THEN PRINT "EXP UNDER"; X : X=0 : RETURN
31115     V=INT(X+.5):W=(X-V)/12 : X=1+3*W*W : GØSUB 31015
```

```
31120     X=(2*W+X)/(1-W) : X=X*X*X*X
31125     IF V=0 THEN RETURN
31130     Y=2.718282
31135     IF V<0 THEN V=-V : Y=1/Y
31140     FOR W=1 TO V : X=X*Y : NEXT W : RETURN
```

(iii) $x' = \ell n\ x$

Input argument variable X

Uses variables V,W,X,Y,Z

Output variable X

Calling procedure

```
GØSUB 31205
```

BASIC coding based on square root method of previous Summer School*

```
31200     REM X=LØG(X)
31205     IF X<=0 THEN PRINT "LØG ERR"; X:X=-1E38 : RETURN
31210     V=1 : Y=X
31215     W=X : GØSUB 31025
31220     IF W<.8 THEN V=V*2 : GØTØ 31215
31225     IF W>1.3 THEN V=V*2 : GØTØ 31215
31230     X=6*(W-1)*V/(W+4*X+1) : RETURN
```

(iv) $x' = x^y$

Input argument variables X and Y

Uses variables U,V,W,X,Y,Z

Output variable X

Calling procedure

```
GØSUB 31255
```

BASIC coding using earlier routines

```
31250     REM X=X↑Y
31255     U=Y : GØSUB 31205 : X=U*X : GØSUB 31105 : RETURN
```

---

* Newton, P.J.F. ed. [1977] - Down but never out - the mathematics and computation of exponentials arising in the fields of physics, chemistry, biology...AAEC/S19.

CHAPTER 4

MODELS AND SIMULATION

Lecture by

B.E. CLANCY

CONTENTS

4.1 INTRODUCTION

During this Summer School, you will be examining some possible ways in which we will use up a finite energy resource - oil. These possibilities are described by models - really equations which describe how the oil will be used. They can only be used if we can obtain solutions to the equations; my lectures are devoted to the problem of solving these equations and getting hard numbers as a result.

4.2 EXPONENTIAL GROWTH MODEL

The simplest model is based on the assumption that at any time t, the consumption rate $\frac{dx}{dt}$ is simply proportional to x, the total consumption up to that time. In symbols

$$\frac{dx}{dt} = \lambda x,$$

with λ being a known constant.

To solve this equation we use our first trick, which is to note that

$$\frac{dx}{dt} = 1/\frac{dt}{dx} \quad ,$$

so our original equation can be written

$$\frac{dt}{dx} = \frac{1}{\lambda x} = \frac{1}{\lambda}\,\frac{1}{x} \quad .$$

We solve this formally by integrating to get

$$t = \int \frac{1}{\lambda} \cdot \frac{1}{x}\, dx = \frac{1}{\lambda}\int \frac{1}{x}\, dx \quad .$$

You will (by now) recognise that the integral involves the natural logarithmic function so

$$t = \frac{1}{\lambda}\, \ell n(x) + C \quad ,$$

where C is some constant which we have to determine. We do this by going back to the starting time for our problem, the year t_o, at which stage the total production had reached the value x_o. Since our solution must agree with this piece of information,

$$t_o = \frac{1}{\lambda}\, \ell n(x_o) + C \quad ,$$

where C is the same constant. When we subtract this equation from the previous one, the constant C is eliminated and we have

$$t - t_o = \frac{1}{\lambda}\, \ell n(x) - \frac{1}{\lambda}\, \ell n(x_o) \quad .$$

from which we obtain, after some algebra,

$$\lambda(t - t_o) = \ell n \left(\frac{x}{x_o}\right) \quad .$$

Taking the appropriate antilogarithms (exponentials)

$$e^{\lambda(t - t_o)} = \frac{x}{x_o} \quad ,$$

and finally our solution is

$$x = x_o e^{\lambda(t - t_o)} \quad ,$$

or

$$x = x_o \exp\{\lambda(t - t_o)\} \quad .$$

If we differentiate this equation with respect to t, we get an expression for the consumption rate $\frac{dx}{dt}$:

$$\frac{dx}{dt} = \lambda x_o e^{\lambda(t - t_o)}$$

which demonstrates that our solution does agree with the original equation

$$\frac{dx}{dt} = \lambda x \quad .$$

Of course, if these did not agree with one another there would be some error in our analysis.

The last three equations represent the uncontrolled exponential growth situation. Since there is a finite amount of the resource available, x cannot exceed this amount. Because this model predicts that x grows without limit, we must ultimately reject the model even though the results may be reasonable for times relatively close to t_o.

4.3 THE HUBBERT MODEL

This model assumes that consumption rate $\frac{dx}{dt}$ is proportional not only to the total consumption to date but also to the fraction of the resource left to be exploited. If the total amount of the resource which can ever be obtained is denoted by x_∞, the fractional amount consumed at any time is $\frac{x}{x_\infty}$ and the fraction left to be exploited is $1 - \frac{x}{x_\infty}$. In symbols this model states that

$$\frac{dx}{dt} = \lambda x\left(1 - \frac{x}{x_\infty}\right) \quad .$$

In early times, x is small so $1 - \frac{x}{x_\infty}$ is little different from 1, hence this model is little different from our first model and its solution will be like that for our first solution - *while x is small.*

To solve this equation, we start by using our first trick, and write the equation as

$$\frac{dt}{dx} = 1/[\lambda x(1 - \frac{x}{x_\infty})] \quad , \quad \text{or}$$

$$\frac{dt}{dx} = \frac{x_\infty}{\lambda x(x_\infty - x)} \quad ,$$

where λ and x_∞ are fixed constants and, when we try integrating, we get

$$t = \frac{1}{\lambda}\int \frac{x_\infty}{x(x_\infty - x)}\, dx \quad .$$

To do this integration, we use a second trick which is simply to know that the complicated fraction

$$\frac{x_\infty}{x(x_\infty - x)}$$

can be written as a sum of two simple fractions, each of which can be integrated. There are standard methods in text books for doing this but here we simply note the result

$$\frac{x_\infty}{x(x_\infty - x)} = \frac{1}{x} + \frac{1}{x_\infty - x}$$

and our integration problem becomes

$$t = \frac{1}{\lambda}\left\{\int \frac{1}{x}\, dx + \int \frac{1}{x_\infty - x}\, dx\right\} \quad .$$

Each of these integrals involves the logarithmic function and we have

$$t = \frac{1}{\lambda}\{\ell n(x) - \ell n(x_\infty - x)\} + C \quad .$$

The fixed constant C has to have a value which makes our solution agree with the starting values which again say that $x = x_o$ when $t = t_o$. So we must have

$$t_o = \frac{1}{\lambda}\{\ell n(x_o) - \ell n(x_\infty - x_o)\} + C \quad .$$

The constant C is again eliminated when we subtract these two equations and

$$t - t_o = \frac{1}{\lambda}\{\ell n(x) - \ell n(x_\infty - x)\} - \frac{1}{\lambda}\{\ell n(x_o) - \ell n\ (x_\infty - x_o)\} \quad ,$$

or $$\lambda(t - t_o) = \ell n \frac{x\ (x_\infty - x_o)}{x_o(x_\infty - x)} \quad .$$

Taking antilogarithms, we get

$$e^{\lambda(t - t_o)} = \frac{x(x_\infty - x_o)}{x_o(x_\infty - x)}$$

and after some more algebra we arrive at our solution

$$x = \frac{x_o x_\infty E}{x_\infty - x_o + x_o E} \quad ,$$

where E stands for $e^{\lambda(t - t_o)}$ or $\exp\{\lambda(t - t_o)\}$.

If we differentiate this solution with respect to t, we recover the original equation

$$\frac{dx}{dt} = \lambda x(1 - \frac{x}{x_\infty})$$

in the form

$$\frac{dx}{dt} = \frac{\lambda(x_o x_\infty - x_o)E}{(x_\infty - x_o + x_o E)^2} \quad .$$

During the early years, $\lambda(t - t_o)$ is small and the exponential term

$$E = e^{\lambda(t - t_o)}$$

is very close to 1. Writing our solution as

$$x = \frac{x_o x_\infty E}{x_\infty + x_o(E - 1)} \quad ,$$

we see that when $t \simeq t_o$ we obtain the approximation

$$x \simeq \frac{x_o x_\infty E}{x_\infty} = x_o e^{\lambda(t - t_o)} \quad ;$$

this model gives results close to those of the simple exponential growth model.

However, this approximation is not warranted for later years when t,E get large. Then we do better by writing the solutions as

$$x = x_\infty \frac{x_o E}{x_o E + x_\infty - x_o} \quad .$$

From this we see that when E is large, x gets closer and closer to x_∞ but is always smaller than x_∞ because the multiplying factor is always less than unity.

Because this model does not predict total consumption greater than the available amount, we would have more confidence in predictions made from its use - and indeed such predictions have been made. One defect with this model is that it makes no allowance for the effect that the price of the resource may have on the production rate. This defect is remedied in our third model.

4.4 PRICE DEPENDENT MODEL

This final model - the one which will be examined fully during the Summer School - includes an allowance for the effect that product price may have on consumption rates. Quite simply, the Hubbert model formula is multiplied by another factor to give

$$\frac{dx}{dt} = \lambda \left(\frac{p_o}{p(t)} \right)^\beta x \left(1 - \frac{x}{x_\infty} \right)$$

in which $p(t)$ is a product price,

p_o is the price at time $t = t_o$, and

β is another constant.

The significance of the additional factor is discussed in Chapter 1 - we simply accept that it is a plausible correction to make to the model.

Our immediate problem is to try to solve this equation. For this we need our third and final trick, which is to assume that the price function $p(t)$ is constant during any one calendar year - only changing on New Year's Day! During any one calendar year, our equation is

$$\frac{dx}{dt} = \lambda \left(\frac{p_o}{p(t)} \right)^\beta x \left(1 - \frac{x}{x_\infty} \right)$$

where $\lambda \left(\frac{p_o}{p(t)} \right)^\beta$ is a fixed constant. This equation is the same as

that for the Hubbert model and the solution for that model will apply here, but only for the particular calendar year. The complication that arises is that we have a different equation to solve each year.

Let us look at the equations for the particular year

$$t = t_i \text{ to } t = t_i + 1$$

during which the price has a constant value which we call p_c .
Our equation is

$$\frac{dx}{dt} = \lambda\left(\frac{p_o}{p_c}\right)^{\beta} x\left(1 - \frac{x}{x_\infty}\right)$$

and at the beginning of this year the cumulative consumption x had a value x_i .

If we use the same techniques as we did for the Hubbert model, we obtain the solution in the form

$$x = \frac{x_i x_\infty E}{x_\infty - x_i + x_i E} ,$$

where $$E = \exp\left\{\lambda\left(\frac{p_o}{p_c}\right)^{\beta}\left(t - t_i\right)\right\} .$$

We are interested particularly in the value of x at the end of the year when $(t - t_i) = 1$. At this time, the function E becomes simply

$$E = \exp\left\{\lambda\left(\frac{p_o}{p_c}\right)^{\beta}\right\} \qquad 4.1$$

and the previous formula still applies for x. The consumption for the calendar year we call Δx; this is simply the difference between x at the end of the year, and $x = x_i$ at the beginning of the year.
We find that

$$\Delta x = \frac{x_i(x_\infty - x_i)\ (E - 1)}{x_\infty + x_i\ (E - 1)} . \qquad 4.2$$

These last two formulae are all we need to get the main details of the whole solution. To get the total consumption for any one year, we find the current price p_c for that year and evaluate E from formula (4.1). We must also have available the value for the cumulative production x_i to the start of the year of interest. Formula (4.2) then lets us calculate the year's production Δx and, of course, we can then calculate the cumulative production to the end of the year by adding Δx to x_i.

To 'solve' the model, we simply apply this procedure for the first year, then the second year, and so on. This would be terribly tedious to do by hand, but computers are good at doing the same thing over and over.

While accepting that this procedure gives all the information we need to analyse the resource depletion, some of you may feel dissatisfied that a formal solution of the equation is nowhere written down. We can in fact construct such a solution, details of which are in Appendix 4A.

4.5 COMPUTER SIMULATION

Your coding task during the Summer School is to write a computer program which simulates the depletion of the oil resource according to the price dependent model. From the output of the program you will recover answers to two specific questions:

(a) In which year will the oil consumption rate reach a maximum and at what level?

(b) In which year will cumulative consumption reach 90 per cent of the total resource x_∞?

Your program will probably be constructed with five blocks as follows:

Block 1.

Define values for fixed constants -

$p_o = 6.11$ (price during year 1909)

$\lambda = 0.07$

$\beta = 0.07$

$t_o = 1909$ (the end of the calendar year 1909)

$x_o = 4.29$ (total consumption to the end of 1909)

Block 2.

Specify the data which vary from run to run -

M = an integer which specifies which pricing model is to be used

X_∞ = usually 2000, but sometimes 3000.

Block 3.

An initialisation block where you will define initial values for variables which change during the course of the simulation, *i*.e. set

$p_c = p_o$

$x_i = x_o$

$t = t_o$

Block 4.

An iteration segment in which all the calculations are made. After t has been increased by 1 to point to the 'next' year, formula (4.1) is used to get a value for the exponential term E, and formula (4.2) is then used to get a value for the year's consumption Δx. This is added to x_i, the value for x at the start of the year, so that the variable x_i then represents the cumulative consumption to the end of this year. After sending current values of t, Δx, x_i and p_c to the printer and plotter (Appendix 6D) one step through the iteration block is complete. If t is less than the last year of interest, a value for p_c, next year's price is determined and your program goes back to repeat segment 4; otherwise it proceeds to

Block 5.

Any final tidying up of the program is done here. Specifically you will signal that the case is finished, flush out the plot and then stop.

The first few times that your job is submitted to the computer, you will find that you have made coding errors which the computer will detect. After this stage, there will probably be logical errors in your coding. This means that your job will appear to run successfully but the results will be wrong - not because *the computer made a mistake* but because your program told the computer to do the wrong thing. To find these errors, we suggest that you run your program with a standard set of data for which a standard set of results is available against which you can check your output. If and when you agree with the standard results, it is just possible that your program is correct; it can then be used with different data to explore the consequences of the model.

GOOD LUCK

APPENDIX 4A

COMPLETE SOLUTION OF THE PRICE DEPENDENT MODEL

Starting from the equation

$$\frac{dx}{dt} = \lambda\left(\frac{p_o}{p(t)}\right)^{\beta} x\left(1 - \frac{x}{x_\infty}\right)$$

we get easily to the form

$$\left(\frac{p_o}{p(t)}\right)^{\beta} \frac{dt}{dx} = \frac{x_\infty}{\lambda x - (x_\infty - x)} .$$

If we could find a variable u which depended on t through the equation

$$\frac{du}{dt} = \left(\frac{p_o}{p(t)}\right)^{\beta} ,$$

our equation would have the form

$$\frac{du}{dt} \cdot \frac{dt}{dx} = \frac{x_\infty}{\lambda x(x_\infty - x)} ,$$

or

$$\frac{du}{dx} = \frac{x_\infty}{\lambda x(x_\infty - x)} .$$

This is identical in form to that equation solved when discussing the Hubbert model; its solution is

$$x = \frac{x_o x_\infty E}{x_\infty - x_o + x_o E} ,$$

where $E = \exp\{\lambda(u - u_o)\}$

and x_o is the value of x when $u = u_o$.

Now the equation

$$\frac{du}{dt} = \left(\frac{p_o}{p(t)}\right)^{\beta} ,$$

with $u = u_o$ when $t = t_o$,

has the formal solution

$$u = u_o + \int_{t_o}^{t} \left(\frac{p_o}{p(t)} \right)^{\beta} dt$$

and the function E is now given by

$$E = \exp \left\{ \lambda \int_{t_o}^{t} \left(\frac{p_o}{p(t)} \right)^{\beta} dt \right\}$$

which, inserted into the formula

$$x = \frac{x_o x_\infty E}{x_\infty - x_o + x_o E} \quad ,$$

is our final solution.

For the model M = 1 we can actually carry out the above integration analytically. From Appendix 1A we have

$$p(t) = p_o \left\{ 1 + b(t - t_o) \right\}^{1/\beta}$$

therefore we have

$$E = \exp \left\{ \lambda \int_{t_o}^{t} \frac{dt}{1 + b(t - t_o)} \right\}$$

$$= \exp \left\{ \frac{\lambda}{b} \left[\ell n \left\{ 1 + b(t - t_o) \right\} \right]_{t_o}^{t} \right\}$$

$$= \exp \left\{ \frac{\lambda}{b} \ \ell n \left\{ 1 + b(t - t_o) \right\} \right\}$$

and finally

$$E = \left\{ 1 + b(t - t_o) \right\}^{\lambda/b} \quad .$$

The worksheets give a few values for x calculated using this analytic result. You may like to calculate a few other values yourself using your pocket calculator or the mini- and microcomputers.

CHAPTER 5

HYBRID COMPUTING AND MODELLING

Lecture by

C.P. GILBERT

CONTENTS

5.1 INTRODUCTION

5.1.1 Dynamic Systems

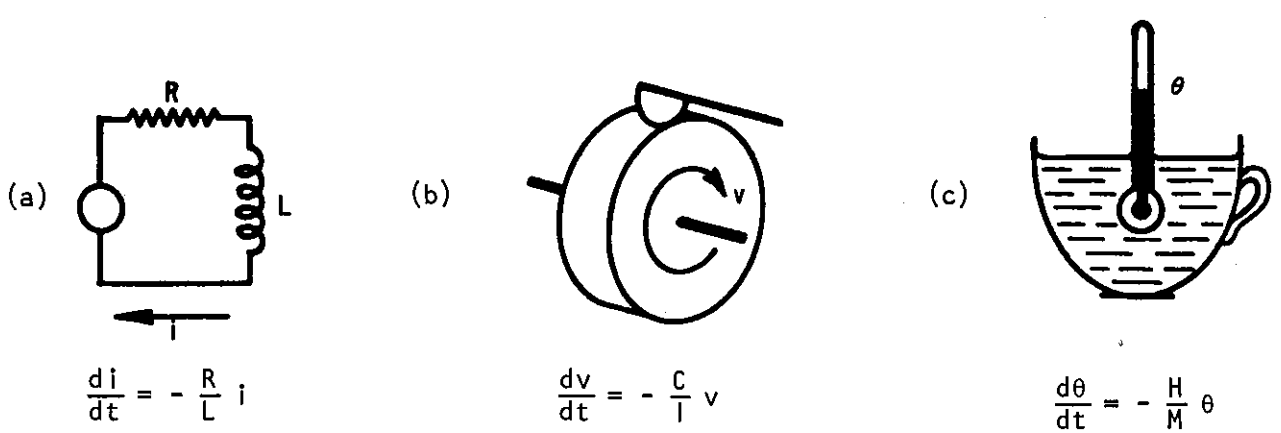
Many advances in science and engineering are possible only because of our ability to use mathematical equations to describe the behaviour of complicated systems.

Here we are mainly concerned with what are known as *dynamic* systems, *i.e.* those that vary with time, which are usually described using differential equations. An example of a dynamic situation is the movement of a ball bouncing on an uneven surface and, if necessary, equations could be formulated to describe this motion.

Having derived methods of representing dynamic systems, it will be shown that they can be used for other types of differential equations.

5.1.2 Analogues

When dynamic systems are examined in detail, one important property emerges. Many electrical, mechanical, biological and other systems can be described by equations of the same *form*, although the actual numbers involved may be different in each case. Figure 5.1 shows simple examples



$\frac{di}{dt} = -\frac{R}{L} i$ $\qquad$ $\frac{dv}{dt} = -\frac{C}{I} v$ $\qquad$ $\frac{d\theta}{dt} = -\frac{H}{M} \theta$

Figure 5.1 (a) Current i in an inductive circuit.
(b) Velocity v of a flywheel with a brake
(c) Temperature θ of a cup of hot water

of three systems, each with energy decay. The current i in an inductive circuit, the velocity v of a flywheel with a brake, and the temperature θ of a cup of hot water which is cooling down, can all be described by equations of the form:

$$\frac{dx}{dt} = -Kx \quad . \qquad (5.1)$$

Systems which resemble each other in this way are called *analogues* of one another and, if each is given an equivalent disturbance, they will all behave in exactly the same manner, although at different speeds.

Thus although the three systems are different in physical form, their *dynamic* properties are identical. There is then the possibility that we can examine one of them (that happens to be convenient) in order to find out how the others behave. As we shall see, this can assist us with the solution of very complicated sets of differential equations.

5.1.3 Simulation

One way to examine the dynamic behaviour of a nuclear reactor, say, would be to do the following experiment. A disturbance would be purposely injected, and the reactor power and temperature would be measured as they varied with time. Unfortunately, with a *full-size* power reactor the experiment would be slow, very expensive and possibly dangerous. However, if we could find some sort of analogue of the reactor (*i.e.* another physical system, or *model*, having the same dynamic behaviour), then it would be much simpler, safer and cheaper to do the same experiment on the analogue. This idea has been found to be so successful in some applications that instead of looking for convenient analogues in a haphazard way, special pieces of equipment have been built solely for this purpose.

These *analogue computers*, as they are called, consist of a number of units which can be put together like building blocks to form analogues of different systems; accurate measurements can then be made on the resulting model. The process of doing an experiment on a computer model instead of on the real system is known as *simulation*.

5.1.4 Computing Operations

As will become clear, addition and integration are the most important processes that an analogue computer has to perform, and a number of methods are possible.

Figure 5.2(a) 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:

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

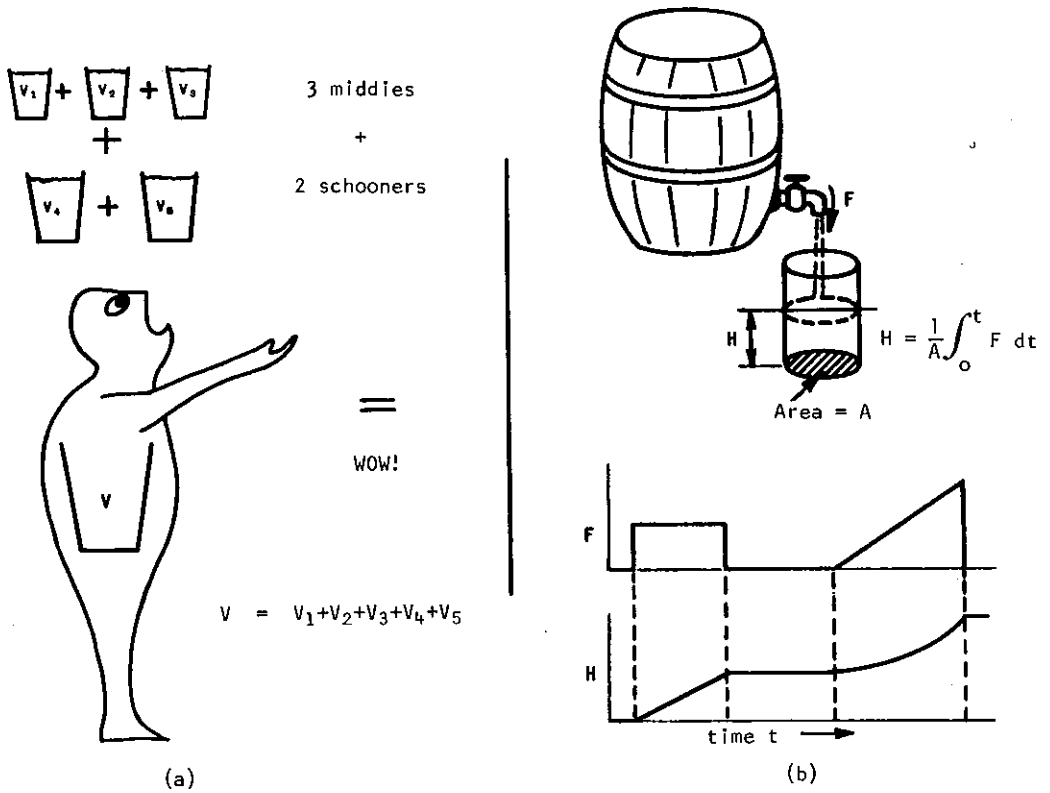


Figure 5.2 Liquid analogues for (a) addition, using volumes, and (b) integration

The more important process of *integration* can be achieved as shown in figure 5.2(b). 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/second) as determined by the tap:

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

These simple analogues would be of little use in practice; they are slow, unsuitable for interconnection, and the variables are difficult to measure accurately. However, there are much better analogue processes available, the most useful of them using an electronic amplifier as described in the following section.

5.2 ELECTRONIC ANALOGUE COMPUTER COMPONENTS

There are only three analogue components that we need to understand at this stage, the first being an electrical component, the others being based on the use of electronic 'operational' amplifiers.

The *potentiometer* is simply a means of reducing the size of a voltage by an amount which can be set very accurately (0.01 per cent). The circuit and conventional representation are shown in figure 5.3(a)

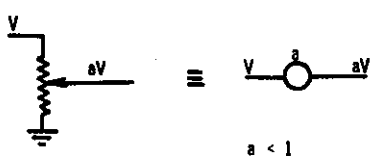
in which the input voltage V is reduced to aV at the output, with $0 \leqslant a \leqslant 1.0$. The potentiometer is the only adjustable element in an analogue computer.

The second component to be considered is the *adder*, whose output is the algebraic sum of a number of input voltages. The output voltage is inverted, *i.e.* a positive input gives a negative output and *vice versa*. Each individual input may be increased by a factor of ten if desired, but unless defined otherwise this gain is assumed to be unity.

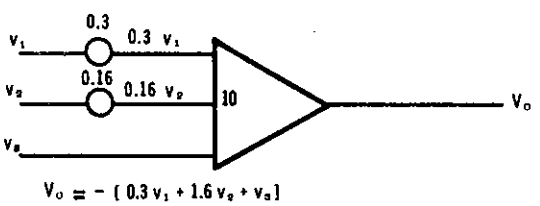
Figure 5.3(b) shows the conventional representation of an adder which, with the potentiometers, gives the relationship

$$V_o = - [0.3\ v_1 + 1.6\ v_2 + v_3] \qquad (5.2)$$

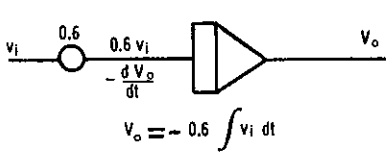
Note that all voltages are measured with respect to earth although the earth connection is not shown. The voltages may remain at a fixed level for many seconds, or may vary at any frequency up to about a MHz.



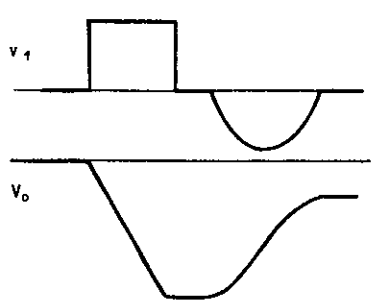
(a) A potentiometer



(b) An adder, using an 'operational' amplifier



(c) An integrator, using an 'operational' amplifier



(d) Integrator wave forms

Figure 5.3 The conventional diagrams for the main computing components

Finally, the most important component is the *integrator*, shown in figure 5.3(c). Like the adder, it inverts and may increase the input voltage by a factor of ten if desired. For the circuit shown, the performance is defined by:

$$V_o = -0.6 \int_o^t v_1 \, dt$$

or
$$-\frac{dV_o}{dt} = 0.6\, v_1 \quad . \qquad (5.3)$$

Thus a constant value of v_1 causes the output voltage to change at a constant rate (figure 5.3(d)), a sine input gives a cosine output and so on. More than one input voltage can be applied, the output then being minus the integral of the sum of the inputs.

Accuracies better than 0.1 per cent can be obtained without difficulty using analogue components of the type discussed.

5.3 ELECTRONIC ANALOGUE COMPUTERS

5.3.1 General

An electronic analogue computer consists of a number of components suitable 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, price, temperature 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-size machine might have about 100 operational amplifiers, including perhaps 30 integrators, and could thus perform 30 integrations at the same time. Also, most analogue computers have a number of logic elements such as gates, monostables, *etc*. This *patchable* logic can be interconnected to suit any given problem.

5.3.2 Equation Solution

Consider the circuit of figure 5.4. 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, and at this time the output V = b. Because of the potentiometer, the integrator input is λV, and so the circuit obeys the equation

$$-\frac{dV}{dt} = \lambda V \qquad \text{or} \qquad \frac{dV}{dt} = -\lambda V \quad . \qquad (5.4)$$

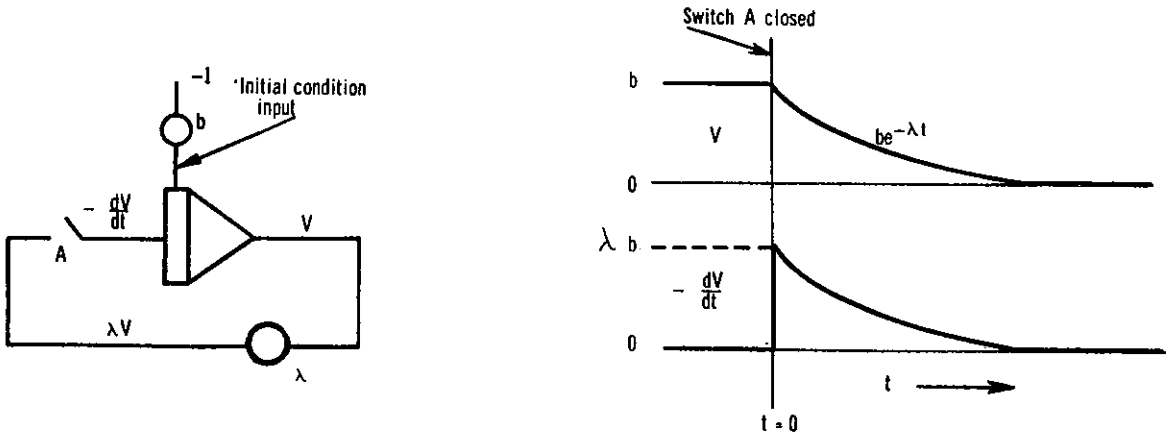


Figure 5.4 A circuit for the solution of $\frac{dV}{dt} = -\lambda V$ and typical waveforms. The input via b only provides the starting voltage

This is of the same form as equation 5.1, which describes the systems of figure 5.1, and so the circuit of figure 5.4 is simply one more analogue, having the same dynamic properties as the other three systems. As you know from a previous lecture, the solution to equation 5.4 is an exponential: if we let $V = ke^{-\lambda t}$, where k is an unknown constant, then differentiating we get

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

which is the same as equation 5.4. This demonstrates that $V = ke^{-\lambda t}$ is *a* solution of equation 5.4. Since we have made V = b at t = 0, substitution shows that k = b, and so *the* solution is $V = be^{-\lambda t}$.

The circuit of figure 5.4 'solves' equation 5.4 by producing a voltage proportional to $be^{-\lambda t}$ each time switch A is closed. V starts off positive and, *via* the integrator, forces itself to get smaller. As it does so, its rate of change also gets smaller, which is precisely what equation 5.4 tells us in a more compact way.

Switches such as A and many other controls required by the computer are usually omitted from the computing circuit diagram - their presence is assumed. In fact, closing switch A is equivalent to switching the computer from the *INITIAL CONDITION* mode to the *OPERATE* mode.

Summarising, if we should wish to examine one of the systems of figure 5.1, possibly with a very complicated series of disturbances, the simplest and most accurate way of doing the experiment would be to apply

a voltage representing the disturbances to the circuit of figure 5.4.

5.4 PROBLEM SOLUTION

To obtain the above solution, we started with a computer circuit and *analysed* its behaviour. The usual process is the other way round - we are given a problem situation and have to *design* a circuit which will simulate it, resulting in the process shown in figure 5.5. 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, provides considerable insight into the operation of the original system. In fact, one major advantage of simulation is that it provides an effective means of learning and, where the problem system is understood in detail, simulators can be made to behave so realistically that they are used to train operators. Such training simulators are widely used in the nuclear and aircraft industries and, although originally they used analogue computers, they are now more frequently based on digital computers.

Even in cases where the original problem is not well-understood (such as the Summer School topic), it is still a great advantage to be able to examine a model in detail, and rapidly explore the effects of changing the problem parameters. A simulator is ideal for this task.

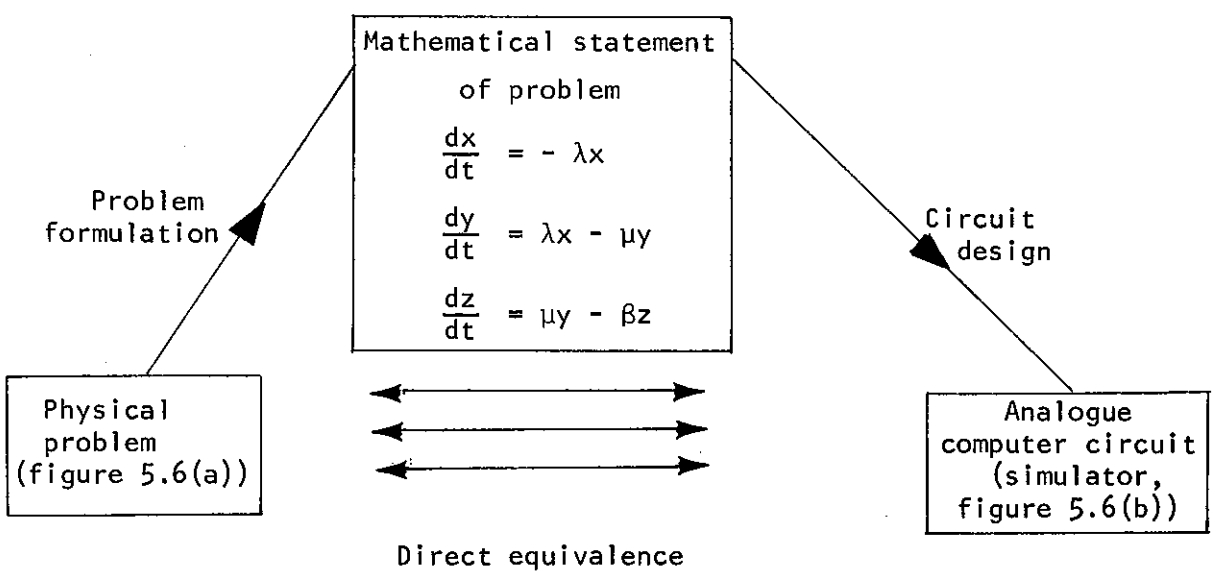


Figure 5.5 Pictorial representation of the analogue method of problem solving

As an example of the process of figure 5.5, consider the series of tanks in figure 5.6(a), each having a drain hole through which it leaks into the next tank. The depth of water in the first tank is x, and the surface moves (or the depth changes) with velocity dx/dt, which depends on the flow of water in and out. For the first tank the inflow is zero, although the experiment starts (the plug is pulled out, or the computer is switched from *INITIAL CONDITION* to *OPERATE*)with an initial

depth x_o. The outflow depends on the size of the hole, denoted by λ, and the depth (*i.e.* pressure) of water. Thus

velocity of surface = inflow - outflow

i.e. $$\frac{dx}{dt} = 0 - \lambda x \quad ,$$

or $$\frac{dx}{dt} = -\lambda x \qquad (5.5)$$

with the initial condition $x = x_o$ at $t = 0$. By comparison with equation 5.4, we know that x will fall exponentially (and we have found one more analogue of the systems of figure 5.1).

However, the second tank, which starts empty, has an inflow from tank 1 as well as the normal outflow; its rate of change of depth is thus

$$\frac{dy}{dt} = \lambda x - \mu y \quad , \qquad (5.6)$$

with $y = 0$ at $t = 0$. Similarly for the third tank

$$\frac{dz}{dt} = \mu y - \beta z \quad , \qquad (5.7)$$

with $z = 0$ at $t = 0$, and one could go on indefinitely.

This system of tanks gives a very clear idea of how one radioactive material decays into another, which itself decays (at a different rate) into a third, because the equations describing that situation are identical to equations 5.5 to 5.7, *i.e.* the two systems are analogous.

Our simple exponential solution only fits the first tank, the change in depth in the others being complicated by their varying inflow. However, we have successfully *formulated* the problem of figure 5.6(a), and can now go on to design the computer circuit.

From figure 5.4, we know that we can solve equation 5.5, using integrator 1 of figure 5.6(b) to represent tank 1. The potentiometer introduces λ, the size of the drain hole (or the decay constant of a radionuclide).

Consider now equation 5.6. Let us assume that a signal representing $-\mu y$ is available; then with the existing λx signal we can make up dy/dt. This is integrated (and inverted) in integrator 2 to give -y, and so we can supply the wanted $-\mu y$ signal from the potentiometer. The circuit of integrator 3 solving equation 5.7 can be found in exactly the same way except that the signs are all reversed, and so we have developed an analogue computer circuit to simulate the water levels in the three

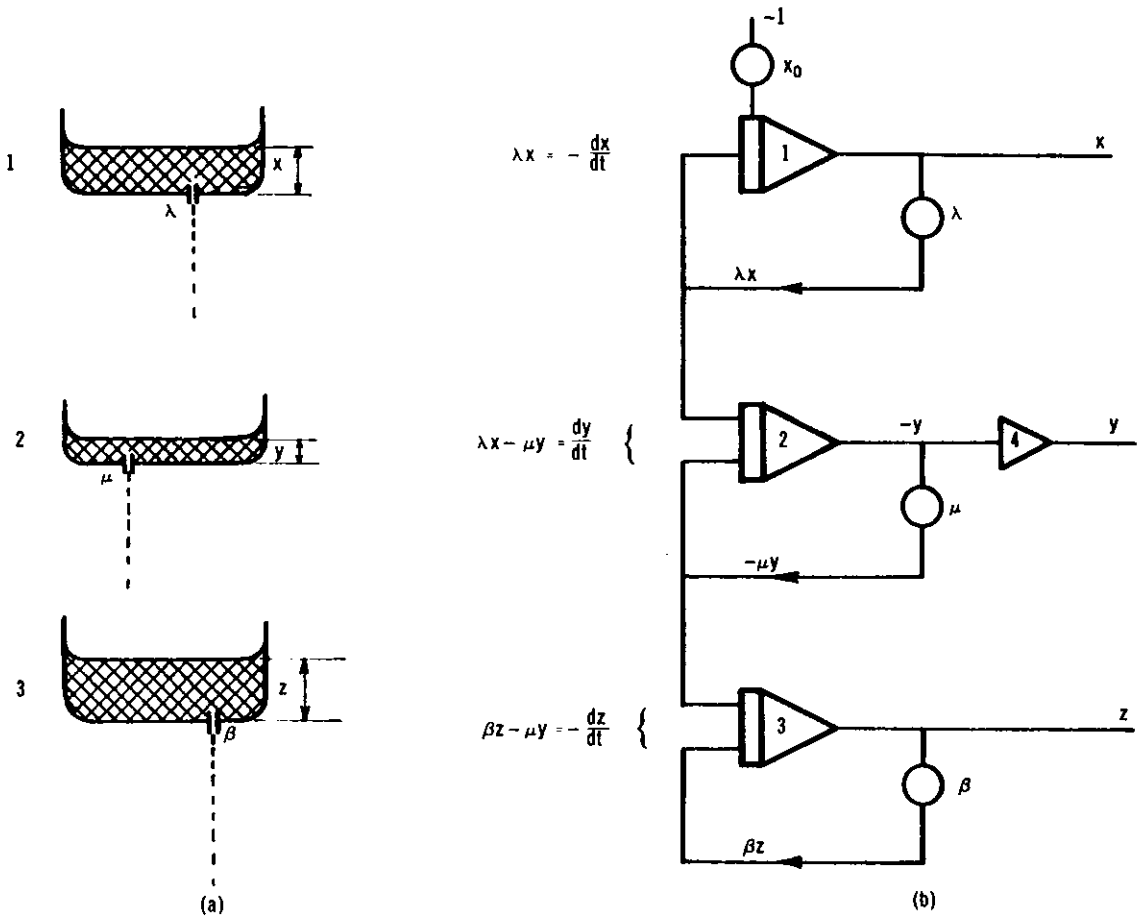


Figure 5.6 (a) A problem in hydraulics
(b) A simulator to represent the problem

tanks. An inverting amplifier (4) allows y to be viewed the right way up.

Notice from the demonstration that all the computing operations occur simultaneously (in parallel) not in sequence as in a digital computer, and that the solution arises at a definite speed, *i.e.* the same speed as the levels move in the tanks in our case. By using smaller capacitors in the integrators, the problem can be solved at least 1000 times faster and, if many integrations are involved, the overall operation is much faster than can be achieved by a digital computer. However, this high speed cannot be properly utilised by a human operator.

5.5 HYBRID COMPUTERS

A fairly recent development is the *hybrid computer*, which consists of an analogue computer, a general purpose digital computer, and an

interface. The latter provides the facilities required for the two machines to cooperate effectively (figure 5.7).

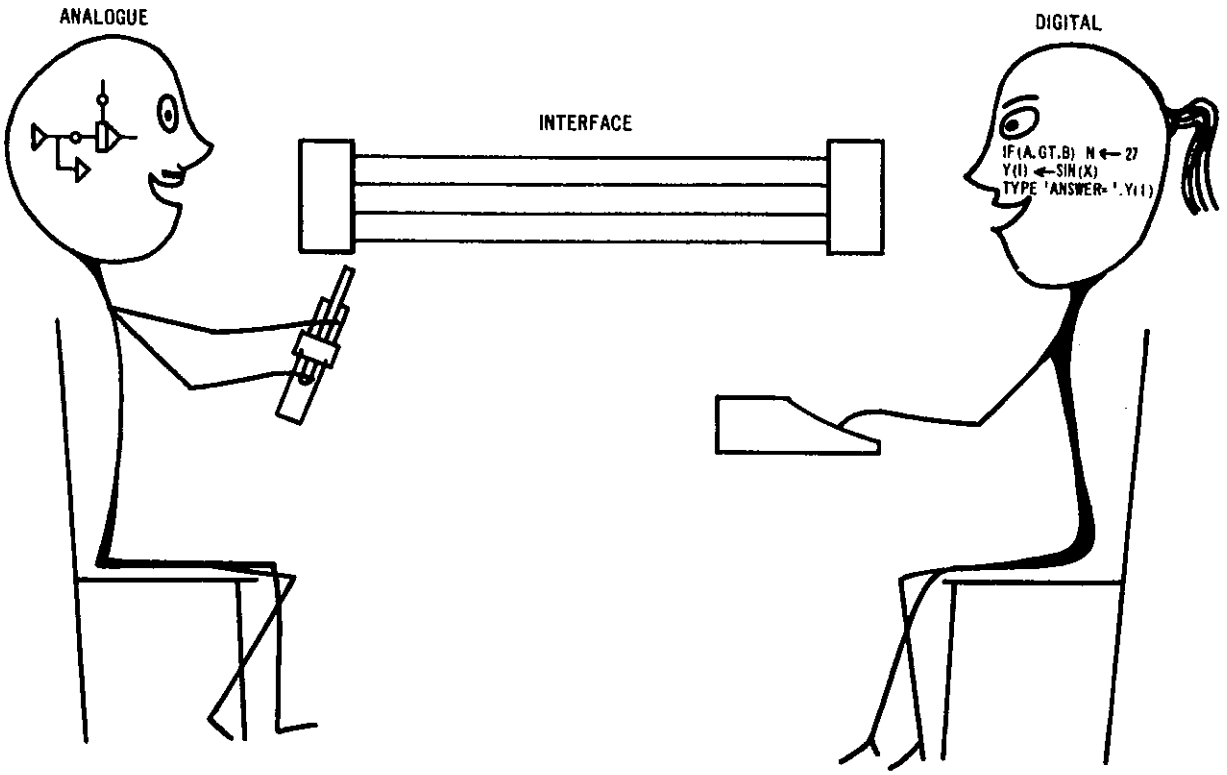


Figure 5.7 A hybrid computer

The analogue computer allows high speed, parallel computation. The digital computer can be programmed to:

- Perform the scaling calculations, and check the connections of the analogue circuit and the settings of the potentiometers.
- Act as a very high speed operator which readjusts the computer before each solution, as determined by the preceding solution.
- Perform parts of the computation which the analogue computer finds difficult.

One might also express the same idea by saying that the analogue computer becomes one of the peripherals upon which the digital computer can call when required.

As an example, suppose we wanted to find the value of λ for an experimental result thought to be an exponential. The operator could use the circuit of figure 5.4 and, by comparing the output with the wanted curve, adjust the potentiometer to get a better fit.

After a number of trial and error solutions he could get a reasonable match and the value of λ would be given by the potentiometer setting. This process would be tedious and inaccurate when performed manually. However, with the digital section of the hybrid computer performing the comparison and resetting the analogue section, many trial solutions would be performed in one second, and an accurate result could be obtained very quickly.

Another application of a hybrid computer is described in the next section.

5.6 SIMULATION OF THE USAGE OF A LIMITED RESOURCE

5.6.1 Representation

In earlier lectures, a model has been introduced which represents the consumption of oil between the years 1910 and 2050 in a simple fashion. The equations are restated here as:

$$\frac{dx}{dt} = \lambda\, x(1-x/x_\infty)\, p_b \qquad (5.8)$$

$$p_b = (p_o/p)^\beta \qquad (5.9)$$

where x = total quantity of oil consumed up to time t in gigabarrels (10^9 barrels),

x_o = consumption up to the end of 1909 or beginning of 1910,

x_∞ = total quantity of oil available for consumption,

p = price of oil at time t in constant 1982 US dollars,

p_b = price function at time t,

p_o = price in 1909, also taken as the price in 1910,

λ = 0.07, and

β = 0.07.

The equations will be solved in the hybrid computer system of figure 5.8. This circuit contains several unfamiliar components whose main properties will now be described.

The first is an analogue multiplier (unit 03) which is simply a device which accepts two input voltages v_1 and v_2, and generates a third voltage whose value is given by

$$v_3 = v_1 \times v_2/10.$$

All the other new components are parts of the interface of figure 5.7. The function of an analogue-to-digital converter (ADC) is to accept an analogue input voltage, measure its value, and then to represent the latter as a pattern of bits that the digital computer will recognise.

For instance, the value of x_∞ can be set on potentiometer 02 in figure 5.8, and *via* ADC1 this value can be read by the digital computer when instructed to do so by the program.

The digital-to-analogue converter (DAC) performs the reverse operation. A value of price (p) calculated by the digital program can be loaded into DAC3; this subsequently produces a voltage proportional to *minus* the price which can be fed to the analogue circuit. This voltage (-p) is maintained until the DAC is reset to a different value by the program.

The last new component combines the properties of two of the earlier ones. It is a multiplier, but only one input is an analogue voltage; the other comes from the digital computer *via* a DAC, and so the device is known as a digital-to-analogue multiplier (DAM). Thus in the figure, the digital computer program calculates $-1/x_\infty$ and loads it into DAM5: conceptually this is converted into a voltage $(+1/x_\infty)$ and multiplied by the analogue input (-x) to give an output of $-x/x_\infty$. The input x may vary at any time, but x_∞ is held constant until reset by the digital program.

5.6.2 Digital Computer Operation

In this model, the price of oil is known from 1910 to 1982, and is predicted from 1983 to 2050; there are four ways of determining price (see Appendix 6C), and the value of M which defines the method to be used is requested by the program. The historical price data are fed to the computer together with the program, and the values of β and x_∞ are read *via* the ADCs, as adjusted by the operator from time to time.

Price itself can be used in the analogue computation, but it is more convenient to work out the price function p_b for every year, and store the values away in a table from which they can be retrieved one at a time. The values of price itself are stored in another table and are fed to the analogue circuit simply because it is easier to measure and record them that way.

Summarising then, the digital program works out values of price and the price function for every year, and feeds them to the analogue circuit when requested. Also, it switches the analogue circuit in and out of *OPERATE* mode, either under the operator's direct control, or on a repetitive basis.

5.6.3 Analogue Computer Operation

When the computer circuit is switched from the *INITIAL CONDITION* to the *OPERATE* mode, the output of integrator 00 will have the initial

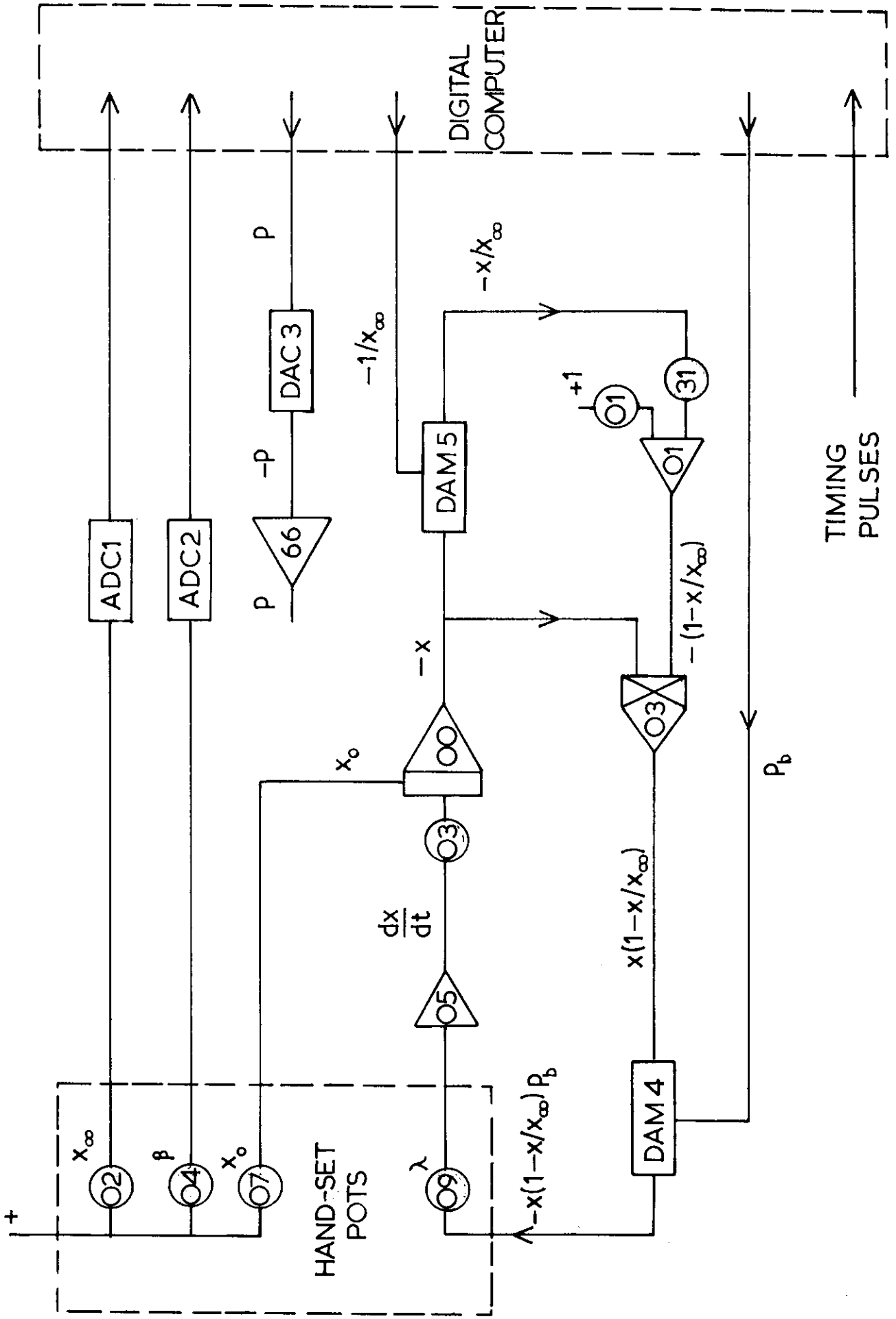


Figure 5.8 Hybrid computer circuitry

value $-x_o$ as set by potentiometer 07. This voltage is multiplied by $1/x_\infty$ in DAM5, and combined with a fixed voltage in adder 01 to give $-(1-x/x_\infty)$. The latter is multiplied by $-x$ in multiplier 03 to give $x(1-x/x_\infty)$, and then by p_b in DAM4 to give the right-hand side of equation 5.8, apart from the λ term. This is introduced in potentiometer 09 to provide $\frac{dx}{dt}$ at the output of amplifier 05, and hence at the input of integrator 00. This ensures that, starting from $-x_o$, the output of integrator 00 is the solution of equation 5.8 (or more precisely, *minus* the solution). The value of $\frac{dx}{dt}$ is the *rate* of oil consumption of course.

The analogue circuit is time scaled so that one second represents ten years of problem time. Thus the analogue computer starts in '1910', and precisely ten times a second it sends a synchronising pulse (derived from a crystal oscillator) to the digital computer. The latter responds to each pulse by feeding back values of p and p_b for the 'year' just starting.

After a total of 14.1 seconds the year 2050 has been reached, and the digital computer switches the analogue computer out of *OPERATE* mode. In addition to the values of p and p_b which the digital computer sends out once a 'year', it also sends the value of $-1/x_\infty$ once per solution. Also, if β or x_∞ are adjusted by the operator at any time, the digital program recalculates its stored values of p and p_b when the analogue computer stops at the end of a solution.

The circuit outputs of interest can be observed on a voltmeter or an oscilloscope, or drawn out by a chart recorder to give a permanent record.

This system will operate very much faster (ten years in one millisecond, say) and in this case it must be run repetitively and have its outputs observed as stationary traces on an oscilloscope. Adjustment of the various parameters gives an immediate change in the output pattern. To obtain a permanent record, any particular response can be photographed on the oscilloscope, or the computer can be slowed down to enable the recorder to be used.

Naturally, the simulation can be stopped to enable M to be changed to give a different method of price prediction.

5.7 EXPONENTIALS AND EXPERIENCE

5.7.1 Increasing Exponentials

In earlier sections of this chapter, we have talked mainly about quantities which decay exponentially ($e^{-\lambda t}$) since these are very common

in natural processes. However, it is quite possible for λ itself to be negative (corresponding to a drain hole which squirts water *into* a tank), and so we end up with e^{kt} where k is positive, giving an increasing exponential of the type shown in figure 5.9. (It was shown earlier that this would occur if there were an unlimited amount of oil, available at a fixed price.)

Clearly such a response cannot continue indefinitely; it *must* stop somewhere, and in practice tanks overflow, structures break or reactors melt if such a process goes on for too long. Our oil consumption model was constructed so as to take account of limited oil resources, and so this type of process does not arise.

Compound interest on a sum of money gives exponential growth; 5 per cent compound interest leads to a 'doubling' time of about 14 years, although in this case the transient is usually terminated by the withdrawal of the money from the bank.

5.7.2 Populations

Consider a colony of 100 grubs. Given sufficient food and space, an average of 20 eggs are produced by each grub per month, of which 10 eggs hatch out and produce grubs which survive to the egg laying stage. Remembering that the original grubs die at the end of the month, the grub population increases by a factor of ten each month (figure 5.9).

Months	0	1	2	6	12	n
Population N	10^2	10^3	10^4	10^8	10^{14}	$10^{(n+2)}$

(If the grubs are each 1 millimetre long, 10^{14} of them placed end-to-end would stretch for 10^8 kilometres! The moon is only 4×10^5 km away.)

The population curve can be fitted exactly by the equation

$$N = 100\ e^{2.3t} \quad ,$$

where t is in months, and so the original differential equation must have been

$$\frac{dN}{dt} = 2.3\ N$$

with $N = 100$ at $t = 0$.

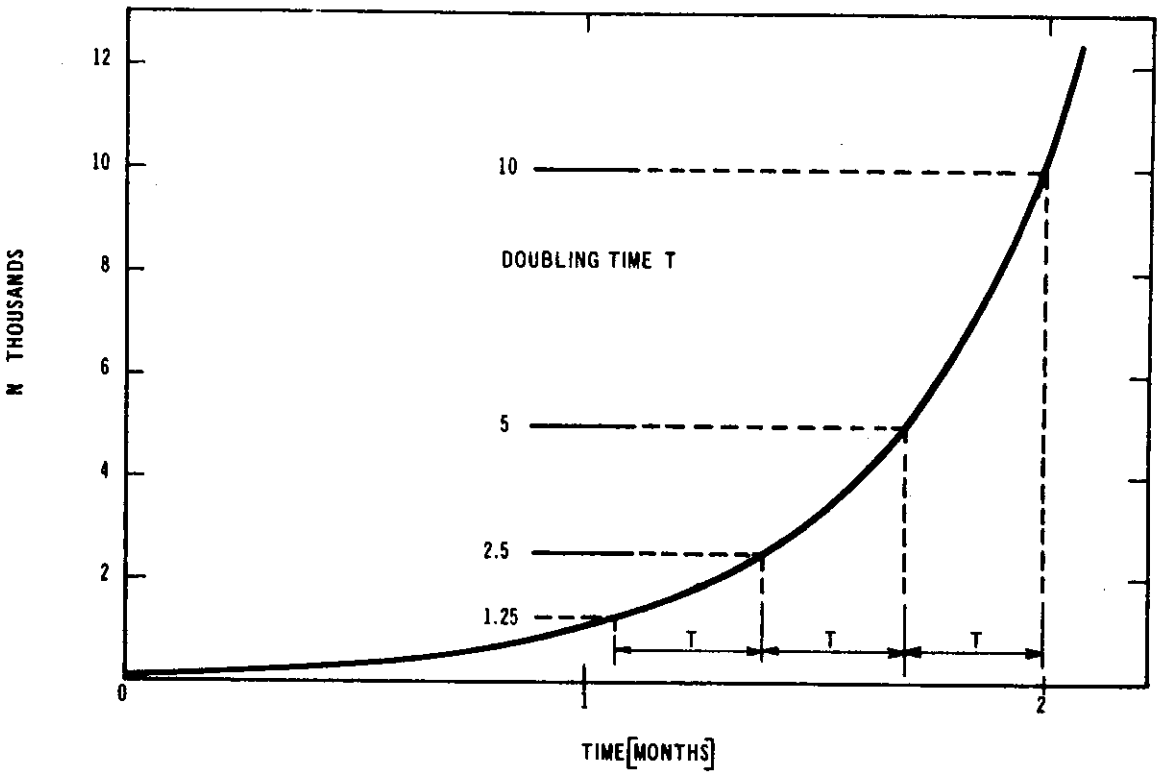


Figure 5.9 The exponential $N=100e^{2\cdot3t}$ showing the growth of a population of grubs

Thus the grub population is expanding exponentially with a doubling time of about nine days.

The most frightening thing about such an increase is its insidious speed. If you only observe the past (which is usually all that we are able to do), it is difficult to realise just how quickly things will change in the future, and any delay can turn a difficult situation into an impossible one. Fortunately for us, most natural populations run out of food or space, or reach some other limitation, possibly of the type discussed next, a predator-prey situation.

5.7.3 Sharks and Little Fishes

The fish population is similar to the grub population: there is plenty of food and the oceans are large, so their numbers would increase exponentially, were it not for the sharks who eat fish. The rate of change of the fish population F is

$$\frac{dF}{dt} = k_1F - k_2F.S$$

The first term on the right hand side represents the normal growth rate, and the second the rate at which fish are eaten, depending upon both the number of fish *and* the number of sharks.

For the sharks,

$$\frac{dS}{dt} = - k_4 S + k_3 F.S$$

The first term accounts for the sharks who die or who leave the area simply because there are too many other sharks around already; the second term represents the shark's birthrate, which again depends upon the product F.S (the number of shark parents, and the food supply). While this model is oversimplified it has some interesting properties, including cyclic, or oscillatory behaviour, alternating between famine and plenty (for the sharks).

5.7.4 The Really Super Important Problem

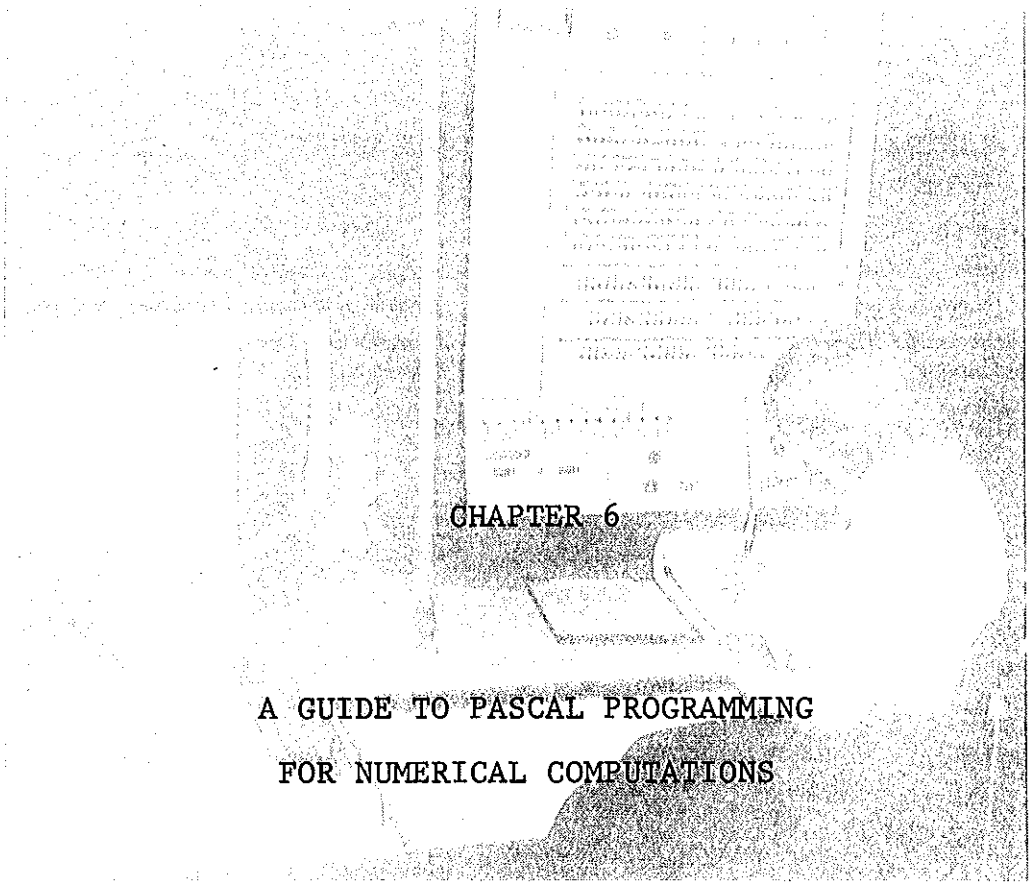
The one population which has been encouraged to expand unchecked is the human population. For over 300 years it has been growing *more* than exponentially, *i.e.* initially with a doubling time of 250 years, but now, at a level of over 4.4 gigapersons, with a doubling time of only 33 years.

Not only this but, as illustrated in chapters 1 and 2, the human race is using up non-renewable resources (oil, coal, metals, *etc.*) and generating pollution at rates which are also growing more than exponentially, both because of the rising number of people and because of a rising material standard of living. Clearly this type of growth cannot continue very much longer or there will be no room left, insufficient food and few raw materials.

In view of what we know of exponentials, it is clear that the human race must manage its affairs better in the future by finding ways of limiting both the population and its usage of the world's resources to levels that our planet can sustain. Unless this is done, nature will apply one of her own unpleasant methods of limitation - famine and disease, probably preceded and accompanied by war.

Also we know that every delay in coming to grips with the problem makes matters worse - in fact a delay of only about 30 years doubles the size of this problem.

The past four or five generations have worked hard to provide food and better material standards of living, and so have helped to increase the population and accelerate the use of natural resources. The present generation is continuing to do this, owing to sheer inertia and bewilderment, but at least it has realised that a serious problem exists. It will be the responsibility of the next generation to start dealing with these formidable difficulties.



CHAPTER 6

A GUIDE TO PASCAL PROGRAMMING FOR NUMERICAL COMPUTATIONS

Lecture by

J.M. BARRY

CONTENTS

6.1 INTRODUCTION

Each digital computer is capable of obeying a number of basic instructions. These instructions vary for different computers, but they have several common attributes:

(a) The ability to perform the four arithmetic operations (addition, subtraction, multiplication and division).

(b) The ability to perform logical operations (is A $\geqslant$ B?).

(c) The ability to perform 'housekeeping' instructions (*e.g.* moving numbers from addressable memory 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 to develop programs in the basic machine language of the specific computer. In the early days of computing, it was necessary for scientists and mathematicians to concern themselves with the intricacies of binary coding. The long delays and inconvenience of this form of user-machine communication accelerated the growth of programming languages that the problem solver could use more readily. Many languages (FORTRAN, BASIC, COBOL, ALGOL, PLI, APL, ACL, Pascal, *etc.*) have been developed for scientific, commercial and other applications. Pascal is chosen as the main vehicle for problem solving at this Summer School because it reflects modern developments in language structure. There are no computers that obey programs written in Pascal directly. It is usually necessary for programs in 'high level' languages such as Pascal to be translated into an appropriate set of machine language instructions. This process is known as *compilation*.

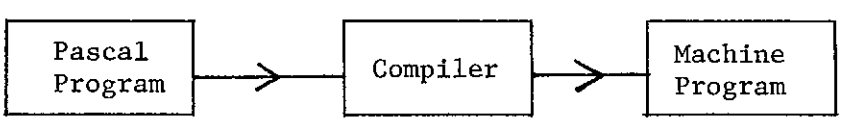


Figure 6.1 - Compilation of a Pascal program

The Pascal source statments are translated to a set of machine language instructions by a Pascal compiler (figure 6.1). The compiler is itself a program (usually supplied by the machine manufacturer, but in this instance we will be using a locally developed one called Pascal 8000) that first checks to ensure that the Pascal 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 IBM3033 computer will not be our concern at this Summer School as we are interested primarily in using the computer as a tool for mathematical problem solving.

6.2 OVERVIEW OF PASCAL PROGRAMMING

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

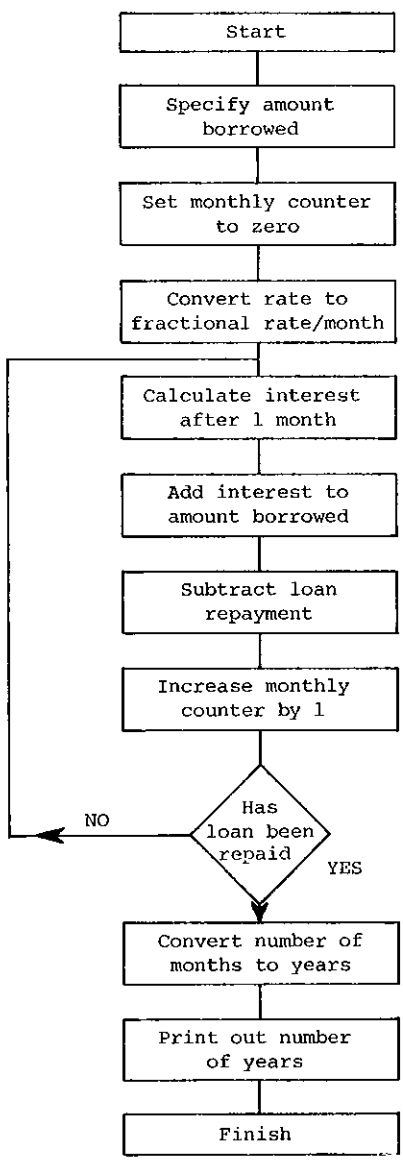


Figure 6.2 Flow chart for compound interest problem

Problem If $30 000 is borrowed at a rate of 13% (monthly reducible) and repayments of $400 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 detail logically the steps that are needed to solve the problem, in much the same way as we would if we were going to tackle the problem with a desk calculating machine, slide rule, or pen and paper. Some people find it helpful to draw a flow chart (figure 6.2) showing the steps involved, whereas others prefer to visualise all the steps in their mind. 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 6.3).

```
(* PROGRAM BY J.M.BARRY TO DETERMINE THE NUMBER
   OF YEARS IT WILL TAKE TO REPAY A LOAN.
   THE PRINCIPAL BORROWED, INTEREST RATE AND MONTHLY
   REPAYMENT ARE TO BE READ FROM A PUNCHED DATA CARD
*)

PROGRAM  REPAYMENT(INPUT,OUTPUT);

         VAR  PRINCIPAL,RATE,PAYMENT,
              NO_OF_YEARS,
              INTEREST_RATE,MONTHLY_INTEREST   : REAL;
              MONTH_COUNTER                    : INTEGER;

BEGIN
READLN (PRINCIPAL,RATE,PAYMENT);
MONTH_COUNTER:=0;
INTEREST_RATE:=RATE/(100*12);

WHILE  PRINCIPAL>0  DO

  BEGIN
  MONTHLY_INTEREST:=PRINCIPAL*INTEREST_RATE;
  PRINCIPAL:=PRINCIPAL+MONTHLY_INTEREST;
  PRINCIPAL:=PRINCIPAL-PAYMENT;
  MONTH_COUNTER:=MONTH_COUNTER+1
  END;

NO_OF_YEARS:=MONTH_COUNTER/12;
WRITELN(' NUMBER OF YEARS = ',NO_OF_YEARS)

END.
```

The data card sufficient for this problem would be

```
30000.0 12.0 400.0
```

Figure 6.3 Sample program and data for compound interest problem

After obtaining some idea of how programming fits together, we shall investigate the rules behind the use of the Pascal language. Because of the limited time at our disposal during the Summer School it is necessary to omit many of the details of the Pascal language, particularly those concerned with character manipulation and more complex data types. Despite this constraint, sufficient details are given in these notes for the student to tackle most numerical computational problems.

6.3 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. During the tutorial sessions, coding sheets will be supplied.

At the Summer School, input to the computer will be limited to the punched card medium. The following character set is available on the card punches at our disposal:

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

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

(iii) 13 special characters +,-,*(multiplication), /(real division), .,',(,),;,=,<,>,:; and

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

Pascal statements may be punched anywhere on the 80 column card, with more than one statement appearing on each card (if appropriate). Comments are enclosed between (* *) and may span several cards. They are listed by the computer but otherwise ignored.

6.4 ARITHMETIC DATA

We will treat two different types of constant sufficient for handling data (numbers) in most scientific problems:

(a) *INTEGER* (or fixed point) constants:
a whole number without a decimal point whose absolute value is $\leq 2^{31} - 1 = (2147483647)$.
Valid integer constants: 0 -5 +357 7005192
Invalid integer constants: 27. 5,132 9812735997 27.0

(b) *REAL* (or floating point) constants:
up to sixteen decimal digits with a whole number part, a decimal point, a fractional part and an optional exponent. The absolute magnitude is approximately 10^{-78} to 10^{75}.

Valid real single precision constants:

5.0 0.6 +0.61 7.91 5.3E+2(=5.3 x 10^2)

5.3E2(5.3 x 10^2) -0.051E-03 (-.051 x 10^{-3})

Invalid real single precision constants

1 3,471.2 1.E 6. .7

Distinctions are carefully drawn between the two types of constant for electronic rather than mathematical reasons. The electronic 'hardware' necessary for *INTEGER* arithmetic operations is less complex and consequently for most machines is 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.

In addition to arithmetic data Pascal supports other forms, for example character type and even more complex structures and types of your own making. Such data support is advantageous for non-numerical computing, however, it is not essential for solving the Summer School problem, so discussion is avoided here.

6.5 VARIABLES

A variable is a symbolic name used to identify a data item that will occupy a location of directly addressable 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:=0.0
```

This 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 is interpreted as the assignment of the result of the right hand expression to the left hand location. Consequently, an expression such as

```
A:=A+1.0
```

does not yield any algebraic result but rather is interpreted as increasing the old value associated with A by 1.0 to give a new result also called A. The old value of A is, of course, lost.

Variable names may be of any length, however, only the first eight characters are significant, *i.e.* variable identifiers with the same first eight characters are considered to be the same identifier. Variable identifiers may be composed of letters or digits but the first must be a letter. Valid variable identifiers are

TIME , X3B , I5 , T .

For ease of identification, very long variable names such as

THISISALONGVARIABLE

may be punched as

THIS_IS_A_LONG_VARIABLE

The underscore '_' is not part of the variable name itself, it is ignored by the Pascal compiler and merely serves to assist in understanding the program.

For our purposes, we restrict the use of variables to *REAL* or *INTEGER* form. All variable names and their type must be defined through use of the declaration command VAR.

```
VAR     X,Y,Z     : REAL;
        I,COUNTER: INTEGER;
```

Unlike some other languages there are no default options as to the type of variable. All variables used must be declared, and while this may seem annoying at first sight, it reduces possible confusion when the body of code is written. For example it avoids hours of wasted effort spent in debugging a program when TIME, for example, is incorrectly punched in as TlME (a not infrequent error).

Variable names may also be used with subscripts in Pascal. Such variables can be used to represent vectors and matrices which you may have encountered already in your mathematics courses:

V[3] or V(.3.) is the Pascal representation of the vector component v_3,

A[3,4] or A(.3,4.) is the Pascal representation of the matrix element a_{34}.

Unfortunately, the symbols [,] are unavailable on the card punch machines, so the alternative notation is necessary should you wish to use subscripted quantities. (Further discussion of SUBSCRIPTED variables is delayed until section 6.12.)

6.6 SIMPLE INPUT AND OUTPUT (I/O)

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

```
X:=6.3
```

Should we wish to alter the data on which a program is to operate without changing the program itself (a most frequent requirement, particularly when non-programmers are going to prepare data to run someone else's program), then a READLN statement may be used.

The Pascal procedure READLN may be invoked to read values from any specified input file into a set of listed variables. The input file (along with an output file) is identified through the program declaration, *e.g.*

```
PROGRAM REPAYMENT(INPUT,OUTPUT);
```

For the Summer School, the declaration INPUT refers to a set of punched data cards. (These are separated physically from the program cards. The program cards are positioned first in the deck of cards, and a special separator card, which we supply, informs the computer of the division.)

The READLN procedure initiates the reading of a list of data items. In our case these are on punched data cards.

```
READLN(PRINCIPAL,RATE,PAYMENT)
```

causes three numbers to be read and stored in the variables PRINCIPAL, RATE and PAYMENT. After the third item has been transferred, a subsequent read will commence with a new input record (card), so any additional information which may have been punched on the last card is lost. For the loan repayment problem in section 6.2, the three data items are on the same data card; the items are separated by at least one blank character. It is possible, however, to have the data on a number of input records. In this case, sufficient input records are read so that three data items are transferred. It is important that each data item corresponds to the type of variable for which it is intended.

At this Summer School, the declaration of OUTPUT in the previous program refers to the line printer. Messages and output from your program may be directed to the line printer by the WRITELN procedure. For example the statement

```
WRITELN ('NUMBER OF YEARS =', YEARS)
```

would display

```
NUMBER OF YEARS =   1.1666666666666666E+01
```

As you can see, this form of output, known as free-format output, has certain advantages in that the Pascal rules for its use are easily understood. There is not a great deal of control over the layout and an untidy form emerges frequently. This is unfortunate, particularly when printed output for publication is required. This free-format form of output is sufficient, however, for you to handle the Summer School problem. For those wishing to know a little more about I/O control to obtain 'prettied up' output see Appendix 6D.

6.7 ARITHMETIC OPERATIONS AND EXPRESSIONS

We consider six of the arithmetic operations available to Pascal users:

(i)	addition	+	*e.g.* A+B	
(ii)	subtraction	-	*e.g.* A-B	
(iii)	multiplication	*	*e.g.* A*B	
(iv)	division (integer)	DIV	*e.g.* A DIV B	
(v)	division (real)	/	*e.g.* A/B	
(vi)	exponentiation	**	*e.g.* A**B	a^b

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 operators 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) * / DIV left to right precedence
(iii) + - left to right precedence.

Consequently, the expression

```
X+(Y/A)-(3.0*U)+P*(S**4)/3.0
```

could have been correctly abbreviated to

```
X+Y/A-3.0*U+P*S**4/3.0
```

The integer variables or constants deserve special attention. Integer division of one integer by another results in the truncation of any fractional remainder.

```
VAR   I,K : INTEGER  ;
I:=9;
K:=I DIV 2
```

would result in K taking the value of 4. This property can often be exploited to the programmer's advantage when testing for even integers;

```
K:=I - I DIV 2 *2
```

would assign 1 to K if I is odd, and 0 if even. The expression I/2, however, would be performed as a real division although in this case both operands are of integer type.

If an expression consists of different types of operand, the mode of the result is determined by the type of operand at various stages of the calculation. With regard to integer and real modes, the latter is considered to be the higher mode, *e.g.*

```
VAR   X:REAL    ;
      Y:INTEGER;
```

the expression X*Y would be evaluated in the higher mode and a real result computed. For

```
X*(Y+3)
```

the addition is done in the integer mode and the multiplication is performed in real mode. In assignment statements, the mode on the left should be the same as that on the right, with the exception that it is possible for an integer quantity to be assigned to a real variable.

```
X:=Y
```

is permitted, however

```
Y:=X
```

is not accepted by Pascal.

Should you wish to assign a real quantity to an integer variable the TRUNC (or ROUND) functions (section 6.9)

```
Y:=TRUNC(X)
```

should be used to remove the decimal component.

6.8 STATEMENT SEPARATOR

Statements in Pascal may be separated by the use of a semi-colon

(;). The semi-colon does not form part of a statement itself and should not be interpreted as a statement terminator. Its function is to separate statements in a block of code or to separate one block from another. To someone starting programming, the subtlety of this distinction may seem puzzling to say the least! Should you be confused as to when a ; is required, remember that unnecessary separators are treated by the Pascal compiler as null statements and will not lead to any harm. Take great care, however, with use of inappropriate separation in the IF-THEN-ELSE construct (section 6.11).

6.9 SUPPLIED MATHEMATICAL FUNCTIONS

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

```
X:=EXP(T)
```

To use most supplied mathematical functions, 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, *e.g.*

```
A:=EXP(A-C)+SQRT(15.0)
```

The following table lists frequently required functions supplied by the Pascal compiler:

Mathematical Function	Function Name (Argument)
square root, $\sqrt{x}$	SQRT(X)
square, x^2	SQR(X)
exponential, e^x	EXP(X)
natural logarithm, log x (or ℓn x)	LN(X)
sine of an angle (in radians), sin x	SIN(X)
cosine of an angle (in radians), cos x	COS(X)
arctangent (result in radians), $\tan^{-1} x$	ARCTAN(X)
absolute value (real numbers), $\|x\|$	ABS(X)
truncation	TRUNC(X)
rounding	ROUND(X)

The way in which these functions work is clear enough except for the TRUNC and ROUND functions. TRUNC(3.7) returns 3 and TRUNC(-3.7) returns -3; ROUND(3.1) is 3, ROUND(3.9) is 4 and ROUND(-3.6) is -4.

In addition to the above standard functions of Pascal, we require the use of special functions to undertake some of the preliminary exercises at the end of these notes, and to solve the Summer School problem. These functions, which are not part of the Pascal compiler, are supplied by the Summer School organisers. Because they are additional functions, it is necessary for you to identify them by an appropriate declaration at the start of your Pascal program (explanation of such details will come in section 6.13).

Two of these functions necessary for the preliminary exercises are the random number generators. The first, RAND, generates a sequence of random numbers starting each time with the same number, so that the sequence generated can be repeated at a later time. It is a little like having purchased a printed book of random numbers (Yes! Such things are available.) and commencing each time at the start. This can be of use when developing and testing a new program, in which case a constant environment may be of assistance. The second function, RND, produces a random sequence of random numbers. You may find that this fits in more nicely with your basic intuition.

Both generators can be invoked in a similar manner:

```
Y:=RAND                    Y:=RND
```

Each results in a random number being assigned to the variable Y, where the random number lies in the range

$$0 < y < 1.$$

Unlike the previous functions there is no need to pass an argument to the random number generator.

6.10 CONTROL STATEMENTS

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

```
PROGRAM LIST(INPUT,OUTPUT);

LABEL  1;
VAR  X  : REAL;

   BEGIN
1: READLN(X);
   WRITELN(X);
   GOTO 1
   END.
```

This program would cause cards with one number punched to be read and listed on the printer with no escape mechanism until the supply of data cards was exhausted, in which case an error condition would arise when the program attempted to read a non-existent card. At that point, the program would fail. Although the GOTO statement allows transfer of control, it is of little real use on its own.

Most early high-level computer languages, through their close reflection of the computer architecture, relied heavily on some form of GOTO statement. This was appropriate at the time because they appeared easy to understand and code. Much modern thought in language design is entirely opposed to the use of GOTO statements, largely because they make large programs difficult to follow and increase the possibilities of incorrectly specifying the required operations. This is particularly true for program systems developed in a team environment. Considering that today, computer program development involves more cost than the machine itself, it is not surprising that well structured, easy-to-follow and self-documenting programs are demanded. Pascal goes a long way towards satisfying these demands. It does, however, permit the use of a GOTO statement but its use is discouraged should another construct be available (which it nearly always is!). Consequently little more attention will be paid to the GOTO statement on its own or as an adjunct to the IF construct.

The value of a computer lies in its ability to repeat a set of statements until the task at hand is completed. A typical task might be to print out all the names and addresses from a data file. The statements to do this must be executed repeatedly until the required task is completed. A group of such statements is commonly referrred to as a loop. Three types of Pascal looping are considered:

(i) WHILE.

(ii) REPEAT.

(iii) FOR.

In the compound interest example, we saw an example of a WHILE statement. WHILE allows a block of statements to be repeated until a certain condition is satisfied. In this instance, the loop was repeated while the loan was not repaid (*i.e.* while PRINCIPAL > 0). When the principal was repaid, instead of the loop being repeated control passed immediately to the statement

```
NO_OF_YEARS:=MONTH_COUNTER/12;
```

following the end of the block, which is identified through the BEGIN ... END construct.

The WHILE statement is used when the number of times the loop needs to be repeated is not known in advance. In more formal terms the WHILE statement may be described

```
WHILE e  DO  s,
```

where e is a logical expression and s is a single or block of statements that are repeated while the logical expression e remains true.

Suppose we wish to evaluate and print out y given by

$$y = x^3 + x^2 + x + 1$$

for x = 1.,1.1,1.2,...,10. The following program would suffice.

```
PROGRAM EVALUATE_Y(OUTPUT);

    VAR  X,Y  :  REAL;

BEGIN
X:=1.0;

WHILE X<=10.0  DO

  BEGIN
  Y:=1.0+X*(1.0+X*(1.0+X));
  X:=X+0.1;
  WRITELN (' Y= ',Y)
  END

END.
```

The block form of a WHILE statement is shown in figure 6.4.

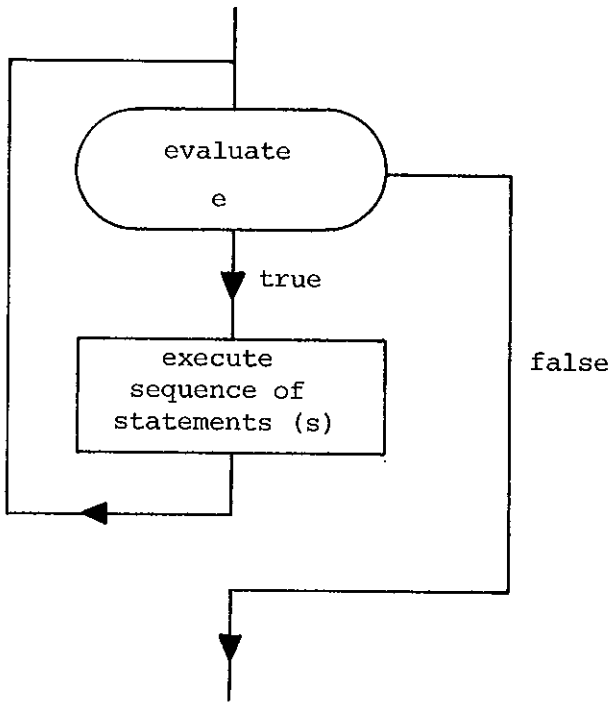


Figure 6.4 Schematic representation of WHILE

The REPEAT statement (figure 6.5) similarly offers a means of repeating a loop. The REPEAT statement has the form

REPEAT s UNTIL e

Here s represents a single statement or set of statements that are executed repeatedly until the logical expression e is satisfied. In schematic form, the REPEAT statement appears as

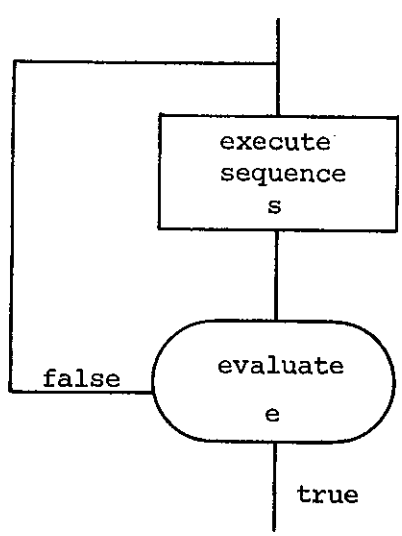


Figure 6.5 Schematic representation of REPEAT

Like the WHILE statement, the REPEAT statement is used when the number of times a loop must be repeated is not known in advance. It is argued that any task which can be performed with a WHILE statement may also be done with a REPEAT, hence a language such as Pascal need consider only one or the other form. There is an essential difference, however, between the WHILE and REPEAT statements. The body of code must be executed at least once for the REPEAT because the test comes at the end of the loop. It is possible for the body of the WHILE to be avoided completely because the test is at its head.

Now consider the previous example, performed this time with the use of the REPEAT statement.

```
PROGRAM EVALUATE_Y(OUTPUT);

    VAR  X,Y  :  REAL;

BEGIN
X:=1.0;

REPEAT

  Y:=1.0+X*(1.0+X*(1.0+X));
  X:=X+0.1;
  WRITELN (' Y= ',Y);
  UNTIL  X>10.0

END.
```

The final type of looping mechanism considered here is the FOR statement. Unlike the WHILE and REPEAT statements, FOR is used only when the number of times the loop is to be repeated is known in advance. One form of FOR statement is

FOR control variable:=initial expression TO final expression DO s

At the Summer School, the control variable is of integer type and the initial and final expressions must yield values of the same type. The control variable must not be altered in the subsequent block. Again s represents a single statement or sequence of statements (a block) that are executed repeatedly, with the control variable first taking the value of the initial expression and then being increased in steps of one until the block is repeated for the final value of the expression. In figure 6.6 the schematic form of the FOR statement is given.

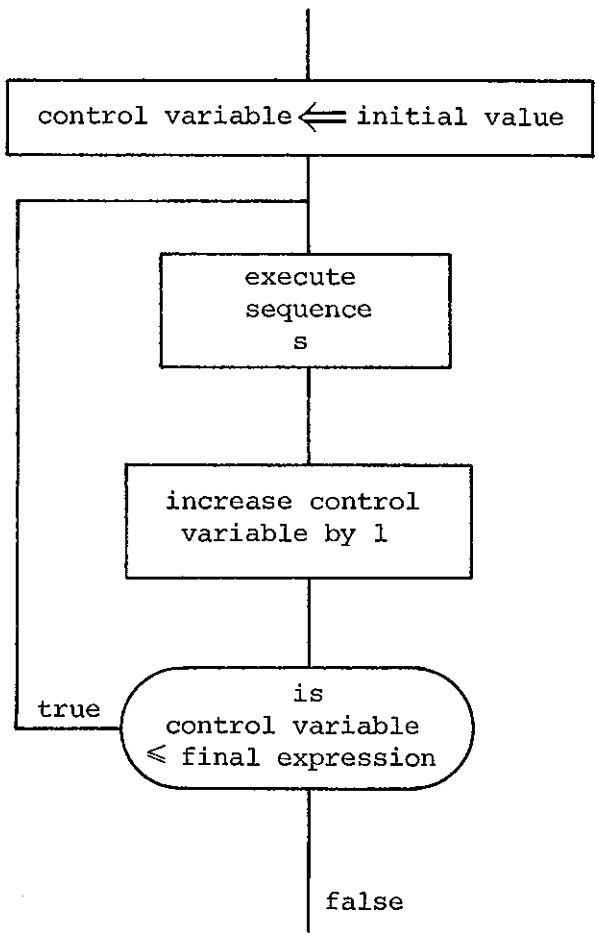


Figure 6.6 Schematic representation of FOR

At the completion of the DO block, the control variable is undefined - *i.e.* the programmer must re-initialise it before it can be used elsewhere in the program. Consider now an example showing the use of the FOR loop. Suppose our problem is to find the sum of the first 20 integers, *i.e.* 1+2+3+...+20 and to print out a running sum as we proceed. Ignoring any appeal to mathematical analysis, the following code would be sufficient:

```
PROGRAM SUM20(OUTPUT);

  VAR  SUM,I  :  INTEGER;

BEGIN
SUM:=0;

FOR I:=1 TO 20 DO
  BEGIN
  SUM:=SUM+I;
  WRITELN(' SUM OF ',I,' INTEGERS IS ',SUM)
  END

END.
```

Of course it is possible to have the above example coded in terms of the WHILE or REPEAT structures. The code for each would be

```
PROGRAM SUM20(OUTPUT);

  VAR  SUM,I  :  INTEGER;

BEGIN

SUM:=0;
I:=1;

WHILE  I<21  DO

  BEGIN
  SUM:=SUM+I;
  WRITELN(' SUM OF ',I,' INTEGERS IS ',SUM);
  I:=I+1
  END

END.
```

for the WHILE construct. In the case of the REPEAT construct the code is

```
PROGRAM SUM20(OUTPUT);

  VAR  SUM,I  :  INTEGER;

BEGIN

SUM:=0;
I:=1;

REPEAT
  SUM:=SUM+I;
  WRITELN (' SUM OF ',I,' INTEGERS IS ',SUM);
  I:=I+1
UNTIL  I>20

END.
```

It is obvious that the code for the WHILE and REPEAT constructs is a little longer than that of FOR. This is because the automatic steps of the FOR loops (*i.e.* the incrementing and testing of the control variable) must now be undertaken in more specific terms.

As a further example consider the problem of finding the mean of 10 numbers. Let us assume that the numbers are real and that they are punched 1 number per card. Because we know the number of times the loop must be repeated in advance, we shall use the FOR construct.

```
PROGRAM AVERAGE(INPUT,OUTPUT);

VAR MEAN,SUM,X :  REAL;
    I :           INTEGER;

BEGIN

SUM:=0.0;

FOR  I:=1 TO 10 DO

  BEGIN
  READLN(X);
  SUM:=SUM+X
  END;

MEAN:=SUM/10.0;
WRITELN(' AVERAGE = ',MEAN)

END.
```

Sometimes it is necessary to nest one loop inside another. Suppose we have 100 punched data cards, with one number punched per card, and our aim is to find the mean of each group of 10 and print out that value. The following listing is a complete program capable of doing this - FOR loops are used because the number of times repetition is required is known in advance.

```
(*
   PROGRAM BY SMITH TO FIND THE MEANS FOR 10 GROUPS
                            OF 10 NUMBERS.
*)

PROGRAM AVG(INPUT,OUTPUT);

  VAR  X,SUM,MEAN:  REAL;
       I,J       :  INTEGER;

BEGIN

FOR I:=1 TO 10 DO

  BEGIN
  SUM:=0.0;

    FOR J:=1 TO 10 DO

      BEGIN
      READLN(X);
      SUM:=SUM+X
      END;

  MEAN:=SUM/10.0;
  WRITELN(' AVERAGE OF GROUP ',I,' IS ',MEAN)
  END

END.
```

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
```

Let us consider further the logical expression e that is used with the WHILE and REPEAT statements. A logical expression returns the value of TRUE or FALSE. At this Summer School, we will restrict the use of the logical expression to testing the equality or inequality of arithmetic expressions. The simplest logical expression can be constructed through use of the following relational operators:

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<>	not equal to

The expressions

```
PRINCIPAL > 0.0
X <= Y
I <> J
X + SQRT (Z) <= SIN (Y)
```

are simple forms of logical expressions returning a TRUE or FALSE (Boolean) value.

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
OR	result is TRUE if either expression is TRUE.

Suppose we have a deck of cards of indeterminate number, and that three real numbers are punched on each card. Does each trio of numbers possibly constitute the sides of a triangle? The following code will read each data card in turn, test the possibility that each triplet forms a triangle and stop when a card is encountered whose punched numbers do not correspond to the sides of a triangle.

```
PROGRAM TRIANGLE(INPUT,OUTPUT);

  VAR  A,B,C  :  REAL;

BEGIN

READLN(A,B,C);

WHILE (A+B>C) AND (A+C>B) AND (B+C>A) DO

  BEGIN
  WRITELN(' TRIANGLE POSSIBLE FOR ',A,B,C);
  READLN(A,B,C)
  END;

WRITELN(' FINISHED TEST ')

END.
```

In this example, a FOR loop is not appropriate because the number of repetitions is unknown at the start. The repeat structure would have been inappropriate because it is necessary to execute the body of the loop before the logical test is encountered. Should the first data card not have been a triangle, this would have had an inappropriate result.

Should we have a complex logical expression involving both AND and OR operators, it is necessary to have some rules for precedence of evaluation. Like the rules for arithmetic operators, expressions enclosed in parentheses are evaluated first; AND has a higher precedence than OR, and evaluation proceeds from left to right when operators are of equal precedence.

6.11 CONDITIONAL STATEMENTS

It is frequently necessary to execute a single statement dependent upon some condition, or at some point to execute one of a number of possible statements dependent upon some condition. The IF statement of Pascal (which comes in two basic forms) may be used to achieve this end:

(a) IF e THEN s

(b) IF e THEN s ELSE t

where e represents a logical expression, and s and t represent single statements or blocks of code to be executed, depending on the result of the logical expression. The diagramatic representation of the simpler form (a) is shown in figure 6.7.

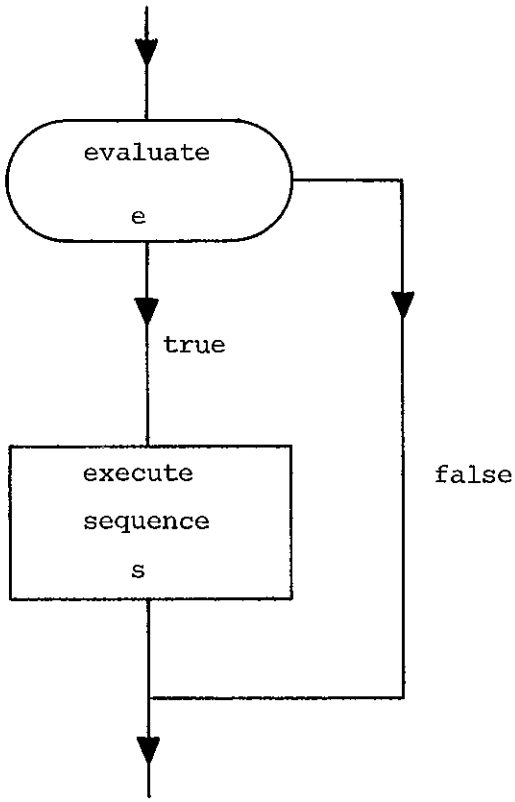


Figure 6.7 Diagrammatic representation of IF-THEN

Imagine for a moment that there were no absolute value function (ABS) and that we wished to implement our own code for it; the following lines would then suffice:

```
    :
READLN(X);
IF X<0.0  THEN  X:=-X;
WRITELN(X);
    :
```

When a value of X is read that is negative, the value of X is negated. When a positive value is read for X, the negation operation is avoided altogether and control passes onto the WRITELN statement.

The more involved form of the IF statement (b) is represented in schematic form in figure 6.8:

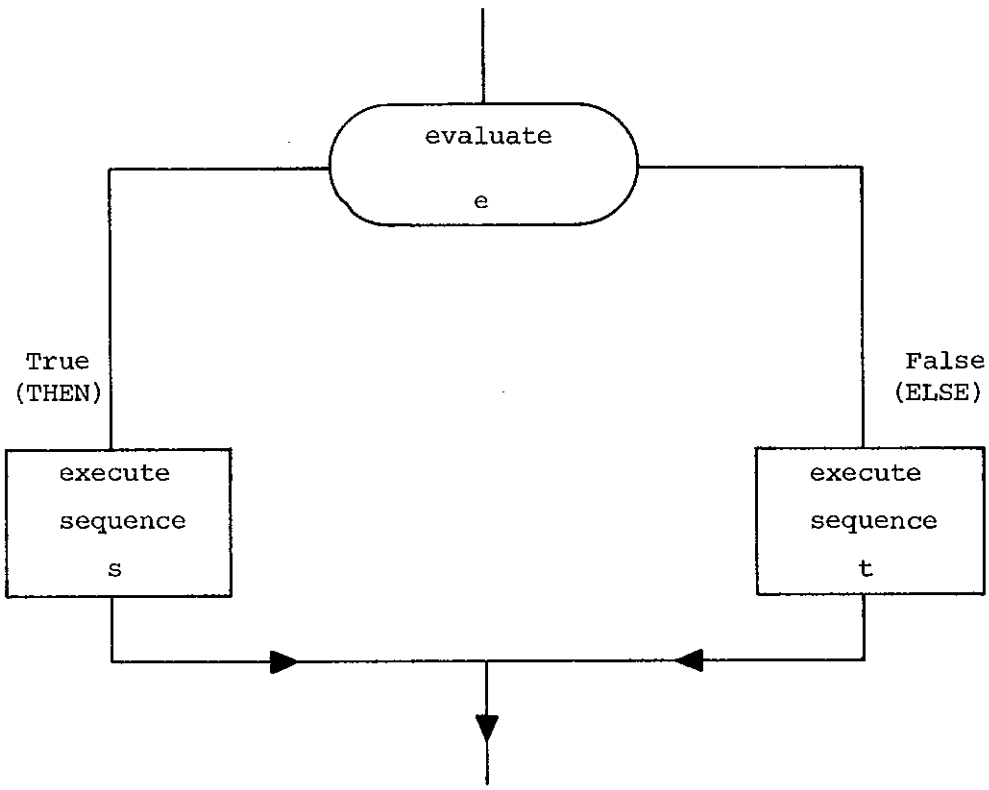


Figure 6.8 Diagrammatic representation of IF-THEN-ELSE

where a two-way selection is now possible, depending upon the value of the logical expression. Suppose we wish to read ten real numbers and find separately the sum of those that are negative and positive. The IF-THEN-ELSE construct enables us to do this very neatly.

```
PROGRAM POSNEG (INPUT,OUTPUT);

  VAR  X,SUM_POSITIVE, SUM_NEGATIVE : REAL;
       I                            : INTEGER;

BEGIN
SUM_POSITIVE:=0.0;
SUM_NEGATIVE:=0.0;

FOR  I:= 1 TO 10 DO

  BEGIN
  READLN(X);
  IF X>0.0 THEN SUM_POSITIVE:=SUM_POSITIVE+X
           ELSE SUM_NEGATIVE:=SUM_NEGATIVE+X
  END;

WRITELN(' POSITIVE ',SUM_POSITIVE,'NEGATIVE ',SUM_NEGATIVE)

END.
```

Note that the ELSE is not a separate statement, consequently there is no separator (;) between the THEN clause and the ELSE.

One may make the IF-THEN-ELSE construct do more as the following example indicates. Suppose we wish to test that the three numbers correspond to the lengths of possible sides of a triangle, and to print a message to this effect along with the perimeter of the triangle. The following code serves this purpose:

```
PROGRAM TRIANGLE(INPUT,OUTPUT);

  VAR  A,B,C,PERIMETER  :  REAL;

BEGIN
READLN(A,B,C);

  IF (A+B>C) AND (B+C>A) AND (A+C>B)

     THEN BEGIN
          WRITELN(' TRIANGLE POSSIBLE FOR ',A,B,C);
          PERIMETER:=A+B+C;
          WRITELN(' PERIMETER IS ',PERIMETER)
          END
     ELSE WRITELN(A,B,C,' DO NOT FORM TRIANGLE')

END.
```

This shows how a block of instructions can be executed conditionally as part of the THEN clause of an IF statement. (Blocks naturally may be used for the ELSE clause as well.)

Suppose we wish to decide whether A, B and C correspond to the sides of a triangle, then to determine if the triangle is equilateral, isosceles or scalene. To do this, the IF statements may be nested as indicated in the following code:

```
PROGRAM TRIANGLE(INPUT,OUTPUT);

  VAR  A,B,C  :  REAL;

BEGIN
READLN(A,B,C);

IF (A+B>C) AND (B+C>A) AND (A+C>B)
  THEN
       IF (A=B) AND (B=C)
         THEN WRITELN(' TRIANGLE EQUILATERAL ')
         ELSE IF(A=B) OR (B=C) OR (A=C)
                THEN WRITELN(' TRIANGLE ISOSCELES')
                ELSE WRITELN(' TRIANGLE SCALENE')
  ELSE WRITELN(' A,B,AND C DO NOT FORM A TRIANGLE')

END.
```

At first sight, the idea of a nested IF may appear to be ambiguous. For example

IF e_1 THEN IF e_2 THEN s_1 ELSE s_2

needs special rules. In Pascal, the above construct is interpreted as though it were written

```
IF e1 THEN
   BEGIN
      IF e2 THEN s1
            ELSE s2
   END
```

Consequently, the execution of s_2 is dependent upon e_2 (as well as e_1). If we wished to have s_2 independent of e_2 (*i.e.* for s_2 to depend directly on e_1) it would be necessary to code the expression as

```
IF e1 THEN
      BEGIN
      IF e2 THEN s1
      END
   ELSE s2
```

It is not necessary to lay out the code as shown above; it could all have been punched on one line, however, it is far easier to understand in the form shown.

6.12 SUBSCRIPTED VARIABLES

Many mathematical operations require the use of vectors and matrices. Pascal supplies a means of handling 1, 2 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 Pascal as V(.I.) on the card punch machine. (Should you have access to a fancy input device, such as a terminal with a more enhanced keyboard, V[I] is a preferable notation to use.) Elements of an array or vector can be used in Pascal in the same way as ordinary variables;

```
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*KP-7.)
```

At the Summer School, the subscripts used to refer to vector or array elements are restricted to INTEGER type. They may be constants, variables or expressions. The Pascal compiler reserves one location to store non-subscripted variables. 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 the non-executable VAR statement. For example, suppose we wished to associate 15 locations with the vector v (*i.e.* $v_1, v_2, \ldots, v_{15}$), then a declaration could be

```
V: ARRAY (.1..15.) OF REAL,
```

if v were to contain 15 real elements. The loop

```
FOR I=1 TO 8 DO V(.2*I-1.):=0.0
```

would set the odd components of v to zero. The subscripts in Pascal may be non-positive; for example the declaration

```
X: ARRAY (.0..16.) OF REAL;
```

would associate 17 elements with the vector X, the first being identified by X(.0.).

It is usually easiest for students of mathematics to associate subscripted variables in a computer language with the mathematical concept of an array or vector. This approach will be continued at the Summer School. In the commercial programming world, where mathematical applications take second place to business needs, it is worth noting that subscripted variables still play an essential role. They no longer have the obvious mathematical meanings, and frequently the programmer has little (if any) idea of what constitutes a vector or matrix. Pascal supports many other array concepts, however, at the Summer School these possibilities are ignored in favour of the ordinary mathematical outlook.

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.* one number per card):

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

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

```
(*
CALCULATES THE MEAN AND STANDARD DEVIATION OF 10 NUMBERS
*)
PROGRAM STATS(INPUT,OUTPUT);

  VAR X                          : ARRAY (. 1..10 .) OF REAL;
      SUM,AVG,SUMSQ,SDEV         : REAL;
      I                          : INTEGER;

BEGIN

FOR  I := 1 TO 10 DO READLN (X(.I.));

SUM:=0.0;
FOR  I:=1 TO 10 DO SUM:=SUM+X(.I.);

AVG:=SUM/10.0;
SUMSQ:=0.0;

FOR  I:=1 TO 10 DO SUMSQ:=SUMSQ+SQR(X(.I.)-AVG);

SDEV:=SQRT(SUMSQ/9.0);
WRITELN(' MEAN AND STANDARD DEVIATION ARE ',AVG,SDEV)

END.
```

Here we use the vector X to store ten numbers before finding the mean and standard deviation. Before employing vectors in a program, *make sure they are really necessary*. In a previous example (section 6.10) the mean of a set of numbers was required. There was no need in that case to retain the ten numbers because the sum accumulated when each number was read from a punched card. When the standard deviation is calculated by the above formula, however, it is necessary to retain all the numbers in memory, so a vector is required.

The next example demonstrates a program that computes the vector sum $\underset{\sim}{s}$ of two vectors $\underset{\sim}{u}$ and $\underset{\sim}{v}$:

$$\underset{\sim}{s} = \underset{\sim}{u} + \underset{\sim}{v} \ .$$

For $\underset{\sim}{u} = (3,5,2)$

and $\underset{\sim}{v} = (4,2,7)$

then $\underset{\sim}{s} = (3+4,\ 5+2,\ 2+7)$

$= (7,7,9)$

Mathematically we say that the i^{th} component of $\underset{\sim}{s}$ is formed as

$$s_i = u_i + v_i \qquad 1 \leqslant i \leqslant 3$$

The program will read the three pairs of data from separate punched cards as shown

u	v
3.0	4.0
5.0	2.0
2.0	7.0

into two vector arrays (U and V), compute the vector sum in S and print out each component of S on a separate line.

```
PROGRAM VECTOR(INPUT,OUTPUT);

  VAR U,V,S : ARRAY (. 1..3 .) OF REAL;
      I     : INTEGER

BEGIN

(* FIRST READ IN THE DATA *)

FOR I:=1 TO 3 DO READLN( U(.I.),V(.I.) );

(* FORM VECTOR SUM *)

FOR  I:=1 TO 3 DO S(.I.):=U(.I.)+V(.I.);

(* WRITE OUT RESULTS *)

WRITELN(' VECTOR S ');

FOR I:=1 TO 3 DO WRITELN(S(.I.))

END.
```

(In this example, it would have been possible to perform the vector addition operation without the use of subscripted variables - how? See appendix 6A for a solution. Such an operation, however, is frequently a small part of a much larger program where it is necessary to store the data in subscripted variables.)

When arrays of higher order than the one-dimensional vector treated are needed, the ARRAY declaration informs the compiler of the number of dimensions (*i.e.* the number of subscripts and the total storage required for the array).

```
VAR     A : ARRAY(. 1..5,1..5 .) OF REAL;
```

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

```
VAR A,B,C : ARRAY (. 1..5,1..5 .) OF REAL
      :
FOR I = 1 TO 5 DO
FOR J = 1 TO 5 DO C(.I,J.):=A(.I,J.)+B(.I,J.);
      :
```

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

6.13 PROCEDURES AND FUNCTIONS

We have met (section 6.9) the special mathematical functions supplied through the Pascal compiler. The user is able to supply two types of routines of his own when necessary:

- Function
- Procedure

Need for special routines arises:

(i) when 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; and

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

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

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

which, for speed of computation, is best written as

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

When we have eventually constructed the code for this function, it may be invoked through Pascal statements such as

```
Y:=F(X)
```

```
Y:=F(X)+6.0
Y:=F(X-3.0)+7.0
```

The following example demonstrates the use of a FUNCTION and the main program code necessary to invoke it so that the above function f(x) is evaluated for x=0.,1.,...,10.

```
PROGRAM EVALUATE_FUNCTION (OUTPUT);

  VAR   X,Y  : REAL;
        I    : INTEGER;

FUNCTION F(Z : REAL) : REAL;

BEGIN
F:=1.0+Z*(1.5+Z*(3.2+6.0*Z))
END;

(*  MAIN  PROGRAM  *)

BEGIN
X:=0.0;

WHILE  X<=10.0 DO
  BEGIN
  Y:=F(X);
  WRITELN(Y);
  X:=X+1.0
  END

END.
```

FUNCTION CODE

MAIN PROGRAM CODE

We notice, in the simple example, that the code for the FUNCTION F has been separated from and must precede the main body of code.

Because the variables X,Y and I were declared at the head of the program, they are available if required in all portions of the program. These are known as global variables.

The variable Z used to define the FUNCTION F is known as a local variable and may be invoked only through the scope of the FUNCTION itself. Any reference to it outside the FUNCTION would not be permitted by the Pascal compiler. The scope of a FUNCTION is identified easily through the appropriate BEGIN...END delimiters within the FUNCTION.

The FUNCTION must be declared to be of REAL type to avoid truncation of the fractional portion of the result. The FUNCTION parameter Z is a dummy but it must correspond in type to the actual parameter in the calling program (X). When the FUNCTION is invoked through

```
Y:=F(X)
```

the value of X is transferred effectively to Z for evaluation. The function operates on a copy of X in Z and must transfer the calculated result back by assigning it to F, the name of the function. In this case, the function makes no attempt to alter the value of Z. Should you desire the function to alter the transferred value for some reason, the dummy parameter must be declared to be changeable. This is done by the alternative function declaration

```
FUNCTION F(VAR Z:REAL):REAL;
```

Consider now a second example involving a FUNCTION. Ignoring an appeal to mathematical analysis, suppose we create a new mathematical function

$$g(x) = 1 + \frac{1}{x} + \frac{1}{x^2} + \frac{1}{x^3} + \ldots$$

and that a sufficient number of terms are included so that $|\frac{1}{x^n}| < 0.00001$. The following program defines the function g(x) and evaluates it for x = 2.0,2.1, ... , 3.0.

```
PROGRAM EVALUATE_G(OUTPUT);

  VAR X,Y :  REAL;

FUNCTION G(Z : REAL ) : REAL;

VAR SUM,TERM : REAL;

BEGIN
SUM:=0.0;
TERM:=1.0;

REPEAT SUM:=SUM+TERM;
       TERM:=TERM/Z;
UNTIL  ABS(TERM)<0.00001;

G:=SUM
END;

(* MAIN PROGRAM *)

BEGIN
X:=2.0;

WHILE X<3.0 DO

  BEGIN
  Y:=G(X);
  WRITELN(X,Y);
  X:=X+0.1
  END

END.
```

In this example, additional variables SUM and TERM are introduced in the formulation of FUNCTION G(Z). These variables are defined within the scope of G, hence are local to it and unavailable for use outside of it.

At this Summer School, we will use a special FUNCTION PRICE to give us the price of oil for different energy scenarios. This function will be supplied by the School, so you need not worry about the code within it. You will, however, need to know how to tell your program that this is a special function not supplied by the Pascal compiler (DECLARE, section 6.13) and know how to invoke it (appendix 6B).

The PROCEDURE is the more powerful version of a sub-program and usually performs more involved operations than those for which the FUNCTION is designed. Typical tasks for which procedures are used would include finding the roots of equations, multiplications or other operations on matrices, and solving sets of linear equations. Unlike the FUNCTION the PROCEDURE is not restricted to returning one result as part of an arithmetic expression.

The following code shows the use of PROCEDURE 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 parameters for the PROCEDURE, whereas the PROCEDURE is responsible for returning the two roots and an indication as to whether real roots were possible. It is assumed the coefficient (a,b,c) are real numbers supplied on data cards (three numbers per card). The number of data cards is indeterminate, however, all coefficients are known to be less than 10^6 in absolute value. Consequently the end of the data is signalled by a card with coefficients outside this range.

```
PROGRAM SOLVER (INPUT,OUTPUT);

  VAR C1,C2,C3,R1,R2 :  REAL;
                IER :  INTEGER;

PROCEDURE  QUAD(A,B,C: REAL; VAR X1,X2 : REAL; VAR K : INTEGER);
  VAR DISCRIMINANT : REAL;

BEGIN
DISCRIMINANT:= B*B-4.0*A*C;
IF  DISCRIMINANT>0.0 THEN

                          BEGIN
                          DISCRIMINANT:=SQRT(DISCRIMINANT);
                          R1:=(-B+DISCRIMINANT)/(2.0*A);
                          R2:=(-B-DISCRIMINANT)/(2.0*A);
                          K:=0
                          END

                      ELSE
                          K:=1

END;

(* MAIN PROGRAM *)

BEGIN
READLN (C1,C2,C3);

WHILE ABS(C1)<1.0E6 DO
  BEGIN

  QUAD(C1,C2,C3,R1,R2,IER);
  IF  IER=0  THEN  WRITELN(' REAL ROOTS ARE ',R1,R2)
             ELSE  WRITELN(' NO REAL ROOTS');
  READLN(C1,C2,C3)
  END

END.
```

Like the previous FUNCTIONS the PROCEDURE QUAD is positioned after the initial declarations but before the body of the main program. Unlike the FUNCTION which is used to return a single value as part of an arithmetic assignment, the PROCEDURE QUAD is invoked by a single statement.

```
QUAD(C1,C2,C3,R1,R2,IER);
```

The main program passes three coefficients of the quadratic, while the PROCEDURE will return the roots in X1 and X2 and an indication (K=0 or 1) as to the sign of the discriminant. The three values are returned in R1,R2 and IER. Because these variables are assigned values by the

PROCEDURE, it is necessary to indicate that the appropriate dummy parameters may be assigned values by use of the VAR declaration.

During solution of the Summer School problem, you will find it advantageous to display the output in graphical form. Appendix 6C demonstrates how this may be done using the supplied PROCEDURES PLOT and PLOTPR.

Throughout the Summer School, special supplied FUNCTIONS and PROCEDURES are necessary. The details of the parameter lists are given in the appropriate appendices. Unlike the Pascal function such as SQRT, these are not part of the standard repertoire of routines. It is necessary for you to make sure the Pascal compiler knows there are special routines of which it has no intrinsic knowledge. This may be done by the special Summer School command (not normally part of the Pascal language):.

DECLARE $name_1$, $name_2$, ... ;

This is placed immediately after the VAR declarations and before the first executable statement. For example

```
DECLARE  RND  ;
```

would initialise the random number generator RND.

6.14 CONSTANTS

Frequently, programmers wish to associate particular values with certain names and use these identifiers throughout the program. These values are not to be altered within the program. A common requirement is to associate an approximation for π with the identifier PI. This is done through a CONST statement which precedes the VAR declaration. In the following example, the areas of circles of radii 1.0,1.1,1.2,...,2.0 cm are calculated.

```
PROGRAM AREA (OUTPUT);

CONST  PI=3.1415926;

VAR  R,AREA  :  REAL;

BEGIN

R:=1.0;

WHILE R <=2.0 DO

  BEGIN
  AREA:=PI*R**2;
  WRITELN (' AREA OF CIRCLE WITH RADIUS ',R,' CMS IS ',AREA);
  R:=R+0.1
  END

END.
```

The constant PI is not mentioned in the VAR declaration because it is not a variable; however, it has a type associated with it from its definition. Here it is REAL. The declaration

```
CONST  T=3;
```

causes T to be of the INTEGER type. Use of the CONST declaration provides some protection to the program which might accidentally alter the value of a fundamental constant.

6.15 A COMPLETE PROGRAMMING EXAMPLE

The following example demonstrates the stages of development involved in determining an approximation to π using a computer to simulate a dart board. If we had a circular dart board mounted on a square background, as shown in figure 6.9, then it would be possible to determine experimentally an approximation for π. When a dart is thrown

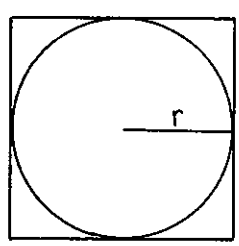


Figure 6.9 Dart board

randomly to land in the square, it may land within the circle or outside it. The probability of it landing within a certain section is proportional to the area of that section:

Relative proportion of darts landing in the circle

$$= \frac{\text{area of circle}}{\text{area of square}}$$

$$= \frac{\pi r^2}{(2r)^2}$$

$$= \frac{\pi}{4}$$

$$\therefore \pi = 4 \times \text{relative proportion of darts landing in the circle.}$$

Consequently, by randomly throwing darts and measuring the relative frequency of those falling within the circle, π can be determined approximately. Instead of throwing darts, a computer can be employed to do

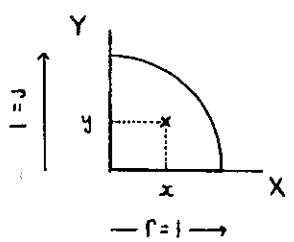


Figure 6.10 Quadrant simplification of dart board

this by direct simulation. The process can be simplified, as shown in figure 6.10, by taking a quadrant of a circle of unit radius. By selecting two random numbers, using our random number generator, we may let these two numbers (say x and y) represent the coordinates of the point where the dart lands. This point may be within the circle or outside it. If we measure the distance d of the point from the origin

$$d = \sqrt{x^2+y^2}$$

we can determine if it lies in the circle or not depending upon whether $d \leqslant 1$ or $d > 1$ (or, equivalently, whether $d^2 \leqslant 1$ or $d^2 > 1$).

A flow chart to describe the steps involved is shown in figure 6.11 for a sample of 1000 darts.

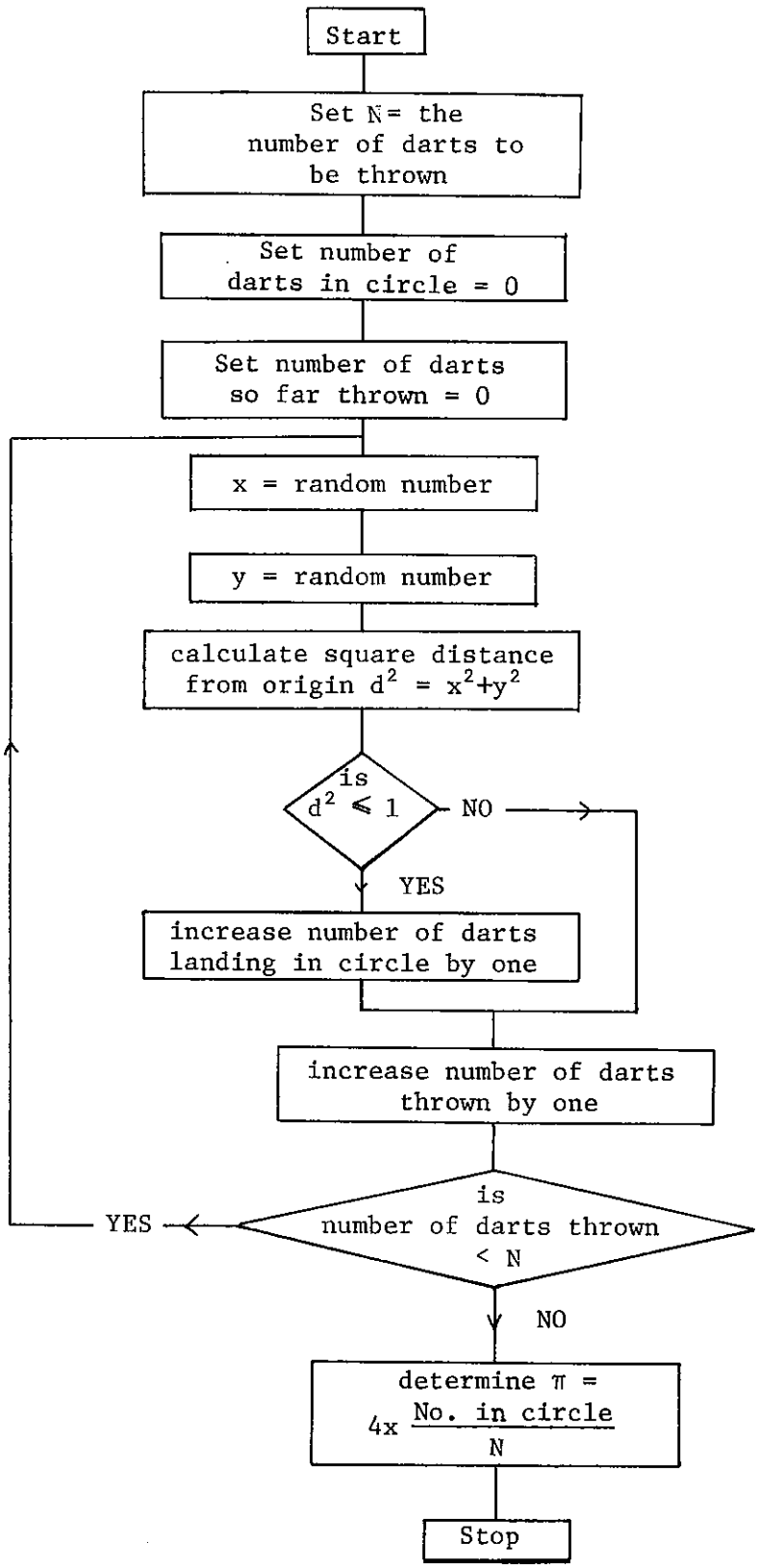


Figure 6.11 Flow chart for π problem

A Pascal program sufficient to produce an approximation for π is

```
PROGRAM PI(OUTPUT);

  CONST  N=2000;

  VAR  NO_THROWN,NO_IN_CIRCLE : INTEGER;
       X,Y,DSQ,PIE            : REAL;

DECLARE RND

BEGIN

NO_IN_CIRCLE:=0;

FOR  NO_THROWN:=1 TO N DO

     BEGIN
     X:=RND;
     Y:=RND;
     DSQ:=X*X+Y*Y;
     IF DSQ<1.0 THEN NO_IN_CIRCLE:=NO_IN_CIRCLE+1
     END;

PIE:=4.0*NO_IN_CIRCLE/N;
WRITELN(' PI = ',PIE)

END.
```

6.16 ERRORS IN PROGRAMMING

The Pascal compiler will inform us in no uncertain terms of any syntactical errors we make in coding a program. Such errors are easy to detect and correct. The computer is a totally obedient servant; provided that 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. In reverse to British justice, all programs should be considered guilty (of containing bugs) until proved 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 debugging the 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 in 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 in which we are interested.

There are three ways in which the problems of the scientific programmer are different to those of the more commercially oriented programmer. 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?

6.17 PRACTICE EXAMPLES

Before you attempt to code the Summer School problem described in chapter 3, try these practice examples. The answers to the questions are given in appendix 6G, but don't be too hasty to seek these out until you have had a go yourself.

Q1. (a) In the list below, which items are variable names or numbers?
(b) What is the mode (integer or real) of each number in the list?
(c) Are any invalid? (Why?)

List (1) 1. (2) ABC (3) SOUTH_AMERICA (4) 14 (5) .6 (6) FIVE
(7) 6IX (8) 0 (9) BOS (10) A*B (11) 5,132.6 (12)IRE-LAND
(13) WATER_2_DRINK (14) -0.001E-10

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

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

(2) $y = \frac{\sqrt{b^2-4ac}}{2a}$

(3) $y-x = a-\pi y$ (π=3.1415926)

What values would be stored in the variable on the left of the following arithmetic statements? Are any of the statements invalid?

```
CONST   A=3.8;
VAR   U:REAL;
    K,I:INTEGER;
```

(4) `I:=A;`

(5) `I:=ROUND(A);`

(6) `I:=TRUNC(A);`

(7) `U:=A/2.;`

(8) `U:=A/2.0;`

(9) `K:=TRUNC(A+2.9);`

(10) `I:=(K-1)DIV 2;`

(11) `A:=A/2.0;`

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:

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

(2) $y = e^x$

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

(4) $v = \tan x$

(5) $c = \ell n \left|\dfrac{1}{1+a^3}\right|$

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

Q4. What, if anything, is incorrect in the following section of Pascal code? Assume that the declarations and other statements that would normally precede each is appropriate.

(1) `WRITELN(' RESULT IS ',A);`

(2) `X:=A+B*SQRT(D);`

(3)
```
FOR I:=1 TO N DO X:=X+1.;
```

(4)
```
FOR I:=1.0 TO 10.6 DO X:=X+1.0;
```

(5)
```
FOR I=1 TO 5 DO ANYTHING;
```

(6)
```
X:=0.0;
I:=0;
WHILE I<100 DO BEGIN
                X:=X+1.0;
                I:=I+1
                END;
```

(7)
```
X:=0.0;
I:=0;
WHILE I<100 DO BEGIN
                X:=X+1.0;
                WRITELN(X)
                END;
```

(8)
```
IF X>=Y THEN WRITELN(X);
        ELSE BEGIN X:=X+1.0;
                   WRITELN(X)
             END;
```

(9)
```
X:=-X IF A<B
```

(10)
```
VAR X:REAL;
    I:INTEGER;
X:=I;
```

(11)
```
VAR X:REAL;
    I:INTEGER;
I:=X;
```

(12)
```
VAR X:REAL;
    I:INTEGER;
I:=TRUNC(X);
```

(13)
```
IF A=B=C THEN X:=A+B+C;
```

(14)
```
VAR T,X : ARRAY (. 0..100 .) OF REAL;
```

(15)
```
VAR TT,XX : ARRAY (. 0..100,0...100 .) OF INTEGER
```

(16)
```
PROGRAM CALCULATE (OUTPUT);
VAR X,I:REAL;

FUNCTION COST (Z:REAL):REAL;
BEGIN
COST:=2.0*Z+6.0
END;
(* MAIN PROGRAM  *)
BEGIN
I:=1.0;
WHILE I<10.0 DO
                 BEGIN
                 X:=COST(I);
                 WRITELN(I,X);
                 I:=I+1.0
                 END
END.
```

(17)
```
PROGRAM CALCULATE (OUTPUT);
VAR X:REAL;
    I:INTEGER;

FUNCTION COST (Z:REAL):REAL;
BEGIN
RESULT:=2.0*Z+6.0
END;
(* MAIN PROGRAM  *)
BEGIN
I:=1
WHILE I<10 DO
               BEGIN
               X:=COST(I):
               WRITELN(I,X);
               I:I+1
               END

END.
```

Q5. Write a Pascal program to calculate and print the perimeters of a set of circles whose radii are 1., 1.5, 2., 2.5, ... , 10. cm respectively ($\pi \approx 3.1415926$).

Q6. Write a Pascal program to sum to first 20 terms of the series

$$1 + 2 + 4 + 7 + 11 + 16 + \ldots$$

Q7.

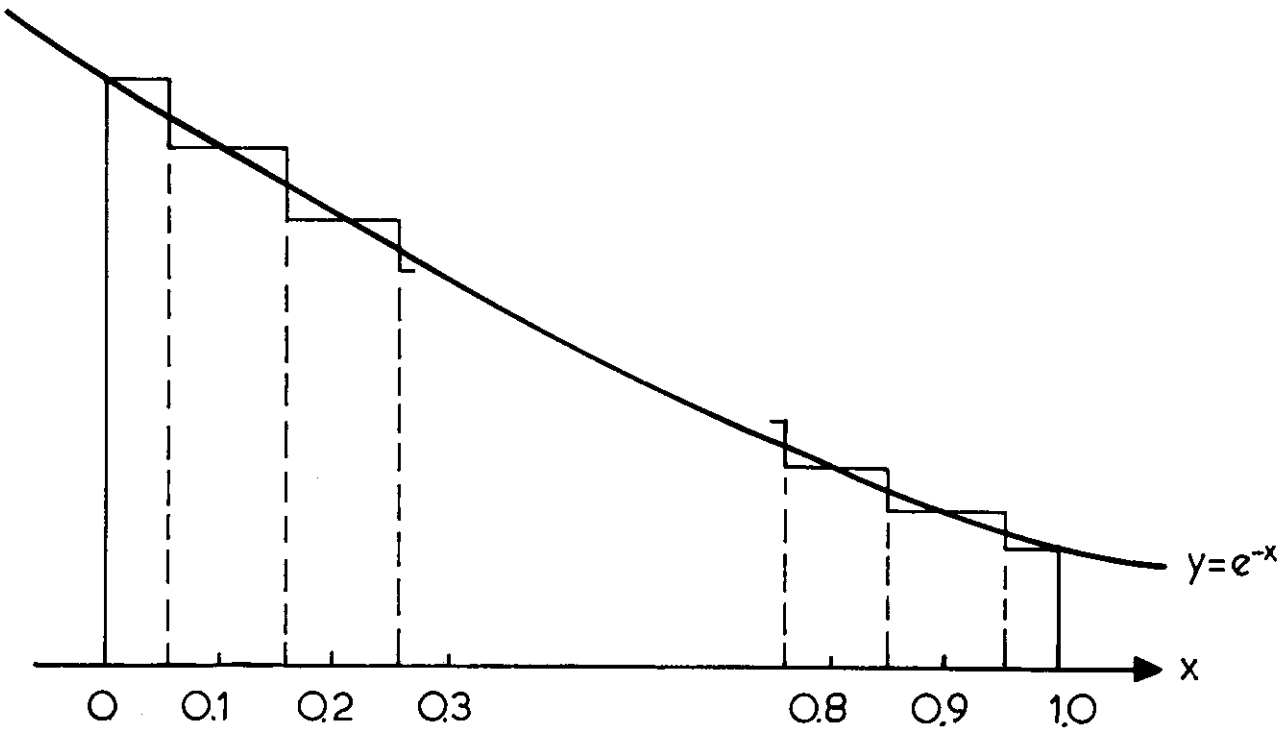


Figure 6.12 A numerical approximation to $\int_0^1 e^{-x}dx$

If you are asked to evaluate $\int_0^1 e^{-x}dx$ (*i.e.* find the area under the curve) then you could, with a knowledge of high school mathematics, quickly give the correct answer (I hope). At times we can be given very difficult functions to integrate and the only recourse is then to a numerical approximation. One way of approximating $\int_0^1 e^{-x}dx$ numerically is to sum the area of the histogram sections approximating e^{-x} in figure 6.12. This is done easily, but we must take particular care when treating the segments at x=0 and x=1. With the approximation shown above, we write

$$\int_0^1 e^{-x}dx \approx .1\left(\frac{e^0}{2} + \sum_{i=1}^{9} e^{-i*.1} + \frac{e^{-1}}{2}\right) .$$

(i) Write a Pascal program to evaluate $\int_0^1 e^{-x}dx$ with the above discrete approximation.

(ii) How good is your answer? Do you notice any improvement if you run your program a second time with double the number of subdivisions?

Mathematicians would write the above technique of integration (known as the histogram method) for a general function f(x) as

$$\int_a^b f(x)dx = h\left[\frac{f(a)}{2} + \sum_{i=1}^{n-1} f(x_i) + \frac{f(b)}{2}\right] \quad ,$$

where $h = \frac{b-a}{n}$, n+1 is the number of points x_i at which the function $f(x_i)$ is evaluated.

Q8. When you have finished the Pascal program required in the previous question, consider the following very carefully, because it has a direct bearing on the way in which you will perform the integration required in 'the Summer School problem'.
Integration of certain 'economic' information cannot be achieved by the direct application of the above approximation technique.
Suppose we are given the following information (displayed in graphical form in figure 6.13) concerning the number of kilolitres of petrol sold by a local service station over a reasonably short time.

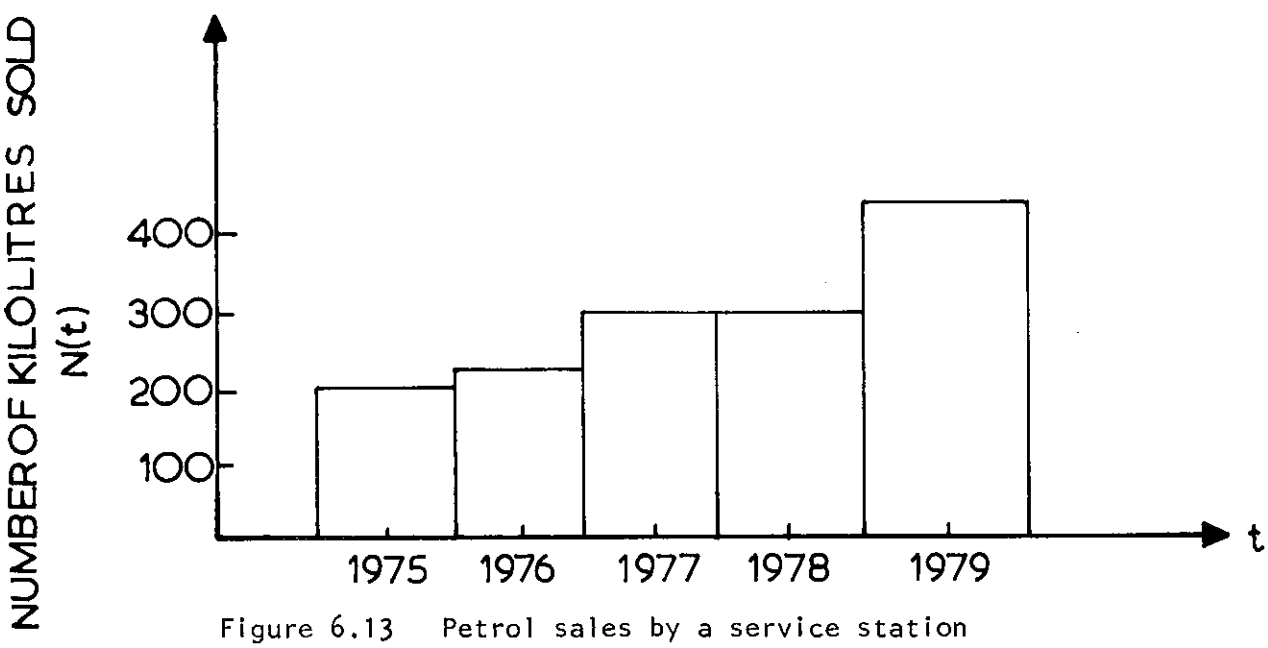


Figure 6.13 Petrol sales by a service station

If N(t) represents the number of kilolitres of petrol sold in year t, then we may say that the number of kilolitres sold during the period 1975 through 1979 is given by

$$\int_{1975}^{1979} N(t)dt \quad .$$

If we were to apply the above numerical integration technique directly to this problem, we would not accumulate half the sales in the years 1975 and 1979. The reason for this is that in producing

the graphical display, the economist no longer regards the time variable as a continuous one since he is really using yearly time averages. He has suitably 'discretised' time for his own purposes; the mathematician should be aware of this when attempting the integration.

The integral is simply evaluated as

$$\int_{1975}^{1979} N(t)dt = N(t=1975) + N(t=1976) + N(t=1977) + N(T=1978) + N(t=1979) \quad ,$$

which follows the style we require for the Summer School problem (chapter 3).

Q9. Write a Pascal statement that will generate a random real number in the range $0<y<10$.

Q10. Write a Pascal statement that will generate a random real number in the range $0.1<y<0.6$.

Q11 Write a Pascal statement that will generate a random integer from the set 1, 2, 3, ..., 15

Q12. Bill Smith travels to work 5 days each week by bus. Bill is an extremely methodical person who arrives at the bus stop at precisely 8.00 a.m. The government bus service is also extremely punctual and its vehicles call at Bill's stop at 8.01, 8.06 and 8.11 a.m. Each of these is capable of getting Bill to work on time. His employer, although somewhat flexible and tolerant, will dismiss him should he arrive at work late more than once a month (one month = four working weeks) averaged over the period of employment. Can Bill reasonably expect to retain his job in the long term if the number of passengers each bus can pick up at his stop varies randomly between 0 and 15 inclusive, and if he could find any random number of people up to 10 in front of him in the queue? Ignore any appeal to probability theory and write a Pascal program using the random number generator RND to simulate the bus stop situation and help estimate Bill's employment security with his present firm. Run the daily simulation for 1000 such mornings and print out the number of days per month Bill is late. Before you write any Pascal code, you might like to draw a flow chart to make sure you clearly understand the steps involved.

Q13. (For advanced students only.)

The techniques of numerical integration outlined in question 6 are fairly basic and obvious. What is perhaps not so obvious to high school students studying calculus for the first time is that such 'crude' techniques at times can be highly effective. The second technique is particularly good for treating economic data. Economists frequently report (as is done here) the price level for each year as a price averaged over the whole year. When this is done, the use of more sophisticated and involved techniques would be meaningless anyhow!

An integration scheme for continuous functions that is more sophisticated than the histogram method of question 5 is Simpson's rule. You may have encountered this in your high school mathematics course. Simpson's rule approximates the integral of a function f(x) by

$$\int_a^b f(x)dx = \frac{h}{3}\left[f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \ldots + 4f_{n-1} + f_n \right]$$

where $h = \frac{b-a}{n}$ and $f_i = f(a + ih)$ for $i = 0,1,\ldots,n$.

Write a Pascal program using Simpson's rule to approximate

(i) $\int_1^2 (5x^4 + 4x^3 + 6x^2 + 4x + 1)dx$ with 9 points, *i.e.* 8 equal mesh intervals.

(ii) $\int_0^1 (4x^3 + 3x^2 + 2x + 4)dx$ with 5 points, *i.e.* 4 equal mesh intervals.

From your numerical results, what comment could you make concerning the accuracy of this method of integration?

Q14. (For advanced students only.)

Read in a set of ten numbers punched one per card. Write a code that will sort these numbers in descending order.

Q15. (For advanced students only.)

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.

(2) Read the estimate x_0 of a root of the equation $f(x) = 0$ from a second card.

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

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

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

$$\frac{|x_{n+1} - x_n|}{|x_n|} < 0.001 \quad .$$

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

(6) Allow only nine iterations. If convergence has not been achieved, print a message warning of this.

(7) Repeat from (1).

Q16. (For advanced students only.)

A mathematical quantity S defined by the infinite series

$$S = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} \ldots$$

needs to be computed and tabulated for x = 0, .1, .2, .3, ..., 1. Write a Pascal program to do this using sufficient terms so that $|S_{n+1} - S_n| < 0.00001$. (Remember n! = 1 × 2 × 3 × ... × n.)

Q17. (For very advanced students only.)

Assume that the Harbour Bridge authorities wish to reduce the average time a motorist spends waiting to get through the toll gate section of the bridge during peak hours. At present, there are 11 toll collecting stations - 7 automatic and 4 manual. Can the traffic situation be improved by exchanging an automatic station for a manual station or vice-versa? The authorities are unable to alter the number of stations by more than one either way for various reasons that do not concern you here. Your task is to help the authorities make the correct decision. They have studied traffic flow across the bridge for some time and have come up with the statistics given below. You have been engaged as a programmer to enable the authorities to try out the various configurations and to estimate the average time each car spends waiting to get through the toll area.

(1) The correct toll for all vehicles is 20¢ per vehicle.

(2) There are 7 automatic and 4 manual stations.

(3) Every 5 seconds either 7, 8 or 9 cars enter the toll area.

(4) On the average, 6.4 out of every 10 cars will select the automatic station.

(5) Of the drivers selecting an automatic station, 8 out of 10 will choose the station with the shortest queue. The other drivers will select a queue at random.

(6) Of the drivers selecting the manual station, 7 out of 10 will choose the station with the shortest queue. The other drivers select a queue at random.

(7) It takes 5 seconds to process the correct money at an automatic station, provided the coins are not dropped.

(8) If the coins are dropped at an automatic station, it takes 20 seconds to retrieve them.

(9) At a manual station the following times are involved, provided the toll money is not dropped:

20¢	5 seconds
$1	10 seconds
$2	15 seconds
⩾ $5	20 seconds

(10) If the money is dropped at a manual station, it takes 5 seconds to retrieve.

(11) One out of every hundred motorists will drop the money.

(12) One out of every hundred motorists will stall the vehicle at an automatic station and take 10 seconds to re-start.

(13) Two out of every hundred motorists will stall at a manual station and take 10 seconds to re-start.

Write a Pascal program to simulate the above bridge situation for a one-hour time span. Determine the average time a motorist spends waiting, and the average length of the queue. Also determine the length of the longest queue that forms. Given the constraints mentioned previously what is the best action open to the authorities?

NOTES

APPENDIX 6A

ALTERNATIVE SOLUTION TO THE SUMMATION PROBLEM OF SECTION 6.12

```
PROGRAM VECSUM(INPUT,OUTPUT);

VAR  S,U,V : REAL;
     I     : INTEGER;

BEGIN

WRITELN(' VECTOR S ');

FOR I:= 1 TO 3 DO

                BEGIN
                READLN(U,V);
                S:=U+V;
                WRITELN(S)
                END

END.
```

APPENDIX 6B

A MEANS OF OBTAINING THE PRICE OF OIL

When writing the code to handle the oil depletion model for this year's Summer School, a special Pascal function is required to return the price of oil. The function is supplied by us and designed to allow you to test your depletion model on a number of different scenarios. The justifications for using an exogenous price pattern for the model will be discussed by other lecturers. Here we are interested only in the way the function PRICE is to be invoked. Through the assignment

```
P:= PRICE(M,T,DXDT,X,XINF)
```

the price of oil for a particular year T is returned. The full description of the parameters is given here; however, you will have to refer to other sections of the notes to see the full significance of some of them.

The various entities used here have the following meaning:

M is an integer used to express which price model we wish to use in our analysis,
- 0 a constant price model, $p(t)=p_0$;
- 1 an analytically given price model (see appendix 1A);
- 2 the *most realistic model* based on published world data up to the present time with a 'guessed' price extrapolation based on resource depletion;
- 3 a pessimistic speculative model based on the data of model 2 but including a superimposed price hike in 1990 that is similar to that of 1974 (about a factor of 3);

T is the year (remember to make it REAL);

DXDT is a real quantity and represents dx/dt - this is time dependent although, for the purpose of our approximation, we can use the average value of oil consumed in the previous year, DX;

X is a real quantity and represents the total quantity x(t) of oil consumed in the world in Gbbls (gigabarrels) to time t - once again use the quantity calculated at the previous time step;

XINF is a real quantity and represents the total amount of world oil available for consumption x_∞=2000 Gbbls; and

PRICE returns a real value for the 1982 constant US dollar price of a barrel of oil at time t.

APPENDIX 6C

GRAPHICAL OUTPUT

'A picture is worth a thousand words', so the Kodak advertisement runs. A nice graph to present scientific data is worth a thousand lines of cluttered printout. To help you obtain a good idea of how your economic model behaves, a special plotting program is supplied by us to display the results in the form of a graph with labelled axes. To produce the picture quickly, we will compose the lines of the graph with * symbols on the printed output. Of course, very accurate plotters are available with resolutions up to a thousandth of an inch, but use of such devices involves a time delay before you receive the finished product from the computing facility. Consequently, for speed we will restrict the Summer School graphical display to the simpler form of output.

To use the Summer School plot packages, two concepts must be understood:

(i) For each new point to be displayed on the graph, x and y coordinates must be passed to a special procedure that will *later* do the plotting. The coordinates of each point are accumulated by the procedure until all the points to be displayed have been transmitted by you. The coordinates are transmitted by invoking the procedure PLOT in the following way

```
PLOT (T,DX,X,PC)
```

where T, DX, X and PC (oil price) are the real variables of our Summer School problem and the order, first T, then DX, *etc.* is important.

(ii) When all the points have been accumulated and you wish to print the final product, invoke another procedure, this time PLOTPR. This is done by the statement

```
PLOTPR
```

without any arguments. This causes all the data previously transmitted to be displayed with axes labelled appropriately.

For example we might have

$$x(t)=x_0 e^{\lambda(t-t_0)}$$

- the unlimited, fixed price model - say with $t_0 = 1909$, $x_0 = 4.29$ and $\lambda = 0.07$. Then we want a plot to cover our time span of interest. We could code this as

```
PROGRAM PLOT(OUTPUT);
CONST
TO=1909.0;
LAMBDA=0.07;
PC=5.26;
XO=4.29;

VAR   T,X,DX   : REAL;

DECLARE PLOT,PLOTPR

BEGIN
(* ACCUMULATE DATA FOR PLOTTING *)

T:=TO;

WHILE T<= 2050.0 DO

                  BEGIN
                  X:=XO*EXP(LAMBDA*(T-TO));
                  DX:=LAMBDA*X;
                  PLOT(T,DX,X,PC);
                  T:=T+1.0
                  END;

(* NOW DRAW GRAPH *)

PLOTPR
END.
```

APPENDIX 6D

IMPROVING CONTROL OVER OUTPUT

All the simple output statements treated so far are sufficient to enable the Summer School problem to be solved. For those who wish to know more about control over output this section is included. Rather than state formal rules, the explanation is performed through examples. For

```
X:=-1.36427815E+02
```

the statement

```
WRITELN('␣X=␣',X)
```

causes

```
X= -1.3642781499999998E+02
```

to be printed on the line printer. It will be noticed that the '␣X␣=␣' starts from column 1. The leading blank in '␣X␣=␣' is stripped from the output and used to control the skipping of lines on the printer. The blank causes a single line to be skipped. The following symbols have special significance if they are used as the first character of the output destined for the line printer:

␣	single space
0	double space
-	triple space
1	new page
+	no advance of the paper (useful for over-printing)

The WRITELN procedure after writing the WRITE parameters causes a new output record to be created (*i.e.* WRITELN adds to an existing record but afterwards appends end-of-line). The WRITE procedure functions similarly to WRITELN in that it also adds on to an existing record, but does not lead to the creation of a new one when the WRITE parameters have been added. For example, the sequence

```
WRITELN('␣A','␣B␣');
WRITE('␣CD');
WRITELN('␣E');
```

would produce

```
A␣B␣
CD␣E
```

The default options for REAL variables allow 24 columns of output to display the number in floating point format, whereas INTEGER variables take 12 columns (leading blanks being inserted where necessary).

Further control over field positions may be obtained as indicated through the following examples. Assume that X is a real variable set to -1.36427815E+02 and I is an integer that has been assigned the value 5392.

```
                                                  printed control
                                                  ↓ ←printed record
WRITELN(X:11,I:6)                  produces       -1.364E+02␣␣5392

WRITELN(X:9:2)                     produces       ␣␣136.43

WRITELN('␣X=␣',X:9:2,'␣I=',I:7)    produces       X=␣␣␣136.43␣I=␣␣␣5392
```

APPENDIX 6E

MORE ABOUT INPUT

On input, the procedure READLN causes the data items associated with the input list to be read and transferred to the appropriate variables. On completion of the transfer, the file is positioned at the start of the next record (card). The procedure READ simply causes numbers next in line on the current input record to be transferred without the input file being advanced to the next record. For example with

```
READLN(X,Y);
READ(Z);
```

the first two numbers are transferred to the variables X and Y and the input file is advanced to the beginning of the next line. The subsequent READ procedure extracts a third number from the next record and places it in Z.

Pascal provides two special functions that are of assistance with input control.

(i) EOF.

(ii) EOLN.

Each function is Boolean (*i.e.* it returns a value of TRUE or FALSE). EOF (end of file) returns a value of FALSE when data cards are still to be processed. It returns a TRUE value only after last data card is read (*i.e.* when an end of file condition is detected). Any attempt to read additional cards after the end of file condition is raised would then cause the program to 'crash' at the execution phase. To prevent this situation from arising when the number of data cards is indeterminate, the EOF function is of use. The following program reads and lists a set of data cards (of unspecified number) containing real numbers and finds their mean value:

```
PROGRAM MEAN(INPUT,OUTPUT);

VAR  SUM,X,MEAN :  REAL;
     NUMBER     :  INTEGER;

BEGIN

SUM:=0.0;
NUMBER:=0;
```

```
WHILE NOT EOF DO

              BEGIN
              READLN(X);
              SUM:=SUM+X;
              NUMBER:=NUMBER+1
              END;

MEAN:=SUM/NUMBER;
WRITELN(' MEAN = ',MEAN)
END.
```

The block of statements to read and accumulate the numbers punched on a set of data cards is executed until the expression

```
NOT EOF
```

is false. The NOT is a logical operator which negates the value of EOF. When the last card has been read, EOF is TRUE and is negated to FALSE by the NOT operator. Hence the block of code is avoided and the mean is calculated and printed.

The EOLN (end of line) function works similarly in that it returns a value of TRUE when the last data item on a current input record has been read.

APPENDIX 6F

CASE STATEMENT

When a two-way decision is required based on the value of a logical expression, the IF-THEN-ELSE construct is ideal. When the choice involves more options, the CASE statement can be of considerable value. A typical case structure is shown in figure 6.14.

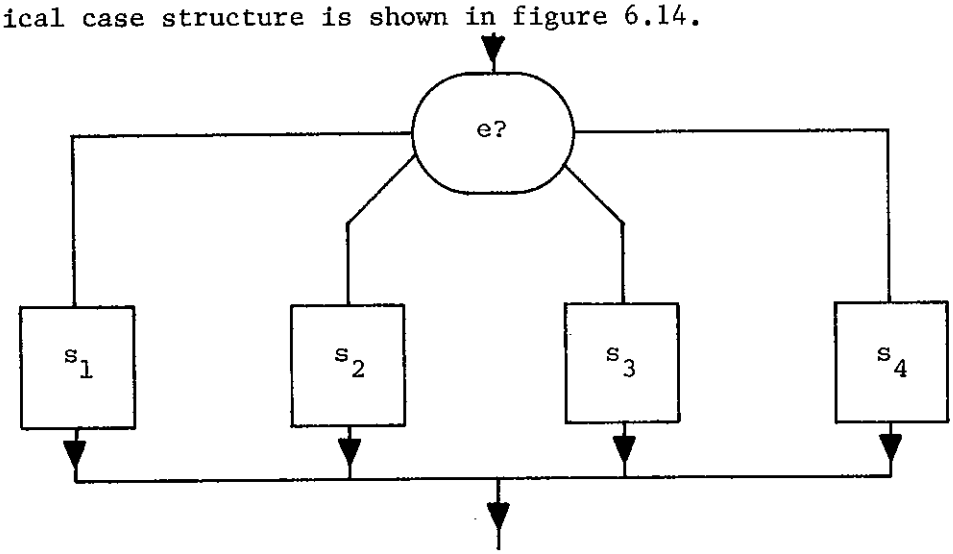


Figure 6.14 CASE - statement structure

Assume I and J are integer variables, then a typical example could be

```
CASE     I+2*J     OF
  1:      X:=X+1.0;
  2:      X:=SIN(X);
 10:      X:=COS(X);

OTHERWISE
         X:=0.0;
         Z:=1.0;
END
```

The CASE selector (I+2*J) must be of INTEGER type for the Summer School data structures. The expression is evaluated and its value determines the subsequent course of action defined by the case-statement-labels. For a value of 1, 2 or 10, the first three statements are chosen respectively. Any other value directs execution to the OTHERWISE. It is not necessary to include an OTHERWISE clause should the programmer not require a 'catch-all'. Failure of the value of the case selector to correspond to a case-statement-label (in the absence of the OTHERWISE), however, leads to an error at the execution phase of the program.

APPENDIX 6G

ANSWERS TO SELECTED QUESTIONS

Q1.

(1) invalid
(2) variable
(3) variable
(4) integer number
(5) invalid
(6) variable
(7) invalid
(8) integer number
(9) variable
(10) invalid (expression)
(11) invalid
(12) invalid (expression)
(13) variable
(14) real number

Q2.

(1) `Y:=0.5*(B+C)`

(2) `Y:=SQRT(B*B-4.0*A*C)/(2.0*A)`

(3) `Y:=(X+A)/4.1415926`

(4) invalid statement (cannot assign a real to an integer)

(5) `I=4`

(6) `I=3`

(7) invalid

(8) `U=1.9`

(9) `K=6`

(10) `I=2`

(11) invalid (attempt to alter a constant)

Q3.

(1)
```
S:=SQRT(X*X+Y*Y+Z*Z)
```

(2)
```
Y:=EXP(X)
```

(3)
```
W1:=EXP(X);
W2:=1.0/W1;
U:=(W1-W2)/(W1+W2)
```

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

(5)
```
C:=-LN(ABS(1.0+A*A*A))
```

(6)
```
W1:=A*X;
Y:=(EXP(W1)+EXP(-SQRT(W1)))*0.33333333
```

Q4.

(1) correct

(2) correct

(3) 1. should be 1.0

(4) limits for a FOR loop must be integer

(5) correct

(6) correct

(7) probable error in logic; there is no escape from the WHILE block because I is never incremented

(8) incorrect ; not permitted before ELSE

(9) incorrect

(10) correct

(11) real to integer assignment not permitted

(12) correct

(13) logical expression should be (A=B) AND (B=C)

(14) correct

(15) incorrect 0...100 should be 0..100

(16) correct

(17) the result of a function evaluation must be returned through the appropriate function name.

Q5.

```
PROGRAM PERIMETER(OTPUT);

CONST PI=3.1415926;

VAR R,PERIMETER : REAL;

BEGIN

R:=1.0;

WHILE R <= 10.0 DO
      BEGIN
      PERIMETER:=2.0*PI*R;
      WRITELN(' PERIMETER OF CIRCLE ',R,' CM IS ',PERIMETER);
      R:=R+0.5
      END

END.
```

Q6.

```
PROGRAM SUM(OUTPUT);

VAR   SUM     : REAL;
      J,ITEM : INTEGER;

BEGIN
```

```
SUM:=0.0;
ITEM:=1;

FOR J := 1 TO 20 DO

                BEGIN
                SUM:=SUM+ITEM;
                ITEM:=ITEM+J
                END;

WRITELN(' SUM ',SUM)
END.
```

Q7.

```
PROGRAM INTEGRAL(OUTPUT);

VAR  SUM  :  REAL;
     I    :  INTEGER;

BEGIN

SUM:=0.5;

FOR I:=1 TO 9 DO  SUM:=SUM+EXP(-I*0.1);

SUM:=0.1*(SUM+EXP(-1.0)/2.0);
WRITELN(' INTEGRAL = ',SUM)

END.
```

Q9.

```
Y:=10.0*RAND
```

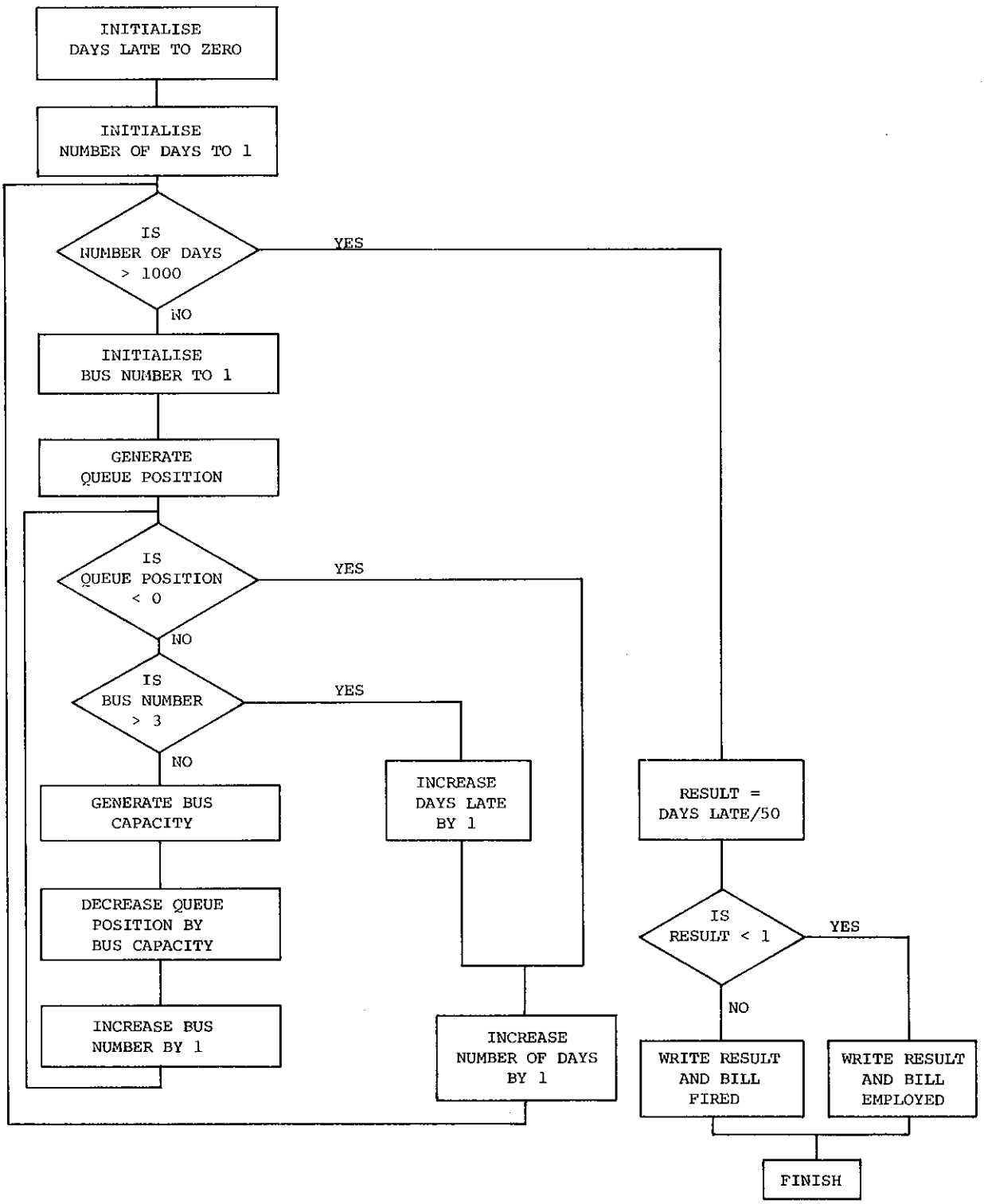
Q10.

```
Y:=0.5*RAND+0.1
```

Q11.

```
   I:=    TRUNC(RAND*15.0)+1
or I:=    ROUND(RAND*14.0)+1
```

Q12.


INITIALISE DAYS LATE TO ZERO
INITIALISE NUMBER OF DAYS TO 1
IS NUMBER OF DAYS > 1000
YES
NO
INITIALISE BUS NUMBER TO 1
GENERATE QUEUE POSITION
IS QUEUE POSITION < 0
YES
NO
IS BUS NUMBER > 3
YES
NO
GENERATE BUS CAPACITY
DECREASE QUEUE POSITION BY BUS CAPACITY
INCREASE BUS NUMBER BY 1
INCREASE DAYS LATE BY 1
INCREASE NUMBER OF DAYS BY 1
RESULT = DAYS LATE/50
IS RESULT < 1
YES
NO
WRITE RESULT AND BILL FIRED
WRITE RESULT AND BILL EMPLOYED
FINISH

```
PROGRAM BUS(OUTPUT);

VAR  NO_OF_DAYS,NO_DAYS_LATE,NO_QUEUE,BUS_NO,
     NO_ON_BUS,BUS_CAPACITY  : INTEGER;
     DAYS_LATE               : REAL;

DECLARE RND

BEGIN
NO_DAYS_LATE:=0;

FOR NO_OF_DAYS := 1 TO 1000 DO

     BEGIN
     NO_QUEUE:=TRUNC(RND*11+1);
     BUS_NO:=1;

     WHILE (NO_QUEUE>0) AND (BUS_NO<=3) DO

         BEGIN
         BUS_CAPACITY:=TRUNC(RND*16);
         NO_QUEUE:=NO_QUEUE-BUS_CAPACITY;
         BUS_NO:=BUS_NO+1
         END;

     IF NO_QUEUE > 0 THEN NO_DAYS_LATE:=NO_DAYS_LATE+1

     END;

DAYS_LATE:=NO_DAYS_LATE/50.0;
IF DAYS_LATE < 1.0 THEN WRITELN(' BILL EMPLOYED ')
                   ELSE WRITELN(' BILL FIRED ')

END.
```

Q13.

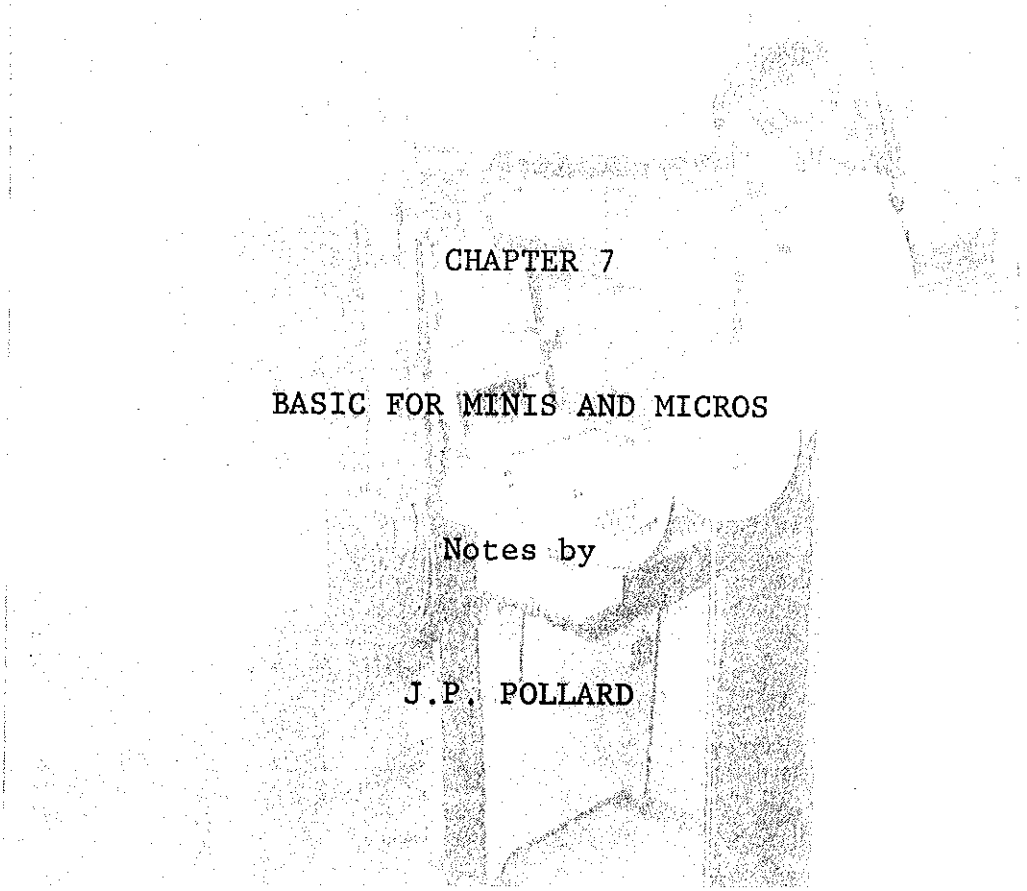
```
PROGRAM INTEGRAL (OUTPUT);

VAR  A,B,H,SUM  :  REAL;
     N,I        :  INTEGER;

FUNCTION  F(Z:REAL) : REAL;
BEGIN
F:=4.0+Z*(2.0+Z*(3.0+4.0*Z)
END;

(* MAIN PROGRAM *)

BEGIN

(* SPECIFY NUMBER OF POINTS, LOWER AND UPPER INTEGRAL BOUNDS *)

N:=5;
A:=0.0;
B:=1.0;

(* CALCULATE STEP SIZE *)

H:=(B-A)/(N-1);
```

```
(* NOW PERFORM INTEGRATION *)
(* FIRST DO TWO END POINTS IE X=A AND X=B *)

SUM:=F(A)+F(B);

(* THEN DO THOSE WITH COEFFICIENT 4 *)

I:=2;
REPEAT
SUM:=SUM+4.0*F(A+(I-1)*H);
I:=I+2;
UNTIL I>N-1;

(* THEN DO THOSE WITH COEFFICIENT 2 *)

I:=3;
REPEAT
SUM:=SUM+2.0*F(A+(I-1)*H);
I:=I+2;
UNTIL I>N-2;

SUM:=H*SUM/3.0;
WRITELN(' INTEGRAL = ',SUM )
END.
```

Q14.

```
PROGRAM SORT(INPUT,OUTPUT);

VAR  TEMP    : REAL;
     I,J     : INTEGER;
     X : ARRAY (.1..10.) OF REAL;

BEGIN

FOR I:= 1 TO 10 DO  READLN( X(.I.) );

FOR I:=1 TO 9 DO

          BEGIN
          FOR J:=I+1 TO 10 DO
                    IF X(.J.)>X(.I.) THEN
                                        BEGIN
                                        TEMP:=X(.J.);
                                        X(.J.):=X(.I.);
                                        X(.I.):=TEMP
                                        END;

          END;

FOR I := 1 TO 10 DO WRITELN(X(.I.))

END.
```

Q15.
```
PROGRAM CUBIC (INPUT,OUTPUT);
VAR  A,B,C,D,XN,XNP1   :  REAL;
     I                 :  INTEGER;

BEGIN
(* PROGRAM TO FIND ROOTS OF A CUBIC
   COEFFICIENTS OF CUBIC READ FROM PUNCHED CARD *)

WHILE  NOT EOF  DO
                BEGIN

                READLN(A,B,C,D);
(* READ ESTIMATE FROM PUNCHED CARD *)
                READLN(XNP1);
                I:=1;
                REPEAT
                XN:=XNP1;
                XNP1:=XN-(A+XN*(B+XN*(C+D*XN)))/(B+XN*(2.0*C
                        +XN*3.0*D));
WRITELN(XNP1,XN);
                I:=I+1;
                UNTIL   (ABS((XNP1-XN)/XN)<0.001)
                   OR   (I>9);

                IF  ABS((XNP1-XN)/XN)<0.001
                    THEN WRITELN(' ROOT IS ',XNP1)
                    ELSE WRITELN(' NO ROOTS POSSIBLE ')
                END

END.
```



CHAPTER 7

BASIC FOR MINIS AND MICROS

Notes by

J.P. POLLARD

7.1 INTRODUCTION

At the present time, the possible use of microcomputers in the high school has aroused much interest. Some schools already have such machines; others are contemplating getting them. Well, you may ask, what is a microcomputer? How does it differ from a large scientific computer, such as the IBM3031 to be used at this Summer School? The real answer to these questions will come when we see one for ourselves. Just now, we will be content to note that the most important feature is probably the price. A reasonable machine can be obtained for about $1000 to $2000. Almost all microcomputers available accept the BASIC language - some nothing else - hence our interest in that language at this Summer School.

BASIC was originally developed for beginner programmers by Dartmouth College, New Hampshire, USA in 1964. Since then, many different versions ('dialects' of BASIC) have arisen. If we were to restrict ourselves to the subset common to them all, we might find that the only type of statement available would be

```
10 REM  THIS IS IT
```

- a remark made to help us find things in a program when listed but otherwise ignored by the machine.

Here, we will meet a useful version of BASIC, which will at least run on a PDP11/45 minicomputer connected to various terminals at Lucas Heights and which will also run on some microcomputers.

Unlike FORTRAN, BASIC is an interactive language (usually) hence the main output of data is likely to come from a terminal (monitor, teletype or whatever) rather than punched cards. We may thus make up our minds as we go. Depending on what happens, we may readily change the program and data to meet contingencies not originally envisaged.

7.2 GETTING THROUGH

Naturally, the computer system will not want to speak to us unless we are bonafide. We must therefore find a terminal and sign on in an acceptable way. During the week of the Summer School, we are given two 'passwords':

1. initials: SSK (for Summer School Kids), and
2. ID: AM290060

which will get us through to BASIC. In the following cryptic dialogue between us and a computer (not actually the PDP11/45) the underlined

items are *machine responses* and ↘ denotes the carriage RETURN or ENTER key. Here we go...

(1) Push (not hit) space bar

(2) ID: AM290060↘ (this won't print in case someone else is watching)

(3) $2↘

(4) login:SSK↘ (upper or lower case letters here select the mode for the run)

(5) % BASIC ↘

...

(6) ready *(A MICROCOMPUTER WILL PROBABLY START HERE)*

our BASIC program

```
10 REM THIS IS WHERE BASIC GOES - AT LAST!
...
...
```

7.3 LET'S RUN

Here is a simple program to prepare a table of 21 values of $y = e^x$ for x = -10,-9,...,0,...,9,10. A (WATFIV) FORTRAN version is shown for comparison.

BASIC	FORTRAN
⌜ line numbers determine normal program 'flow'	⌜statement numbers not necessary every line
10 REM PRØG. TØ TAB. EXP(X)	C PRØG. TØ TAB. EXP(X)
20 X=-10.	X=-10.
30 FØR I=1 TØ 21	DØ 70 I=1,21
40 Y=EXP(X)	Y=EXP(X)
50 PRINT X,Y	PRINT,X,Y
60 X=X+1.	X=X+1.
70 NEXT I	70 CØNTINUE
80 STØP	STØP
90 END	END
⌞ normally initially 10 apart for later insertions	
We type in the prog. (in any order of line nos.)	We punch the prog.
We type LIST to check statements.	We list the cards to check them.
We type RUN to start prog.	We submit to run as a batch job.

We note the close similarity to FORTRAN (or at least its WATFIV dialect) and the subtle differences (no comma after PRINT for example). The FOR-NEXT loop is similar to the DO-CONTINUE (or whatever terminating statement) loop. A counter (any variable in BASIC, not just I,J,...,N type fixed point variables of FORTRAN) is incremented, usually in steps of 1, until the terminating value is reached. We may however directly specify negative values. For example, we may modify our BASIC program thus:

delete statement 20 by typing

```
20  and nothing else,
```

change statement 30 by entering

```
30 FØR X=-10 TO 10
```

delete statement 60 with

```
60
```

and *change* 70 to

```
70 NEXT X
```

The typed command LIST would then give

```
10  REM PRØG. TØ TAB. EXP(X)
30  FØR X=-10 TO 10
40  Y=EXP(X)
50  PRINT X,Y
70  NEXT X
80  STØP
90  END
```

With a slight modification to the program, we may start with the highest value...

change statement 30 by typing

```
30  FØR X=10 TØ -10 STEP-1
```

As for FORTRAN, the loop limits may actually be specified by variables rather than constants. Some BASICs permit non-integer values (like STEP 0.5), and indeed our BASIC does this whereas others just don't; they work with the integers equal to or just below the numbers.

Like FORTRAN, sensibly nested loops are permitted, *e.g.*

```
100  FØR I=1 TØ 10
110  FØR J=1 TØ I
  :    (coding which does not change the loop variables - I
  :     and J here)
```

```
200  NEXT J
210  NEXT I
```

Note - the biggest line number permitted is usually 32767.

7.4 WHAT'S IN A NAME?

Normally BASIC has only two types of variables:

(1) Numeric variables with names

A,B,C,...,Z and

A0,A1,A2,...A9; B0,B1,B2,...B9; ...Z0,Z1,Z2,...,Z9

- the numeric variable I7 could contain one of the numbers; *e.g.*

0, 27, -2183, 1.63, 3.141592, 7.65E+19, 1.139261E-2

(in fact any positive or negative number between about 1.E-38 to 1.E38, or 0).

(2) String variables, with names

A$, B$, C$,...,Z$ and

A0$,...,Z9$

- the string variable A$ could contain up to a line of text characters, *e.g.* "JOHN", "YES", "NO(S) LEFT", "**MARY'S GO**"

Note - text is enclosed between double quote marks.

Assignment of variables is of the style

```
A1=12.5
C3=A1
A1$="*"
B$=A1$
```

(and some older style BASICs want you to let them know that the statements are assignments thus,

```
LET  A1=12.5
LET  C3=A1,  etc.
```

but please forget you were ever told about this).

Most versions of BASIC permit indexed arrays, *e.g.* A(112), but we will not be concerned with such things here.

7.5 OPERATIONS TO REMEMBER

The BASIC expression

```
Y=(1+X*X)/(1-X↑N)
```

calculates

$$y=(1+x^2)/(1-x^n)$$

and differs only from the FORTRAN counterpart in the use of ↑ (or ^ for some keyboards) for exponentiation (raising to a power). The example

illustrates all you need to remember.

7.6 GO TO AND OTHER DEVIATIONS

The regular flow of BASIC when running is from the smallest line number to the next in order, no matter in which order the statements are typed. We have seen a FOR-NEXT combination to establish a loop back and forth. Here are other statements that cause deviations to the regular flow:

(1) GOTO line number (alternatively GO TO line number)

e.g. GOTO 10

(2) IF $\&_1$ { = | > | < | >= | <= | <> } $\&_2$ THEN statement

=	equals
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
<>	not equal

where $\&_1$ and $\&_2$ are arbitrary *expressions*
and *statement* is almost any BASIC statement, *e.g.*

```
IF A<B-1   THEN Z=1
```

except that (for some BASICs) another IF cannot follow and *only the line number* must be given *for a GOTO*. Thus

```
IF X1>0  THEN  200
```

will transfer control to line 200 if X1 is positive. (Some newer style BASICs do not insist on the THEN as part of the IF - but we will.)

(3) ON & GOTO $n_1,n_2,n_3,\ldots$

where & is an expression (*e.g.* I, B+1, *etc.*)
and if $1 \leqslant \& < 2$ control is transferred to line number n_1,
$2 \leqslant \& < 3$ control is transferred to line number n_2,
$3 \leqslant \& < 4$ control is transferred to line number n_3, *etc.*
In the BASIC program that we will use, if & cannot find an appropriate n, (*e.g.* if & = 0), then an error condition arises.

(4) STOP

- does just that.

(5) END

- similar to STOP but some BASICs can only tolerate one of these and then normally only as the final statement. Being FORTRAN users, we will be happy to finish a BASIC program with END.

7.7 BUILT-INS

Like FORTRAN, BASIC has a suite of built-in functions. These are as follows:

(1) ABS(X)= $\begin{cases} +X & \text{if } X \geqslant 0 \\ -X & \text{if } X<0 \end{cases}$ *e.g.* ABS(-2.7)=2.7

(2) INT(X)= integer, equal to, or just smaller than X, *e.g.* INT(3.14)=3 *but* INT(-3.14)=-4

(3) SQR(X)= $\begin{cases} \sqrt{X} & \text{if } X \geqslant 0 \\ \text{error otherwise} \end{cases}$

(4) EXP(X)= e^X

(5) LOG(X)= $\begin{cases} \ell n\ X \text{ if } X>0 \\ \text{error otherwise} \end{cases}$

(6) SIN(X)= sin x - note that x must be in radians (1 radian = $180/\pi$ degrees)

(7) COS(X)= cos x

(8) TAN(X)= tan x

(9) ATN(X)= $\tan^{-1}x$ (*i.e.* arctan x)

(10) RND(X)= a pseudo-random number, bigger than 0 and smaller than 1

(an argument is to be supplied but is ignored)

(11) RANDOMIZE with no argument

- makes sure that successive runs of the same program give different random numbers for the first use of RND(X) - otherwise a chess playing program would always open 'pawn to queen 4'.

7.8 LET'S PUT IT TOGETHER

We now put together a simple BASIC program to calculate the highest common factor (HCF) of two positive integers M and N using Euclid's algorithm (method):

```
10  REM HCF PRØG
20  M=85
30  N=204
40  GØTØ 80
50  PRINT H
60  STØP
70  REM
80  REM START ØF EUCLID RØUTINE
```

```
 90  H=N
100  N=M-N*INT(M/N)
110  M=H
120  IF N<>0 THEN 80
130  GØTØ 50
140  END
```

Follow the program flow yourself to see what is happening. Our variables change thus:

	initially	1st pass	2nd pass	3rd pass	4th pass	
H	-	204	85	34	17	← answer
N	204	85	34	17	0	← finish ∴
M	85	204	85	34	17	

Surely we don't have to keep changing lines 20 and 30 if we want to calculate the HCF of other numbers? No!

7.9 I/O U

We have seen how to output a number on the terminal with, for example,

```
PRINT U
```

Surely to input a number from a terminal we would use

```
INPUT U
```

- yes!

The ambitious user of BASIC might also want to input a string of characters (perhaps the name of the person running the program). We would simply use, for example,

```
INPUT A$
```

and later we would output with

```
PRINT A$
```

Our HCF program may be modified to input M and N thus:

```
20  INPUT M,N    (and at the appropriate RUN stage we enter,
                  for example, 85,204)
30       and nothing else (to delete the line)
60  GØTØ 20      (to always return to input two more values
                  for M and N)
```

The input described above is for interactive use where we may change our minds depending on the printed results achieved so far. (In a sense we are part of a loop with the computer.) However, say we only

want to enter occasionally varying data such as the constants t_o, x_o, λ, β of our Summer School project. We enter such (essentially fixed) data with a statement consisting of a line number followed by the control word DATA, *e.g.*

```
1000 DATA 1909,4.29,0.07,0.07
```

which may appear anywhere in the program (although being FORTRAN users we would probably put it towards the end, *i.e.* give it a high line number). As with FORTRAN, the data are not entered into the computation until a READ is issued, *e.g.*

```
READ U
```

which will read the next word of data from a DATA statement. If previous READs have exhausted all items of one DATA statement, then further input will come from the next DATA statement in the program (if one exists). Thus we could equally well have used for our example

```
1000  REM TØ,XO
1010  DATA 1909,4.29
1020  REM L,B
1030  DATA 0.07,0.07
```

which we could enter into the program with, for example, the statement

```
READ TØ,XO,L,B
```

When we RUN a job, the READ *pointer* begins from the first word of the first DATA statement and then every successive READ moves the pointer down the DATA list. We cannot *jump over* unwanted data (except with *dummy* READ statements); however, we can always restore the pointer to the beginning with the statement

```
RESTORE
```

7.10 ROUTINE MATTERS

When a portion of BASIC code forms a routine to do a specific job we would like to be able to jump to the routine and return to the line that follows (*cf.* a FORTRAN subroutine). The need becomes more apparent when different parts of the main coding require *calls* to the one routine, for then the return is to be made to different parts of the program. We might as well modify our HCF program to include a subroutine for the Euclid algorithm. We type

```
40  GØSUB 80
130 RETURN
```

Get the idea?

7.11 PRETTY PRINT

If we run our HCF program we find it is a bit terse, for it says

```
?
```

when we are to supply input. In BASIC (and WATFIV, FORTRAN) we can easily arrange to print descriptive information (without the labour entailed by string variables). We simply enclose the information between double quotes (single quotes for WATFIV) as part (or all) of the PRINT statement. Our HCF program could thus be further modified:

```
15  PRINT "ENTER VALUES FØR M AND N"
50  PRINT "THE HCF IS";H
60  GØTØ 15
```

No, the typist did this correctly - a semicolon implies a short skip after the preceding number or characters, whereas a comma implies a long skip so that precise columns of printout are maintained. Try it for yourself! You might even be bold enough to try

```
15  PRINT "ENTER VALUES FOR M AND N";
```

A feature for establishing printed output at specific print positions is the TAB(X) function. The printed columns are usually numbered 0,1,2,...,71 and TAB(X) for X=3.141592, for example, would set the subsequent print to begin at column number 3 (the INT part of X). We might then decide to place our result for the HCF of two numbers in the middle of the line thus:

```
50  PRINT TAB(25);"THE HCF IS";H
```

With a little pondering you might also see how rough graphical output, say for the function Y=EXP(X), may be obtained using

```
PRINT TAB(Y);"*"
```

We are hardly likely to remember all the changes that have been made to our HCF program. Let us do another

```
LIST

 10  REM HCF PROG
 15  PRINT "ENTER VALUES FØR M AND N";
 20  INPUT M,N
 40  GØSUB 80
 50  PRINT TAB(25);"THE HCF IS";H
 60  GØTØ 15
 70  REM
 80  REM START ØF EUCLID RØUTINE
 90  H=N
```

```
100  N=M-N*INT(M/N)
110  M=H
120  IF N<>0 THEN 80
130  RETURN
140  END
```

7.12 HAVE A BREAK

Say that in our HCF program we had mistakenly typed line 100 as

```
100  N=M
```

and we had requested the job to run

```
RUN
```

We would gain the (wrong) impression that the machine was slow for we would have no more response out of it. But no - we are running a program that is caught up in a never ending loop. How can we (or rather the program) escape? Simply this - we just depress the two keys

CTRL Z (actually we hold the first key down and then depress the second).

Thank goodness! Imagine the computer bill if we had had no way to break out of that loop!

A similar feature holds for interrupting a LIST:

CTRL S gives a temporary interruption to the LIST and

CTRL Q resumes the LIST.

We may, however, select to LIST only a portion of our program from the outset. For example, if we only want to list the EUCLID ROUTINE of our HCF program, we would use

```
LIST  80-130
```

7.13 SAVE IT

Having gone to a lot of effort to type in a program, we would obviously like to save it for later use. For the minicomputer setup used at this Summer School, the program would be saved on disk, whereas for a microcomputer setup the program would be saved on an ordinary audio-cassette. Here we go:

```
SAVE   int.HCFJOB
```

where int are *your* 3 initials used throughout the School (not SSK) and HCFJOB (or the like - a name having up to 8 characters) denotes the job. Thus I might use

```
SAVE   JPP.TESTPRGM
```

Should the job already have been saved you would need to say, for example,

```
REPLACE   JPP.TESTPRGM
```

7.14 ANY OLD JOBS

Of course saving a job would not be of much use if we could not reload it (usually at some other time). The command is simply this, for example,

```
ØLD   JPP.TESTPRGM
```

which we would presumably then want to

```
RUN
```

or we could combine the two operations into one

```
RUN   JPP.TESTPRGM
```

7.15 SIGNING OFF

When we have finished a session (and perhaps SAVEd our job), we will terminate with the following signing off process:

(1) BYE - end of connection to BASIC

(2) [CTRL] [D] - end of connection to PDP11/45

(3) [CTRL] [P] - end of connection to the computer system.

7.16 WHAT COULD BE MORE BASIC?

Besides other BASIC statements which will not concern us (for it is here that we encounter most differences from microcomputers) we can use the machine as a calculator. If we leave out a line number, then the instruction is carried out immediately. As a simple example we could calculate $3e^2$ using the coding

```
PRINT 3*EXP(2)
```

which would return the result immediately the carriage return was depressed. Similarly, the line

```
X=3*EXP(2):PRINT X
```

would achieve an equivalent result (except that, in addition, the answer would be stored in X) where ':' separates different statements on the one line. (\ is an alternative statement separator for some versions of BASIC.)

7.17 TRY THIS FOR YOURSELF

During the Summer School, you will be given the opportunity to 'have a go' for yourself. Three jobs are detailed here - everyone should try the first one; a few might get through to the third one.

(1) Write a BASIC program to display pseudo-random numbers. Print 10 such numbers and stop. How could you get a different set of 10 numbers for different runs?

(2) Write a BASIC program to simulate the roll of a dice 1000 times. Report the number of throws that achieve a 1. Use the RANDOMIZE statement and run the job 3 times to get some *feel* for the dispersion of the results about the expected mean of 167.

(3) If you have finished the Summer School project, write a BASIC program for the game MRMIND described in Appendix 7A.

Note Appendix 7B gives a list of BASIC statements that may help you to get started.

APPENDIX 7A
FOR THOSE WHO HAVE FINISHED THE SUMMER SCHOOL PROJECT: MONTE CARLO SIMULATION

7A.1 INTRODUCTION

The study of (1) neutrons moving through absorbing material; (2) queuing problems of people at a supermarket exit; (3) preparation of high school timetables; and (4) fun and games can be assisted by simulation of the processes. The simulation consists of studying hosts of different possibilities pseudo-randomly chosen on a computer. Since you will undoubtedly find some games to play at a computer terminal, the idea here is to give you some idea of *what happens after you hit carriage-return at the terminal.*

A basic requirement of Monte Carlo simulation is to generate pseudo-random numbers r_1, r_2, r_3, ... from a given starting value to r_0. Since we can write down the outcome, the sequence is reproducible (provided that we stay with the one computer) but, to all intents and purposes, we can consider the set of numbers $\{r_0, r_1, r_2, \ldots, r_n\}$ to be random and uniformly distributed throughout the interval (0,1). With BASIC we obtain a random number in the following way:

```
RANDOMIZE     - initial statement at beginning of the program
  :             to yield a 'random' starting number.
R=RND(0)      - later on, then R is a random number (and,
                since we began with RANDOMIZE, the BASIC
                system returns to us its next random number
                rather than the first of a reproducible
                sequence).
```

Then `I=INT(10*R)`
is a random integer (digit) 0,1,2,3,4,5,6,7,8 or 9 since INT takes the integer part of 10 times R. Our next use of the RND function will then return a different number, and so on. After, say, 1000 calculations of I we might produce the results.

I	No. of times
0	103
1	104
2	98
3	103
4	93
5	100
6	98
7	92
8	101
9	108

7A.2 ON WITH THE JOB

Use these ideas to write a BASIC version of

MISTER MIND

(Whereas *master mind* uses assorted colours, *mister mind* uses integer digits 0 to 9. The game is an adaptation of BAGLES taken from the DEC manual, 101 BASIC COMPUTER GAMES, 1975, p.22.)

In brief, the computer is required to obtain 3 random integer digits (same as 'I' earlier) imagined to be the leading, middle and trailing digits of a '3 digit number' between 000 and 999. A player is permitted to have 20 goes at attempting to guess the digits. After every go, the program is to return a clue in the form of two numbers

D = the number of correctly positioned digits, and

M = the number of matches of the player's digits with the machine's digits, including any already counted in D.

A simple program would input player's digits one at a time. A more complicated program would input a three-digit number and internally strip off the digits.

As an example, say the sought after number is 346 and the opponent's number is 640; then we print

D=1 (*i.e.* 1 correct digit, the 4)

and M=2 (*i.e.* 2 matched digits, the 6 and the 4).

Or if the sought after number is 355 and the opponent's number is 505 then we have

D=1 (*i.e.* the last 5)

and M=4 (*i.e.* the first 5 matches the last two and the last 5 does the same).

APPENDIX 7B

A LIST OF BASIC STATEMENTS VIA AN EXAMPLE:

SOLUTION OF QUADRATIC EQUATIONS

```
ID:    AM290060  (the machine actually replies ********)
$2
```

see chapter

getting through 7.2

```
login: ssk
```

Note - depending on the mode of reply of the password SSK (upper or lower case) so that mode will be maintained for the run. However most BASICs work with capitals only.

```
% basic

Ready

old jpp.quad                                          old 7.4

Ready

list                                                  list 7.3
jpp22-Nov-7807:08 AM
10 rem soln of quadratic eqns                         rem 7.1
20 print "soln of quadratic eqns"                     print 7.3,7.9,7.11
30 read n                                             read 7.9
40 print "first";n;" examples"                        ;with print 7.11
50 for i=1 to n                                       for 7.3
60 read a,b,c
70 gosub 900                                          gosub 7.10
80 next i                                             next 7.3
90 rem over to user
100 print  : print "please supply name";               : mult.stmts.per line 7.16
110 input a$                                          input 7.9 & string
120 print "now your turn ";a$                         variables($) 7.4
130 print
140 print a$;" please enter coeffs in a*x*x+b*x+c=0"
150 print "a,b,c=";
160 input a,b,c
170 gosub 900
180 go to 130                                         goto 7.6
900 rem print problem then solve
910 print "soln of";a;"*x*x+";b;"*x+";c;"=0 is..."
920 gosub 1000
930 return
1000 rem routine for extracting roots (real or complex)
1010 if a=0 then print "that's not a quadratic" : stop   if 7.6,stop 7.6
1020 d=b*b-4*a*c                                      numeric variables 7.4
1030 k=1                                              & operations 7.5
1040 b$="     "                                       string assignment 7.4
1050 if d<0 then d=abs(d) : k=2 : b$="+/-i"           functions 7.7
1060 w=2*a          (note - all this applies if d negative)
1070 x1=-b/w
1080 x2=sqr(d)/w
1090 on k go to 1100,1130                             on goto 7.6
1100 w=x1
1110 x1=x1+x2
1120 x2=w-x2
1130 print "x1&x2=";x1;b$;x2
1140 return                                           return 7.10
2000 rem test data n then (a,b,c),...
2010 data 2                                           data 7.9
2020 data 1,-3,2
2030 data 7,1,2
32767 end                                             end 7.6

Ready
```

```
run
JPP22-Nov-7807:09 AM
soln of quadratic eqns
first 2  examples
soln of 1 *x*x+-3 *x+ 2 =0 is...
x1&x2= 2          1
soln of 7 *x*x+ 1 *x+ 2 =0 is...
x1&x2=-.714286E-1 +/-i  .529728

Please supply name? John
now your turn John

John Please enter coeffs in a*x*x+b*x+c=0
a,b,c=? 1,0,1
soln of 1 *x*x+ 0 *x+ 1 =0 is...
x1&x2= 0 +/-i 1

John Please enter coeffs in a*x*x+b*x+c=0
a,b,c=? 12,8,2
soln of 12 *x*x+ 8 *x+ 2 =0 is...
x1&x2=-.333333 +/-i  .235702

John Please enter coeffs in a*x*x+b*x+c=0
a,b,c=? 8,4,1
soln of 8 *x*x+ 4 *x+ 1 =0 is...
x1&x2=-.25 +/-i  .25

John Please enter coeffs in a*x*x+b*x+c=0
a,b,c=? 4,8,1
soln of 4 *x*x+ 8 *x+ 1 =0 is...
x1&x2=-.133975       -1.86603

John Please enter coeffs in a*x*x+b*x+c=0
a,b,c=?
Ready

bye
%
connect  time      2:33.00
user cpu time         1.74
sys  cpu time         4.16
$
END-SESSION
```

run 7.3 & 7.14

CTRL Z break 7.12

BYE
CTRL D
} signing off 7.15
CTRL P

CHAPTER 8

THE AAEC CENTRAL COMPUTER AND ITS NETWORK

Lecture by

D.J. RICHARDSON

CONTENTS

8.1 INTRODUCTION

Computing at the Australian Atomic Energy Commission's Research Establishment commenced in the early 1960s when the Commission rented a small IBM1620 computer, with 60 000 characters of storage. The Commission has rented its central computer ever since; this has enabled it to keep pace with the growing computing needs of the last two decades by changing central computing equipment as the need arose. At present, the central computer installation is based upon an IBM3033 central processing unit with over 8 million bytes of main memory.

Design and development of the Research Establishment's computer network started just over 10 years ago. Since then it has grown to include over a dozen interlinked computers supporting more than 60 computer user terminals for interactive computing.

8.2 CENTRAL COMPUTER HARDWARE

The IBM3033 central processor is a member of the IBM370 range of computers. Auxiliary storage comprises four IBM3350 disk modules, each with two disk spindles, and four IBM3420-8 magnetic tape drives which provide virtually unlimited magnetic tape storage facilities.

Each IBM3350 disk spindle can store up to 317 million bytes of data. An average text-book with 500 pages, each page having 45 lines of text and each line having 70 characters, contains approximately 1½ million characters. Thus the contents of over 200 average text-books could be stored on each IBM3350 spindle. In addition to disks and magnetic tape units, a card reader, line printer and paper tape reader are attached to the central processor.

The IBM3033 central processor is controlled by two micro-processors, each equipped with a computer operator's console. These micro-processors, either of which may take over in the event of failure of its partner, control the powering up and down and the input/output (I/O) activity of the whole installation.

8.3 CENTRAL COMPUTER SOFTWARE

In the early days of computing, each computer user was required to provide every facet of the computer job personally. The computer merely provided the necessary hardware for running the job. The first user aids were assembler programs, which enabled mnemonics to be used instead of absolute computer instructions, and input/output subroutine packages, which greatly simplified the task of communicating with the computer. Users would queue in succession to read their card decks or paper tape rolls into the computer. It was not possible, in those days, to hide

one's program mistakes!

The first computer operating systems merely provided a means for queuing jobs on the computer input device, with an embryonic control program which enabled control (hopefully!) to pass from the completion of one job to the start of the next.

As computer software knowledge increased, more and more functions were built into the system control program, including the handling of all input or output on behalf of the user. Safeguards were provided in the hardware, to ensure the possibility of protecting this system control program against accidental or deliberate damage.

The introduction of the concept of a computer hardware interrupt, which enabled external events or unexpected internal program events to cause a transfer of control within the central processor, allowed the development of multi-processing. Modern computers use interrupts to enable present-day control programs to run many jobs, apparently in parallel, and to schedule their own work-loads.

Limitations in available main memory for central processors led to the development of the concept of virtual storage. Real storage is divided into units, usually of 2048 or 4096 bytes, and areas of the auxiliary disk storage are set aside and also divided up into these unit sizes. Each user can then be allocated an apparent area of virtual storage for programming use. Although active parts of the program must reside in real storage, inactive parts may be kept on disk. The control program uses an address translation mechanism to keep track of the various units of a user program, transferring these units from or to disk as required.

The control program in use on the IBM3033 computer is called MVS (Multiple Virtual Storages). This control program supervises all aspects of central computer use, allowing both batch and interactive jobs, and maintains a log of such usage, together with an analysis of encountered errors, whether due to computer hardware or program software.

8.4 DEVELOPMENT OF THE AAEC COMPUTER NETWORK

The first stage of the AAEC computer network was the connection of a Digital Equipment Corporation mini-computer, a PDP9L computer with 8192 eighteen-bit words, to the existing central computer, an IBM360/50. This was achieved by a locally designed link in such a way that the PDP9L appeared like a standard range of IBM hardware to the central processor (Richardson, 1969). The second stage of the network was the

development of a high speed party line, called the Dataway, which provided a means for interconnecting the various mini-computers in use at the Research Establishment, including the PDP9L computer, such that any two computers on the Dataway could communicate with each other in a symmetric manner (Ellis, 1970).

The first computers connected to the Dataway were the PDP9L computer - and hence, indirectly, the central computer - and a Data General mini-computer, the NOVA 2600. The NOVA computer had previously provided an interactive computing service to five attached teletypes, using the language ACL-NOVA designed at the Research Establishment (Bennett and Sanger, 1973).

With the effective connection of the NOVA computer to the central computer, a network communication system for the NOVA computer was developed to enable the terminals attached to the NOVA computer to access facilities on the central computer as well as the local ACL-NOVA language. This NOVA communication system was called DATERCOM (Sanger, 1976). In a manner analogous to the telephone network's STD system, the symbol $ as a first character was reserved to indicate a desire to communicate outside the NOVA system, *i.e.* to the central computer.

The third stage was the development of a relatively inexpensive polled party line connection to the NOVA computer, for up to 64 terminals and display units whose speeds did not exceed 300 characters per second. This connection was given the name SMUT (Serial Multiple User Terminals system - Ellis, 1977).

As more and more mini-computers were attached to the Dataway, communications other than to and from the central computer were needed; to facilitate these, a more general computer-to-computer protocol was devised (Richardson & Sanger, 1978). DATERCOM was also extended to allow a terminal to give the desired computer destination as a decimal number immediately following the first $ symbol, the default (no number) being the central computer.

At the present time, 14 mini-computers are directly connected to the Dataway. In addition, there are terminals directly connected to the Dataway using a sub-set of the full Dataway connection protocol. These directly connected terminals appear like one-byte computers to the Dataway!

A micro-NOVA computer attached to the Dataway services four automatic answer telephone modems and lines for remote users to dial in to the AAEC network.

Apart from the central computer, two large Digital Equipment PDP11 computers, used respectively for graphics applications and a library information service, are the most popular network destination computers, especially for text editing. These two computers each run under the UNIX operating system developed at Bell Laboratories in the U.S.A.

The AAEC computer network is open-ended in that it is not necessary for any one node to be aware of all other nodes but only of those nodes of relevance to it.

At present, research at the AAEC in the computer network area is aimed at providing an ever widening range of facilities to the computer user, especially those facilities related directly to the central IBM3031 computer. The use of micro-computers to provide alternative paths and cross links within the AAEC network is also being investigated.

8.5 USER COMPUTING PERSPECTIVES

The interface between a computer system and the computer user is of vital importance. In the early days of computing, this interface was a card or paper tape reader and a line printer which either directly printed output or listed cards punched as intermediate output. This type of interface still exists in some small firms and University environments. In most present-day installations, users with this type of computer input and output, interface with computer staff, providing coded sheet input and receiving printed output *via* pigeon holes or a mailing system. The growth of interactive, personal terminals as user-computer interfaces has once again provided users with a 'feel' for the computer to which they are connected.

To be of maximum benefit to a user, a computer terminal should:

(i) be easy to use, and

(ii) provide an adequate fast response.

The first condition requires adequate documentation, either separately from the terminal, or available by interrogation through the terminal itself. The second is more subtle. There is consensus that a response to any terminal input should be received within one or two seconds, otherwise the user ceases to view the terminal as a transparent aid to requirements. Some installations program a minimum response time into their interactive systems so that users will be insulated to some degree from computer load fluctuations!

Computer terminals, either printing or video display, will be the main user-to-computer communication media for many years to come. The

possibility of using voice input has been under investigation for some years now, but two main difficulties have yet to be overcome: recognition of speech itself, with its varying accents; and interpretation of natural language commands. Even so, limited use is being made of voice input in some specialised applications.

One thing is certain. The use of computers, and the ease of their use, will continue to grow and improve.

8.6 REFERENCES

Bennett, N.W. & Sanger, P.L. (1973) - The Development of the ACL Language and its Implementation ACL-NOVA. Aust. Comp. J., 5(3) 105-113.

Ellis, P.J. (1970) - A Multiplexed Computer-Computer, Computer-Device Data Link. AAEC/E206.

Ellis, P.J. (1977) - SMUT - Serial Multiple-User Terminals System. AAEC/E433.

Richardson, D.J. (1969) - A Generalised Computer to Computer Link for an IBM 360 Computer. Aust. Comp. J., 1(5) 273-276.

Richardson, D.J. & Sanger, P.L. (1978) - Inter-Computer Communication within a Multinodal Computer Network. Proc. 8th Australian Computer Conference, 28th August to 1st September, Canberra, A.C.T. pp. 1591-1608.

Sanger, P.L. (1976) - The AAEC Dataway Terminal Communication System. Proc. 7th Australian Computer Conference, 30th August to 3rd September, Perth, W.A. pp. 610-622.

CHAPTER 9

NOT ENOUGH ENERGY - OR NOT ENOUGH TIME?

Lecture by

K.J. MAHER

(CSIRO DIVISION OF ENERGY TECHNOLOGY)

CONTENTS

9.1 SUMMARY OF RESULTS FROM THE MODEL

With the Summer School drawing to a close, the time has come to summarise the results from our world oil depletion model and discuss briefly the implications for the future.

Your calculations have produced a disturbing result : for each of the oil price assumptions investigated (M = 0 to M = 3), the model gives a projection for the annual rate of oil consumption, $\frac{dx}{dt}$, which reaches a peak before the end of this century. If this turns out to be true, less than twenty years are available for the world to prepare for the increasing gap that will appear between the desired consumption rate and the declining oil production rate. Massive investments will be required to provide new energy sources, conversion technologies and end-use devices.

Another projection provided by the model is that 90 per cent of the oil resource will be consumed before the year 2040 under all the price assumptions considered. The optimistic assumption that the oil resource is 50 per cent larger than commonly estimated (*i.e.* 3000 instead of 2000 gigabarrels) extends the time taken to reach 90 per cent exhaustion by only 13 years. Similarly, the elapsed time to the year of peak annual consumption is extended by only eight years.

9.2 LIMITATIONS OF THE MODEL

A surprising characteristic of the model is its insensitivity in terms of overall behaviour to oil price. We should perhaps be a bit suspicious of this and examine more closely our assumptions about the price elasticity-of-consumption coefficient, β. Is it too small? The values quoted in the literature for the coefficient are commonly in the range 0.0 to 0.4 (National Energy Advisory Committee, 1980). If there had been more time, we might have carried out a sensitivity study by repeating the calculations with selected values of β from within the above range. Perhaps we were wrong in assuming β to be a constant. What if β increased when the oil price went above some threshold value?

In the first lecture, I noted that, reluctantly, we had neglected income effects on oil consumption. We did this because, with only one week available, it was necessary to restrict the complexity of the model.

It is known, however, that the oil consumption rate is linked to measures of national economic activity such as the gross domestic product (GDP). After the oil price jump of 1973-74, the annual growth rate of the combined GDPs of the nations of the world fell from the boom value

of seven per cent recorded in 1973 to below one per cent in 1975. World oil consumption fell by over two per cent in the same period.

For the next five years consumption rose steadily again. Then, following upon the dramatic oil price rise of 1979-80, came yet another deep world economic recession. World oil consumption fell 4 per cent in 1980 and a further 3 per cent in 1981.

After both price jumps oil use slumped, partly because of a conservation response to the price rise (an effect that we modelled) and partly because of a fall in the level of economic activity that gave rise to the oil demand (an effect that we neglected).

To include the income response, we would need to define a measure of economic activity for the world such as the sum of all national GDPs. Writing this measure as $y(t)$ and using the notation given in Chapter 1, we could restructure the model into the form

$$\frac{dx}{dt} = \lambda x_o \left(1 - \frac{x(t)}{x_\infty}\right) \left(\frac{P_o}{p(t)}\right)^\beta \left(\frac{y(t)}{y_o}\right)^\alpha , \qquad (9.1)$$

where $y_o = y(t_o)$ and α is an income elasticity-of-consumption.

If the world economy were to grow exponentially with

$$y(t) = y_o e^{\gamma(t-t_o)}$$

such that $\alpha\gamma = \lambda$, then equation 9.1 would revert to our Summer School model (equation 1.18).

Some careful statistical work would be required to separate price and income effects when the values of α and β were determined. Values would be chosen to give the best statistical 'fit' of the model's output to the historical values of $x(t)$. Statistical work of this kind is known as 'econometrics', an active area of economic research with a decidedly mathematical flavour.

With a model based on equation 9.1 it would be possible, for example, to explore the effects on oil consumption of either a global economic slump or sluggish economic growth following another dramatic jump in oil price in 1990 (the M = 3 case). It would be found that the time required for 90 per cent exhaustion would be pushed back by another decade or so. The extra time would be bought at great human cost as measured in terms of lives affected by unemployment, poverty and hunger.

Another limitation of our model is its global viewpoint. Greater insight is obtained with larger models that break the world up into oil exporting regions trading with oil importing regions. The total oil resource, x_∞, is split into components which are allocated to the

geographic areas. The model projects future levels of production and consumption in each region, and the volume of oil trade.

If a little time is spent considering the geographical and political aspects of oil supply and demand, it will be easy to imagine events that could lead to a full scale oil crisis well before x(t) approaches the 50 per cent depletion point. For example, under normal circumstances between six and seven gigabarrels of oil a year carried in tankers through the narrow Straits of Hormuz at the mouth of the Persian Gulf into the oceans of the world. Armed conflict between nations in the Gulf region could choke that vital flow.

I have dwelt on the limitations of our model to encourage some healthy scepticism. There is a rather dangerous tendency on the part of many people to treat trend projections from computer models as predictions blessed with an aura of truth. Of course, the real situation is that computer models can fail to predict accurate trends as easily any seat-of-the-pants forecaster. However, the advantages of computer modelling over instinctive guesswork (mental modelling) are that

- the assumptions underlying the model are explicitly presented and are subject to scrutiny by others (not easily done with mental models);
- more variables, relationships and data can be taken into account than in mental models; and
- the model can readily repeat the calculations in a short time to test the sensitivity of the results to various data changes.

9.3 SOLUTIONS TO THE OIL PROBLEM

Having tested your confidence in the results from the Summer School model, let me hasten to reassure you that larger models incorporating all my suggested improvements deliver essentially the same message as our simple construction, *i.e.* the world must radically alter its energy system to permit substantial substitution away from natural crude oil before the year 2000. By that year, or before if political constraints increase, conventional crude oil production will probably have passed its highest peak. If the steps taken to reduce dependence on oil prove to be inadequate, the world economic system will arrive at its own 'solution' to equation 9.1. The 'solution' will be to push the price, p(t), sufficiently high and economic output, y(t), sufficiently low to bring the oil consumption rate, $\frac{dx}{dt}$, into line with diminishing production.

Figures 1.1 and 1.2 make it abundantly clear that, for at least the next 100 years, the problem is not the availability of energy resources, but rather the lack of time available to switch from oil to other energy forms.

Yet considerable time will be required to phase out the vast system of refineries, oil storage and distribution centres, oil burning furnaces and boilers, petrol- or diesel-fuelled cars, trucks, bulldozers, tractors, trains and ships and jet-fuelled aircraft. Little wonder that a major thrust of energy research and development is directed into alternative liquid fuel technologies that would permit much of the existing oil system to be retained.

9.4 ALTERNATIVE LIQUID FUELS

To produce alternative liquid fuels we can draw potentially upon such technologies as the following;

(i) Mining sources of oil that were not included in our total resource quantity, x_∞. These are the tar sands, heavy viscous crude oils and solid oil shales. They are often expensive to mine and pretreat to a form suitable as refinery feedstock. There are usually significant environmental effects associated with their extraction.

(ii) The conversion of natural gas to the liquid fuel methanol which can be used directly in specially adapted vehicles or further transformed into petrol. This option is one solution to the immediate problem, but it reduces the natural gas resource which is no bigger than the oil resource.

(iii) The conversion of biomass such as sugar cane, wheat or cassava into ethanol or the conversion of tree cellulose into methanol. Oil bearing seeds and plants with oil-like saps may also make a contribution. For these methods to make a significant impact, large areas of land would have to be allocated and the energy required to harvest and process the crops would have to be less than the energy produced. The attractive aspect of biomass conversion is that it is a renewable energy resource.

(iv) The conversion of coal to oil or methanol. Because the coal resource is large, this method offers great promise as a source of liquid fuels for the next century. An early version of the technology is already operating in South Africa. However, advanced versions are only at the research and development or pilot plant stage. This means that more than ten years may elapse

before the first advanced commercial plants begin operating and nearly twenty years before large numbers of plants and associated coal mines are in place. The time sequence being followed by coal-to-oil technology for initial research and development, pilot plant construction and assessment, first commercial plant assessment and large scale implementation must be traced by any novel capital intensive energy system.

Unfortunately, the central result from our oil model is that we have no more than twenty years to establish the replacement technologies. You see now why the world should be engaged in a frantic race to accelerate the progress of the new energy systems. The urgency of the problem is only now being recognised in the political sphere. In many ways the five valuable years after the first oil price shock of 1973-74 have been under-utilised.

Assuming that high economic growth rates resume and that no major political constraints on oil supply emerge, the International Institute for Applied Systems Analysis (IIASA) has recently projected liquid fuel supply for the world, excluding Russia, China and Eastern Europe (Sassin, 1980). It is likely that the actual supply will fall below the IIASA high growth projection shown in Figure 9.1.

9.5 SUBSTITUTES FOR LIQUID FUELS

Transportation systems based on liquid-fuelled vehicles are basic to our way of life; in fact, they have determined much of the present structure of industrialised society. For example, consider the urban sprawl of Sydney where people often live over 20 kilometres from their jobs. This is largely a consequence of the availability of the private petrol- or diesel-fuelled car. Commitments by society on this scale are not easily changed and, therefore, there will be great pressure to reserve the remaining oil for efficient light weight transport vehicles.

It will be necessary to provide advanced cracking and re-forming facilities at oil refineries to extract the maximum amount of light and middle oil fractions such as aviation turbine fuel, petrol and automotive diesel oil from crude oil. Production of heavy oil fractions will be minimised. These heavy fractions are presently burnt in furnaces for process heat and in boilers to generate steam for electricity production or process heat. Natural gas, coal, electricity and solar energy can be used to phase oil out of the industrial process heat and residential/commercial

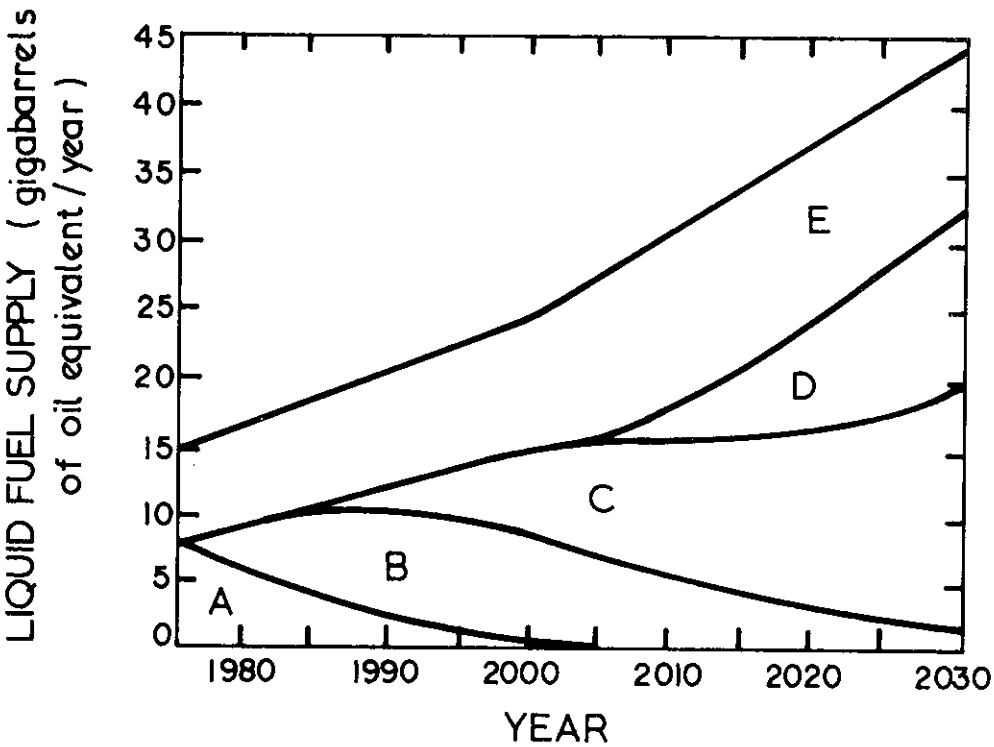


Figure 9.1 Liquid fuel supply projected according to the IIASA high-growth scenario for the world, excluding the centrally planned economies. The fuel sources represented include known reserves of non Middle East natural oil (A), new reserves of non Middle East natural oil (B), unconventional forms of oil such as tar sands, oil shales, heavy crudes and other products of enhanced-recovery techniques (C), synthetic fuels made by the liquefaction of coal (D), and natural oil produced in the Middle East and northern Africa predominantly by OPEC producers (E). [After Sassin 1980].

space heat sectors. I should remind you that natural gas suffers from the same resource limits as oil and eventually the full burden will fall on coal, electricity and solar energy in these sectors.

Coal and nuclear fission power can be used now to substitute for oil and natural gas in the electricity generating sector. Solar and nuclear fusion energy will probably be harnessed eventually to produce large quantities of electricity but not in time to solve the immediate oil problem.

Heavy demands will be made on coal, despite its environmental problems, to feed industrial boilers, furnaces and coal-to-oil/gas plants. The added requirement that coal should provide the main energy source for new electricity generating stations promises to place severe strains on the coal resource and its mining and transportation system. It was anticipated by many analysts, in the 1960s and early 1970s, that nuclear power would relieve some of the pressure on coal by providing the bulk of new electricity generating capacity. However, the recent combination of lowered rates of growth in electricity consumption, increasingly polarised public opinion and political hesitancy has placed a question mark over the future role that nuclear power will play in the solution to the oil problem.

In the IIASA study referred to above, it was found that significant additions to nuclear capacity were required in both the high-growth and low-growth economic development scenarios considered. Sassin (op.cit.) writes:

> "The projected figures of the high-growth scenario show that over the 50-year span of the IIASA study natural gas will maintain its present share of approximately 20 per cent of the world energy market but that oil will gradually decline, from 40 per cent in 1980 to 20 per cent in 2030. To make up for this shortfall and, what is equally important, to meet the demand for liquid secondary-energy forms an increasing amount of coal will have to be converted into synthetic fuels. The diversion of this much coal from electric-power generation will in turn have to be partly compensated for by the further penetration of nuclear power into the market for generating electricity. Because of anticipated resource limits on the supply of natural uranium, breeder reactors will assume an ever increasing share of the world energy-supply market from the year 2000 on.

Renewable energy sources, hydroelectric power and geothermal power will add up to a fairly constant share of somewhat less than 10 per cent of the total energy supply, an estimate that implies a substantial increase in the absolute power-generation levels for all these comparatively minor supply categories".

From the base year of the IIASA study (1975) to 2030, the total annual world primary-energy consumption rate is projected to rise from 43 gigabarrels oil equivalent (Gbbl. o.e.) to 189 Gbbl. o.e. in the high growth scenario (Figure 9.2) and to 116 Gbbl. o.e. in the low-growth one (Figure 9.3)

The IIASA projections are based on the conservative assumption that the world's population will double from 4 to 8 billion within the next 50 years and will eventually approach a stable level of 10 billion. A central tenet of the IIASA model is that the rates of economic growth in the under-developed world will exceed those of the developed world. This holds out the hope of a more equal distribution of wealth between nations. It is a vision sustained in the minds of conventional energy analysts by the assumption that an adequate energy supply system with high technology components will be available. However, some of the components, particularly nuclear power, are currently under attack.

9.6 ENERGY AND LIFESTYLES

Technologists who are familiar with the excellent safety record of nuclear power, and who are convinced that the problems of waste disposal and nuclear fuel cycle safeguards can be solved successfully, are often bewildered by the degree of hostility shown by groups in opposition to nuclear technology and their complementary enthusiasm for small decentralised energy systems. Douglas (1980) writes:

"One sign of this lack of comprehension is seen in the labels usually applied to these opposition groups by industry spokesmen and the press. 'Radical environmentalists' is one favourite, although environmental concerns have now frequently been merged with broader economic issues. 'Nuclear opponents' is another common label, although the groups often seek to block construction of other large energy facilities as well. A better term would be 'energy agrarians' - for the vision shared by activists in this loose movement involves not only a decentralisation of energy sources but a return to simpler technologies and life styles in general".

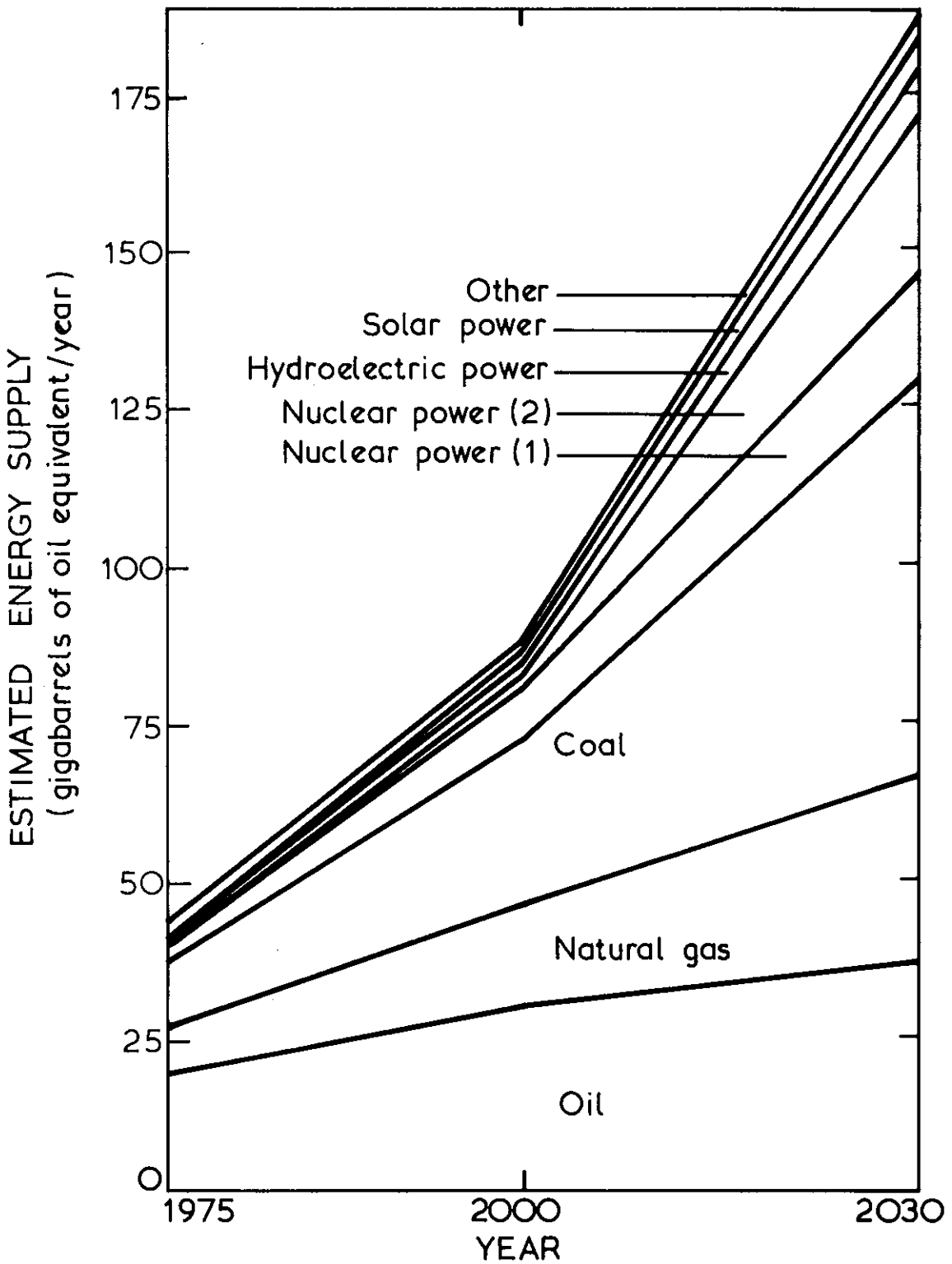


Figure 9.2 Global consumption rate of a variety of primary energy forms according to the high-growth IIASA scenario. Nuclear power sources are divided in this projection and that shown in figure (9.3) into conventional fission reactors (Nuclear 1) and advanced breeder-type fission reactors (Nuclear 2). The category 'other' includes biomass conversion. [After Sassin 1980].

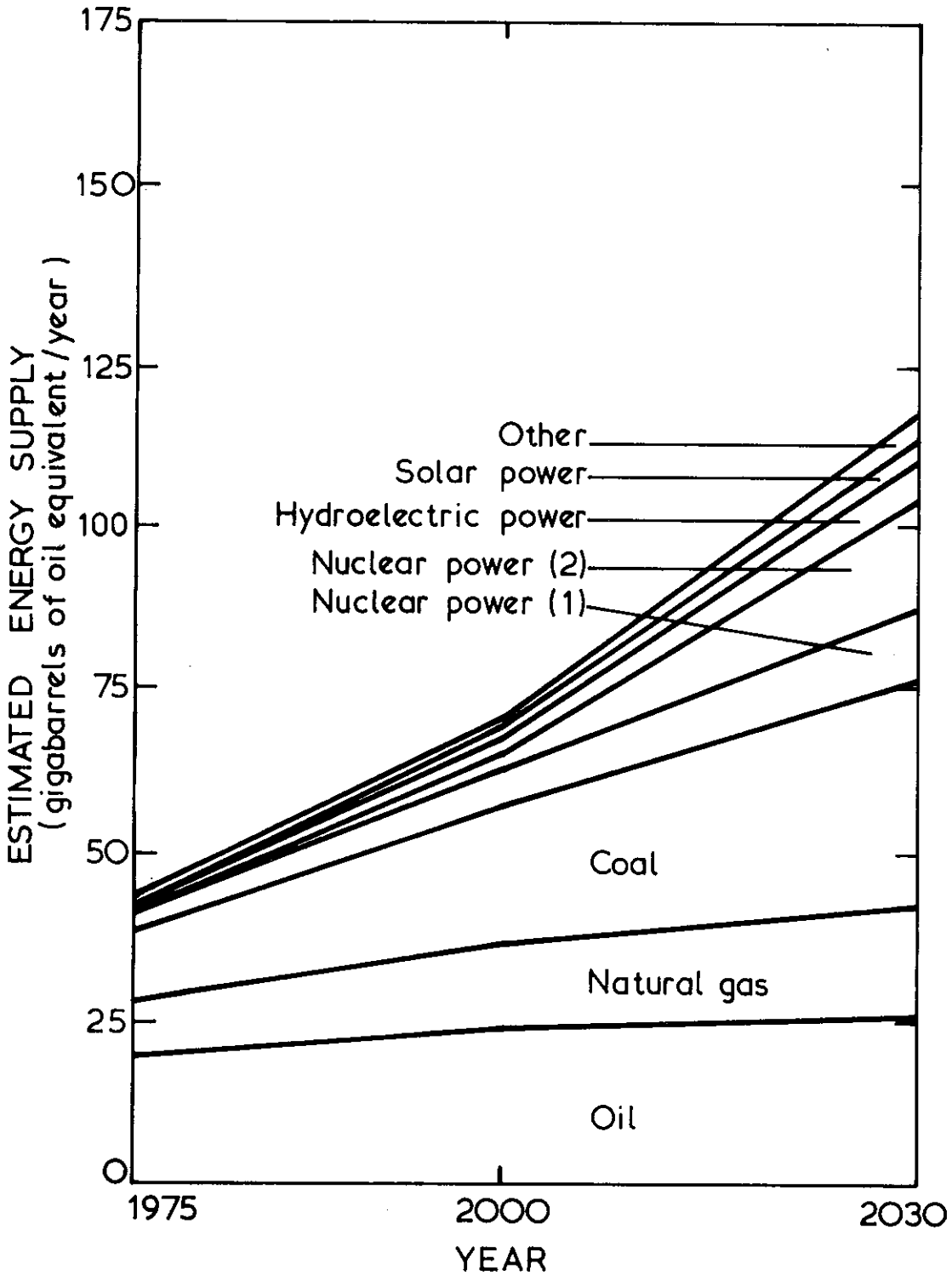


Figure 9.3 Global consumption rate of a variety of primary energy forms according to the low-growth IIASA scenario. [From Sassin 1980].

Whatever our vision of the future, be it the high technology solution of advanced centralised coal, nuclear, solar and fusion systems, the 'small is beautiful' solution of decentralised systems largely based on direct solar and biomass energy sources, or some middle path with all technologies playing complementary roles, we must ensure that the transition is feasible. Advocates need to convince us that their respective visions can be achieved without economic collapse and populations reduced by starvation or by the destruction of much that is beautiful, clean and wild in our natural environment.

9.7 FINAL THOUGHTS

Perhaps it is idealistic to say this, but I would wish that the multiple decisions on what is to be an acceptable energy system will be taken democratically following reasoned debate that considers all the important economic, sociological, technical and environmental aspects of the problem. Energy models can provide planners and policy makers with valuable insights as the debate proceeds.

I speak for all the Summer School staff in expressing our hope that the School has added to your awareness of the beauty and power of mathematics and the humanitarian usefulness of computers. We wanted to show you how these tools can be applied to gain insight into one of the most perplexing and dangerous problems facing our world.

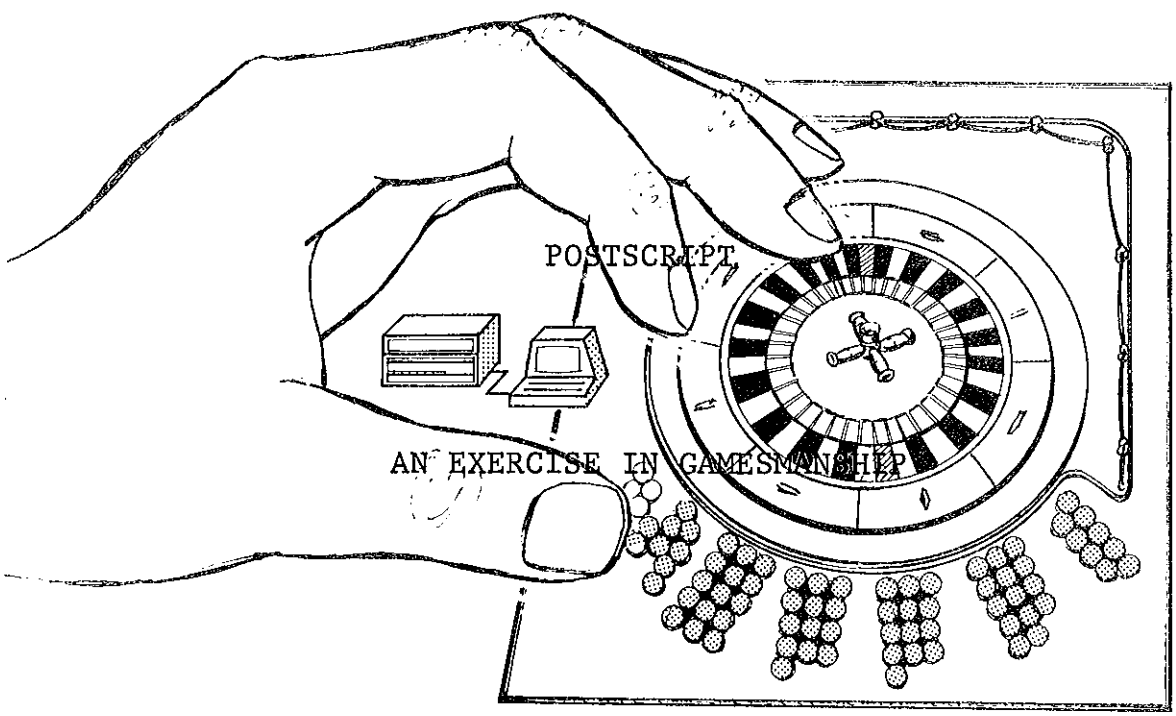
We would like to think that this experience will help you to decide whether you wish to enter a profession that uses mathematics and computers. It may even stir your interest in the professional areas associated with energy supply, conversion and end-use. If you become involved with an energy industry, you will encounter the breaking storm of the oil problem in all its fury. And, while the coming turbulence seems almost inescapable, there will be a great deal of challenge and excitement in keeping afloat and pointed in the right direction.

9.8 REFERENCES

Douglas, J.H. (1980) - Hidden agendas in the energy debate. New Scientist, 87:208.

National Energy Advisory Committee (1980) - Liquid fuels: longer term needs, prospects and issues. Report No.9, Australian Government Publishing Service, Canberra.

Sassin, W. (1980) - Energy. Scientific American, Vol. 243, No.3:107.



POSTSCRIPT

AN EXERCISE IN GAMESMANSHIP

POSTSCRIPT

An Exercise in Gamesmanship

During the Summer School week, at different times to be indicated, you will be given the opportunity to try your hand at gamesmanship. You are expected to finish the game shown half completed on the next few pages. (The underlined items indicates user replies as distinct from computer prompts.) On the last day, you may have time to try other games (these would be full games of 10 moves). Towards the end of the Summer School, the computer will search for

(1) the best results for the first game, and

(2) the best results for all games.

All the best as you play TWOIL.

JMB & JPP

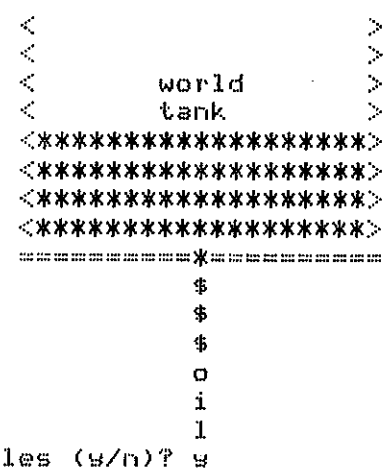
Here is a GAME for the Summer School

> T W O I L <

For the first game you begin where

sskJPP left off (see over)

```
ID: ********          (actually ID: AM290060)
$2

login: ssk

Please remove (or pack) as many files as possible.
Disk space is easily consumed.

Dataway users please do not use the o command in the editor.

% basic


Ready

run twoil
```




```
 do you want the rules (y/n)? y

The aim of the game 'twoil' is for you to make
the biggest possible profit from the sale
of the world's oil compared with other players
over a 10 year period 1990 to 1999 .
Accumulated profit is assumed invested at 10% per annum.
Lady luck only plays a small part in the game compared with skill.

Push RETURN to continue?

A move consists in a player
(1) identifying him or her self with the appropriate
summer school id, eg sskjpp (but yours not mine) - or an alias:

(2) selecting 2 nos. <r,e>- the % of income for
the year being considered (t), to be directed into
<r>- research & development (r&d), and
<e>- exploration.
Note that initially 30 % of your income is directed into
operating and marketing costs for your product
but investment in r&d -<r>- reduces this.
The more money directed into exploration -<e>-
the greater the 'world oil tank' appears
and ultimately a bigger yearly profit - but is it worth it?

Push RETURN to continue?
```

```
A few aspects associated with running 'twoil' follow ...

(1) Your choice of alias name must be unique cf. others at your terminal
    and should be brief (no more than 5 characters).
(2) If your alias begins with > then your output will be abbreviated.
(3) You may change your alias by using ssk(initials) for a go.
(4) If you are the only person at the terminal then immediate RETURN will do.
(5) For quick exit from 'twoil' reply stop instead of quit
    as quit is for 1 player.

Push RETURN to continue?

your ss id, or alias if defined, or quit please? sskjpp
enter an alias please? John
-------------------------------------------------------------------------------
sskjpp(John) game 0     : 1989 :
                        ::::::::              note g=1.e9
(  known oil   )      <  total oil  >       !accum. profit!
(  reserves    )      < production  >       !              !
(              )      <             >       !              !
( 740  gbbls   )      < 660  gbbls  >       !   g$ 0       !
(              )      <             >       !              !
(**************)      <  *********  >       !              !
(**************)      <*************>       !              !
(**************)      <*************>       !              !
================#=====================#=====================#========
                          * price/10bbls= 514 $ totals
          operation  ==-=-> %= 0  efficiency%  g$= 0
                r&d  ==-=-> %= 0       * 0     g$= 0
        exploration  ==-=-> %= 0       * 0     g$= 0
*** a new game ***
do you want this (y/n)? y
sskjpp(John) what % of income to <r,e> (0<=r+e<= 70 )? 5,5
-------------------------------------------------------------------------------
sskjpp(John) game 1     : 1990 :              new game
                        ::::::::
(  known oil   )      <  total oil  >       !accum. profit!
(  reserves    )      < production  >       !              !
(              )      <             >       !              !
( 796  gbbls   )      < 682  gbbls  >       !   g$ 696     !
(     ***      )      <             >       !              !
(**************)      <  *********  >       !              !
(**************)      <*************>       !              !
(**************)      <*************>       !      $$$     !
================#=====================#=====================#========
                          * price/10bbls= 531 $ totals
          operation  ==-=-> %= 30 efficiency%  g$= 348
                r&d  ==-=-> %= 5       * 79    g$= 58
        exploration  ==-=-> %= 5       * 57    g$= 58
your ss id, or alias if defined, or quit please? John
sskjpp(John) what % of income to <r,e> (0<=r+e<= 70 )? 5,5
```

```
sskjpp(john) game 1      : 1991 :
                         ::::::::
(  known oil   )      <  total oil  >      !accum. profit!
(  reserves    )      < production  >      !              !
(              )      <             >      !              !
( 848  sbbls   )      < 705  sbbls  >      !   s$ 1531    !
(    *****     )      <             >      !              !
(************)       < ********** >       !              !
(************)       <************>       !              !
(************)       <************>       !    $$$$$     !
=========#============================#=================#
                          * price/10bbls= 550 $ totals
            operation  =-=-> %= 29 efficiency%  s$= 712
                  r&d  =-=-> %= 5       * 82      s$= 121
          exploration  =-=-> %= 5       * 59      s$= 121
your ss id, or alias if defined, or quit please? john
sskjpp(john) what % of income to <r,e> (0<=r+e<= 70 )? 5,5
```

```
sskjpp(john) game 1      : 1992 :
                         ::::::::
(  known oil   )      <  total oil  >      !accum. profit!
(  reserves    )      < production  >      !              !
(              )      <             >      !              !
( 894  sbbls   )      < 728  sbbls  >      !   s$ 2521    !
(   *******    )      <             >      !              !
(************)       <************>       !              !
(************)       <************>       !              !
(************)       <************>       !   $$$$$$$    !
=========#============================#=================#
                          * price/10bbls= 569 $ totals
            operation  =-=-> %= 28 efficiency%  s$= 1091
                  r&d  =-=-> %= 5       * 81      s$= 188
          exploration  =-=-> %= 5       * 60      s$= 188
your ss id, or alias if defined, or quit please? john
sskjpp(john) what % of income to <r,e> (0<=r+e<= 70 )? 5,5
```

```
sskjpp(john) game 1      : 1993 :
                         ::::::::
(  known oil   )      <  total oil  >      !accum. profit!
(  reserves    )      < production  >      !              !
(              )      <             >      !              !
( 936  sbbls   )      < 753  sbbls  >      !   s$ 3689    !
( ********** )       <             >      !              !
(************)       <************>       !              !
(************)       <************>       !              !
(************)       <************>       ! $$$$$$$$$$   !
=========#============================#=================#
                          * price/10bbls= 589 $ totals
            operation  =-=-> %= 27 efficiency%  s$= 1481
                  r&d  =-=-> %= 5       * 83      s$= 261
          exploration  =-=-> %= 5       * 65      s$= 261
your ss id, or alias if defined, or quit please? john
sskjpp(john) what % of income to <r,e> (0<=r+e<= 70 )? 5,5
```

```
sskjpp(john) game 1      : 1994 :
                         ;;;;;;;;
(  known oil   )      <  total oil  >      !accum. profit!
(  reserves    )      < production  >      !              !
(              )      <             >      !              !
( 969  gbbls   )      < 778  gbbls  >      !   g$ 5055    !
( ************ )      <      *      >      !              !
(**************)      <**************>     !              !
(**************)      <**************>     !     $$$      !
(**************)      <**************>     !$$$$$$$$$$$$$!
=======#=================#=================#=========
                          * price/10bbls= 610 $ totals
            operation  ==-=> %= 26 efficiency%  g$= 1878
                  r&d  ==-=> %= 5       * 84     g$= 338
          exploration  ==-=> %= 5       * 64     g$= 338
your ss id, or alias if defined, or quit please? quit
quit (stop, suspend or retire) required for ...
your ss id, or alias if defined, or quit please? john
sskjpp(john)
enter 'r' to resign, 's' to suspend or 'stop' to stop all? stop
*-stop-* for sskjpp(john)- done

*>>> all jobs stopped <<<*

Ready

bye
%   (actually CTRLd)
connect  time      8:10.00
user cpu time        18.52
sys  cpu time        12.32

$   (actually CTRLp)
END-SESSION
```

WORKSHEETS

Mainly to be filled in from results
on the main computer.

NOTE These worksheets are to be filled in and shown to your tutor during the Friday session. The tutor will check your results and advise you to move on from one worksheet to the next or will suggest that you check your program, modify it and have another attempt at the same worksheet.

FIND You will find a summary of the model and data in Appendix 1B.

GO Some ideas for setting up a program for the main computer are given in Section 4.5 and Appendix 6C.

TEST RUNS You should first test your program - worksheets 1 and 2.

PRODUCTION RUNS After your tutor has approved your results reported on worksheets 1 and 2, carry out the production runs. Do one run before you undertake the next in case the program is not entirely free of bugs.

FINALLY What does it all mean? A further sheet will be issued. You should enter your observations on the sheet which should be handed in.

Some additional exercises are given as worksheet 6.

NOTES

WORKSHEET NO. 1

CASE $M = 0$, $x_\infty = 2000$.

WHY This run is a program test. Your computed results should follow that of the Hubbert model (Section 4.3):

$$x(t) = \frac{x_o\, x_\infty\, E}{x_\infty - x_o + x_o\, E} \quad ,$$

$$E = e^{\lambda(t - t_o)} \quad ,$$

and $$DX(t) = x(t) - x(t - 1) \quad .$$

RESULTS In case you don't have a calculator available a few results are given.

t	DX	Your DX	x	Your x	p_c	Your p_c
1910	0.310		4.600		6.11	
1911	0.333		4.933		"	
1982	26.656		525.212		"	
1983	27.557		552.769		"	
1984	28.433		581.201		"	
1999	34.865		1078.603		"	
2000	34.674		1113.278		"	
2001	34.400		1147.678		"	
2049	3.549		1949.705		"	
2050	3.320		1953.026		"	

NOW Show your results to your tutor.

NOTES

WORKSHEET NO. 2

CASE M = 1, x_∞ = 2000 Gbbls.

WHY This run is a further program test - it will test your use of the price function. Your computed results should follow the model given in Appendix 1A and Appendix 4A.

$$x(t) = \frac{x_o \, x_\infty \, E}{x_\infty - x_o + x_o \, E} \quad ,$$

$$E = \{1 + b \, (t - t_o)\}^{\lambda/b} \quad ,$$

and $DX(t) = x(t) - x(t - 1)$.

NOTE The price function, PRICE, returns an average value for the year (actually the value appropriate to the middle of the year, $t - \frac{1}{2}$) which you should bear in mind in any comparison with the table below.

RESULTS In case you don't have a calculator available a few results are given.

t	DX	Your DX	x	Your x	$p_c(t-\frac{1}{2})$	Your p_c
1910	0.310		4.600		6.11	
1911	0.332		4.932		6.35	
1982	20.044		415.618		34.60	
1983	20.750		436.368		35.39	
1984	21.452		457.820		36.19	
1999	29.386		851.661		50.49	
2000	29.604		881.265		51.61	
2001	29.771		911.036		52.75	
2049	7.346		1862.715		145.24	
2050	6.989		1869.704		148.22	

NOW Show your results to your tutor.

NOTES

WORKSHEET NO. 3

CASE M = 2, x_∞ = 2000 Gbbls.

WHY This is the first 'production' run. The model is the most realistic of those to run at the Summer School.

NOTE The main results we want to determine from this (and subsequent runs) are indicated in Section 4.5, namely:

(a) In which year will the oil consumption reach a maximum and at what level?

(b) In which year will cumulative consumption reach 90 per cent of the total resource x_∞?

RESULTS Enter your results in the table.

t	DX	x	P_C
1982			
1983			
1984			
some values around the peak in DX			
some values around $x = 0.9\ x_\infty$			

MAIN RESULTS (a) Peak year is _____ and oil consumption is _____ Gbbls.

(b) Year when oil 90% used up is _____ .

NOW Show your results to your tutor.

NOTES

WORKSHEET NO. 4

CASE M = 3, x_∞ = 2000 Gbbls.

WHY The pricing model used here is likely to be very extreme with a rise by a factor of 3 in oil price in 1990. The model gives us a feel of the sensitivity of our results to oil price.

RESULTS Enter your results in the table.

t	DX	x	P_C
1982			
1983			
1984			
some values around the peak in DX			
some values around $x = 0.9\ x_\infty$			

MAIN RESULTS (a) Peak year is _____ and oil consumption is _____ Gbbls.

(b) Year when oil 90% used up is _____ .

NOW Show your results to your tutor.

NOTES

WORKSHEET NO. 5

CASE M = 2, x_∞ = 3000 Gbbls.

WHY As a further step in our sensitivity analysis we will consider the influence on our results of changing one other basic parameter of the study, namely, the total resource available, x_∞. Surely this change from 2000 to 3000 will make a substantial change in our results.

RESULTS Enter your results in the table.

t	DX	x	p_c
1982			
1983			
1984			

some values around the peak in DX

some values around $x = 0.9\ x_\infty$

MAIN RESULTS (a) Peak year is _____ and oil consumption is _____ Gbbls.

(b) Year when oil 90% used up is _____ .

NOW Show your results to your tutor. If your tutor is satisfied with your results ask for the special handout question 'What does it all mean'?

NOTES

WORKSHEET NO. 6

SOME FURTHER SIMULATION PROBLEMS FOR THOSE FINISHED THE EARLIER WORKSHEETS

You don't have to do these in the order given - take your pick

(1) Queuing theory, or the theory of stochastic service systems, is an important area of applied probability theory. There is a vast amount of literature on the subject. The basic principles are often treated in elementary textbooks on probability. Here is a simple example.

If you haven't done so already, finish the Pascal program to simulate Bill Smith's job expectancy according to the details of the Pascal notes chapter 6, Q10. A flow chart is available later in the chapter to help you write the program.

(2) The writing of computer games is popular on microcomputers. Certainly games can be a useful way to learn about some aspects of computer simulation.

Read the BASIC notes (Chapter 7) then write a BASIC program to play the game MISTER MIND according to the details of the BASIC notes (p. 7.13). If you know the BASIC language, then refer to sections 7.2 and 7.15 for logon and logoff details.

(3) A queuing theory example is given as Q14 in chapter 6 - try this.

(4) Another example is given below. (This is somewhat easier than Q14.)

A stream of customers arrives at a service facility as follows. The customers arrive at the time epochs 1, 2, 3, 4, ..., and for each such time epoch the probability is p that a customer will arrive. At the service facility there is one attendant who immediately starts to serve an arriving customer if he is not busy with already arrived customers. The service has the following structure: a service that is going on at a time epoch is ended during the next time interval with probability p_1. Customers arriving when the attendant is busy wait for service in a queue.

Write a Pascal program that simulates such a service system. The system is supposed to be empty of customers at time epoch 0. The program should indicate the average time it takes for a customer to be served, the maximum time and the number of customers served as well as the number remaining in the queue after following epochs 10,20,30,...

Try running the program for 500 time epochs with

(a) $p=0.4$, $p_1 = 0.5$

and (b) p=0.5, $p_1 = 0.4$

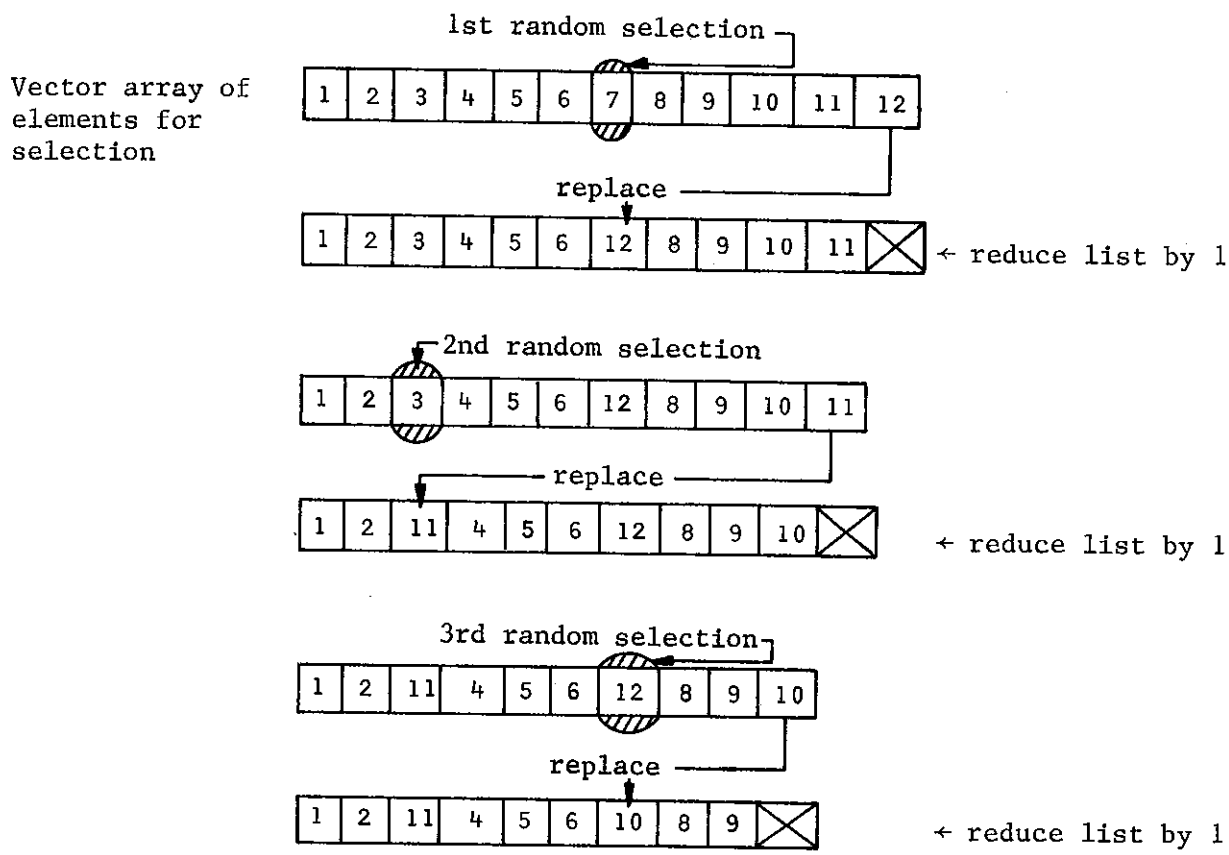
(5) One great potential of the microcomputer is in the area of computer assisted instruction (CAI).

Write a BASIC program (on the mini and perhaps on a micro) to teach a youngster the n's time table - 1xn, 2xn,...,12xn where n(1,2,..., 12) is selected by the youngster. Two modes should be available ...

(a) the computer should run through the n's time table (say with 3 seconds pause between printing each line of the table), or

(b) a table element should be selected by the computer at random with three attempts permitted for the youngster to give his answer before the computer supplies the correct result.

Exhaustive selection should be used for mode (b) - that is, if 7 has already been used once then it should not be used again in a round of 12 goes. Here's a clue of what to do ...



Got the idea?

Note: Vectors are dimensioned in BASIC thus

```
130  DIM  N1(12)
```

although they are automatically dimensioned if the biggest index is 10 or less. A zero index is also permitted although not required in this example.

(6) Don't forget to play some moves in the TWOIL game presented in the postscript given earlier.