



# MERCURY AUTOCODE MANUAL

---

**Second Edition**

by

**R. A. Brooker**

**B. Richards**

**E. Berg**

**R. H. Kerr**

*of*

*The Computing Machine Laboratory*

*The University, Manchester.*

## PREFACE TO THE SECOND EDITION

The present volume is a revised edition of the Manchester Mercury Autocode System of programming, which was previously printed in May 1959. In this revised edition the material has been entirely re-arranged, and certain additional information has been included. Some of this new material describes facilities added to the system after the previous edition was prepared, and for some of the descriptions, and for the facilities, described in Appendix 4, we are grateful to the Mercury Group at Imperial Chemical Industries Ltd., Wilton Works.

As in the previous version we have included the first ten Library Programmes to illustrate the general specification of such programmes. Two of these are now obsolete, namely 508 and 509, and have been replaced by 516 and 524, prepared by Dr. D. Morris (Manchester University) and Mr. R. Maddison (Oxford University) respectively. Many other programmes have also been developed however, and include such operations as determining the latent roots and vectors of matrices, curve fitting by least squares, miscellaneous data-processing operations, etc., and prospective users are advised to consult the installation concerned as to the current state of the library.

Not every installation may have precisely the same facilities in their Autocode system, but this discrepancy is not likely to affect the ordinary user. Those described in the body of this report, i.e., Parts 1, 2, 3 and 4, are almost certainly common to every installation, whilst those described in the Appendices may differ from one installation to another.

One of the reasons for revising the description is that it is proposed to write Translators to accept Mercury Autocode programmes on the new ATLAS and ORION computers. There are, however, features of Mercury which it may be impracticable (and in some cases undesirable) to simulate in detail on the new machines. These include details of the representation of information on punched paper tape and in the machine, the detailed specifications of library programmes, running times, and manual operating. For example, the new machines use different ranges of numbers and precisions of calculation, and will generally yield greater accuracy in a shorter time.

Consequently, in preparing this edition, an attempt has been made to collect in appendices information about facilities for which there is no assurance of acceptance by Atlas or Orion. Even so, the body of the manual still contains details (such as the precision of numbers, the range of exponents, times of operation, space occupied by the programme and the preparation of tapes) that are either easily recognisable as specific to Mercury or explicitly associated with Mercury.

This Manual is a working handbook for the Mercury Autocode System. Those coming to a computer for the first time might find it useful to read first the "Introduction to the Mercury Autocode" (Ferranti List CS 241). The document "Autocode Examples" for Mercury (Ferranti List CS 270) may also be found useful.

Specifications of further Library programmes written in Autocode are available through the Mercury Library Service. Titles of Autocode programmes are contributed to the Mercury Programme Interchange scheme.

In other documents issued by Ferranti Ltd. it will be noticed that  $\psi$  has been used in place of  $\phi$  (for function) as used in this document. This is because  $\phi$  is used for another purpose (to signify figure shift) in most Ferranti computer literature. Some sets of tape-editing equipment show both symbols on the key, but the correction can easily be made mentally.

Those principally responsible for this very successful Autocode System are the authors of this Manual. They have very kindly made it available for publication by Ferranti Ltd., who gratefully acknowledge this fact.

## PREFACE TO THE FIRST EDITION

The present volume is a description of the Manchester Mercury Autocode system of programming, which was previously distributed in four parts, namely

1. Basic Facilities by R.A. Brooker.  
August, 1958.
2. Operational Facilities by R.A. Brooker and B. Richards.  
August, 1958.
3. Matrix and Vector Operations by E. Berg and R.A. Brooker.  
October, 1958.
4. A Basis for the Programme Library by R.A. Brooker and R.H. Kerr.  
January, 1959.

These are reproduced here with only minor alterations and corrections. No attempt has been made to re-cast the material in any way.

Several people have helped in this project. Dr. B. Richards has been of considerable assistance at all stages of the work, and Dr. A.R. Curtiss and Miss M. Biram of A.E.R.E., Harwell did some preliminary work on the matrix scheme. I am also extremely grateful to Dr. J. Howlett of A.E.R.E., Harwell and Dr. G.E. Thomas of I.C.I. Central Instrument Laboratory, who have helped to make the project successful in operation, by arranging appropriate educational facilities and computing services.

Thanks are also due to Miss B.M. Dent and Mr. B. Birtwistle, of Metropolitan-Vickers Electrical Company Ltd., who made available facilities for typing and reproducing the earlier edition. Much of this earlier material was drafted by Miss C.M. Popplewell who has also edited the present volume.

R. A. BROOKER.

## CONTENTS

	Page No.
Part 1      Basic Facilities	1
Part 2      Further Facilities	16
Part 3      Matrix Operations	29
Part 4      The Programme Library	33
Appendix 1      A service for the punching and execution of Autocode Programmes on the Manchester University Mercury Computer	41
Appendix 2      Notes for Programmers who wish to prepare their own tapes and/or run them on the machine personally.	48
Appendix 3      Interpretation of Machine Orders	52
Appendix 4      Facilities that are available only on the Manchester and I.C.I. machines	55
Appendix 5      A selection of library programmes available for Mercury	61



# Part I

## Basic Facilities

### General properties

An Autocode *programme* consists of an ordered sequence of *instructions* and other items of information. Each of these is written on a separate line and employs the following symbols:-

a b c d e f g h u v w x y z  $\pi$   
i j k l m n o p q r s t  
. 0 1 2 3 4 5 6 7 8 9  
+ -  $\neq$  = >  $\geq$  \* ( , )  $\approx$   $\rightarrow$  /  $\phi$  ' ?

The programme is ultimately presented to the machine in the form of a length of perforated paper tape which is scanned by a photoelectric tape reader, the input unit of the machine. The *programme tape* is prepared by means of a manual keyboard perforator on which are engraved the standard symbols. The material is *punched* in the conventional fashion, namely from left to right and down the column. Each instruction is followed by two special symbols *CR* (carriage return) and *LF* (line feed) which are provided for this purpose. There is also an *erase symbol* \* which is used for overpunching mistakes.

The instructions read from the tape are placed in the *instruction store* of the machine. the numerical quantities to which they refer are kept in the *number store* of the machine. The programme will include instructions to set the initial values of such quantities, either directly or by reading them from a further *data tape* by a process similar to that by which the instructions themselves were read into the machine.

The instructions fall into two classes, the *arithmetical* instructions which perform the calculation proper, and the *control* instructions which "organise" the calculation (e.g., arrange to repeat cycles of arithmetic, select alternative courses of action, or, as already mentioned, read further numerical data into the machine). This latter class of instructions are the characteristic features of *automatic* calculating machines which distinguish them from desk machines which are "controlled" by the operator himself. Both kinds of instruction need to refer to the working store so that it is appropriate to start by describing the notation used to refer to the numbers stored therein.

### The working quantities

The numbers recorded in the *working store* are also of two kinds, *general variables* and *indices*, which, like the two kinds of instructions, relate mainly to the calculation proper and its organisation. Thus the variables have numerical values in the range  $10^{-70} < x < 10^{70}$  and are recorded to a precision of eight/nine decimals, while the indices are restricted to integral values in the range  $-512 \leq i \leq 511$ .

### The variables

These are divided into three sets as follows:  
There are 480 *main variables* which can be divided into a maximum of 15 groups associated with the *variable letters*,

$$a \ b \ c \ d \ e \ f \ g \ h \ u \ v \ w \ x \ y \ z \ \pi$$

For example they can be arranged as a single group of 480 variables

$$v_0 \ v_1 \ v_2 \ \dots \ v_{479}$$

by writing the *directive*

$$v \rightarrow 479$$

Alternatively they could be arranged in three equal groups, thus

$$a_0 \ a_1 \ a_2 \ \dots \ a_{159}$$

$$b_0 \ b_1 \ b_2 \ \dots \ b_{159}$$

$$c_0 \ c_1 \ c_2 \ \dots \ c_{159}$$

the necessary directives being

$$a \rightarrow 159$$

$$b \rightarrow 159$$

$$c \rightarrow 159$$

It is intended that these groups shall reflect any natural grouping of the quantities occurring in the problem, and provided that the total number of variables does not exceed 480 the number and size of the groups is at the disposal of the programmer.

In addition to the main variables, there are fifteen *special variables* represented by the letters

$$a \ b \ c \ d \ e \ f \ g \ h \ u \ v \ w \ x \ y \ z \ \pi$$

employed without a suffix. These will be a common feature to every programme which cares to use them. The special variable  $\pi$  may be assumed to have the value 3.14159... until otherwise altered.

Finally there are 14 *primed special variables* denoted by

$$a' \ b' \ c' \ d' \ e' \ f' \ g' \ h' \ u' \ v' \ w' \ x' \ y' \ z'$$

(note: there is no  $\pi'$ ).

All the above quantities are kept in the working part of the number store. An *auxiliary store* normally provides up to 10,752 farther variables, but these are less easily accessible and will not be introduced at this stage, since the working store will be sufficient for many applications.

## Indices

Indices are represented by the 12 letters

i j k l m n o p q r s t

Although permitted integral values in the range  $-512 < i < 511$ , emphasis is placed on positive values because they are primarily intended to be combined with variables in the form

e.g.  $x_i$  or  $x_{(n-1)}$  or  $x_{(s+50)}$

to represent a free suffix; that is, these expressions may represent any one of the variables

$x_0, x_1, x_2, \dots$

depending on the particular value of the index in question. Thus if  $n = 4$ , then  $x_{(n-1)}$  will refer to  $x_3$ . The last two expressions illustrate the most general form which a suffix may take, namely, (index  $\pm$  integer). Whatever form it takes, however, the computed value of a suffix must lie within sensible limits. In calculations of a repetitive nature an index will assume a range of values, and to arrange this it is necessary to be able to compute with indices as separate items in much the same way as variables.

In preparing the input tape, all expressions are recorded in a *one-dimensional* form, thus  $x_3$ ,  $x_i$ , and  $x_{(s+50)}$  appear as  $x3$ ,  $xi$ , and  $x(s+50)$  respectively. Consequently it is not possible to distinguish, in a product, between (say)  $xi$  meaning  $x_i$  and  $xi$  meaning "x times i". In order to resolve this difficulty, a convention will be introduced later for ordering the factors in a product.

## Numerical Values

Explicit numerical values will have to be introduced into the programme at some stage, so that it is necessary to explain how these are written. The standard form is

integral part      decimal point      fractional part

omitting what is unnecessary. Thus, as is the case in writing suffices, the decimal point can be omitted in whole numbers. However, absolute standardisation of form is not necessary, so that e.g., "fifteen" may be written as

15      15.      15.0      015.0

All these and similar variations will be accepted by the machine.

Similarly  $\sqrt{2}$  to six significant figures may be written as:-

1.41421\*      or      -01.41421      or      001.4142100 etc.

and  $2/3$  to the same precision as:-

0.666667      or      .666667      or      000.666667000 etc.

Any number of digits are allowed in the integral part and up to 24 in the fractional part. (In all cases only the first 9 or 10 significant figures are relevant because inside the machine numbers are restricted to a precision of 29 binary digits.)

### The arithmetical instructions

The basic form of the instructions for computing variables may be illustrated by the following example:

$$y = 2mn a_{(m+1)} + a_m n + m a_n + 0.01m + 0.01n$$

which gives the new value of the variable to be altered (in this case  $y$ ) in terms of other quantities.

In general the right hand side may involve any number of products which may each have any number of factors either variables, indices, or constants. As already mentioned it is necessary to distinguish in the "one-dimensional" form

$$y = 2 mna(m+1) + amn + man + 0.01m + 0.01n$$

between  $am$  meaning  $a_m$  and  $a \times m$ . The convention adopted is that an index immediately following a variable letter is treated as a suffix so that the above expression is interpreted as

$$y = 2 \times m \times n \times a_{(m+1)} + (a_m \times n) + (m \times a_n) + (0.01 \times m) + (0.01 \times n)$$

As a consequence of these rules, numerical factors will usually be placed at the beginning of a product.

Further examples of instructions in this general class are:-

$$a = 0 \quad x_k = x_k + 1 \quad x_n = x_0 + nh \quad x = i$$

Products can also be divided by a single quantity

$$\text{thus} \quad u = x/a + y/b + z/c$$

$$\text{and} \quad v = 2 \pi u/n$$

are possible instructions.

It is recommended that  $\pi_1, \pi_2, \dots$ , be used as temporary working space, e.g. where a complex algebraic expression requires several instructions, the intermediate answers are denoted by  $\pi_1, \pi_2$ , etc.

In formulating the r.h.s. of an arithmetical expression it is recommended that single terms be written last, e.g.,

$$y = uvw + ab + f + g$$

as in this form the machine will take less time to evaluate the answer.

The basic form of the instructions for computing indices may be illustrated by the following example.

$$i = 2mn + m + n + 1$$

The only differences between this and the previous class of instruction are that the quantities on the right hand side are restricted to indices or whole numbers, and that the use of the solidus is not permitted. Further examples of instructions in this class are:-

$$i = 0 \quad n = n + 1 \quad r = 10 p + q$$

## Functions

The following instructions are a means of introducing certain elementary functions into the programme. Here the l.h.s.  $y$  stands for any variable, and the argument  $x$  for any r.h.s. expression.

$y = \phi \text{ sqrt}(x)$   
 $y = \phi \text{ sin}(x)$   
 $y = \phi \text{ cos}(x)$   
 $y = \phi \text{ tan}(x)$   
 $y = \phi \text{ exp}(x)$   
 $y = \phi \text{ log}(x)$  i.e., log to base  $e$   
 $y = \phi \text{ mod}(x)$  i.e., modulus of  
 $y = \phi \text{ int pt}(x)$  i.e., integral part of  
 $y = \phi \text{ fr pt}(x)$  i.e., fractional part of  
 $y = \phi \text{ sign}(x)$  i.e.,  $y = 1$  if  $x \geq 0$ ,  
 $y = -1$  if  $x < 0$ .

Examples of instructions in this class are

$u = \phi \text{ cos } (lx/a + my/b + nz/c)$   
 $a = \phi \text{ log } (xx + yy)$   
 $w = \phi \text{ sqrt } (xxx)$   
 $x_n = \phi \text{ cos } (nd)$

The following class of instructions involves functions of two variables, which may each be replaced by r.h.s. expressions.

$z = \phi \text{ divide } (x, y)$  i.e.,  $x/y$   
 $z = \phi \text{ arctan } (x, y)$  i.e.,  $\text{arctan } (y/x)$  ( $-\pi < z < \pi$ , the quadrant being determined as if  $x, y$  were proportional to  $\cos \phi, \sin \phi$  respectively)  
 $z = \phi \text{ radius } (x, y)$  i.e.,  $\sqrt{x^2 + y^2}$

Examples of instructions in this class are

$z = \phi \text{ divide } (x + y, x - y)$   
 $u = \phi \text{ arctan } (aa - bb, 2 ab)$   
 $a_1 = \phi \text{ radius } (x_1, y_1)$

Finally there are the instructions:-

$i = \phi \text{ int pt}(x)$  for converting a variable (or variable expression) to an index (the largest whole number less than or equal to  $x$ )  
 $y = \phi \text{ poly}(x) a_0, n$  for calculating  $y = a_0 + a_1 x + \dots + a_n x^n$ .

As before,  $y$  is any variable and  $x$  any variable expression. The parameter  $a_0$  is any first member of a group, and  $n$  is any index or whole number (but not an index expression)

$y = \phi \text{ parity}(n)$        $y = (-1)^n$ , where  $n$  is any index expression.

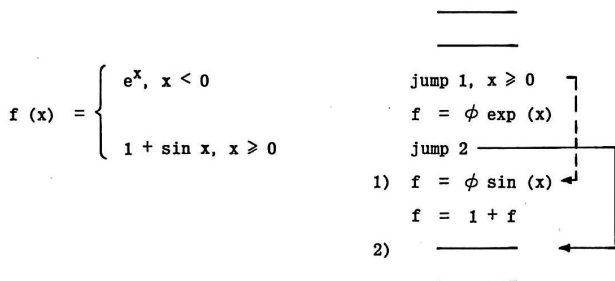
$i = \phi \max(x_0, n, m)$  } for determining the index of the maximum (or  
 $i = \phi \min(x_0, n, m)$  } (minimum) element of the set  $x_n, x_{n+1},$   
 .....  $x_m$

Here  $i$  denotes any index,  $x_0$  any first member, and  $n, m$  are indices or whole numbers. If there is no unique maximum (or minimum) element, then that with the least index value is taken.

### The control instructions

These are the instructions which organise the calculation and for this reason are sometimes known as the "red tape".

The most important instructions in this class are the *jump* instructions. Arithmetical instructions are normally obeyed in the order in which they are listed, but from time to time it is necessary to select alternative courses of action as in the following sequence of instructions for calculating:-



In this example the first instruction is a *conditional* jump, i.e., if the condition (in this case  $x \geq 0$ ) is satisfied then control "jumps" to the instruction *labelled* 1) and then continues to obey instructions from there onwards; otherwise, if the condition is not satisfied, then the next instruction is obeyed in the usual way. Any instruction can be labelled in this way with an integer in the range 1 - 127 inclusive.

The second jump instruction in the above example is an *unconditional* jump and needs no further explanation. The general form of a conditional jump instruction is

jump  $n, \alpha \geq \beta$  (or  $\neq >$ )

where  $n$  is a specific label and  $\alpha, \beta$  are the quantities being compared. These must be either both variables (including a numerical constant) or both indices (including a numerical integer). In this case a minus sign can be written before a constant if it is negative, but elsewhere numbers are treated as essentially positive and the + and - signs are treated as operators. It is *not* possible to compare a variable directly with an index without first converting the index to variable form. Examples of conditional jump instructions are

jump 1,  $x \geq y$

jump 8,  $1.41421 > a$

jump 50,  $i \neq 2$

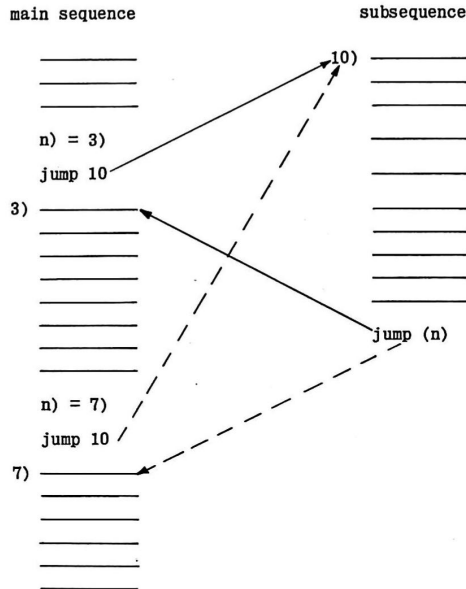
jump 97,  $r = s$

Associated with the above are the instructions

jump (n)                      n) = 3)                      n) = m)

where n, m denote any index letters.

The instruction n) = 3) makes a subsequent jump (n) equivalent to jump 3. One use for this will be to mark the point of return when calling in a subsequence, thus:



In the same way the instruction n) = m) makes a jump (n) instruction transfer control to one of several different points depending on the computed value of m. This device is known as a multi-way switch. When an index is used to 'remember' a label in this way it cannot, at the same time, be used for other purposes.

The execution time of n) = m) is very much longer (18 millisecs) than that of n) = 3) (120 microsecs).

There is a special form of conditional jump *check* (x, y, e, 3), where x, y, e are each any variable or constant, and 3 is any label in the same chapter. This has the effect "jump to the instruction labelled 3 if  $e > |x - y|$ , otherwise continue with the next instruction". It can thus be used to compare the computed value x with the expected value y to an absolute accuracy of e. In the case of failure, the value of x could be printed. This instruction may be found useful when developing a new programme.

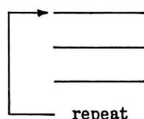
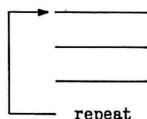
#### Cycles of Operations

Two special instructions are provided to simplify cycles of operations. These take the form

$$i = p(q)r$$

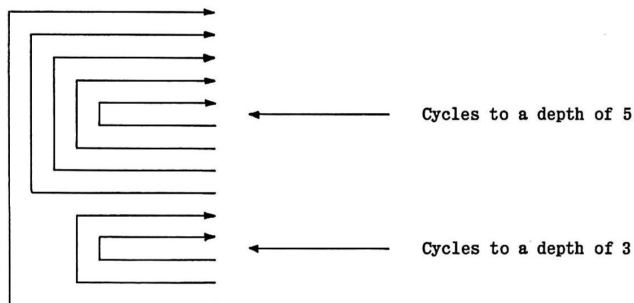
or

$$i = p(-q)r$$



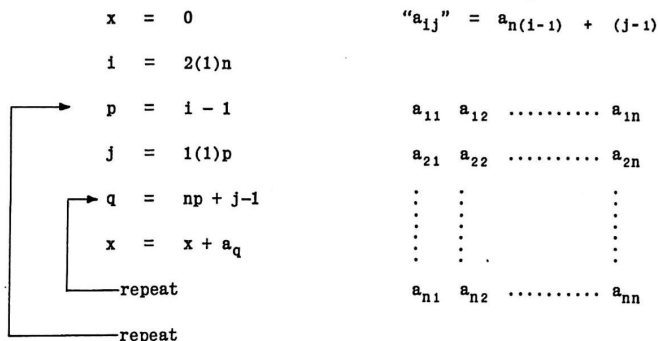
and arrange to execute the intervening instructions for values of  $i$  running from  $p$  by increments of  $q$  (or  $-q$ ) to  $r$ . Any index may be used in place of  $i$  and  $p, q, r$ , may be any indices or positive integers, subject of course to the restriction that  $r-p$  is a multiple of  $q$ , otherwise the cycle will continue indefinitely.

Each instruction of a pair is associated automatically one with the other by the computer during the initial input of the programme. The process of association will permit the use of instruction pairs entirely within other instruction pairs to a total depth of 8. It is not possible, however, to interleave pairs nor to duplicate either member of a pair, i.e. there must be a one to one correspondence of the opening and closing instructions of each loop, e.g.



This example is typical and illustrates the type of flow arrangement which is permissible. Note that no lines cross.

The following sequence



for example, illustrates a cycle within a cycle (for calculating the sum of the super diagonal elements of a square matrix).



Note that a cycle is always obeyed at least once and that if *i*, *q*, or *r* are written on the left hand side of an instruction in the loop the operation of the loop may be affected. At the end of the last repetition the values of *i* and *r* are equal.

End

The single word instruction *end* is used to terminate a calculation.

#### Input from a data tape

Instructions are provided to read numerical information from the input tape into the working store. These are

read (x)

read (i)

which mean "read the next number on the tape and set the specified variable (or index) to this value". As each number is read the tape is advanced to bring the next number to the reading station. Numbers must therefore be punched in the order in which they are required. Each number is written in the manner already described (preceded by minus sign if negative) and when punched must be terminated by *CR LF* or a double space *SP SP*. the "read" instruction will also accept numbers punched in the floating decimal form, for example, -2.5,-3 (i.e.,  $-2.5 \times 10^{-3}$  or -0.0025). The two component numbers can be punched in any acceptable form except that now the terminal combination of the first is replaced by the "comma". The exponent *b* takes integer values in the range  $|b| < 127$ . Further examples of data punched in floating decimal form are:-

1,-100	i.e.,	$10^{-100}$
-2.5,1	i.e.,	-25
1000,-3.0	i.e.,	1

the last number being in *non-standard* form. In all cases it is necessary to check that the size of the resulting product does not exceed the capacity of the machine, i.e.,  $|a \cdot 10^b| < 2^{256}$ .

Note: the floating decimal form applies *only* to numbers on a *data tape*, it must *not* be used for constants appearing in the programme.

#### Output

To print results the simplest procedure is to write a ? symbol before or after the arithmetical instructions giving the relevant value of the quantity in question, e.g.,

$x = zyy - 1 ? \quad i = i + 1 ?$

This will cause the new value of *x* (or *i*) to be printed immediately after computation. Each number is printed on a new line to 10 significant figures, so that results obtained in this fashion will be listed in a single column at the left hand margin of the page.

In case the results are required in tabular form, the following instructions are provided

print (x) m, n

space

newline

The first instruction prints the numerical value of  $x$  (any variable expression) in fixed decimal point style with  $m, n$  positions allowed respectively for the integral and fractional parts. The parameters  $m, n$  will usually be preassigned positive integers, e.g.,

print (2/r) 1, 5

but they may also be indices. The numerical value of  $x$  is rounded off by adding  $\frac{1}{2}10^{-n}$ . Integers are treated precisely. If  $n = 0$  the decimal point is omitted. One figure is always printed before the decimal point but other non-significant zeros are suppressed. If negative a minus sign appears before the first digit printed, otherwise a space. If numbers exceed about  $10^{10}$  they will be printed in floating decimal form (See next paragraph).

The *print* instruction will print numbers in standard floating decimal form on setting  $m = 0$ . Three characters (after the "comma") are allowed for the exponent - corresponding to a number printed with  $m = 2, n = 0$ . The true "zero" of the machine, namely  $0.2^{-256}$ , is printed as 0, -128 which will be interpreted correctly if subsequently read by the "read" instruction in spite of the fact that the exponent -128 lies outside the permissible range. (If  $a = 0$  the number  $(a, b)$  is treated as  $0.2^{-256}$  whatever the value of  $b$ .)

The query (?) print is equivalent to a *newline* followed by a formal *print* with  $m = 0, n = 10$  for variables, or  $m = 1, n = 0$  for indices.

Each number is automatically followed by two spaces but extra spaces can be "programmed" by means of the *space* instruction. A standard teleprinter carriage allows for up to 68 characters across the page. The instruction *newline* is equivalent to the operations of *line feed* and *carriage return* on a teleprinter.

All numbers printed by the machine can subsequently be read in again by means of the *read* instruction, i.e., input and output are complementary.

#### An Example

Tabulate Sievert's integral  $\int_0^y e^{-a \sec x} dx$  for  $y = 1(1)90^\circ$  and particular positive

values of  $a$ . The method adopted is to tabulate the integrand for  $x = 0(\frac{1}{2}) 90^\circ$  and then calculate the integral step-by-step using Simpson's rule to evaluate the increments; thus

$$\int_0^{y+h} = \int_0^y + \frac{h}{6} \left[ f(y) + 4f\left(y + \frac{h}{2}\right) + f(y+h) \right]$$

where  $h = \frac{\pi}{360}$

	$f \rightarrow 180$	
8	$h = \pi/360$	(5 ms.)
2	$s = 0(1)179$	1
10	$b = \phi \cos (sh)$	(23 ms.)
12	$f_s = \phi \exp (-a/b)$	(23 ms.)
4	repeat	4
4	$f_{180} = 0$	4
4	$y = 0$	4
2	$r = 1(1)90$	1
4	newline	(90 ms.)
7	print (r)2,0	(150 ms.)
2	space	(60 ms.)
11	$s = 2r - 1$	19
10	$f = f_{(s-1)} + 4f_s + f_{(s+1)}$	22
9	$y = y + hf/3$	(5 ms.)
7	print (y) 1,6	(333 ms.)
4	repeat	4
	end	

The case  $s = 180$  is excluded from the first loop in order to avoid forming  $-a/b$  for  $b = 0$ . Otherwise the programme itself needs no explanation. The numbers on the left give the number of *registers* occupied by each instruction in the instruction store, while those on the right give the execution time in units of  $60 \mu s.$ , unless otherwise stated. Most functions take approximately 23 ms., unless treated as *quickies* (see Part 2), in which case the time is reduced to about 6 ms. Approximate rules for estimating the space and time requirements are given at the end of Part 2.

## Chapters

For technical reasons, large programmes have to be partitioned into *chapters*, each chapter being restricted in length to 832 registers. (The above programme is well within this limit and would thus constitute a single chapter.) Each chapter has its own labelling system, and to jump from one chapter to another an across instruction is employed, e.g.,

across 2/3

means "jump to the instruction labelled 2 in chapter 3". Since this is a comparatively time consuming instruction (160 ms.) it should not be included within an inner loop and as far as possible partition into chapters should correspond to distinct parts of the calculation.

The diagram illustrates the layout of a multi-chapter programme. Each chapter is headed with the *chapter number* and the variable setting directives, and terminates with the directive word *close*. If the directives in chapter 3 are the same as those in chapter 2 then we may simply write:-

```
chapter 3
variables 2
instructions
close
```

While if extra directives are to be added we may write:-

```
chapter 3
variables 2
x → 99
```

chapter 1

directives

instructions

close

chapter 2

directives

instructions

close

chapter 3

directives

instructions

close

Note however that this substitutional directive may only refer to a *previous* chapter on the programme tape. It is necessary to arrange chapters in ascending numerical order as shown.

The associated instructions

i = p (q) r	l) = 3)	or	l) = m)
.	.		
.	.		
.	.		
.	.		
repeat	jump (l)		

must occur in the same chapter, although the intervening instructions may temporarily switch control to another chapter. (Normally the indices 'l' and 'o' should be used as little as possible since they are readily confused with 1 and 0. In the above case, however, a connection exists between l and label, and there is little risk of confusion.)

### The directive "psa"

This means "print space available" and is intended to be inserted between the last instruction of a chapter and the *close* directive. The effect is to print, on a newline, the chapter number followed by the number of unused registers at the end of the chapter. The directive may also be used at any point in the chapter so as to record how the chapter space is used up.

### The significance of the variable directives

When resetting directives in subsequent chapters it may be necessary to appreciate the significance of these statements. Thus e.g.,

chapter 1

a → 99	allocates	[	0 - 99	to	a <sub>0</sub> , a <sub>1</sub> , ....., a <sub>99</sub>	
b → 99			storage	100 - 199	to	b <sub>0</sub> , b <sub>1</sub> , ....., b <sub>99</sub>
c → 99			locations	200 - 299	to	c <sub>0</sub> , c <sub>1</sub> , ....., c <sub>99</sub>

chapter 2

c → 99	allocates	[	0 - 99	to	c <sub>0</sub> , c <sub>1</sub> , ....., c <sub>99</sub>	
x → 49			storage	100 - 149	to	x <sub>0</sub> , x <sub>1</sub> , ....., x <sub>49</sub>
y → 49			locations	150 - 199	to	y <sub>0</sub> , y <sub>1</sub> , ....., y <sub>49</sub>

Thus the "c's" of chapter 1 are not those of chapter 2. If it is intended that they should be, then the latter directives might be recast as follows

x → 49
y → 49
π → 99 (waste)
c → 99

A further consequence of this scheme is that " $x_{50}$ " is identical with  $y_0$ , " $x_{51}$ " with  $y_1$ , and so on. Such "overlapping" references are sometimes useful. However one must not try to refer to " $x_{-1}$ " since there are no variables preceding those defined by the *x* directive.

### Subchapters

At any point within a chapter it is possible to call in a "subchapter" and subsequently to return to the original chapter at the instruction following the point of departure. This is done by means of a *down* instruction in the main chapter and an *up* instruction in the subchapter. For example

down 2/3

calls in chapter 3 as a subchapter and enters it at the instruction labelled 2. When the subchapter has completed its task the single word instruction

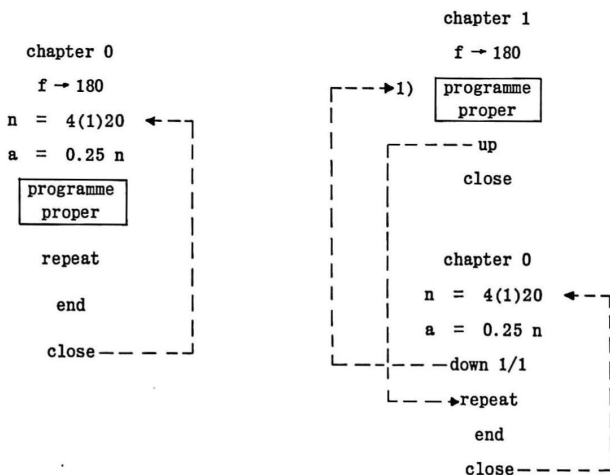
up

will return control to the main chapter at the instruction following the original *down* instruction. Alternatively the *up* instruction may be used in any chapter reached by means of *across* instructions from the original subchapter. A subchapter may have its own sub-subchapter, but there the regression stops.

The special variable  $\pi$  is reset to 3.14159... at every chapter change, i.e., as a result of the instructions *across*, *down*, *up*; and also (see later) *preserve* and *restore*.

### Starting the programme

Following the last chapter on the programme tape is *chapter 0*. This is constituted in the same way as any other chapter, but the *close* directive terminates the programme input process and initiates the programme itself at the first instruction (whether labelled or not) of chapter 0. If there is only one chapter in the programme this could be chapter 0, but it is more usual to regard chapter 0 as a steering chapter. The two alternatives are illustrated below where the calculation of Sievert's Integral, given earlier, is arranged for values of the parameter  $a = 1(0.25)5$



### The "rmp" instruction

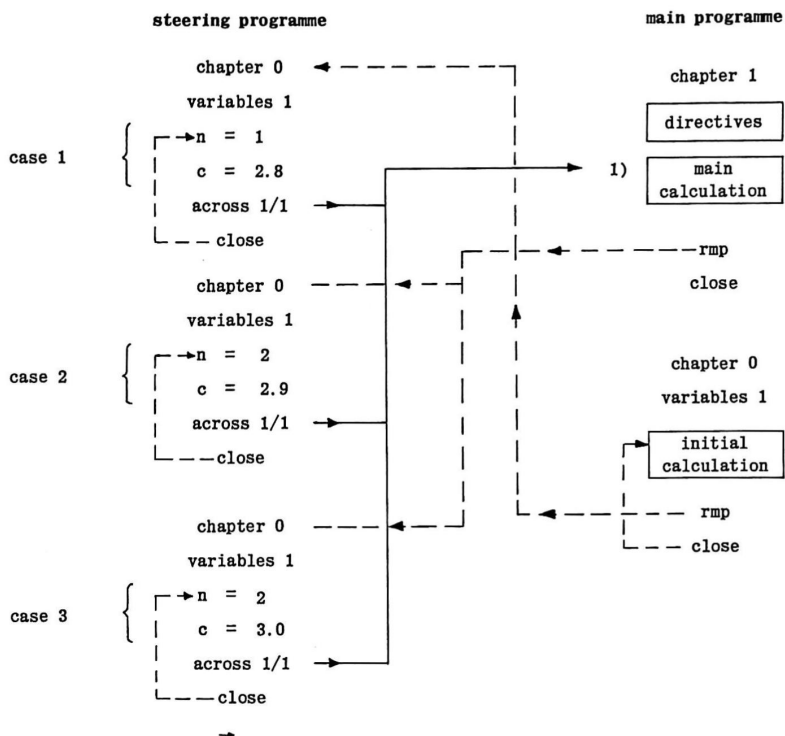
It has been stated above that a programme consists of chapters numbered 1, 2, 3,.....0, and that when chapter 0 has been read the machine stops reading programme and starts obeying it. At this time the number store and any part of the instruction store not filled with instructions are in a standard state.

However, there is a special instruction *rmp* (read more programme) whose execution makes the machine resume reading programme. Thus chapter 0 can be followed either by a completely new programme which will entirely supersede the old one, or by a new chapter 0 which will supersede the old chapter 0. Thus a succession of steering chapter 0's may be written. In either case after reading another chapter 0 the machine again stops reading programme and starts obeying it. Whereas when execution of the original programme started the number store was in a standard state, the new (or modified) programme finds the numbers left in the number store by the previous programme.

### Steering programmes and rescue procedure

When a calculation has to be repeated for miscellaneous values of certain parameters, it is sometimes convenient to initiate each individual case by restarting with a supplementary chapter 0, which sets up the parameters in question before entering the programme proper. Such a succession of chapter 0's is called a steering programme. The machine can proceed to successive cases *automatically* by terminating the programme cycle with the *rmp* instruction. Since numerical results are unaffected by restarting, the successive chapter 0's need only refer to those parameters which are altered in passing from one case to the next. However, this may rule out the possibility of initiating the individual cases out of sequence, or of "rescuing" the calculation in the event of machine breakdown.

For this reason it is preferable to make the individual chapter 0's independent by resetting if necessary all initial data. If there is a large mass of data, such as a table, to be generated, and which is common to each case, then this should be initiated by the main programme and recorded for future reference in an appropriate part of the store. If this part of the programme also terminates with *rmp*, the machine can proceed automatically to read the first chapter 0 of the steering programme. If the calculations have to be spread over more than one machine session, or in the event of serious machine breakdown, the main programme will have to be read again. If the programme has been suitably written, this can immediately be followed by any chapter 0 on the steering tape. The following diagram illustrates the arrangement of programme for three "cases" of a calculation specified by two parameters.



### The hoot instruction

The instruction *hoot* interrupts the calculation to give a single note (1 kc/s) of duration approximately 1 sec. on the loudspeaker. This may prove useful when planning a production run with a steering tape. By arranging to *hoot* as each case is finished, the operator is thereby reminded that the machine is behaving correctly, and should it "jump out of control" the interruption would be noticed. In this event the operator can restart the last case manually.

## Part 2

### Further Facilities

The information given so far will enable the reader to attempt a fairly wide range of problems - with limited storage requirements. The use of the auxiliary store is discussed in the next few pages which describe several additional facilities. The first of these is a device for speeding up the execution time of chapters involving functions.

#### Quickies

Every time one of the functions

sqr	(48)	cos	(36)	log	(42)	tan	(42)	chapter 1
radius	(48)	sin	(36)	exp	(50)	arctan	(58)	variables

is referred to, 17 millisecs are spent in transferring the necessary set of instructions (subroutine) from the magnetic drum to the instruction store. Provided there is room, however, they can be included in the chapter itself, and in this case the average execution time is reduced from about 23 millisecs to 6 millisecs. Functions treated in this way are known as *quickies*. The number of registers required for each function is given above in parentheses. All that is necessary is to list them (each preceded by  $\phi$ ) in order of preference at the end of the chapter in question, immediately before the *close* directive (as in the accompanying diagram). Any functions for which there is not room will be treated in the usual way.

instructions

 $\phi$  exp

 $\phi$  sqrt

 $\phi$  sin

close

In the case of "sin" and "cos" these functions involve the same set of instructions, so that if one is treated as a quicky, the other will be also. The same applies to "sqrt" and "radius". Finally it should be mentioned that the functions

mod      int pt      fr pt      divide

are automatically treated as quickies, so that there is no need to include them in a quicky list.

#### Rounded and unrounded arithmetical operations

In those arithmetical instructions involving a *variable* expression on the right hand side, each sum, difference, and product is formed to a maximum precision of 29 binary digits and rounded by making the last digit odd. If required, the rounding operation can be omitted by using the  $\approx$  sign instead of  $=$ . In this case the result will normally be biased, but if the quantities involved can be expressed precisely in 29 significant binary digits or less, then the  $\approx$  sign provides a means of performing exact arithmetical operations (excluding division) on variables. These will usually be restricted to integral values, however, since





$\phi_7 (d-n+1) a_1, n$  Transfers the variables  $a_1, a_2, \dots, a_n$  to auxiliary locations  $d-n+1, \dots, d$ .

$\phi_6 (f) g, 1$  Replaces the special variable  $g$  by a number from the auxiliary store.

It is very often appropriate to associate letters with groups of numbers in the auxiliary store, e.g., when dealing with matrices. For this purpose fourteen new working variables

$$a', b', c', d', e', f', g', h', u', v', w', x', y', z'$$

are introduced. These are essentially similar to the special variables  $a, b, c, \dots, z$  and indeed may be used as such if desired. It is intended, however, that they be used to designate the starting locations of groups of auxiliary variables. Thus for example by setting  $a' \approx 1000$  we define a group of unlimited extent located in 1000, 1001, etc. If the group is an  $(n \times n)$  matrix and recorded so that the  $(i, j)$ th element stands in  $a' + (i-1)n + (j-1)$  then the instruction  $\phi_6(a' + in - n) b_{0,n}$  transfers the  $i$ -th row of the matrix to the working store.

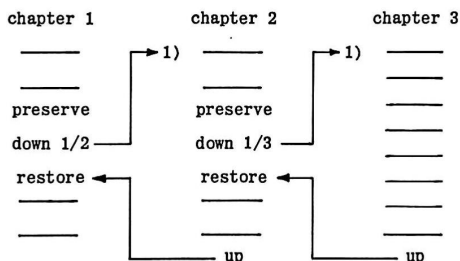
The execution time for a group transfer cannot be given very precisely but is less than

$$\left\{ 17 \left[ \frac{n}{32} \right] + 34 + 0.36 n \right\} \text{ msec.}$$

where  $[ ]$  denotes "integral part of", and  $n$  is the number of variables transferred.

#### The instructions "preserve" and "restore"

These apply to the use of subchapters. If necessary the working variables can be preserved during the operation of the subchapter and restored on return to the main chapter. This can be done by writing the word *preserve* before the *down* instruction, and the word *restore* immediately after it. The variables in question are dumped in hidden locations of the auxiliary store. There are two dumps, the first being used by the master chapter when calling in a subchapter, and the second by a subchapter when calling in a sub-subchapter. A programme may thus extend over three levels, as illustrated in the following diagram



A consequence of this arrangement is that any results calculated by the subchapter will have to be recorded in the auxiliary store in order to preserve them. Alternatively, if they are left in the working store, then the *restore* instruction can be postponed until they have been dealt with in the master chapter.

It is recommended, however, that subchapters be designed so as to select from, and return to, the auxiliary store all relevant material. In this form they provide a convenient means of arranging a calculation for possible future use as a "packaged" programme. The subchapters can be written in terms of auxiliary groups  $a', b', c', \dots$ , without

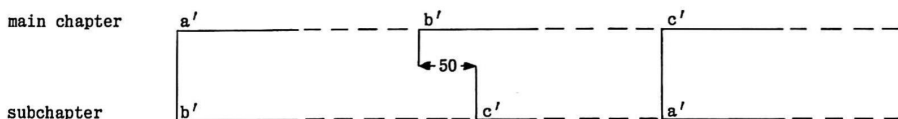
assigning numerical values to these variables. Instead they can be defined in terms of those used in the main chapter by including appropriate instructions between the words *preserve* and *down*. The original values of  $a'$ ,  $b'$ ,  $c'$ , etc. can then be restored on returning to the main chapter. In this way each 'level' of organisation may use its own frame of reference for the auxiliary variables. As an example, suppose that both the main chapter and the subchapter employ three groups of auxiliary variables denoted in both cases by  $a'$ ,  $b'$ ,  $c'$ , and starting relative to each other as follows

$$a'_{(\text{sub})} = c'_{(\text{main})}$$

$$b'_{(\text{sub})} = a'_{(\text{main})}$$

$$c'_{(\text{sub})} = b'_{(\text{main})} + 50$$

These relations may be represented diagrammatically thus:-



The instructions for calling in the subchapter and redefining the auxiliary variables are then as follows

preserve

$$\pi \approx \mathfrak{a}'$$

$$a' \approx c'$$

$$c' \approx b' + 50$$

$$b' \approx \pi$$

down ?/?

restore

The special variable  $\pi$  has been used as a “shunting station” since in any case it will be reset on entering the new chapter ( $\pi$  is automatically reset to 3.14159... as a result of the instructions *across*, *down*, *up*, *preserve*, and *restore*). Normally it is not advisable to alter the value of  $\pi$ .

## Operations with complex numbers

Operations with complex numbers written as number pairs are provided by instructions of the form

$$(u, v) = (x, y)$$

$$(u, v) = (x, y) + (a, b)$$

$$(u, v) = (x, y) - (a, b)$$

$$(u, v) = (x, y) * (a, b)$$

$$(u, v) = (x, y) / (a, b)$$

$$(u, v) = \phi \sqrt{x, y} \quad (u > 0)$$

$$(u, v) = \phi \log (x, y) \quad (\pi > v > -\pi)$$

$$(u, v) = \phi \exp (x, y)$$

Here  $u, v$ ;  $x, y$ ;  $a, b$  denote any variables or (except in the case of  $u, v$ ) signed constants. Examples of instructions in this class are

$$(f_i, g_i) = (f_{(i-1)}, g_{(i-1)}) + (f_{(i+1)}, g_{(i+1)})$$

$$(x, y) = \phi \text{ sqrt } (1, 1)$$

$$(a, b) = \phi \log (-0.5, h)$$

Note that no instruction may contain more than one operation.

The complex functions involve the use of certain real functions, as follows

$$\text{complex} \begin{cases} \text{sqrt involves sqrt} \\ \text{exp} \quad \quad \quad \text{exp, sin, cos} \\ \text{log} \quad \quad \quad \text{log, arctan} \end{cases}$$

Thus in order to treat the complex functions as quickies, it is sufficient to list the relevant real functions. The ?-print facility does not apply to the complex operations.

### Double Precision Instructions

Certain limited double-length facilities have been incorporated into Autocode. These consist of the four basic arithmetical operations and, in addition, a reciprocal operation and a single copy.

The instructions take the form:-

$$(i) \quad ((x, y)) = ((a, b)) \theta ((c, d))$$

$$(ii) \quad ((x, y)) = 1/((a, b))$$

$$(iii) \quad ((x, y)) = ((a, b))$$

where  $\theta$  is replaced by one of  $+ - * /$  corresponding to addition, subtraction, multiplication and division, and where  $a, b, c, d, x, y$  may be replaced by any variable, either working or special (see examples).

The combination  $((x, y))$  will be interpreted as the two parts of the double length number in the above operations,  $x$  being the more significant half and  $y$  the less significant half. On Mercury  $x$  is represented to a precision of 28 binary digits and  $y$  to 29, i.e. a total of 57 binary digits. Hence  $y$  should be at least  $2^{-28}$  ( $\sim 10^{-9}$ ) times smaller than  $x$ ; this condition will automatically hold true for pairs  $x, y$  computed by the above operations, but note that the copy in no way alters the numbers being copied. (On Atlas or Orion of course the corresponding figures will be quite different.)

There remains the problem of single to double-length conversion. Any variable or constant, expressed to a precision of at most 28 binary digits, e.g., an integer less than  $10^8$ , can be converted to standard double length form by associating with it a zero less significant half. Thus if  $a_{(j-3)}$  contains an exact quantity then to obtain  $a_{(j-3)}/3$  to double precision, one writes

$$((x, y)) = ((a_{(j-3)}, 0)) / ((3, 0))$$

Notice that, if  $a_{(j-3)}$  is not exact but contains a rounding error, then the operation is worthless, as  $x$  will contain the same relative error and  $y$  will be meaningless.

With the above considerations, the facilities described in the second paragraph of this section may be extended by noting that any pair  $((a, b))$  or  $((c, d))$  in equation (i) (ii) may be replaced by  $((u, 0))$  where  $u$  is any variable or constant which can be represented exactly.

The following examples will illustrate these facilities.

$$\begin{aligned} ((a_i, a_{i+1})) &= ((b_j, c_j)) / ((5, 0)) \\ ((x', y')) &= 1 / ((a_{(k-3)}, b_{(k-3)})) \\ ((x, y)) &= ((2, 0)) * ((a, 0)) \end{aligned}$$

Double length pairs should, for convenience of use, and to avoid errors, be stored either in corresponding positions under two different letter, e.g.  $x_3, y_3$ , or as adjacent numbers under the same letter, e.g.  $v_j, v_{j+1}$ , when the even suffices will indicate the more significant part.

It is expected that double length working will be mainly used to minimise the effects of destructive cancellation in "ill-conditioned" problems; and that there will be no great need for double length input and output routines. For this reason these operations will be available only as the library sub-programme -512.

However, even without this, something can be done.

A number consisting of integral and fractional parts can be read as two distinct numbers and combined to form a double precision quantity as follows:

$$\begin{aligned} \text{read } (x) \text{ integral part } 0 \leq |x| &\leq 2^{28} \\ \text{read } (y) \text{ fractional part } 0 < |y| &< 1 \\ ((u, v)) &= ((x, 0)) + ((y, 0)) \end{aligned}$$

conversely we can print a number in this form, thus

$$\begin{aligned} ((u, v)) \\ x &= \phi \text{ int pt } (u) \\ \text{print } (x) \text{ ?,0} \\ y &= \phi \text{ fr pt } (u) \\ ((u, v)) &= ((y, 0)) + ((v, 0)) \\ \text{print } (u) \text{ 1,?} \end{aligned}$$

Finally, in any chapter in which any of the double-length operations are used, the directive

double length

must be inserted after the variable directives for that chapter and before the first instruction. Thus

$$\begin{aligned} a &\rightarrow 4 \\ b &\rightarrow 4 \\ \text{double length} \\ 3) x &= 4 \end{aligned}$$

will satisfy the requirements. This directive will use up 128 of the available 832 registers.

The space occupied by these instructions is (i) 13 - 23 registers, (ii) 9- 15 registers, (iii) 6 - 10 registers. The times for the operations are:-

addition 70  $\mu$ min., subtraction 90  $\mu$ min., multiplication 120  $\mu$ min.  
reciprocal 480  $\mu$ min., division 600  $\mu$ min., copy 14  $\mu$ min.

### Step-by-step integration of differential equations

Special facilities are provided for the integration of differential equations. The equations must be written in the form

$$f_i = \frac{dy_i}{dx} = f_i(x, y_1, y_2, \dots, y_n) \quad (i = 1, 2, \dots, n)$$

involving the special variables  $x$ , the main variables  $y_i$ ,  $f_i$  and the index  $n$ . In addition the special variable  $h$  is used for the step length and the main variables  $g_i$ ,  $h_i$ , ( $i = 1, 2, \dots, n$ ) are introduced for "working space". The programmer must write a subsequence for calculating the  $f_i$  in terms of  $x$  and the  $y_i$ , and which must not alter  $y_i$ ,  $g_i$ ,  $h_i$  nor  $n$ ,  $h$ ,  $x$ . The entry should be labelled and the sequence should terminate with the special instruction 592, 0 (see Appendix 3). With these arrangements the effect of the instruction

int step ( $m$ )

(where  $m$  is the entry to the subsequence) is to advance the integration by one step so that the initial and final values of the independent and dependent variables are respectively

$$\begin{array}{cc} x & x+h \\ y_i(x) & y_i(x+h) \end{array}$$

The method employed is that of Runge-Kutta, with truncation error of  $O(h^5)$ . However the truncation error also depends on the higher derivatives of the function and for this reason the step length may be adjusted between steps if desired. The time per step is  $(10n + 4T)$  millisecs, where  $T$  is the time (in millisecs) of the subsequence.

### Example

Tabulate the solution of the equations:-

$$\frac{dy_1}{dx} = y_1^2 - 1.23 y_1 y_2 + 2.47 y_2^2$$

$$\frac{dy_2}{dx} = 1.01 y_1^2 - 0.84 y_1 y_2 + 1.59 y_2^2$$

for  $x = 0(.02)1$

with the initial conditions:-

$$1) \quad x = 0; \quad y_1 = 0, \quad y_2 = 1$$

$$2) \quad x = 0; \quad y_1 = 1, \quad y_2 = 0$$

chapter 0	notes
$f \rightarrow 2$	
$g \rightarrow 2$	
$h \rightarrow 2$	
$y \rightarrow 2$	
$n = 2$	no. of equations
$h = 0.02$	step length
$m = 1(1)2$	} selects initial conditions
$i) = m)$	
jump (i)	
11) int step (10)	
newline	} tabulates results
print (x) 1, 2	
space	
print ( $y_1$ ) 2, 4	
print ( $y_2$ ) 2, 4	} tests for end of range
jump 11, $0.99 > x$	
repeat	
end	
10) $f_1 = y_1 y_1 - 1.23 y_1 y_2 + 2.47 y_2 y_2$	} auxiliary sequence
$f_2 = 1.01 y_1 y_1 - 0.84 y_1 y_2 + 1.59 y_2 y_2$	
592, 0	
1) $x = 0$	} subsequence for setting initial conditions (1)
$y_1 = 0$	
$y_2 = 1$	
jump 11	
2) $x = 0$	} ditto (2)
$y_1 = 1$	
$y_2 = 0$	
jump 11	
close	

Note: the instructions

int step (m)

```

      .
      .
      .
m) ———
      .
      .
      .
      592, 0

```

must occur in the same chapter, although the auxiliary sequence may involve transfer of control to another chapter.

### Miscellaneous Facilities

Mercury Autocode allows the basic machine instructions to be included in a programme if appropriate. In general it will not be possible to translate such instructions for Atlas and Orion, and for this reason we have confined their description to an Appendix. However, certain of them are normally used to obtain alpha-numeric output and to generate pseudo-random numbers. These operations are done as follows:-

#### Alpha-numeric output

The effect of the "620, n" instruction is to punch (print) the character given in the accompanying table of tape values. Thus e.g., to print the sequence

case n    x    =    current value of x

on a new line we write:-

new line	(also puts printer on figure shift)
620, 27	letter shift
620, 3	c
620, 1	a
620, 19	s
620, 5	e
print (n) 4, 0	prints current value of index n (followed by 2 spaces)
620, 27	letter shift
620, 24	x
space	(also puts printer on figure shift)
620, 10	=
print (x) p, q	prints current value of the special variable x. The number of spaces separating the "=" and the leading figure of x will depend on the size of x and the value of p.



## Tape Code for Mercury Computer

Tape	Value	FS	LS
.	0	FS	FS
. .	1	1	A
. .	2	2	B
. . .	3	*	C
. . .	4	4	D
. . . .	5	(	E
. . . .	6	)	F
. . . . .	7	7	G
. . . . .	8	8	H
. . . . .	9	≠	I
. . . . .	10	=	J
. . . . .	11	-	K
. . . . .	12	≈ (ν)	L
. . . . .	13	LF	M
. . . . .	14	SP	N
. . . . .	15	,	O
. . . . .	16	0	P
. . . . .	17	>	Q
. . . . .	18	≥	R
. . . . .	19	3	S
. . . . .	20	→	T
. . . . .	21	5	U
. . . . .	22	6	V
. . . . .	23	/	W
. . . . .	24	φ (x)	X
. . . . .	25	9	Y
. . . . .	26	+	Z
. . . . .	27	LS	LS
. . . . .	28	.	.
. . . . .	29	' (n)	?
. . . . .	30	CR	π (£)
. . . . .	31	*	*

Symbols in brackets may be found on some sets of equipment.

## The generation of pseudo-random numbers

The recurrence relation

$$x_{r+1} = a x_r \pmod{2^{29}}$$

is used for this purpose, where  $a = 43^5$ .

The most convenient method of working this recurrence on Mercury involves the use of un-standardised floating point numbers. The following sequences are employed

$$\left. \begin{array}{l} i = 29 \\ j = 955 \\ k = 202 \\ l = 140 \\ 400, 58 \\ 410 (a) \end{array} \right\} \text{sets } a = 43^5$$

i = 0	} sets $x_0$
j = 1	
k = 0	
l = 0	
400, 58	
410 (x)	

Thereafter we can generate a new random number (in the range  $0 < x < 1$ , but un-standardised) by the sequence

400 (x)

540 (a)

410 (x)

The least significant digits of  $x$  are not reliable. Having obtained a new  $x$ , the instruction (for example)

$t = \phi \text{ int pt } (100 x).$

yields a random integer in the range 0 - 99 inclusive.

The initial value of  $x_0$  can be changed by altering the 'k' component, e.g.,

i = 0

j = 1

k = 1

l = 0

The above facility will enable the user to write programmes for plant simulation and allied problems.

#### Estimation of time and space for Autocode programmes

AUTOCODE INSTRUCTIONS	Space occupied in machine instructions	Time in $\mu\text{min.}$
Input of Instructions and Directives	-	3,000 per instruction or directive
Arithmetical Expressions		
$\Sigma x$ ; Each letter	1	1
Each + - =	1	1
Each index	4	10
Each constant	3	1
Each solidus	4	60
Each implied multiplication sign	0	5
$\Sigma i$ ; Each letter or constant	1	0
Each + - =	0	1
Each implied multiplication sign	4	10

*Continued*

## AUTOCODE INSTRUCTIONS

Space occupied  
in machine  
instructionsTime in  
 $\mu$ min.

## Functions (Arguments as above)

$\phi$ divide	6	60
$\phi$ mod, intpt, frpt, sign, parity	6	10
$\phi$ sqrt, sin, cos, tan, exp, log, arctan, radius	6	100 + 300 (access)

## Control

label	0	-
jump 2	1	1
jump 2, conditional	6	6
n) = 3)	2	2
n) = m)	6	300
jump (n)	2	2
i = p(q)r	6	2
repeat		5
across 2/1	5	2,000
down 2/1	6	
up	3	
r m p	3	3,000
end, halt	1	

## Auxiliary Transfers

$\phi_6 (x + y_j) b_k, n$	10	600 + 14n
$\phi_7 (x + y_j) b_k, n$	10	600 + 22n
preserve, restore	4	3,000

## Input/Output

read ( $a_{(i+1)}$ ) 10 characters	6	1,750
space	2	1,000
new line	3	2,000
print ( $\sum x$ ) m, n	8	3,000 + 500 (m+n)
? i	2	4,000
? x	2	8,000

## Notes

1. *Space.* Although, owing to the wide variety of permissible forms, it is not possible to say exactly how many instructions will fit into a single chapter, the figure of 100 has been found to be a reasonable estimate. If a chapter becomes too long, this will be indicated during input and the 100 subsequent characters will be printed out. The *psa* directive used during development of the programme will also give a good idea of how the space is being used up.

In general this is all that need be done about the space occupied, but occasionally it may be necessary to estimate the space more exactly, e.g., to avoid a chapter change in an inner loop. In this case the table may be used.

Consider the expression  $z_i = y_{(i+1)} + x y_{(i-1)} + 3i - 4/x$ .

For each of the 9 letters count 1 machine register, giving 9 registers

For each +, -, =	{	not	"	1	"	"	"	4	"
For each index		occurring	"	4	"	"	"	4	"
		in a							
For each constant	{	suffix	"	3	"	"	"	6	"
For each solidus			"	4	"	"	"	4	"

i.e. 27 registers

2. *Time.* In many cases the ratio of computing to output time is so small that only the output need be counted when estimating the time. In other cases there is no need to analyse anything except the inner loop since nearly all the time is spent doing these instructions.

$$\text{e.g. } z_i = y_{(i+1)} + x y_{(i-1)} + 3i - 4/x$$

For each of the 9 letters count 1  $\mu$ min giving 9  $\mu$ min

For each +, -, =	{	not	count	1 $\mu$ min	"	4 $\mu$ min
For each index		occurring	"	10 $\mu$ min	"	10 $\mu$ min
		in a				
For each constant	{	suffix	"	1 $\mu$ min	"	2 $\mu$ min
For each solidus			"	60 $\mu$ min	"	60 $\mu$ min
For each implied multiplication			5 $\mu$ min	"	5 $\mu$ min	

90  $\mu$ min

3. *Accuracy.* The above figures have an accuracy of about 10% when applied to typical instructions.

## Part 3

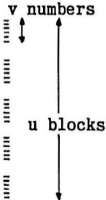
# Matrix Operations

### Special Operations to Facilitate Matrix and Vector Arithmetic

These operations assume that rectangular matrices are recorded as groups of auxiliary variables, the rows being stored 'end-to-end'. More precisely, the special variable  $a'$  will be associated with a matrix  $A$  of  $m$  rows, and  $n$  columns. When the location of element  $a_{ij}$  is given by  $a' + in + j$ ,  $i=0(1)m-1, j=0(1)n-1$ . In other words,  $a'$  identifies only the starting point of the corresponding group of auxiliary variables (see Part 2). The dimensions are specified by additional parameters. Although in the table below,  $a', b', c'$  have been used to specify matrices  $A, B, C$ ; we could just as well have used  $x', y', z'$  (or indeed any variables or whole numbers) to represent  $X, Y, Z$ . The dimensions of the matrices are (with one exception) also specified by any variables or whole numbers;  $u, v, w$  are used in the equations below. Finally, where a scalar variable is involved, e.g., in the calculation of a determinant, we have used  $x$  for this purpose.

In all these equations, the matrix operands are selected from the auxiliary (magnetic drum) store, and the resulting matrix returned there. Naturally, this involves the use of the high speed working store, but arrangements are made to preserve the contents in a 'secret' dump, and restore them after the operation. (Note: this 'secret' dump is also used in the  $\text{mp}$  operation, to hold the contents of the working store when converting more Autocode instructions.)

### Table of Matrix operations

$\phi 8(a', u, v, m, n)$	Prints the ( $u \times v$ ) matrix $A$ in a single column, each row being followed by an extra line-feed. i.e. $u$ blocks of $v$ numbers. Each element is printed in conventional fixed-point form, with provision for $m, n$ decimal places before and after the point. This routine automatically switches over to floating-point style for numbers greater than about $10^{10}$ (see Part 1).	
$\phi 9(a', u, v, n)$	As above, but printing in floating-decimal style, in the form $a.b$ . $n$ is the number of decimal places in $a$ ; $m$ is irrelevant, and therefore omitted. $\phi 8$ and $\phi 9$ will handle matrices of all sizes up to $u = 511$ or $v = 511$ .	
$\phi 10(a', u)$	Reads the vector or matrix $A$ consisting of $u$ elements in all, in fixed- or floating-point form, from punched tape.	
$a' = \phi 11(b', c', u)$ $a' = \phi 12(b', c', u)$ $a' = \phi 13(b', x, c', u)$ $a' = \phi 14(b', x, c', u)$ $a' = \phi 15(b', u)$	$A = B + C$ $A = B - C$ $A = B + xC$ $A = B - xC$ $A = B$	Linear combination of groups i.e., vectors or matrices, each containing $u$ elements. $1 \leq u$ .

$a' = \phi 16(b', u, v)$   $A_{(v \times u)} = B_{(u \times v)}^T$  Matrix transpose.  $1 \leq u, 1 \leq v \leq 479$ .

$a' = \phi 17(b', u)$   $A = B + I$   
 $a' = \phi 18(b', u)$   $A = B - I$   
 $a' = \phi 19(b', x, u)$   $A = B + xI$   
 $a' = \phi 20(b', x, u)$   $A = B - xI$

Add or subtract a multiple of the unit matrix to or from B, order u.  $1 \leq u < 239$ . In these and the following four equations u refers to the *order* of the (square) matrix, *not* to the total no. of elements as in  $\phi 11$ - $\phi 15$ .

$a' = \phi 21(b', d', u)$   $A = B + D$   
 $a' = \phi 22(b', d', u)$   $A = B - D$   
 $a' = \phi 23(b', x, d', u)$   $A = B + xD$   
 $a' = \phi 24(b', x, d', u)$   $A = B - xD$

Add or subtract a multiple of the diagonal matrix D (stored as a vector) to or from B, order u.  $1 \leq u \leq 239$ .

$x = \phi 25(a', u)$   $x = |A|$  Replace x (any working variable) by the determinant of A, order u.  $2 \leq u \leq 97$ . A is destroyed (see Note 2).

$a' = \phi 26(b', c', u, v, w)$   $A_{(u \times v)} = B_{(u \times w)} C_{(w \times v)}$   
 $a' = \phi 27(b', c', u, v, w)$   $A_{(u \times v)} = B_{(u \times w)} C_{(v \times w)}^T$

Matrix multiplication.  $1 \leq u, v, w \leq 129$

$a' = \phi 28(b', m, n)$   $A_{(m \times n)} = B_{(m \times m)}^{-1} A_{(m \times n)}$  Matrix division.

Here, the dimensions are specified by *indices*, or whole numbers. (This is the exception mentioned in the preamble.)  $2 \leq m \leq 97$ ,  $1 \leq n \leq 97$ . B is destroyed, and A is replaced by the result. (See Note 2.)

### Examples

1)  $a' = \phi 20(b', 0.33333, 15)$   $A = B - \frac{1}{3}I$  where A and B are (15x15) matrices.

2)  $x' = \phi 26(y', z', 30, 40, d)$   $X_{(30 \times 40)} = Y_{(30 \times d)} Z_{(d \times 40)}$

It is possible to vary the dimensions of matrices operated on by setting them elsewhere. Here, d would be computed (as a precise integer) before the multiplication takes place. Care must be taken to allow room for the largest matrices.

3)  $a' = \phi 26(b', c', 30, 1, 30)$  Product of matrix and vector, namely:

$$A_{(30 \times 1)} = B_{(30 \times 30)} C_{(30 \times 1)}$$

4)  $a' = \phi 16(a', 4, 3)$   
 $\phi 8(a', 3, 4, m, n)$

If a matrix has to be printed in array form,  $\phi 16$  may be used to transpose the matrix *on top of itself* before output. In this example,  $A_{(4 \times 3)}$  is first transposed, and then printed in 3 columns of 4 numbers with m,n layout. Always remember to *re-transpose* if A is to be used for further calculations (see Note 1).

5)  $a' = \phi 17(a', 20)$

If  $A=0$  (as would be the case if the auxiliary store were clear - 0.2<sup>0</sup>), then this operation sets  $A=I$ , a 20x20 unit matrix.

6)  $a' = \phi 28(b', 20, 20)$

If A is a 20x20 unit matrix, and B is a 20x20 general matrix, then this is equivalent to  $A=B^{-1}$ . B itself is destroyed.

7)  $x \rightarrow 14$

$\vdots$

$n = 0(1)14$

Forms $x_0$ to $x_{14}$
----------------------------

Matrices can be built up row by row in the auxiliary store by using  $\phi 7$ .

$\phi 7(a' + 15n)x_0, 15$

repeat

In all the examples, we have used the primed special variables to specify matrix addresses, but there is no reason why we should not use other variables, or absolute constants.

e.g.  $e_1 = \phi 20(e_2, 0.333333, 15)$

$1000 = \phi 20(1225, 0.3333, 15)$

$0 = \phi 26(-900, 30, 30, 1, 30)$

(for the significance of *negative* addresses, see below)

#### Storage limitations

The size of the auxiliary store will naturally limit the size and number of matrices which can be handled. If the highest chapter number is  $N$ , then the auxiliary storage space available extends from location 0 to  $10,751 - 512N$ . Thus, if the programme contains chapters 0, 1, 2, then the last available auxiliary location is 9727.

Further variables are obtained by using the space occupied by the main Autocode programme, and the dumps. These correspond to *negatively* numbered locations:-

1) Input Programme gives	-1	to	-1536
2) Subchapter dump "	-1537	to	-2048
3) Main Chapter dump "	-2049	to	-2560
4) Secret dump "	-2561	to	-3072

The secret dump is listed here for reference purposes only, since it is already used as private working space by the matrix routines themselves. It may however be used as auxiliary storage when these are not employed, and when the *rmf* facility is not wanted. Note that overwriting the Autocode input programme also means dispensing with the *rmf* facility.

#### Notes

- Any duplication of group names on the left- and right-hand-sides of equations using  $\phi 16, \phi 26, \phi 27$  must be confined to groups of *not more than 160* elements, e.g.,

$$a' = \phi 26(a', b', u, u, u)$$

where  $u$  is not greater than 12.

- It is assumed in  $\phi 28$  that the elements of the  $B$  matrix are all represented to approximately the same absolute accuracy. The same applies to the  $A$  matrix in  $\phi 25$ .

In  $\phi 28$  the value of the determinant is computed as an incidental operation by building up the product of the pivots. Unfortunately this may lead to accumulator overflow unless both sides of the equations are suitably scaled. Thus, for example, if all the pivots are approximately  $10^6$ , this would lead to accumulator overflow after about 12 reductions.

Times of operations for square matrices of order  $n$

Size $n$ Function	2	10	20	40	60	80	100
$\phi$ 11 - 15	$\frac{1}{2}$	1	$2\frac{1}{2}$	7	15	25	38
$\phi$ 16	1	$1\frac{1}{2}$	$2\frac{1}{2}$	10	27	59	
$\phi$ 17 - 24	1	2	3	6	10	14	25
$\phi$ 25	1	4	14	70	209	459	
$\phi$ 26, 27	1	$2\frac{1}{2}$	16	105	345	695	
$\phi$ 28	1	9	40	220	686	1560	

Time in seconds



## Part 4

# The Programme Library

This part describes the arrangements for using routines consisting of one or more chapters as "sub-programmes" of a larger programme.

### Specification of parameters

The master programme has to provide the sub-programme with certain information before the latter can carry out its task. Thus it has to be told where to find the operands and where to place the results. In either case these may take the form of individual numbers and/or sets of numbers, e.g., series, matrices, etc. In addition the sub-programme may refer to other programmes (or smaller routines), as in the case of the quadrature programme where the "argument" is a function. These programme parameters are provided when "calling-in" the sub-programme. The "preserve" and "restore" instructions were introduced so that a subchapter could use the working store quite independently of the master chapter, and hence the names of the main variables (e.g.,  $a \rightarrow 19$ ) in the subchapter need not bear any relation to those in the master chapter. However, the special variables  $a, b, c, \dots, a', b', c', \dots$ , and the indices  $i, j, k, \dots$ , all refer to fixed locations in the working store and it is by means of these that the master chapter communicates with the subchapter. Thus after the "preserve" instruction in the main chapter these quantities can be reset to new values which are then available to the subchapter, which will be designed to select its information in accordance with the following conventions.

- (a) Individual arguments, field dimensions, etc., will be selected from

$a \ b \ c \ d \ e \ f \ g \ h \ u \ v \ w \ x \ y \ z$

$i \ j \ k \ l \ m \ n \ o \ p \ q \ r \ s \ t$

- (b) Fields of numbers (e.g., vectors, matrices) will be kept in the auxiliary store, and their locations (i.e., the location of 1st element or other reference position) specified by primed variables, i.e.,

$a' \ b' \ c' \ d' \ e' \ f' \ g' \ h' \ u' \ v' \ w' \ x' \ y' \ z'$

The primed variables will also specify where the sub-programme is to place its results, which must *all* be transferred to the auxiliary store before returning to the main programme. This applies to individual numbers as well as sets of numbers, for otherwise, if left in the working store, they would be destroyed by the subsequent "restore" instruction in the main programme. Instead they are recovered from the auxiliary store by means of  $\phi_e$  instructions after restoring the contents of the working store.

### Example

Consider a programme for the best solution of a set of linear "equations" in the sense of least squares. Let these be

$$\sum_{j=1}^n a_{ij} x_j = b_i \quad i = 1(1)m, m \geq n.$$

The programme will have to be supplied with the numerical values of  $m$  and  $n$ , and with means to refer to the coefficient matrix and the vector of the right hand sides. Thus, for instance, it may assume that  $a_{ij}$  stands in  $a' + (i-1)n + (j-1)$ , and  $b_i$  in  $b' + (i-1)$ ,  $a'$ ,  $b'$  being specified on entry. In addition the programme may require further variables as working space in which to set up the normal equations, and provision for this will have to be made in the main programme. Finally provision for the results  $x_i$  has to be made.

The specification of the programme parameters might therefore read as follows:-

$m, n$             field dimensions  
 $a', b'$           location of coefficient matrix and r.h.s. vector  
 $w'$             first of  $\frac{1}{2}n(n+1)$  consecutive auxiliary variables\*  
 $x'$             location of result vector ( $x_i$  will be placed in  $x' + (i-1)$ ).

Except insofar as the auxiliary working space is unnecessarily large, the following programme would meet this specification. Matrix operations are employed. Denoting the original "equations" by  $Ax = b$  the normal equations are  $A^T A x = A^T b$ .

$$\begin{aligned} u &= m \\ v &= n \\ y' &\approx w' + mn \\ w' &= \phi_{16}(a', u, v) & A^T \\ y' &= \phi_{26}(w', a', v, v, u) & A^T A \\ x' &= \phi_{26}(w', b', v, 1, u) & A^T b \\ x' &= \phi_{28}(y', n, 1) & (A^T A)^{-1} A^T b \\ \text{up} \end{aligned}$$

The programme is unsatisfactory for two reasons. Since there is no matrix instruction which will give the product  $A^T A$  directly (in this respect the list is inadequate) it is necessary to form  $A^T$  explicitly. This involves extra working space (for  $mn$  variables) and extra computing time. Secondly no advantage is taken of the fact that  $A^T A$  is symmetrical, so that  $\frac{1}{2}n(n-1)$  extra elements are computed and stored. Altogether, therefore, the auxiliary working space could be reduced from  $mn + n^2$  to  $\frac{1}{2}n(n+1)$  - and possibly further still - by programming the calculation directly, using only the  $\phi_6$  and  $\phi_7$  instructions to refer to the auxiliary store.

\* A desirable minimum. This assumes that only the distinct elements of the symmetric coefficient array are calculated, and that the r.h.s. can be stored in the space allocated for the result vector,  $x$ .

The following sequence illustrates, for a particular case, how such a programme would be called in from the main programme

---

```

preserve
m   =   70
n   =   50
a'  =   0
b'  =  3500
x'  =  3570
w'  =  3620
down ??           → enters subprogramme
restore          ← returns from subprogramme
 $\phi_6(3570)?, 50$ 

```

---

The last instruction recovers the results from the auxiliary store. The "down" instruction has been deliberately left incomplete as it serves to introduce the new assembly facilities.

#### Assembly of the programme

A new directive of the form "programme -N" enables a group of chapters constituting a single sub-programme to be given a number (N) which can be referred to in the "down" and "across" instructions, which have been extended to deal with this situation. To illustrate this facility a typical layout is shown below:

```

programme -1
  chapter 1
  chapter 2
  .
  .
  .
  chapter 5
programme -2
  chapter 1
  chapter 2
programme -550
  chapter 1
programme -584
  chapter 1
  chapter 2
  chapter 3
  chapter 0

```

Any programme can be assigned a number in the range 1 to 1023, and it is suggested that the range 501 upwards be reserved for library programmes. Thus in the above example the first two programmes would have been specially written for the problem, while the last two programmes would be drawn from the library.

If now it is required to call in programme -2 as a subprogramme, this is done by means of a modified "down" instruction, e.g.,

down 1/1 - 2

which refers to chapter 1 of programme 2. The "across" instructions can be used in a similar fashion.

The first directive (programme -1) is not necessary if these are "master" chapters and not referred to by the other subprogrammes (except implicitly via the "up" instruction).

The directive "variables *N*" now assumes a relative significance: it refers to a previous chapter in the same subprogramme. In Mercury the directive "variables 0" refers to the last chapter in the previous subprogramme. In the master chapters or in the final chapter 0 it refers to the previous chapter 0 in the machine, i.e., the chapter 0 of the last complete programme. It is possible to take advantage of this in certain situations but in this case the current programme must be read in by an 'rmp' instruction or a manual restart; otherwise the previous chapter 0 will be destroyed.

### Auxiliary routines

Consider for example a programme to evaluate a triple integral

$$\iiint f(x,y,z) \quad dx \, dy \, dz$$

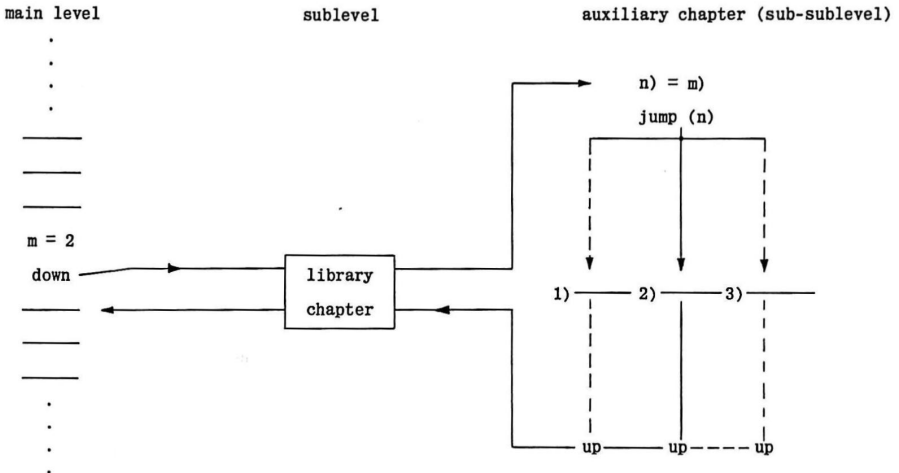
Depending on the actual process used, the integrand has to be calculated at certain specified points  $x, y, z$  so that if the programme is to be used for a "general" function this must be calculated by an auxiliary routine provided by the user. This should be designed so that, given the particular arguments  $x, y, z$  the routine calculates the corresponding function value and places it in a preassigned location. If there is a substantial amount of calculation involved, the auxiliary routine can be a separate chapter and be treated as a subprogramme of the library programme, that is a sub-subprogramme of the main programme. The programme parameters of the "function" programme would be the arguments  $x, y, z$  and the auxiliary location (say  $f'$ ) of the result  $f(x, y, z)$ .

Alternatively the function programme can be called in by an "across" instruction, in which case it must also terminate with an "across" instruction, returning control to a pre-assigned point in the library programme. This arrangement has the advantage that it confines the whole integration process to one 'level'. In both cases the library programme will assign a definite programme number to the auxiliary 'function' chapter because it is not (easily) possible to make this a programme parameter.

### By-pass parameters

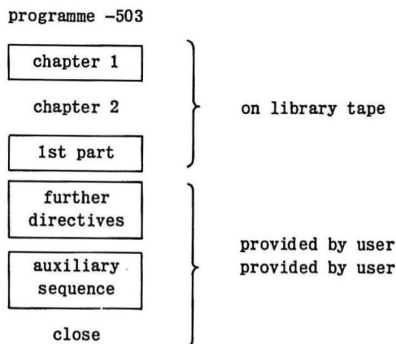
It is not likely that the library programme will make use of all the special variables and indices, in which case those that remain undisturbed can serve as "by-pass" programme parameters linking the main programme directly with the auxiliary function programme (whether treated as a "sub" or "across" programme of the library programme). This may be useful if the function involves certain parameters, e.g.,  $f(x,y,z; a,b,c)$  and it is desired to specify the values  $a,b,c$  in the main programme before starting the integration process.

The auxiliary chapter may also consist of several different function routines, any of which can be selected from the main programme by using a "by-pass" index to serve as a multiway switch in the function chapter, as illustrated by the following diagram.



#### Incomplete chapters

If the 'function' is of a fairly simple nature it is desirable to incorporate the auxiliary routine as an integral part of the library chapter in which it is used, and eliminate the operations of chapter changing and "preserve" and "restore". The quadrature programme -503, e.g., consists of two chapters, the first of which computes the coefficients of the quadrature formula, while the second carries out the integration proper. It is intended that the instructions for calculating the integrand shall be included in the second chapter, which is otherwise only about one-third full. The structure of the programme is therefore as follows:



In such a programme it is obviously necessary to know how much of the chapter space is available for the auxiliary sequence, what labels can be used, and so on. It is convenient to assume that it would be entered at the first instruction and terminated by returning control to a preassigned point in the first part of the chapter, e.g., label 127.

Once again communication with the auxiliary sequence is by means of the special variables and indices, but since there is now no "preserve" instruction these are limited to those which are not used for other purposes by the first part of the chapter. On the other hand however, the absence of the "restore" operation means that they can now be used for communication in *both directions*, so that e.g., a function value calculated by the auxiliary sequence can be recorded directly as a working variable or index. Those available special variables and indices which are not used to communicate between the two parts of the chapter can serve as "by-pass" parameters to link the main programme with the auxiliary sequence. As already explained this allows the quadrature programme to be used on different integrands, although the multiway switch device described in the previous section is relatively time consuming in the present context [ $n = m$ ] takes 18 millisecs to execute]. Instead the different function sequences can be called in by a series of conditional jump instructions, thus

```

jump 1, m = 1
jump 2, m = 2
jump 3, m = 3
etc.

```

This is particularly suitable if there are only 2 or 3 different functions involved.

It is necessary to say something about the directives which are written at the head of the auxiliary sequence. These are limited only in their *extent*, i.e., the total number of variables specified must not exceed the number left unspecified in the first part of the chapter. There is no limitation however in the *letters* which can be employed, and if necessary these may duplicate those used in the first part. Thus e.g., we may write:

```

chapter 1
a → 119          (0-119)
b → 119          (120-239)

1st part

a → 119          (240-359)
c → 119          (360-479)

2nd part

close

```

The duplicate "a" directive simply means that wherever (say)  $a_3$  occurs in the first part of the chapter it refers to the absolute location 3, whereas in the second part it refers to location 243. To summarise: in order to be able to write an auxiliary sequence for a library programme of the kind considered above, the specification must include the following information.

1. proportion of chapter available (or number of registers)
2. permissible range of labels
3. entry\* and exit
4. permissible working space
  - (i) number of main variables available
  - (ii) list of special variables and indices available (by-pass parameters)

\* Unless otherwise specified this would be the first instruction, i.e., at the head of the sequence.

5. programme parameters (i.e., where to find the arguments and where to place the results).

**Some examples of library programmes**

A selection of library programmes will be found in Appendix 5.





# Appendix I

## A service for the punching and execution of Autocode Programmes on the Manchester University Mercury Computer.

The majority of small programmes, and in some cases, longer programmes, will be handled by the operating staff both as regards running on the machine, and also punching, if the customer has no access to editing equipment.

For this purpose, each programme must be accompanied by a MEMO (Autocode Service Operating Record) giving rough details of the expected behaviour. Both sides of this MEMO should be completed if a manuscript is submitted for punching, but if prepared tapes are sent, the punching details may be omitted. A copy of the MEMO is included at the end of this appendix.

### Titles and Reference Numbers

It is recommended that all programmes should bear a title and for this purpose a special *title* directive has been introduced. The word *TITLE* is written at the head of the programme and causes the characters on the next line (the titling sequence) to be copied immediately on to the output punch. Titles extending over more than one line require a second title directive.

A reference number will be given to each programme submitted for punching, in addition to the customer's own title. The first programme received from Messrs. J. Smith will be called JS/1 and subsequent corrected versions JS/1/1, JS/1/2 etc. until a working (correct) version is reached, JS/1/5, say. Further production runs will then be called JS/1/5.1, JS/1/5.2 etc. The second distinct programme will be called JS/2 and so on.

### Punching notes

Enumerate all physically distinct tapes and give the last University reference.

### Operating notes

- (a) Estimated total running time

A rough estimate is sufficient and in many cases it is only necessary to calculate the output time at the speed of the punch.

- (b) Number of chapters and highest numbered auxiliary variable

This information is required in case one magnetic drum is out of action so that only programmes of restricted size can be run.

- (c) Should queries be suppressed? Are matrix operations used?

Yes or no required.

(d) **Tapes**

Give details of any physically distinct tapes and the order in which they are required.

(e) **Expected behaviour**

A few details concerning the rhythm of input, computing time and output are required. There is no point in giving estimated numerical values of any results because the machine is not directly connected to a printer. The results are output on tape which is often printed *off-line* after the computing service session is over.

Details of how the programme terminates should also be given, i.e. *end* or *mp* or whether the operator is required to stop the machine manually.

It is appropriate to distinguish between production runs with a tested programme and development runs of a new programme. It is recommended that development and production should not be combined in a new programme.

**Operator's comments**

(a) **The programme conforms to the expected behaviour**

If the programme conforms to the expected behaviour, it is allowed to run and the results are returned to the customer. If the time estimated is grossly inaccurate, it may be stopped by the operator although it appears to be behaving correctly. A programme which produces a form of repetitive output (as seen on tape) is also stopped, although the rhythm may be as expected.

It is desirable to include some checks in the programme, either by repetition of calculations or by other mathematical relations. For development runs, it is usual to request a run of a few minutes on the first trial, so that the programmer may ensure that the results are correct, before attempting production runs.

(b) **The programme does not conform to the expected behaviour**

In this case, a repeat run is done to obtain consistency and to eliminate the possibility of machine error, both for the satisfaction of the operator and the customer. No charge is made for this repeat but it will be indicated on the MEMO. If the customer does suspect the machine, a second repeat may be requested on another day. If this is again consistent, it rules out any possibility of machine error, and a charge is made, but if this differs from the previous runs, it is repeated until consistency occurs and a charge is made for only one run of the set.

Types of unexpected behaviour are:-

1. The programme may reach *end* too early.
2. The machine may indicate accumulator overflow
3. The machine may call for more data.
4. The programme may come to a dynamic stop and display a characteristic fault number.

**Autocode Faults**

Encountered during Input	Encountered during execution of programme
1. Chapter overflow	8. Chapter entry label not set.
2. Label set twice	32. Spurious character encountered during the <i>read</i> operation.
3. Label not set	33. Calls for sqrt of negative argument.
4. Instruction too large	34. Calls for exponent of large argument ( $\geq 177$ ).
5.       "       "       "	35. Calls for logarithm of negative argument.
6. Directives > 480	
7. Directives not set	
9. 1st instruction of chapter ( $\neq 0$ ) unlabelled.	
10. Incorrectly punched machine instruction.	13. Reference to a non-existent subprogramme (see Part 4).
11. Non-correspondence of cycles.	
12. Duplication of index counts.	

These programming blunders are tested for in the assembly programme. In the case of sqrt, exp and log the machine will jump to a sequence labelled 100) in the event of the argument being incorrect, if such a sequence exists. Otherwise the fault number will be displayed. In the case of fault (1), the machine also indicates the point at which overflow took place, by punching out the following 100 characters.

Fault (8) is strictly speaking an input fault but it is more convenient to test for this in the chapter changing sequence, during the execution of the programme.

If one of the above mistakes is made, the fault number will be indicated but the converse is not always true.

The machine may stop on input on some function other than a dynamic stop, or on a dynamic stop with a number displayed which is greater than 35. In both cases it is probably because a non-permissible form of instruction is encountered.

It is possible for some non-permissible instructions to be accepted by the input routine and to be converted into faulty programme. This may cause the machine to stop, jump out of control or result in an incorrect numerical answer. If the programme is entered successfully, however, it is a fairly simple matter to locate faults using the ? facility.

**Restarts**

It is recommended that any runs greater than 15 minutes in length should be programmed to restart.

In the event of a machine failure, it is possible for the operator either to re-enter the current chapter 0 (with new data) or to feed in a new chapter 0. Small programmes would be restarted from the beginning.

Any restarting facilities should be indicated on the MEMO.

### Notes on Checking Autocode Programmes

Check that:-

1. The chapter number and final *close* are set for each chapter.
2. The DIRECTIVES are  $\leq 480$ : N.B. starting from 0 .
3. No variable or index has been used on the R.H.S. until it has been set on the L.H.S.
4. The values of all indices lie in the range  $-512 \leq i \leq 511$
5. LABELS
  - 1) In the range 1 - 99 (or 1 - 127 in special cases).
  - 2) The first instruction of each chapter is labelled.
  - 3) Each one referred to is set.
6. The numbers following "ACROSS" instructions refer to LABEL/CHAPTER.
7. CYCLES
  - 1) Never more than 8 deep.
  - 2) Each " $i = p(q)r$ " should have an associated 'REPEAT' in the same chapter.
  - 3) Do not reset " $i$ " *INSIDE* the cycle governed by " $i$ ".
  - 4) Each cycle should terminate, i.e.  $r-p$  must be a positive multiple of  $q$ .
  - 5) " $p$ " and " $r$ " must not be negative integers, although they can be indices taking negative values.
8. Check throughout for SPELLING in all WORDS.
9. Check that " $\phi$ " appears in all FUNCTIONS.
10. BRACKETS
  - 1) Round arguments of functions, e.g.  $x = \phi \sqrt{y_1 y_2 - 1}$
  - 2) Round modified suffices, e.g.  $x_{(i+3)}$ , but NOT round the " $i$ " of  $x_i$ .  
N.B.  $x(3+i)$  is INCORRECT, the form is  $x(\text{INDEX} \pm \text{INTEGER})$ .
  - 3) Round letters used after WORDS, e.g.  $\text{READ}(Y): \text{JUMP}(i)$  but NOT in direct jumps, e.g.  $\text{JUMP } 3$ .
11. CONDITIONAL JUMPS, e.g.  $\text{jump } 3, \alpha > \beta$   
Check that
  - 1)  $\alpha$  and  $\beta$  are both variables (including a signed or unsigned numerical constant)
  - 2) " " " " " indices (including a signed or unsigned integer)
12. In the  $n = 3$  type of instruction, associated with  $\text{jump } (n)$ , do not use " $n$ " as numerical "3". It is only a label.
13. Check that the titling sequence is correct.  
e.g. TITLE CR LF  
and a second TITLE CR LF if two lines are required.
14. Check that the programme terminates with either END or RMP.

15. Check that the symbols  $\rightarrow$  CR LF appear at the end of each physically separate tape.
16. Leave at least 6" of blank tape at the beginning of each physically separate tape.
17. DATA TAPES

Check that all numbers terminate with either CR LF or SP SP *INCLUDING THE LAST ONE.*

**Supplementary checks if the Runge-Kutta routine is being used**

e.g., an instruction of the form INTSTEP (10)

Check that:

1. There exists a sequence labelled 10), terminating with 592,0 and that the 3 instructions appear in the same chapter.
2. The sequence calculates  $f_1, f_2, \dots, f_n$  and does *not* use  $f_1$  to  $f_n, g_1$  to  $g_n$  and  $h_1$  to  $h_n$  as working space.
3. Directives  $h, y, f$  and  $g$  have been set  $\rightarrow$  (numerical value of)  $n$ .
4.  $h$  (= step length),  $n$  (= no. of equations),  $x$  (initial value), and  $y_1$  to  $y_n$  (initial values) have been SET before INTSTEP is obeyed.

MANCHESTER UNIVERSITY COMPUTING MACHINE LABORATORY  
AUTOCODE SERVICE PROGRAMME RECORD

To be filled in by customer	For official use
Reference _____	Reference _____
Date posted _____	Date rec'd. _____
Programmer _____	Date returned _____
Checked by _____	Supervisor _____

Tick whichever applies below.

GENERAL INFORMATION

1. This is a new programme. ☐
2. This is a first production run of Programme No. \_\_\_\_\_ which is now working, please preserve for future use.
3. This is a production run of a previous programme (No. \_\_\_\_\_)
4. This is a further test of programme No. \_\_\_\_\_

Has programme to be altered?

- |                |   |  |                       |
|----------------|---|--|-----------------------|
| YES<br>because | { | (a) punching error   | <input type="radio"/> |
|                |   | (b) mistake in programme                                     | <input type="radio"/> |
|                |   | (c) modification to help locate mistake<br>(e.g. ? printing) | <input type="radio"/> |
|                |   | (d) modifications to improve programme                       | <input type="radio"/> |
| NO<br>but      | { | (e) mistake in existing data tape                            | <input type="radio"/> |
|                |   | (f) further tests needed with fresh data                     | <input type="radio"/> |
|                |   | (g) suspect machine error                                    | <input type="radio"/> |

5. The following programmes are now obsolete: please destroy/return.

PUNCHING INSTRUCTIONS

1. I am returning print-out of programme :      please modify as indicated. ☐
2. Punch new programme tape. ☐
3. Punch new data tape. ☐

Punched by \_\_\_\_\_ Date \_\_\_\_\_



## Appendix 2

### Notes for Programmers who wish to prepare their own tapes and/or run them on the machine personally.

#### Preparation of Tapes

The programme and data are presented to the machine as one or more lengths of perforated paper tape which are scanned by the photo-electric reader - the input unit of the machine. These tapes are prepared on a manual keyboard perforator, the keys of which correspond to the standard symbols listed in Part 1.

The material is punched in the conventional manner, namely from left to right and down the column. Each line must be followed by two special symbols *CR* (carriage return) and *LF* (line feed). Mistakes may be overpunched with the *erase* symbol *\**. There is also a *space* symbol *SP* which corresponds to a space one character wide. Numbers on the data tape which are written on the same line must be separated by at least two spaces.

If the programme is to be run from time to time with different sets of data, then it will be convenient to prepare the main programme and the data as physically distinct tapes. For the same reason a steering programme will usually be prepared as a separate tape. All such tapes should be terminated by *→ CR LF* (for reasons explained below) and must of course be presented in the correct logical order corresponding to the 'programme layout'.

#### Library programme tapes

Although library programmes could be presented to the machine as physically distinct tapes as described above, it is not a good idea to festoon the console of the machine with a large number of different tapes. In order to keep the number of physically distinct tapes to a minimum, all library programmes should be copied on to (and so form part of) the main programme tape. The library copies will be found to be headed with a *title* sequence

title

programme - n

which should not be confused with the directive

programme - n.

The latter is necessary to the assembly of the programme, while the former can be omitted from the final problem tape if desired. (For this purpose it is separated from the programme proper by a length of blank tape.)

#### Output tapes

The machine itself produces perforated tape (which can be printed on a teleprinter) when it reads a *title* sequence or when it executes a *print* (or *?*) instruction or a *620,n* instruction. A tape produced exclusively by using *print* (or *?*) instructions can be subsequently used as a data tape.



## Operation of the machine

The Autocode programme is a binary tape and is put into the machine by means of *Teleinput*. It occupies sectors 0 to 31 and 80 to 127 inclusive, which should then be isolated. The latter group corresponds to the *negative* auxiliary locations -1, -2, ....., -1536, and can be used as such if necessary, although this means dispensing with the *rmp* facility. A separate tape is provided for the matrix operations (other than  $\phi 8$ ,  $\phi 9$ ,  $\phi 10$  which are on the main Autocode tape) - which overwrites sectors 480 to 511 inclusive. These are normally occupied by the Engineers' Test routines and for this reason the maintenance engineer should be informed that they have been overwritten.

As already explained it is a convention that all physically distinct tapes (programme or data) should be terminated with  $\rightarrow CR LF$ . Should the machine attempt to scan this sequence, the loudspeaker will give one of two characteristic signals, depending on whether the scanning instruction was a *read* (a rapidly varying note) or an *rmp* (slowly varying note). In either case it signifies to the operator that the machine is calling for a new tape. After reloading the reader the machine can be made to continue by pressing handswitch 9. When splicing two tapes together the sequence should be omitted, otherwise the operator will have to stand by to surmount it manually.

There is also a *halt* instruction which stops the programme and gives a medium speed intermittent note on the loudspeaker. This is surmounted in the same way by pressing handswitch 9.

A facility exists for including or eliminating ?-prints from the translated programme depending on the setting of handswitch 4 during the translation process. This enables one to proceed at once to "production" by eliminating the "development" printing without having to reperfurate the tape.

Detailed operating instructions are given below:

### I. To input the AUTOCODE LIBRARY TAPE by TELEINPUT

1. All block-isolation switches DOWN (except 7 on DRUM 1).
2. Autocode Library tape in the reader.
3. Key 2 (ONLY) of bottom row of handswitches UP.
4. ITB.
5. Switch on to CONTINUOUS.

The tape is read in, terminating with a continuous hoot.

6. Switch off to SINGLE.
7. ISOLATE switches 0 and 3 on DRUM 0.

### IF MATRIX OPERATIONS USED

1. Put switch 7 on DRUM 1 DOWN.
2. MATRIX TAPE in reader. Proceed as above to (6).
3. ISOLATE switch 7 on DRUM 1.

### II. To input an AUTOCODE PROGRAMME TAPE

This is the usual procedure when putting a fresh programme into the machine, and has the effect of resetting all the stores to a standard state. This is essential should it prove necessary to repeat the run for a consistency test.

1. Programme tape in reader.
2. Set the bottom row of handswitches:
  - (a) ALL ZERO - normal input, or
  - (b) KEY 4 (ONLY) UP - if query printing is required.
3. ITB.
4. Switch on to CONTINUOUS.

The programme is now translated and entered on reading a starting chapter 0.

#### IIIA. Faults encountered during INPUT

1. The machine may come to a 99 stop (PF = all zeros).  
CAUSE: NO matrix tape in machine OR incorrect form of instruction.
2. The machine may stop and record ACCUMULATOR OVERFLOW (the 2 most significant digits of YA are different).  
CAUSE: Incorrect form of instruction.
3. The parity light may come ON with PF = 0010100 ... but NO drum selection light ON.  
CAUSE: Incorrect form of instruction OR ERROR on tape.
4. The machine may come to a LOOP STOP (PF = 0000001000) *IN THIS CASE, LOOK IN B7* which will give the FAULT NUMBER (see Appendix 1).  
Certain versions of AUTOCODE do not stop on encountering an input fault, but continue to translate and list all the faults discovered during the input attempt.

#### IIIB. Faults encountered during OPERATION

1. The machine may come to a loop stop (as Note 4 in IIIA). LOOK IN B7 for FAULT NUMBERS 8, 32, 33 to 35. Fault 32 can be surmounted after correcting the tape, by resetting control to 15.0.
2. The machine may record ACCUMULATOR OVERFLOW. The numbers have exceeded capacity, i.e.  $> 10^{77}$ .

#### IV. Action required when the hooter sounds

1. HALT - medium intermittent hoot. Depress KEY 9 to surmount.
2. HOOT - one single hoot. No action necessary.
3. END - continuous hoot. Final and insurmountable.
4. The warning sequence → CR LF (normally at end of tape).  
Slow intermittent hoot - calling for more programme.  
Fast intermittent hoot - calling for more DATA.  
Depress KEY 9 to surmount in both cases.

#### V. Methods of RESTARTING

1. To re-enter the current chapter 0
  - (a) Put up KEYS 9 and 1 ONLY (of the bottom row of handswitches).
  - (b) ITB.
  - (c) Switch on.

The current chapter 0 will be re-entered at the first instruction.

2. To read in a new chapter 0 (or an entirely new programme) from tape.

In this case the number stores are not reset to a standard state, and the modified (or new) programme will find them in the state in which they were left by the previous calculation.

(a) Put up KEY 0 ONLY (of the bottom row of handswitches).

(b) ITB.

(c) Switch on.

The machine will read in the tape.

N.B. In both cases the INDICES are destroyed.

To preserve the indices do *NOT* press I.T.B. as in (b), instead CLEAR CONTROL as follows:-

- b1) Set MAN/AUTO switch to MAN,
- b2) Put KEY 3 up on TOP row of handswitches,
- b3) Clear MIDDLE row of handswitches,
- b4) Press PREPULSE button,
- b5) Return MAN/AUTO switch to AUTO.

## Appendix 3

### Interpretation of Machine Orders.

Facilities are provided for writing part of an Autocode programme in conventional machine instructions\* in order to speed up inner loops, etc. The instructions must be written in the form

function and b-digits and address digits

and may be written with the address either in the conventional form or in the symbolic form described below. The function and b-digits are in both cases written in the conventional way. These instructions may be labelled in the same way as Autocode instructions.

**Conventional addresses.** Conventional addresses are written in terms of medium length register numbers with a *comma* between the functional part and the address part, e.g.

200, 15.10+

407, 2

The usual facilities for writing the address in any consistent page-and-line form and for the use of the symbols  $\neq$  with instructions of the n,b type still apply.

Only fixed addresses are allowed with this facility: no forms of relative addresses are possible.

Care must be taken when using conventional addresses that no vital part of Autocode is overwritten; their use is therefore deprecated apart from the possible use of a few functions such as 620 etc., since all the facilities can be obtained by means of the symbolic form.

**Symbolic addresses.** In the symbolic form the address is written in brackets and is interpreted as follows:

<i>Instruction</i>	<i>Interpretation of the address</i>
(i) 400 (x)	address of x
(ii) 200 (i)	address of i
(iii) 590 (3)	address of the instruction labelled 3.

The first example applies to any of the accumulator codes 40 to 45 and 50 to 55. The quantity in the brackets can be a variable such as  $x_1$ ,  $x_3$ ,  $x_4$ ,  $x_{(i-3)}$  or any signed or unsigned numerical constant. In the case of  $x_1$ ,  $x_{(i-3)}$ , the b-digit must always be a zero.

The second example applies to any of the machine codes 00 to 07, 20 to 27. In this case the quantity in brackets can be an index or a signed or unsigned integer.

The third example applies to all jump instructions.

\* See Programmers' Handbook for the Ferranti Mercury Computer. List C.S.225.

Machine codes of the type n,B are not allowed in symbolic address form, thus 300(3) should be written in the form 300,3 etc.

As an illustration of the way time can be saved by the use of conventional machine orders in an inner loop, consider the following example. The problem is the summation of the polynomial

$$y = a_0 + a_1 x + \dots + a_{10} x^{10}$$

and the AUTOCODE programme is as follows:-

```

u = a10
r = 9(-1)0
u = ux + ar
repeat

```

The number of instructions can be almost halved by writing them directly in the symbolic form described above. Thus:-

```

300, 10 sets B7
407 (a0) transfers a10 to the accumulator
1) 330, 1 reduces B7
    500 (x) } forms ux + ar
    427 (a0) }
    280 (1) tests B7
    410 (u) plants u

```

This could be shortened still further if the polynomial were arranged in the form:-

$$a_0 x^{10} + a_1 x^9 + \dots + a_{10}$$

for then the cycle could be re-written

```

400 (a0)
300, -9
1) 500 (x)
    427 (a10)
    380 (1)
    410 (u)

```

The times for the three methods are respectively 185, 105 and 95 micromins. This example is one of the worst possible, but serves to stress that Autocode is generally much less than a factor of two slower than the conventional method.

#### Interpretation on Atlas and Orion

In general it will not be possible to interpret Autocode programmes involving Mercury machine instructions correctly on the Atlas and Orion computers. As already explained, however, special provision will be made in the case of the "592,0" instruction, the "620" instruction, which is used for alpha-numeric output, and the instructions needed for generating pseudo-random numbers.

## COMPUTING STORE SPACE

Page	Register	Use
0	L0 to L2	Not available
	M4 to M31	Division Subroutine
	L32 to L38	Workspace (may be used in machine orders)
	L40 to L56	Not available
	H58 to H63+	Indices in order (i,j,k, ... t)
1	M1.0 to M1.63	Used for auxiliary transfers and complex square root
2 to 14	M2.0 to M14.63	Programme
15	M15.0 to M16.63	Buffer store to read functions into
16 to 30	L16.0 to L30.62	Main variables (see Part 1, p.12)
31	L31.0 to L31.26	Special variables a', b', c' ... z'
	L31.28 to L31.54	Special variables a, b, c, ... z
	L31.56	Special variable $\pi$
	L31.58	$0 \times 2^{-256}$
	L31.60	$-1 \times 2^0$
	L31.62	$-0.75 \times 2^{29}$

## Drum Space

Drum	Column	Sectors	Use	Auxiliary Variable
0	0	0 to 31	Autocode Programme	Not available
	1	32 to 47	Secret Dump	-3072 to -2561 (4)
		48 to 63	Main Chapter Dump	-2560 to -2049 (3)
	2	64 to 79	Sub Chapter Dump	-2048 to -1537 (2)
		80 to 95	Autocode Programme	-1536 to -1 (1)
	3	96 to 127		
	4	128 to 159		0 to 1023
	5	160 to 191		1024 to 2047
1	6	192 to 223		2048 to 3071
	7	224 to 255		3072 to 4095
	0	256 to 287		4096 to 5119
	1	288 to 319		5120 to 6143
	2	320 to 351		6144 to 7167
	3	352 to 383		7168 to 8191
	4	384 to 415	Chapter 5	8192 to 8703 (5)
			Chapter 4	8704 to 9215 (5)
	5	416 to 447	Chapter 3	9216 to 9727 (5)
			Chapter 2	9728 to 10239 (5)
	6	448 to 479	Chapter 1	10240 to 10751 (5)
			Chapter 0	Not available
	7	480 to 511	Matrix Programmes	Not available

## Notes:

- (1) Can be used if "rmp" facility dispensed with
- (2) Can be used if no "preserve" in a sub chapter
- (3) Can be used if no "preserve" in a Main chapter
- (4) Can be used if "rmp" facility dispensed with and if no matrix routines used
- (5) If the highest chapter number is N then the last auxiliary variable which may be used is 10751 - 512N

## Appendix 4

### Facilities that are available only on the Manchester and I.C.I. machines.

#### Use of short integers in machine instructions

In addition to the ability to use both conventional and symbolic instructions, the Autocode programmer may introduce *short integers* and *long numbers*. These numbers can be written in the usual way. However, it is not permitted to place a decimal point in a short integer - which means that the page and line form cannot be used but must be converted to line form. There is also a slight restriction in that these numbers must be listed at the head of the chapter, i.e. between the variable directives and the first instruction. The conventional addresses of the instructions in a chapter starting at the first, are 2.0, 2.1, ..... etc. To refer to the first number in the list of numbers at the head of the chapter, the conventional address 2.0 must, therefore, be used. If this first number is a long number (which occupies two locations) then the address of the second number will be 2.2.

The short integers must each be preceded by one of the symbols  $\neq$  or  $>$ . They must be preceded by a minus sign where negative, but no positive signs are allowed. The  $+$  sign may be used with the  $>$  symbol when it will be interpreted as  $\frac{1}{2}$ . In the case of positive integers this will be equivalent to the next half register, i.e. the right-hand half register, while for negative (non-zero) integers, it will give the preceding half register (right-hand half register). Thus the following are equivalent:

$$= -3 \quad > -1+ \quad \neq -6 \quad = 1021$$

as are the following:

$$= 1 \quad > 0+ \quad \neq 2$$

Note: It is not permitted to write  $> -0+$ .

Long numbers must be punched in fixed-point style and should be preceded by either a  $+$  or a  $-$  sign, otherwise the number may be punched with the same freedom as constants occurring in instructions.

The following example should illustrate all the facilities and, as usual, the first instruction in each chapter (other than chapter 0) must be labelled.

```

chapter 1
a → 10
b → 10
x → 400
= 1, ≠ 2
> 3, > -3+
= -1, ≠ -2
= 1023, ≠ 2046
> 0+, = 1
+ 3.1415926535
- 1
+ 0.5
- 10
= 1, = 2
= 3, = 4
+10
1) read (x)
n = 3
.
.
.
.
close

```

Short integers.

Since the long number must go into an even line an empty line will be inserted before the long number in the translated programme and allowance should be made for this in referring to subsequent numbers in the list.

More short integers.

In this case the long number will go into the next register as this will be an even register.

The only way to ensure that the first instruction goes into an even register is to precede it by a long number. Although very rarely will it be necessary for the instruction to be so placed.

### Generation of pseudo-random numbers

The autocode instruction

$$y = \phi \text{ random } (x, n)$$

where  $n$  is any index, can perform 3 different operations depending on whether the chosen index  $n$  is set to 0, 1, or 2. These three integers can be substituted directly in place of  $n$  in the instruction, if desired, and in fact, this will be the more usual procedure.

### Starting a sequence of pseudo-random numbers

Each term in a sequence of pseudo-random numbers is dependent on the previous term, and to initiate a satisfactory sequence the first term must conform to certain rules and the binary form of this number will for convenience have to be prepared in a non-standardised form.

The instruction to load  $y$  with a suitable starting value can have either of the two forms

$$y = \phi \text{ random } (x, 0)$$

$$\text{or } y = \phi \text{ random } (x, n) \text{ where } n \text{ contains } 0,$$



in both cases  $y$  can be any variable,  $x$  can be any variable or an integer, and  $n$  can be any index. The value chosen for  $x$  or its substitute integer can have any odd value in the range 1 to  $10^6$ . It is, therefore, possible to initiate many different random number sequences.

#### Generation of a sequence of rectangularly distributed pseudo-random numbers

Having set a starting value in a location, the 'residue' method may be used to obtain a sequence of random numbers.

The instruction

$$y = \phi \text{ random } (x, n)$$

where  $x$  and  $y$  are any variables, and  $n$  has the value 1 or is replaced by the integer 1, uses this method to place in  $y$  the next number in that sequence obtained by operation on  $x$ . It is, therefore, sensible that  $y$  be the same as  $x$  ( $x$  can only be a variable), for otherwise the sequence will not advance. The variable  $y$  will be rectangularly distributed in the open interval (0, 1).

To obtain a random integer in the range (0,100) the pair of instructions

$$x = \phi \text{ random } (x, 1)$$

$$i = \phi \text{ int pt } (101 x)$$

could be used.

#### Generation of a sequence of normally distributed pseudo-random deviates

An approximately normally distributed pseudo-random deviate can be obtained by the addition of a number of rectangular numbers, and the mechanism of the *residue class* generating procedure already described is used for this purpose.

The instruction to maintain a normally distributed sequence has the form

$$y = \phi \text{ random } (x, 2)$$

$$\text{or } y = \phi \text{ random } (x, n) \text{ where } n \text{ contains } 2,$$

in both cases  $y$  can be any variable,  $x$  can be any other variable and  $n$  can be any index. The contents of variable  $y$  will approximate to a normal random deviate with zero mean and unit standard deviation. The distribution will lie in the open interval (-6, 6). The sequence generated in  $y$  is dependent on a sequence of rectangularly distributed variates which are automatically generated in  $x$ . Twelve values in the sequence of  $x$  are used to generate one term in  $y$  and after obeying this instruction the twelfth term in the  $x$  sequence is retained in  $x$  ready for further use.

#### Examples

Load the locations  $a_1, \dots, a_n$  with (i) rectangular variates and (ii) with normal variates.

```

(i)      x =  $\phi$  random (1, 0)
          i = 1 (1) n
          → x =  $\phi$  random (x, 1)
           ai = x
          ← repeat

(ii)     x =  $\phi$  random (1, 0)
          i = 1 (1) n
          → ai =  $\phi$  random (x, 2)
          ← repeat

```

### Time and space

The times taken for the various operations are

(0) 15  $\mu$ min.      (1) 20  $\mu$ min.      (2) 200  $\mu$ min.

plus an additional time of 300  $\mu$ min. access. This access time may be eliminated by incorporating  $\phi$  random as a quicky by writing

$\phi$  random

at the end of the chapter in the usual manner. The space required for the instruction is 7 registers in all cases.

### Sunvic Logger

With the advent of automatic devices for the production of punched paper tape, such as card-to-tape converters, analogue-to-digital devices and in particular one such device produced by Sunvic Controls Limited, it has been found desirable to be able to process tapes which, by virtue of their length, may contain several punching errors. It is, therefore, necessary to be able to distinguish between numbers correctly punched and others which may have characters inserted or omitted or incorrectly punched (thereby becoming spurious characters). Decimal characters may be inserted or omitted and it is obviously necessary to be able to detect these. Many of these devices assign a fixed number of decimal places to each item of information and it is intended to make use of this uniformity of output. It is with these points in mind that the following Autocode instruction has been prepared.

The instruction

sunvic logger (x, n, 3)

where x is any one variable, n any index, and 3 any label in the same chapter, has the effect: set x equal to the next number on the data tape, n equal to the number of decimal digits in that number and then proceed with the next instruction. However, should any spurious characters be encountered the machine will jump to the sequence labelled 3 with x and n set as above.

The decimal digits referred to above are the ten characters, 0, 1, ..., 9. The spurious characters referred to above are

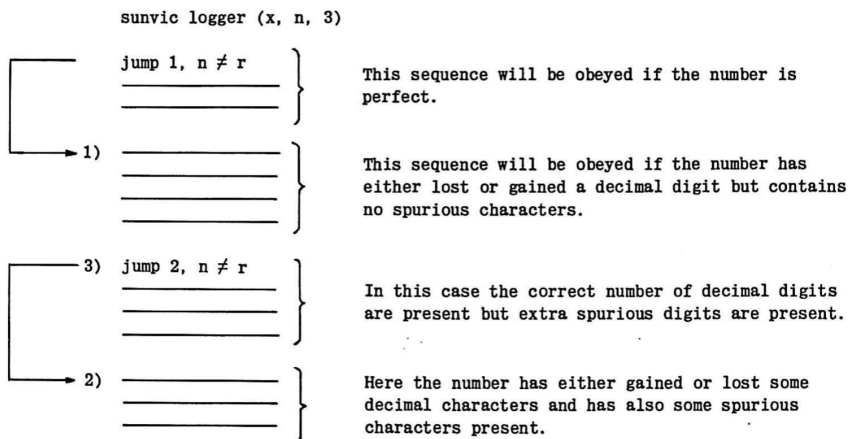
\* ( ) ≠ ≈ , > ≥ /  $\phi$  + LS ' .

The characters FS LF and SP are ignored outside a number, while inside a number they will be regarded as spurious. The ERASE symbol is ignored everywhere. A double space or a CR will terminate a *number* - a number being any sequence of characters, legitimate or spurious, but not any of those which are ignored. The only remaining symbol is the minus sign, which will have its usual significance but it will also be taken to be the start of the number and hence any preceding decimal digits will be ignored completely. Outside a number, the  $\rightarrow$  causes the machine to halt in the usual  $\rightarrow$  hoot loop, and this can then be surmounted, but inside a number it will be treated as spurious.

The restrictions given above mean that blank tape and line feeds are ignored before numbers, but that a group of spurious characters suitably terminated will be regarded as a number, however, in this case the machine will jump to label 3.

There remains the case where decimal digits have been mispunched as ERASE or a minus sign. This occurrence can be detected as one should know how many decimal digits to expect in each number, and it is then a simple matter to detect this, as the conversion of a decimal digit into a minus or an ERASE will appear to the machine as if that decimal digit is missing altogether. This is best done by following the *sunvic logger* instruction with a conditional *jump*.

Thus, suppose one expects to read a number with  $r$  decimal digits, the following sequence could be used if it is desired to attempt a diagnosis of the fault, should any occur.



#### Caption

The instruction *caption* may be written anywhere in a chapter, and when obeyed will effect the output of all the characters presented on the next line of the programme. Any of the combinations in the tape code may be used\* and spaces between characters, or from the start of the line to the first character, will be accurately reproduced on the output tape.

\* Erase is ignored completely.

For example the sequence

```

newline
i = 1(2)5
└─┐ caption
   │ temperature
   │ print (i) 1, 0
   └─┐ repeat

```

will produce

temperature 1                      temperature 3                      temperature 5

Thus the carriage return at the end of the line containing "temperature" is not included in the output.

Matrix operations  $\phi_{29}$ ,  $\phi_{30}$ ,  $\phi_{31}$

The existing matrix operations  $\phi_8$  to  $\phi_{28}$  have been supplemented by the following facilities.

$a' = \phi_{29}(u)$	A becomes the unit matrix I of order u where $2 \leq u \leq 129$
$a' = \phi_{30}(u)$	A becomes the null matrix 0 of order u where $2 \leq u \leq 129$
$d' = \phi_{31}(a', u)$	Extract D, the diagonal of A, and store it as a vector of length u.

#### Further Machine Instructions

In addition to the basic instruction codes given in the Ferranti Handbook CS.225, the (decimal) codes 78, 90 to 97, 11 and 31 described in CS.188A (Sept. 1960) are also available. However the code 91 is translated into the octal code 123 *not* 137.

At the time of writing only the Manchester University and I.C.I. machines are equipped to interpret the octal code 123.

# Appendix 5

## A selection of library programmes available for Mercury.

The following pages describe some library programmes currently available. A complete list will be supplied by the installation at which the programme is to be run.

### ESTIMATION OF THE LIMIT OF A NUMERICAL SEQUENCE: PROGRAMME - 501

Given the first  $(n+1)$  terms of a numerical sequence

$$a_0, a_1, a_2, \dots, a_n,$$

the programme estimates the limit,  $n \rightarrow \infty$ , according to one of the following methods.

1. By "meaning", i.e., by forming the derived sequences

$$a_r^{(s)} = \frac{1}{2} (a_r^{(s-1)} + a_{r-1}^{(s-1)}), \quad a_r^{(0)} = a_r,$$

which under suitable conditions tend more rapidly to a limit. This will be the case if  $\sum_r (a_r - a_{r-1})$  is an *alternating series*, and  $f(r) = (-)^r (a_r - a_{r-1})$  is analytic in the half plane  $\text{Re}(r) > 0$ .

2. Assumes that the  $n$ th term is of the form

$$A_0 + A_1 \lambda_1^n + A_2 \lambda_2^n + \dots + A_m \lambda_m^n, \quad |\lambda_r| < 1.$$

$A_0$  is then the limit in question. For details of the method see Wynn, M.T.A.C., April 1956 (p.91).

3. Assumes that the  $n$ th term is of the form

$$\frac{A_0 + A_1 n + A_2 n^2 + \dots + A_m n^m}{B_0 + B_1 n + B_2 n^2 + \dots + B_m n^m}.$$

The limit  $A_m/B_m$  is selected from the even order differences in a table of Thiele's reciprocal differences of the sequence  $a_n$ .

4. Assumes that the  $n$ th term is of the form

$$A_0 + \frac{A_1}{n} + \frac{A_2}{n^2} + \frac{A_3}{n^3} + \dots$$

The limit  $A_0$  is formed by polynomial extrapolation based on the data  $f(x_r) = a_r$ , where  $x_r = 1/r$ . (See Salzer, J. of Maths. & Phys., Vol. 33, p.356 (1954).)

In all four methods the data is introduced in the order  $a_n, a_{n-1}, a_{n-2}$  etc., proceeding as far back as a term  $a_{n-m}$  such that the estimate  $L^{(m)}$  based on these values differs

from  $L^{(m-1)}$  by less than a preassigned limit  $e$ , i.e.,  $|L^{(m)} - L^{(m-1)}| \leq e$ ; or until the introduction of further terms fails to improve the estimate; or until  $m = n$ . In all cases the programme provides (1) the final estimate, (2) the accuracy obtained, i.e.,  $|L^{(m)} - L^{(m-1)}|$ , and (3) the quantity  $m$  ( $m$  may be zero or even  $-1$ ).

The programme consists of a single chapter called in by "down 7/1 - 501", the entry label corresponding to the method selected, i.e., for method 3 use "down 3/1 - 501", and similarly for 1, 2, and 4.

#### Programme Parameters

n	Extent of sequence ( $n \leq 470$ ).
e	(absolute) accuracy required.
a'	sequence starts in $a'$ , $a' + 1$ , ... , $a' + n$ .
b'	results (1), (2), (3) are placed in $b'$ , $b' + 1$ , $b' + 2$ , respectively.

Time:  $(370 + km(m-1)/2)$  ms, where  $k = 1.5, 5, 6, 6$  for methods 1, 2, 3, 4 respectively.

#### SIMPLE QUADRATURE: PROGRAMME - 502

This programme evaluates the integral  $\int_a^b f(x)dx$ , where  $f(x)$  is assumed to be free of singularities in the immediate neighbourhood of the real interval  $(a, b)$ .

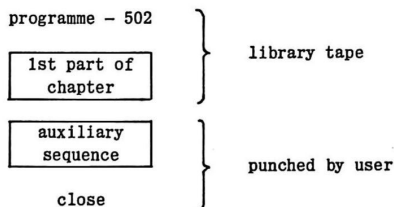
The method used is that of Gauss, employed over suitable subdivisions of the interval. Integration is first attempted over the whole range by comparing the results of a 5-point and 6-point formula. If these agree to a preassigned accuracy the integration is regarded as successful. Otherwise the interval is halved and the same process applied to each half. If either of these proves too large it is subdivided still further, and so on.

The method of comparison ensures a fixed absolute accuracy in every contribution, the criterion being  $|I_6 - I_5| < e$ , where  $I_5$  and  $I_6$  denote the two estimates and  $e$  is a programme parameter.

It may happen that this accuracy cannot be achieved however far the subdivision process is carried, e.g., because the integrand cannot be evaluated to sufficient accuracy. In this case the process is terminated immediately it fails to show any improvement, and the discrimination parameter  $e$  is replaced by the best value  $|I_5 - I_6|$  actually obtained. The final accuracy is recorded along with the answer in the auxiliary store.

#### Structure of Programme

The programme consists of a single chapter in two parts. The first part generates the quadrature formula and carries out the integration proper. The second part is provided by the user and is the auxiliary routine for calculating the integrand. The layout of the programme tape is as follows.



The auxiliary sequence will of course include its own directives.

#### Programme parameters

a } b }	limits of integration
e	limit of accuracy
u'	auxiliary location of result: final accuracy placed in u' + 1

#### Specification of auxiliary sequence

- chapter space available: 539 registers.
- labels 3 to 99 available.
- conventional entry: exit by jumping to label 101).
- permissible working space:
  - 440 main variables available,
  - by-pass parameters: all special variables and indices other than a, b, e, u' and i, j, k, l, m.
- programme parameter: x. Auxiliary sequence must replace x by f(x).

#### How to call in the library programme

This is done by "down 1/1 - 502". An alternative entry "down 2/1 - 502" provides a printed record of the integration process on the following lines. The subintervals over which the integration process was successful are listed together with the contributions from these intervals. Thus, e.g.,

0.00	-	0.25	0.99326	
0.25	-	0.50	0.00669	(this is the case of $\int_0^{2.0} e^{-x} dx$ )
0.50	-	1.00	0.00005	

means that the 5/6 point formulas are appropriate for the subintervals

$$\left(a, a + \frac{b-a}{4}\right), \left(a + \frac{b-a}{4}, a + \frac{b-a}{2}\right) \text{ and } \left(a + \frac{b-a}{2}, b\right).$$

It is realised that the user is usually more interested in calculating a set of integrals, depending on a parameter, rather than one particular integral. It is suggested that one or two members of the set be investigated with the aid of the above post-mortem facility, with a view to writing a "tailor made" programme for the whole set. This would be more economical in machine time.

Time: depends on the nature of the integrand, e.g.,  $\int_0^{20} e^{-x} dx$  to 4D takes 2.2 sec.

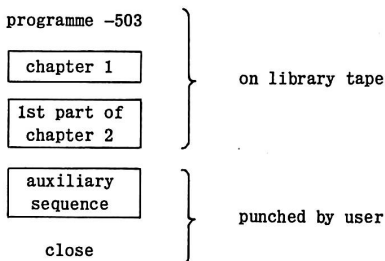
#### QUADRATURE PROGRAMME - 503

This programme calculates  $\int_a^b (b-x)^u (x-a)^v f(x) dx$ , where  $u > -1$ ,  $v > -1$  and  $f(x)$  has no singularities in the immediate neighbourhood of the real interval  $(a,b)$ .

#### Structure of the programme

The programme is similar to 502 but contains an additional chapter to calculate the quadrature formulas involved. These are the 5 and 6 point Gaussian formulas corresponding to the weight functions (1)  $(1-x)^u (1+x)^v$  in the range  $(-1,1)$ , (2)  $(1+x)^v$  in the range  $(-1,1)$ , and (3)  $(1-x)^u$  in the range  $(-1,1)$ . A printed record of the weights and zeros can be obtained by using the alternative "post mortem" entry. They will be found listed in a self-explanatory fashion prior to the analysis of the subdivision process which is similar to that of programme -502. The programme calculates the weights and zeros to six significant figures only, so that this places an upper limit to the accuracy.

Chapter 2 performs the same function as in programme -502. The programme tape is made up as follows.



#### Programme parameters

a } b }	limits of integration
u } v }	order of singularities
e	limit of accuracy (should not exceed six significant figures)
u'	auxiliary location of result: final accuracy placed in $u' + 1$



**Specification of auxiliary sequence**

1. chapter space available: 423 registers.
2. labels 3 to 99 available.
3. conventional entry: exit by jumping to label 101).
4. permissible working space:
  - (i) 320 main variables available,
  - (ii) by-pass parameters: all special variables and indices other than a, b, u, v, e, u'; i, j, k, l, m, n, o.
5. programme parameter: x. Auxiliary sequence must replace x by f(x).

**Calling in the programme**

If the answer only is required then enter with "down 1/1 - 503". If the weights and zeros of the quadrature formula and an analysis of the subdivision process are required, then the alternative entry, "down 2/1 - 503" is used.

Time: depends on the nature of the integrand, e.g.,

$$\int_0^{16} x^{\frac{1}{2}} e^{-x} \left(1 + \frac{x}{4}\right)^{\frac{1}{2}} dx$$

to 4D, takes 2.8 secs.

**QUADRATURE FOR INFINITE INTEGRALS: PROGRAMME -504**

This programme evaluates

$$\int_c^{\infty} f(x) dx,$$

where  $f(x)$  has no singularities in the immediate neighbourhood of the real interval  $(c, \infty)$ .

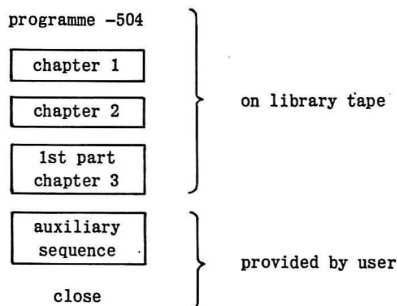
In this programme the user specifies a step length  $h$  and the programme forms a sequence by accumulating the contributions from each step. After the first six steps it tries to find a limit of this sequence by choosing one of the methods which are described in programme -501. The choice of the method is left to the user, and is specified by setting  $p = 1, 2, 3$  or  $4$  which correspond to the four entry points in programme -501. Thus, e.g., if the integrand contains a factor  $\cos kx$ , it would be appropriate to use method (1) with  $h = \pi/k$ .

**Structure of Library programme**

The library programme consists of three chapters. Chapter 2 is identical to programme -501, while chapter 3 corresponds to programme -502, except that in each case the final "up" instruction has been replaced by "across 7/1" and "across 6/1" respectively, in order to keep the programme on "one level".

The specification for the auxiliary sequence in chapter 3 is identical to that in programme -502, with the exception of certain by-pass parameters. The user who wishes to employ a limit process other than those used in programme -501 may do so by substituting an appropriate chapter 2, drawn up in accordance with the specification of programme -501 - except, as already mentioned, that the "up" instruction is replaced by "across 7/1".

The programme tape is made up as follows



#### Programme parameters

c	lower limit of integral
h	step length
p	specifies entry point for limit process
e	accuracy required
c'	sequence obtained from successive steps is placed in c', c'+1, c'+2, etc., as far as is necessary for convergence.
b'	results of sequence summation (see programme -501)
u'	answer and accuracy in u' and u'+1

#### Specification of auxiliary sequence

- chapter space available: 539 registers.
- labels 3 to 99 available.
- conventional entry: exit by jumping to label 101).
- permissible working space:
  - 440 main variables available,
  - by-pass parameters: all special variables and indices other than a, b, c, h, e, a', b', c', d', u'; i, j, k, l, m, n, p.
- programme parameter: x. Auxiliary sequence must replace x by f(x).

## Calling in the library programme

If the answer only is required use "down 1/1 - 504". The entry "down 2/1 - 504" will cause the machine to print out for each step, (1) the number of the step, (2) the breakdown of each step into its successful integration units (see programme -502), and (3) after the first six steps it will print out the best estimate obtained from the limit process, together with an estimate of the accuracy of the result.

Time: depends on the nature of the integrand, e.g.,

$$\int_0^{\infty} \frac{\cos x}{(1+x^2)^2} dx$$

to 4D, takes 7 sec.

## HARMONIC ANALYSIS: PROGRAMME -505

Given numerical values of a periodic function  $f(x) = f(x+L)$  at  $2n+1$  equally spaced points  $x_r = x_0 + (rL/2n)$ ,  $r = 0(1)2n$ , covering the period  $L$ , the programme calculates the coefficients of the harmonic approximation

$$f(x) = a_0 + \sum_{i=1}^n \left\{ a_r \cos \frac{2\pi r}{L} (x - x_0) + b_r \sin \frac{2\pi r}{L} (x - x_0) \right\}.$$

It is assumed that the values corresponding to  $x_0$  and  $x_0 + L$  are equal. The coefficient  $b_n$  is undetermined. If the function is even with respect to the mid-point of the range, then only the cosine terms are present and only the 1st  $n+1$  values  $f_0$  to  $f_n$  are needed to determine the  $a$ 's. Similarly if the function is odd (w.r.t. to the mid-point) only the sine terms are present, and  $f_0$  to  $f_n$  suffice to determine the coefficients  $b_1$  to  $b_{n-1}$  ( $f_0 = f_n = 0$  in this case).

The programme has three entry points corresponding to:

- (1) the general case, (2) the even case, and (3) the odd case.

Method: See Hildebrand's "Introduction to Numerical Analysis", p.373.

## Programme parameters

n	represents the number of intervals in a half period.
r'	the data $f\left(x_0 + \frac{rL}{2n}\right)$ stands in $r' + r$ , where <ol style="list-style-type: none"> <li>1. <math>r = 0(1)2n-1</math> in the general case (the last value corresponding to <math>r = 2n</math> being omitted since it is identical to <math>r = 0</math>),</li> <li>2. <math>r = 0(1)n</math> in the even case,</li> <li>3. <math>r = 1(1)n-1</math> in the odd case.</li> </ol> [even, odd w.r. to mid-point of the range implies even, odd w.r. to $x_0$ .]
a'	the coefficients of the cosine terms $a_0, a_1, \dots, a_n$ will be placed in $a', a'+1, \dots, a'+n$ (can be omitted in odd case).
b'	the coefficients of the sine terms $b_1, b_2, \dots, b_{n-1}$ will be placed in $b'+1, \dots, b'+n-1$ (can be omitted in even case).

The programme consists of a single chapter called in as follows

by "down 1/1 - 505" for the general case,  
 by "down 2/1 - 505" for the even case,  
 by "down 3/1 - 505" for the odd case.

Time: approximately  $(300 + 6n^2 + 28n)$  millisecs. in general case and  $(300 + 3n^2 + 13n)$  millisecs. in odd or even cases.

#### SOLUTION OF ALGEBRAIC EQUATIONS: PROGRAMME -506

This programme calculates the zeros of the polynomial

$$a_0 z^n + a_1 z^{n-1} + \dots + a_n = 0,$$

where the  $a$ 's can be real or complex.

Method used is that described by D.E. Muller in M.T.A.C., Oct.1956 (p.208).

Each root is found by an iterative process and followed by removal of the corresponding factor from the equation. The relative accuracy required in the roots is specified by a programme parameter  $e$  which is employed as a criterion for convergence, thus

$$\left| \frac{x^{(m+1)} - x^{(m)}}{\frac{1}{2}(x^{(m+1)} + x^{(m)})} \right| < e,$$

where  $x^{(m)}$ ,  $x^{(m+1)}$  are two successive iterates to a root  $x$ . Thus, to obtain 4 significant figures, set  $e = 0.0001$ .

#### Programme parameters

$n$	order of equation ( $\leq 116$ )
$e$	relative accuracy of roots
$a'$	real parts of $a$ 's stand in $a'$ , $a'+1, \dots, a'+n$
$b'$	imaginary parts of $b$ 's stand in $b'$ , $b'+1, \dots, b'+n$
$c'$	real parts of roots will be placed in $c'$ , $c'+1, \dots, c'+n-1$
$d'$	imaginary parts of roots will be placed in $d'$ , $d'+1, \dots, d'+n-1$

The programme consists of a single chapter called in as follows

1. if the  $a$ 's are complex by "down 1/1 - 506"
2. if the  $a$ 's are real by "down 2/1 - 506" (in this case it is not necessary to specify  $b'$ ).

Times: 1 min for  $n = 25$ , 4 min for  $n = 50$ , and 16 min for  $n = 100$ .

**AUTO- AND CROSS-CORRELATION: PROGRAMME -507**

Given two sequences (which may be identical):

$$x_1, x_2, \dots, x_N,$$

$$y_1, y_2, \dots, y_N,$$

the programme calculates the quantities

$$Z_s = \frac{A_s - B_s D_s / (N - s)}{\sqrt{\left\{ \left( C_s - \frac{B_s^2}{N - s} \right) \left( E_s - \frac{D_s^2}{N - s} \right) \right\}}}, \text{ for } s = 0(1)p,$$

$$A_s = \sum_{t=1}^{N-s} x_t y_{(t+s)},$$

$$B_s = \sum_{t=1}^{N-s} x_t, \quad C_s = \sum_{t=1}^{N-s} x_t^2,$$

$$D_s = \sum_{t=s+1}^N y_t, \quad E_s = \sum_{t=s+1}^N y_t^2.$$

**Programme parameters**

w	the number of terms N (limited only by the space available in the auxiliary store)
x'	the x sequence stands in x'+1, x'+2, ..., x'+w
y'	the y sequence stands in y'+1, y'+2, ..., y'+w
p	is the maximum displacement ( $\leq 379$ )
z'	the results are placed in z', z'+1, ..., z'+p

Time: approximately  $pN/160$  secs.

The programme consists of a single chapter called in by "down 1/1 - 507".

**TABULAR SOLUTION OF DIFFERENTIAL EQUATIONS : PROGRAMME -516**

This is an improved version of programme -508.

The programme will tabulate the solution of a set of differential equations

$$\frac{dy_i}{dx} = f_i(y_1, y_2, \dots, y_n; x), \quad i = 1(1)n,$$

at a uniform interval  $d$ , starting from the initial conditions  $y_i(x_0)$  at  $x = x_0$ .

The step-by-step Runge-Kutta process is used ("int step"). To ensure a given accuracy in each interval, the results of taking first  $p$  steps of size  $d/p$ , and then  $(p+1)$  steps of size  $d/(p+1)$ , are compared. If these differ by less than a preassigned small quantity  $e$  (a programme parameter) we proceed to the next interval; otherwise  $p$  is advanced by 1 until agreement is obtained, or until there is no further improvement. (In this case  $e$  is replaced by a more realistic quantity.) For the first interval we start with  $p = 1$ , and in subsequent intervals we start with a value of  $p$  less than or equal to that which proved successful in the previous interval. Thus the programme always seeks to reduce the number of steps performed.

Experience with programme -508 showed that in some instances it made a premature decision that the accuracy required was unattainable. This version of the programme accumulates more information before making a decision, and the above situation may thereby be avoided. Further, whenever it is found impossible to achieve the accuracy required, then the error parameter  $e$  is reset to a value not less than the accuracy which has been obtained. Irrespective of which of the two entries into the programme has been used, the interval and the accuracy obtained, are printed whenever it is found necessary to adjust  $e$ .

Though the maximum number of equations in the set has been nominally fixed as 50, it can be altered to any other number  $N$  (say) in the range  $1 \rightarrow 76$ , by changing the first six directives at the head of the programme from:

$u \rightarrow 50$		$u \rightarrow N$
$v \rightarrow 50$		$v \rightarrow N$
$w \rightarrow 50$		$w \rightarrow N$
$y \rightarrow 50$	to	$y \rightarrow N$
$f \rightarrow 50$		$f \rightarrow N$
$g \rightarrow 50$		$g \rightarrow N$

This change will of course affect the amount of working space available, which is, in general,  $160 - 6(N - 50)$ .

The programme is a single chapter and an auxiliary sequence to compute the  $f_1$ 's must be added thus:

programme -516	}	on the library tape
<div style="border: 1px solid black; padding: 2px; display: inline-block;">1st part of chapter</div>		
<div style="border: 1px solid black; padding: 2px; display: inline-block;">auxiliary sequence</div>	}	punched by user
close		

#### Specification of auxiliary sequence

1. chapter space available: 475 registers.
2. labels 3) to 90) available.
3. conventional entry: exit by jumping to label 101)\*.

\* Not 592, 0 as in the case of an auxiliary sequence used directly with "int step".

4. permissible working space:
- (i) 160 main variables (or  $160 - 6(N - 50)$  as above),
  - (ii) by-pass parameters: all special variables and indices other than  $h, x, k, l, o$ .
5. the auxiliary sequence must be designed to place

$$f_i(y_1, y_2, \dots, y_n, x) \text{ in } f_i, i = 1(1)n.$$

#### Programme parameters

$y'$	Assumes $x_0$ is placed in $y'$ , and $y_1(x_0)$ in $y' + i$ . Subsequently the programme places $x_0 + jd$ in $y' + j(n+1)$ and $y_1(x_0 + jd)$ in $y' + j(n+1) + i$ .
$n$	number of equations.
$m$	number of steps.
$d$	tabular interval.
$e$	accuracy required.
$e'$	final accuracy placed in $e'$ .

#### To call in the programme

- down 1/1 - 516 simply generates the solution, printing the value of  $p$  and the comparative accuracy, only for the intervals in which  $e$  is modified.
- down 2/1 - 516 generates the solution and prints the values of  $p$  and the comparative accuracy for every interval.

#### TABULATION: PROGRAMME -524

This is a revised version of Programme -509.

*Purpose.* A single-chapter programme for the sectional tabulation of a matrix  $a_{ij}$ ,  $i = 0(1)m$ ,  $j = 0(1)n$ , as a two dimensional array. Thus e.g., if  $m = 6$ ,  $n = 7$ .

$i \quad j \rightarrow$  number of columns, limited by width of teleprinter page.

↓	_____	_____	_____	_____	_____
	_____	_____	_____	_____	_____
	_____	_____	_____	_____	_____
	_____	_____	_____	_____	_____
	_____	_____	_____	_____	_____
	_____	_____	_____	_____	_____

remaining columns

_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

### Description

If the number of columns is such as to exceed the width of the standard teleprinter page (68 characters) then the array is taken out sectionally, the 1st row of each section being separated from the last row of the previous section by 10 line feeds. This allows the paper roll to be guillotined and the sections pasted up in juxtaposition. Ten line feeds are punched before the first section, but only one is punched after the last section. Each column is separated by two spaces.

A programme parameter  $q$  specifies the number of decimal places, i.e., it is the parameter which is normally used in the instruction "print (x)p,q". The associated quantity  $p$ , the number of digital positions to allow in the integral part is determined by first scanning the entire array and noting the maximum element in each column. This fixes the parameter  $p$  for that column. If for any column,  $p$  should exceed 9, then the column is printed in floating point form with  $q = 7$ , so that significant figures in the smaller elements in the column are not lost.

If, in a particular column to be printed in fixed point after appropriate rounding to  $q$  decimals,  $r$  is the largest number of non-zero significant figures in the fractional part of any entry, then all the entries in the column will have at least  $q-r$  zeros at their right hand ends. The entries in the column will then be printed to  $r$  decimals. Floating point columns are always printed 0,7. For example, if the first column was in fact an argument of the form  $0(0.01)1.00$ , say, then this column would only be printed to 2 decimals, whatever value  $q(\geq 2)$  had on entry. The values of the argument could have small rounding errors either up or down.

The programme is intended to be used in conjunction with programme -516 for tabulating the solution of a set of differential equations. In this case we have  $a_{i,0} = x_0 + id$  and  $a_{ij} = y_j (x_0 + id)$ ,  $j = 1(1)n$ ,  $i = 0(1)m$ . The programme can also be used however in place of the  $\phi_0, \phi_j$  instructions, to print a matrix, recorded in the conventional row-wise fashion.

The main differences between this programme and programme -509 are that stored values can be rounded up or down, some tests have been altered to deal more suitably with certain critical cases, and that columns requiring floating point lay-out are always printed 0,7.

### Programme Parameters

$m$	$m + 1$ is the number of rows.
$n$	$n + 1$ is the number of columns.
$q$	maximum number of decimal places for printing fixed point.
$a'$	the element $a_{ij}$ stands in $a' + i(n+1) + j$ , for $i = 0(1)m$ , $j = 0(1)n$ .

The programme is called in by "down 1/1 - 524".

### SOLUTION OF NORMAL EQUATIONS OF LEAST SQUARES: PROGRAMME -510

Given a set of  $m$  linear equations in  $n$  unknowns ( $m \geq n$ )

$$A x = b,$$

where  $A = [a_{ij}]$ ,  $x = [x_j]$  and  $b = [b_i]$ ,  $i = 1(1)m$ ,  $j = 1(1)n$ , this programme sets up



and solves the normal equations

$$A'A x = A'b, \text{ i.e., } Cx = d \text{ (say).}$$

Pivotal condensation is employed taking advantage of the symmetry of  $C$  and selecting the pivot at each stage from the reduced diagonal.

#### Programme parameters

$a'$	The original matrix $a_{ij}$ stands in $a' + (i-1)n + j - 1$ .
$b'$	Refers to the vector of r.h. sides: $b_i$ stands in $b' + (i-1)$ .
$m$	is the number of equations
$n$	is the number of variables
$x'$	the solution $x_j$ is recorded in $x' + (j-1)$ .
$c'$	1st of $\frac{1}{2}n(n+3)$ consecutive auxiliary locations used to record the normal equations, which are destroyed in the process of solution.

The programme can also be used to solve a symmetric set of equations  $Cx = d$ , provided these are recorded in the form

$$c_{11} \ c_{12} \dots c_{1n} \ d_1; \ c_{22} \ c_{23} \dots c_{2n} \ d_2; \ c_{33} \ c_{34} \dots c_{3n} \ d_3, \text{ etc.,}$$

starting at a preassigned location  $c'$  in the auxiliary store. More precisely:

$$c_{ij} \text{ stands in } c' + (n+2-\frac{1}{2}i)(i-1) + j - i,$$

$$d_i \text{ stands in } c' + (n+2-\frac{1}{2}i)(i-1) + n+1 - i.$$

The solution  $x_j$  will be recorded in  $x' + (j-1)$  as in the first case.

Note: as before, the normal equations are destroyed in the process of solution.

#### Entering the programme

down 1/1 -510      for least squares solution,

down 2/1 -510      for solution of symmetric equations only (in this case we can omit to specify  $a'$ ,  $b'$ ,  $m$ ).

#### Time of execution:

for setting up normal equations:  $mn(52+0.7n)$  millisecs. (approx.),

for solving normal equations:  $60n^2$  millisecs. (approx.).

# Ferranti Ltd

## COMPUTER DEPARTMENT

*Enquiries to:*

London Computer Centre, 68 NEWMAN STREET, LONDON, W.1

Telephone MUSEum 5040

*and*

21 PORTLAND PLACE, LONDON, W.1

*Office, Works, and Research Laboratories:*

WEST GORTON, MANCHESTER, 12, Telephone EAST 1301

*Research Laboratories:*

LILY HILL, BRACKNELL, BERKS.

*This document is published with the kind permission of Manchester University by Ferranti Ltd. as part of their service to users of Ferranti computers.*

*The document must not be reproduced in whole or in part without the written permission of Manchester University.*

<u>SYMBOL</u>	<u>UC</u>	<u>LC</u>	<u>SYMBOL</u>	<u>UC</u>	<u>LC</u>
0000.001	FREE	FREE	1000.000	FREE	FREE
0000.010	NEWLINE	NEWLINE	1000.011	C	c
0000.100	TABULATE	TABULATE	1000.101	E	e
0000.111	UC	UC	1000.110	F	f
0001.000	FREE	FREE	1001.001	I	i
0001.011	FREE	FREE	1001.010	J	j
0001.101	PUNCH ON	PUNCH ON	1001.100	L	l
0001.110	PUNCH OFF	PUNCH OFF	1001.111	O	o
0010.000	SPACE	SPACE	1010.001	A	a
0010.011	FREE	FREE	1010.010	B	b
0010.101	BACKSPACE	BACKSPACE	1010.100	D	d
0010.110	LC	LC	1010.111	G	g
0011.001	FREE	FREE	1011.000	H	h
0011.010	FREE	FREE	1011.011	K	k
0011.100	STOP	STOP	1011.101	M	m
0011.111	/	:	1011.110	N	n
0100.000	0	'	1100.001	Q	q
0100.011	3	<	1100.010	R	r
0100.101	5	=	1100.100	T	t
0100.110	6		1100.111	W	w
0101.001	9	)	1101.000	X	x
0101.010	a	z	1101.011	FREE	FREE
0101.100	h	?	1101.101	FREE	FREE
0101.111	.	,	1101.110	ESCAPE	ESCAPE
0110.001	1	[	1110.000	P	p
0110.010	2	]	1110.011	S	s
0110.100	4	>	1110.101	U	u
0110.111	7		1110.110	V	v
0111.000	8	(	1111.001	Y	y
0111.011	ß	π	1111.010	Z	z
0111.101	+	&	1111.100	FREE	FREE
0111.110	-	*	1111.111	ERASE	ERASE

## BASIC AUTOCODE FACILITIES

### The Program

Each program must begin with about six inches of blank tape followed by, e.g.

Title

Joe Bloggs - Question 4

Chapter 0

and must end with the word "close" followed by the data tape.

Variables (a, b, c, d, e, f, g, h, u, v, w, x, y, z and  $\pi$ ).

These are numbers correct to 8 significant decimal digits within the range  $\pm 10^{+70}$ .

Indices (i, j, k, l, m, n, o, p, q, r, s and t)

These are integers in the range  $-512 \leq i \leq 511$ .

Variables may have a suffix e.g.  $a_1, a_i, a_{(i-3)}$ .

The number of variables  $a_i$  is specified by a directive  $a \rightarrow N$ . A maximum of 480 suffixed variables is permitted.

$\pi$  has the value 3.14159.....

### Arithmetic Instructions.

These instructions take the form

variable = general expression.

A general expression (g.e.) is a sum of terms. Each term is a product of factors, possibly divided by a single factor. A factor may be a fixed point number, a variable or an index.

e.g.  $y_i = 2mn a_{(n+1)} - n a_n/a_o + 0.1 n + 1/7$

n.b.  $an \equiv a_n$ ,  $na \equiv n \times a$ ,  $ann \equiv a_n \times n$ .

An index can also be set equal to an expression of similar form involving only indices and integers. The use of the solidus is then, not permitted.

### Elementary Functions.

The instructions take the form

variable =  $\psi F(g.o.)$

F may be sqrt, sin, cos, tan, exp, log, mod, int pt, fr pt or sign.

An index can be set by  $i = \psi \text{int pt } (g.o.)$

There are also functions with two arguments. These instructions take the form

variable =  $\psi G(g.o., g.o.)$

G may be divide, arctan or radius.

### Jump Instructions.

Unconditional : jump 2 or jump (n)

Conditional : jump 2,  $\alpha$  or  $\beta$

$\alpha$  may be =,  $\neq$ , > or  $\geq$ .  $\alpha$  and  $\beta$  must be of the same type i.e. indicos/integers or variables/fixed point numbers.

n) = 3) makes jump (n)  $\equiv$  jump 3. For cycles of instructions use

$i = p(q)r$   
 $\equiv$   
 $\equiv$   
 repeat

$i = p(-q)r$   
 $\equiv$   
 $\equiv$   
 repeat

### Input.

Numbers may be read from a data tape by

read (x) or read (n)

The number is punched in either in fixed or floating point form and terminated by C.R. L.F. or SP. SP.

### Output.

The value of a g.o. can be printed by

print (g.o.) n, n

This prints n figures before the point and n after. If n  $\equiv$  0 the number is punched in floating point form. One can also use

newline and space .

### Quickies.

To save time list the functions being used e.g.

$\psi$  sin,  $\psi$  exp, before writing "close".

### Runge-Kutta

To integrate the differential equations

$$\frac{dy_i}{dx} = f_i(x, y_1, y_2, \dots, y_n)$$

a subsequence of instructions must be provided to calculate the functions  $f_i$ . The first instruction must be labelled and the last must be 592, 0.  $h$  must be set equal to the step and n to the number of equations. The instruction

int step (m)

(where m is the entry to the subsequence) will advance the solution by one step. The variables  $g_1 - g_n$  and  $h_1 - h_n$  are used and must not be disturbed.