THE PROGRAMING OF LAPLACE'S EQUATION FOR SOLUTION BY ITERATION

(on Whirlwind I)

by

John H. Holland

Submitted in Partial Fulfillment of the
Requirements for the Degree of
Bachelor of Science

at the
Massachusetts Institute of Technology
1950

Signature of Author _____
Department of Physics, May 19, 1950

Signature of Thesis Supervisor _____

Cambridge, Massachusetts
May 19, 1950

Professor Joseph S. Newell
Secretary of the Faculty
Massachusetts Institute of Technology
Cambridge, Massachusetts

Dear Sir:

This thesis, entitled The Programing of Laplace's Equation for Solution by Iteration (on Whirlwind I), is presented herewith in partial fulfillment of the requirements for the degree of Bachelor of Science in Physics at the Massachusetts Institute of Technology.

Respectfully,

John H. Holland

ACKNOWLEDGEMENT

The author wishes to acknowledge at this point the extremely helpful advice given to him by Professor Kopal at various crucial stages in the development of this thesis. Thanks are also to be extended to R. R. Everett and J. C. Proctor for their aid in providing access to the library of Project Whirlwind.

TABLE OF CONTENTS

		Page
	Letter of Transmittal	iii
	Acknowledgement	iv
	Table of Contents	v
	Summary	vii
1	Introduction	1
2	Mathematical Definition of the Problem	5
	2.1 An equivalent arithmetic form of Laplace's equation	7
	2.2 The mesh and boundary conditions	10
	2.3 The iteration procedure	11
3	General Considerations Pertaining to a Digital Computer	14
4	The Program	19
	4.1 Problems in programing the basic iteration	20
	4.1.1 arrangement of data storage	20
	4.1.2 scale-factor problems	22
	4.2 Problems in advancing the cycle to a new u_o	23
	4.2.1 generation of a new u_o	23
	4.2.2 detection of boundary values	24
	4.3 Stopping the Iteration	25
	4.4 Mathematical checks	29

5 Evaluation 31

5.1 Number of orders and storage registers 32

5.2 Number of operations and sample solution times 32

5.3 Advantages and disadvantages to the program 34

Footnotes 35

Bibliography 37

Appendix I - The Code 38

Appendix II - The Flow Diagram 48

SUMMARY

This paper contains a discussion of the problems involved in programing and coding Laplace's equation for solution on a large-scale digital calculator. A relaxation method is chosen as the means of solution and the consequent problem of stopping the iteration before roundoff error becomes appreciable is confronted. The resulting code and an evaluation of it are included in the latter part of the report.

1

INTRODUCTION

1 Introduction

The solution of Laplace's equation ($\nabla^2 u = 0$) is an important step in a vast number of the problems encountered in physics and electrical engineering. Many of its solutions are certainly widely known to physicists and electrical engineers; but, though this equation is the simplest involving the Laplacian operator (∇^2), it still may prove at times an extremely difficult task to obtain an explicit solution. When the boundary conditions are complicated (in shape for instance) the solution is at best an exacting process. Yet, where the object is to obtain a specific set of values for u within the bounded region, and this certainly is not an infrequent case, several methods of solution exist whose application is a mere matter of routine.

Given this information, one might begin to consider whether or not it is possible to mechanize such a routine so that little or no human intervention would be necessary in its application. It is possible; indeed the mechanization of such routines comprises the purpose of most large-scale computers in existence today.

It is the object of the present paper to demonstrate one way of setting up Laplace's equation so that it can be solved by such a machine, more specifically the machine being constructed under Navy Project Whirlwind.[1] It is hoped not only that the method chosen has resulted in a program which is simple and

compact compared with the other programs possible, but that its development will also make clear some of the more important considerations to be taken into account in any application of the method.[2] At every stage an attempt has been made to keep the development general enough so that it may be possible to utilize the results obtained, or part of them, in other problems of a similar nature.

The method to be used was determined partly through the fact that Whirlwind is a digital computer, and partly through consideration of work in this area of investigation already completed by staff members of the prroject. The fact that Whirlwind is a digital device makes the employment of the methods of numerical analysis almost mandatory. Once the equation has been reduced by these methods to the proper arithmetic form, which consists of a determinate set of simultaneous linear algebraic equations, two broad approaches to the solution are open: 1 - solution of the resultant equations by elimination, or 2 - solution by the so-called approximate methods. Since a fairly comprehensive Project Whirlwind engineering note has already been published on the program for solution of simultaneous equations by elimination it was decided that the present paper should be directed toward utilizing one of the approximate methods.[3] Reference to another Whirlwind engineering note made it clear that some modification of the single-step iteration method developed by Gauss and Seidel should be used.[4] Later discussion will show that the iterative method chosen

will be preferable in most cases to elimination, both as to simplicity of code and as to the accuracy attainable.

With the foregoing considerations in mind the direction the present paper was to take had been almost completely determined. However it was early discovered that if the investigations were to be completed within the alloted time it would be necessary to somewhat restrict the scope of the method chosen. Two such restrictions were made:

1 A lattice or mesh of square sections was employed to cover the bounded region.

2 The boundary conditions were considered to be given as values of u at the external points of the lattice.

The significance of these restrictions will be discussed on pages 7 ff. and pages 10 ff., respectively, of this report. It will be seen that neither of these restrictions seriously limits the generality of the Gauss-Seidel method when it is applied to the solution of partial differential equations on machines of the Whirlwind type/

2

MATHEMATICAL DEFINITION OF THE PROBLEM

2 Mathematical Definition of the Problem

2.1 An Equivalent Arithmetic Form of Laplace's Equation

2.2 The Mesh and Boundary Conditions

2.3 The Iteration Procedure

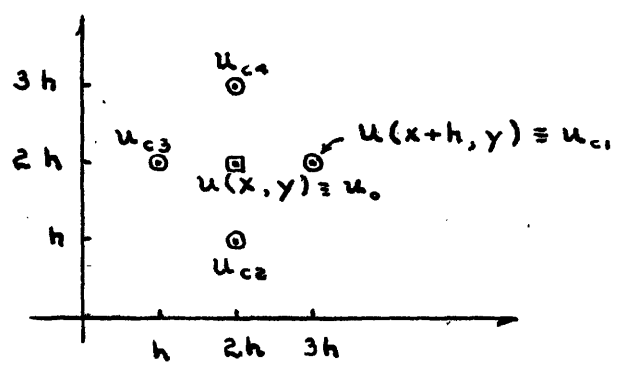
Laplace's equation (in vector notation $\nabla^2 u(x,y) = 0$), as noted in the introduction, is one of the most familiar equations of mathematical physics. The equation makes its appearance in an important position in almost every field of physics; as examples consider static elasticity (the wave equation with zero time variation), diffusion, steady flow of heat or electricity, irrotational motion of an incompressible fluid, and perhaps most familiar of all the potential distribution of a charge-free space. Important though these particular problems may be, the equation of Laplace gains its greatest import from the fact that it is the simplest equation involving the Laplacian operator, which appears in such basic equations as the wave equations of wave-mechanics. Thus a knowledge of the solutions of Laplace's equation provides a foundation for the solution of a great many of the problems formulated by theoretical physics.

Since Laplace's equation is a partial differential equation (in expanded form, $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$), its general solution involves arbitrary functions. In the usual case this solution is also required to meet a set of "boundary" conditions so that the arbitrary function becomes particular. A necessary and sufficient condition for the boundary conditions to specify a particular solution in this two dimensional case is that the value of u

be specified at every point on the boundary of the region under consideration. If the boundary conditions are given in another form, for instance as a derivative of u at the boundary, it is possible in most cases to reduce them to the form given above; thus there will be little loss of generality if for the purposes of this investigation it is assumed that the boundary conditions are available as values of u on the boundary.

2.1 An Equivalent Arithmetic Form of Laplace's Equation

The reduction of Laplace's equation to a form suitable for use by Whirlwind is possible through use of the calculus of finite differences. One possible derivation is given by Courant in his Advanced Methods of Applied Mathematics.[5] A short synopsis follows.-



$$\begin{cases} x = mh \\ y = nh \end{cases}$$

$(m = 0, 1, 2 \ldots)$
$(n = 0, 1, 2 \ldots)$

Fig. 1.

With reference to the diagram (Fig. 1.), Courant defines a forward and a backward difference as follows:

forward difference $u_x(x,y) = \dfrac{u(x+h,y) - u(x,y)}{h}$

backward difference $u_{\bar{x}}(x,y) = \dfrac{u(x,y) - u(x-h,y)}{h}$

then, $u_{\bar{x}x} = \dfrac{u_{\bar{x}}(x+h,y) - u_{\bar{x}}(x,y)}{h}$

and $\nabla_h^2 u =$ difference operator $= u_{x\bar{x}} + u_{y\bar{y}}$

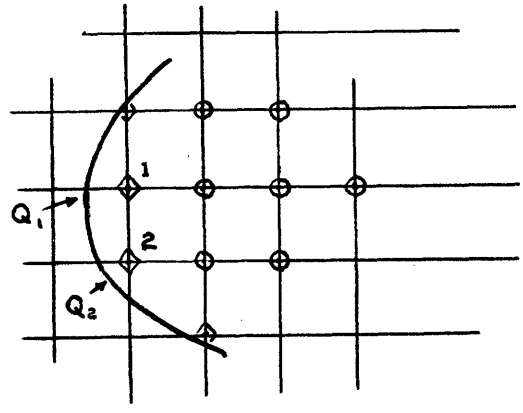
or $\nabla_h^2 u = \dfrac{u_{c1} + u_{c2} + u_{c3} + u_{c4} - 4u_0}{h^2}$

For the problem of this thesis, i. e. $\nabla^2 u = 0$

$$u_0 = \frac{u_{c1} + u_{c2} + u_{c3} + u_{c4}}{4} \qquad 1)$$

Equation 1) is the basic iteration equation. Here the Laplacian operator has been replaced by a finite difference operator which involves only arithmetic operations on the ordinates.

The use of the arithmetic equivalent of Laplace's equation to obtain a solution over an extended finite region originally was accomplished by an adaptation of the mesh analogy used by Southwell.[6] The analogy consists of exchanging the continuous variation of u in the region of interest for a "net" whose internal nodal points eventually take on the value of u at the equivalent point in the region and whose external (boundary) nodes are fixed at the corresponding boundary values.



⊙ - interior point (arbitrary value of u)

◈ - boundary point (fixed value of u)

Fig. 2.

The procedure for solution, once appropriate values have been fixed for the external nodes (cf. Fig. 2.), consists first in assigning arbitrary values to all other nodes; the net is then relaxed by a systematic application of the difference operator. One system of relaxation begins by choosing one node as a reference or center, applying the basic iteration equation (eq. 1), and replacing the value at the reference point by the one so obtained. This process is then carried out over the net until all nodes have had their original arbitrary values replaced in this manner; the whole relaxation is repeated as many times as necessary until the successive values at any given node do not change sensibly.

Since the choice of an iterative method over an elimination procedure was more or less an arbitrary one, it would be well perhaps to give a preliminary comparison of the two approaches. Elimination on the one hand has a smaller total number of arithmetic operations (indicating a shorter solution time); on the other hand iteration involves a more routine approach and roundoff error is kept to a minimum. It would seem that elimination is suited to use for small systems of equations where the roundoff inherent to exact procedures is not the limiting factor. Iteration would seem more suited to problems where high accuracy is a requirement and where region of solution is large (many mesh points). Further information concerning exact and especially elimination procedures may be found in Project Whirlwind engineering note E-161.[7]

2.2 The Mesh and Boundary Conditions

It will be noted that the finite difference operator derived is suitable for use on meshes of square section only. That other types of mesh are possible is evident; a triangular mesh (i. e. a mesh where each node is connected to six other nodes) has often been used in the solution of problems by the relaxation method.[8] The advantage of different basic mesh shapeslies in the way they can be fitted to the continuous boundary surrounding the region of interest. This advantage obviously decreases as the mesh interval gets smaller since the smaller the interval the greater the number of points in the vicinity of the boundary. If the capacity of Whirlwind is used as an example it can be assumed that it will be possible to calculate a minimum of 1000 values of u for as many nodes. This would correspond to upwards of a hundred points lying near the enclosing boundary. It seems reasonable that with this many points (and the corresponding small interval) the accuracy of the approximation to the boundary will not be sensibly affected by the mesh shape.

To give some concrete idea of how the fixed values for the boundary nodes are obtained it might be good to say a word about the simplest way of setting down these values. Only mesh points interior to the boundary are used, the points adjacent to the boundary being assigned the value of the boundary function near them. For instance, in Fig. 2 the value of the boundary function at Q_1 and Q_2 would be assigned to points 1 and 2 respectively.

There are of course other methods for obtaining the fixed values of the boundary points (e. g. see L. Fox, Proc. Roy. Soc. A190, 31-59, 1947) and which one is chosen is a matter of the ultimate accuracy needed in the boundary values. Since the boundary values so obtained are presented to the computer in the form of initial data, they have little effect on the program for solution and will thus be considered as given quantities henceforth.

2.3 The Iteration Procedure

The exact relaxation procedure decided upon is similar to one described by Courant. It may be summarized as follows:

1. Assume for $u(x,y)$, at all interior points, values, preferably between the maximum and minimum boundary values (giving a first approximation ^1u_i).
2. Order the interior net points (1, 2, 3, ...N) in some arbitrary manner, P_1, P_2, ...P_N.
3. Using the basic iteration formula (eq. 1) centered on the point P_1, replace the first approximation $^1u_1 \equiv {}^1u(P_1)$ by the value obtained from eq.1, thus obtaining $^2u_1$. Do the same for u_2, u_3, ...u_N, thus arriving at a second approximation for $\underline{u}$. If the formula (eq. 1) includes any earlier changed values the changed value is used, i. e. if ^nu_j is the value being replaced, then if $i < j$, the value $^{n+1}u_i$, previously computed, is used instead of ^nu_i.
4. Continue this process obtaining ^3u, ^4u, etc. un-

-til maximum accuracy has been reached.

It can be readily demonstrated that this method is equivalent to the Gauss-Seidel single-step iteration. The Seidel method can be represented schematically by the following set of equations:

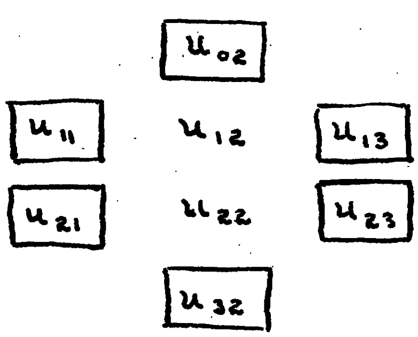
$$A_{11}\ {}^{n+1}u_1 + A_{12}\ {}^{n}u_2 + A_{13}\ {}^{n}u_3 + \dots + A_{1j}\ {}^{n}u_j = B_1$$

$$A_{21}\ {}^{n+1}u_1 + A_{22}\ {}^{n+1}u_2 + A_{23}\ {}^{n}u_3 + \dots + A_{2j}\ {}^{n}u_j = B_2$$

$$- -$$

$$A_{j1}\ {}^{n+1}u_1 + \dots \qquad \dots + A_{jj}\ {}^{n+1}u_j = B_j$$

where ${}^{n}u_i$ represents the nth correction to u_i. Thus, if the nth correction of u be considered given, the n+1 correction is obtained for u_1 by solving equation 1 for u_1 (considering u_1 for the moment as unknown but using all the other values as given), similarly equation 2 is solved for u_2, and so on until the n+1 correction has been obtained for all u's, i. e. i = 1, 2, 3, ...j. This is exactly the system described in the first paragraph of this section, albeit somewhat differently. A simple example may make the equivalence clearer.



{the values enclosed in boxes are the fixed boundary values.

Fig. 3.

Applying the method described in the first paragraph of this section, the following equations are obtained:

$$4^{n+1}u_{12} - {}^{n}u_{13} - {}^{n}u_{22} - {}^{n}u_{11} - {}^{n}u_{02} = 0$$

$$4^{n+1}u_{22} - {}^{n}u_{23} - {}^{n}u_{32} - {}^{n}u_{21} - {}^{n+1}u_{12} = 0$$

But the boundary values are fixed at all times and hence may be transposed to the right side of the equations.[9] Transposing and rearranging.-

$$4^{n+1}u_{12} - {}^{n}u_{22} = u_{13} + u_{11} + u_{02} = b_1$$

$$-{}^{n+1}u_{12} + 4^{n+1}u_{22} = u_{23} + u_{32} + u_{21} = b_2$$

but this is exactly the matrix arrangement given above for the Seidel process.

The equivalence just shown is especially significant when one considers the conclusions concerning iterative methods which were reached in Project Whirlwind engineering note E-148.

> " 1) A single-step iteration method should be used because
>
> a) It is easy to obtain sufficient conditions for convergence.
> b) A means for instructing the machine when to stop iterating is available.
> c) Experimental evidence seems to indicate that the product (number of iterations needed to obtain a given accuracy) X (amount of computation per iteration) is less for single-step methods than with the others examined.
>
> 2) The single-step method should be the Seidel method because it requires less computation than the other single-step methods known. "[10]

3

GENERAL CONSIDERATIONS PERTAINING TO DIGITAL COMPUTATION

3 General Considerations Pertaining to Digital Computation

The previous section has traced out the operations imposed by the mathematics of the problem; the present section recounts the operations necessary to adapt the problem to solution by a large-scale digital computer such as Whirlwind; the next section will detail the manner in which the requirements of the tow earlier sections are mutually satisfied, resulting in the "program".

While an attempt will be made in the sections that follow to exclude the more unique qualities of Whirlwind (i. e. those not likely to be found in other large-scale digital computers), it must be remembered that the word computer and similar expressions when used refer first and foremost to Whirlwind. All program notation appearing hereafter is that suggested in Project Whirlwind conference note C-93; the actual order code used is that described in Project Whirlwind engineering note E-235.[11] In some of the work which follows, notably the code and the more detailed parts of the program, it must be assumed that the reader has a working knowledge of the foregoing notes.[12]

The first consideration in any program is the systematic input of data to the computer. In the present case the data consists of boundary conditions and initial estimates for $\underline{u}$ at the interior mesh points. The interior points should be ordered in such a way that the computer can obtain the $\underline{u}$'s

for the basic iteration with a minimum of coding. At the same time, since the fixed boundary values are ordered along with the interior points (because the boundary values will at times appear in the basic iteration, cf. Fig. 3 and accompanying example) the machine must have some means of detecting them to prevent their becoming modified during the iteration.

Once the input has been accomplished it is possible to proceed to the iteration. It is evident that some system must be found, within the limit set by orders which the computer can interpret, to provide the correct data for the basic iteration to the arithmetic element in the proper sequence. After the proper artimetic operations have been preformed it will be necessary to return the corrected $\underline{u}$ to its proper position in the storage and then advance the cycle so that it will operate on the $\underline{u}$ next in order.

Finally some method will have to be provided to stop the iteration automatically when the accuracy obtained can no longer be increased. Also at this point as well as throughout the iteration some purely mathematical checks should be applied to detect, and if possible eliminate, any inadvertant arithmetic errors introduced. It would likewise be advantageous to be able to estimate the relative accuracy of the final answer.

In addition to the foregoing very general considerations, there is one problem introduced by Whirlwind which is common to many but not all digital computers ... the scale factor.

The problem results from the finite size of the storage registers. The storage of Project Whirlwind is of the parallel type, a complete number being stored in a register having a simgle address. The storage registers can hold one fifteen binary digit number (equivalent to approximately five decimal places) with its sign, or alternately one complete order. It has been decided that the number interval represented by the range from plus fifteen places to minus fifteen places should be $1 - 2^{-15}$ to $-(1 - 2^{-15})$. (This interval was chosen from the others possible, such as the interval 2^{15} to -2^{+15}, for a reason which will soon become evident). The number range being thus limited, a sufficiently large number can and will overflow the limits of the storage register. While there is an automatic alarm to indicate this condition it is not remedied automatically; therefore special precautions must be taken at every stage of operation to prevent such an overflow.

The scale-factor problem is two-fold --- all data must be multiplied by an appropriate scale-factor to bring it within the limits of the number system (i. e. all data must be reduced to an absolute value less than one for Whirlwind), secondly precautions must be taken at every stage of arithmetic operation to prevent an overflow. An arithmetic overflow will occur whenever two numbers greater than $\frac{1}{2}$ are added (since the sum will then be greater than one) or when a division is such as to give a number greater than one. It is now evident why a system where all the numbers are less than one was chosen;

multiplication being a frequent arithmetic procedure, more so than division, it was desired to make an overflow by multiplication a physical impossibility (i. e. two numbers whose absolute value is less than one cannot have a product greater than one).

It should be noted that orders must in general be stored in sequence, the sequence in which they will be applied, since the computer obtains a new order (unless specifically instructed otherwise by an order) by adding one to the address of the order of the moment and then "looking up" the new order at this modified address. This restriction will affect the actual coding of the orders a great deal.

4

THE PROGRAM

4 The Program

For convenience the results of the last two sections have be rearranged into the four general catagories which follow:

4.1 Problems in Programing the Basic Iteration

4.1.1 arrangement of data storage so that the $\underline{u}$'s may be obtained with minimum coding.

4.1.2 scale-factor problems.

4.2 Problems in Advancing the Cycle to a New $\underline{u}_o$

4.2.1 generation of new $\underline{u}_o$.

4.2.2 detection of boundary values.

4.3 Stopping the Iteration at Optimum Accuracy

4.4 Mathematical Checks

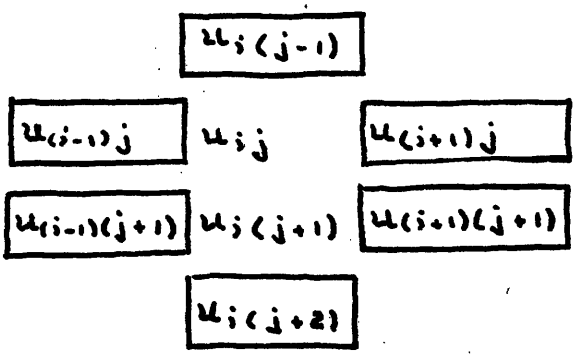
4.1 Problems in Programing the Basic Iteration

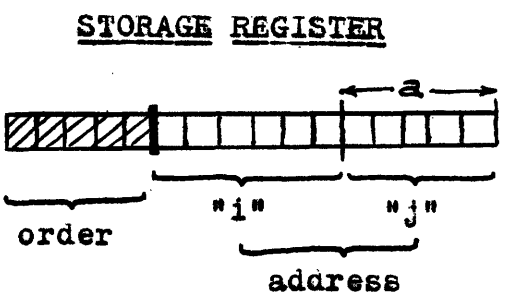
4.1.1 Arrangement of Data Storage

Given any u_o, it will be necessary to obtain the four connecting $\underline{u}$'s for the basic iteration (see eq. 1. and Fig. 1.). Restated with reference to coding this statement becomes, "Given the address for any u_o it must be possible to obtain by means of standard orders the address of the four connecting $\underline{u}$'s." Thus, the address of any $\underline{u}$ must be related in some standard and unvarying manner to the address of the four connecting $\underline{u}$'s. The necessary relation is accomplished through dividing the address of any given $\underline{u}$ into two parts; the first part signifies the column number, the second part signifies the row number. Since the address of any register is just a number this can be

accomplished by using the last a places for the row number, the preceding places in the address section being used for the column number.[13] To obtain the connecting u's above and below u_0 it is only necessary to add plus or minus one to the portion of the address used to designate the row number: a similar operation will generate the adjacent u's.

Consider this process, together with Whirlwind-type storage registers and number system, as applied to the grouping of Fig. 3 .-





Addresses in Decimal Notation			Addresses in Binary Notation		
	0201			00001000001	
0102	0202	0302	00000100010	00001000010	00001100010
0103	0203	0303	00000100011	00001000011	00001100011
	0204			00001000100	
base 10	a = 2		base 2	a = 5	

Fig. 4.

If the address of the register containing u_o is given, then u_{c1} , for instance, may be obtained by adding 1 x (base)^a ; thus if the address of u_o is 0203, then the address of u_{c1} is equal to $0203 + 1 \times 10^2 = 0303$, as indeed it is.

4.1.2 Scale-factor Problems

The scale-factor problem makes itself evident in two distinct ways. First of all the original input data must be so adjusted that it will not overflow the storage registers when read into the computer. Since this problem is essentially a "preliminary measure" which has no effect on the main program it will be given no further consideration here. However, the other scale problem, that of overflow during arithmetic operation, affects the computer whenever an addition or a division is preformed. Essentially all arithmetic operations of this type appear during the basic iteration cycle, thus it becomes necessary to inspect all operations therein for any possible overflow and then to code the problem in such a way that an overflow becomes impossible.

Inspection shows that the only possiblity for overflow appears during the formation of u_o from the connecting u_c's. The basic equation requires thatthe four u_c's be summed and divided by four. The most economical way to accomplish this, forgetting for the moment the possibility of overflow, is to add the four u_c's and then divide by 4. It is immediately obvious that such a procedure involves the possibility of overflow, for if any of the u_c's are close to 1 in value, then

it is easily possible for the sum of the u_c's , before division, to exceed 1 and the register capacity. The digits overflowing the register are lost and the computation must be stopped. The alternate method of dividing each u_c by 4 and then summing allows no possibility of overflow since each u_c divided by 4 must be less than $\frac{1}{4}$, and hence the sum of the four u_c's must be less than 1. Thus, the somewhat longer method of dividing each u_c in turn by 4 proves to be the feasible one.

4.2 Problems in Advancing the Cycle to a New u_o

4.2.1 Generation of a New u_o

Once the basic iteration has been preformed on a given u_o, it becomes necessary to advance the cycle so that it will be preformed on the u next in order. The scan of the overall process is duch that the iteration cycle proceeds from left to right and then, upon reaching the boundary of that row, begins again at the left of the next row down. It can be seen that as long as u_o is not adjacent to a boundary value on the right the problem of finding the address of the new u_o simply involves adding 1 to the column number (i) of the address of the present u_o (cf. section 4.1.1 if this terminology is not clear). The condition obtaining when u_o is adjacent to a boundary value is intimately connected with the problem of detection of boundary values and will be discussed along with this problem in the next section.

4.2.2 Detection of Boundary Values

Boundary values must be ordered along with the values of $\underline{u}$ being solved for, yet the boundary values must not be altered by the basic iteration when it has advanced to the point where the address of the new u_o coincides with the address of the boundary value.[14] Thus some means must be provided in the code for detecting when the tenative address for the new u_o coincides with a boundary address. This detection is accomplished through storing the address of the righthand boundary values (future discussion will show that these are the only ones involved) in an additional set of registers whose address is equal to the row number (j) plus a constant (c). In this manner by detecting the row number (j) in the address of the tenative u_o , adding the constant $\underline{c}$, and comparing the contents of the register whose address is $j+c$ with the tenative address of u_o it will be possible to see if the tenative address coincides with the address of the boundary value for that row. As an example (cf. Fig. 4).-

Let $c = 15$

and let register 17 contain 0302,

register 18 contain 0303

$15+j \equiv \underline{RC}$ (the address of the righthand boundary value for row j)

thus given $\underline{RC}\ u_o = 0203$,

$j = 3$

and $15 + j = 18 = \underline{RC}$ (0303, or the address of the boundary value for row 3).

If the tenative u_0 does coincide with the righthand boundary value, then some system must be set up to obtain the first $\underline{u}$ to be solved for in the next row down. This can be accomplished in the same manner as the boundary value detection, using this time a new constant $\underline{c}'$ and storing the address of the desired $\underline{u}$ in the register whose address is $j+c'$. By forming the address $j+c'$ as above and transferring the contents of that register to the programing unit the basic iteration will be preformed on the first non-boundary u in the next row down.

4.3 Stopping the Iteration

The problem involved here is the result of roundoff error. If the iteration could be carried out with no roundoff it would be a monotonicly convergent process tending toward the correct value of $\underline{u}$ in the limit; however, with roundoff the process first converges toward $\underline{u}$, then as roundoff error accumulates the values obtained obtained begin to oscillate about some mean value, with amplitudes proportional to the accumulated roundoff.

If there were no roundoff the iteration could be stopped by instructing the machine to go to the output program whenever the difference between successive values of $\underline{u}$ became smaller than a certain amount. With roundoff such a procedure is no longer possible for there is no assurance that the difference between two successive values of $\underline{u}$ will reach a given value before the oscillation sets in. The oscillation, being

a function of the accumulated roundoff error, builds up and of course precludes the possibility of two successive values of $\underline{u}$ being less than a specified amount, except by chance, if they were not so before the oscillation was appreciable.

Consideration will show that what is desired in this case is a function related inversely to the composite accuracy of all the $\underline{u}$'s. Such a function will decrease as long as the overall accuracy of the solution (i. e. the solution for the region under consideration) increases, but as soon as the roundoff becomes of the same order of magnitude as the difference between the approximate $\underline{u}$ and the correct $\underline{u}$ the function begins to increase. Consequently the difference between two successive values of the function will be positive until the point of optimum accuracy is reached, at which time the sign will change to negative and remain so. Thus, by inspecting the sign of this difference after each relaxation, it would be possible to terminate the calculation at the point of highest accuracy.

There is a function discovered by Boussinesq which answers these qualifications. It is an integral function which he developed while using the method of least squares to solve the differential equation,

$$u'' + \lambda A(x) u + f = 0 .$$ [15]

As modified for use in this report the function takes the form,

$$\iint [\nabla^2 u(x,y) - 0]^2 \, dx \, dy$$

To be used on a digital computer the equation must be reduced to an equivalent arithmetic form. The following derivation

was accomplished using the trapezoidal rule for general n .-

consider $$P(x,y) \equiv \left[\nabla^2 u(x,y)\right]^2$$

and $$P_{ij} \equiv \left[\nabla^2 u_{ij}\right]^2$$

where the subscripts "i" and "j" have the same meaning as in Fig. 4.

if h_x and h_y are the ordinate and abscissa intervals respectively,

$$\frac{1}{h_x h_y} \iint \left[\nabla^2 u(x,y)\right]^2 dx\,dy = \frac{1}{h_x h_y} \iint P(x,y)\,dx\,dy$$

$$= \frac{1}{h_y} \int \left[\frac{1}{2} P_{1j} + \sum_{i=2}^{(n-1)} P_{ij} + \frac{1}{2} P_{nj}\right] dy$$

where the sum enclosed in the brackets is a function of y (or eventually j), p_{1j} being the first value of p in row j and p_{nj} being the last.

$$= \frac{1}{h_y} \int \frac{1}{2} P_{1j}\,dy + \frac{1}{h_y} \int \sum_{i=2}^{(n-1)} P_{ij}\,dy + \frac{1}{h_y} \int \frac{1}{2} P_{nj}\,dy$$

$$= \frac{1}{2}\left[\frac{1}{2} P_{11} + P_{12} + \ldots + P_{1(n-1)} + \frac{1}{2} P_{1m}\right]$$

$$+ \left[\frac{1}{2} \sum_{i=2}^{n-1} P_{i1} + \sum_{j=2}^{n-1} P_{i2} + \ldots + \frac{1}{2} \sum_{j=2}^{n-1} P_{im}\right]$$

$$+ \frac{1}{2}\left[\frac{1}{2} P_{n1} + P_{n2} + \ldots + \frac{1}{2} P_{nm}\right]$$

where p_{1m} is the first value of p in the last row, p_{2m} the second value,..., and p_{nm} the last value of p in the last row.

N. B. n is variable with j, that is the last value in row 2 may be at a different ordinate than the last value in any row $j \neq 2$.

The equivalent arithmetic form of the Boussinesq integral may now be written -

$$\iint \left[\nabla^2 u(x,y)\right]^2 dx\,dy = S$$

$$= h_x h_y \left[\frac{1}{4}\left(p_{11} + p_{1n} + p_{m1} + p_{mn}\right) + \frac{1}{2}\left(\sum_{j=2}^{m-1} p_{1j} + \sum_{j=2}^{m-1} p_{nj}\right) + \sum_{i=2}^{n-1}\sum_{j=2}^{m-1} p_{ij} \right]$$

N. B. There is the implicit assumption here that all rows contain at least two points, the trapezoidal rule being inapplicable as such to a row with u given at only one point.

Inspection of the arithmetic form given above reveals three cases which the computer must detect and react to:

1. The computer must multiply the first and last values of p_{ij} in the first and last rows by $\frac{1}{4}$.
2. The computer must multiply the first and last values in all other rows by $\frac{1}{2}$.
3. The computer must leave all other values of p_{ij} unmodified.

The rules applying in all three cases are routine, and the code needed to modify p_{ij} according to them is simple, being very similar to the code used to detect boundary values.

Several things should be noted about the final result above. First of all, the assumption that every row has at least two values of u being solved for has a negligible effect on

the solution, if one considers that any orientation of the boundary may be used and that the mesh interval may be made such as to allow at least two points in the rows next to the top and bottom (the only places where the appearance of less than three points is probable).

Secondly it should be especially noticed that the value computed for $\nabla^2 \underline{u}$ at any point utilizes the same summation of the connecting u_o's as the basic iteration does while computing the new u_o .[16] Thus, a great saving of orders can be attained by combining the first part of the basic "stop iteration" cycle with the equivalent part of the basic iteration cycle. The way in which this was accomplished is shown in the detailed flow diagram of the combined cycles (Appendix II).

4.4 Mathematical Checks

The arithmetic checks necessary actually have been taken care of implicitly in the preceding sections. To see this consider the reasons for such checks: 1) elimination of any inadvertant arithmetic errors (usually the result of some temporary malfunction of the machine), 2) ascertainment of the relative accuracy of the final result. The first check is made unecessary by the fact that the iteration procedure tends to be self-correcting in the long run, i. e. numerical mistakes in replacement (of $^{n-1}\underline{u}$ by $^{n}\underline{u}$) will be corrected automatically if the iteration process is carried out a sufficient number of times. The second check, on the relative accuracy of the final result, has already been accomplished in the procedure

used to stop the iteration, which works precisely on this principle. By examining S^n, that is the value of the Boussinesq integral for the n relaxation, one immediately has an estimate of the square of the error for the solution as a whole (i. e. an average of the squares of the individual errors in $\underline{u}$).

5

EVALUATION

5 Evaluation

5.1 Number of Orders and Storage Registers

5.2 Number of Operations and Sample Solution Times

5.3 Advantages and Disadvantages to the Program

5.1 Number of Orders and Storage Registers

Number of Orders (cf. Appendix I, Order Code): 93

Number of Storage Registers (see Appendix I for designations):

Type	Number
j + c	$\underline{m}$ = total no. of rows
j + c'	$\underline{m}$
B 1.n	$\underline{n}$ = total no. of u_{ij} / total no. of lattice pts.
R	93
B 2	8
C	12
Total (T)	$\underline{n} + 2\underline{m} + 113$

If n is large then $\underline{m} \cong \sqrt{\underline{n}}$ and the total number of storage registers is related to the number of lattice points by the following approximate equation:

$$T = n + 2\sqrt{n} + 113$$

5.2 Number of Operations and Sample Solution Times

Let $\underline{m}$ and $\underline{n}$ retain the definitions of section 5.1 and let $\underline{a}$ equal the number of times the overall iteration cycle is preformed

Total Number of Operations (sections are those used in Appendices I and II):

Section	No. of Orders	Times Used in One Overall Iteration
2.1	11	$n+1$
2.2	3	$n-m+1$
2.2.1	2	m
2.3	61	n
3.1	3	m
3.1.1	6	$m-1$
3.2	3	1
3.2.1	4	1
Total	93	

Total operations: $[(n+1)11 + (n-m+1)3 + 2m + 61n + 3m + 6(m-1) + 3 + 4]a =$

$= [75n + 8m + 15]a$

If the approximation of 5.1 is used the following relation obtains: $\cong [75n + 8\sqrt{n} + 15]a$

If it is assumed that Whirlwind will preform 20,000 operations per second, then the following relation holds between the number of lattice points and the solution time:

If $n = 1600$, the solution time is $6a$ sec.

If $n = 2500$, the solution time is $9a$ sec.

5.3 Advantages and Disadvantages to the Program

Advantages:

The program is self-correcting to a high degree (cf. sec. 4.4).

The relative accuracy of the final solution can be estimated (cf. sec. 4.3).

The program for the solution is essentially simple (cf. Appendix II).

The number of orders is relatively small (cf. Appendix I).

The program can handle a very large system of lattice points (an attribute of the iterative approach).

Disadvantages:

For the purposes of machine computation iteration has but one disadvantage ... it involves a relatively greater number of operations than an "exact" method. However, an inspection of the sample solution times given in 5.2 will show that this disadvantage has become a negligible one in the case of machines as rapid as Whirlwind.

FOOTNOTES

1 See reference 7 of the Bibliography.

2 There is every reason at this time to believe that the first half of this statement has been fulfilled; as for the latter half, it is for the reader of this paper to decide in what measure that goal has been attained.

3 See Ref. 7, E-161 of the Bibliography.

4 See Ref. 7, E-148 of the Bibliography.

5 See Ref. 1 of the Bibliography.

6 See Ref. 4 of the Bibliography.

7 See Ref. 7 of the Bibliography.

8 See Ref. 4, chapt. 2, of the Bibliography.

9 It should be noted that the boundary values are at all times fixed and therefore are never modified by the basic iteration (eq. 1.), although they appear in the equation whenever u_0 is adjacent to the boundary.

10 See Ref. 7, E-148, p. 32, of the Bibliography.

11 See Ref. 7

12 A summary of these notes would of a certainty be meaningless unless the summary were to become little more than a direct copy. This, and the fact that a knowledge of the general philosophy of digital computer programing is necessary if the detailed portions of the rest of the report are to be of any interest to the reader, dictate the assumption made.

13 Because the orders in Whirlwind are stored in the same type of register as the data, there are sixteen places available. The first five places are used to designate the order, the last eleven places are alloted to the address number.

14 Boundary values appear in the basic iteration equation whenever $\underline{u}$ is next to any boundary, and hence they must be addressed in the same manner as the rest of the $\underline{u}$'s so that they can be obtained by the process of section 1.1 .

15 See Ref. 3, p 94 ff, of the Bibliography.

16 The stop cycle is actually checking the overall result of the last cycle. Referring to the flow diagram (Appendix II) it becomes evident that the iteration is actually being stopped one cycle after the optimum cycle. Some careful, though non-rigorous, consideration given to this indicates that the decrease in accuracy will be slight. On the other hand, the losses in both storage and operation time are great, amounting to the order of 50% if the same method is used to stop the iteration at the exact cycle of optimum accuracy.

BIBLIOGRAPHY

1 R. Courant, Advanced Methods in Applied Mathematics.

2 Grinter

3 Nat. Res. Council Bull. 92, Numerical Integration of Differential Equations (1933).

4 Southwell

5 L. Fox, Proc. Roy Soc. A190, 31-59 (1947).

6 L. Fox, Quar. J. Mech. & App. Math. 1, 253-279 (1948).

7 Project Whirlwind Publications:
Project Whirlwind, Naval Contract N5ori60, Servomechanisms Laboratory, Massachusetts Institute of Technology

C-93 Notations for Coding

C-94 Example of Coding Procedure

E-148 The Solution of Systems of Linear Algebraic Equations by Successive Approximation

E-161 Code for Solution of Simultaneous Equations by Elimination

E-235 Description of Whirlwind Codes

R-127 The Whirlwind I Computer Block Diagrams

R-165 The Convergence of the Gauss-Seidel Iterative Method

Appendix I

THE CODE

Appendix I

Explanatory Notes

1 Code Symbols: The code symbols are used with the meanings given them in Project Whirlwind engineering note E-235. All other notation is either that suggested in Project Whirlwind conference note C-93 or else it is identical with that used in the body of the Thesis.

2 Storage Register Designations:

The prefix R signifies a program order register.

The prefix B signifies data which does not remain fixed during the program.

> the B 1.n registers store all values of u_{ij} including the boundary values, the actual numerical addresses for these registers being related in the manner described in section 4.1.1 of the Thesis.

The prefix C indicates registers whose contents will not be changed during the iteration process; these values may be stored at any available position.

The registers whose addresses are symbolized by j + c and j + c' are the registers mentioned in sections 4.2.2 and 4.3, respectively, of the Thesis.

N. B. The R, B, j + c, and j + c' registers must all be stored in sequence, as numbered.

ORDER CODE FOR SOLUTION OF

LAPLACE'S EQUATION BY ITERATION

(Input and Output Orders Not Included)

Register No.	Order		Explanation
Last input order	ca	C11	puts RC (first u of first row in AC
(Sec. A 2.1 - Is u_{ij} a Boundary Value?)			
R1	td	B2.4	transfers address of u_{ij} to B2.4
R2	td	B2.5	" " " " " B2.5
R3	sl	n	eliminates all but row no. j, giving j x 2^{n-15} in AC
R4	sr	n	gives j x 2^{-15} in AC
R5	ts	B2.1	transfers j x 2^{-15} to B2.1
R6	ad	C1	gives j plus c, equivalent to RC (address of boundary value for row j)
R7	td	R9	transfers j plus c to address section of order R9
R8	td	R20	transfers j plus c to address section of order R20
R9	ca	(j plus c)	(address from R7); AC contains address of boundary value at end of row j
R10	su	B2.4	ascertains if RC u_{ij} is equal to RC boundary value; (-) if u_{ij} boundary value (+) if " " "
R11	cp	R19	if u_{ij} would be a boundary value program proceeds to next step; if u_{ij} is not a boundary value program goes to order R19

(Sec. A 3.1 - Is u_{ij} the Final Boundary Value?)

R12	ca	C5	puts max. row no. in AC
R13	su	B2.1	ascertains if u_{ij} would be the final boundary value (-) if u_{ij} would be (+) if u_{ij} would not be
R14	cp	R84	if u_{ij} would be the final boundary value program proceeds to next step; if u_{ij} is not final boundary value program goes to order R84

(Sec. A3.2 - Is $S^n > S^{n-1}$?)

R15	ca	B2.3	puts S^n in AC
R16	su	B2.9	(-) if $S^n \geq S^{n-1}$ (+) if $S^n < S^{n-1}$
R17	cp	R90	if $S^n \geq S^{n-1}$ program proceeds to next step if $S^n < S^{n-1}$ program goes to order R90

(Sec. A4 - Output program)

R18	sp	RC first of output orders

The three sections which follow are a cycle resulting from order R11:

(Sec. A 2.2 - Is $u_{(i+1)j}$ a Boundary Value?)

R19	ao	B2.4	gives complement of $u_{i+1,j}$ in AC; stores $u_{i+1,j}$ in B2.4
R20	ad	(j plus c)	(address from R7); ascertains if RC $u_{i+1,j}$ is equal to RC boundary value; (-) if $u_{i+1,j}$ is a boundary value; (+) if $u_{i+1,j}$ is not a boundary value

R21	cp	R23	if $u_{i+1,j}$ boundary value proceeds to next order; if $u_{i+1,j}$ boundary value program goes to order R23

(Sec. A 2.2.1 - "Mark")

R22	ca	C7	puts 2 x 2^{-15}, the "mark", in AC
R23	ts	B2.2	transfers "mark" to B2.2

(Sec. A 2.3 - Combined Basic Iteration and Basic Stop Iteration Cycles)

R24	ca	B2.5	puts address of u_{ij} in AC
R25	td	R49	transfers address of u_{ij} to R49
R26	td	R69	" " " " to R69
R26	ad	C6	forms address $u_{i,j+1}$
R27	td	R34	transfers address
R28	ad	C7	forms address of $u_{i,j-1}$
R29	td	R37	
R30	ad	C8	forms address of $u_{i+1,j}$
R31	td	R41	
R32	ad	C9	forms address of $u_{i-1,j}$
R33	td	R45	
R34	ca	RC $u_{i,j\ 1}$	(address from R27); puts $u_{i,j+1}$ in AC
R35	sr	2	gives $u_{i,j+1}/4$ in AC
R36	ts	B2.6	transfers quantity to B2.6
R37	ca	RC $u_{i,j-1}$	(address from R29); puts $u_{i,j-1}$ in AC

R38	sr	2	gives $u_{i,j-1}/4$ in AC
R39	ad	B2.6	forms $(u_{i,j+1} + u_{i,j-1})/4$
R40	ts	B2.6	
R41	ca	RC $u_{i\ 1,j}$	(address from R31); puts $u_{i\ 1,j}$ in AC
R42	sr	2	divides it by 4 (as in R38)
R43	ad	B2.6	forms sum of two u's summed in R39 and the present $u_{i\ 1,j}$
R44	ts	B2.6	
R45	ca	RC $u_{i-1,j}$	(address from R33); puts $u_{i-1,j}$ in AC
R46	sr	2	divides by 4
R47	ad	B2.6	forms sum of the three u's summed in R43 and the present u_{i-1j}, this sum is equivalent to $(u_{c1} + u_{c2} + u_{c3} + u_{c4})/4$
R48	ts	B2.6	

("stop iteration" sub cycle)

R49	su	RC u_{ij}	(address from R25); puts (sum u_c's)/4 - u_{ij} in AC
R50	sl	2	gives (sum u_c's) - $4u_{ij}$ in AC
R51	ts	B2.7	
R52	ca	B2.5	puts RC u_{ij} in AC
R53	sl	n	eliminates all but jsection of the address of u_{ij}
R54	ad	C4	(-) if $j = 0$ (+) if $j > 0$
R55	cp	R58	if $j = 0$ program proceeds to next order if $j > 0$ program jumps to order R58

R55	ca	B2.7	puts p_{ij} in AC
R56	sr	1	multiplies p_{ij} by $\frac{1}{2}$
R57	ts	B2.7	
R58	ca	B2.5	puts address of u_{ij} in AC
R59	sl	n	eliminates all but j section of address
R60	su	C12	(-) if j < max. row no. (+) if j = " " "
R61	cp	R73	if j < max row no. proceeds to next order if j = max. row no, program goes to R73 (to multiply p_{ij} by $\frac{1}{2}$)
R62	ca	B2.2	puts mark, if any, in AC
R63	ad	C4	(-) if no mark (+) if any mark
R64	cp	R77	if no mark program proceeds to next order if any mark program jumps to R77 to multiply p_{ij} by $\frac{1}{2}$ and reduce the number of marks by one
R65	ca	B2.7	brings p_{ij} with proper coeff. into AC
R66	ad	B2.9	forms partial sum of p_{ij}'s leading to S^n
R67	ts	B2.9	
R68	ca	B2.6	puts (sum u_c's)/4 in AC
R69	ts	B1.n	(address from R25); puts $^{n}u_{ij}$ in place of $^{n-1}u_{ij}$
R70	ca	B2.5	puts address of u_{ij} in AC
R71	ad	C10	gives the address of the new u_{ij} in AC

R72	ap	R1	repeats the operations from the beginning on the new u_{ij}

The section which follows is a sub-sub cycle resulting from order R61

R73	ca	B2.7	puts p_{ij} in AC
R74	sr	1	multiplies p_{ij} by $\frac{1}{2}$
R75	ts	B2.7	
R76	sp	R62	returns program to the sub cycle

The section which follows is a sub-sub cycle resulting from order R64

R77	ca	B2.7	puts p_{ij} in AC
R78	sr	1	multiplies p_{ij} by $\frac{1}{2}$
R79	ts	B2.7	
R80	ca	B2.2	puts "mark" in AC
R81	su	C6	reduces "mark" by one
R82	ts	B2.2	
R83	sp	R65	returns program to the sub cycle

The section which follows is a sub cycle resulting from order R14

(Sec. A 3.1.1 - Generate RC first $u_{i,j+1}$)

R84	ca	B2.1	puts row no., j in AC
R85	ad	C2	gives j + c', equivalent to RC (address of first u_{ij} in row j+1)
R86	td	R87	
R87	ca	j c'	(address from R86); puts address of first $u_{i,j\ 1}$ in AC

R88	td	B2.5	places RC (first $u_{i,j+1}$) in B2.5
R89	sp	R23	starts program on first order of sec. A 2.3

The section which follows results from order R17

(Sec. A 3.2.1 - Prepare to Repeat the Iteration Process)

R90	ca	B2.9	puts S^n, now S^{n-1}, into AC
R91	ts	B2.3	
R92	ca	C11	puts address of first u in first row in AC
R93	sp	R1	restarts process

DATA REGISTERS

B - Registers

B 1.n RC nth u_{ij} (using left-right, up-down scan)

B 2.1	contains the row no., j	;	quantity stored by R5
B 2.2	contains the "mark" this register should initially be set at 1 x 2^{-15} by the Input program.	;	quantity stored by R23
B 2.3	contains S^{n-1}	;	quantity stored by R90
B 2.4	contains B 1.n	;	quantity stored by R1 modified by R19
B 2.5	contains B 1,n	;	quantity stored by R2 or R88
B 2.6	contains (sum of u_c's)	;	quantity stored by action of R36, R40, R44, and R48
B 2.7	contains p_{ij}	;	Quantity stored by R51
B 2.9	contains (sum of p_{ij}'s) which becomes equal to S^n	;	quantity stored by R69

c + j ≡ RC address of boundary value at end of row j

c' + j ≡ RC address of first u_{ij} in row j 1

C - Registers

C1	contains	c	C7	contains	-2 x 2^{-15}
C2	"	c'	C8	"	1 x $2^{a-15} + 2^{-15}$
C3	"	0/000000000000000	C9	"	-2 x 2^{a-15}
C4	"	1/111111111111111	C10	"	1 x 2^{a-15}
C5	"	max. row no., m	C11	"	address first u_{ij}
C6	"	1 x 2^{-15}			

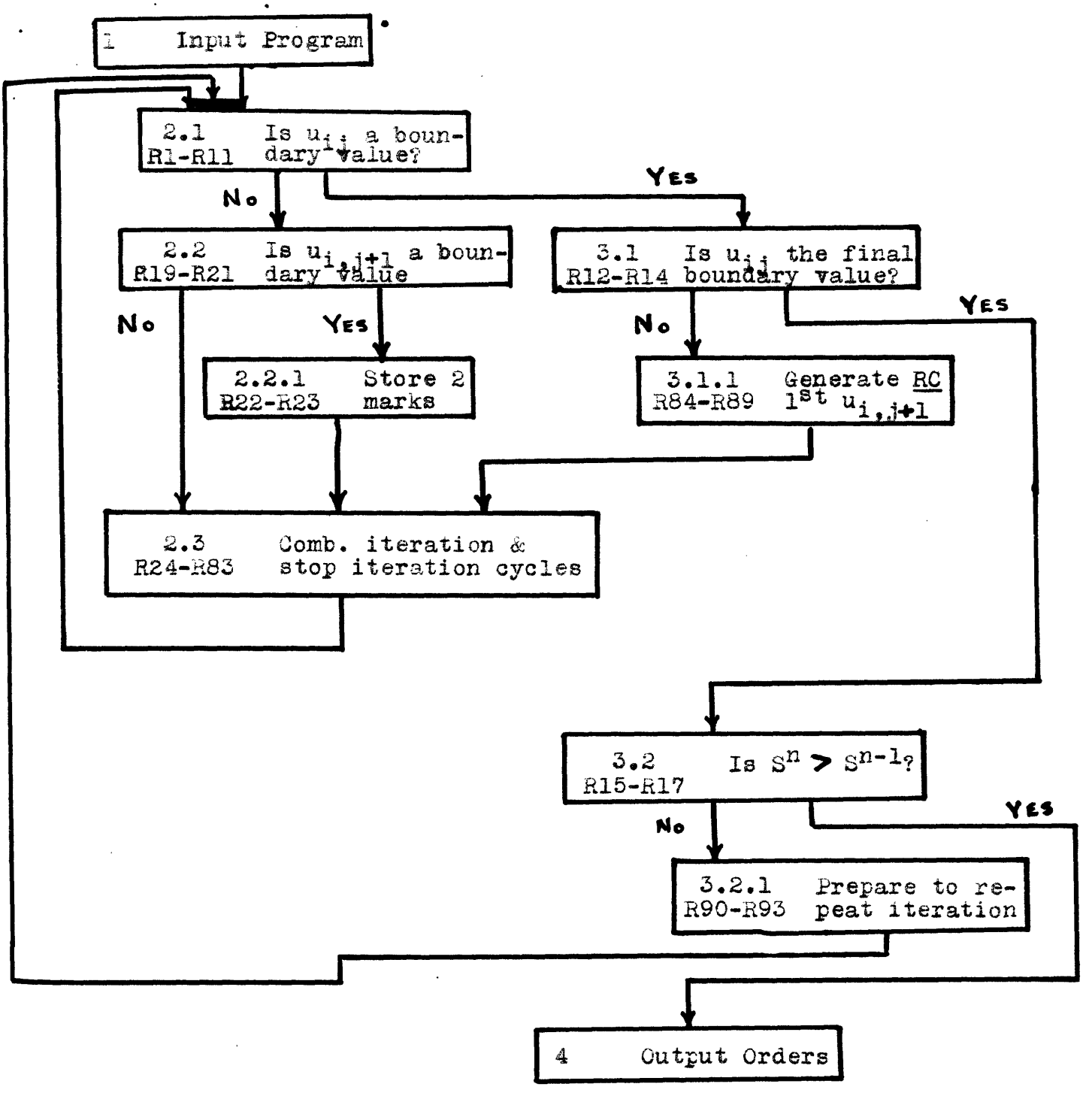
C12 contains (max. row no. - 1) x 2^{n-15}

Appendix II

THE FLOW DIAGRAM

Appendix II

The flow diagrams in this Appendix serve to show the relation between the different parts of the program. The arrows indicate the direction in which the program proceeds. Each box represents some general operation being preformed; the following information is contained in each box (reading from left to right): an outline number, the orders used to complete the operation of the box, and the operation being preformed. Boxes having two lines extending from their lower sides are those in which a conditional program is involved and in which the computer must answer the question asked in the box and then proceed accordingly.



FLOW DIAGRAM