COO-614-54

ICR QUARTERLY REPORT

MASTER

Number 13
May 1, 1967

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

THE UNIVERSITY OF CHICAGO

THE INSTITUTE FOR COMPUTER RESEARCH

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

The Institute for Computer Research

CESTI PRICES

The University of Chicago

HQ \$ 3.00, MN .65

ICR Quarterly Report

Number 13
May 1, 1967

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

Prepared under the auspices of AEC Contract No. AT(11-1)-614.

Victor H. Yngve, Principal Investigator

PATENT REVIEW APPROVES RELEASE. PROCEDURES
GOVERNING PATENT REVIEW AND RELEASE ARE ON
FILE IN RECEIVING SECTION.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Staff of the Institute for Computer Research

Naomi Alton

Robert Ashenhurst

Jurgen Bounin

Dolores Brennan

Kenneth Burns

Rachel Chakrin

Elaine Chow

David Eicher

Charles Fischer

Etta Franklin

William Georgas

Robert Graves

Richard Herwitt

Leonardo Herzenberg

Paul Kosinski

Hirondo Kuki

George Kreisheimer

Vincent Kruskal

John Makcoviak

Glenn Manacher

Richard Miller

William Mixon

Lily Monheit

John Munn

Nonyne Nelson

Michael Neumann

Daniel Norman

Alex Orden

Walter Petryshyn

Charles Robinson

Clemens Roothaan

Lyle Settle

John Shepherd

Cheng-kiat Teng

Serena Torres

Nancy Trombetta

Linda Glennie

Michael Williams

Victor Yngve

Table of Contents

Section I

- A. Maniac III Engineering Activity
- B. The Man-Machine Project (C.C.J. Roothaan)

Section II

- A. The Unclass List-Processing Language: A Preliminary Description
(G.K. Manacher and R.B.K. Dewar)

Section III

- A. On the Approximation Solvability of Nonlinear Equations (W.V. Petryshyn)
- B. A Bidirectional Simplex Algorithm (A. Orden and V. Nalbandian)

MANIAC III ENGINEERING ACTIVITY

1. Maniac III

The real-time trap facility of Maniac III has been activated in a simple form and is in use for communication between Maniac III and the PDP-8 or 360-30.

2. Interface Computers

The scratch-pad memory has now been fully tested in operation under the control of the prototype interface computer.

3. PDP-8 Computer

The second magnetic tape unit has now been connected and is available for use.

4. Pattern Manipulator

The SEC Vidicon tube referred to in Section III-B of ICR Quarterly Report No. 11 has been received, and work on this project has commenced.

5. General

Much of the time of the engineering staff has been occupied in discussion of possible interface computer specifications. Until some decisions are recorded on the interface computer register structure and instruction set, little further work can be done on these devices.

The engineering staff has therefore been working on utility programs for the PDP-8, a symbolic editor, and multicomputer system simulator.

MANIAC III USE

January 30 - July 14

(Programmer and production use; debugging and production times are not separated in this table.)

Systems Work	207.5
Engineering Computer Time	24.0
Language Studies (Robinson)	97.2
Investigations of Projective Planes (Manacher)	20.0
Student Time	217.8
Miscellaneous	<u>26.5</u>
Total	553.4 hr.

THE MAN-MACHINE PROJECT*

C. C. J. Roothaan

Progress Report No. 9

(for the period 1 January 1967 through 31 March 1967)

System/360 Mathematical Subroutines

We have offered to revise the System/360 mathematical subroutines (programmed by Mr. Kuki and others under an earlier contract with IBM) so that they take advantage of forthcoming hardware changes (in particular, the new guard digit for long form floating-point arithmetic).

Scientific Subroutine Package for System/360

Mr. Kuki has analyzed several subroutines in the recently released System/360 Scientific Subroutine Package. The principal aim of this investigation is to improve speed, accuracy, and documentation wherever possible and to evaluate the overall merits of individual subroutines in the package.

One Computer ASP

Messrs. Kruskal, Kosinski, and Williams have been studying the functional characteristics of IBM's recently released ASP system with a view towards constructing a similar operating system for a stand-alone

*Work on this project, supported by a grant from the International Business Machines Corporation with C. C. J. Roothaan as principal investigator, is carried on under the joint auspices of the Institute and the Computation Center.

computer. The crucial issue, about which a decision will soon be made, is whether it is possible to define a data structure which is useful for standard OS/360, which fulfills the particular needs for input/output files under ASP, and which also provides convenient access for conversational file manipulation by users at remote consoles.

POOR (Page Oriented Output Routine)

Subroutines, one level above POOR, are being designed to facilitate conventional output. In particular, we will provide for printing large matrices, for dumping, and for interpreting conventional format statements.

7094 POOR: Professor Roothaan and Mr. Sherrard have added editing and centering features. These are essentially debugged. The higher level routines are being worked on. Mr. Eicher is preparing a full writeup for users.

360 POOR: Programmed by Mr. Sevcik, the first version is working and has been used extensively in several applications. Editing and centering features are being designed as are also higher level routines.

CHAP (CHaos Assembly Program)

Miss Monheit has continued work on the assembler which produces correct absolute code. The loader for relocatable code (being worked on) will fit programs into memory segments acquired from the CHAOS buffer pool. CHAP and the loader will probably be finished in the next three months. This will enable us to provide a library of flexible, fully multiprogrammed utility functions which can be run simultaneously with CHAOS.

Task Scheduler and Setup Processor

Mr. Kubaitis has continued with the Task Scheduler and Setup Processor. Both are being debugged. When these features are working, all tasks (including those requiring operator setup) will be scheduled by CHAOS.

THE UNCLLL LIST-PROCESSING LANGUAGE: A PRELIMINARY DESCRIPTION

G.K. Manacher and R.B. K. Dewar

Abstract

For some time it has been apparent that there is a need for a powerful, general, low-level list language, which will at once permit the programmer to manipulate lists in a highly compact, dynamic way, and at the same time to generate code that is as efficient, or nearly as efficient, as code he would generate by hand.

"Low-level" means several things: The control of data structures should be microscopic and precise; the basic operations should correspond to machine instructions; the syntax should permit complex constructions that will compile into very efficient code. Furthermore, the programmer should find that he is never obliged, out of convenience or necessity, to employ a device in the language that will result in inefficient generated code.

The language described herein satisfies these requirements, yet it is written in a supple, convenient, "high-level" syntax. It requires a minimum of precise data declarations. It is reminiscent both of a substantial subset of ALGOL and of L6¹. Its data elements are very few in number, consisting of a few literals and of L6-type list elements or fields. The rule of association insures that a minimum of operands will be computed in a Boolean expression. Access to fast registers is easy.

Basic facilities include fixed- and floating- point arithmetic, Boolean arithmetic, and bit-manipulation operators. The function mechanism, though quite powerful, is extremely low-level; but the difference between this mechanism and that of LISP and ALGOL are so specialized that they are for most applications irrelevant. Analogues of pass-by-address, pass-by-subroutine-name, and

¹The present document assumes some familiarity with reference 1.

II A-2

SNOBOL-style function-failure exit are also incorporated. Finally, a set of special precoded functions are included which permit quick access to the space-allocation and function machinery.

This language, we believe, represents the most efficient approach to list processing available. Its generality makes it a useful tool in compiler construction, network and flow analysis, theorem proving, graphics, etc. In addition, it appears suitable as a target language for higher level special-purpose languages.

Implementations for the IBM 7094 and 360 are in progress; both versions will run under IBM systems. The implementation techniques may be generalized to other machines.

The following conventions are used in this description:

Capital letters and special characters represent text which actually appears in the program. Small letters indicate a general type of terminal.

Square parentheses indicate a section of a construction which is optional.

The comma has been used as the break character in the description, but in most implementations blank is an acceptable alternative. (Comma is required however in the MACRO FAP implementation). In any case the parenthesis also acts as a break character so that many commas generated by this description are not needed in actual use: e.g., (A, +, (C, *, D), /, (\$GHJ)) may be written (A, +(C, *, D)/(\$GHJ)).

q, q_1, q_2 always refer to a quantity that may be a literal, field, bug, expression or function call.

c, c_1, c_2 refers to a 'q' whose value can be changed, i.e., a bug or field.

1. Basic Structure of UNCLL-Statements.

A UNCLL statement consists of three parts:

The first part is an optional label to be used as a transfer name etc.

The second part identifies the type of statement.

The third part is a series of clauses and control words. The clauses are either expressions, gotos or function calls and are represented in the following

cls.

There are several basic types of statement:

The FENTRY and related statements are described in detail under
FUNCTION DEFINITION.

The THEN statement is of the form

THEN cls, cls, cls, cls...

The cls are evaluated in sequence from left to right. The last only
may be a goto, which consists of a label optionally surrounded
by parentheses. In the case of an expression, the primary
operator is usually an assignment operator. If it is not, an
assignment to the left is implied: thus

THEN (A, +, BAD) means THEN (A, =, A, +, BAD)

There are four conditional statements:

IFALL cls₁, cls₁, cls₁,... THEN, cls₂, cls₂... [ELSE, cls₃, cls₃]

IFANY

IFNONE

IFNALL

The cls₁ are a series of tests interpreted as follows: A zero value
is taken as false, and non-zero as true. Thus Boolean expressions
have their natural truth values.

The cls₁ are thus a series of conditions, the statement type gives the
logical connective to be applied (IF ALL are true, IF ANY are
true, IF NONE are true, IF Not ALL are true).

If the connective is satisfied, the sequence of clauses after the THEN
and before the ELSE is executed in the manner of a THEN type

statement. Control then passes; in the absence of a goto, to the next line.

If the connective is unsatisfied, the sequence of clauses after the ELSE (if present) is executed in the same way.

T may be used in place of THEN and E in place of ELSE.

NOTE: This means that the labels T, THEN, E, ELSE may not be used. The THEN (or T) may be optionally omitted if the THEN field consists of a single goto. The ELSE (or E) may optionally be omitted if the THEN field ends with a goto.

Note that the 'tests' are evaluated left to right; as soon as the outcome is known, the evaluation of tests stops: e.g. in an IFNALL statement, control goes to the THEN section as soon as false condition is found.

Note that the commas between the clauses may usually be omitted since most clauses are surrounded with parentheses.

IF is a synonym for IFALL.

IFNOT is a synonym for IFNONE.

A blank statement type identifier is synonymous with THEN.

The OPTION statement

This has the form

OPTION opt 1, opt 2, opt 3...

opt 1, 2, 3... are the names of options being taken some of which may depend on the implementation.

Examples: NAME (see under fields)

ROUND (causes 7094 floating point operations to be rounded)

2. Bugs and Fields

Bugs

Bugs have names consisting of a letter followed by one or more numbers. An arbitrary number of bugs may be used. A bug is defined by its appearance in an implicit or explicit assignment or in a function call by address. Bugs are local to a single assembly, i. e. global to all functions within one assembly. Depending on the implementation, there is a facility for making bugs global to several assemblies. Another implementation dependent feature allows special fast registers, such as index registers, to be used as bugs. Such fast bugs differ from other bugs in that they are global to all assemblies.

Fields

Fields are specified by concatenating a bug name with one or more field names.

A field name consists of a letter optionally followed by one or more numbers. Two operations are available for defining fields at compile time, these definitions being static. This is the `field` and `field` operators.

name DEF word, offset, length

name is the name of the field

offset is a left offset in bits

length is the length in bits

word is the word in the block

name DEFV offset, length

defines a vector of fields and is equivalent to

name' 1 DEF 1, offset, length

name' 2 DEF 2, offset, length

name' 3 DEF 3, offset, length

and so forth.

In addition name will be a synonym for name'1.

Note that length is less than or equal to one word, but that fields may overlap word boundaries, however such fields are relatively slow.

The fastest fields are those corresponding to hardware instructions.

Runtime fields

Defining fields at compile time allows the compilation of highly efficient code. However it is occasionally useful to define fields in a dynamic manner at execution time. Another advantage of run-time definition is that it results in more compact code (at the expense of speed). To define a run-time field the declarative statement

name DEFR

or

DEFR (name, name, name,...)

is used, where name is the name of the field. Before executing a statement involving the field, it must be compiled by calling the DEFLD function (see system functions).

The separator '-' may be used in place of concatenation if desired; its use will frequently improve program readability.

Examples of fields:

AB1

A3B24C2

ADF

W2-X3-A4

Note: A compile time option (NAME) allows the use of arbitrary names beginning with an alphabetic for bugs and fields. If this option is taken the separator character must be used.

3. Literals

Decimal literals (floating point)

These are numbers with a decimal point and possibly an exponent; they may not start with a zero.

examples: 4.5 .01 76.32E10 +4.0 -3.14159 .0

Decimal literals (fixed point-integers)

These are integers not starting with a leading zero and possibly followed with a binary point place, which gives the bit number which is to correspond to the rightmost bit of the literal.

examples: 24 12B35 6B34 (these are identical on a 36-bit machine)

-1 -56 +19

Octal literals

These are octal integers preceded by a zero and unsigned.

examples: 077777 0156 0

NOTE: Octal 0 is equal to decimal 0.

Hollerith literals

These consist of a series of characters surrounded by quotes.

A quote is represented by two quotes; thus "" is the literal quote. The resulting literal is right justified characters, left filled with zeros.

examples: 'A'B' ',, ',0' 'A' '876' '00('

Pointer literals

These consist of = followed by a location, bug or function name, or the name of a cell used in any other part of the assembly; their value is a pointer to the cell. They are most often used in conjunction with indirect function calls and formatted I/O which requires format pointers.

examples: =LOC1 =DEFLD

NOTE: fast index bugs and parameters and locals in functions have no address.

4. Function Calls

Function calls are enclosed in a level of parentheses. There are two types of call:

Direct call

(\$name [, q, q, q][, /fail])

name is the explicit name of the function, and

q are the arguments. If the call is by value (true of all system and most programmer functions) then q may be any quantity (literal, function call, field, expression). Normally the call is by value, however if a bug or field is preceded by a * (e. g. as in (\$A, B, *CA)) then the final value of the corresponding local in the function is copied back on return. This is termed a call by address although it is not strictly one.

fail is the location to which control is to be transferred if the called functions fail. If this is not present and the function fails, a fatal error occurs.

Indirect call

(\$*fld[, q, q, q, ...][, /fail]).

fld is a field or bug containing a pointer to the function to be called, typically generated by an '=' literal. This feature allows the name of a function to be passed into another function. The effect of this call is exactly as if the call had been explicit.

q and fail have the same meaning as in the direct call.

All functions return a value (zero by default if none is specified); however, this value is ignored if the function is used at the outermost level in a sequence of operations.

5. Function Definitions

Function definitions have the form:

```
name      FENTRY [(1,1,1,1,1 . . . .)]
          |
          |
          |
```

[sname SENTRY]

[RETURN expr]

[sname SENTRY]

[label RETURN expr]

ENDF [expr]

name is the name of the function.

p, p are the names of the recursively saved locals. As many of

these as correspond with the parameters in the call are

initialized with the values of the calling parameters. The

rest are not initialized to a set value, and may be used for

scratch.

sname specifies a subentry point with the same l's

Two kinds of call are available: call by value and call by address.

In the latter case the argument in the call is preceded by a '*'. The final value of the l (which may be freely changed during the function) replaces the value of the calling parameter on return.

In the call by value, where the calling parameter has no '*', the final values of the l's are of no significance to the caller.

The mechanism for function calls is similar to that used by OEDIPUS.²

RETURN [expr] returns the value of expr as the result of the function and goes up a level. Zero is returned if expr is missing.

The simple goto RETURN also goes up a level returning 0. The goto FAIL goes up a level and takes the failure return.

ENDF is the same as RETURN but also delimits the function definition block.

The function structure is static in that all references to bugs other than the locals of the current function are to global values which remain unchanged by any call.

Example: FCT - the factorial function

```

FCT  FENTRY  (A)
      IF      (A,GT,1) THEN (R2)
      RETURN  1
R2   ENDF    (A,*, $FCT(A, -, 1))

```

Another way of defining this function is:

```

FCT  FENTRY  (A)
      ENDF    (1, IF(A, EQ, 1) ELSE, A, *, $FCT(A, -, 1))

```

An example of a function normally called by address: The XCHM function interchanges its arguments' values if the first is less than the second.

```

XCHM FENTRY  (A2, A3)
      IF      (A2, GT, A3) T(RETURN) E(A2, IC, A3)
      (RETURN)
      ENDF

```

NOTE: functions may not contain goto's outside their scope.

6. Expressions

Expressions have the general form:

(operand1, operator[, operand2])

Thus all operators are infix. There are no prefix operators, although several operators have no right argument. This means that there need be no restrictions on the names of operators. Each operator places its own restrictions on the operands; these are described in detail for each operator.

The parentheses surrounding the expression may be omitted if the expression occurs as the right argument of another expression. This is also true of function calls. This leads to strict right association, thus:

(A, +, B, *, C, -, D) means (A, +(B, *(C, -, D)))

(A, +, \$BC, Q) means (A, +(\$BC, Q))

Some operators known as assignment operators, result in side effects themselves. Thus, (A, =, 78) changes the value of A. If one of these operators is used in a context requiring it to return a value, the value returned is that of the first operand after the operation has taken place.

For example, multiple replacement may be written:

(A, =, BWW, =, 077777)

7. Operators

A. Boolean Operators

These operators test some condition, then result in a return of 1

if the condition is true, and 0 if false. None of them change the values of their arguments.

Numeric Relational operators

All these operators take as arguments fixed or floating numbers as arguments (the mode on both sides must be the same); the arguments are in the form of literals or fields or expressions or function calls.

(q_1, EQ, q_2)	q_1 is <u>E</u> Qual to q_2
(q_1, NE, q_2)	q_1 is <u>N</u> ot <u>E</u> qual to q_2
(q_1, GT, q_2)	q_1 is <u>G</u> rea <u>T</u> er than q_2
(q_1, GE, q_2)	q_1 is <u>G</u> reater than or <u>E</u> qual to q_2
(q_1, NG, q_2)	q_1 is <u>N</u> ot <u>G</u> reater than q_2
(q_1, LT, q_2)	q_1 is <u>L</u> ess <u>T</u> han q_2
(q_1, NL, q_2)	q_1 is <u>N</u> ot <u>L</u> ess than q_2
(q_1, Z)	q_1 is <u>Z</u> ero
(q_1, NZ)	q_1 is <u>N</u> on- <u>Z</u> ero

Pointer comparison operators

These operators deal with pointers, in most implementations PT is the same as EQ but PT is preferred.

(q_1, PT, q_2)	q_1 <u>P</u> oints to the same block as q_2
(q_1, NP, q_2)	q_1 does <u>N</u> ot <u>P</u> oint to the same block as q_2

Logical operators

The arguments to these operators are Boolean quantities; nonzero means true, zero means false. Most often these arguments will be other

Boolean expressions but they may be any quantities.

(q_1, OR, q_2) q_1 OR q_2 (or both) is (are) true

(q_1, NOR, q_2) neither q_1 NOR q_2 is true

(q_1, AND, q_2) q_1 AND q_2 are both true

(q_1, NTB, q_2) q_1 and q_2 are NoT Both true

(q_1, XOR, q_2) q_1 eXclusive OR q_2 is true

(q_1, LEQ, q_2) q_1 is Logically Equivalent to q_2 (both true or both false)

(q_1, NOT) q_1 is NOT true

NOTE: With some of these operators (e. g. OR), the result may be evaluated from only one operand. All these operators evaluate their LEFT operand first and then the second operand is evaluated only if necessary.

Bit vector comparison operators

The arguments for these operators are considered as vectors of bits (up to a full word long).

(q_1, BE, q_2) The Bit vectors q_1, q_2 are Equal (identical)

Note: due to multiple representations of numbers

(eg +0 and -0 on the 7094) this is not the same as EQ.

(q_1, BN, q_2) The Bit vectors q_1, q_2 are Not equal

B. Arithmetic Operators

These operators give an arithmetic result.

Fixed Point (Integer)

Any length arguments may be used but only full words can hold negative integers. The result of storing a negative integer in a smaller field is not defined.

$(q_1, +, q_2)$	the sum of q_1 and q_2
$(q_1, -[, q_2])$	q_1 minus q_2 ; if the right argument is missing just the negative of q_1
$(q_1, *, q_2)$	the product of q_1 and q_2
$(q_1, /, q_2)$	q_1 divided by q_2 (integer part)
(q_1, MOD, q_2)	the remainder of q_1 divided by q_2

Floating Point

Only full word operands can be used with floating point operations.

$(q_1, ++, q_2)$	the floating sum of q_1 and q_2
$(q_1, --[, q_2])$	q_1 minus q_2 ; if the right argument is missing, just the negative of q_1
$(q_1, **, q_2)$	the product of q_1 and q_2
$(q_1, //, q_2)$	q_1 divided by q_2

Pointer Arithmetic

These operations should always be used for pointer arithmetic, rather than $+$ and $-$ since the hardware addressing may be in units other than L6 words.

$(q_1, I[, q_2])$	the pointer obtained by Incrementing q_1 by q_2 words
$(q_1, D[, q_2])$	the pointer obtained by Decrementing q_1 by q_2 words

If the right argument is missing 1 is assumed.

C. Bit Vector Operators

These operators treat their arguments as vectors of bits, extended left with zero bits to full word length.

(q_1, BO, q_2)	<u>Bitwise Or</u> of q_1, q_2
(q_1, BA, q_2)	<u>Bitwise And</u> of q_1, q_2
(q_1, BX, q_2)	<u>Bitwise eXclusive or</u> of q_1, q_2
(q_1, L, q_2)	The contents of q_1 <u>Left</u> shifted by q_2 bits
(q_1, R, q_2)	The contents of q_1 <u>Right</u> shifted by q_2 bits
(q_1, C)	The bitwise <u>Complement</u> of q_1

Any of the 16 logical operators is available as follows:

Construct a truth table:

LEFT	RIGHT	RESULT (e. g. for AND)
0	0	0
0	1	0
1	0	0
1	1	1

The name of the operator is 0001.

D. Transfer Operator

(q, GO) GO to the address contained in q (Note that q is not a label)

E. Assignment Operators

$(c, =, q)$ The value of q is stored in c .

(c, P, q) c is made to point where q points.

(c_1, IC, c_2) the contents of c_1, c_2 are InterChanged

Note that these operators give the changed result of the left operand as a value if used at lower levels.

Thus

$(A, =, 1)(B, =, 2)(C, =, A, IC, B)$

results in C having the value of 2.

F. Conditional Operators

$(q_1, IF, q_2, ELSE, q_3)$

or (q_1, IF, q_2, E, q_3)

These two operators are always used in conjunction as shown

q_2 is treated as Boolean. The result is q_1 IF q_2 is true (non-zero) or ELSE q_3 if q_2 is false (zero). Only one of these is evaluated.

Note that if q_2 is an expression it must be parenthesized.

8. System Functions

These functions are part of the system and may be called in the normal manner described under function calls.

Storage Allocation Functions

$(\$BLK, size)$	Allocates a <u>B</u> lock of length size, 'returns' a pointer to the block. More complex calls are also available.
$(\$REL, pointer, size)$	<u>R</u> ELeases to the allocator the block of size <u>size</u> pointed to by <u>pointer</u> , returns 0.
$(\$SIZE, pointer)$	Returns the <u>S</u> IZE of the block pointed to by <u>pointer</u>

(\$CLEAR, p, p, p...) CLEARs storage by returning all blocks
to the allocator except those pointed to by

p, p, p, returns 0

(\$AVAIL, size) Returns the number of blocks of length size,
which are AVAILable after splitting larger
blocks.

Note that the allocator can allocate blocks of any size, and that the
entire area of the block may be used by the program.

Functions for manipulation of function calls etc.

(\$TRACE, fpointer) Initiates a diagnostic TRACE of the function
pointed to by fpointer, returns 0.

(\$STOPTR, fpointer) STOPs the diagnostic TRace initiated by
\$TRACE, returns 0.

(\$LEVTO, fname) Jumps up function LEVELs until the current
function name is that given by the Hollerith
literal fname. A name of zeros causes return
to the highest level. There is no returned
value.

(\$LEVUP, nlevels) Jumps UP nlevels LEVELs of function call.
Thus (\$LEVUP, 1) is equivalent to RETURN.

(\$LEVEL) Returns the current function level count (not
including itself, thus (\$LEVEL) at the highest
level returns zero)

(\$DATE)	Returns today's date in a form depending on the implementation.
(\$TIME)	Returns the time of day.
(\$NONES)	Returns the number of one bits in its argument.
(\$DLS, arg1, arg2, nbits[,sizearg2])	<p><u>Double Left Shift</u></p> <p><u>arg2</u> is a bit vector of length <u>sizearg2</u> (a full word is assumed if this argument is missing). It is positioned to the right of <u>arg1</u> and the double quantity is left shifted by <u>nbits</u> bits. The result is arg1 shifted left with the leftmost bits of arg2 having entered at the right.</p>
(\$DRS, arg1, arg2, nbits[,sizearg1])	<p><u>Double Right Shift</u></p> <p>This is like DLS except that the shift is to the right and the size of arg1 is to be given.</p>
(\$EXP, A, B)	Returns the floating value <u>A</u> to the power of the floating value <u>B</u> .
(\$SQRT, arg)	Returns the square root of floating argument <u>arg</u>
(\$MAX, a, a, a...)	Returns the maximum value of the <u>a's</u> , which are either all floating point or all integers.
(\$MIN, a, a, a...)	Returns the minimum value of the <u>a's</u> .
(\$FIX, a)	Returns, as an integer, the integer part of the floating point number a.

(\$FLT, a)

Returns the value of the integer a converted to floating point.

Field Definition Functions

(\$DEFLD, name, word, offset, length)

DEfine (i. e. compile) run time FiELD

name is a pointer literal to the name of the field

word, offset, length have the same meanings as in the DEF declaration

(see compile time fields)

(\$SAVFD, name)

SAVe then FiELDefinition of the field pointed to by

name on a special stack used only for this purpose.

(\$RESFD, name)

REStore the FiELDefinition of the field pointed to by name from the last entry of the pushdown.

Note on the definition of fields

The DEFR declaration results in the reservation of a block large enough for any field compilation, the first cell of this block is given the name of the field so that a pointer literal points to the block.

Input-Output Functions

These are described in detail for a particular implementation; in general they will give access to the Fortran formatted and unformatted I/O available as well as allow several applications, such as automatic I/O of linked lists, particularly appropriate to L6.

Diagnostic Print Functions:

\$(LDUMP)	Dumps all the blocks in use as well as other information such as Bug contents, push down levels, etc. Returns the number of blocks in use.
(\$BDUMP)	Like LDUMP but dumps only those blocks pointed to by bugs. Returns the number of such blocks.
(\$PRBLOK; pointer)	Dumps the block pointed to by <u>pointer</u> . Returns 0.
(\$PRSTR, pointer, nam1, nam2...)	<u>P</u> Rints the <u>S</u> T <u>R</u> ucture defined as follows, <u>pointer</u> points to a block, whose fields of <u>nam1</u> , <u>nam2...</u> point to other blocks. All blocks in the structure are dumped. Loops do not cause duplicate printing.

This list of functions is not intended to be complete, but rather to serve as a model for the kind of features to be implemented.

9. Definition of Operators

An additional feature whose syntax has not yet been fixed will be the facility for definition of new operators as open macros.

10. Acknowledgements.

We wish to thank Messrs. K. C. Knowlton, M. D. McIlroy, W. S. Brown,

V. Kruskal, and J. Pomeranz for insightful comments that led to many of the design and implementation features of the language.

REFERENCES

1. Knowlton, K.C., A programmer's description of L⁶, Comm. ACM., Volume 9, No. 8, August 1966, pps. 616-625.
2. Brown, W.S., An operating environment for dynamic-recursive computer programming systems, Comm. ACM, 6, (1965), 371-377.

Addendum

I. Let any one of IFANY, IFALL, IFNONE, IFNALL be represented by CST.

The forms

[label]

CST $cls_1, cls_1, \dots$ THEN

BEGIN

.

(A)

ENDC

CST $cls_1, cls_1, \dots$ THEN, $cls_2, cls_2, \dots$ ELSE

BEGIN

.

.

.

(B)

ENDC

and

CST $cls_1, cls_1, \dots$ THEN

BEGIN

.

.

.

ENDC

ELSE

BEGIN

.

.

.

(C)

ENDC

will define compound statements bracketed by BEGIN and ENDC. Each compound statement is a sequence of UNCLLL statements written in the usual fashion; moreover, compound statements may appear inside one another. The restrictive syntax in which compound statements may appear makes their meaning clear; for instance, in form A, the compound statement is executed when the predicate CST is satisfied, and skipped over if it is not.

If it is convenient to write a compound statement after T but the corresponding "ELSE" can be written simply, the form

```
CST   cls1, cls1, ... T
```

```
BEGIN
```

```
.
```

```
.
```

```
.
```

```
ENDC
```

```
ELSE  cls2, cls2, ....
```

will be allowed. A compound statement after ELSE is thus permitted only when ELSE stands alone.

II. There will be provision for hexadecimal literals on machines for which these are appropriate.

III. It should be noticed that the provision of the GO indirect-transfer operator allows the construction of closed parameterless subroutines within functions that can communicate thereby with the function's parameters.

W. V. Petryshyn

Introduction. Let X and Y be two normed linear spaces and let $\{X_n\} \subset X$ and $\{Y_n\} \subset Y$ be two sequences of finite dimensional subspaces. Let A be a (possibly) nonlinear mapping of X into Y , P_n a mapping of X into X_n and Q_n a mapping of Y into Y_n . Let $\{A_n\}$ be a sequence of mappings of X_n into Y_n defined by $A_n = Q_n A|_{X_n}$.

As in our recent papers [13, 14] we consider here the problem of giving a constructive proof of the existence and the uniqueness of solutions of the equation

$$(1) \quad Ax = f, \quad f \in Y,$$

as strong limits of solutions of $x_n \in X_n$ of the approximate equations

$$(2) \quad A_n x_n = Q_n(f)$$

without the assumption (used in [14]) that X and Y are reflexive Banach spaces. To make the nature of our problem more precise we need the following definitions (see also the papers of the author [13, 14] and Browder [4, 5]).

Definition 1. A quadruple of sequences $(\{X_n\}, \{Y_n\}, \{P_n\}, \{Q_n\})$ is called an approximation scheme for the Eq. (1) if for each n the dimension of X_n equals the dimension of Y_n , the mappings P_n and Q_n are continuous, and for each x in X and y in Y , $P_n x \rightarrow x$ and $Q_n y \rightarrow y$ (where we use $\rightarrow$ to denote the strong convergence in either X or Y according to context).

Definition 2. Eq. (1) is said to be uniquely approximation-solvable with respect to a given approximation scheme $(\{X_n\}, \{Y_n\}, \{P_n\}, \{Q_n\})$, if there exists an integer $N > 0$ such that, for each $n \geq N$ and each f in Y , Eq. (2) has a unique solution x_n in X_n , $x_n \rightarrow x$ in X and x is the unique solution of Eq. (1).

Note that the mappings P_n and Q_n are not assumed to be projections or even linear though for most of the direct methods (2) in the approximate solution of Eq. (1) they are linear mappings induced by any one of the following groups of methods: the projection or Galerkin type methods (for references see [13]), the collocation methods (see, for example, Vainikko [15]), and the finite difference methods (assuming, as in Lax-Richtmyer [7], that exact and approximate solutions belong to the same space X). Furthermore, as will be seen from the proof of Theorems 1 and 2 below, P_n and Q_n need not even be continuous.

We note also that if $X = Y$ is a reflexive Banach space, $X_n = Y_n$ for each n , and $P_n = Q_n$ is a linear projection, Definitions 1 and 2 defining the concept of a strong projectional-solvability of Eq. (1) were first given by the author [13] while in their present extended form they were used in [14]. Let us add that a slightly different formulations of Definitions 1 and 2 were first introduced by Browder [5] (for somewhat similar approach see also Aubin [1]).

We say that the operator A satisfies condition (C) if it has the following property:

Let $(\{X_m\}, \{Y_m\}, \{P_m\}, \{Q_m\})$ be an arbitrary approximation scheme for Eq. (1). If $\{x_m\}$ is any sequence in X with $x_m \in X_m$ such that

$x_m \rightarrow x$ (i.e., x_n converges weakly to x) in X and $A_m x_m \rightarrow g$ for some
 g in Y , then $Ax = g$.

In [13] the author proved that if $X(=Y)$ is a reflexive Banach space and $P_n(=Q_n)$ is a linear projection, then under certain conditions on A (to be given below) Eq. (1) is strongly projectionally-solvable provided that A satisfies condition (C). In [4, 5] Browder showed that if, instead of condition (C), we assume the existence of a solution x in X of Eq. (1) for any given f in Y , then somewhat weaker assumptions on the spaces X and Y and the operator A ensure the unique approximation-solvability of Eq. (1). In [14] the author investigated the relation between condition (C) and the condition of the solvability of Eq. (1) (i.e., condition (S)) and has shown that at least when X is reflexive the assumption of condition (C) is equivalent to the assumption of condition (S).

The purpose of this paper is to show that a modification of condition (C), which we call condition (H), allows us to derive the necessary and sufficient conditions for Eq. (1) to be uniquely approximation-solvable without the assumption that X and Y are reflexive Banach spaces. At the same time we prove the equivalence between our condition (H) and the solvability condition (S) for general normed linear spaces X and Y .

Thus it is shown that the class of nonlinear functional equations (1) satisfying the solvability condition (S) used by Browder [5] is in fact the same as the class satisfying our condition (H). When X and Y are also assumed to be reflexive Banach spaces, we show that, under our conditions on A , conditions (C), (H) and (S) are all equivalent. Finally, a number of other

results for equations perturbed by completely continuous operators are also obtained.

1. Approximation-solvability. The main result of this section is the following theorem.

Theorem 1. Let X and Y be two normed linear spaces and let $(\{X_n\}, \{Y_n\}, \{P_n\}, \{Q_n\})$ be an approximation scheme for Eq. (1). Suppose that there exists an integer $N > 0$ and a continuous strictly increasing function $\alpha(r)$ from $R^+ = \{r | r \geq 0\}$ to R^+ with $\alpha(0) = 0$ and $\lim_{r \rightarrow \infty} \alpha(r) = \infty$ such that A_n is continuous in X_n for each $n \geq N$, $Q_n A P_n x \rightarrow Ax$ for each x in X and

$$(3) \quad \|A_n x - A_n y\| \geq \alpha(\|x - y\|) \text{ for each } n \geq N \text{ and all } x \text{ and } y \text{ in } X_n,$$

Then Eq. (1) is uniquely approximation-solvable if and only if the mapping A satisfies the following condition (H): If $(\{X_m\}, \{Y_m\}, \{P_m\}, \{Q_m\})$ is an arbitrary approximation scheme for Eq. (1) and $\{x_m\}$ is any bounded sequence in X with x_m in X_m such that $A_m x_m \rightarrow g$ for some g in Y , then there exists a subsequence $\{x_{m_i}\}$ of $\{x_m\}$ and an element x in X such that $x_{m_i} \rightarrow x$ and $A_{m_i} x_{m_i} \rightarrow Ax (=g)$, as $m_i \rightarrow \infty$.

Proof. (Uniqueness.) Suppose that $u \neq v$ and $Au = Av$. Because, for each u and v in X , $P_n u$ and $P_n v$ lie in X_n , (3) implies that for each $n \geq N$

$$\|Q_n A P_n u - Q_n A P_n v\| \geq \alpha(\|P_n u - P_n v\|).$$

Since $P_n u \rightarrow u$, $P_n v \rightarrow v$ and $Q_n A P_n x$ and $Q_n A P_n x \rightarrow Ax$ for each x in X , the passage to the limit in the above inequality implies that $0 \geq \alpha(\|u - v\|)$. The

latter inequality contradicts the properties of $\alpha(r)$ and shows that A is one-to-one.

(Existence). By (3) and our assumptions, A_n is a one-to-one continuous mapping of X_n into Y_n for each $n \geq N$, and hence, by the Brouwer Theorem on Invariance of Domain, the range $R(A_n)$ is an open set in Y_n ; furthermore, it follows from (3) and the continuity of A_n that $R(A_n)$ is also a closed set in Y_n for each $n \geq N$. Indeed, if $\{y_m\} \subset R(A_n)$ and $y_m \rightarrow y$ in Y_n , then there exists a sequence $\{x_m\} \subset X_n$ such that $y_m = A_n x_m$ and, by (3), $\|y_\ell - y_k\| \geq (\|x_\ell - x_k\|)$. If $\beta(r)$ denotes the continuous strictly increasing function of R^+ into R^+ with $\beta(0) = 0$ which is the inverse of $\alpha(r)$, then since $\{y_m\}$ is a convergent sequence

$$\|x_\ell - x_k\| \leq \beta(\|y_\ell - y_k\|) \rightarrow 0, \quad (\ell, k \rightarrow \infty).$$

Thus there exists an element x in X_n such that $x_m \rightarrow x$ and $A_n x_m \rightarrow A_n x = y \in R(A_n)$ by the continuity of A_n for $n \geq N$, i.e., $R(A_n)$ is closed. Since $R(A_n)$ is a nonempty set in Y_n , which is both open and closed in Y_n , it follows that $R(A_n) = Y_n$.

Hence for each $n \geq N$ and for each given f in Y there exists a unique element x_n in X_n such that $A_n x_n = Q_n(f)$. For the sequence $\{x_n\}$ thus determined, (3) and the fact that $\|Q_n A(0)\| \leq K_0$ for some constant K_0 imply that

$$\|A_n x_n\| \geq \|A_n x_n - A_n(0)\| - \|A_n(0)\| \geq \alpha(\|x_n\|) - K_0$$

whence, in view of (2) and the fact that $\|Q_n(f)\| \leq K$ for some constant K , constant K , $\alpha(\|x_n\|) \leq K + K_0$, which implies that $\|x_n\| \leq \beta(K_0 + K)$, i.e.,

$\{x_n\}$ is a bounded sequence with $x_n \in X_n$ and such that $A_n x_n = Q_n(f) \rightarrow f$,

as $n_i \rightarrow \infty$. Hence, condition (H) implies that there exists a subsequence $\{x_{n_i}\}$ of $\{x_n\}$ and an element x in X such that $x_{n_i} \rightarrow x$ and $A_{n_i} x_{n_i} \rightarrow Ax$ as $n_i \rightarrow \infty$. Because $A_{n_i} x_{n_i} = Q_{n_i}(f) \rightarrow f$ as $n_i \rightarrow \infty$, we obtain that $Ax = f$, i.e., x is a solution of Eq. (1). Since, as was shown above, x is the unique solution of Eq. (1) for a given f in Y we conclude a posteriori that the selection of the subsequence was not necessary. Consequently, the entire sequence $\{x_n\}$ converges strongly to x , i.e., Eq. (1) is uniquely approximation-solvable.

Converse. Let $(\{X_m\}, \{Y_m\}, \{P_m\}, \{Q_m\})$ be an arbitrary approximation scheme for Eq. (1) and let $\{x_m\}$ be any bounded sequence in X with x_m in X_m such that

$$(4) \quad A_m x_m = g_m \rightarrow g \quad (m \rightarrow \infty)$$

for some g in Y . To prove that A satisfies condition (H) note first that, by our conditions on Q_m ,

$$(5) \quad \|g_m - Q_m(g)\| \leq \|g_m - g\| + \|g - Q_m(g)\| \rightarrow 0 \quad (m \rightarrow \infty).$$

Since by hypothesis, Eq. (1) is uniquely approximation solvable, for every $m \geq N$ and every g in Y there exists a unique element y_m in X_m such that

$$(6) \quad A_m y_m = Q_m(g)$$

$y_m \rightarrow y$ in X as $m \rightarrow \infty$, and y is the unique solution of the equation $Ay = g$.

Subtracting (4) from (6) we obtain the equation

$$Q_m y_m - A_m x_m = Q_m(g) - g_m$$

whence, in virtue of (3), for all $m \geq N$ we derive the inequality

$$\alpha(\|y_m - x_m\|) \leq \|A_m y_m - A_m x_m\| = \|Q_m(g) - g_m\|.$$

This, (5) and the properties of $\beta(r)$ imply that

$$(7) \quad \|y_m - x_m\| \leq \beta(\|Q_m(g) - g_m\|) \rightarrow 0 \quad (m \rightarrow \infty).$$

Since $y_m \rightarrow y$, (7) implies that $x_m \rightarrow y$ as $m \rightarrow \infty$, and $Ay = g$. Consequently, (4) implies that $A_m x_m \rightarrow A_y$ and thus shows that A satisfies condition (H).

Theorem 2. Under the hypothesis of Theorem 1 the following equivalent assertions are valid:

(A₁) A satisfies condition (H)

(A₂) Eq. (1) is uniquely approximation-solvable.

(A₃) For any given f in Y , Eq. (1) has a solution x in X .

Proof. By Theorem 1 and Definition 2, to prove Theorem 2 it suffices to show that (A₃) implies (A₁). Let $(\{X_m\}, \{Y_m\}, \{P_m\}, \{Q_m\})$ be an approximation scheme for Eq. (1) and let $\{x_m\}$ be any bounded sequence in X with $x_m \in X_m$ such that $A_m x_m \rightarrow g$, as $m \rightarrow \infty$, where g is some element of Y . To show that A satisfies condition (H) note first that, by (A₃), the equation $Ax = g$ has a solution y in X which, in view of our hypothesis on A and (3), is unique. Furthermore, since $A_m x_m \rightarrow g = Ay$ as $m \rightarrow \infty$, (3) and our conditions on A imply that for all $m \geq N$

$$\alpha(\|x_m - P_m y\|) \leq \|A_m x_m - g\| + \|Ay - Q_m A P_m y\| \rightarrow 0$$

as $m \rightarrow \infty$. Hence $\|x_m - P_m y\| \rightarrow 0$ and thus $x_m \rightarrow y$, since $P_m y \rightarrow y$ as $m \rightarrow \infty$. Moreover, by our hypothesis on $\{A_m x_m\}$ and the fact that $Ay = g$,

we obtain the convergence $A_m x_m \rightarrow Ay$. Thus assertion (A_1) follows.

Remark 1. Looking over the proofs of Theorems 1 and 2 we note that nowhere did we use the continuity property of the mappings P_n and Q_n assumed in Definition 1. Consequently, Theorems 1 and 2 remain valid without the assumption that P_n and Q_n are continuous. This fact may on occasion prove to be important and useful particularly in the application of these theorems to the collocation and finite difference methods.

Let us remark in passing that combining the results of Theorem 2 with those of Theorem 2 in [14] we have the following useful theorem.

Theorem 3. If additionally we assume that X and Y are separable reflexive Banach spaces, then under the hypotheses of Theorem 1 the following equivalent assertions are valid:

(A_1) A satisfies condition (H).

(A_2) Eq. (1) is uniquely approximation-solvable.

(A_3) For any given f in Y , Eq. (1) has a solution x in X .

(A_4) A satisfies condition (C).

An immediate and practically useful corollary of Theorem 2 is the following theorem.

Theorem 4. Let X and Y be two normed linear spaces and let $(\{X_n\}, \{Y_n\}, \{P_n\}, \{Q_n\})$ be an approximation scheme for Eq. (1); suppose that A is a continuous mapping of X into Y and that $Q_n(y)$ converges to y strongly in Y and uniformly for y in a compact subset of Y . Suppose further that there exists an integer $N > 0$ and a continuous strictly increasing function $\alpha(r)$ from R^+ to R^+ with $\alpha(0) = 0$ and $\lim_{r \rightarrow \infty} \alpha(r) = \infty$ such that the inequality (3) holds. Then under the above hypotheses, assertions (A_1) , (A_2) and (A_3) of Theorem 2 are equivalent.

Proof. Let us remark that in virtue of Theorem 2, to prove Theorem 4 it suffices to show that A_n is continuous in X_n for all sufficiently large n and that $Q_n A P_n x \rightarrow Ax$ for each x in X as $n \rightarrow \infty$.

First note that since A and Q_n are continuous and $P_n x \rightarrow x$ for each x in X , it follows that A_n is continuous in X_n for each n and that $A P_n x \rightarrow Ax$ for each x in X . Since $\{A P_n x\}$, being a convergent sequence, is a relatively compact subset of Y , it follows from our additional conditions on Q_n that $Q_n A P_n x \rightarrow Ax$ for each x in X as $n \rightarrow \infty$. Thus, by the above remark, Theorem 4 follows from Theorem 2.

Remark 2. Under the hypotheses of Theorem 4, with $\alpha(r)$ satisfying slightly weaker conditions, the equivalence of assertions (A_2) and (A_3) was first obtained by Browder in [5].

2. Perturbed problems. Our next theorems deal with the problem of constructive solutions of perturbed equations

$$(8) \quad Ax + Bx = f, \quad f \in Y,$$

as strong limits of solutions $x_n \in X_n$ of approximate equations

$$(9) \quad A_n x_n + B_n x_n = Q_n(f) \quad (A_n = Q_n A|_{X_n}, B_n = Q_n B|_{X_n}),$$

where B is a completely continuous mapping of X into Y (i.e., B is continuous and maps bounded sets in X into relatively compact sets in Y) and A satisfies certain conditions to be specified below.

Theorem 5. Suppose that X and X are normed linear spaces, $Q_n(y)$ converges to y strongly in Y and uniformly for y in a compact subset of Y , B is completely continuous and A satisfies condition (H). Suppose further that for a given f in Y and each $n \geq N$ the solutions x_n of Eq. (9) exist and are uniformly bounded by a constant independent of n .

If for any given f in Y Eq. (8) has at most one solution x in X , then Eq. (0) is approximation-solvable (i.e., Eq. (9) has a solution x_n in X_n for each $n \geq N$ such that $x_n \rightarrow x$ as $n \rightarrow \infty$, and x is the unique solution of Eq. (8)).

Proof. By the second part of our hypothesis for each $n \geq N$ there exists at least one element x_n in X_n such that $A_n x_n + B_n x_n = Q_n(f)$ for any given f in Y ; we choose one x_n for each n . It follows from our hypotheses that to establish the strong convergence of the approximate solutions x_n of Eq. (9) to an element x in X which is the solution of Eq. (8), it suffices to show that there exists an infinite subsequence $\{x_{n_i}\}$ of $\{x_n\}$ which converges strongly to x . Now, since $\|Q_n(f)\| \leq K$ for some constant $K > 0$, it follows from the second part of our condition that $\{x_n\}$ is uniformly bounded. Using the complete continuity of B , we may assume that $Bx_n \rightarrow w$ as $n \rightarrow \infty$, where w is some element of Y . Hence, by our additional condition on Q_n , $Q_n Bx_n \rightarrow w$ as $n \rightarrow \infty$. Consequently $A_n x_n = Q_n(f) - B_n x_n \rightarrow f - w$ as $n \rightarrow \infty$. Since the sequence $\{x_n | x_n \in X_n, n \geq N\}$ is bounded and A satisfies condition (H), there exists an infinite subsequence $\{x_{n_i}\}$ of $\{x_n\}$ and an element x in X such that $x_{n_i} \rightarrow x$ and $A_{n_i} x_{n_i} \rightarrow Ax = f - w$ as $n_i \rightarrow \infty$. On the other hand, the continuity of B and the additional property of Q_{n_i} imply that $B_{n_i} x_{n_i} \rightarrow Bx = w$. Hence x is a solution of Eq. (8) which, by our hypothesis, is unique, i.e., Eq. (8) is approximation-solvable.

Let us remark that though the second hypothesis of Theorem 5 is rather restrictive, nevertheless it is not artificial and often can be verified in applications. Analogous conditions are used by other authors (see, for example, Krein and Prozorovskaya [6]). Condition (a) below which was introduced by Browder [3] is essentially of the same strength. Furthermore,

if X is a reflexive Banach space, $Y = X^*$ and P_n and $Q_n = P_n^*$ are linear projections in X and X^* , respectively, then the second hypothesis of Theorem 5 is implied by the coerciveness condition

$$\frac{|(Au, u)|}{\|u\|} \rightarrow \infty \text{ as } \|u\| \rightarrow \infty,$$

which has been successfully used by a number of authors (see, for example, Browder [3, 5], Leray and Lions [8] and Lions [9]).

A consequence of our Theorem 5 is the following useful theorem proved by Browder [5] under the additional hypothesis that, for a given f in Y , Eq. (8) possesses a solution x in X .

Theorem 6. Suppose that X and Y are normed linear spaces, $Q_n(y)$ converges to y strongly in Y and uniformly for y in a compact subset of Y , A is continuous and B is completely continuous. Suppose also that there exists an integer $N > 0$ and a function $\alpha(r)$ as in Theorem 1 such that the inequality (3) holds for each $n \geq N$. Suppose further that A maps X onto Y and that for each $n \geq N$ the following condition (a) is satisfied.

(a) For each given f in Y and each $n \geq N$, Eq. (9) has a solution x_n in X_n ; there exists a continuous real-valued function $m(r)$ of R^+ into R^+ such that if for any $n \geq N$, x in X_n and h in Y_n we have $A_n x + B_n x = h$, then $\|x\| \leq m(\|h\|)$. Then Eq. (8) is approximation-solvable provided that, for any given f in Y , Eq. (8) has at most one solution x in X .

Proof. Note first that as in the proof of Theorem 4 our conditions on A and Q_n imply that A_n is continuous in X_n for each n and that $Q_n A P_n x \rightarrow Ax$

as $n \rightarrow \infty$, for each x in X . Furthermore, since A maps X onto Y , in virtue of our conditions on A , Theorem 2 shows that A satisfies condition (H). Finally, the first part of condition (a) implies that for each $n \geq N$ there exists at least one solution x_n of Eq. (9) while, since $\|Q_n(f)\| \leq K$ for some constant $K = K(f)$, the second part of condition (a) implies that the sequence $\{x_n\}$ is uniformly bounded. Thus all conditions of Theorem 5 are satisfied and, therefore, the validity of Theorem 6 follows from Theorem 5.

3. Remark on condition (H). In [14] we discussed various classes of linear and nonlinear functional equations in Hilbert spaces and reflexive Banach spaces for which condition (c) or (in view of Theorem 3) equivalently condition (H) is easily verifiable. In this section we discuss a rather general class of nonlinear operators acting in a nonreflexive Banach space for which condition (H) holds.

We recall that a separable Banach space X is said to have property $(\Pi)_C$ if there exists a sequence $\{X_n\}$ of finite dimensional subspaces of X , a sequence of linear projections $\{P_n\}$ defined on X and a constant $C \geq 1$ such that

$$(10) \quad P_n X = X_n, \quad X_n \subset X_{n+1}, \quad \overline{\bigcup_n X_n} = X, \quad \|P_n\| \leq C, \quad P_n P_j = P_j \quad \text{for } n \geq j.$$

Theorem 7. Let X be a separable Banach space with property $(\Pi)_1$. If the mapping A of X into X has the form $A = I - S - T$, where T is completely continuous and S is strictly contractive, i.e.,

$$(11) \quad \|Sx - Sy\| \leq \gamma \|x - y\| \quad \text{for all } x \text{ and } y \text{ in } X \text{ with } \gamma < 1,$$

then A satisfies condition (H).

Proof. Let $\{P_m\}$ satisfy (10), $(\{X_m\}, \{P_m\})$ be an arbitrary projectional approximation scheme for the equation

$$(12) \quad Ax = x - Sx - Tx = f, \quad f \in X,$$

and $\{x_m\}$ be any bounded sequence in X with $x_m \in X_m$ such that

$$(13) \quad g_m \equiv A_m x_m \rightarrow y$$

as $m \rightarrow \infty$, for some y in X . Since $\{x_m\}$ is bounded and T is completely continuous we may assume that $Tx_m \rightarrow w$ and, therefore,

that $T_m x_m \rightarrow w$ for some w in X . Thus to prove Theorem 7 it suffices to show that the operator $I-S$ satisfies condition (H). Now, by (13), we have

$$(14) \quad h_m \equiv (I-S_m)x_m = g_m - T_m x_m \rightarrow g - w \equiv h$$

as $m \rightarrow \infty$. When we put $L = I-S$ and $L_m = I-S_m$ then clearly

$P_n L P_n x \rightarrow Lx$ for each x in X and, in view of (10) and (11), for each m and all x and y in X_m

$$(15) \quad \|L_m x - L_m y\| \geq (\|x-y\|) \text{ with } \alpha(r) = (1-\gamma)r.$$

Thus the operator L satisfies all the conditions of Theorem 1. Since S is a strict contraction, it follows by the usual iteration argument that $L=I-S$ is a bicontinuous mapping of X onto X . Hence, by Theorem 2, L satisfies condition (H), i.e., $x_m \rightarrow x$ in X and $L_m x_m \rightarrow Lx = x - Sx = g - w$. But then, since T is continuous, $T_m x_m \rightarrow Tx = w$. Therefore, $A_m x_m \rightarrow Ax$

as $m \rightarrow \infty$, i.e., A satisfies condition (H).

Remark 3. It follows from the proof of Theorem 7 that if L is an arbitrary operator for which condition (H) holds and T is completely continuous, then the operator $L + T$ also satisfies condition (H). Thus, for example, if $A = L + T$ is an operator in Hilbert space H with L demicontinuous, bounded and complex monotone, then it follows easily that A satisfies condition (H). Let us add that in proving the last assertion one need not use the Zarantonello-Browder theorem [2, 16] which states that L is onto. However, if we drop the condition that L is bounded, then we would use the onto property of L .

4. In proving Theorem 7 we derived, as a byproduct, the strong projectional-solvability of the equation

$$(16) \quad x - Sx = f, \quad f \in X,$$

with S satisfying (11). This in itself appears to be a new result.

Let us also observe that if in Theorem 7 we assume that the operators S and T are linear, then the operator $A = I - S - T$ satisfies also the inequality (3) of Theorem 1 for $\alpha = \text{const.}$ In fact, we have the following theorem which should prove to be useful in various applications.

Theorem 8. Let X be a separable Banach space with property $(\pi)_1$.

If the mapping S and T of Theorem 7 are linear, then the equation

$$(17) \quad Ax = x - Sx - Tx = f, \quad f \in X,$$

is strongly projectionally-solvable provided $Ax = 0$ implies $x = 0$.

Proof. In view of Theorem 1 and 7, it suffices to show that there exist $N > 0$ and $\alpha > 0$ such that (3) holds, i.e.,

$\|A_n x - A_n y\| \geq \alpha \|x - y\|$ for each $n \geq N$ and all x and y in X_n .

Now, if (3) were not true, then there would exist a sequence $\{x_n | x_n \in X_n, \|x_n\| = 1\}$ with $x_n \rightarrow x$ in X such that $A_n x_n \rightarrow 0$ as $n \rightarrow \infty$. Since $x_n \rightarrow x$ implies that $A_n x_n \rightarrow Ax$, it follows that $Ax = 0$. Furthermore, since $T_n x_n \rightarrow Tx$, $g_n \equiv x_n - P_n S x_n = P_n A_n x_n + P_n T x_n \rightarrow Tx \equiv g$, and, since $\|P_n\| = 1$, for each n and each y in X we have $\|(I - P_n S)y\| \geq (1 - \delta) \|y\|$. Hence $(I - P_n S)$ is a one-to-one mapping of X onto X and

$$(13) \quad \|(I - P_n S)^{-1}\| \leq (1 - \delta)^{-1} \text{ for each } n.$$

Now, it is easy to see that $x_n \rightarrow x$. Indeed, since $Ax = 0$,

$$\begin{aligned} \|x_n - x\| &= \|(I - P_n S)^{-1} T x - (I - X)^{-1} T x\| \leq \|(I - P_n S)^{-1} g_n - (I - P_n S)^{-1} T x\| + \\ &+ \|(I - P_n S)^{-1} T x - (I - S)^{-1} T x\| \leq \|(I - P_n S)^{-1}\| \|g_n - T x\| + \\ &+ \|(I - P_n S)^{-1}\| \|(I - S) - (I - P_n S)\| \|(I - S)^{-1} T x\|. \end{aligned}$$

The above inequality and (18) imply that $x_n \rightarrow x$ as $n \rightarrow \infty$. Hence

$A_n x_n \rightarrow Ax = 0$ and $x_n \rightarrow x$. Therefore $\|x\| = 1$ in contradiction to the assumption that $Ax = 0$ implies $x = 0$.

Remark 5. Let us note that if X is a real Banach space with property $(\Pi)_C$, then condition (H) is related to the concept of P -compactness introduced and studied by the author in [10, 11, 12]. First let us recall that a nonlinear mapping T of X into X is said to be projectionally-compact (P -compact) if $T_n = P_n T|_{X_n}$ is continuous in X_n for all sufficiently

large n and if, for any given number $p > 0$ and any bounded sequence

$\{x_n | x_n \in X_n\}$, the strong convergence of $\{T_n x_n - p x_n\}$ implies the existence

of a strongly convergent subsequence $\{x_{n_i}\}$ and (necessarily) of an element x in X such that $x_{n_i} \rightarrow x$ and $T_{n_i} x_{n_i} \rightarrow Tx$ as $n_i \rightarrow \infty$. Thus, comparing the above definition with condition (H) we see that

T is P -compact if and only if for any given $p > 0$ the operator $A = T - pI$ satisfies condition (H) and A_{n_i} are continuous in X_{n_i} for all sufficiently large n_i .

The above observation implies that the operator $A_0 = T - p_0 I$ may satisfy condition (H) for some specific value of $p = p_0$ without necessarily having this property for $p \neq p_0$. Thus, we see that the concept of P -compactness involves a property associated with a one-parameter family of operators of the form $\{T - pI\}$, while the condition (H) involves a similar property for a specific value of the parameter p . In particular, a given mapping A satisfied condition (H) if there exists a constant $p_0 > 0$ such that $T = A + p_0 I$ is compact. The last remark may prove to be useful in various applications.

References

1. J.P. Aubin, Nonlinear stability-approximation of nonlinear operational equations (to appear).
2. F.E. Browder, Nonlinear elliptic boundary value problem, Bull. Amer. Math. Soc. 69 (1963), 557-566.
3. _____, Remarks on nonlinear functional equations, II, Illinois J. Math. 7 (1965), 608-618.
4. _____, Nonlinear accretive operators in Banach spaces (to appear).
5. _____, Approximation-solvability of nonlinear functional equations in normed linear spaces (to appear in Archive Rat. Mech. and Anal.).
6. S.G. Krein and O.I. Prozorvskaya, On the approximate methods in the solution of incorrectly posed problems, J. Comp. Math. and Math. Phys. 3 (1963), 120-130.
7. P.D. Lax and R.D. Richtmyer, Survey of stability of linear finite difference equations, Comm. Pure Appl. Math. 9 (1956), 267-293.
8. J. Leray and L.-L. Lions, Quelques résultats de Visik sur les problèmes elliptiques nonlinéaires par les méthodes de Minty-Browder, Bull. Soc. Math. France, 93 (1965), 97-107.
9. L.-L. Lions, Sur certaines équations paraboliques nonlinéaires, Bull. Soc. Math. France, 93 (1965), 155-175.
10. W.V. Petryshyn, On a fixed point theorem for nonlinear P-compact operators in Banach space, Bull. Amer. Math. Soc. 72 (1966), 329-344.
11. _____, On nonlinear P-compact operators in Banach space with application to constructive fixed-point theorems, J. Math. Anal. Appl. 15 (1966), 228-242.
12. _____, Further remarks on nonlinear P-compact operators in Banach space, J. Math. Anal. Appl. 16 (1966), 243-253.
13. _____, Projection methods in nonlinear functional analysis (to appear in J. Math. Mech.)
14. _____, Remarks on the approximation-solvability of nonlinear functional equations (to appear in Archive Rat. Mech. Anal.)
15. G. Vanikko, On the convergence of the collocation method for nonlinear differential equations, J. Comp. Math. and Math. Phys. 6 (1966), 35-42.
16. E.H. Zarantonello, The closure of the numerical range contains the spectrum, Bull. Amer. Math. Soc. 70 (1964), 781-787.

A BIDIRECTIONAL SIMPLEX ALGORITHM

A. Orden and V. Nalbandian

Abstract

A simplex type algorithm is presented which deals uniformly with (a) ordinary linear programming problems, (b) problems with upper bounded variables, and (c) problems with convex piecewise linear objective functions, e. g., absolute value terms. Problems of type (b) and (c) can be solved by suitable transformation into ordinary linear programming form, but are handled by the unified algorithm without such transformations. Comparative computer runs have been made which indicate that direct solution by the unified algorithm is considerably more efficient than conversion into ordinary linear programming form followed by use of a regular simplex routine. Computer tests have also shown that the algorithm offers a worthwhile alternative to the use of artificial variables as a starting procedure for ordinary linear programming problems.

1. Introduction

The general problem to be considered is to minimize a sum of piecewise linear, convex functions of variables $x_1, \dots, x_n$ under linear inequality and/or equation constraints:

$$(1) \quad \left\{ \begin{array}{l} \text{Minimize } \phi(x_1, \dots, x_n), \\ \phi = \sum_{j=1}^n f_j(x_j), \quad (\text{piecewise linear convex functions}) \\ \text{under constraints} \\ \sum_{j=1}^n a_{ij}x_j \begin{cases} = \\ \leq \\ \geq \end{cases} b_i, \quad i = 1, \dots, m. \end{array} \right.$$

Note that nonnegativity of the variables has not been specified.

This class of problems can be solved by use of well known transformations into equivalent linear programming problems which involve considerably enlarged sets of variables and constraints [1]. The algorithm which will be described below does not require such transformations.

A standard linear programming problem, with a linear objective function $\sum_j c_j x_j$, a set of linear constraints, and requiring nonnegative variables, can be expressed in the form of (1) by dropping the nonnegativity requirements, and replacing the terms $c_j x_j$ by the piecewise linear functions:

$$(2) \quad f_j(x_j) = \begin{cases} -\omega x_j & \text{for } x_j \leq 0 \\ c_j x_j & \text{for } x_j \geq 0 \end{cases}$$

where ω is an arbitrarily large positive number. The functions $f_j(x_j)$ in (2) have arbitrarily high (positive) values at all $x_j < 0$, hence, in a minimization problem, the values of $\sum_j f_j(x_j)$ for all vectors which contain any negative terms are greater than for any nonnegative vector.

It will be shown later that this artifice, whereby the domains of the variables in ordinary linear programming become $-\infty < x_j < \infty$ instead of $x_j \geq 0$, provides quite a satisfactory substitute for the use of artificial variables.

On the basis of (2) and a similar approach for upper-bounded variables, (1) is to be considered applicable throughout this paper to the various types of objective function which are illustrated in Figure 1:

Fig. 1a indicates the general case of convex, piecewise linear functions.

Fig. 1b indicates the functions $f_j(x_j)$ for dealing with ordinary linear programming problems.

Fig. 1c indicates that the piecewise linear functions may be absolute value functions, $|\alpha_j x_j - \beta_j|$.

Fig. 1d indicates functions $f_j(x_j)$ for use in lieu of upper-bounded variables. In place of upper bounds, $x_j \leq \mu_j$, the functions $f_j(x_j)$ are given an arbitrarily large slope at $x_j \geq \mu_j$.

Fig. 1e combines 1a and 1d.

The technique for dealing with arbitrarily large slopes which is used in our computer routines, and which may be considered to be part of the algorithm, is to represent the slope of each sector of a function $f_j(x_j)$ by a two-component "lexicographically ordered" vector. The first component is a coefficient of ω , and is dominant over the second component. For example, $[2, 4]$ is greater than $[0, 7]$ which

is greater than $[0, 3]$, which is greater than $[-1, 0]$. The successive cases may be thought of as $(2\omega + 4)$, $(0\omega + 7)$, $(0\omega + 3)$, $(-\omega + 0)$, where ω is some fixed, arbitrarily large number. Assuming that this technique applies, there is no need to distinguish among the various cases shown in Figure 1.

A general description of the algorithm will be given in Section 2. It is a "bidirectional simplex algorithm" in the sense of being similar to the regular simplex algorithm except that in each iteration a nonbasic variable may be increased or decreased, whereas in the regular simplex method a nonbasic variable is always increased. The computations consist primarily of pivot transformations in a simplex tableau. A similar algorithm, worked out in terms of use of dual variables rather than in terms of tableau transformations, has appeared in translation from the Russian literature [2]. Section 3 reports on computational experience with the bidirectional algorithm on several of the types of problem to which it can be applied. Section 4 provides a flow chart and a precise representation of certain steps which are stated only in general terms in Section 2. Although the basic difference from the regular simplex method is the bidirectional aspect noted above, there are also several points at which small adjustments can be made in the tableau instead of full pivot transformations. Although these are refinements which are not fundamental to the algorithm, they are significant with respect to computational efficiency.

Figure 1. Illustrative convex, piecewise linear objective function terms.

ω and $-\omega$ indicate arbitrarily large positive and negative slopes.

2. General Description of the Algorithm

By introducing slack variables into inequality constraints, let the system constraints in (1) be expressed as a system of equations. (The objective function term $f_j(x_j)$ for a slack variable has slope $-\omega$ at $x_j \leq 0$ and slope 0 at $x_j \geq 0$). The system of m equations can be transformed by a direct series of pivot transformations into the "reduced" form (3). The signs of the constants β_g are unrestricted. (We assume here that the constraint system has rank m , but if not, redundancy is uncovered during the reduction process, and superfluous rows can be dropped.) The reduced constraint system may be written:

$$(3) \quad x_g + \sum_h a_{gh} x_h = \beta_g,$$

where $\{g\}$ is a set of m indices of "basic variables" and $\{h\}$ is a set of $(n-m)$ indices of "nonbasic" variables.

In the ordinary simplex method a "basic solution" is associated with (3) by setting all $x_h = 0$, which results in $x_g = \beta_g$ for all g . Here, instead, we assign each nonbasic variable x_h to any one of its "breakpoint positions" -- the points at which the associated functions, $f_h(x_h)$ have a change of slope. Equations (3) then determine specific values for the basic variables x_g . A vector $[x_1, \dots, x_n]$ thus determined will be called a "breakpoint solution." Note for future reference that the number of breakpoint solutions of a system of linear equations with piecewise linear objective functions is finite -- there is a finite set of

forms (3) into which a given system can be transformed, and for each such form there is a finite number of ways to assign the nonbasic variables to breakpoints.

As indicated in Figure 2, let the breakpoints of the functions $f_j(x_j)$ be identified from left to right by an index p , and the breakpoint positions along the x -axis be denoted by x_j^p , $p = 1, 2, \dots, \bar{p}$. Let the successive chords be identified by an index s , and their slopes be denoted by c_j^s , $s = 1, 2, \dots, \bar{s}$.

Figure 2.

Let the leftmost breakpoint value x_h^1 be assigned to each of the nonbasic variables. Then the breakpoint solution at hand is

$$(4.1) \quad x_h = x_h^1 \quad \text{over the nonbasic index set } \{h\} ,$$

$$(4.2) \quad x_g = \beta_g - \sum_h \alpha_{gh} x_h^1 \quad \text{over the basic index set } \{g\} .$$

Assume that none of the basic variables whose values are given by (4.2) are at breakpoints of their respective objective functions. This is a nondegeneracy assumption which, as in the standard simplex algorithm, can be disposed of later on. Then each x_g of (4.2) is an interior point of one of the sectors of the associated function $f_g(x_g)$. Let the sector index be s_g , and the slope of that sector be $c_g^{s_g}$.

Let $\{\Delta x_h\}$ be arbitrary departures of the nonbasic variables from the values specified in (4.1). The corresponding changes of the basic variables are

$$(5) \quad \Delta x_g = - \sum_h \alpha_{gh} \Delta x_h$$

Let $\Delta \emptyset$ be the change of the overall objective $\emptyset$ relative to the value which is associated with the current breakpoint (4). Since we have assumed that the basic variables as given by (4.2) are not at breakpoints, for sufficiently small values of the quantities Δx_h (whether positive or negative) the slope $c_g^{s_g}$, associated with each basic variables remains constant. The total change on the objective function attributable to changes in the values of the basic variables is given by

$$(6) \quad \sum_g c_g^{s_g} \Delta x_g = - \sum_g c_g^{s_g} \sum_h \alpha_{gh} \Delta x_h$$

The non-basic variables are, however, at breakpoints, and we have

$$(7.1) \quad \Delta f_h(x_h) = c_h^1 \Delta x_h \quad \text{for } \Delta x_h < 0,$$

$$(7.2) \quad \Delta f_h(x_h) = c_h^2 \Delta x_h \quad \text{for } \Delta x_h > 0.$$

Thus we have a single expression (6) for the change of ϕ associated with the changes in the basic variables, and alternatives (7.1) or (7.2) for changes associated with the nonbasic variables. The overall change of ϕ may be written

$$\Delta \phi = \sum_h \left\{ \begin{array}{ll} c_h^1 \Delta x_h & \text{for } \Delta x_h < 0 \\ c_h^2 \Delta x_h & \text{for } \Delta x_h > 0 \end{array} \right\} - \sum_g c_g^s \sum_h \alpha_{gh} \Delta x_h,$$

or

$$(8) \quad \phi = \sum_h \left\{ \begin{array}{ll} (c_h^1 - \sum_g c_g^s \alpha_{gh}) \Delta x_h & \text{for } \Delta x_h < 0 \\ (c_h^2 - \sum_g c_g^s \alpha_{gh}) \Delta x_h & \text{for } \Delta x_h > 0 \end{array} \right\}.$$

Let $z_h = \sum_g c_g^s \alpha_{gh}$ over the index set $\{h\}$; then (8) takes the form

$$(9) \quad \Delta \phi = \sum_h \left\{ \begin{array}{ll} (c_h^1 - z_h) \Delta x_h & \text{for } \Delta x_h < 0 \\ (c_h^2 - z_h) \Delta x_h & \text{for } \Delta x_h > 0 \end{array} \right\}.$$

Equation (9) expresses the change of ϕ relative to its value at the current breakpoint solution, as it depends on two alternative sets of "reduced cost" terms $\{c_h^1 - z_h\}$ and $\{c_h^2 - z_h\}$. The two expressions correspond with the expression " $c_j - z_j$ " generally used in describing the basic simplex algorithm.

Suppose the upper set of reduced cost terms in (6) are all nonpositive, and at the same time, the lower set are all nonnegative, i. e., that

$$(10) \quad \left\{ \begin{array}{l} c_h^1 - z_h \leq 0 \\ c_h^2 - z_h \geq 0 \end{array} \right\} \quad \text{for all } h.$$

Then, by (9), for all combinations of values of Δx_h sufficiently small so that no basic variable reaches a breakpoint we have

$$\Delta\phi \geq 0.$$

In this case the breakpoint solution, specified by (4), is a local minimum for problem (1), and since we are dealing with a convex objective function on a convex set, it is a global minimum. Relation (10) is the terminating condition for the algorithm (assuming it has a finite Min.).

When the terminating condition does not hold there is at least one nonbasic variable such that either

$$(11.1) \quad -c_h^1 + z_h < 0$$

or

$$(11.2) \quad -c_h^2 + z_h > 0$$

Suppose (11.1) holds for a nonbasic variable x_{h*} . Then

$\Delta\phi$ can be made negative by taking $\Delta x_{h*} < 0$ while holding all other nonbasic variables constant, i. e., ϕ can be reduced from the value associated with the current breakpoint solution by moving x_{h*} in the negative direction from the current breakpoint position x_{h*}^1 . Similarly, if (11.2) holds, ϕ can be reduced by taking $\Delta x_{h*} > 0$, i. e., by moving x_{h*} in the positive direction from x_{h*}^1 .

Conditions (11.1) and (11.2) cannot hold simultaneously, for if this were the case we would have $c_h^2 < c_h^1$, which would violate the assumption that the functions $f_j(x_j)$ are convex. Thus each nonbasic variable which does not satisfy the terminating conditions (10) can reduce ϕ , either by use of $\Delta x_h > 0$ or by $\Delta x_h < 0$, but not both.

As in the basic simplex algorithm, it is efficient to make computational transformations on a tableau in which the reduced cost coefficients are adjoined to the matrix of the constraint equations. Let $\delta_h = -c_h^2 + z_h$ and $\bar{\delta}_h = -c_h^1 + z_h$. It is sufficient to carry the terms δ_h in the tableau, rather than both δ_h and $\bar{\delta}_h$. At the points in the algorithm at which the terms $\bar{\delta}_h$ are needed, they can be determined by the relation

$$(12) \quad \bar{\delta}_h = \delta_h + c_h^2 - c_h^1,$$

where c_h^2 and c_h^1 are the slopes of $f_h(x_h)$ for $\Delta x_h > 0$ and $\Delta x_h < 0$, with respect to the current breakpoint.

Assuming (10) does not hold, we proceed to a cycle of the algorithm.

Let h^* be the index of a nonbasic variable which maximizes $|-c_h + z_h|$ over the terms which satisfy (11.1) or (11.2). The variable x_{h^*} will be called the "driving variable."

The major cycle of the algorithm consists of minimizing ϕ with respect to the driving variable x_{h^*} , holding all other nonbasic variables fixed at their current breakpoints. Each major cycle may involve subcycles as follows.

A change of the driving variable (in the negative direction if it satisfies (11.1) and in the positive direction if it satisfies (11.2)) induces changes $\Delta x_g = \alpha_{gh} \Delta x_{h^*}$ in the basic variables. The following possibilities can arise:

- (a) the driving variable reaches a new breakpoint before any of the basic variables reach a breakpoint;
- (b) one of the basic variables reaches a breakpoint;
- (c) an unbounded change can be made in the driving variable without encountering (a) or (b).

In case (a) the value that the reduced cost coefficient assumes if the driving variable is taken beyond the new breakpoint is determined.

If (11) is still satisfied, the major cycle is continued by proceeding beyond the new breakpoint with the same driving variable, and outcome (a), (b) or (c) may again occur. If (11) is no longer satisfied, a new major cycle is started. In the latter event, since an interchange of a basic and nonbasic variable is not involved, only minor adjustments need be made in the computational tableau in preparation for the next major cycle.

Case (b) involves a breakpoint solution in which one of the basic variables has reached a breakpoint, and the driving variable is no longer at a breakpoint. The reduced cost coefficient of the driving variable is adjusted to reflect a change of slope of one of the basic variables (a change of one of the terms c_g^s in (8) to c_g^{s+1} or c_g^{s-1}). If it continues to satisfy (11), the major cycle is extended with the same driving variable, and (a), (b), or (c) may again occur. If the driving variable no longer satisfies (11) a pivot transformation is performed on the tableau. This converts the current driving variable to a basic variable, and the basic variable which has reached a breakpoint to a nonbasic variable. The tableau is then ready for a new major cycle.

Case (c) terminates the process. It indicates that $\emptyset$ is unbounded below.

In each step--whether the first step in a major cycle, or a subsequent subcycle in which the change of the driving variable is extended--the process arrives at a new breakpoint solution. Assuming nondegeneracy, each step reduces $\emptyset$, and since the number of breakpoint solutions is finite, the algorithm must terminate in a finite number of steps. Just as in the

standard simplex algorithm, degeneracy can be resolved by lexicographic perturbation, and computational experience has indicated that including such a technique in computer routines is unnecessary.

The specific computations required for the above steps are described in Section (4). The subcycles which extend the use of a driving variable beyond the first new breakpoint solution, without pivoting, involve various adjustments to the right-hand-side terms of the constraint equations, to the reduced cost coefficients, and to the total objective ϕ . The number of breakpoint solutions traversed is usually considerably greater than the number of pivot transformations. On this account, plus the fact that pivoting is on a smaller matrix, computational efficiency can be expected to be better than that of methods in which the original problem is converted into an equivalent ordinary linear programming problem.

When the bidirectional algorithm is applied to an ordinary linear programming problem each variable has just one breakpoint (see Figure 1b), consequently the term "breakpoint solution" need not be distinguished from the standard "basic solution." The basic variables may, however, take on negative as well as positive values. The algorithm can produce subcycles, as described above, when there are negative terms in the breakpoint solution, but not when all terms have become nonnegative.

(a) When negative values are present, the reduced cost coefficients γ_h or $\bar{\gamma}_h$ of the nonbasic variables are dominated by the effects of the large negative slopes $-\omega$ of one or more basic variables. The first term

of each of the lexicographically ordered pairs for γ_h and $\bar{\gamma}_h$ is in general nonzero, and consequently the second term is inactive. In this case the algorithm may have subcycles, as described above. A change of the current driving variable causes a basic variable to become zero, but reevaluation of the reduced cost coefficient of the driving variable may show that ϕ can be reduced further by continuing with the same driving variable. More than one basic variable may change from negative to nonnegative in a single major cycle.

(b) On the other hand, when all of the basic variables are positive and the nonbasic variables are zero, the behavior of the algorithm is identical to that of the regular simplex method. (1) A nonbasic variable cannot go negative, for that would increase ϕ due to the slope $-\omega$ at negative values of that variable. (2) When a suitable nonbasic variable is moved in the positive direction, ϕ decreases until some basic variable becomes zero, and would increase with any further change of the driving variable, since a basic variable would become negative and the slope $-\omega$ would apply. Thus no subcycles of the type described above occur--a pivot transformation is required after each step.

The phase of the bidirectional algorithm in which negative basic variables are present corresponds with the use of artificial variables in the ordinary simplex method. Computer runs to compare the number of pivot transformations required by the two algorithms during this phase are reported in Section 3C.

The situation on bounded variable problems, indicated by Figure 1d, is similar. As long as there are nonbasic variables which are either negative or greater than the upper bound, subcycles may occur in the bidirectional algorithm. When all variables have become feasible, the operation of the algorithm becomes identical with the ordinary bounded variable simplex technique (see [1], pp. 370-376).

3. Computational Efficiency

We have investigated the computational efficiency of the bidirectional algorithm in the following ways.

- A. Computer time comparisons on a few sample problems between the bidirectional algorithm and a widely used library routine for linear programming. Use of the latter involved conversion of problems with piecewise linear objective functions into equivalent linear programming problems. The objective functions were of two types:
 - (a) Functions $f_j(x_j)$ with five segments, as in Fig. 1a.
 - (b) Absolute value functions, as in Fig. 1c.
- B. Comparison, on a considerable number of sample problems, of the number of pivot transformations required by the bidirectional algorithm with that required by a bounded-variable version of the regular simplex algorithm.
- C. Comparison of the number of pivot transformations required to obtain an initial feasible solution in ordinary linear programming

problems by use of the bidirectional algorithm with that required by a "phase-1" simplex method using artificial variables.

A. Time Comparisons.

As a basis for computer time comparisons we used the linear programming routine RSMFOR, which originated at the Rand Corporation, and is distributed by the SHARE organization [2].

(a) Objective Functions with Five Segments.

Four sample problems (generated by procedures which are described in part B below) were solved by both routines.

The constraint matrices for solution by the bidirectional algorithm were 20 by 40, including a full set of slack variables. The matrices for solution by RSMFOR, involving introduction of bounded variables for four of the sectors of the functions $f_j(x_j)$, were 80 by 160.

Total net execution time (excluding system time) was 0.70 minutes with RSMFOR, and 0.40 minutes with the bidirectional algorithm. The latter was faster, despite the fact that the routine was written in the relatively crude and inefficient mode of transforming the entire tableau at each pivot stage, whereas RSMFOR is a fairly sophisticated routine which implements the product-form-of-the-inverse technique in order to operate efficiently on sparse matrices.

The time difference is mostly, but not entirely, attributable to the fact that RSMFOR worked on a much larger system of constraints, resulting from the conversion of problems with piecewise linear objective functions into regular linear programming problems. The average density of the sample problem matrices, in the form in which they were handled by RSMFOR, was 5.2%.

(b) Absolute Value Objective Functions

Similar time comparison runs were made on a batch of four problems with objective functions of the form

$$\phi = \sum_j |x_j|. \quad \text{The constraint matrices for the bidirectional}$$

algorithm were 20 by 40, with no slack variables. The average density of nonzero terms was 29%.

To convert such problems into ordinary linear programming form each variable is replaced by a pair of nonnegative variables. The vector of constraint coefficients associated with one of the pair of new variables is identical with that of the original variable, and the vector of the other new variable is the negative of the original vector. The new objective function is the sum of the $2n$ new variables.

Execution time for the batch (excluding system time) was 0.80 minutes with RSMFOR and 0.47 minutes with the bidirectional algorithm. The regular simplex method again

suffered substantial loss of efficiency because it had to deal with larger constraint matrices.

B. Number of Pivot Transformations--Objective Functions With Five Segments.

As indicated in Section 1, the bidirectional algorithm makes minor adjustments in the computational tableau, rather than full pivot transformations, at some of the breakpoints. These breakpoints would call for full pivoting if the problem were being solved by a regular simplex algorithm routine. In order to study the implications of this aspect of the algorithm on computational efficiency we made comparative runs on some 62 sample problems.

As a basis for these comparisons a routine was written for the simplex algorithm with bounded variables [1]. (RSMFOR was not suitable for comparison of the number of pivots as it uses a technique, called "double pricing", which tends to increase the number of pivots but improves computational efficiency when using the product-form-of-the-inverse [4].) In the bounded variable approach, each variable is replaced by a sum of as many bounded nonnegative variables as it has slope sectors. Each of the new variables has the same vector of constraint coefficients as the original variable. Its objective function coefficient is given by the slope of the corresponding line segment in the piecewise linear objective function of the original variable, and its upper bound is given by the difference between the x-coordinates of the end points of the corresponding segment.

The runs were made on sample problems with randomly generated constraint coefficients and objective functions. Each position in the original constraint matrix was assigned a zero or nonzero value with specific

probability, and similarly a positive or negative sign. The absolute values of the nonzero terms were drawn from the uniform distribution on $(0, 1)$.

All elements of the right-hand-side vector were made positive; they were drawn from the uniform distribution on $(0, 1)$ for one half of the set of problems and from $(0, 10)$ for the other half. Each problem was given a full set of slack variables, which made it initially feasible. (The issue of finding an initial feasible solution is considered under part C below).

The objective function associated with each nonslack variable consisted of five segments. The leftmost sector was given an arbitrarily steep negative slope, indicating nonnegativity constraints. The slope of the second segment was determined by adding a random number from $(0, 1)$ to the constant -1.2 . The slopes of the third, fourth, and fifth sectors were each determined by adding a random number from $(0, 1)$ to the slope of the preceding segment. The x-coordinate of the first breakpoint was set at zero, and the x-coordinate of each subsequent breakpoint was determined by adding a random number from $(0, 2)$ to the preceding breakpoint.

Table 1 gives summary statistics on 62 sample problems which were solved by both the bidirectional algorithm and the bounded-variable simplex algorithm. The number of pivot transformations required overall by the bidirectional algorithm was 18.4% smaller than required by the bounded variable simplex routine.

Table 1. Summary Statistics on Number of Pivots: 62 Sample Problems, with 5-Segment Objective Functions

	<u>Range</u>	<u>Average</u>
Number of constraints	20-60	31.6
Number of original variables, including slacks	40-100	57.4
Density of the constraint matrix, excluding the slack columns	20-30%	26.1%
Negative terms in the constraint matrix, excluding the slack columns	0-15%	6.3%
Number of pivot transformations:		
Bidirectional algorithm	0-69	20.8
Ordinary simplex method with the bounded variable technique	0-99	25.5
Number of right-hand-side adjustments without pivoting:		
Bidirectional algorithm	0-23	6.3
Bounded variable simplex	0-43	13.5
Number of basic variable segment changes, without pivoting, in the bidirectional algorithm	0-149	14.7

C. Number of Pivots to Find an Initial Feasible Solution in Ordinary Linear Programming Problems.

We noted in Section 2 that on ordinary linear programming problems the behavior of the bidirectional algorithm is identical with that of a regular simplex algorithm after a basic feasible tableau has been found, but that in the preliminary phase--that of finding an initial feasible tableau--the bidirectional algorithm has the opportunity to save on pivots by drawing more than one negative valued basic variable into the nonnegative domain in the course of a single major cycle, i. e., using a single driving variable.

In order to study this issue initial feasible solutions were found by both methods for 52 linear programming constraint systems, $Ax = b$, $x \geq 0$. Each element of the constraint matrix A was assigned a nonzero value with a specific probability, and a negative sign with a specific probability. These probabilities were given several values in the course of the overall set of runs. The absolute values of the non-zero terms were drawn from a uniform distribution on $(0, 1)$. The vector b was formed by adding together the columns of A .

The program used for starting the bidirectional algorithm is as follows. In the first row of the matrix $[A|b]$, the component of A with the largest absolute value is selected as the pivot element, and a pivot transformation is performed. The same procedure is repeated with each successive row, until a matrix containing m distinct unit vectors is obtained. At any stage, if the absolute values of all components in a row are extremely small, the row is discarded (the corresponding equation

is redundant). It is clear that the number of pivots in this part of the process is no greater than the rank of the matrix A . Upon completion of this process, the system is in the proper form for proceeding with the bidirectional algorithm using objective functions as in Figure 1b for all variables.

The starting procedure for the regular simplex algorithm was programmed as follows. In the matrix $[A|b]$ all right-hand-side terms are first made nonnegative by multiplying rows in which b is negative by -1 . A pivot element a_{rl} which preserves nonnegativity in the "b column" is then selected by finding the element

$$a_{rl} = \min_{a_i > 0} \left\{ b_i / a_{il} \right\}.$$

$$i = 1, \dots, m$$

The matrix is then transformed by pivoting at a_{rl} , and column l becomes a member of the set of basic vectors. The same procedure is repeated for each subsequent column of A . Whenever the potential pivot element for a new column lies in the same row as one of the earlier pivots, the program goes on to the next column without pivoting. This procedure, if it succeeds in pivoting in every row, yields a basic feasible tableau, and thus completes the desired process. When this does not occur, an artificial variable is introduced into each remaining row, after which the sum of the artificial variables is minimized by the

regular simplex algorithm ("phase-1"): In the set of sample problems the number of artificial variables introduced was, on the average, 37% of the number of constraint equations.

Results: The total number of pivot transformations required for finding feasible tableaus for 52 sample problems by the approach described above for the bidirectional algorithm was 14.3 smaller than that required by the artificial variables method. The data are summarized in Table 2.

Table 2. Number of Pivot Transformations in Finding an Initial Basic Feasible Solution to an Ordinary Linear Programming Problem.

	<u>Range</u>	<u>Average</u>
Number of constraints	20-60	30.0
Number of original variables (excluding artificials)	40-80	58.5
Density of the original constraint matrix	20-30%	25.4%
Negative terms in the constraint matrix	0-15%	7.7%
Number of pivot transformations:		
Bidirectional algorithm	21-69	35.3
Artificial variables technique	20-89	41.2
Number of artificial variables required by the ordinary simplex method	1-39	11.2
Number of row adjustments, without pivoting, in the bidirectional algorithm	0-55	13.2

4. Flow Diagram and Explanations

In this section we shall give a condensed flow diagram. The computations called for in the diagram are described in detail in subsequent explanatory notes. The following new symbols are used in the flow diagram.

β_i : the elements of the transformed requirements vector in the current tableau. β_i reflect the current values of the basic variables.

γ_h : the reduced cost of the nonbasic variable x_h if $\Delta x_h > 0$.

$\bar{\gamma}_h$: the reduced cost of the nonbasic variable x_h if $\Delta x_h < 0$.

θ_i : the smallest absolute amount of change in the value of x_{h^*} , in the direction which will yield a decrease in ϕ , that will cause the i th basic variable to reach a breakpoint value. The i th basic variable (x_{g_i}) is one whose unit coefficient in the current transformed tableau is in the i th row of the tableau.

θ : the smallest absolute amount of change in the value of x_{h^*} , in the direction which will yield a decrease in ϕ , that will cause x_{h^*} to reach a breakpoint value.

∞ : a number that exceeds the largest value attainable by any variable in the problem, assuming the optimal

solution is finite.
The flow diagram follows.

Figure 3. Flow Diagram of the Bidirectional Simplex Algorithm

The numbers in parentheses beside certain boxes of the flow diagram refer to explanatory notes below, following the definitions of certain additional needed symbols.

α_{ij} : the elements of the current transformed matrix A .

$\bar{s}_j$: the number of straight line segments comprising

$f_j(x_j)$

$\bar{p}_j$: the number of breakpoints in the graph of $f_j(x_j)$.

x_j^p : the p th breakpoint x -coordinate of $f_j(x_j)$. The breakpoints are numbered consecutively from left to right beginning with the number 1.

c_j^s : the slope of the s th straight line segment comprising $f_j(x_j)$.

s_i : the sector number of $f_{g_i}(x_{g_i})$ in which the value of x_{g_i} falls at the start of the current iteration.

t_i : the number of the most recent sector of $f_{g_i}(x_{g_i})$ in which x_{g_i} falls during the current iteration.

p_h : the breakpoint number of $f_h(x_h)$ at which the value of the nonbasic variable x_h rests at the beginning of the current iteration.

σ : an indicator of whether x_{h^*} is to be increased ($\sigma = +1$) or decreased ($\sigma = -1$) during the current iteration, in order to effect a decrease in ϕ .

σ_* : an indicator of whether $x_{g_{i*}}$ (i_* is determined by the following criterion: $\theta_{i*} = \min_i \{\theta_i\}$) increases ($\sigma_* = +1$) or decreases ($\sigma_* = -1$) during the current

iteration, as the value of x_{h*} changes in the direction which yields a decrease in θ .

p_* : the number of the latest breakpoint to which x_{h*} has moved during the current iteration.

t_* : the sector number of $f_{h*}(x_{h*})$ along which x_{h*} will move in case of a further change in its value.

p_* : the breakpoint number of $f_{g_{i*}}(x_{g_{i*}})$ whose x-coordinate coincides with the value of $x_{g_{i*}}$ just before the latter leaves the basis.

t_* : the number of the new sector of $f_{g_{i*}}(x_{g_{i*}})$ along which the value of $x_{g_{i*}}$ will move in case of any further change in the value of x_{h*} .

g_*, s_* : variables used for temporary recording of the values of g_{i*} and s_{i*} respectively.

Explanatory notes pertaining to the flow diagram:

(1) Select a set of m linearly independent vectors of A as the starting basis. If there are any slack vectors, it is preferable (from the standpoint of efficiency) to include them in the starting basis. Pivot in each basic column of the matrix of detached coefficients $[A|b]$ so as to obtain a starting tableau in reduced form. Set the initial value of p_h equal to 1 for all h in $\{h\}$. Adjust β_i , $i = 1, 2, \dots, m$, as follows:

$$\hat{\beta}_i := \hat{\beta}_i - \sum_{h \in \{h\}} \hat{\alpha}_{ih} x_h^1.$$

Each $\hat{\beta}_i$ now represents the value of the i th basic variable x_{g_i} . Determine $s_i, i = 1, 2, \dots, m$. $t_i := s_i, i = 1, 2, \dots, m$.

$$(2) \quad \gamma_h := -c_h^{p_h+1} + \sum_{i=1}^m c_{g_i}^{s_{g_i}} \alpha_{ih}, \text{ all } h \in \{h\}.$$

$$\bar{\gamma}_h := \gamma_h + c_h^{p_h+1} - c_h^{p_h}, \text{ all } h \in \{h\}. \text{ Find } \gamma_* = \max_h \{\gamma_h, -\bar{\gamma}_h\} = \gamma_{h^*}$$

or $-\bar{\gamma}_{h^*}$ as the case may be. $\sigma_* := +1$ if $\gamma_* = \gamma_{h^*}$; $\sigma_* := -1$ if

$\gamma_* = -\bar{\gamma}_{h^*}$. σ_* is needed later.

(3) For $i = 1, 2, \dots, m$, $\hat{s}_i := s_i$ if $s_i = 1$, $\hat{s}_i := s_i - 1$ if $s_i > 1$.

$$\theta = \sum_{i=1}^m f_j(x_j) = \sum_{i=1}^m [f_{g_i}^{s_i}(x_{g_i}^{s_i}) + c_{g_i}^{s_i} (x_{g_i}^{s_i} - x_{g_i}^{\hat{s}_i})] + \sum_{h \in \{h\}} f_h^{p_h}(x_h^{p_h}).$$

(4) θ_i may be computed as follows: for $i = 1, 2, \dots, m$, $\theta_i := \infty$

if $\hat{\alpha}_{ih^*} = 0$; or if $\sigma_* \hat{\alpha}_{ih^*} > 0$ and $s_i = 1$, or if $\sigma_* \hat{\alpha}_{ih^*} < 0$ and $s_i = \bar{s}_{g_i}$.

$$\theta_i := |(x_{g_i}^{s_i-1} - \hat{\beta}_i) / \hat{\alpha}_{ih^*}| \text{ if } \sigma_* \hat{\alpha}_{ih^*} > 0 \text{ and } s_i > 1.$$

$$\theta_i := |(x_{g_i}^{s_i} - \hat{\beta}_i) / \hat{\alpha}_{ih^*}| \text{ if } \sigma_* \hat{\alpha}_{ih^*} < 0 \text{ and } s_i < \bar{s}_{g_i}.$$

θ_* may be computed as follows: $p_* := p_{h^*}$, $t_* := p_*$ if

$\sigma_* < 0$; $t_* := p_* + 1$ if $\sigma_* > 0$. $\theta_* := \infty$ if $\sigma_* < 0$ and $t_* = 1$,

or if $\sigma_i > 0$ and $t_i = \bar{s}_{h^*}$, $\theta_i := x_{h^*}^t - x_{h^*}^{t-1}$ if $\sigma_i < 0$ and $t_i > 1$, or if $\sigma_i > 0$ and $t_i < \bar{s}_{h^*}$.

$$(5) \quad \gamma_* := \gamma_* - c_{h^*}^{p_*+1} + c_{h^*}^{p_*}$$

$$(6) \quad \sigma_* := -1 \text{ if } \sigma_i \hat{\alpha}_{i^*h^*} > 0, \quad \sigma_* := +1 \text{ if } \sigma_i \hat{\alpha}_{i^*h^*} < 0; \sigma_i \hat{\alpha}_{i^*h^*}$$

cannot be zero given that the program has branched to box number (6)

since $\sigma_i \hat{\alpha}_{i^*h^*} = 0 \Rightarrow \theta_{i^*} := \infty$ in box (4) of the flow diagram.

$$t_* := t_{i^*} + \sigma_* \quad \gamma_* := \gamma_* - |\hat{\alpha}_{i^*h^*} (c_{i^*}^{t_*} - c_{i^*}^{t_{i^*}})|$$

$$(7) \quad x_{h^*} := x_{h^*} + \sigma_* \theta_i \quad \text{For } i = 1, 2, \dots, m, \quad \theta_i := \theta_i - \theta_* \text{ if } \theta_i < \infty$$

$$t_i := t_i + \sigma_* \quad ; \quad \theta_i := \infty \text{ if } \sigma_i = -1 \text{ and } t_i = 1, \text{ or if } \sigma_i = +1 \text{ and } t_i = \bar{s}_{h^*}; \text{ otherwise, } \theta_i := x_{h^*}^{t_i} - x_{h^*}^{t_i-1} \quad p_* := p_* + \sigma_i$$

$$(8) \quad \gamma_{h^*} := \gamma_{h^*} + c_{h^*}^{p_{h^*}+1} - c_{h^*}^{p_*+1} \quad \Delta x := x_{h^*}^{p_*} - x_{h^*}^{p_{h^*}} \quad p_{h^*} := p_*$$

$$\hat{\beta}_i := \hat{\beta}_i - \hat{\alpha}_{ih^*} \Delta x, \quad i = 1, 2, \dots, m.$$

$$(9) \quad p_* := t_* \text{ if } \sigma_* = -1; \quad p_* := t_* - 1 \text{ if } \sigma_* = +1.$$

$$\hat{\beta}_{i^*} := \hat{\beta}_{i^*} - x_{g_{i^*}}^{p_*} \quad g^* := g_{i^*} \quad \text{Pivot, using } \hat{\alpha}_{i^*h^*} \text{ as the pivot element.}$$

$$p_* := p_{h^*} \quad h^* \rightarrow \{g\}, \quad s^* := s_{i^*}, \quad s_{i^*} := p_{h^*} + 1, \quad t_{i^*} := t_* \quad ; \quad g^* \rightarrow \{h\},$$

$$p_{g^*} := p_* \quad \hat{\beta}_{i^*} := \hat{\beta}_{i^*} = x_{h^*}^{p_*} \quad \gamma_{g^*} := \gamma_{g^*} + c_{g^*}^{s^*} - c_{g^*}^{p_*+1}$$

$$(10) \quad x_{h*} := x_{h*} + \sigma_* \theta_{i*} \quad \text{For } i = 1, 2, \dots, m, \quad \theta_i := \theta_i - \theta_{i*}$$

if $\theta_i < \infty$. $\theta_{i*} := \infty$ if $\sigma_* = -1$ and $t_* = 1$, or if $\sigma_* = +1$ and

$$t_* = s_{i*}, \quad \theta_{i*} = (x_{i*}^{t_*} - x_{i*}^{t_*-1}) / |\hat{\alpha}_{i*h*}|, \quad t_{i*} := t_*.$$

(11) For each i , $i = 1, 2, \dots, m$, such that $t_i \neq s_i$, repeat:

$$\Delta c := c_{g_i}^{t_i} - c_{g_i}^{s_i}, \quad \gamma_h := \gamma_h + \hat{\alpha}_{ih} \Delta c \quad \text{for all } h \in \{h\}.$$

$$s_i := t_i, \quad i = 1, 2, \dots, m.$$

REFERENCES

- [1] G. B. Dantzig, "Linear Programming and Extensions," pp. 482-486 and 370-376, Princeton University Press, 1963.
- [2] E. G. Golstein, "A Certain Class of Nonlinear Extremum Problems," Soviet Mathematics Vol. 1, No. 4 (1960), pp. 863-866. Translation from Doklady, Academy of Sciences of the USSR.
- [3] Richard Clasen, "Linear Programming Routine RSMFOR," SHARE Distribution Agency, No. 1379, 1962.
- [4] D. M. Smith and W. Orchard Hays, "Computational Efficiency in Product Form LP Codes," pp. 211-218 in "Recent Advances in Mathematical Programming," R. L. Graves and P. Wolfe, Editors, McGraw-Hill, 1963.