IMPACT OF DEMOGRAPHIC CHANGE ON INDUSTRY STRUCTURE IN AUSTRALIA

A joint study by the Australian Bureau of Statistics, the Department of Employment and Industrial Relations, the Department of Environment, Housing and Community Development, the Department of Industry and Commerce and the Industries Assistance Commission

SOLUTION OF NON-LINEAR PROBLEMS BY ITERATIVE USE OF A LINEAR PROGRAMMING PACKAGE

by

A. C. Edington
CSIRO
Division of Computing Research

and

John D. Harrower
Industries Assistance Commission

Preliminary Working Paper No. SP-08, Melbourne, January, 1977

IMPACT Project, Industries Assistance Commission, 606 St. Kilda Road, Melbourne, Victoria 3004 Australia
Area Code 03 Telephone 51 8611

CONTENTS

		Page
1.	INTRODUCTION	1
2.	THE MATHEMATICAL PROBLEM	2
	2.1 General Description	2
	2.2 The Algorithm	4
3.	THE COMPUTING TASK	11
	3.1 General Computing Approach	11
	3.2 Job Control Flow	13
4.	OUR PROGRAM STRUCTURE	16
	4.1 Implementation	16
	4.2 Program Organization	16
	4.3 Program Structure	18
5.	AN EXAMPLE PROBLEM SOLVED	18
6.	SUMMARY	25
Appendix A:	Preparing Data for the Linear Programming Package	27
Appendix B:	Obtaining the solution from the Linear Programming Package	29
Appendix C:	Use of the "BOUNDS" Option	30
Appendix D:	Job Control Example	33
Appendix E:	Program Structure and Example Program Listing	36
Table 2.1:	Nomenclature	9
Figure 2.1:	Basic Solution Strategy	5
Figure 2.2:	Detailed Algorithm	10
Figure 3.1:	Flowchart of System Control	14
Figure 4.1:	Commands for the Program	17
Figure D.1:	Job Control Commands	34
Figure E.1:	List of Subroutines used in the Computer Program	37

SOLUTION OF NON-LINEAR PROBLEMS BY ITERATIVE USE OF A LINEAR PROGRAMMING PACKAGE*

by

A.C. Edington and John D. Harrower

1. INTRODUCTION

This paper describes a computer program which allows the user to carry out the following sequence of steps : (1) input initial data to a linear programming package ; (2) solve the LP problem ; (3) read the LP solution ; (4) test the LP solution according to a user-specified set of termination criteria ; (5) if the termination criteria are not met, then the program allows for user-specified revisions of the data input in the light of the previously read LP solution and returns to (2). In summary, our program is a user-adaptable method for going in and out of an LP package.[1]

A typical application for the program is in the solution of non-linear programming problems. Assuming that we have a problem in which it is possible to make a piece-wise linear approximation for a non-linear objective function and any non-linear constraint functions,

* We benefited from the helpful comments of John Sutton and Peter Dixon in the preparation of this paper.

1. Other computing systems are available, for example, see Smith (1965).

we can obtain an approximate solution by solving an LP problem.[1] However, on applying a termination test, we may find the approximate solution to be inadequate. In that event, we revise our piece-wise approximations and recompute the LP solution. A second problem in which the iterative use of an LP package has proved useful is in the solution of Walrasian general equilibrium models.[2] In fact, it was this particular application which prompted our current research effort.[3]

The paper is organized as follows : first the mathematical problem is described; then the computing task is defined in terms of our general computing approach and job control flow; the program structure is outlined and finally an example problem is solved to illustrate the solution technique.

2. THE MATHEMATICAL PROBLEM

2.1 General Description

The types of non-linear optimization problems we are considering here are problems which are large but have the following simplifying properties :

---

1. For some basic theory on solving non-linear programming problems by piece-wise linear approximations see Hadley (1964, ch. 4).

2. See for example, Dixon (1976b).

3. Our program is being used to solve SNAPSHOT, a long run economy-wide planning model. See Dixon, Harrower and Powell (1976).

a) the number of non-linear costs[1] and/or constraints is small ;

b) the remainder of the model consists of linear costs and/or constraints ;

c) the non-linearities are continuous functions ;

i.e., the models are nearly linear programming models except for some costs and/or constraints which are simple curves instead of linear functions.

The closeness of these non-linear models to linear models means that they can be approximated by appropriately constructed fully linear models. This property is important because it is much easier to solve large linear problems than it is to solve large non-linear problems. Linear programming packages are highly specialized and can handle very large problems efficiently. Programs for general non-linear problems however are much more complex and as a result are slower and limited in capacity.

The mathematical problem is thus to use linear programs to help solve non-linear problems. It is worth mentioning that some of the larger linear programming packages do have options to solve some special sorts of non-linear problems. For example, non-linear costs representing say economies of scale, can be solved using separable programming or special-ordered-set techniques. Decision (YES/NO) variables and integer functions can also be handled by using the more advanced integer programming algorithms.

---

1. Costs - a term meaning the coefficients in the linear program's objective function.

The non-linear problems we concern ourselves with in this paper are ones with both non-linear costs and non-linear constraints. These are slightly more complicated than the examples above and cannot be solved directly with a linear programming package.

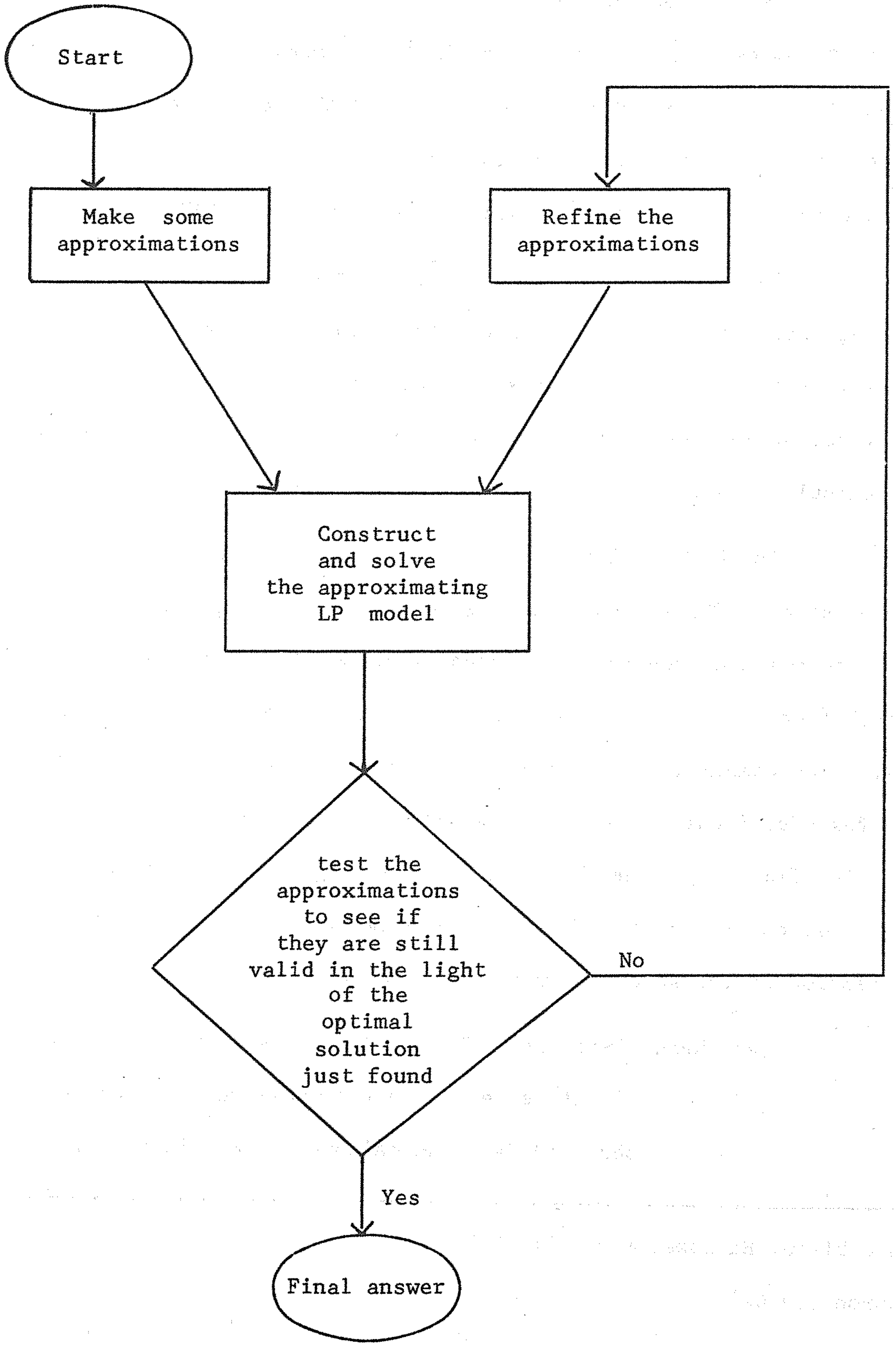
2.2 The Algorithm

Non-linear models are sometimes solved by hand by making linear approximations to the non-linearities and solving the linear program hoping that the approximations are still valid in the optimal solution. If some of the assumptions are found to be grossly inaccurate, they are changed and the model re-run.

We follow the same lines except that the whole process of approximating and refining of the approximations is done under computer control. In this way much more accurate solutions can be obtained and can be computed very quickly once the programs have been set up.

The solution process is illustrated in Figure 2.1.

FIGURE 2.1 BASIC SOLUTION STRATEGY


Start
Make some approximations
Refine the approximations
Construct and solve the approximating LP model
test the approximations to see if they are still valid in the light of the optimal solution just found
No
Yes
Final answer

While the strategy involved is fairly simple, the logistics to implement the process on the computer have been a barrier for modellers.

In the remainder of this paper we describe the computational method that was developed to solve the SNAPSHOT[1] model according to the solution algorithm proposed by Dixon.[2] Most of the ideas are also applicable to other non-linear models and the method is described in fairly general terms with only occasional references to SNAPSHOT.

In large models such as SNAPSHOT the amounts of computer processing required to set up the model (with initial approximations), to solve the LP, and to check the solution, can all be quite large. One of the first activities in planning the solution algorithm was to break up the calculations into a series of simple and manageable steps.

The Dixon solution algorithm for SNAPSHOT is written in a series of steps. These steps may be classified as in Table 2.1. The solution consists of sequentially calculating one set of variables after another. (Such a method of solution is applicable to other non-linear models.) The computer algorithm was built up from these steps, together with a few additional steps which simplified the processing of the LP model. The final algorithm is illustrated in Figure 2.2. The notation for this flowchart (and for the computer programs) is given in Table 2.1. A description of the solution procedure is as follows :

(a) Read fixed data - Read and store variables and matrices of data external to the model such as constants, observed data and calibration coefficients.

---

1. See Dixon, Harrower & Powell (1976).

2. Dixon (1976a).

(b) Initialise iterative variables - A process of making assumptions or predictions of the values that some simple variables will take in the final solution.

(c) Calculate re-estimated variables - Based on the predictions made in step (b), linear approximations to the non-linear parts of the model are made.

(d) Generate the LP model - The data from steps (a), (b) and (c) are used to calculate the coefficients in the LP model.

(e) Solve the LP - As will be explained in Section 3, a linear programming package external to the solution controlling program is used to solve the LP sub-model.

(f) Read the LP solution - Extract from the LP solution variables of interest such as solution values (e.g. activity levels) and the nature of the solution (e.g. which constraints were binding, marginal costs, etc.)

(g) Calculate Post-LP variables - Other variables can now be calculated from solution values in the linear programming model.

(h) Test for final solution - Various tests are made on the iterative variables, LP variables, and post-LP variables to determine whether or not the solution has converged to sufficient accuracy.

(i) Adjust the iterative variables. If the errors in the LP approximation are too great, the assumptions are refined in preparation for a re-run starting at step (c). The adjustment rules are very important since they control the speed of convergence as well as whether the iterating process converges or not. Ideally the adjustments are dynamically controlled so that smaller variations are made as the model gets closer to the optimal solution.

(j) Calculate final solution variables. When an acceptable solution is finally found, various extra quantities based on the solution values are calculated and printed out.

Although the above algorithm was initially prepared for the SNAPSHOT model, it seems to be generally applicable to any non-linear problem of a similar nature.

TABLE 2.1 : NOMENCLATURE

1) Fixed Data - data which is exogenous to the model and does not change from iteration to iteration.

2) Iterative Variables - variables which are first "guesstimated" and then adjusted (after each trial solution is tested). They remain constant for a particular iteration but vary between iterations.

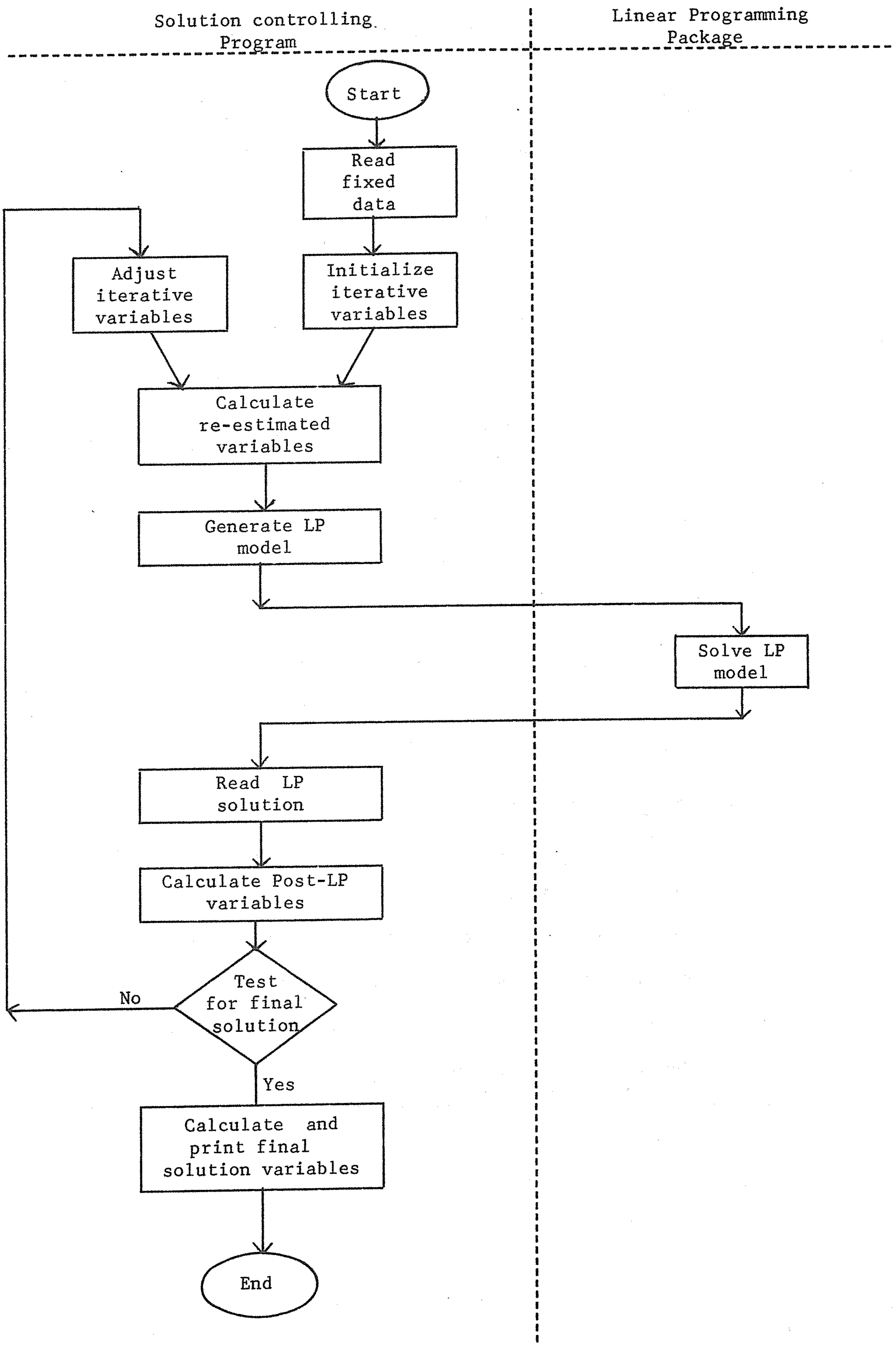
3) Re-estimated variables - these are functions of the fixed data, the iterative variables and the most recent linear program (LP) solution. They are mainly used to generate coefficients in the linear program matrix.

4) LP and Dual variables - values directly relating to the LP solution, e.g. activity levels, shadow prices (marginal values), etc.

5) Post-LP variables - these are functions of all of the above variables and are used, with the previous variables and data, to test the trial solution.

6) Final solution variables - these are variables which are computed after a satisfactory trial solution has been found. These variables are the endogenous variables in the model specification.

FIGURE 2.2 DETAILED ALGORITHM


Solution controlling Program
Linear Programming Package
Start
Read fixed data
Initialize iterative variables
Adjust iterative variables
Calculate re-estimated variables
Generate LP model
Solve LP model
Read LP solution
Calculate Post-LP variables
Test for final solution
No
Yes
Calculate and print final solution variables
End

3. THE COMPUTING TASK

3.1 General Computing Approach

The size of the linear program (LP) to be solved can be quite large; several hundred rows or more. This precludes the use of an LP subroutine from a subroutine library because the capacity of such routines is generally less than 100 rows. Thus the LP problems must be solved using one of the large capacity, but stand alone, LP packages. The main computing task then is to set up a system which makes it easy to transfer input problems to the LP package, to solve them, and to transfer the solution values back from the package. This is needed so that many iterations can be done in one computer run without the need for human intervention.

Normally LP packages expect their input data on cards. However, the packages can usually be instructed to read data card images from an alternate (programmer defined) file which is already stored on the system.

This feature can be used to get the package to read a file which is in fact prepared by a separate program. The FORTRAN programmer for example can use simple FORMAT statements[1] to write a file of "card images" which is acceptable to the LP package. Note that no cards are actually punched but temporary internal files are just transferred from one program to another.

Similarly, after the LP problem has been solved, instead

1. See Appendix A for more details and examples of writing card images from FORTRAN.

of printing the results on the printer, they can be written on to another internal file. That file can then be transferred back to the user's program and the results can be extracted by (say) READ statements in FORTRAN.[1]

Thus it is possible to repeatedly generate an LP problem, solve it, analyse the results and generate the next LP problem etc. The programming to control this flow of steps is described in more detail in the next section. There are, however, several points which will be discussed here first.

(1) Since the LP models generated during each iteration are similar it is reasonable to expect that the solutions will also be similar. This property can be used to advantage, by giving the old solution to the LP package to use as a starting point in finding the next solution. In LP terms this process is described as saving the old solution basis and using it as an initial basis for the following run.[2]

(2) If the LP model has many constraints of the form

$$x_i \leq b_i$$

i.e. constraint rows with only one variable, then they can usually be more efficiently represented as Bounds on the corresponding column variables. The number of rows in the LP model can be reduced significantly using this feature. See Appendix C for more details.

1. See Appendix B for an example of reading a solution file.

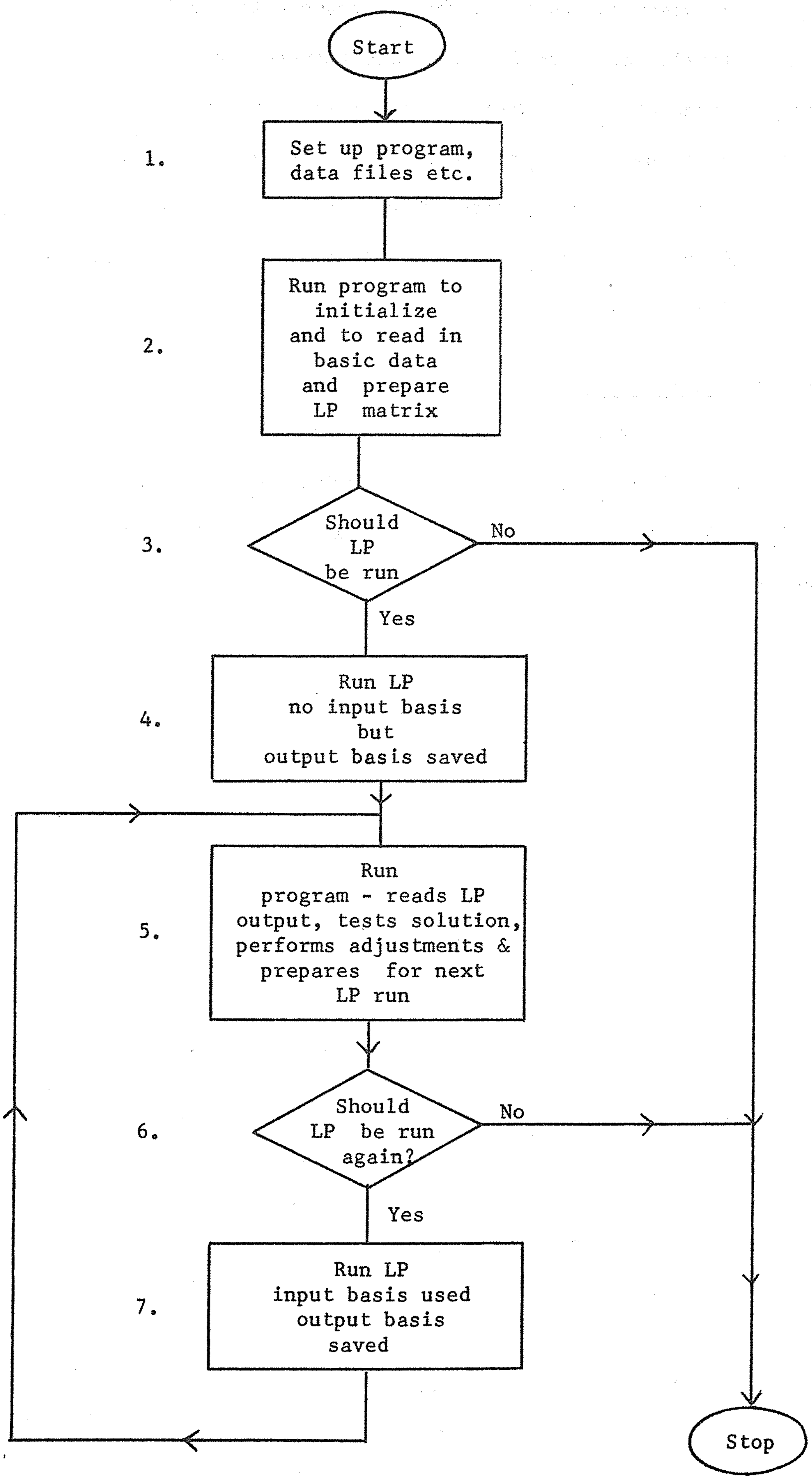
2. In our experience on the 200 row SNAPSHOT problem this technique obtained the solution in 15% of the time that would have been required if the previous solution had not been saved.

(3) Depending on the number of changes to the LP matrix between iterations, there is a decision between modifying the old matrix or generating a completely new matrix. Generally we would suggest that if, say, more than 10% of the matrix changes, then it is better to regenerate the matrix. The programming is easier and the extra computing time is not significant, compared with the full solution times.

3.2 Job Control Flow

The flow of control between the solution program and the LP package is fairly straight forward and is illustrated in Figure 3.1 Note how the solution basis from each LP run is saved and entered as a starting point for the subsequent LP run.

FIGURE 3.1 FLOWCHART OF SYSTEM CONTROL


Start
1.
Set up program, data files etc.
2.
Run program to initialize and to read in basic data and prepare LP matrix
3.
Should LP be run
No
Yes
4.
Run LP no input basis but output basis saved
5.
Run program - reads LP output, tests solution, performs adjustments & prepares for next LP run
6.
Should LP be run again?
No
Yes
7.
Run LP input basis used output basis saved
Stop

In using the LP package there are two options possible:

(a) to imbed our program in the LP package[1] and to call the LP routines directly

or (b) to keep our program separate and to use job control language (JCL) features to transfer control between our program and the LP package.

For the particular LP package that we used, APEX, the transfer of input and output data via files was the same for both (a) and (b). Thus the difference between the two was only in the simple area of passing control between the two programs. We chose to follow option (b) on the basis that it would be easier to debug two separate small programs than one large combined program. This choice enabled us to develop and debug our program independently of the LP package and let us avoid the chance of creating extra bugs by having to modify the LP package itself. We believe that choice (b) is a good one for anyone doing the same sort of thing.

We note in passing that the flowchart Figure 3.1 applies equally to choices (a) or (b).

Controlling the flow of execution via job control language (JCL) commands is relatively easy. If there are JCL equivalents to "IF" and "GOTO" then the flowchart Figure 3.1 can be implemented directly. If there are no such controls then the looping can be achieved by duplicating sets of the JCL commands for each loop.

1. The QUSER option in APEX allows user programs to be called from APEX.

Since the solution controlling program must end its execution before the LP package is run, all of its data in memory will be lost. In order for it to continue later from the point where it left off, it must save all of its data. This can be done by writing all of the relevant variables and matrices on to a temporary file. When the LP package has finished and the control program is initiated again, the data can be read in and normal execution can continue.

The JCL commands used on the CSIRO system for the Example problem in Section 5, are listed in Appendix D.

4. OUR PROGRAM STRUCTURE

4.1 Implementation

The computing was done on a Control Data Cyber-76 computer located at the CSIRO Division of Computing Research in Canberra. The main program was written in FORTRAN and the Control Data Linear Programming Package APEX-1 was used to solve the LP.

During this implementation emphasis was placed on developing a generalized control program which could be easily adapted later to other iterative linear programming applications.

4.2 Program Organization

Each of the boxes in the flowchart (see Figure 2.2) was chosen to be a single problem step. For model and program debugging, an extra printing step was added after each of the calculation steps. All of the steps were then coded as separate subroutines, and a main program was written to control the step by step execution of the problem. The main program could have been written as a series of subroutine calls in a predetermined order but a more flexible alternative method was chosen. A simple set of commands was defined to represent each of the steps, and a main program was written to read in the commands one by one and execute them. This command structure gave the flexibility during debugging to execute as little or as much as was wanted, without disturbance to the program code. Several extra commands were added to the basic ones, to add extra facilities to the program.

A set of commands to solve a complete problem is shown in Figure 4.1. (See Section 5) .

FIGURE 4.1. COMMANDS FOR THE PROGRAM

```
           TITLE     BLEND CHECKOUT RUNS
           READ FIXED DATA
           PRINT FIXED DATA
           INIT ITERATIVE VARIABLES
           PRINT ITERATIVE VARIABLES
LOOP       CALC RE-ESTIMATED VARIABLES
           PRINT RE-ESTIMATED VARIABLES
           GENERATE LP MATRIX
           SOLVE LP
           READ LP SOLUTION
           PRINT LP VARIABLES
           CALC POST-LP VARIABLES
           PRINT POST-LP VARIABLES
           TEST FOR FINAL SOLUTION (BRANCH TO "FINAL" IF SO)
           ADJUST ITERATIVE VARIABLES
           PRINT ITERATIVE VARIABLES
           GO TO      LOOP
FINAL      CALC SOLUTION VARIABLES
           PRINT SOLUTION VARIABLES
           PRINT FINAL SOLUTION
           END
```

4.3 Program Structure

The package was written incorporating ideas from modular programming. Use of these ideas simplified the program coding and debugging.

The main program does no mathematical work whatsoever. Its only task is to read a command, decide which command it is, and to call the appropriate subroutine to do the mathematics for that step. Each of the individual "worker" subroutines was deliberately coded to handle the mathematics for only one simple step of the algorithm. In this way, its function was clear and it could be easily identified if an error occurred. If a single step was complicated, some of the subtasks were coded into "helper" subroutines which were then called by the "worker" when needed. Thus the trap of writing large, hard to understand sections of code was avoided. All of the subroutines were small (less than a page of code) and easily understandable and updateable.

Further details on the program as applied to the BLENDER problem (see Section 5) can be found in Appendix E.

5. AN EXAMPLE PROBLEM SOLVED

In order to illustrate this iterative linear programming technique a small linear programming problem was modified to simulate a non-linear programming problem. With this small problem it was possible to concentrate attention upon the iterative features whilst putting size considerations to one side.

The problem chosen for our small and easily handled simulation model is a gasoline blending problem. The problem allows ready adaptation to an iterative formulation. A linear quality specification is removed from the linear programming problem and replaced by a test rule whose role is to simulate a non-linear constraint (which would have to be external to the LP and handled iteratively). The objective function reflects profit, and by allowing for some economies of scale in the production of one gasoline component a non-linearity is introduced into the objective function. The example follows :

A gasoline blender buys gasoline components, blends them, and sells the treated blends as gasoline. Each of the components, and the blended mixture, have different costs, octanes and quantity constraints.

The fixed data is contained in the table below :

Component	Cost/barrel	Octane	Quantity (Barrels)
Naphtha-1	P_1 (Non-linear*)	85	X_1 ($\leq$ 5000)
Naphtha-2	P_2 (\$4.375)	84	X_2 (unlimited)
Cracked Naphtha	P_3 (\$4.75)	89	X_3 ($\geq$ 200)
Blended Mixture	**Selling Price**		
Gasoline	P_4 (\$4.50)	Non-linear**	Q ($\leq$ 10000)

*	Cost of Naphtha	=	\$4.30	if quantity $\leq$	2000
	or	=	\$4.25	if quantity $>$	2000

** Octane of the blend: depends on the octanes and the volume ratios of the components

The blender can sell up to 10,000 barrels of gasoline, which must have an octane number of at least 85, at $4.50 per barrel. How much of each gasoline component should he buy to maximize his profit? The solution to our example can be seen by inspection and our iterative solution technique will be readily illustrated using this simple example problem.[1]

1. The solution to our simple simulation problem is apparent from inspection. The blender will initially choose all the naphtha-1 he possibly can (5000 barrels), since he makes the highest profit from it, and it also satisfies the quality specification (having an octane of 85). He will next make a profit by blending in naphtha-2, but to satisfy the octane specification he will have to mix in some (as little as possible) cracked naphtha in the ratio of 1:4 (cracked naphtha to naphtha-2). Despite the high cost of cracked naphtha he still makes a profit on each additional barrel of gasoline blended because four times as much of the profitable naphtha-2 is blended with it. This gasoline blended from cracked naphtha and naphtha-2 will be produced to the maximum of 5000 barrels; total gasoline sales having then reached the maximum 10,000 barrels.

The problem specification then is :

Choose $Q, X_1, X_2, X_3,$ to

maximize $P_4Q - P_1X_1 - P_2X_2 - P_3X_3$,

subject to

$$\begin{aligned}
X_1 + X_2 + X_3 &= Q \,, \\
Q &\leq 10{,}000 \,, \\
P_1 = 4.30 \text{ if } X_1 &\leq 2000 \,, \\
= 4.25 \text{ if } X_1 &> 2000 \,, \\
X_3 &\geq 200 \,, \\
X_1 &\leq 5000 \,, \quad \text{and} \\
85X_1 + 84X_2 + 89X_3 &\geq 85Q \,; \\
\text{where } X_1\ X_2 &\geq 0 \,, \\
\text{and} \qquad Q &\geq 200 \,.
\end{aligned}$$

(i) <u>Simulation of the Non-Linear Constraint</u>

To simulate a non-linearity it is assumed that the octane number specification is non-linear. Rather than model this non-linearity as a piece-wise approximation we decide to do this by formulating one of the LP variables as an iterative variable. This iterative variable will then be fixed for any one LP solution, then adjusted according to a set of rules after the solution is obtained, fed back into the LP and then the LP re-run. This procedure is repeated until the test rules are satisfied.

For the purposes of the gasoline blending problem we have chosen the cracked naphtha variable (X_3) as the iterative variable and the octane number specification (≥ 85) as the test rule. These are chosen because the octane number specification cannot be satisfied unless there is a significant amount of cracked naphtha present. Thus by initially setting the cracked naphtha at a sufficiently low level we can be sure that the program will not satisfy the octane test rule, and thus the iterative variable will have to be adjusted and the LP re-run. The octane test rule we will use is that for the V^th^ iteration:

$$- X_2(V) + 4 X_3(V) \geq 0, \text{ and}$$

the adjustment rule for the LHS < 0 is that

$$X_3(V + 1) = X_3(V) + 200 .$$

There are two features of formulating a variable as an iterative variable which assumes a fixed value for a particular linear program solution.

The first feature is that because the iterative variable is fixed for the LP run it influences the value the "normal" variables will adopt in the LP solution only via its effects on the LP coefficients and RHS values. The only direct effect (in this case) will be to reduce the value of the objective function (PROFIT) by a fixed amount ($.25\ X_3$) . This direct effect will not influence the value of variables in the LP solution. Hence iterative variables need not appear in the objective function of the linear program.

The second feature of formulating an LP variable as an iterative variable is that the row elements of the column in the LP which corresponds to the iterative variable can be incorporated into the values of the RHS column in the LP . This feature again arises because the iterative variable is constant for the particular LP run and can therefore be removed from the body of the LP matrix (with suitable adjustments being made to the RHS values of any rows in which the iterative variable appeared.)

(ii) Changing Coefficients in the Objective Function

To simulate a varying coefficient in the objective function of the linear program it will be assumed that there are some economies of scale in the production of naphtha-1 (X_1). At production rates less than or equal to 2000 barrels the cost to produce a barrel of naphtha-1 is $4.30. At rates greater than 2000 barrels, however, costs fall to $4.25 per barrel. From previous LP runs of the problem without the scale economies and constant costs of \$4.25 per barrel we know that more than 2000 barrels of naphtha-1 will be blended for regular gasoline because naphtha-1 attracts the highest profit per barrel to sales. Raising initial costs to \$4.30 per barrel still maintains the naphtha-1 component of gasoline as the most profitable. Hence if we initially set the profit coefficient to \$(4.50 - 4.30 = 0.20) for X_1, run the LP, adjust this coefficient after the first LP run to \$(4.50 - 4.25 = 0.25) and re-run the LP we will be simulating a changing of coefficients in the objective function of the linear program.

Thus the PROFIT row is set initially at :

	X_1	X_2	X_3
PROFIT	.20	.125	-.25

$\uparrow$

will be changed to 0.25 if X_1 of more than 2000 barrels is called for in the LP run

In terms of the algorithm flowchart as depicted in Figure 2.2 and the notation developed for the proposed iterative linear programming algorithm (Table 2.2) the small simulation (test) problem was formulated as :

a) iterative variable :	volume of cracked naphtha to be bought
b) re-estimated variable :	cost of naphtha-1 (based on the volume of naphtha-1 currently being bought)
c) the LP model :	maximise the profit (from selling gasoline) while observing supply and demand constraints, and taking into account the cost of the components
d) LP variables :	the volumes of naphtha-1 and naphtha-2 required (as determined by the LP)
e) Post-LP variable :	the octane of the blend (calculated from the octanes and LP-variable values)
f) Solution criteria :	is the octane of the blend at least 85?
g) adjustment rule :	vary the amount of cracked naphtha
h) final solution variables:	actual profit made overall.

The solution to the problem proceeded according to the flowchart outlined in Figure 2.2 thus:

1) guess an initial amount of cracked naphtha and P_1 ;
2) use the LP to determine the optimal volumes of naphtha-1 and naphtha-2 ;
3) calculate the octane of the mixture ;
4) depending on how the octane of the mixture compared with the octane requirement for gasoline, either adjust the volume of cracked naphtha blended and try again, or:
5) when a suitable blend is found, calculate the profit made by selling the gasoline, and print it out together with other relevant solution variables.

The solution to our example simulation problem was obtained after four iterations. A profit of $1500 is made by the blender who sells 10,000 barrels of gasoline of 85 octane by purchasing:

5000	barrels of naphtha-1 at $4.25 per barrel
4000	barrels of naphtha-2 and
1000	barrels of cracked naphtha

For further details of the example problem refer to Appendix E.

6. SUMMARY

In economics, and possibly other areas, the availability of an iterative linear programming technique which allows ready access and exit on a repetitive basis to a standard linear programming package is of much value. This paper documents such a technique. A simulated non-linear problem was also solved by application of the procedures outlined.

In the program design for the test problem, emphasis was placed on flexibility and on the provision of easily used facilities for debugging. Flexibility in the design was important since the final purpose of the program was to provide a prototype readily adaptable for use in large non-linear programming problems such as that involved in the jointmax algorithm for the solution of SNAPSHOT. A small set of commands was used to enable step by step execution of the problem as well as optimal printout facilities.

At the date of writing the code developed for the prototype problem has been adapted successfully for use in SNAPSHOT. Details will form the subject of a later research report.

REFERENCES

Control Data Cyber 70 Computer Systems Models 72, 73, 74, 76, 6000 Series Computer Systems, 7600 Computer Systems, "APEX-1 Reference Manual", Control Data Corporation, U.S.A., 1974.

Peter B. Dixon, "A Jointmax Algorithm for the solution of SNAPSHOT". Impact of Demographic Change on Industry Structure in Australia, Preliminary Working Paper No. SP-03, Industries Assistance Commission, Melbourne, April 1976 a.

Peter B. Dixon, "The Computation of Economic Equilibria : A Joint Maximization Approach", Seminar Paper No. 47, Department of Economics, Monash University, Melbourne, Australia 1976 b.

Peter B. Dixon, John D. Harrower and Alan A. Powell, "SNAPSHOT, A Long-term Economy-wide Model of Australia: Preliminary Outline", Impact of Demographic Change on Industry Structure in Australia, Preliminary Working Paper No. SP-01, Industries Assistance Commission, Melbourne, February 1976.

G. Hadley, Linear Programming, Addison-Wesley, Massachusetts, 1962.

G. Hadley, Non-Linear and Dynamic Programming, Addison-Wesley, Reading, Massachusetts, 1964.

H. V. Smith, "A Process Optimization Program for Nonlinear Systems: POP II", IBM Gen. Program Library 7090 H9 IBM 0021, 1965.

PREPARING DATA FOR THE LINEAR PROGRAMMING PACKAGE

As explained in Section 3.1, data is transferred to the LP package by writing a file of card images. The package then uses its normal input facility to read the data.

The individual card images can be created in Fortran by formatted WRITE statements.

For example, the following writes a matrix coefficient card in standard MPS format:[1]

```
        WRITE (file,40) column-name,row-name,value
    40  FORMAT(4X,A10,A10,F12.3)
```

To make the program more readable we defined small subroutines to handle the writing out of the data.

With these subroutines the programmer can concentrate on what is to be written without worrying about the details of how it is to be written. As an example, the following were used to prepare MPS row and column cards respectively :

```
        CALL ROWCARD  (row-type, row-name)
        CALL COLCARD  (column-name, row-name, value)
```

Typical calls were

```
        CALL ROWCARD  ("N", "OBJ")
        CALL COLCARD  ("VNIR", "OBJ", SELL-COST)
```

1. See Chapter 6 of the APEX-1 manual.

Within subroutine COLCARD the following logic was added:

a) Zero coefficients were ignored. A high percentage of the matrix coefficients can be zero and need not be transferred to the LP package.

b) Depending on the size of the coefficient, the appropriate format was chosen so that significant digits were not lost, e.g. numbers over 100 were written with F12.3, numbers less with F12.7.

The special MPS marker cards ("ROWS, "COLUMNS", etc) used to separate the different sorts of data cards, can be produced by normal FORMAT statements :

e.g.,

```
            WRITE  (file,100)
    100     FORMAT (4HROWS)
```

When the LP matrix has been generated (i.e. all of the relevant data card images written) the file should be rewound ready for input to the LP package.

In Appendix D (Figure D.1) the command "SOLVE=MATRIX" can be seen on the APEX card. This is the command to read the LP Matrix file (MATRIX) prepared by our FORTRAN program BLEND.

OBTAINING THE SOLUTION FROM THE LINEAR PROGRAMMING PACKAGE

The APEX-I package has an option which writes the solution information on to an internal file which can then be read by a Fortran program. A binary record[1] is written for each row and column variable, giving such information as row (or column) name, activity level, status and other auxiliary information such as dual and slack values. The Fortran program can read the records one by one and extract the details as it needs.

We took particular care in our subroutines to check that the row and column names of the solution records were as we expected them. This was to guard against the matrix generating routine and the solution reading routine getting out of step during future changes. The additional programming to do the checking was minimal because it was incorporated into the routine which read the solution file. Calls such as

```
Call  GETCOLM  (column-name, activity, marginal),
```

and

```
Call  GETROW   (row-name, row-activity, dual)
```

fetched the next records, checked their names and returned the values of interest.

In general the adding of precautions such as those above help speed debugging and protect against unnoticed bugs.

1. Appendix B in the APEX-1 manual gives details on the format of these records.

APPENDIX C

USE OF THE "BOUNDS" OPTION

During the authors' development of this solution sequence for related linear programming problems some features of the dual values for bounded constraint rows were formulated. Whilst this work has no direct bearing on the contents of this paper it was nevertheless thought to be of sufficient interest to be documented here.

The APEX-1 linear programming system provides a "BOUNDS" option for users.[1] This option allows a constraint row, in which only one variable appears, to be formulated outside of the actual LP matrix. Thus a constraint row (CONROW 1), in which only one variable (x_i) appears, can be handled without being explicitly introduced into the basis.[2]

The constraint, then, is of the form

(1) CONROW 1 $0 \leq x_i \leq d_i$,

where d_i is the RHS value .

This constraint is called an upper bound on the variable. If there are many such constraints, and if they are treated as conventional LP rows, they can rapidly build up the size of the basis. The special simplicity of these constraints means that they can be handled without being explicitly introduced into the basis.

1. See Control Data (1974), page 6 - 10.

2. See Hadley (1962), for a detailed derivation of the properties of upper and lower bounds.

Define a system of constraints which do not contain upper bounds.

(2) $$Ax = b \ ,$$

where A is an m x n matrix .

Assume that, in addition, each variable x_i has an upper bound $d_i > o$. Let $d = [d_1, \ldots, d_n]$ and $x_s = [x_{s1}, \ldots, x_{sn}]$ where x_{si} is the slack variable needed to convert (1) into an equation, i.e., (1) becomes

(3) $$x_i + x_{si} = d_i \ ,$$

where $x_{si} \geq o$,

and $i = 1 , \ldots , n$.

Then, the set of constraints including the upper bounds can be written :

(4) $$\begin{bmatrix} A & O \\ I_n & I_n \end{bmatrix} \begin{bmatrix} x \\ x_s \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}, \quad x \geq o, \quad x_s \geq o \ .$$

Now consider a new system for which the variables are defined as :

(5) $\bar{x}_i = d_i - x_i$, or $x_i = d_i - \bar{x}_i$, $i = 1 , \ldots, n$, and $\bar{x}_i \geq o$.

Thus,

(6) $x_i a_i$ becomes $d_i a_i - \bar{x}_i a_i$.

In this new system of variables $\bar{x}_i$, the matrix A of the coefficients for this system is the same as that for the original system. However b is replaced by

(7) $$\sum_{i=1}^{n} d_i a_i - b \quad .$$

Thus to convert a system $Ax = b$, $o \leq x \leq d$, into a system without upper bounds, the equivalent system

(8) $$A\bar{x} = Ad - b \ , \ \bar{x} \geq o \ , \ x = d - \bar{x} \ ,$$

is computed.

In the SNAPSHOT jointmax algorithm[1] there are 2n such sets of upper bounds. Now initially, $n = 105$ and the implementation of the "BOUNDS" option will reduce the basis size by 210 rows - a significant (65%) saving.

In using the BOUNDS option the APEX-1 package does not give the marginal values for these constraint rows since they are formulated as upper bounds and therefore are not explicit as dual values in the LP solution. The jointmax algorithm, however requires these marginal values for the variable values in the algorithm solution.

Upon inspection of the LP matrix it becomes apparent that for cases where the variable is constrained to the upper bound the dual value of the bound is the bounded variable's marginal (D_j) value. Where the variable is below the value of its upper bound the marginal value to the solution of an increment in the bound is of course zero. The values of these variables can therefore be gained by a simple test checking whether or not the bound is binding and assigning the variable the value of its D_j or zero respectively, depending upon the test result.

1. Dixon (1976a) pages 9 and 10; equations (2.3) and (2.5) .

APPENDIX D

JOB CONTROL EXAMPLE

This appendix gives an overview of the job control commands used to perform the iterative linear programming algorithm on the Cyber-76 at CSIRO. While the commands are fairly specific to this machine they are given to illustrate the algorithm.

Figure D.1 shows the control cards used for each of the steps described in Figure 3.1.

The program BLEND controls the mathematics of the solution algorithm and the program APEX is the LP package.

The loop control facilities (IF, GOTO and TAG) are locally developed utilities on the FUSE library. The controlling flag "E" is set within BLEND to different values to indicate conditions such as continue with LP, errors found, or final solution found.

FIGURE D.1 JOB CONTROL COMMANDS

STEP 1

ATTACH (FUSE) LIBRARY (FUSE)	Utility library
ATTACH (APEX, . . .)	Linear programming package
ATTACH (BLEND, . . .)	Our program
ATTACH (FX DATA, .. .)	Fixed Data
ATTACH (ACTION, . . .)	Commands for our program
SET (N = 5) NUMBER OF ITERATIONS (LOOPS)	Maximum number of iterations allowed

STEP 2

BLEND (ACTION, FXDATA, MATRIX, SAVE)	Initial run of program

STEP 3

IF (E, GE, 2, END)	Stop if errors

STEP 4

RFL (35000)	(Memory adjustment)
APEX (SOLVE = MATRIX, MAX, FS, O = SOLUTN, YSB = BASIS 2)	Initial run of LP
REDUCE .	(Memory adjustment)

STEP 5

TAG (LOOP)	Label at the start of the Loop
BLEND (ACTION, FXDATA, MATRIX, SOLUTN, SAVE)	Our program

STEP 6

```
IF(E, GE, 2, END)                                      Stop if errors or if
                                                       final solution found
```

STEP 7

```
REWIND (BASIS 1, BASIS 2)                              Prepare last

COPY (BASIS 2, BASIS 1)                                LP basis for input

REWIND (BASIS 1, BASIS 2)                              to next LP run

RFL (35000)

APEX (SOLVE = MATRIX, MAX, FS, 0 = SOLUTN,

                      INB = BASIS 1, YSB = BASIS 2)    Run the LP

REDUCE.

GOTO (LOOP)                                            Return to step 5 above
```

STEP 8

```
TAG (END)                                              End of run
```

APPENDIX E

PROGRAM STRUCTURE AND EXAMPLE PROGRAM LISTING

This appendix presents a listing of the FORTRAN program used to solve the BLEND problem. It is given to illustrate the general structure of the program and some of the finer specific details should not be taken as being generally applicable.

The index of subroutines (Figure E.1) shows how most of the high level subroutines correspond to individual "commands" described in Figure 3.1 and Figure 4.1.

Communication between the various subroutines is via the COMMON block "ARRAYS".

This method was chosen so that a central "data dictionary" of variables could be defined and documented and made easily available to all routines.

Care was taken in defining the COMMON variable names to avoid accidental clashes with local variables. Also a pre-processor (UPDATE) was used to automatically duplicate the COMMON cards in each subroutine. This both reduced the housework in setting up the programs and allowed speedy changes and additions to be made to the whole package during its development.

While this example program may seem to be oversimplified, it was developed as a framework ready for the much larger SNAPSHOT problem. The relatively quick development of the SNAPSHOT program showed this approach to be a successful one.

FIGURE E.1 LIST OF SUBROUTINES USED IN THE COMPUTER PROGRAM

SUBROUTINE NAME	SUBROUTINE TASK
BLEND	MAIN PROGRAM; CONTROLS EXECUTION BY READING USER COMMANDS ONE BY ONE AND CALLING APPROPRIATE SUBROUTINES
STARTUP	TO INITIALIZE OR PICK UP AFTER AN APEX RUN
TITLE C	TO STORE AWAY TITLE FROM TITLE CARD
READ FD	TO READ IN FIXED DATA
PRNT FD	TO PRINT OUT THE VALUES OF THE FIXED DATA VARIABLES
INIT IT	TO INITIALIZE ITERATIVE VARIABLES
PR ITER	TO PRINT OUT CURRENT VALUES OF ITERATIVE VARIABLES
RESTIM8	TO CALCULATE RE-ESTIMATED VARIABLES (WHICH ARE FUNCTIONS OF ITERATIVE VARIABLES)
PR ESTM	TO PRINT OUT CURRENT VALUES OF ESTIMATED VARIABLES
GENER8	TO GENERATE THE LP MATRIX DATA CARDS FOR APEX
CARD IM	WRITES ONE MARKER CARD IMAGE FOR APEX
ROW CARD	WRITES ONE FORMATTED APEX "ROW" CARD
COL CARD	WRITES ONE FORMATTED APEX "COLUMN" CARD (OR "RHS" CARD)
SOLV LP	GET READY TO GO INTO APEX
READ LP	READ IN LP SOLUTION FROM APEX FILE AND STORE AWAY LP AND DUAL VALUES
LP CHECK	TO READ IN THE NEXT APEX RECORD AND CHECK ITS NAME
GET ROW	GET LP SOLUTION VALUES FOR A ROW VARIABLE
GET COLM	GET LP SOLUTION VALUES FOR A COLUMN VARIABLE
PRNT LP	PRINTS OUT LP VARIABLES
POST LP	TO CALCULATE THE VARIABLES WHICH ARE FUNCTIONS OF THE LP VARIABLES
PR POST	PRINT POST-LP VARIABLES
TEST F S	TEST FOR FINAL SOLUTION
ADJUST	TO ADJUST THE ITERATIVE VARIABLE TOWARDS BETTER VALUES
GO TO L	GO TO LABEL
C SOLN	TO CALCULATE FINAL SOLUTION VARIABLES
PR SOLN	PRINT OUT CURRENT VALUES OF SOLUTION VARIABLES
P FINAL	TO PRINT NICELY THE FINAL VALUES OF INTEREST IN THE BIG MODEL
END IT	TO END EXECUTION OF THIS MODEL

AUXILIARY ROUTINES:

LINER	TO COUNT LINES AND PRINT NEW PAGE WHEN NEEDED
PTITLE	PRINTS TITLE AND PAGE HEADINGS
LOG	TO LOG A STATUS MESSAGE ON OUR SOLUTION LOG FILE
ACT EOF	TO ACT ON END-OF-FILE ON THE COMMAND FILE
ERROR	COMMAND ERROR
CHECK L	CHECK LABEL FIELD OF ACTION CARD
MATCH L	TO MATCH LABELS TO SEE IF ALL REFERENCED LABELS EXIST

```
*COMDECK ARRAYS      ................................  *COMDECK ARRAYS

      COMMON /ARRAYS/ FWA
C    FWA - FIRST WORD OF /ARRAYS/
C------FIXED DATA------------------------------------------------------
C    DFCOST2 - COST (PER BARREL) OF NAPRTHA-2
C    DFCOSTC - COST (PER BARREL) OF CRACKED NAPTHA
C    DFSELLG - SELLING PRICE OF GASOLINE
C    DFMAX1  - MAXIMUM  AMOUNT OF NAPTHA-1 AVAILABLE
C    DFMAXG  - MAXIMUM  AMOUNT OF GASOLINE THAT CAN BE SOLD
C    DFOCT1  - OCTANE VALUE OF NAPTHA-1
C    DFOCT2  - OCTANE VALUE OF NAPTHA-2
C    DFOCTC  - OCTANE VALUE OF CRACKED NAPTHA
C    DFOCTG  - OCTANE VALUE OF GASOLINE (REQUIRED VALUE)
C    ...  (ARRAYS OF ROW AND COLUMN NAMES WOULD APPEAR HERE IN LARGER
C        PROBLEMS)
     ,,DFCOST2,DFCOSTC,DFSELLG,DFMAX1,DFMAXG
     ,,DFOCT1,DFOCT2,DFOCTC,DFOCTG
C------ITERATIVE VARIABLES----------------------------------------------
C    VIVCNR  -  VOLUME OF CRACKED NAPTHA REQUIRED (BARRELS)
     ,,VIVCNR
C------REESTIMATED VARIABLES--------------------------------------------
C    VRCOST1 - COST OF NAPTHA-1 (PER BARREL)
     ,,VRCOST1
C------LP VARIABLES-----------------------------------------------------
C    VLVN1R  - VOLUME NAPTHA-1 REQUIRED
C    VLVN2R  - VOLUME NAPTHA-2 REQUIRED
C    VLOBJ   - OBJECTIVE VALUE OF THE LP SOLUTION
     ,,VLVN1R,VLVN2R,VLOBJ
C------POST-LP VARIABLES------------------------------------------------
C    VPOCTB  - OCTANE VALUE OF THE CURRENT BLEND
C    VPVGAS  - VOLUME OF GASOLINE BEING PRODUCED
     ,,VPOCTB,VPVGAS
C------FINAL SOLUTION VALUES--------------------------------------------
C    VSPROFT - PROFIT OF CURRENT SOLUTION
     ,,VSPROFT
C------HOUSEKEEPING VARIABLES-------------------------------------------
C    ITERATN - ITERATION NUMBER OF CURRENT PROBLEM
C    MAXIT   - MAXIMUM NUMBER OF ITERATIONS ALLOWED
C    LUNACT  - LUN FOR THE COMMAND FILE.  (MAY BE REWOUND BY THE PROGRAM)
C    LUNFD   - LUN FOR FIXED DATA
C    LUNLPM  - LUN FOR MATRIX CARDS (TO BE INPUT TO APEX)
C    LUNLPS  - LUN FOR APEX SOLUTION FILE (TO BE READ BY THIS PROGRAM)
C    LUNSAVE - LUN FOR SAVING INTERNAL VALUES BETWEEN EXCURSIONS TO APEX
C    TITLE(8)- TITLE TO PRINTED ON OUR SOLUTION LOG
C    ACTION(8)- CURRENT COMMAND CONTROLLING THIS PROGRAM
C    GOOD    - SET TO .TRUE. WHEN A COMMAND IS RECOGNISED DURING TESTING
C    TEST    - TEST MODE. NO EXECUTION IN ROUTINES WHEN TEST=.TRUE.
C    NERROR  - NUMBER OF COMMAND ERRORS DETECTED DURING TEST MODE
C    PAGENO  - PAGE NUMBER IN OUR TITLES
C    LINENO  - CURRENT LINE NUMBER ON PAGE
C    DDATE   - DATE OF RUN
C    DTIME   - TIME OF RUN
C    WORD(7)/IWORD(7) - 7 WORD BINARY RECORD FROM APEX SOLUTION FILE
     ,,ITERATN,MAXIT,LUNACT,LUNFD,LUNLPM,LUNLPS,LUNSAVE
     ,,TITLE(8),ACTION(8),GOOD,TEST,NERROR,PAGENO,LINENO,DDATE,DTIME
     ,,WORD(7)
C    LWA - LAST WORD OF /ARRAYS/
     ,,LWA
      INTEGER TITLE,ACTION,PAGENO
      LOGICAL  TEST,GOOD
      DIMENSION IWORD(7)
      EQUIVALENCE (IWORD,WORD)

*COMDECK TEST        ..............................  *COMDECK TEST
C..DURING TEST MODE, GOOD COMMANDS ARE RECOGNISED BUT NOT EXECUTED
      GOOD=.TRUE.
      IF(TEST) RETURN

*COMDECK LABELS      ..............................  *COMDECK LABELS
      COMMON /LABELS/ MAXLABL,NLABEL1,LABEL1(10),NLABEL2,LABEL2(10)
C   LABEL1  -  LABELS IN LABEL FIELD OF COMMANDS
C   LABEL2  - LABELS REFERENCED IN "GO TO" COMMANDS
      DATA MAXLABL/10/
*DECK BLEND          ..................................   *DECK BLEND
      PROGRAM  BLEND  (
     ,    LUNACT,LUNFD,LUNLPM,LUNLPS,LUNSAV,OUTPUT )

C..TO SOLVE THE BLENDING PROBLEM BY ITERATIVELY RUNNING APEX LP MODELS
C.. WITH ADJUSTMENTS AFTER EACH ITERATION.
*CALL ARRAYS
      DATA LUNACT/6LLUNACT/,LUNFD/5LLUNFD/,LUNLPM/6LLUNLPM/
      DATA LUNLPS/6LLUNLPS/,LUNSAVE/6LLUNSAV/

C......IF FIRST CALL TO BLEND,  INITIALIZE.
C....IF RE-ENTERING BLEND, RESTORE ARRAYS.
      CALL  STARTUP

      IF(TEST)  CALL LOG (-20,".LABEL.   .COMMAND.")
      IF(TEST)  CALL LOG (1,1H  )

   20 READ (LUNACT,40) ACTION
   40 FORMAT(8A10)
      IF(EOF(LUNACT).NE.0)  GO TO 9900
  100 CONTINUE
      GOOD=.FALSE.
      IF(.NOT.TEST)  CALL LOG (1,1H  )
      CALL  LOG  (-80,ACTION)
C..IGNORE COMMENTS
      IF(LOCH(ACTION,1).EQ.1R*)  GO TO 20
C...ONLY THE FIRST TEN CHARACTERS WILL BE CHECKED, BUT THE FULL
C...DESCRIPTIONS ARE GIVEN HERE FOR DOCUMENTATION.
      IF(ACTION(2).EQ."TITLE                            ")  CALL  TITLE C
      IF(ACTION(2).EQ."READ FIXED DATA                  ")  CALL  READ FD
      IF(ACTION(2).EQ."PRINT FIXED DATA                 ")  CALL  PRNT FD
      IF(ACTION(2).EQ."INIT ITERATIVE VARIABLES         ")  CALL  INIT IT
      IF(ACTION(2).EQ."PRINT ITERATIVE VARIABLES        ")  CALL  PR ITER
      IF(ACTION(2).EQ."CALC RE-ESTIMATED VARIABLES      ")  CALL  RESTIM8
      IF(ACTION(2).EQ."PRINT RE-ESTIMATED VARIABLES     ")  CALL  PR ESTM
      IF(ACTION(2).EQ."GENERATE LP MATRIX               ")  CALL  GENER8
      IF(ACTION(2).EQ."SOLVE LP                         ")  CALL  SOLV LP
      IF(ACTION(2).EQ."READ LP SOLUTION                 ")  CALL  READ LP
      IF(ACTION(2).EQ."PRINT LP VARIABLES               ")  CALL  PRNT LP
      IF(ACTION(2).EQ."CALC POST-LP VARIABLES           ")  CALL  POST LP
      IF(ACTION(2).EQ."PRINT POST-LP VARIABLES          ")  CALL  PR POST
      IF(ACTION(2).EQ."TEST FOR FINAL SOLUTION          ")  CALL  TEST FS
      IF(ACTION(2).EQ."ADJUST ITERATIVE VARIABLES       ")  CALL  ADJUST
      IF(ACTION(2).EQ."GO TO                            ")  CALL  GO TO L
                                                                (ACTION(3))
      IF(ACTION(2).EQ."CALC SOLUTION VARIABLES          ")  CALL  C SOLN
      IF(ACTION(2).EQ."PRINT SOLUTION VARIABLES         ")  CALL  PR SOLN
      IF(ACTION(2).EQ."PRINT FINAL SOLUTION             ")  CALL  P FINAL
      IF(ACTION(2).EQ."END                              ")  CALL  END IT
```

```
C...PICK UP UNRECOGNIZED COMMANDS
      IF(.NOT.GOOD)  CALL ERROR
C...CHECK THE LABEL FIELD
      IF(TEST)  CALL  CHECKL (ACTION(1))
      GO TO 20

 9900 CALL ACT EOF
C...IN TEST MODE -- ACTEOF WILL RETURN AFTER CHANGING MODE TO EXECUTE
C...IN NON-TEST MODE -- ACTEOF WILL CALL ENDIT AND STOP
      GO TO 20
      END
*DECK STARTUP   .....................................  *DECK STARTUP
      SUBROUTINE  STARTUP
C...TO INITIALIZE OR PICK UP AFTER AN APEX RUN
*CALL ARRAYS
      DATA TITLE/8*1H /
C..THE NUMBER OF ITERATIONS (LOOPS) IS SET BY THE CONTROL CARD
C            VARIABLE "L"
      MAXIT=ISET(1RN)
C..THE CONTROL CARD VARIABLE "E" IS USED TO CONTROL EXECUTION AS
C    FOLLOWS -      E=0  INITIAL (STARTUP) VALUE
C                   E=1  CONTINUE ITERATING
C                   E=2  END OF PROBLEM
C                   E=3  ERROR,  HALT PROBLEM
C..TEST FOR STARTUP OR NO
      ISTATUS=ISET(1RE)
C..RESET STATUS TO "ERROR" IN CASE WE BOMB OUT DURING THIS PROGRAM
      CALL  SET  (1RE,3)
      IF(ISTATUS.EQ.0)  GO TO 500
      IF(ISTATUS.EQ.1)  GO TO 1000
      STOP "STARTUP - ILLEGAL VALUE FOR 'E' FLAG"
C
C..STARTUP  (FIRST EXECUTION)
  500 REWIND LUNACT
      TEST=.TRUE.
      TITLE(1)=10HSOLUTION
      TITLE(2)=         10H CONTROL
      TITLE(3)=                   10H COMMANDS.
      PAGE NO = 0
      CALL DATE (DDATE)
      CALL TIME  (DTIME)
      CALL P TITLE
      RETURN
C
C..RESTORATION FROM APEX
 1000 CONTINUE
C..RESTORE /ARRAYS/
      REWIND LUNSAVE
      BUFFER IN (LUNSAVE,1) (FWA,LWA)
      IF(UNIT(LUNSAVE))  1050,1999,1999
 1050 RETURN

 1999 STOP "STARTUP - CANT RESTORE FROM INTERNAL SAVE FILE"
      END
*DECK TITLEC    .....................................  *DECK TITLEC
      SUBROUTINE  TITLE C
C...TO STORE AWAY TITLE FROM TITLE CARD
*CALL ARRAYS
*CALL TEST
      DO 20 I=1,6
   20 TITLE(I)=ACTION(I+2)
C..START NEW PAGE WITH NEW TITLE
```

```
      CALL   P TITLE
      RETURN
      END
*DECK READFD    .......................................   *DECK READFD
      SUBROUTINE  READ FD
C...TO READ IN FIXED DATA
*CALL ARRAYS
*CALL TEST
      READ (LUNFD,40)NNAME,DUMMY,DFCOST2,DFCOSTC,DFSELLG
      IF(NNAME.NE."COSTS")  STOP "READFD - 'COSTS' CARD BAD"

      READ (LUNFD,40) NNAME,DFMAX1,DUMMY,DUMMY,DFMAXG
      IF(NNAME.NE."LIMITS")  STOP "READFD - 'LIMITS' CARD BAD"

      READ (LUNFD,40) NNAME,DFOCT1,DFOCT2,DFOCTC,DFOCTG
      IF(NNAME.NE."OCTANES")  STOP "READFD - 'OCTANES' CARD BAD"
   40 FORMAT (A10,7F10)
      RETURN
      END
*DECK PRNTFD    .......................................   *DECK PRNTFD
      SUBROUTINE  PRNT FD
C...TO PRINT OUT THE VALUES OF THE FIXED DATA VARIABLES
*CALL ARRAYS
*CALL TEST
      CALL LINER (9)
      PRINT *,"                         ",DFCOST2 ,
     , "  DFCOST2 - COST (PER BARREL) OF NAPTHA-2        "
      PRINT *,"                         ",DFCOSTC ,
     , "  DFCOSTC - COST (PER BARREL) OF CRACKED NAPTHA "
      PRINT *,"                         ",DFSELLG ,
     , "  DFSELLG - SELLING PRICE OF GASOLINE   "
      PRINT *,"                         ",DFMAX1  ,
     , "  DFMAX1  - MAXIMUM  AMOUNT OF NAPTHA-1 AVAILABLE        "
      PRINT *,"                         ",DFMAXG  ,
     , "  DFMAXG  - MAXIMUM  AMOUNT OF GASOLINE THAT CAN BE SOLD"
      PRINT *,"                         ",DFOCT1  ,
     , "  DFOCT1  - OCTANE VALUE OF NAPTHA-1     "
      PRINT *,"                         ",DFOCT2  ,
     , "  DFOCT2  - OCTANE VALUE OF NAPTHA-2     "
      PRINT *,"                         ",DFOCTC  ,
     , "  DFOCTC  - OCTANE VALUE OF CRACKED NAPTHA        "
      PRINT *,"                         ",DFOCTG  ,
     , "  DFOCTG  - OCTANE VALUE OF GASOLINE (REQUIRED VALUE)    "
      RETURN
      END
*DECK INITIT    .......................................   *DECK INITIT
      SUBROUTINE  INIT IT
C..TO INITIALIZE ITERATIVE VARIABLES
*CALL ARRAYS
*CALL TEST
      DATA ITERATN/0/
      PRINT *,"                ITERATION NUMBER  ",ITERATN
      CALL LINER (1)
C..START AT 200 BARRELS OF CRACKED NAPTHA
      VIVCNR=200
      RETURN
      END
*DECK PRITER    .......................................   *DECK PRITER
      SUBROUTINE PR ITER
C...TO PRINT OUT CURRENT VALUES OR ITERATIVE VARIABLES
*CALL ARRAYS
*CALL TEST
```

```
      PRINT *,"                              ",VIVCNR
     , "  VIVCNR  -  VOLUME OF CRACKED NAPTHA REQUIRED (BARRELS)"
      CALL LINER (1)
      RETURN
      END
*DECK RESTIM8   .......................................   *DECK RESTIM8
      SUBROUTINE RESTIM8
C...TO CALCULATE RE-ESTIMATED VARIABLES (WHICH ARE FUNCTIONS OF ITERATIVE
C.....VARIABLES)
*CALL ARRAYS
*CALL TEST
C..CALCULATE PRICE OF NAPTHA-1   (DEPENDENT ON VOLUME USED)
      IF(VLVN1P.GT.2000.0)  GO TO 20
      VRCOST1=4.30
      RETURN
C..HIGH VOLUME
   20 VRCOST1=4.25
      RETURN
      END
*DECK PRESTM    .......................................   *DECK PRESTM
      SUBROUTINE  PR ESTM
C...TO PRINT OUT CURRENT VALUES OF  ESTIMATED VARIABLES
*CALL ARRAYS
*CALL TEST
      PRINT *,"                              ",VRCOST1 ,
     , "  VRCOST1 - COST OF NAPTHA-1 (PER BARREL)         "
      CALL LINER  (1)
      RETURN
      END
*DECK GENER8    .......................................   *DECK GENER8
      SUBROUTINE  GENER8
C..TO GENERATE THE LP MATRIX DATA CARDS FOR APEX.
C...THE MATRIX FILE  "LUNLPM" IS REWOUND BEFORE AND AFTER THIS ROUTINE
      INTEGER CARD(8)
*CALL ARRAYS
*CALL TEST
      REWIND LUNLPM
C...NAME CARD
      ENCODE ( 80,10,CARD) ITERATN
   10 FORMAT(*NAME             ITRN*,I2.2)
      WRITE (LUNLPM,20) CARD
   20 FORMAT(8A10)
      CALL LOG (30,CARD)
C.
C..CONSTRAINT NAMES AND TYPES
      CALL  CARD IM  ("ROWS")
C.....COST ROW
C...THE OBJECTIVE ROW REPRESENTS THE PROFIT GAINED FROM NAPTH 1 AND 2
C.....(THE COST OF CRACKED NAPTHA WILL BE TAKEN AWAY IN THE FINAL
C          SOLUTION VARIABLES
C..."OBJ" IS TO BE MAXIMIZED
      CALL ROWCARD ("N","OBJ")
C.....NAPTHA-1 LIMIT
      CALL ROWCARD ("L","LIMITN1")
C.....GASOLINE MARKET LIMIT
      CALL ROWCARD ("L","LIMITGAS")
C.
C...COLUMNS
      CALL  CARD IM  ("COLUMNS")
C.....VOLUME NAPTHA-1 REQUIRED
      CALL COLCARD ("VN1R","OBJ",DFSELLG-VRCOST1)
      CALL COLCARD ("VN1R","LIMITN1 ",1.0)
      CALL COLCARD ("VN1R","LIMITGAS",1.0)
C.....VOLUME NAPTHA-2 REQUIRED
      CALL COLCARD ("VN2R","OBJ",DFSELLG-DFCOST2)
      CALL COLCARD ("VN2R","LIMITGAS",1.0)
C.
C...RIGHT HAND SIDE
      CALL  CARD IM  ("RHS")
      CALL COLCARD ("RHS","LIMITN1",DFMAX1)
      CALL COLCARD ("RHS","LIMITGAS",DFMAXG-VIVCNR)
C.
      CALL  CARD IM  ("ENDATA")
      REWIND LUNLPM
      RETURN
      END
*DECK CARDIM    ......................................  *DECK CARDIM
      SUBROUTINE  CARD IM  (IMAGE)
C...WRITES A MARKER CARD IMAGE ONTO THE APEX DATA FILE
*CALL ARRAYS
      WRITE (LUNLPM,20) IMAGE
   20 FORMAT(A10)
      RETURN
      END
*DECK ROWCARD   ......................................  *DECK ROWCARD
      SUBROUTINE ROW CARD  (ROWTYPE,ROWNAME)
C..WRITES ONE FORMATTED APEX "ROW" CARD
*CALL ARRAYS
      WRITE (LUNLPM,40) ROWTYPE,ROWNAME
   40 FORMAT( X,A1,2X,A10)
      RETURN
      END
*DECK COLCARD   ......................................  *DECK COLCARD
      SUBROUTINE COL CARD  (COLNAME,ROWNAME,VALUE)
C...WRITES ONE FORMATTED APEX "COLUMN" CARD  ( OR "RHS" CARD)
*CALL ARRAYS
C...CHOOSE A FORMAT APPROPRIATE FOR THE SIZE OF THE COEFFICIENT.
      IF(VALUE.LT.100.)  GO TO 100
      WRITE (LUNLPM,40) COLNAME,ROWNAME,VALUE
   40 FORMAT( 4X,A10,A10,F12.3)
      RETURN
  100 WRITE (LUNLPM,140) COLNAME,ROWNAME,VALUE
  140 FORMAT( 4X,A10,A10,F12.7)
      RETURN
      END
*DECK SOLVLP    ......................................  *DECK SOLVLP
      SUBROUTINE  SOLV  LP
C..GET READY TO GO INTO APEX
*CALL ARRAYS
*CALL TEST
C...SAVE ALL INTERNAL VALUES     /ARRAYS/
      REWIND LUNSAVE
      BUFFER OUT (LUNSAVE,1) (FWA,LWA)
      IF(UNIT(LUNSAVE))  20,99,99
C..SET STATUS TO "ITERATE"
   20 CALL SET  (1RE,1)
      STOP  "SOLVE LP - GOING INTO APEX"

   99 STOP "SOLVELP - CANT SAVE ON INTERNAL FILE"
      END
*DECK READLP    ......................................  *DECK READLP
      SUBROUTINE  READ LP
C...READ IN LP SOLUTION FROM APEX FILE AND STORE AWAY LP AND DUAL VALUES
C....(SEE APEX MANUAL PAGE B-2  FOR THE APEX FILE FORMAT)
```

```
C......(LUNLPS  IS REWOUND BEFORE AND AFTER BEING READ BY THIS ROUTINE)
*CALL ARRAYS
*CALL TEST
      REWIND  LUNLPS
C..1ST RECORD
      READ (LUNLPS) WORD
      IF(EOF(LUNLPS).NE.0)  GO TO 998
    5 CALL LOG (16,16HPROBLEM NAME WAS )
      CALL LOG (10,WORD(1))
      VLOBJ=WORD(7)
C..CLEAR SECOND RECORD
      READ (LUNLPS)
C..OBJECTIVE ROW
      CALL  LP CHECK  ("OBJ")
      IF(WORD(7).LT.0)  GO TO 9999
      CALL LOG  (20,20H OPTIMAL LP SOLUTION     )
C...CONSTRAINT ROWS
C......WORD1=ROWNAME, WORD2=ROW ACTIVITY, WORD3=SLACK, WORD6= DUAL
      CALL GETROW ("LIMITN1",ACTIVTY,DUAL)
      CALL GETROW ("LIMITGAS",ACTIVTY,DUAL)
C
C..COLUMNS
C......WORD1=COLUMN NAME, WORD2=COLUMN ACTIVITY,  WORD6 = MARGINAL
      CALL GET COLM ("VN1R",ACTIVTY,VMARGNL)
      VLVN1R=ACTIVTY
      CALL  GET COLM ("VN2R",ACTIVTY,VMARGNL)
      VLVN2R=ACTIVTY
C.
      CALL LP CHECK("$$END$$")
C...GET RID OF SOLUTION FILE SO WE CANT ACCIDENTALLY READ IT AGAIN.
      REWIND LUN LP S
      ENDFILE  LUNLPS
      REWIND  LUNLPS
      RETURN

  998 CALL LOG  (31,"SOLUTION FILE FROM APEX IS BAD  ")
      STOP  "READ LP - CANT READ APEX SOLUTION"
C
 9999 CALL LOG (36,  "LP PROBLEM INFEASIBLE OR NON OPTIMAL"  )
      STOP  "READLP - LP PROBLEM INFEASIBLE OR NON OPTIMAL"
      END
*DECK LPCHECK   ......................................  *DECK LPCHECK
      SUBROUTINE  LP CHECK  (NAME)
C..TO READ IN THE NEXT APEX RECORD AND CHECK ITS NAME
*CALL ARRAYS
      READ (LUNLPS) WORD
      IF(IWORD(1).EQ.NAME)  RETURN
      CALL LINER (8)
      PRINT 40,IWORD(1),NAME
   40 FORMAT(*     ----ERROR IN 'READLP'.   MISSMATCHED NAMES*/
     ,         *         APEX RECORD   *,A10/
     ,         *         PROGRAM EXPECTS   *,A10)
C...GET A PROGRAM TRACEBACK
      CALL STRACE
      STOP "LPCHECK - LP NAMES DONT MATCH"
      END
*DECK GETROW    ......................................  *DECK GETROW
      SUBROUTINE  GET ROW  (ROW NAME,ACTIVTY,DUAL)
C..GETS NEXT APEX ROW RECORD, CHECKS ITS NAME AND RETURNS SOLUTION VALUES
C.......ROWNAME - ROW NAME (A10 FORMAT)   (INPUT)
C...    ACTIVTY - ROW ACTIVITY LEVEL   (OUTPUT)
C...      DUAL   - DUAL OR MARGINAL VALUE  (OUTPUT)
```

```
      INTEGER ROWNAME
*CALL ARRAYS
      CALL  LP CHECK  (ROWNAME)
      ACTIVTY=WORD(2)
      DUAL=WORD(6)
C...MORE VALUES COULD BE EXTRACTED IF THEY WERE NEEDED
      RETURN
      END
*DECK GETCOLM   ........................................  *DECK GETCOLM
      SUBROUTINE  GET COLM  (COLNAME,ACTIVTY,VMARGNL)
C...GET NEXT APEX COLUMN RECORD, CHECK ITS NAME AND RETURN SOLUTION VALUE
C.......COLNAME - COLUMN NAME (A10 FORMAT)   (INPUT)
C...      ACTIVTY - ACTIVITY LEVEL OF THIS COLUMN VARIABLE  (OUTPUT)
C...      VMARGNL - MARGINAL VALUE OF THIS COLUMN VARIABLE
      INTEGER COLNAME
*CALL ARRAYS
      CALL  LP CHECK  (COLNAME)
      ACTIVTY=WORD(2)
      VMARGNL=WORD(6)
C...MORE VALUES COULD BE EXTRACTED IF NEED BE...
      RETURN
      END
*DECK PRNTLP    ........................................  *DECK PRNTLP
      SUBROUTINE  PRNT  LP
C..PRINTS OUT  LP VARIABLES
*CALL ARRAYS
*CALL TEST
      CALL LINER (2)
      PRINT *,"                          ",VLVN1R  ,
     , "  VLVN1R  - VOLUME NAPTHA-1 REQUIRED     "
      PRINT *,"                          ",VLVN2R  ,
     , "  VLVN2R  - VOLUME NAPTHA-2 REQUIRED     "
      RETURN
      END
*DECK POSTLP    ........................................  *DECK POSTLP
      SUBROUTINE POST LP
C..TO CALCULATE THE VARIABLES WHICH ARE FUNCTIONS OF THE LP VARIABLES
C....JUST READ IN
*CALL ARRAYS
*CALL TEST
C...VOLUME OF BLEND
      VPVGAS=VLVN1R+VLVN2R+VIVCNR
C...OCTANE OF THE BLEND
      VPOCTB=(DFOCT1*VLVN1R+DFOCT2*VLVN2R+DFOCTC*VIVCNR) / VPVGAS
      RETURN
      END
*DECK PRPOST    ........................................  *DECK PRPOST
      SUBROUTINE  PR POST
C...PRINT  POST-LP VARIABLES
*CALL ARRAYS
*CALL TEST
      CALL LINER (2)
      PRINT *,"                          ",VPOCTB  ,
     , "  VPOCTB  - OCTANE VALUE OF THE CURRENT BLEND    "
      PRINT *,"                          ",VPVGAS  ,
     , "  VPVGAS  - VOLUME OF GASOLINE BEING PRODUCED    "
      RETURN
      END
*DECK TESTFS    ........................................  *DECK TESTFS
      SUBROUTINE TEST F S
C...TEST FOR FINAL SOLUTION
*CALL ARRAYS
```

```
*CALL TEST
C...OCTANE TEST.          BLENDED GAS   VS.   MARKET REQUIREMENT
      IF(VPOCTB.GE.DFOCTG)  GO TO 50
C...NOT GOOD ENOUGH
      CALL LOG (23,"SOLUTION NOT ACCEPTABLE")
      RETURN
C... O.K.
   50 CONTINUE
      CALL LOG (20,"SOLUTION ACCEPTABLE")
C... FORCE A GOTO
      CALL LOG (-41,"           GO TO      FINAL      (GENERATED)")
      CALL GOTO L  ("FINAL")
      RETURN
      END
*DECK ADJUST     ......................................  *DECK ADJUST
      SUBROUTINE  ADJUST
C...TO ADJUST THE ITERATIVE VARIABLES TOWORDS BETTER VALUES
*CALL ARRAYS
*CALL TEST
C...NEXT ITERATION
      ITERATN=ITERATN+1
      IF(ITERATN.GT.MAXIT)  GO TO 990
      CALL LINER (1)
      PRINT *,"                ITERATION NUMBER  ",ITERATN
C...INCREASE THE AMOUNT OF CRACKED NAPTHA BY 200 BARRELS
      VIVCNR=VIVCNR+200
      RETURN
C
  990 CALL LOG (45,"**** MAXIMUM NUMBER OF ITERATIONS EXCEEDED  ")
      STOP "ADJUST - MAXIMUM NUMBER OF ITERATIONS EXCCEDED"
      END
*DECK GOTOL      ......................................  *DECK GOTOL
      SUBROUTINE  GO TO L  (LABEL)
C...GO TO LABEL
*CALL ARRAYS
*CALL LABELS
      GOOD=.TRUE.
      IF(TEST)  20,100
C...IN TEST MODE STORE THE LABEL
   20 NLABEL2=NLABEL2+1
      IF(NLABEL2.GT.MAXLABL)  STOP "GO TO L - TOO MANY LABELS"
      LABEL2(NLABEL2)=LABEL
      RETURN
C.
C...IN  EXECUTE  MODE , GO FIND THE LABEL
  100 REWIND LUNACT
  110 READ (LUNACT,120) LABELA
  120 FORMAT(A10)
      IF(EOF(LUNACT).NE.0)  GO TO 999
C...IGNORE COMMENTS
  150 IF(LDCH(LABELA,1).EQ.1R*)  GO TO 110
      IF(LABELA.NE.LABEL)  GO TO 110
C...GOT IT
      BACKSPACE LUNACT
      RETURN
C.
  999 PRINT 994,LABEL
  994 FORMAT("   *** CANT FIND THE LABEL  [",A10,"]   JOB STOPPED.")
      STOP "GO TO L - CANT FIND LABEL"
      END
*DECK CSOLN      ......................................  *DECK CSOLN
      SUBROUTINE C SOLN
```

```
C...TO CALCULATE FINAL SOLUTION VARAIBLES
*CALL ARRAYS
*CALL TEST
C...GET PROFIT OF FULL MODEL
      VSPROFT=VPVGAS*DFSELLG - (VLVN1R*VRCOST1 + VLVN2R*DFCOST2
     ,                                  +VIVCNR*DFCOSTC )
      RETURN
      END
*DECK PRSOLN     ....................................   *DECK PRSOLN
      SUBROUTINE  PR SOLN
C...PRINT OUT CURRENT VALUES OF SOLUTION VARIABLES
*CALL ARRAYS
*CALL TEST
      PRINT *,"                          ",VSPROFT ,
     , "  VSPROFT - PROFIT OF CURRENT SOLUTION  "
      CALL LINER (1)
      RETURN
      END
*DECK PFINAL     ....................................   *DECK PFINAL
      SUBROUTINE P FINAL
C..TO PRINT NICELY THE FINAL VALUES OF INTEREST IN THE BIG MODEL
*CALL ARRAYS
*CALL TEST
      CALL LINER (5)
      PRINT *,"  PROFIT IS $",VSPROFT
      PRINT *,"   SELLING ",VPVGAS," BARRELS OF ",VPOCTB,
     ,                               " OCTANE BLENDED GASOLINE"
      PRINT *,"   BUYING ",VLVN1R," BARRELS OF NAPTHA-1 @ $",VRCOST1,
     ,                                          " PER BARREL"
      PRINT *,"   BUYING ",VLVN2R," BARRELS OF NAPTHA-2"
      PRINT *,"   BUYING ",VIVCNR," BARRELS OF CRACKED NAPTHA."
      RETURN
      END
*DECK ENDIT      ....................................   *DECK ENDIT
      SUBROUTINE  END IT
C...TO END EXECUTION OF THIS MODEL
C...SWITCH OFF GLOBAL LOOP CONTROL AND STOP
*CALL ARRAYS
*CALL TEST
      CALL SET  (1RE,2)
      STOP "       END BLEND"
      END
*DECK LINER      ....................................   *DECK LINER
      SUBROUTINE LINER  (NLINES)
C..TO COUNT LINES AND PRINT NEW PAGE WHEN NEEDED
*CALL ARRAYS
      LINENO=LINENO+NLINES
      IF(LINENO.GT.55)  CALL P TITLE
      RETURN
      END
*DECK PTITLE     ....................................   *DECK PTITLE
      SUBROUTINE PTITLE
C..PRINTS TITLE AND PAGE HEADING
*CALL ARRAYS
      PAGENO=PAGENO+1
      PRINT 40,DDATE,DTIME,PAGENO,TITLE
   40 FORMAT(*1            B L E N D    V1*,
     ,  40X,A10,2X,A10,
     ,*     PAGE*,I3,/
     ,  /5X,8A10,/ )
      LINENO=3
      RETURN
      END
*DECK LOG       ......................................  *DECK LOG
      SUBROUTINE LOG  (NCHARS,MESSAGE)
C...TO LOG A STATUS MESSAGE ON OUR SOLUTION LOG FILE
C......IF NCHARS  IS NEGATIVE, FLAG THE MESSAGE WITH CPU TIME.
      DIMENSION MESSAGE(8)
      NWORDS=(IABS(NCHARS)+9) / 10
      IF(NCHARS.GT.0)  GO TO  100
C..FLAG
      CALL SECOND  (CPTIME)
      CALL LINER (1)
      PRINT 40,CPTIME,(MESSAGE(I),I=1,NWORDS)
   40 FORMAT(4H >>>,T90,*CPU=*,F5.2,T5,8A10)
      RETURN
C.
  100 PRINT 140,(MESSAGE(I),I=1,NWORDS)
  140 FORMAT(16X,8A10)
      CALL LINER (1)
      RETURN
      END
*DECK ACTEOF    ......................................  *DECK ACTEOF
      SUBROUTINE  ACT EOF
C...TO ACT ON END-OF-FILE ON THE COMMAND FILE
*CALL ARRAYS
      IF(TEST)  100,2000
C...TEST MODE (LISTING OUT DIRECTIVES)
  100 CONTINUE
C...CHECK FOR LABEL ERRORS
      CALL  MATCH L  (NERROR)
      IF(NERROR.EQ.0)  GO TO 1000
C...COMAND ERRORS
      CALL LOG (1,1H )
      CALL LOG (-37,37H***ERRORS IN COMMANDS.  JOB STOPPED.  )
      STOP "ACTEOF - ERRORS IN COMMANDS"
C.
C.. NO ERRORS.  NOW GO BACK AND EXECUTE THE COMMANDS FOR REAL
 1000 TEST=.FALSE.
      REWIND LUNACT
      CALL LOG (1,1H )
      CALL LOG (-38,"NO COMMAND ERRORS.  EXECUTION STARTED.")
      DO 1050 I=1,8
 1050 TITLE(I)=1H
      TITLE(1)=10HEXECUTION
      TITLE(2)=         10HLOG.
      RETURN
C.
C...END OF REAL COMMANDS
 2000 CALL LOG (-41,"          END   (GENERATED BY END-OF FILE)")
      CALL END IT
      END
*DECK ERROR     ......................................  *DECK ERROR
      SUBROUTINE ERROR
C...COMMAND ERROR
*CALL ARRAYS
      PRINT *," ------------**********  UNRECOGNISED COMMAND"
      CALL LINER (1)
      NERROR=NERROR+1
      IF(NERROR.GT.50)  STOP "ERROR - TOO MANY COMMAND ERRORS"
      RETURN
      END
*DECK CHECKL    ......................................  *DECK CHECKL
      SUBROUTINE  CHECK L  (LABEL)
```

```
C...CHECK LABEL FIELD OF ACTION CARD
*CALL LABELS
      IF(LABEL.EQ.10H          )  RETURN
      NLABEL1=NLABEL1+1
      IF(NLABEL1.GT.MAXLABL)  STOP "CHECK L - TOO MANY LABELS"
      LABEL1(NLABEL1)=LABEL
      RETURN
      END
*DECK MATCHL     ......................................  *DECK MATCHL
      SUBROUTINE  MATCH L  (NERRORS)
C...TO MATCH LABELS TO SEE IF ALL REFERENCED LABELS EXIST
*CALL LABELS
      IF(NLABEL2.EQ.0)  RETURN
C..FOR EACH LABEL REFERENCED, CHECK IT EXISTS
      DO 100 I=1,NLABEL2
      IF(NLABEL1.EQ.0)  GO TO 40
       DO 20 J=1,NLABEL1
        IF(LABEL1(J).EQ.LABEL2(I))  GO TO 100
   20  CONTINUE
   40 CALL LINER (1)
      PRINT 60,LABEL2(I)
   60 FORMAT("   *** MISSING LABEL  (",A10,")")
      IF(LABEL2(I).EQ."FINAL")  PRINT *,
     ,          "       (THIS LABEL IS REFERENCED BY THE 'TEST FOR FINAL
     ,SOLUTION' CARD)"
      NERRORS=NERRORS+1
  100 CONTINUE
      RETURN
      END
```