LINPROG: A linear-programming code developed at Risø

Kirkegaard, Peter; Lang Rasmussen, Ole

Publication date:
1990

Document Version
Publisher's PDF, also known as Version of record

LINPROG:
A Linear-Programming Code Developed at Risø

Risø-M-2797

Peter Kirkegaard and Ole Lang Rasmussen

Risø National Laboratory, DK-4000 Roskilde, Denmark
March 1990

Abstract A computer code LINPROG written in Standard FORTRAN 77 has been developed at Risø for solving medium- to large-scale linear programming problems. It runs primarily on a VAX-8700 computer, but also on other systems where virtual memory is available. LINPROG uses the revised simplex method with the Forrest-Tomlin updating scheme of the inverse basis. Sparse-matrix techniques are applied throughout. A comprehensive test and verification study has been performed with data sets provided by local users and with data sets available in the literature.

ISBN 87-550-1541-7
ISSN 0418-6435

Grafisk Service Risø 1990

Contents

1 Introduction 5

2 LINPROG User's Guide 7
- 2.1 Preparation of input: Control File . . . 7
- 2.2 Preparation of input: Matrix File . . . 9
- 2.3 Interpretation of output . . . 12
- 2.4 Program files . . . 16
- 2.5 Dump/restart facility . . . 16
- 2.6 Binary output and report writers . . . 17
- 2.7 LINPROG running on a VAX computer . . . 19
- 2.8 Installation of LINPROG . . . 20

3 The Simplex Method: An Overview 20
- 3.1 Standard form of the LP . . . 21
- 3.2 The basis exchange mechanism . . . 21
- 3.3 The simplex algorithm . . . 23

4 Useful Matrix Relations 24
- 4.1 The Sherman-Morrison identity . . . 24
- 4.2 Elementary matrices . . . 24
- 4.3 Column replacement updating . . . 25
- 4.4 Permutation matrices . . . 25
- 4.5 LU-factorization . . . 26
- 4.6 Decomposition of triangular matrices into elementary matrices . . . 26

5 Elementary Product Forms 27
- 5.1 The standard product form . . . 27
- 5.2 Product form of the LU type . . . 28

6 Re-inversions 30
- 6.1 Decomposition arithmetics and eta vectors . . . 30
- 6.2 Pre-ordering of rows and columns . . . 31
- 6.3 Pivot selection strategy for sparsity preservation . . . 31
- 6.4 Modifications to ensure numerical stability . . . 33
- 6.5 Sparse-matrix implementation details . . . 33
- 6.6 Algorithmic description of the inversion . . . 34
- 6.7 Possible improvements . . . 35

7 The Forrest-Tomlin Procedure 35
- 7.1 General outline . . . 35
- 7.2 Updating of product representation . . . 38
- 7.3 Interfacing Forrest-Tomlin with simplex . . . 41
- 7.4 Implementation . . . 42

8 Miscellaneous Features 43
- 8.1 Bounds and ranges . . . 43
- 8.2 Simplex initialization: CRASH and zero-slack elimination . . . 47
- 8.3 Finding a first feasible solution: Phase 1 . . . 48
- 8.4 Pricing strategies. Cycling. . . . 53
- 8.5 Scaling, LP tolerances, and numerical stability. . . . 54

8.6 Sparse-matrix and other programming techniques 57

9 Test of LINPROG **58**
9.1 Sources of test problems 58
9.2 List of test results 59
9.3 Comments on the test problems 63

10 Summary and Conclusions **64**

Acknowledgements **65**

References **66**

APPENDIX **68**
A1: Report writers in FORTRAN and PASCAL 68
A2: Algorithmic description of LINPROG 71
A3: Subroutines in LINPROG 73

Index **75**

1 Introduction

The computer code LINPROG is intended for solving medium- and large-scale problems of *linear programming* (LP) in which we want to minimize (or maximize) a given linear function, called the *objective function*, subject to a set of linear equations and inequalities, called *constraints* (the equivalent terms *restraints* and *restrictions* are also in common use.) Let us write our *linear program* in the following form:

$$\left.\begin{array}{lll} \text{Minimize} & z = \sum_{j=1}^{n_s} c_j x_j & \\ \text{subject to} & \sum_{j=1}^{n_s} a_{ij} x_j \quad \mathrm{R}_i \quad b_i & (i = 1, \ldots, L) \\ \text{and} & x_j \geq 0 & (j = 1, \ldots, n_s). \end{array}\right\} \tag{1}$$

The symbol R_i stands for a *restriction type*, which must be either $\leq$, $\geq$, $=$, or dummy (meaning there is no restriction). The problem (1) has n_s nonnegative *structural variables* and L constraints.

Linear programming problems arise in many different fields of application, and typically the LP models come into use where resources are limited in some way or another. At Risø LINPROG and its predecessors have been used to solve linear programs for finding optimal fuel management schemes in a nuclear reactor, for predicting flow distributions of various types of energy in complex nationwide energy systems, and for allocating scarce resources in economic planning.

We give below an illustrative example of a small LP problem and choose for this purpose the classical *diet problem*. Suppose for example that the budget manager of a nursing home wishes to purchase, at minimum cost, suitable quantities of a number of available food types, so that the daily diet provides the occupants with prescribed minimum values of nutrients (protein, vitamin, etc.). In our example, which is taken from [1], there are $n_s = 3$ food types: poultry, spinach, and potatoes. The daily requirements of nutrients per person are at least 65 grams of protein, 90 grams of carbohydrate ("energy"), 200 milligrams of calcium, 10 milligrams of iron, and 5000 international units of vitamin A. Table 1 shows the costs and nutritive food values per gram of each food type. If the quantities of poultry, spinach, and potatoes, are denoted x_1, x_2, x_3, respectively, we can formulate the diet problem as a linear program conforming with

	Poultry	Spinach	Potatoes
Costs (cents)	0.40	0.15	0.10
Protein (g)	0.20	0.03	0.02
Carbohydrate (g)	0	0.03	0.18
Calcium (mg)	0.08	0.83	0.07
Iron (mg)	0.014	0.02	0.006
Vitamin A (I.U.)	0.80	73	0

Table 1. Nutritive value of foods (from Nazareth)

(1):

$$
\left.
\begin{array}{lll}
\text{Minimize} & z = & 0.40x_1 + 0.15x_2 + 0.10x_3 \\
\\
\text{subject to} & & 0.20x_1 + 0.03x_2 + 0.02x_3 \geq 65 \\
& & 0.03x_2 + 0.18x_3 \geq 90 \\
& & 0.08x_1 + 0.83x_2 + 0.07x_3 \geq 200 \\
& & 0.014x_1 + 0.02x_2 + 0.006x_3 \geq 10 \\
& & 0.80x_1 + 73x_2 \geq 5000 \\
\text{and} & & x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0.
\end{array}
\right\} \qquad (2)
$$

We shall return to this example in connection with the User's Guide for LINPROG, Section 2.

The formulation (1) may be considered as the default problem setup for LINPROG, for as we shall see later, the code can deal with a number of modifications and extensions to (1). For example, it can be used to maximize instead of minimize. Moreover the nonnegativity conditions $x_j \geq 0$ for structural variables can be replaced by two-sided bounds $\ell_j \leq x_j \leq u_j$, or x_j may be either a fixed or a free variable. Other common LP extensions, such as parametric and integer programming, are not included in LINPROG.

LINPROG solves the LP problem by the *simplex method* (see Section 3). It is a "stand-alone" code written in standard-conforming ANSI FORTRAN 77, except for a few small subroutines (see Appendix A3). However, its use is not restricted to a FORTRAN environment. All data transfers between LINPROG and the user are made via files. The input matrix format complies with the de-facto standard for commercial LP codes, in particular IBM's system MPSX[2].

At Risø the LINPROG code is installed at a VAX-8700 computer, but it is able to run on other systems as well with only minor modifications, provided virtual memory is available. In fact we have successfully tested LINPROG on an Apollo DN10000, a SUN 3/75, and a UNISYS A6 computer. (We have also made a PC-version of the code with reduced array bounds, but this is able to solve only small problems.)

LINPROG has undergone a great deal of development, enhancement and modification over the years, in order to keep abreast with the demands from the users for solving larger and larger problems in an efficient and reliable way. The current version is able to solve LP problems that contain thousands of variables and constraints, hence with perhaps many millions of elements a_{ij} in the constraint matrix. In typical practical problems, only a small fraction of these elements will differ from zero. LINPROG takes advantage of this by using "sparse-matrix" techniques.

In addition to LP test problems supplied by Risø researchers, we have collected a great deal of sample cases from outside. The NETLIB collection of LP data sets, which is in the public domain [3], is one of the most important and has proved very valuable in testing and debugging LINPROG.

There are several reasons why we preferred to develop and maintain our own LP system at Risø instead of buy or rent a commercial one. Apart from saving the cost of an external system, we obtain the flexibility of being able to install the code on different computers, when this is appropriate. With our own code it is also easier to provide users with programmatic interfaces to their application software.

One of our main design criteria for LINPROG was to keep the program manageable in size. We have therefore concentrated on implementing what we consider to be the most important facilities, abandoning many of the more sophisticated capabilities found in larger commercial systems.

The contents of this report is organized in such a way that Section 2, the LINPROG User's Guide, contains all the practical information a potential user will need in order to prepare an LP input for LINPROG, solve the problem, and interpret the results. The rest of the report contains a documentation of the mathematical methods, supplemented with some programmatic details, and a short account of test results. This material is not needed for routine running of LINPROG.

2 LINPROG User's Guide

At this place we shall present a User's Guide for LINPROG, and in doing so we make repeated use of the small sample problem in the Introduction.

We shall give instructions on how to prepare input data for the code, and we shall also explain the various parts of the output produced.

In the following a number of technical LP phrases will be used. Although they will be explained in later sections, they might confuse the unprepared user, so we shall give a brief LP "vocabulary" below:

Activity:	Value of some LP variable.
Basic index set:	The set of variable indices corresponding to the current basis.
Basis:	The subset of the LP variables that determine the current solution are said to be *basic* or to form the *basis*. The precise definition is given in Section 3.
Constraint Matrix:	Totality of the coefficients a_{ij} in (1).
Feasibility:	A set of variables is said to define a *feasible point* or a *feasible solution* if all constraints are satisfied.
Infeasibility:	We have *infeasibility* when not all constraints can be satisfied.
Inversion:	The subset of basic variables defines the *basis matrix*. From this we may obtain the solution by a matrix inversion. This is discussed in Section 6.
Rhs:	Right-hand side of the constraint matrix.

2.1 Preparation of input: Control File

For a standard run of LINPROG you need two files for your input data. The first one is called the *Control File*. It contains a verbal instruction set for solving the LP problem(s) in question, with one instruction per line. Each instruction contains a ***keyword***, or a keyword followed by a value. An example:

```
SOLUTION
MAXITS 10000
EXEC
BINOUT
DUMP
EXEC
PEND
```

The position of a keyword within a line is immaterial, only the line is limited to 72 characters. To be recognized, keywords must be unabridged. All letters must be in uppercase. The keywords EXEC and PEND are special in the sense that they control the job stream: EXEC initiates another LP, and PEND tells that there are no more problems to solve. The other keywords are called *numerical* if they are followed by a value, otherwise they are *action* keywords. Commonly, the numerical keywords and the action keywords are called *descriptive* keywords. Between the top of the Control File and the first EXEC, or between two EXECs, the descriptive keywords may come in any order, and this order has no bearing on the order of performing the corresponding LINPROG tasks. Together they specify modes for solving a single problem. All descriptive keywords are optional.

We give below the complete list of action keywords available in the current release of LINPROG:

ECHO	Prints an echo of the Matrix File (Section 2.2)
ELTAB	Prints column and row counts of nonzero LP matrix elements

PICTURE Prints a picture of the distribution of nonzeros in the LP matrix

DUMP Dumps the basic index set on a file

RESTART Restarts LINPROG by reading the basic index set from a file

MAX Tells that this LP is a maximization problem (by default it would be minimization)

SOLUTION The solution will be printed, both its row and its column part (if omitted, only the optimal value will be printed)

BINOUT Stores the solution as unformatted (binary) records on an output communication file which can later be read by another program

We have designed the default settings in LINPROG in such a way, that you probably will need the numerical keywords only on rare occasions. They fall in two groups. In the first group the associated values are integers, in the other they are floating-point quantities. First we list the 5 keywords with integral values:

MAXITS Gives an upper limit of simplex iterations. By default, or by specifying 0, no limit is assumed.

MAXCPM Gives an upper limit of the CPU time allocated to a *single* problem. By default, or by specifying 0, no limit is assumed (the time will be checked after each re-inversion, cf. Section 6.)

LOGFRQ Produces a log of the simplex iteration at every LOGFRQ iteration step. By default, or by specifying 0, no iteration printout will be given. Negative values of LOGFRQ are also allowed. In that case |LOGFRQ| gives the frequency of the iteration printout, and in addition a map of the variables in the initial basis is printed (the numbering is the same as for the ROWS and COLUMNS section output explained in Section 2.3).

MITRE Nnmber of simplex iterations between two consecutive re-inversions (Section 6). By default MITRE = 100. Maximum value is MITRE = 1000.

MSCALE Scaling option for constraint matrix.
MSCALE = 0: No scaling
MSCALE = 1: Row scaling
MSCALE = 2: Column scaling
MSCALE = 3: Row and column scaling
By default, MSCALE = 1. The scaling methods are discussed in Section 8.5.

The integer specifying the value may be placed anywhere after the keyword up to position 72. For example:

```
MAXCPM    120
          MITRE50
   LOGFRQ -10
```

Here we have set an upper bound of 120 minutes CPU time for the problem, we have changed the re-inversion frequency from 100 to 50 iterations, and we ask for a printout of simplex iterations at every 10 steps, including a map of the initial basis.

The numerical keywords with floating-point values are all used to redefine program tolerances. These are explained in Section 8.5, and the default settings are given in Table 3 there. The 9 keywords BIG, EPSCHC, EPSCHR, EPSFEA, EPSINA, EPSLU, EPSPIV, EPSRIN, and ZERPIV are the same as the FORTRAN names in Table 3. New values may be entered in "scientific" notation using the FORTRAN input convention. As an example,

```
   EPSFEA  1.0E-8
```

raises the feasibility tolerance from its default value 10^{-10} to 10^{-8}.

An equal sign is allowed between a numerical keyword and its value, but is not required. For example:

```
EPSCHC = 1.0E-9
MSCALE= 3
```

Finally we repeat the two mandatory keywords used for job control:

EXEC	Tells LINPROG to proceed with one LP problem
PEND	Tells LINPROG to stop the job stream

2.2 Preparation of input: Matrix File

The second input file is called the *Matrix File*. Its organisation must comply with the Standard MPS format. This format, which is used by most commercial LP systems, was originally developed by IBM, and in the following specifications we shall adhere to the prescriptions given for the MPSX system of IBM (cf. the MPSX manual [2]). As an illustration we show a printout of the Matrix File corresponding to our sample problem:

```
NAME          DIETNAZA
ROWS
 N  COST
 G  PROTEIN
 G  ENERGY
 G  CALCIUM
 G  IRON
 G  VITAMINA
COLUMNS
    POULTRY   COST          0.40   PROTEIN       0.20
    POULTRY   CALCIUM       0.08   IRON         0.014
    POULTRY   VITAMINA      0.80
    SPINACH   COST          0.15   PROTEIN       0.03
    SPINACH   ENERGY        0.03   CALCIUM       0.83
    SPINACH   IRON          0.02   VITAMINA        73
    POTATOES  COST          0.10   PROTEIN       0.02
    POTATOES  ENERGY        0.18   CALCIUM       0.07
    POTATOES  IRON         0.006
RHS
    DEMANDS   PROTEIN        65.0   ENERGY        90.0
    DEMANDS   CALCIUM       200.0   IRON          10.0
    DEMANDS   VITAMINA     5000.0
ENDATA
```

The first record is a NAME line which gives the data set a name. It will appear as an identification of the LINPROG output. The name chosen here is DIETNAZA. The last record in the file is an ENDATA line which signals the end of the data set. In between there are a number of "sections", each initiated by a headline. There are five possible sections, corresponding to the headlines ROWS, COLUMNS, RHS, RANGES, and BOUNDS, in that order. The first three are mandatory, the last two are optional (neither of them is present in this example). Lines other than headlines contain the problem data. They use six predefined position fields:

Field	Position range	Contents
1	2 - 3	Indicator field (N, E, L, G, UP, LO, etc.)
2	5 - 12	Name field
3	15 - 22	Name field
4	25 - 36	Value field
5	40 - 47	Name field
6	50 - 61	Value field

Names can contain any ASCII character and must not exceed 8 characters in length. Indicators and names are left-justified in their fields. Values are numeric and must not exceed 12 characters in length; they may be integral or may contain a decimal point. Even an exponential field is allowed; in fact the numeric input is compatible with formatted FORTRAN inputting.

The ROWS Section: This section is mandatory and defines the rows in the problem. These are the objective function and the problem constraints. In the ROWS Section, a line is given for each row. Such a line contains:

- The restriction type of the row (in position 2):
  N = Non-constrained row, usually objective function
  E = Equality constraint
  L = Less-than-or-equal-to constraint
  G = Greater-than-or-equal-to constraint

- The name of the row (in Field 2, position 5 - 12)

The lines in this section may appear in any order. In particular, the objective row need not be the first one. LINPROG requires precisely one object function. The convention is adopted that the *first* row of type N is selected as the object function.

The COLUMNS Section: This section is mandatory. It defines a name for each of the structural variables and lists the *nonzero* entries in the corresponding column position of the objective function and the constraint matrix. All the elements for a column must be grouped together, but they need not appear in the same order as in the ROWS Section. For each line in the COLUMNS Section:

Field 2 contains the name of the variable.

Field 3 contains the name of a row in which the variable has a nonzero coefficient.

Field 4 contains the value of this coefficient.

Fields 5 and 6 are optional. If used, they contain another row name and corresponding coefficient.

The RHS Section: This section is mandatory and deals with the rhs (right-hand side) of the problem. It gives the rhs a name (DEMANDS in our example) and specifies all the nonzero rhs values. Except for Field 2, which contains the name of the right-hand side, the fields in the lines of the RHS Section contain the same type of information as the corresponding fields of the COLUMNS Section. LINPROG allows the user formally to specify multiple right-hand sides with different names, but the program can deal with only a single rhs aud will in that case process the *first* one.

The RANGES Section: This section is optional. It contains the ranges for rhs values. To give an example, suppose that we modify the diet problem (2) in the Introduction by adding the coustraint that the daily consumption of carbohydrate ("energy") should not exceed 120 g. With the constraint already present this meaus that the carbohydrate consumption must lie in the interval [90, 120]. Instead of adding a new constraint, which would involve more computations, we give only the *range* for the rhs. This is done in the RANGES Section as follows:

```
NAME          ...
ROWS
...
COLUMNS
...
RHS
...
RANGES
    RANGE1    ENERGY          30.0
ENDATA
```

In this example, there is one rauge set name, RANGE1, which appears in Field 2. Field 3 contains the name of the row to which the range applies and Field 4 the value of the range.

Note that it is possible to define several ranges for a given problem (LINPROG also allows the user to specify formally more than one set of ranges with different names, but only the *first* set will be processed by the code).

In general, the RANGES Section is used to define constraints of the form

$$\ell \leq \sum a_j x_j \leq u, \tag{3}$$

by specifying one of the bounds ℓ or u in the RHS Section (as b say), together with a *range value* r in the RANGES Section. This range value may be negative, and LINPROG uses the following (historical) rules for calculating ℓ and u, given b and r:

Type	Sign of r	Lower limit, ℓ	Upper limit, u
E	+	b	$b+\|r\|$
E	−	$b-\|r\|$	b
G	+ or −	b	$b+\|r\|$
L	+ or −	$b-\|r\|$	b

The BOUNDS Section: This section is optional. It defines bounds for the values of the variables. If no bounds are given for a variable, it is assumed to have a lower bound of zero and no upper bound. To give an example, suppose that we modify the diet problem (2) in the Introduction by adding the constraint that the daily poultry supply is limited to 200 g per inhabitant. We may then add the constraint POULTRY $\leq$ 200 via a BOUNDS Section input, which is cheaper than expressing the bound as an additional constraint row. We then get the following modification of the Matrix File:

```
NAME          ...
ROWS
...
COLUMNS
...
RHS
...
BOUNDS
 UP LIMIT     POULTRY          200
ENDATA
```

In general, each data line in the BOUNDS Section defines a bound for one variable; a number of bound types other that UP may be imposed. For each line in the BOUNDS Section:

Field 1 — specifies the type of bound:
UP for an upper bound, $0 \leq x \leq u$.
LO for a lower bound, $x \geq \ell$.
FX for a fixed-variable bound, $x = a$.
FR for a free variable, $-\infty < x < +\infty$.
PL for a nonnegative variable, $0 \leq x < +\infty$ (default).
MI for a nonpositive variable, $-\infty < x \leq 0$.

Field 2 — defines the name of the bound set.

Field 3 — defines the name of the variable to which the bound applies.

Field 4 — specifies the value of the bound (omitted for FR, PL, and MI).

In our example there is only one bound set called LIMIT containing a single bound. It is possible, however, to define several bounds for a given bound set (LINPROG also allows the user to specify formally more than one set of bounds with different names, but only the first set will be processed by the code.) By using both LO and UP you may specify two-sided bounds like $\ell \leq x \leq u$.

Remember that if both a RANGES and a BOUNDS Section are present, RANGES must come before BOUNDS.

The Matrix File may contain more than one data set. Each data set is initiated by a NAME line and ended by an ENDATA line; the number of data sets should match the number of EXEC lines in the Control File.

2.3 Interpretation of output

We shall now describe the *Result File* printed by LINPROG. Our first sample output is from the diet problem without ranges and bounds. It looks as follows:

```
*****************************************************************
*                      LINPROG VERSION 9003                     *
*                 LINEAR PROGRAMMING CODE WRITTEN BY            *
*              PETER KIRKEGAARD AND OLE LANG RASMUSSEN          *
*        RISOE NATIONAL LABORATORY, DK-4000 ROSKILDE, DENMARK   *
*                          COPYRIGHT (C) 1990                   *
*****************************************************************

DATE OF PROGRAM RUN: 90/03/12 (DAY NUMBER 90/071)

LINPROG 9003 COMPILATION OF JOB FROM COMMAND FILE

        SOLUTION

NAME OF DATA SET: DIETNAZA

 PROBLEM STATISTICS

    6 LP ROWS       9 VARIABLES      22 LP ELEMENTS  DENSITY =  40.74

 THESE STATISTICS CONTAIN ONE SLACK VARIABLE FOR EACH ROW

                      TOTAL  NORMAL  .FREE.   FIXED BOUNDED
 ROWS    (LOG.VAR.)       6       5       1       0       0
 COLUMNS (STR.VAR.)       3       3       0       0       0

PROGRAM TOLERANCES:
BIG    = 1.0E+30     EPSCHC= 1.0E-10     EPSCHR= 1.0E-10
EPSFEA= 1.0E-10     EPSINA= 1.0E-08     EPSLU = 1.0E-13
EPSPIV= 1.0E-09     EPSRIN= 1.0E-02     ZERPIV= 1.0E+00

OTHER PARAMETERS:
LOGFRQ=    0        MAXCPM=    0        MAXITS=    0
MITRE =  100        MSCALE=    1

   0 ZERO SLACKS WERE ELIMINATED IN PHASE 0

ITERATION NUMBER =     1

END OF PHASE 1  (ESTABLISHMENT OF FEASIBILITY)

ITERATION NUMBER =     3

END OF PHASE 2

SOLUTION   (OPTIMAL)

        ...NAME...     ...ACTIVITY...       DEFINED AS

        FUNCTIONAL         174.70817        COST
        RESTRAINTS                          DEMANDS


LINPROG 9003 EXECUTION

SECTION 1 - ROWS
```

```
 NUMBER  ...ROW..  AT  ...ACTIVITY...  SLACK ACTIVITY  ..LOWER LIMIT.  ..UPPER LIMIT.  .DUAL ACTIVITY

      1  COST      BS      174.70817      174.70817-          NONE           NONE         1.00000
      2  PROTEIN   LL       65.00000               .       65.00000           NONE         1.67315-
      3  ENERGY    LL       90.00000               .       90.00000           NONE         0.21401-
      4  CALCIUM   BS      205.49125        5.49125-      200.00000           NONE               .
      5  IRON      LL       10.00000               .       10.00000           NONE         4.66926-
      6  VITAMINA  BS    13621.59533     8621.59533-     5000.00000           NONE               .


LINPROG 9003 EXECUTION

SECTION 2 - COLUMNS


 NUMBER  .COLUMNS  AT  ...ACTIVITY...  ..INPUT COST..  ..LOWER LIMIT.  ..UPPER LIMIT.  .REDUCED COST.

      7  POULTRY   BS      250.48638         0.40000              .           NONE               .
      8  SPINACH   BS      183.85214         0.15000              .           NONE               .
      9  POTATOES  BS      469.35798         0.10000              .           NONE               .

MAX VIOLATION OF RESTRAINTS WAS  1.776E-15
IT OCCURRED IN THE RESTRAINT     PROTEIN    G   6.500E+01

TIME FOR THIS JOB:     0.05 SECONDS.


DATE OF PROGRAM RUN: 90/03/12 (DAY NUMBER 90/071)

LINPROG 9003 COMPILATION OF JOB FROM COMMAND FILE


                          OUTPUT SUMMARY
  CASE    ROWS COLS  ELEM  PH.0 PH.1 ITNS  VIOL         OPTIMUM        CPU-SEC
DIETNAZA     6    3    16     0    1     3 1.8E-15  1.7470817120623E+02    0.05

                                                GRAND TOTAL TIME     0.12
```

Our second sample output is from the modified diet problem with RANGES and BOUNDS. To save space we print only the solution part for this problem:

```
.........
SOLUTION   (OPTIMAL)

         ...NAME...     ...ACTIVITY...       DEFINED AS

         FUNCTIONAL         205.00000       COST
         RESTRAINTS                         DEMANDS
         BOUNDS....                         LIMIT
         RANGES....                         RANGE1


LINPROG 9003 EXECUTION

SECTION 1 - ROWS


 NUMBER  ...ROW..  AT  ...ACTIVITY...  SLACK ACTIVITY  ..LOWER LIMIT.  ..UPPER LIMIT.  .DUAL ACTIVITY

      1  COST      BS      205.00000      205.00000-          NONE           NONE         1.00000
      2  PROTEIN   LL       65.00000               .       65.00000           NONE         5.00000-
A     3  ENERGY    LL       90.00000               .       90.00000      120.00000               .
      4  CALCIUM   BS      511.31250      311.31250-      200.00000           NONE               .
      5  IRON      BS       16.48750        6.48750-       10.00000           NONE               .
      6  VITAMINA  BS    41222.50000     36222.50000-     5000.00000           NONE               .


LINPROG 9003 EXECUTION

SECTION 2 - COLUMNS
```

```
NUMBER  .COLUMNS  AT  ...ACTIVITY...  ..INPUT COST..  ..LOWER LIMIT.  ..UPPER LIMIT.  .REDUCED COST.

     7  POULTRY   UL      200.00000        0.40000              .       200.00000        0.60000-
     8  SPINACH   BS      562.50000        0.15000              .            NONE              .
     9  POTATOES  BS      406.25000        0.10000              .            NONE              .
.....
```

The output from LINPROG begins with a heading followed by a printout of the lines of the Control File for the present LP problem, and the data set name defined in the Matrix File. Next comes some problem statistics in the same format as for example MPSX [2] uses. In the present case we have 5 normal rows and 1 free row. The phrases "normal" and "free" mean inequality-constrained and unconstrained, respectively. Our example has the objective row as the only unconstrained row. After a list of numerical parameter settings, the number of iterations in the various phases of the simplex procedure is printed. This part of the output is usually unimportant in standard runs of LINPROG. In any case, its interpretation becomes clear if you consult the subsequent sections of this report. Notice, however, the message "ESTABLISHMENT OF FEASIBILITY" (after END OF PHASE 1) telling that our LP is indeed feasible; if this were not the case a message "THIS PROBLEM HAS NO FEASIBLE SOLUTION" would be issued. After this comes the solution printout summary. First there is a heading part containing

- The name SOLUTION with a qualifying text (OPTIMAL or INFEASIBLE)
- The value and name of the objective function
- The name of the right-hand side. If bounds and ranges sets are present, the name of these.

After the heading we find the ROWS Section and COLUMNS Section output (but only if the keyword SOLUTION occurs in the Control File). The two sections have similar structure and are formatted in the same way as the MPSX code [2]. Each section is printed as a table of eight columns, with one row of the table corresponding to one row (or column) of the problem. Each row and column is formally associated with a variable. Below we give a short description of the items contained in the eight columns:

1. NUMBER — The sequential number given by LINPROG to the row or column. If the problem contains m rows and n columns, the rows are numbered from 1 through m and the columns from $m+1$ through $m+n$. In our two examples, the numbers of row VITAMINA and SPINACH are 6 and 8, respectively.
2. NAME — This is the eight-character name given to the row or column.
3. AT — This is a two-character code denoting the status that the row or column variable has in the solution. The codes and their meanings are:

   BS — basic and feasible

   EQ — nonbasic and fixed

   UL — nonbasic at upper limit

   LL — nonbasic at lower limit

   ** — infeasible

   There are two different nonbasic status indicators UL and LL, because the LP variables may be upper- and lower-bounded (see also Section 8.1).
4. ACTIVITY — This is the value that the row or column variable takes in the solution:
   - For columns, the activity is simply the value given to the column in the solution. For example, the value of POULTRY is 250.48638 (in gram units).
   - For rows, the activity is defined as the sum of the products of constraint coefficients and the column activities. For example, the row ENERGY, which defines the constraint (cf. (2)) $0.03 \times \text{SPINACH} + 0.18 \times \text{POTATOES} \geq 90$, has the activity $0.03 \times 183.85214 + 0.18 \times 469.35798 = 90$.

5. SLACK ACTIVITY (ROWS Section) and INPUT COST (COLUMNS Section) — For a given row, the "slack activity" is the difference between the right-hand side element for the row and the activity of the row. For a column, the "input cost" is the corresponding element in the objective function.

6. LOWER LIMIT — The lower bound on the row or column activity.

7. UPPER LIMIT — The upper bound on the row or column activity.

8. DUAL ACTIVITY (ROWS Section) — The value given under "dual activity" is meaningful only for rows at one of its bounds. If such a bound could be removed, we would expect a decrease of the total cost, and the dual activity gives the cost reduction per unit relaxation of the limit. In our example the diet must contain at least 65 g protein. In fact, the optimal solution contains precisely 65 g. If the required minimnm of 65 g could be lowered by 1 g, the dual activity indicates a saving of 1.67 cents. But notice that this figure is actually a rate of change, valid close to the solution. Anyway, the dual activity can help to suggest alterations of the specified model constraints.

9. REDUCED COST (COLUMNS Section) — For a given column, the "reduced cost" is a quantity with a similar meaning as the dual activity for a row. Consider for example the output of the second problem, where POULTRY has an activity of 200 (gram) in the optimal solution. Its reduced cost is -0.6, which indicates that if one gram of POULTRY were withdrawn from the diet, a cost reduction of 0.6 cents would be gained (in general the reduced cost is the *decrease* in objective function per unit *increase* of the variable; in our case POULTRY is at its upper bound and can only be decreased, which explains the negative sign of the reduced cost).

Both dual activity and reduced cost can be thought of as "unit costs", the former valid for a row, the latter for a column. Dual activities are sometimes called "shadow prices" or "simplex multipliers". Notice that there is some ambiguity in the sign definition of dual activities among commercial LP systems. We follow here MPSX [2], but MINOS [4] has the opposite sign definition.

For historical reasons we also follow the MPSX convention of presenting negative quantities by using postfix sign notation.

Sometimes an "A" is printed in front of the line corresponding to a row or column variable (we see an instance of this for the variable ENERGY in our second problem). The meaning of this "A" is that there is a possibility of *alternative* optimal solutions, in which the indicated variable (and other variables) might take different values.

An infeasible solution will not prevent LINPROG from printing the ROWS and COLUMNS output sections. As mentioned previously, the status of each infeasible row is then printed as **. In addition LINPROG prints a table of all the infeasible rows. But before drawing conclusions about the origin of the infeasibility, the user should observe that the printout is only a snapshot of the variables at the time LINPROG decided the infeasibility.

After the ROWS and COLUMNS Section output LINPROG concludes the problem by printing the detected maximum violation of constraints and the CPU-time consumption.

As pointed out earlier there may be more than one problem in a LINPROG job stream (these test examples have only one). After the output for all the individual problems, LINPROG concludes by printing an OUTPUT SUMMARY with a brief record of key data for each problem. A summary line contains 10 items with the following contents:

CASE: Name of the data set.

ROWS: Number of rows in the LP, including the objective row.

COLS: Number of structural columns in the LP.

ELEM: Number of nonzero elements in the constraint matrix. The objective row is included, but slack elements are excluded.

PH.0: Number of eliminated zero slacks in Phase 0 (Section 8.2).

PH.1: Number of iterations in Phase 1 (Section 8.3).

ITNS: Total iteration count. This is the sum of PH.0, PH.1 and (if feasibility was achieved) the number of iterations in Phase 2.

VIOL: Maximum detected constraint violation. For an infeasible problem asterisks are printed.

OPTIMUM: The optimal value. For an infeasible problem asterisks are printed.

CPU-SEC: The CPU time for the problem in seconds.

If an error condition occurs during the LP computation, LINPROG will print a message. Numerous error conditions are possible, but in any case the error message should be self-explanatory.

2.4 Program files

LINPROG communicates with the user via a number of *files*. As a FORTRAN-based system, the code associates an internal unit number with each of these files. The connection between units and files is given in Table 2. We have already discussed the Control File, Matrix File, and Result File. The three other files will be described presently. Except for the Result File, which is often a printer file, all files are supposed to reside on the disk storage of the computer, and even the Result File might be directed to disk.

Unit No.	File	Usage	Mode
2	Matrix File	Input	ASCII
6	Result File	Output	ASCII
15	Restart File	Input	Binary
16	Dump File	Output	Binary
19	Control File	Input	ASCII
20	Communication File	Output	Binary

Table 2. FORTRAN unit numbers for LINPROG files

The files 2, 6, and 19 are mandatory, while 15, 16, and 20 are optional.

2.5 Dump/restart facility

LINPROG has a dump/restart facility, which involves the optional files 15 and 16. If the keyword DUMP is specified in the Control File, LINPROG will conclude the execution by a dump on unit 16. This dump does not contain the solution itself, but the so-called *basic index set*, from which the solution can be retrieved by a comparatively cheap *inversion* process. Not only is a dump made at the end of a normal successful execution, but dumps are also made at intermediate points of the computations (at times of re-inversion, see Section 6), such that each dump overwrites the previous one. This facility may prevent a long LINPROG run from being wasted due to unforeseen failure such as the discontinuation of a job, if some resource limit is exceeded. The idea is that in a subsequent run you can specify RESTART in the Control File and use the previous dump file as a restart file. This facility can save much computer time. Moreover it can be used to divide very time-consuming LP runs into manageable portions. It can also be used in "perturbation analysis", where you want to change some coefficients in the LP matrix

or rhs, provided you don't change the number of restraints or variables. In dump/restart files the information is stored in binary (unformatted) form.

It is natural to use DUMP, whenever the keywords MAXCPM or MAXITS (Section 2.1) are in use.

2.6 Binary output and report writers

In large-scale applications of linear programming the input data to the LP solver is normally produced by a special computer program, tailored to the application. Such a program is called a *matrix generator*. Likewise, there will often be a *report-writer* program which reads the LP output and presents the results in the form of edited and formatted tables suited to the particular problem. One way to design a report writer for LINPROG would be to let it read the ASCII Result File, which of course should be directed to the disk for such a purpose. The advantage of this method is that it is fairly general: An ASCII file can be read by a report writer coded in any programming language that suits your purpose. It can be inspected directly and be transferred between different computer systems.

Alternatively, you may let LINPROG produce a binary (i.e. unformatted) *communication file* containing the solution. To do this you include the keyword BINOUT in the Control File (cf. Section 2.1). The binary output file will be connected to unit 20 in LINPROG (cf. Table 2). The merits of the communication file is that it holds the solution in full machine precision; there is no rounding error due to decimal formatting. It needs less disk space than the ASCII-formatted Result File, and both writing and reading of unformatted records are faster than for formatted records.

The LINPROG communication file consists of multiple solution sets, if more than one LP was solved using the BINOUT keyword. A solution set contains an identification section, a row section, and a column section. The precise contents of each are stated in the following record tables.

The identification section has 3 records:

Record # 1		
Item #	Type	Contents
1	CHARACTER*6	Date of run in the form YYMMDD
2	CHARACTER*4	LINPROG version in the form YYMM
3	CHARACTER*8	'MINIMIZE' or 'MAXIMIZE'

Record # 2		
Item #	Type	Contents
1	CHARACTER*8	Name of the data set
2	CHARACTER*8	Name of the objective row
3	CHARACTER*8	Name of the right-hand side
4	CHARACTER*8	Name of RANGES
5	CHARACTER*8	Name of BOUNDS

Record # 3		
Item #	Type	Contents
1	CHARACTER*1	'O' for optimal, 'I' for infeasible.
2	INTEGER	Number of rows, L.
3	INTEGER	Number of columns, NS.
4	INTEGER	Iteration counter
5	DOUBLE PRECISION	Function value

The row section has L records, one for each row:

Record # I (I = 1 , . . . , L)		
Item #	Type	Contents
1	CHARACTER*1	'A' if alternative solution, otherwise blank
2	CHARACTER*8	Name of the row
3	CHARACTER*2	Status of the row
4	DOUBLE PRECISION	Activity of the row
5	DOUBLE PRECISION	Slack activity of the row
6	DOUBLE PRECISION	Dual activity

The column section has NS records, one for each column:

Record # J (J = 1 , . . . , NS)		
Item #	Type	Contents
1	CHARACTER*1	'A' if alternative solution, otherwise blank
2	CHARACTER*8	Name of the column
3	CHARACTER*2	Status of the column
4	DOUBLE PRECISION	Activity of the column
5	DOUBLE PRECISION	Input cost
6	DOUBLE PRECISION	Reduced cost

The items in the row and column sections correspond to the items explained for the printed output in Section 2.3. However, we do not give the number of the variable, nor its bounds, in the binary output. Under the heading "Type" we specify the items using phrases from the FORTRAN language. CHARACTER*n means a text of length n bytes. The numerical types are INTEGER and DOUBLE PRECISION; for many computers these will use 4 and 8 bytes of storage, respectively, by default.

Appendix A1 gives a printout of a paragon form of a report-writer program. This program, REPORT, reads the communication file and prints first a summary for the LINPROG run and then the two sections of the solution. It also stores some of the variables in arrays, anticipating some subsequent post-processing, depending on the actual application.

In addition to the FORTRAN version of REPORT, Appendix A1 contains a listing of an equivalent PASCAL REPORT program running under VAX/VMS. (We have also written a MODULA-2 REPORT program. This is not listed in the present document, but can be requested from the authors).

As a test of the REPORT program, we repeated our second LINPROG test run (the diet problem with RANGES and BOUNDS), this time with a BINOUT keyword in the Control File. Taking the binary LINPROG output as input, REPORT produced the following output (in all three programming languages):

```
PROBLEM DATE                900312
LINPROG VERSION             9003
TARGET                      MINIMIZE

NAME OF DATA SET            DIETMODI
NAME OF OBJECTIVE ROW       COST
NAME OF RIGHT-HAND SIDE     DEMANDS
NAME OF RANGES              RANGE1
NAME OF BOUNDS              LIMIT

PROBLEM STATUS              OPTIMAL
NUMBER OF ROWS                 6
NUMBER OF COLUMNS              3
NUMBER OF ITERATIONS           2
OBJECTIVE VALUE             2.0500000000000E+02

TABULATION OF THE FILED ROW SECTION
```

```
 NAME      NUMBER STATUS   ACTIVITY        SLACK       DUAL ACTIVITY   MARK
COST          1     BS    2.050000E+02 -2.050000E+02   1.000000E+00
PROTEIN       2     LL    6.500000E+01  0.000000E+00  -5.000000E+00
ENERGY        3     LL    9.000000E+01  0.000000E+00   0.000000E+00    A
CALCIUM       4     BS    5.113125E+02 -3.113125E+02   0.000000E+00
IRON          5     BS    1.648750E+01 -6.487500E+00   0.000000E+00
VITAMINA      6     BS    4.122250E+04 -3.622250E+04   0.000000E+00

TABULATION OF THE FILED COLUMN SECTION
 NAME      NUMBER STATUS   ACTIVITY      INPUT COST    REDUCED COST   MARK
POULTRY       7     UL    2.000000E+02  4.000000E-01  -6.000000E-01
SPINACH       8     BS    5.625000E+02  1.500000E-01   0.000000E+00
POTATOES      9     BS    4.062500E+02  1.000000E-01   0.000000E+00
```

2.7 LINPROG running on a VAX computer

At Risø a Digital VAX-8700 computer was used for most of the development work with LINPROG; this computer is also used for the production runs. In the following we give some examples of "job programs" for controlling the execution of LINPROG on VAX. They use the so-called Digital Control Language (DCL) and are themselves files (normally with extension COM). It is often practical to embed the LINPROG Control File (unit 19) in the job program. The first example is very simple; it corresponds to our first sample case:

```
$DEFINE/USER/NOLOG FOR019 SYS$INPUT
$DEFINE/USER/NOLOG FOR002 'P1'
$DEFINE/USER SYS$OUTPUT RESULT.LIS
$RUN RMS$DISK:[RCL.LP]LINPROG
   SOLUTION
   EXEC
   PEND
$EXIT
```

This DCL command file is supposed to have the name LINPROG.COM. To process it in interactive mode you just type @LINPROG <filename>, where <filename> is the name of the Matrix File you intend to use. Alternatively, you may submit the job for batch processing by the SUBMIT command. Risø users should notice the command line beginning with $RUN. It contains the proper location of the executable LINPROG code file.

Next we give a somewhat more sophisticated example, where a restart file is read and a dump file is produced and converted into a restart file for a later run. Also, a communication file is produced, and afterwards this is read by the report-writer program REPORT discussed above:

```
$!            EXECUTE LINPROG PROGRAM
$!            -----------------------
$!
$!CHANNELS: * FOR002 - MATRIX FILE (MPS FORMAT)
$!          * FOR006 - OUTPUT LISTING
$!          * FOR015 - RESTART FILE
$!          * FOR016 - DUMP FILE
$!          * FOR019 - CONTROL FILE
$!          * FOR020 - OUTPUT COMMUNICATION FILE
$!
$!PARAMETERS: P1     - P1.DAT MATRIX FILE (MPS FORMAT)
$!
$DEFINE/USER FOR019 SYS$INPUT
$DEFINE/USER FOR002 'P1'
$DEFINE/USER FOR006 RESULT.LIS
$DEFINE/USER FOR015 LILOAD.DAT
$DEFINE/USER FOR016 LIDUMP.DAT
$DEFINE/USER FOR020 LIBIND.DAT
$RUN RMS$DISK:[RCL.LP]LINPROG
   RESTART
   DUMP
   BINOUT
   EXEC
   PEND
$!
```

```
$!PREPARE LINPROG DUMP FILE TO A FUTURE RESTART
$RENAME LIDUMP.DAT LILOAD.DAT
$!
$!            EXECUTE REPORT WRITER PROGRAM
$!            -----------------------------
$!
$!CHANNELS:  FOR001 - COMMUNICATION FILE PRODUCED BY LINPROG
$!           FOR002 - REPORT-WRITER OUTPUT FILE
$!
$DEFINE/USER FOR001 LIBINO.DAT
$!
$RUN REPORT
$EXIT
```

There is an important restriction you will face as a VAX LINPROG user: the "pagefile quota" for your username must be raised to 25000, and this must be done by your VAX system manager whom you are advised to consult.

Moreover, if you plan to use VAX LINPROG for solving large problems with thousands of constraints, it may be a good idea to get your maximum working set "WSextent" raised from its default value (normally 1024 pages) to a higher value, say 2000 pages, if this is possible. This will give a considerable reduction in the number of "pagefaults" and thereby also of the CPU-time. Again, you should consult your system manager.

VAX FORTRAN provides two kinds of Double Precision: D-floating and G-floating. G-floating has slightly less precision but a much larger number range than D-floating (and complies with the IEEE floating-point standard). It is important that a report writer be compiled with the same Double Precision type as was used for LINPROG. D-floating is the default type, and we have stuck to this choice when compiling LINPROG at Risø.

2.8 Installation of LINPROG

The installation of LINPROG depends on the type of the computer in question. For a VAX machine it should suffice to install the executable code ("EXE"-file). There are a number of array bounds in LINPROG, which are set by the FORTRAN 77 statement PARAMETER (for example the maximum number of LP rows and columns). It might be necessary to increase these limits if very large problems are going to be solved. On the other hand, it might also sometimes be necessary to lower these limits due to computer system limitations. In both cases, a modification of the source program, followed by a FORTRAN compilation and a LINK operation, would be necessary.

For installing LINPROG on computers without virtual memory it is necessary to lower the parametric bounds drastically to make the code fit into memory. On such computers LINPROG would be able to solve only small problems.

3 The Simplex Method: An Overview

We shall now give a short review of the simplex method in the form we use it in LINPROG. A more detailed discussion of the various components of the algorithm is postponed to later sections. Let us also mention that there are many good textbooks dealing with simplex. First there is Dantzig's classical book [5], and among the newer books we could refer to Murtagh [6] and Nazareth [1].

3.1 Standard form of the LP

We begin with a reformulation of our LP using vector-matrix notation:

$$\text{Minimize} \quad z = \mathbf{c}^T\mathbf{x} \tag{4}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \tag{5}$$

$$\text{and} \quad \mathbf{x} \geq \mathbf{0}. \tag{6}$$

It is clear that our original LP formulation (1) can always be brought to this standard form. We could first convert all $\geq$-rows in (1) to $\leq$-rows by multiplication by -1. Next the $\leq$-rows could be transformed to equalities by introducing nonnegative *slack variables*; obviously an inequality of the form $\sum a_j x_j \leq b,\ x_j \geq 0$ is equivalent to an equality $s + \sum a_j x_j = b,\ s \geq 0, x_j \geq 0$, where s is a slack variable. The slack variables contribute to $\mathbf{A}$ in (5) with a subset of the columns of the unit matrix $\mathbf{I}$. Rows with dummy restrictions R_i in (1) are simply deleted from the LP. Assuming these operations to be already done, our LP has now n variables in total, and m restrictions. Thus the *constraint matrix* $\mathbf{A}$ becomes an $m \times n$ matrix. This matrix is supposed to have full rank, $\rho(\mathbf{A}) = m$, that is, the restrictions are independent, so we have $n \geq m$. Moreover, $\mathbf{c}$ and $\mathbf{x}$ are n-vectors with an inner product equal to the *objective function* z in (4), and $\mathbf{b}$ is an m-vector.

LINPROG is able to deal with more general problem formulations than envisaged in (1) or (4) - (6), but we lose little in referring to the standard form to explain our use of simplex. For example, maximizing $\mathbf{c}^T\mathbf{x}$ is equivalent to minimizing $-\mathbf{c}^T\mathbf{x}$. LINPROG can also treat inhomogeneous objective functions $c_0 + \mathbf{c}^T\mathbf{x}$. A less trivial extension is the capability of the program to handle bounded variables and range constraints. We shall describe later (Section 8.1) how these facilities are implemented, using a modification of the simplex method. Another point to mention is the ability of LINPROG to cope with a rank-deficient constraint matrix: The program detects redundant or conflicting restrictions, and by dispensing of the superfluous equations it produces a reduced matrix with full row rank.

3.2 The basis exchange mechanism

Let now a subset of m independent columns of $\mathbf{A}$ be given. Arranged in any order they form a square nonsingular matrix $\mathbf{B}$, which we call a *basis matrix*. The corresponding variables x_j are called the *basic* variables; the remaining variables are said to be *nonbasic*. The fundamental idea in the simplex method is to operate with solutions $\mathbf{x} = \{x_j\}$ in which the $n - m$ nonbasic components are 0, while the m basic variables may be nonzero (some basic variables may be zero, too, and then we have a *degenerate* solution). If a basic solution $\mathbf{x} = \{x_j\}$ satisfies (5) and (6), we call it *basic feasible*. If (5) is satisfied but not (6) we have an *infeasible* "solution". The partition of the variables in basic and nonbasic described here is dynamic in the sense that the simplex process in each iteration step exchanges the state of two variables (x_j, x_k) such that x_j "enters the basis" and x_k "leaves the basis", that is, it becomes zero. In each step we move from one basic feasible solution to another with a better (at least not worse) objective function. In practice we reach the optimum in a finite number of steps; see however the comments given in Section 8.4 on the possibility of "cycling". By the end of the exchange step, $\mathbf{B}$ is transformed to an *adjacent* basis matrix $\overline{\mathbf{B}}$. In order that we can get the simplex method to work, we must provide an initial basic feasible solution. This is achieved by a special technique to be discussed in Section 8.3.

Let us take a closer look of what happens in an exchange step. If we renumber the columns in $\mathbf{A}$ such that the current basic columns precede the nonbasic columns, we may write $\mathbf{A}$ as a partitioned matrix

$$\mathbf{A} = (\ \mathbf{B} \mid \mathbf{N}\ ), \tag{7}$$

with a corresponding partitioning and ordering of $\mathbf{x}$ and $\mathbf{c}$,

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} \mathbf{c}_B \\ \mathbf{c}_N \end{pmatrix}, \tag{8}$$

while (5) becomes

$$\mathbf{B}\mathbf{x}_B + \mathbf{N}\mathbf{x}_N = \mathbf{b}. \tag{9}$$

The components of the two parts of the solution vector $\mathbf{x}$ are

$$(\mathbf{x}_B)_i = x_i \quad (i = 1, \ldots, m), \qquad (\mathbf{x}_N)_j = x_{m+j} \quad (j = 1, \ldots, n-m); \tag{10}$$

Similarly for the cost vector $\mathbf{c}$:

$$(\mathbf{c}_B)_i = c_i \quad (i = 1, \ldots, m), \qquad (\mathbf{c}_N)_j = c_{m+j} \quad (j = 1, \ldots, n-m); \tag{11}$$

The ordering of columns within $\mathbf{B}$ (and within $\mathbf{N}$) is immaterial here (a discussion of how to number the rows and columns in $\mathbf{B}$ will be given later). Following common practice we introduce the *transformed right-hand* vector

$$\boldsymbol{\beta} = \mathbf{B}^{-1}\mathbf{b} \tag{12}$$

and the *pricing vector*

$$\boldsymbol{\pi}^T = \mathbf{c}_B^T\mathbf{B}^{-1}, \tag{13}$$

whose components are called *simplex multipliers*. Then we obtain the following problem formulation in terms of the nonbasic variables:

$$\text{Minimize} \quad z = \mathbf{c}_B^T\boldsymbol{\beta} + (\mathbf{c}_N^T - \boldsymbol{\pi}^T\mathbf{N})\mathbf{x}_N \tag{14}$$

$$\text{subject to} \qquad \mathbf{x}_B = \boldsymbol{\beta} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_N \tag{15}$$

$$\text{and} \qquad \mathbf{x}_B, \mathbf{x}_N \geq \mathbf{0}. \tag{16}$$

The current basic solution is obtained by letting $\mathbf{x}_N = \mathbf{0}$, thus

$$\mathbf{x}_B = \boldsymbol{\beta}. \tag{17}$$

But (14) and (15) tell more, since the $(n-m)$-vector $\mathbf{d} = \{d_j\}$ defined by

$$\mathbf{d}^T = \mathbf{c}_N^T - \boldsymbol{\pi}^T\mathbf{N} \tag{18}$$

governs the increase of the objective function when the nonbasic variables rise from their zero bounds. The d_j are called *reduced costs*. Incidentally, $\mathbf{d}$ is the gradient vector of the objective function in the current space of nonbasic variables.

Assume now that $d_j < 0$ for some j, say $j = q$. We may then introduce the qth nonbasic variable $(\mathbf{x}_N)_q = x_{m+q}$ into the basis. This means that we increase this variable by a certain amount θ, while all the other components of $\mathbf{x}_N$ are kept at zero. This will cause z to decrease with the linear slope $|d_q|$. But how large a step θ can we take without destroying the feasibility? This question is answered by considering (15), which for the present case specializes to

$$\mathbf{x}_B = \boldsymbol{\beta} - x_{m+q}\boldsymbol{\alpha}_q, \tag{19}$$

where we use $\boldsymbol{\alpha}_j$ to denote the jth transformed nonbasic column vector of $\mathbf{A}$,

$$\boldsymbol{\alpha}_j = \mathbf{B}^{-1}\mathbf{a}_{m+j}, \tag{20}$$

still obeying the column numbering of $\mathbf{A}$ laid down by (7). If $\boldsymbol{\beta} = \{\beta_i\}$ and $\boldsymbol{\alpha}_q = \{\alpha_{iq}\}$, we find θ by the classical *ratio test*

$$\theta = \min\{\beta_i/\alpha_{iq} : i = 1, \ldots, m \text{ and } \alpha_{iq} > 0\} \tag{21}$$

If no element $\alpha_{iq} > 0$ exists, we have detected an ***unbounded solution***, $\theta = \infty$. Otherwise let the minimum in (21) occur for $i = p$. When the corresponding step $x_{m+q} = \theta$ is taken, the pth basic variable reaches zero first and is selected to leave the basis. The number p is called the *pivot index*, while α_{pq} is called the *pivot element*. In the degenerate case the step θ will be 0, and we get no improvement in the objective function.

It is instructive to look at the simplex method from a geometrical viewpoint. The set of all feasible vectors x , i.e. those satisfying the constraints (5) and (6), forms a polytope ("simplex") in n-dimensional space $\mathcal{R}^n$. This polytope may be null, but otherwise one of its vertices is a minimizing point for the objective function z in (4). In the simplex method we carry out a systematic search of the vertices. At each point the $n - m$ nonbasic variables take the value zero, while the m basic variables are determined by (17) and (12). The search proceeds from vertex to adjacent vertex along an edge where a single nonbasic variable for the first vertex increases from zero. The edges are chosen so that the objective function decreases. Since the polytope has only a finite number of vertices, the simplex algorithm is finite unless cycling (Section 8.4) occurs.

3.3 The simplex algorithm

We are now ready to present the main steps of the simplex algorithm. The steps are as follows:

- INITIALIZATION Step — Begin with a basic feasible solution (Section 8.3).
- BTRAN Step — Compute the simplex multipliers by (13).
- PRICE Step — Compute the reduced costs $\{d_j\}$ for the nonbasic variables by (18).
- CHUZC Step — Choose the entering nonbasic variable ($j = q$) as one with a negative reduced cost d_j. (A common rule is to choose one with maximum $|d_j|$; see also Section 8.4). If there is no negative reduced cost, the current solution is optimal.
- FTRAN Step — Update the entering column by (20).
- CHUZR Step — Use the ratio test (21) to find a step θ. This test results either in a variable to leave tbe basis (pivot index $i = p$), or in an unbounded solution with $\theta = \infty$.
- PIVOT Step — In the basis matrix $\mathbf{B}$, replace the column associated with the leaving variable with that corresponding to the entering one, to obtain an adjacent basis matrix $\overline{\mathbf{B}}$. Update the β-vector defined in (12). Return to the BTRAN step.

The above acronyms are well-established abbreviations for key operations in linear programming: BTRAN means "backward transformation" (postmultiplication by $\mathbf{B}^{-1}$), CHUZC means "choose a column", FTRAN means "forward transformation" (premultiplication by $\mathbf{B}^{-1}$), and CHUZR means "choose a row".

If we were to solve only small problems, it would be adequate to maintain the so-called *current tableau* $\mathbf{B}^{-1}\mathbf{A}$. That method could be called "direct simplex". In *revised simplex*, which is used here as well in the majority of simplex codes, we maintain only the basis-inverse $\mathbf{B}^{-1}$ itself. The advantages of this procedure are connected to the use of a *product representation* of $\mathbf{B}^{-1}$, which we discuss in Section 5. The product form is expanded each time a simplex step is executed, and eventually we need to shorten the representation; this is done by the so-called *re-inversion* process described in Section 6.

In our list of simplex operations the PIVOT Step deals with the updating mechanism when passing from one basis matrix to an adjacent one. This step, which involves a good deal of computation, exploits the Forrest-Tomlin method described at length in Section 7.

4 Useful Matrix Relations

At this place we recapitulate some well-known results from linear algebra, since they will be needed in the following sections. We shall concentrate on square matrices; all the considered matrices are assumed to be of the order m and nonsingular, hence invertible.

4.1 The Sherman-Morrison identity

We begin with a useful formula for inverting "rank-one" updated matrices:

Proposition 1 (Sherman-Morrisou) *Given vectors* $\mathbf{u}$ *and* $\mathbf{v}$ *in* $\mathcal{R}^m$, *and the nonsingular matrix* $\mathbf{A} \in \mathcal{R}^{m \times m}$. *If* $\mathbf{v}^T\mathbf{A}^{-1}\mathbf{u} \neq -1$, *then*

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}} \tag{22}$$

This result, given for example in Golub and van Loan [7], and in Murtagh [6], is easily proved by a direct check.

4.2 Elementary matrices

An *elementary column matrix* $\mathbf{E}$ is one which differs from the unit matrix $\mathbf{I}$ in just one column. If we denote the pth unit vector by e_p, such a matrix can be written

$$\mathbf{E} = (\mathrm{e}_1, \ldots, \mathrm{e}_{j-1}, \alpha, \mathrm{e}_{j+1}, \ldots, \mathrm{e}_m), \tag{23}$$

or

$$\mathbf{E} = \begin{pmatrix} & & j & \\ 1 & & \alpha_1 & \\ & \ddots & \vdots & \\ & & \alpha_j & \\ & & \vdots & \ddots \\ & & \alpha_m & & 1 \end{pmatrix} \tag{24}$$

If we want to emphasize that we have inserted a column vector $\alpha = (\alpha_1, \ldots, \alpha_m)^T$ at the jth column place, we may write

$$\mathbf{E} = \mathbf{E}_j(\alpha). \tag{25}$$

Elementary column matrices are sometimes called Frobenius matrices. We may write $\mathbf{E}$ in the form of a rank-one update of $\mathbf{I}$:

$$\mathbf{E} = \mathbf{I} + (\alpha - \mathrm{e}_j)\mathrm{e}_j^T \tag{26}$$

Using the Sherman-Morrison identity (22), or by direct means, we find that the inverse is another elementary column matrix,

$$\mathbf{E}^{-1} = \begin{pmatrix} & & j & \\ 1 & & -\alpha_1/\alpha_j & \\ & \ddots & \vdots & \\ & & 1/\alpha_j & \\ & & \vdots & \ddots \\ & & -\alpha_m/\alpha_j & & 1 \end{pmatrix}. \tag{27}$$

Hence

$$(\mathbf{E}_j(\alpha))^{-1} = \mathbf{E}_j(\eta), \tag{28}$$

where

$$\boldsymbol{\eta} \equiv \mathbf{g}_j(\boldsymbol{\alpha}) = \{\eta_{ij}\}, \qquad \eta_{ij} = -\alpha_i/\alpha_j, i \neq j, \quad \eta_{jj} = 1/\alpha_j. \tag{29}$$

Also in this context we call the jj-entry α_j of $\mathbf{E}$ in (24) the *pivot element*, and column j is called the *pivot column*.

In complete analogy with elementary column matrices we speak of *elementary row matrices*. Such a matrix differs from $\mathbf{I}$ in just one row. In this work we shall use only elementary row matrices with a unit pivot element:

$$\mathbf{F} = \mathbf{F}_i(\boldsymbol{\gamma}) = \begin{matrix} \\ \\ i \\ \\ \\ \end{matrix} \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ \gamma_1 & \cdots & 1 & \cdots & \gamma_m & \\ & & & \ddots & & \\ & & & & 1 & \end{pmatrix}, \tag{30}$$

where $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_{i-1}, 1, \gamma_{i+1}, \ldots, \gamma_m)$. The inverse of (30) is another elementary row matrix

$$\mathbf{F}^{-1} = \begin{matrix} \\ \\ i \\ \\ \\ \end{matrix} \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ -\gamma_1 & \cdots & 1 & \cdots & -\gamma_m & \\ & & & \ddots & & \\ & & & & 1 & \end{pmatrix}. \tag{31}$$

4.3 Column replacement updating

Suppose we want to replace the pth column of a nonsingular matrix $\mathbf{B}$ with an arbitrary column vector (not necessarily a column in $\mathbf{B}$), say $\mathbf{a}_k = \{a_{ik}\}$. The updated matrix can then be written

$$\overline{\mathbf{B}} = \mathbf{B}(\mathbf{I} - \mathbf{e}_p\mathbf{e}_p^T) + \mathbf{a}_k\mathbf{e}_p^T, \tag{32}$$

or

$$\overline{\mathbf{B}} = \mathbf{B}(\mathbf{I} + (\boldsymbol{\alpha}_k - \mathbf{e}_p)\mathbf{e}_p^T) = \mathbf{B}\mathbf{E}_p(\boldsymbol{\alpha}_k), \tag{33}$$

where we have introduced tbe "updated" column vector

$$\boldsymbol{\alpha}_k = \{\alpha_{ik}\} = \mathbf{B}^{-1}\mathbf{a}_k. \tag{34}$$

The inverse can now be obtained from (27) or (28) with $j = p$ and $\boldsymbol{\alpha} = \boldsymbol{\alpha}_k$:

$$\overline{\mathbf{B}}^{-1} = \begin{matrix} \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ p \\ \\ \\ \end{matrix} \begin{pmatrix} 1 & & -\alpha_{1k}/\alpha_{pk} & & \\ & \ddots & \vdots & & \\ & & 1/\alpha_{pk} & & \\ & & \vdots & \ddots & \\ & & -\alpha_{mk}/\alpha_{pk} & & 1 \end{pmatrix} \mathbf{B}^{-1}. \tag{35}$$

This result will be used to update the basis-inverse in the simplex procedure.

4.4 Permutation matrices

Sometimes we need a precise way to express what happens when rows and/or columns in a matrix are rearranged. Let an arbitrary permutation

$$\pi = \begin{pmatrix} 1 & 2 & \ldots & m \\ p_1 & p_2 & \ldots & p_m \end{pmatrix} \tag{36}$$

be given. The result of applying π to the rows of a matrix $\mathbf{B}$ is another matrix

$$\mathbf{B}_r = \mathbf{PB}, \tag{37}$$

where the premultiplying factor is given by

$$\mathbf{P} = (\mathbf{e}_{p_1}, \ldots, \mathbf{e}_{p_m}) \tag{38}$$

and is called a *permutation matrix*. If we instead apply π to the columns of $\mathbf{B}$ we would get

$$\mathbf{B}_c = \mathbf{BP}^T, \tag{39}$$

still with $\mathbf{P}$ given by (38). We could also permute both the rows and the columns, using two permutation matrices $\mathbf{P}$ and $\mathbf{Q}$:

$$\mathbf{B}_{rc} = \mathbf{PBQ}^T. \tag{40}$$

In particular, the rows and columns can be symmetrically permuted:

$$\mathbf{B}_{sym} = \mathbf{PBP}^T. \tag{41}$$

Here the diagonal row of $\mathbf{B}$ is preserved in $\mathbf{B}_{sym}$; only the order of its elements is altered.

Any permutation matrix is orthogonal,

$$\mathbf{PP}^T = \mathbf{P}^T\mathbf{P} = \mathbf{I}. \tag{42}$$

4.5 LU-factorization

Any nonsingular matrix can be written as the product of a lower triangular factor $\mathbf{L}$ and an upper triangular factor $\mathbf{U}$, at least when we admit row (and/or column) interchanges. This fact is contained in

Proposition 2 (LU-theorem) *Given a matrix $\mathbf{B} \in \mathcal{R}^{m\times m}$, in which all the leading principal minors are nonsingular, then there exists a unique lower triangular matrix $\mathbf{L} = \{\ell_{ij}\}$ with $\ell_{ii} = 1$ and a unique upper triangular matrix $\mathbf{U} = \{u_{ij}\}$, so that*

$$\mathbf{B} = \mathbf{LU}, \tag{43}$$

or, written out in full,

$$\begin{pmatrix} b_{11} & \ldots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{m1} & \ldots & b_{mm} \end{pmatrix} = \begin{pmatrix} 1 & & \\ \vdots & \ddots & \\ \ell_{m1} & \ldots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & \ldots & u_{1m} \\ & \ddots & \vdots \\ & & u_{mm} \end{pmatrix} \tag{44}$$

A proof of this theorem is given e.g. in Forsythe and Moler [8]. The necessary rearrangement of an arbitrary nonsingular matrix to qualify for the **LU** theorem can always be accomplished by shuffling its rows (i.e. premultiplying it by a permutation matrix), or by shuffling its columns, or both.

4.6 Decomposition of triangular matrices into elementary matrices

For triangular matrices we have a particularly simple factorization into elemeutary column matrices:

$$\mathbf{L} = \begin{pmatrix} 1 & & & & \\ \vdots & \ddots & & & \\ \ell_{k1} & \ldots & 1 & & \\ \vdots & & \vdots & \ddots & \\ \ell_{m1} & \ldots & \ell_{mk} & \ldots & 1 \end{pmatrix} = \prod_{k=1}^{m} \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & \vdots & \ddots & \\ & & \ell_{mk} & & 1 \end{pmatrix} \tag{45}$$

and

$$\mathbf{U} = \begin{pmatrix} u_{11} & \cdots & u_{1k} & \cdots & u_{1m} \\ & \ddots & \vdots & & \vdots \\ & & u_{kk} & \cdots & u_{km} \\ & & & \ddots & \vdots \\ & & & & u_{mm} \end{pmatrix} = \prod_{k=m}^{1} {}^{k} \begin{pmatrix} 1 & & u_{1k} & & \\ & \ddots & \vdots & & \\ & & u_{kk} & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}. \tag{46}$$

There is a similar pair of factorizations into elementary row matrices:

$$\mathbf{L} = \begin{pmatrix} 1 & & & & \\ \vdots & \ddots & & & \\ \ell_{k1} & \cdots & 1 & & \\ \vdots & & \vdots & \ddots & \\ \ell_{m1} & \cdots & \ell_{mk} & \cdots & 1 \end{pmatrix} = \prod_{k=1}^{m} {}^{k} \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ \ell_{k1} & \cdots & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} \tag{47}$$

and

$$\mathbf{U} = \begin{pmatrix} u_{11} & \cdots & u_{1k} & \cdots & u_{1m} \\ & \ddots & \vdots & & \vdots \\ & & u_{kk} & \cdots & u_{km} \\ & & & \ddots & \vdots \\ & & & & u_{mm} \end{pmatrix} = \prod_{k=m}^{1} {}^{k} \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & u_{kk} & \cdots & u_{km} \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}. \tag{48}$$

The factor in (45) corresponding to $k = m$ is the unit matrix, and so is the factor in (47) for $k = 1$; these unit factors are displayed for reasons of symmetry.

For the LINPROG implementation of simplex we need only the forms (46) and (47). Correspouding expressions for the inverse matrices $\mathbf{U}^{-1}$ and $\mathbf{L}^{-1}$ are easily obtained by using (27) and (31). In particular this shows that the inverse of a lower triangular matrix is again lower triangular, and similarly for an upper triangular matrix.

5 Elementary Product Forms

It is an essential feature of large-scale linear programming that the basis matrix $\mathbf{B}$, or rather its inverse $\mathbf{B}^{-1}$, is expressed as a product of elementary matrices. Such a product form of $\mathbf{B}^{-1}$ is used and maintained in the simplex procedure; it is occasionally reconstructed in the process of re-inversion (Section 6). The advantage of the product representation is that we need store only those vectors that form the elementary matrices. These are typically sparse and may be held as packed arrays.

5.1 The standard product form

Let an arbitrary nonsingular matrix $\mathbf{B} \in \mathcal{R}^{m \times m}$ be given. It is instructive to describe the so-called *standard product form* of $\mathbf{B}$, in which each factor is an elementary column matrix, though this form is not directly applied in LINPROG.

To obtain the representation we consider $\mathbf{B} = \mathbf{B}_t$ the result of a t-step column replacement process beginning with the unit matrix $\mathbf{B}_0 = \mathbf{I}$. At step number k we transform the matrix $\mathbf{B}_{k-1}$ to $\mathbf{B}_k$ by replacing column p_k by some "entering" vector $\mathbf{a}_k$. This vector need not be one of the columns of $\mathbf{B}$ itself, as $\mathbf{a}_k$ may later on again be replaced by another column. In Section

4.3 we saw that such a column replacement corresponds to post-multiplication of the matrix $\mathbf{B}_{k-1}$ by an elementary column matrix:

$$\mathbf{B}_k = \mathbf{B}_{k-1}\mathbf{E}_{p_k}(\boldsymbol{\alpha}_k), \tag{49}$$

where

$$\boldsymbol{\alpha}_k := \mathbf{B}_{k-1}^{-1}\mathbf{a}_k. \tag{50}$$

Assuming that the process terminates after t steps with all the columns of $\mathbf{B}$ formed in right positions, we can write down the standard product form from (49):

$$\mathbf{B} = \prod_{k=1}^{t} \mathbf{E}_{p_k}(\boldsymbol{\alpha}_k). \tag{51}$$

More important for LP than $\mathbf{B}$ itself is $\mathbf{B}^{-1}$. The inverse of (51) reads

$$\mathbf{B}^{-1} = \prod_{k=t}^{1} \mathbf{E}_{p_k}(\boldsymbol{\eta}_k), \tag{52}$$

where

$$\boldsymbol{\eta}_k = \boldsymbol{\eta} = \mathbf{g}_{p_k}(\boldsymbol{\alpha}_k) \tag{53}$$

has the components stated in (29). Below we give a constructive description of the replacement process in algorithmic form using "pseudo-PASCAL":

$\mathbf{B}_0^{-1} = \mathbf{I}$;
FOR $k := 1$ **TO** t **DO**
BEGIN
  (* identify pivot index for this step *)
  $j := p_k$;
  (* replace current column j with some vector $\mathbf{a}_k$*)
  $\boldsymbol{\alpha}_k := \mathbf{B}_{k-1}^{-1}\mathbf{a}_k$;
  $\boldsymbol{\eta}_k := \mathbf{g}_j(\boldsymbol{\alpha}_k)$;
  $\mathbf{B}_k^{-1} := \mathbf{E}_j(\boldsymbol{\eta}_k)\mathbf{B}_{k-1}^{-1}$
END;
$\mathbf{B}^{-1} := \mathbf{B}_t^{-1}$;

It is clear that many different replacement sequences may lead from $\mathbf{I}$ to the same matrix $\mathbf{B}$ or $\mathbf{B}^{-1}$. Consequently the representations (51) and (52) are not unique. Nor is the number t of exchange steps. For example, at the end of a re-inversion process we may let $t = m$. Subsequently the simplex optimization process may add new factors to the product, and we may get $t > m$.

5.2 Product form of the LU type

The standard product form described here has been in widespread use in earlier LP codes. Its merit is its simplicity. Modern implementations of LP, however, use a product form which is still made up of elementary matrices, but is related to a triangular factorization of the basis matrix. This complication in structure is more than offset by savings in storage and calculations.

Immediately after a re-inversion (and at the start of the simplex process) we have an **LU**-factorization (43) – (44) of the basis matrix $\mathbf{B}$. Among the different ways of decomposing $\mathbf{U}$ and $\mathbf{L}$ into products of elementary matrices we select those given by (46) and (47), because this choice gives us a product form compatible with the Forrest-Tomlin updating scheme given in Section 7. Hence we write

$$\mathbf{L} = \mathbf{L}_1\mathbf{L}_2 \ldots \mathbf{L}_k \ldots \mathbf{L}_m, \tag{54}$$

where

$$\mathbf{L}_k = \begin{matrix} \\ \\ k \\ \\ \\ \end{matrix} \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ \ell_{k1} & \dots & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \tag{55}$$

and

$$\mathbf{U} = \mathbf{U}_m \mathbf{U}_{m-1} \dots \mathbf{U}_k \dots \mathbf{U}_1, \tag{56}$$

where

$$\mathbf{U}_k = \begin{matrix} \\ \\ \\ k \\ \\ \\ \end{matrix} \overset{k}{\begin{pmatrix} 1 & & u_{1k} & & \\ & \ddots & \vdots & & \\ & & u_{kk} & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}}. \tag{57}$$

The resulting basis-iuverse becomes

$$\mathbf{B}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1} = \mathbf{U}_1^{-1} \dots \mathbf{U}_k^{-1} \dots \mathbf{U}_m^{-1}\mathbf{L}_m^{-1} \dots \mathbf{L}_k^{-1} \dots \mathbf{L}_1^{-1}, \tag{58}$$

where

$$\mathbf{U}_k^{-1} = \begin{matrix} \\ \\ \\ k \\ \\ \\ \end{matrix} \overset{k}{\begin{pmatrix} 1 & & -u_{1k}/u_{kk} & & \\ & \ddots & \vdots & & \\ & & 1/u_{kk} & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}} \tag{59}$$

and

$$\mathbf{L}_k^{-1} = \begin{matrix} \\ \\ k \\ \\ \\ \end{matrix} \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ -\ell_{k1} & \dots & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}. \tag{60}$$

These expressions illustrate the close relationship between **LU** decomposition and matrix inversion in product form.

When the simplex procedure calls for a column exchange in the basis, we must update the representation (58) accordingly. As we shall see in Section 7 the Forrest-Tomlin procedure maintains $\mathbf{B}^{-1}$ in a more general form than (58): After each simplex step it leaves the basis-inverse as a product

$$\mathbf{B}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1} = \mathbf{U}_1^{-1} \dots \mathbf{U}_i^{-1} \dots \mathbf{U}_m^{-1}\mathbf{L}_t^{-1} \dots \mathbf{L}_s^{-1} \dots \mathbf{L}_1^{-1}, \tag{61}$$

where $\mathbf{U} = \mathbf{U}_m \dots \mathbf{U}_1$ and $\mathbf{L} = \mathbf{L}_1 \dots \mathbf{L}_t$. The factors $\mathbf{U}_i$ (and $\mathbf{U}_i^{-1}$) are elementary column matrices, while $\mathbf{L}_s$ (and $\mathbf{L}_s^{-1}$) are elementary row matrices. But $\mathbf{U}_i$ and $\mathbf{L}_s$ need no longer be triangular, nor need $\mathbf{L}$ and $\mathbf{U}$ in (61). Again, the factorization (61) is not unique, and t, the number of factors $\mathbf{L}_s$, may be greater than m.

In LINPROG we store the factors $\mathbf{U}_i^{-1}$ in (61) as a sequence of column vectors in one packed list. This "eta-U" list is an array in the computer (which we assume has virtual memory), but for traditional reasons we also call it the "U file". This file (together with a list of the pivot

indices) determines the factor $\mathbf{U}^{-1}$. We shall also maintain an "eta-L" list or "L file" containing a sequence of row vectors forming the elementary row matrices $\mathbf{L}_s^{-1}$. It determines the other factor $\mathbf{L}^{-1}$. Taken together, the L and U files form the so-called "eta-file"; we say that this file contains "column-eta" vectors and "row-eta" vectors.

6 Re-inversions

We have seen that $\mathbf{B}^{-1}$, the inverse of the basis matrix $\mathbf{B}$, in LINPROG is represented by two so-called "eta lists" associated with the two factors of the $\mathbf{LU}$ product form described in the previous section. For each simplex iteration the lists are augmented with new elementary vectors (to be discussed in Section 7), and this growth causes the time per iteration as well as the roundoff errors to grow. Eventually it becomes necessary to compress the structure, and we then make a fresh inversion of $\mathbf{B}$, using our knowledge of the present set of basic columns. Different heuristic criteria for evoking an inversion exist. One popular way is to let the CPU clock of the computer trigger the inversion. Also the detection of unexpected infeasibilities, or numerical deterioration of the current solution, could release an inversion. In LINPROG the re-inversions are performed at regular intervals of the simplex iteration counter, cf. the parameter MITRE described in Section 2.1.

In the following we present the various ingredients of the inversion procedure. After this we state the total inversion algorithm in compressed form, and finally we discuss some possible ways of improving the implementation scheme in LINPROG.

6.1 Decomposition arithmetics and eta vectors

The output from the inversion process in LINPROG will be a product representation (58) for $\mathbf{B}^{-1}$, which in turn is based on the triangular $\mathbf{LU}$ factorization (43) – (44) of the basis matrix $\mathbf{B}$; for this reason the name "re-factorization" is sometimes used instead of re-inversion [1]. We assume that the rows and columns of $\mathbf{B}$ are already permuted in a way that makes the representation (44) possible; the choice of such permutations is the topic of the following subsections.

The matrix identity (44) for the $\mathbf{LU}$ decomposition determines the entries ℓ_{ij} and u_{ij} of $\mathbf{L}$ and $\mathbf{U}$. In fact (44) can be written as the single scalar equation

$$\sum_{\nu=1}^{\min(i,j)} \ell_{i\nu} u_{\nu j} = b_{ij}; \quad i = 1, \ldots, m; \; j = 1, \ldots, m, \tag{62}$$

together with

$$\ell_{ii} = 1. \tag{63}$$

There are several possible orders in which the elements can be computed. We have chosen an order that matches the construction of the row vector in $\mathbf{L}_k$ of (55) and the column vector in $\mathbf{U}_k$ of (57); the calculations are made clear from the following piece of pseudo-PASCAL code:

FOR $k := 1$ **TO** m **DO**
  BEGIN
    FOR $j := 1$ **TO** $k - 1$ **DO**

$$\ell_{kj} := (b_{kj} - \sum_{\nu=1}^{j-1} \ell_{k\nu} u_{\nu j})/u_{jj}; \tag{64}$$

    FOR $i := 1$ **TO** k **DO**

$$u_{ik} := b_{ik} - \sum_{\nu=1}^{i-1} \ell_{i\nu} u_{\nu k} \tag{65}$$

END;

We see that the loop (64) containing ℓ_{kj} produces the row vector in (55), and the loop (65) containing u_{ik} produces the column vector in (57). We also observe that the last element computed at stage k is u_{kk}, which we call the pivot element. Our decomposition algorithm may be characterized as a non-standard variant of Crout's method.

The row eta vector actually stored by LINPROG at pivot stage k is the left-diagonal part $(-\ell_{k1}, \ldots, -\ell_{k,k-1})$ of row k in $\mathbf{L}_k^{-1}$, equation (60); as the column eta we store the super-diagonal part $(u_{1k}, \ldots, u_{k-1,k})^T$ of column k of $\mathbf{U}_k$, equation (57), while the pivot element u_{kk} is stored in a separate list. In this way we avoid the explicit division by the pivot, and yet have all the information needed for (59).

6.2 Pre-ordering of rows and columns

In the previous algebraic description of the inversion process we assumed that **B** was already organized in a way that permits us to pivot down the diagonal, as implied by the decomposition (44). In this connection it is important to realize that what is essential for a basis matrix **B** is its *collection* of independent columns, not the internal ordering of these columns within **B**. Thus, when the inversion is due, we are given a set of basic columns of the constraint matrix **A** in (5), and we are free to arrange them in any order. The rows of **B** correspond to rows in **A** and can also be re-shuffled; this means that we mention the constraints in a different order. Hence we may arrange both the columns and rows in any way that will suit us; only we must keep track of the necessary permutations by index arrays pointing to the row and column positions, respectively, of the matrix **A**. The final row and column numbering of **B** is simply defined by the order of the pivot assignments of the same rows and columns. With this "chronological" convention for the numbering, the output from the inversion will be an unpermuted triangular factorization (44). Later, when the simplex procedure exchanges one column with another from **A**, the new basic column simply inherits its number from the old one. We shall try to utilize the freedom we have to pre-order the basis matrix in such a way that the sparsity is preserved as well as possible without sacrificing the numerical stability. It is well-known that although we start with a sparse basis **B**, the **LU** decomposition may cause "fill-in" of new nonzero elements (the opposite process, viz. cancellation, is possible though less likely.) A measure of the total fill-in is the excess of nonzeros in the combined matrix formed by **L** and **U** over the nonzeros in **B** itself. This fill-in depends in a critical way on the pivot selection order. If we can reduce it, we get smaller eta lists after the inversion. This in turn means faster subsequent simplex iterations.

6.3 Pivot selection strategy for sparsity preservation

If it were possible to arrange **B** in a strictly lower-triangular form, the **LU** factorization would be trivial, and we would get no fill-in. This ideal goal is seldom attained in practice. Our selection method resembles that described by Orchard-Hays [9], who builds a partly triangular basis with a square block **T** in the middle (see Figure 1). The submatrix **T** is called the *nucleus* (or "bump"). In particular **T** may be null, or it may be equal to **B** itself. The upper-left part of the figure represents "row singletons", while the lower-right part represents "column singletons". These rows and columns can easily be identified successively. Creation of new nonzeros depends on the sparsity pattern of **B** and is governed by (64) and (65); we see from these equations that the fill-in is limited to within and below the nucleus.

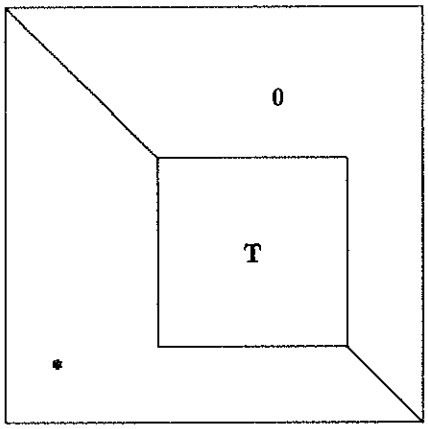

Figure 1. Pre-ordering of **B** *into a block triangular form with a nucleus*

But we can do still better than this. If we pivot literately in the sequeuce implied by Figure 1, we would have to set the column singletons aside, until the nucleus was processed. This would require extra workspace. We avoid it by permuting the rows and columns in **B** such that the rows below the nucleus are moved to the top (in reverse order) with analogous shifts of the columns (cf., Benichou *et al.*, [10]). One can show that this gives the arrangement in Figure 2, and thus leads to a sequence where we first process the column singletons, then the row singletons, and

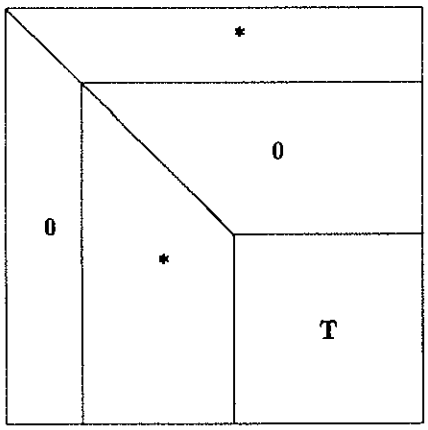

Figure 2. Rearranged pre-ordering with the nucleus at the end

finally the nucleus. An additional gain with Figure 2 is that there is no fill-in outside the nucleus itself.

When we reach the nucleus we must use some pivot selection heuristics. Let us in the first place concentrate on limitation of fill-in, hence preservation of sparsity. A reasonable strategy will be to select first the new pivot row as one with a minimum number of active nonzeros, and then choose the corresponding pivot column as one with a nonzero in the pivot row and with a minimnm count of nonzeros in the column. "Active" means here: "not yet assigned for pivoting". We notice that with this rule for the nucleus there is no need for a separate treatment of the row singletons; they will automatically be selected as the first rows.

When pivots are assigned, the corresponding row and column are de-activated, and appropriate updating of lists take place.

6.4 Modifications to ensure numerical stability

The pivot selection rule described above works exclusively on the sparsity pattern of the basis. As it stands, it does not involve numerical considerations. But it is quite possible that the computed value u_{kk} of the proposed pivot element becomes exactly zero, or at least its absolute value $|u_{kk}|$ becomes very small, and in both cases the pivot must be rejected.

LINPROG uses the following amendment of the selection rule given in Section 6.3: The pivot row which was selected from sparsity considerations is always accepted. When selecting a pivot column, we find two candidates to choose between (they may coincide). The first one is selected by the minimum-count criterion given in Section 6.3; ties are resolved in favour of greater pivot size. The second one has maximum pivot size among all the available columns. Denoting the pivot values for the two candidates p_1 and p_2, respectively, we use the former candidate if $|p_1/p_2|$ exceeds some number ε, otherwise the latter one. If we take $\varepsilon = 1$, we would always select p_2, and this corresponds to the classical (partial) pivoting rule for solving linear equations. On the other hand, a small ε would favour the choice of p_1, i.e. the pivoting for sparsity and limitation of the fill-in. LINPROG uses $\varepsilon = 0.01$ as a standard value. It can be changed to another value by the user with the keyword EPSRIN in the Control File, cf. Section 2.1 and Table 3 in Section 8.5.

The described acceptance test is based on the relative size of the pivot to the maximum size of all available pivots in the row. Hence we must calculate all these pivots. Instead we could have used a cheaper absolute test and accept the pivot that was proposed by the sparsity rule, unless its size drops below an absolute threshold ε_{thr}; in that case one would invoke an "emergency procedure" and select the largest possible pivot size. Our numerical experiments indicate that the relative test often gives better stability than does the absolute test, and that the overall increase in computing time is small. Some inversion schemes use a combination of both tests.

Recall that the computation of a pivot element is done by (65) with $i = k$, while (65) for $i < k$ and (64) give us the new column and row eta vectors, respectively.

6.5 Sparse-matrix implementation details

The "Boolean" part of the inversion requires that we maintain and update a map of the current pattern of nonzeros during our **LU** decomposition. Zeros as a result of cancellation are ignored. In our FORTRAN code we use traditional sparse-matrix techniques, based on a double set of ordered lists: one list, IB, is column-ordered and contains the row numbers of the nonzeros, while the other one, JB, is row-ordered and contains the column numbers of the same elements. In both lists we reserve "elbow room" to accommodate for possible fill-in (initially we provide the same elbow room for each row/column as its count of nonzeros.) Each of the two lists are equipped with three set of pointer arrays. For the column-ordered list they are ICLPT1, ICLPT2, and ICLAST. ICLPT1(J) holds the address in IB that corresponds to the first non-zero in column J. ICLPT2(J) points in the same way to the last nonzero of the column. ICLAST(J) holds the last free elbow-room address for column J. Analogous arrays IRWPT1, IRWPT2, and IRLAST support the row-ordered list.

Should the elbow room for some column or row be used up during the inversion, a simple memory-management procedure is activated: we make a fresh copy of the column or row at the end of the list with a new elbow room equal to the new information length. The previous storage for that column or row is wasted, and no garbage collection is made. This waste may be tolerated, because after the inversion is finished, all the lists are abandoned.

Moreover we must keep track of the still active counts of nonzeros in the rows and in the columns. These counts are used in our pre-ordering method for pivot selection.

Special care must be taken when computing the scalar products entering (64) and (65). The problem arising is that the **L** and **U** elements, which are loaded from the two eta files, correspond to unpermuted rows and columns of the constraint matrix. But we know that both should be permuted to match our common pivot-index numbering, and this makes an internal sortiug

procedure necessary.

6.6 Algorithmic description of the inversion

After this discussion of the different aspects of the inversion method, we shall now put the bricks together and give a compressed algorithmic description (expressed in pseudo-PASCAL). We assume that all the necessary initializations are made. Then we proceed as follows:

```
clsing := TRUE;
(* clsing becomes FALSE when column singletons are exhausted *)
FOR k := 1 TO m DO
(* k is the pivot step counter *)
BEGIN
  200:
  IF clsing THEN
  BEGIN
    Locate a singleton pivotcolumn;
    clsing := (we found a column singleton);
    IF ¬clsing THEN GO TO 200;
    Locate corresponding pivot row;
  END ELSE
  BEGIN (* code for nucleus including row singletons *)
    Locate pivotrow as one with minimum count;
  END (* nucleus *);
  (* selection of pivotrow finished *)
  Build new row eta vector by (64);
  De-activate pivotrow and give it the "chronological" number k;
  IF ¬clsing THEN
  BEGIN (* code for nucleus including row singletons *)
    Scan pivotrow to produce pivotcolnmn candidates 1 and 2;
    (*
    candidate 1 has minimum count (ties broken in favour of pivot size);
    candidate 2 has largest pivot size among all available columns;
    the pivot values are p1 and p2, respectively
    *)
    IF |p1| > ε|p2| THEN pivotcolumn := candidate1
    ELSE pivotcolumn := candidate2
  END (* nucleus *);
  Build new column eta vector by (65);
  De-activate pivotcolumn and give it the "chronological" number k;
  Update basic column indicator;
  Reduce active row and column counts with the de-activated elements;
  Update ordered lists to accommodate fill-in at current pivot step;
  IF no more elbow-room for a column or row THEN
  BEGIN
    (* perform memory management *)
    Copy row or column to end of ordered list;
    Provide fresh elbow room of size equal to copied part
  END
END;
```

6.7 Possible improvements

It is generally agreed that the **LU** decomposition described here is superior to the old "Product Form of the Inverse" (PFI) method, in which $\mathbf{B}^{-1}$ is factored in standard product form using elementary column matrices only, as described in Section 5.1. The inversion time is less, and the resulting eta lists are shorter.

On the other hand, there is still room for improvements of the inversion, not yet exploited in LINPROG. It should be possible to reduce the fill-in by more sophisticated methods of pivot selection. First, one could carry out a complete block triangular rearrangement of **B**. There exist efficient algorithms to do this (see Duff and Reid [11], and Duff [12]). Another way to pursue, proposed by Hellerman and Rarick [13], would be to look at **B** (or each of the triangularization blocks in turn), and by suitable re-shuffling of the rows and columns transform it to a triangular structure superimposed by "spikes". The advantage is that all the fill-in is confined to the spikes, and this tends to reduce the overall inversion work. Both block triangularization and spike techniques are common ingredients of today's commercial LP codes.

Finally, it might be questioned whether our somewhat wasteful and elaborate ordered-list representation of the sparsity pattern, with its overhead of elbow room and memory management, could stand up against an implementation based on linked lists. Very likely the latter concept might be the more economical of the two in modern computer environments.

7 The Forrest-Tomlin Procedure

In this section we turn to the question of updating the factorization of the basis matrix **B**, or its inverse $\mathbf{B}^{-1}$, when successive iteration cycles of the simplex process cause replacements of the columns of **B**. We shall in the following describe the *Forrest-Tomlin* updating method [14, 15, 6, 1], which is suitable for maintaining an **LU**-like product form (61) for $\mathbf{B}^{-1}$.

7.1 General outline

In Section 6 it was pointed out, that when we resume the simplex iterations after a re-inversion, our current basis matrix $\mathbf{B} \in \mathcal{R}^{m \times m}$ will be factored as

$$\mathbf{B} = \mathbf{L}\mathbf{U}, \tag{66}$$

where **L** is lower triangular and **U** upper triangular, provided we adopt the row and column numbering induced by the pivoting sequence of the inversion. We shall do so, adhering to the numbering convention given in Section 6.2.

When a subsequent iteration step of the simplex process transforms **B** to an adjacent basis matrix $\overline{\mathbf{B}}$, the question arises whether it is still possible to write $\overline{\mathbf{B}}$ as a product of factors $\overline{\mathbf{L}}$ and $\overline{\mathbf{U}}$,

$$\overline{\mathbf{B}} = \overline{\mathbf{L}}\,\overline{\mathbf{U}}, \tag{67}$$

such that $\overline{\mathbf{L}}$ can be easily computed by updating **L**, and analogously for $\overline{\mathbf{U}}$. Several schemes exist that accomplish this task. The merit of the Forrest-Tomlin method is that it preserves the triangular factorization during updating, when allowance for permutations is made. Moreover, it creates no fill-in of new nonzeros in the eta lists.

In order to give a general description of the updating mechanism, let us temporarily leave any specific assumptions about **L** and **U** in (66) out of account. For the time being they are just invertible matrices whose product is **B**. Suppose now that we want to carry out an exchange step of the simplex procedure with pivot index p. This means that we must replace column p of **B** by some nonbasic column $\mathbf{a}_k$ of the constraint matrix **A**. We use the ordering (7) for the columns in **A** and assume that the qth nonbasic column was chosen such that $\mathbf{a}_k = \mathbf{a}_{m+q}$. The

resulting adjacent basis matrix $\overline{\mathbf{B}}$ is given by (32) with $k = m + q$, and this we shall factorize in the form (67), or in the equivalent inverse form

$$\overline{\mathbf{B}}^{-1} = \overline{\mathbf{U}}^{-1}\overline{\mathbf{L}}^{-1}. \tag{68}$$

We introduce the "partially updated" incoming vector

$$\mathbf{v} = \mathbf{L}^{-1}\mathbf{a}_{m+q}; \tag{69}$$

if we replace column p of $\mathbf{U}$ by $\mathbf{v}$ and again apply the column replacement formula (32) we obtain the matrix

$$\mathbf{U}' = \mathbf{U}(\mathbf{I} - \mathbf{e}_p\mathbf{e}_p^T) + \mathbf{v}\mathbf{e}_p^T, \tag{70}$$

and evidently we have

$$\overline{\mathbf{B}} = \mathbf{L}\mathbf{U}'. \tag{71}$$

Next we define the vector $\mathbf{r}$ as the unique solution to the equation

$$\mathbf{U}^T\mathbf{r} = \mathbf{e}_p, \tag{72}$$

from which we derive

$$\mathbf{r}^T = \mathbf{e}_p^T\mathbf{U}^{-1}, \tag{73}$$

which states that $\mathbf{r}^T$ equals the pth row of $\mathbf{U}^{-1}$. Now build the elementary row matrix

$$\mathbf{R} = \mathbf{I} - \mathbf{e}_p\mathbf{e}_p^T + \sigma\mathbf{e}_p\mathbf{r}^T = \begin{matrix} \\ \\ p \\ \\ \\ \end{matrix}\begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ \cdots & \cdots & \sigma r^T & \cdots & \cdots \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \tag{74}$$

where the normalization constant σ is chosen such as to render the pivot element of $\mathbf{R}$ equal to one, that is

$$\sigma = 1/r_p. \tag{75}$$

Then define

$$\overline{\mathbf{L}} = \mathbf{L}\mathbf{R}^{-1} \tag{76}$$

and

$$\overline{\mathbf{U}} = \mathbf{R}\mathbf{U}'. \tag{77}$$

These are the updated factors in (67) we are looking for, and (71) shows that their product is indeed $\overline{\mathbf{B}}$.

What can be said about the structure of the updated factors $\overline{\mathbf{L}}$ and $\overline{\mathbf{U}}$ and their inverse? If we first consider $\overline{\mathbf{L}}$, we observe that immediately after an inversion $\mathbf{L}$ is lower triangular and hence can be factored into a product of elementary row matrices, as (47) shows. As $\mathbf{R}^{-1}$ is also an elementary row matrix, the expression (76) shows that the updating process preserves $\overline{\mathbf{L}}$ as a product of elementary row matrices. Of course the same is true for the inverse

$$\overline{\mathbf{L}}^{-1} = \mathbf{R}\mathbf{L}^{-1}. \tag{78}$$

The structure of $\overline{\mathbf{U}}$ can be deduced from (77), (70), (74), and (73). After reduction we find

$$\overline{\mathbf{U}} = (\mathbf{I} - \mathbf{e}_p\mathbf{e}_p^T)\mathbf{U}(\mathbf{I} - \mathbf{e}_p\mathbf{e}_p^T) + \mathbf{v}'\mathbf{e}_p^T, \tag{79}$$

where we have introduced the vector

$$\mathbf{v}' = \mathbf{R}\mathbf{v}; \tag{80}$$

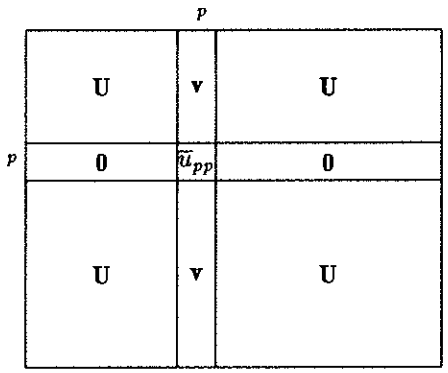

Figure 3. General structure of the matrix $\overline{\mathbf{U}}$

we see that $\mathbf{v}'$ equals $\mathbf{v}$ in all places except the pth where its element is

$$v'_p = \overline{u}_{pp} = \sigma\ \mathbf{r}^T\mathbf{v}. \tag{81}$$

A sketch of $\overline{\mathbf{U}}$ is given in Figure 3. It is seen that the effect of $\mathbf{R}$ is to annihilate all nonpivotal elements of the pivot row of $\mathbf{U}'$.

Let us now assume that $\mathbf{U}$ is upper triangular (as it is after an inversion). Then $\overline{\mathbf{U}}$ takes the form shown in Figure 4. The cyclic permutation $(m, m-1, \ldots, p+1, p)$ has the effect of taking

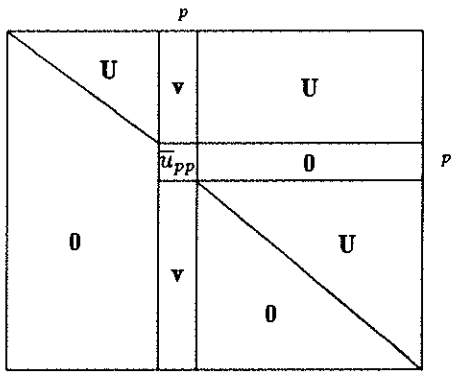

Figure 4. Almost-triangular structure of the matrix $\overline{\mathbf{U}}$

element number p to position m and shifting the elements $p+1, \ldots, m$ one place to the left. If $\mathbf{Q}$ is the corresponding permutation matrix (Section 4.4), then the symmetrically permuted matrix

$$\mathbf{U}^* = \mathbf{Q}\overline{\mathbf{U}}\mathbf{Q}^T \tag{82}$$

will be upper triangular (and because of (73) $\mathbf{R}$ will be upper triangular, too).

More generally, suppose that $\mathbf{U}$ is a SPUT matrix (SPUT = Symmetrically Permuted Upper Triangular). Then one infers that the updated matrix $\overline{\mathbf{U}}$ of Figure 3 is also a SPUT matrix: we need one symmetric permutation to transform Figure 3 to a matrix with the same structure as that in Figure 4, and another one to render this upper triangular. As the composition of two symmetric permutations is again symmetric, this argument shows that the updating process preserves the SPUT property of $\overline{\mathbf{U}}$ also after successive simplex exchange steps.

7.2 Updating of product representation

Next we shall describe how the Forrest-Tomlin process maintains the representation of $\mathbf{B}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$ as a product of elementary row and column matrices. We will use an inductive argument to show that the current representation of the basis-inverse will be given by (61), where m is the order of $\mathbf{B}$ and t is a pivot step counter, which starts from the value m after a re-inversion and increases by 1 for each subsequent simplex iteration. When $t = m$, the expression (61) coincides with the inversion formula (58) as it should. The L-part of formula (61),

$$\mathbf{L}^{-1} = \mathbf{L}_t^{-1} \ldots \mathbf{L}_s^{-1} \ldots \mathbf{L}_1^{-1}, \tag{83}$$

clearly follows from the discnssion in Section 7.1; the new factor $\mathbf{L}_t^{-1}$ to be added at the current step t equals the elementary row matrix $\mathbf{R}$ in (74); hence it can be written in the form

$$\mathbf{L}_t^{-1} = p \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ \ell_{p1} & \ldots & 1 & \ldots & \ell_{pm} \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \tag{84}$$

p being the new pivot index. We note that for the pivot element in (84) we have

$$\ell_{pp} = 1. \tag{85}$$

The U-part of formula (61) is

$$\mathbf{U}^{-1} = \mathbf{U}_1^{-1} \ldots \mathbf{U}_i^{-1} \ldots \mathbf{U}_m^{-1}. \tag{86}$$

Notice the asymmetry between (83) and (86) regarding their number of factors. As we shall presently see, this peculiarity is related to the fact that the Forrest-Tomlin process leaves the number m of elementary factors in the U-inverse fixed.

To examine the U-part of the updating process in more detail, suppose that the previous re-inversion and subsequent simplex steps have resulted in a SPUT matrix $\mathbf{U}$. Then there exists a truly upper triangular matrix

$$\mathbf{U}^* = \begin{pmatrix} u_{11}^* & u_{12}^* & \ldots & u_{1m}^* \\ & u_{22}^* & \ldots & u_{2m}^* \\ & & \ddots & \vdots \\ & & & u_{mm}^* \end{pmatrix} \tag{87}$$

and a permutation (36) which transforms $\mathbf{U}^*$ into $\mathbf{U}$ when applied to the rows as well as the columns. We may express this fact in terms of the corresponding permutation matrix (38) by the equivalent relations

$$\mathbf{U} = \mathbf{P}\mathbf{U}^*\mathbf{P}^T \tag{88}$$

and

$$\mathbf{U}^{-1} = \mathbf{P}(\mathbf{U}^*)^{-1}\mathbf{P}^T, \tag{89}$$

where we use the orthogonality property (42) of $\mathbf{P}$. Referring again to the permutation (36), the relation (88) shows that the element in the p_ith place in the diagonal row of $\mathbf{U}$ is identical with u_{ii}^*. For this reason p_i is called the ith pivot index and u_{ii}^* the ith pivot element.

By formula (46) $\mathbf{U}^*$ can be written as a product of elementary column matrices

$$\mathbf{U}^* = \mathbf{U}_m^* \ldots \mathbf{U}_i^* \ldots \mathbf{U}_1^*, \tag{90}$$

where

$$\mathbf{U}_i^* = \begin{pmatrix} 1 & & u_{1i}^* & & \\ & \ddots & \vdots & & \\ & & u_{ii}^* & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ i \\ \\ \\ \end{matrix} \tag{91}$$

is upper triangular. By (88) we obtain a similar product representation for $\mathbf{U}$:

$$\begin{aligned} \mathbf{U} &= \mathbf{P}\mathbf{U}_m^* \ldots \mathbf{U}_i^* \ldots \mathbf{U}_1^* \mathbf{P}^T \\ &= (\mathbf{P}\mathbf{U}_m^* \mathbf{P}^T) \ldots (\mathbf{P}\mathbf{U}_i^* \mathbf{P}^T) \ldots (\mathbf{P}\mathbf{U}_1^* \mathbf{P}^T) \\ &= \mathbf{U}_m \ldots \mathbf{U}_i \ldots \mathbf{U}_1, \end{aligned} \tag{92}$$

where we again have used (42). Each factor in (92), defined as

$$\mathbf{U}_i = \mathbf{P}\mathbf{U}_i^* \mathbf{P}^T, \tag{93}$$

is an elementary column matrix of the form (cf. (25))

$$\mathbf{U}_i = \begin{pmatrix} 1 & & \cdot & & \\ & \ddots & \vdots & & \\ & & u_{ii}^* & & \\ & & \vdots & \ddots & \\ & & \cdot & & 1 \end{pmatrix} \begin{matrix} \\ \\ p_i \\ \\ \\ \end{matrix} = \mathbf{E}_{p_i}(\mathbf{y}), \tag{94}$$

p_i being the ith pivot index; the column vector $\mathbf{y} = \{y_\ell\}$ has the same elements as column i in (91), only permuted:

$$y_{p_\ell} = u_{\ell i}^*, \qquad \ell = 1, \ldots, m, \tag{95}$$

where, of course, $u_{\ell i}^* = 0$ for $\ell > i$. The inverse of (92) reads

$$\mathbf{U}^{-1} = \mathbf{U}_1^{-1} \ldots \mathbf{U}_i^{-1} \ldots \mathbf{U}_m^{-1}, \tag{96}$$

where, as a consequence of (27) - (29), $\mathbf{U}_i^{-1}$ is again an elementary column matrix,

$$\mathbf{U}_i^{-1} = \begin{pmatrix} 1 & & \cdot & & \\ & \ddots & \vdots & & \\ & & 1/u_{ii}^* & & \\ & & \vdots & \ddots & \\ & & \cdot & & 1 \end{pmatrix} \begin{matrix} \\ \\ p_i \\ \\ \\ \end{matrix} = \mathbf{E}_{p_i}(\mathbf{z}), \tag{97}$$

with the column vector $\mathbf{z}$ defined by

$$z_{p_\ell} = -\frac{y_{p_\ell}}{y_{p_i}} = -\frac{u_{\ell i}^*}{u_{ii}^*}, \qquad \ell = 1, \ldots, m, \quad \ell \neq i \tag{98}$$

and

$$z_{p_i} = \frac{1}{u_{ii}^*}. \tag{99}$$

To perform the updating we first compute $\mathbf{r}^T$ from (73), which in product form reads

$$\mathbf{r}^T = \mathbf{e}_p^T \mathbf{U}_1^{-1} \ldots \mathbf{U}_m^{-1}. \tag{100}$$

This gives us the elementary row matrix $\mathbf{R}$ defined by (74). The new pivot index p is equal to the pivot index p_i for one of the factors $\mathbf{U}_i$ given by (94), say for $i = k$, so we have $p = p_k$; we call $\mathbf{U}_k$ the "pivotal" factor. We can show that we are entitled to discard the "pre-pivotal" factors in (100) and write

$$\mathbf{r}^T = \mathbf{e}_p^T \mathbf{U}_k^{-1} \dots \mathbf{U}_m^{-1}. \tag{101}$$

To see this we insert (89) in (73),

$$\mathbf{r}^T = \mathbf{e}_p^T \mathbf{P} (\mathbf{U}^*)^{-1} \mathbf{P}^T, \tag{102}$$

take the inverse of (90),

$$(\mathbf{U}^*)^{-1} = (\mathbf{U}_1^*)^{-1} \dots (\mathbf{U}_m^*)^{-1}, \tag{103}$$

and use

$$\mathbf{e}_p^T \mathbf{P} = \mathbf{e}_k^T. \tag{104}$$

Then we get

$$\mathbf{r}^T = \mathbf{e}_k^T (\mathbf{U}_1^*)^{-1} \dots (\mathbf{U}_k^*)^{-1} \dots (\mathbf{U}_m^*)^{-1} \mathbf{P}^T, \tag{105}$$

and when we follow $\mathbf{e}_k^T$ through the successive multiplications, the first $k-1$ of these are neutral operations, hence

$$\mathbf{r}^T = \mathbf{e}_k^T (\mathbf{U}_k^*)^{-1} \dots (\mathbf{U}_m^*)^{-1} \mathbf{P}^T. \tag{106}$$

Finally, when we apply (93), (104), and (42), we get (101).

If we omit the pivotal factor from (101), (106) shows that the only effect is to multiply $\mathbf{r}^T$ by a scalar. It is easy to see that the pth element becomes 1, hence we have

$$\sigma \mathbf{r}^T = \mathbf{e}_p^T \mathbf{U}_{k+1}^{-1} \dots \mathbf{U}_m^{-1}; \tag{107}$$

this formula is useful because it is $\sigma \mathbf{r}^T$ rather than $\mathbf{r}^T$ which enters in the updating process, as (74) and (81) indicate.

Next we shall show that the transformed matrix $\overline{\mathbf{U}}$ defined in (79) and depicted in Figure 3, like $\mathbf{U}$ in (92), can be written as a product of m elementary column matrices,

$$\overline{\mathbf{U}} = \mathbf{C} \tilde{\mathbf{U}}_m \dots \tilde{\mathbf{U}}_{k+1} \mathbf{U}_{k-1} \dots \mathbf{U}_1, \tag{108}$$

where the first factor (cf. (80) and (25)) is

$$\mathbf{C} = \begin{pmatrix} & & p & \\ 1 & & v_1 & \\ & \ddots & \vdots & \\ & & \overline{u}_{pp} & \\ & & \vdots & \ddots \\ & & v_m & 1 \end{pmatrix} = \mathbf{E}_p(\mathbf{v}'), \tag{109}$$

and where $\tilde{\mathbf{U}}_i$ $(i > k)$ is defined to be equal to $\mathbf{U}_i$, except for the pth element of its pivot column, which is set to zero.

To show that (108) matches Figure 3, we first notice that the pth column of $\mathbf{C}$ in (109) coincides with the pth column of $\overline{\mathbf{U}}$ in Figure 3; as none of the other factors has a vector different from $\mathbf{e}_p$ in column p, (108) equals Figure 3 in the pth column. Similarly the pth row of $\mathbf{C}$ in (109) coincides with the pth row of $\overline{\mathbf{U}}$ in Figure 3. None of the subsequent factors in (108) will change the row, because we can show that none of the elementary columns in these matrices has a nonzero element in the pth place. For the factors $\tilde{\mathbf{U}}_i$, $(i > k)$, this follows directly from the definition, and in each of the factors $\mathbf{U}_i$, $i < k$, the pth component of its column vector equals the kth component of the column vector of $\mathbf{U}_i^*$, which is zero. Therefore (108) also matches Figure 3 in the pth row. Finally we must show that the general element $\overline{u}_{ij}$, where both i and j differ from p, is the same in Figure 3 and (108). This amounts to show that for $i \neq p$ the two row vectors $\mathbf{e}_i^T \overline{\mathbf{U}}$ and $\mathbf{e}_i^T \mathbf{U}$, where $\overline{\mathbf{U}}$ is given by (108) and $\mathbf{U}$ by (92), coincide, save for

the pth place. We verify this by following $\mathbf{e}_i^T$ through the respective postmultiplications in the expressions (108) and (92). In all nonpivotal positions this row will undergo exactly the same transformations, which again is a consequence of the zero element in the pth row of the factors $\tilde{\mathbf{U}}_i (i > k)$ and $\mathbf{U}_i (i < k)$. This concludes the proof of (108) □.

The inverse form of (108) reads

$$\overline{\mathbf{U}}^{-1} = \mathbf{U}_1^{-1} \ldots \mathbf{U}_{k-1}^{-1} \tilde{\mathbf{U}}_{k+1}^{-1} \ldots \tilde{\mathbf{U}}_m^{-1} \mathbf{C}^{-1}, \tag{110}$$

where the last factor is

$$\mathbf{C}^{-1} = \begin{pmatrix} 1 & & -v_1/\overline{u}_{pp} & & \\ & \ddots & \vdots & & \\ & & 1/\overline{u}_{pp} & & \\ & & \vdots & \ddots & \\ & & -v_m/\overline{u}_{pp} & & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ p \\ \\ \end{matrix}. \tag{111}$$

In analogy with $\tilde{\mathbf{U}}_i$ in (108), $\tilde{\mathbf{U}}_i^{-1}$ in (110) can be obtained by zeroing the pth element of the pivotal column of $\mathbf{U}_i^{-1}$. We see that the first $k-1$ factors of (110) are unaffected by the updating procedure, a consequence of the SPUT form of $\mathbf{U}$, cf. (90) and (92). The column vector in $\mathbf{C}^{-1}$ is added as a new vector to the U file. However, in practice we do not remove $\mathbf{U}_k^{-1}$ physically from the file, we only flag it as deleted, such that it virtually equals $\mathbf{I}$ (in the code the flagging is conveniently made by negating the pivot index.)

7.3 Interfacing Forrest-Tomlin with simplex

The interfacing of the Forrest-Tomlin $\mathbf{LU}$ method to the simplex procedure outlined in Section 3.3 gives rise to some amendments to some of the steps, to be listed below.

First we notice that four types of operations are needed to perform the various tasks in the simplex iterations. They are

1. Forward scan of U file, when $\mathbf{U}^{-1}$ is premultiplied by a row vector,
2. Backward scan of U file, when $\mathbf{U}^{-1}$ is postmultiplied by a column vector,
3. Forward scan of L file, when $\mathbf{L}^{-1}$ is postmultiplied by a column vector, and
4. Backward scan of L file, when $\mathbf{L}^{-1}$ is premultiplied by a row vector.

These operations are made by a traditional sparse-matrix technique, holding the U and L files in packed-array form.

Now assume that the current basis-inverse is given by (61). We may then identify the following simplex operations:

BTRAN operation: First, postmultiply the basic cost vector $\mathbf{c}_B^T$ by $\mathbf{U}^{-1}$:

$$\boldsymbol{\gamma}^T = \mathbf{c}_B^T \mathbf{U}^{-1} = \mathbf{c}_B^T \mathbf{U}_1^{-1} \ldots \mathbf{U}_m^{-1} \tag{112}$$

(forward scan of U file). Next, produce a pricing vector $\boldsymbol{\pi}^T$ using $\boldsymbol{\gamma}^T$,

$$\boldsymbol{\pi}^T = \mathbf{c}_B^T \mathbf{B}^{-1} = \boldsymbol{\gamma}^T \mathbf{L}^{-1} = \boldsymbol{\gamma}^T \mathbf{L}_t^{-1} \ldots \mathbf{L}_1^{-1} \tag{113}$$

(backward scan of L file).

PRICE and CHUZC operations: same as in Section 3.3.

FTRAN operation: If $\mathbf{a}_{m+q}$ is the column to enter the basis, form the partially updated incoming vector by (69) which in product form gives

$$\mathbf{v} = \mathbf{L}_t^{-1} \ldots \mathbf{L}_1^{-1} \mathbf{a}_{m+q} \tag{114}$$

(forward scan of L file). Complete updating of the incoming vector:

$$\boldsymbol{\alpha}_q = \mathbf{B}^{-1}\mathbf{a}_{m+q} = \mathbf{U}^{-1}\mathbf{v} = \mathbf{U}_1^{-1}\ldots\mathbf{U}_m^{-1}\mathbf{v} \tag{115}$$

(backward scan of U file).

CHUZR operation: same as in Section 3.3.

Now that we know which variables will be exchanged we are ready to execute the Forrest-Tomlin updating of the current basis, to transform $\mathbf{B}$ into the adjacent basis matrix $\overline{\mathbf{B}}$ (PIVOT Step in Section 3.3). As we saw in Section 7.2, this means that we add a new vector the the L-list, cf. (83) – (84), and carry out the transformation leading from $\mathbf{U}^{-1}$ to $\overline{\mathbf{U}}^{-1}$ in (110).

Included in the simplex step must also be an updating of the vector $\boldsymbol{\beta} = \mathbf{B}^{-1}\mathbf{b}$ to its new form $\overline{\boldsymbol{\beta}} = \overline{\mathbf{B}}^{-1}\mathbf{b}$. This is computed from the updated incoming vector $\boldsymbol{\alpha}_q = \{\alpha_{iq}\}$ in (115):

$$\overline{\boldsymbol{\beta}} = \begin{pmatrix} 1 & & -\alpha_{1q}/\alpha_{pq} & & \\ & \ddots & \vdots & & \\ & & 1/\alpha_{pq} & & \\ & & \vdots & \ddots & \\ & & -\alpha_{mq}/\alpha_{pq} & & 1 \end{pmatrix} \begin{matrix} \\ \\ p \\ \\ \\ \end{matrix} \boldsymbol{\beta}. \tag{116}$$

Equation (116) follows from the relation

$$\overline{\boldsymbol{\beta}} = \overline{\mathbf{B}}^{-1}\mathbf{B}\boldsymbol{\beta} \tag{117}$$

and the equation (35) (or from (19) and (21)).

7.4 Implementation

A straightforward implementation of the Forrest-Tomlin procedure could be summarized in the algorithm below:

Initialization part:

1. Start with some triangular factorization of the basis $\mathbf{B}$, with corresponding U and L files and pivot index list.
2. Reset iteration counter to $t = m$, corresponding to the situation immediately after an inversion.
3. Let $\boldsymbol{\beta} = \mathbf{B}^{-1}\mathbf{b}$, where $\mathbf{b}$ is the right-hand side of the system.

Simplex iteration loop:

4. Compute $\boldsymbol{\gamma}^T = \mathbf{c}_B^T\mathbf{U}_1^{-1}\ldots\mathbf{U}_m^{-1}$.
5. Compute $\boldsymbol{\pi}^T = \boldsymbol{\gamma}^T\mathbf{L}_t^{-1}\ldots\mathbf{L}_1^{-1}$.
6. Compute reduced costs $d_j = c_{m+j} - \boldsymbol{\pi}^T\mathbf{a}_{m+j}$.
7. Choose entering column $\mathbf{a}_{m+q}$ with most negative d_j. If no column can be found STOP.
8. Compute $\mathbf{v} = \mathbf{L}_t^{-1}\ldots\mathbf{L}_1^{-1}\mathbf{a}_{m+q}$.
9. Compute $\boldsymbol{\alpha}_q = \mathbf{U}_1^{-1}\ldots\mathbf{U}_m^{-1}\mathbf{v}$.
10. Choose pivot row p by the standard ratio test using $\boldsymbol{\alpha}_q$ and $\boldsymbol{\beta}$.
11. Update $\boldsymbol{\beta}$ by the column vector $\boldsymbol{\alpha}_q$.
12. In the U file, neutralize the elementary column matrix $\mathbf{U}_k$ with pivot index $= p$ (negate the pivot index for that column.)
13. Compute $\sigma\mathbf{r}^T = \mathbf{e}_p^T\mathbf{U}_{k+1}^{-1}\ldots\mathbf{U}_m^{-1}$.

14. In the U file, zero all elements in row p of the post-pivotal elementary matrix factors.
15. Compute $\mathbf{v}' = \mathbf{R}\mathbf{v}$, where $\mathbf{R} = \mathbf{I} - \mathrm{e}_p(\mathbf{e}_p^T - 1/r_p\ \mathbf{r}^T)$.
16. Compute new pivot element $\overline{u}_{pp} = v'_p$.
17. Add $\mathbf{C}^{-1} = \mathbf{I} - 1/\overline{u}_{pp}(\mathbf{v}' - \mathbf{e}_p)\mathbf{e}_p^T$ to the U file.
18. Update pivot index list with $p_{t+1} = p$.
19. Add $\mathbf{R}$ to the L file.
20. Increase iteration counter t by one and return to 4 with the updated product forms.
21. End of algorithm.

The steps 4 + 5 form the BTRAN part, 6 is PRICE, 7 is CHUZC, 8 + 9 is FTRAN, 10 is CHUZR, and the remaining steps form together the PIVOT operation.

As pointed out by Forrest and Tomlin [15], a certain amount of computer work may be saved by rearranging the steps described. Their idea is to merge the previous PIVOT updating with the BTRAN for the current simplex step by sandwiching it between operations 4 and 5, and do as much as possible of the U file reading concnrrently. The extended BTRAN works as follows:

- Make first part of operation 4 until the pivotal factor $\mathbf{U}_k^{-1}$, which is neutralized (operation 12).
- Continue the U scan till the last factor $\mathbf{C}^{-1}$ with the concurrent execution of operations 13, 14, and operation 4 after $\mathbf{U}_k^{-1}$.
- Operations 15 and 16, and then operation 17 concurrently on completing operation 4 (multiplication by $\mathbf{C}^{-1}$).
- Remaining PIVOT operations 18, 19, and 20.

The growth of the L and U files in the Forrest-Tomlin method will follow a triangular pattern and is generally slower than for the eta file in the product-inverse method.

No actual permutations are necessary: as shown the column permutations cancel, and the row permutations are indirectly taken care of by pointer arrays.

8 Miscellaneous Features

Until now we have described the fundamental building blocks of LINPROG. In the following we shall add a discussion of more specialized topics. The aim is to give the full background material for understanding how the code works. We shall use the simplex overview in Section 3 as the starting point, and step by step introduce the necessary algorithmic modifications. First we shall see how bounds and ranges in the LP formulation are incorporated in a natural way in the simplex framework. After this we describe the simplex initialization procedure in LINPROG, and next how the code finds a first feasible solution. The use of program tolerances as a means to ensure the numerical stability is outlined, and the available variants of matrix scaling are given. Finally, we survey some programming techniques, particularly those related to sparse-matrix representations.

8.1 Bounds and ranges

In the standard form of the LP given in Section 3.1, the only constraint on the solution vector $\mathbf{x} = \{x_j\}$ was the nonnegativity condition (6). But bounding constraints of the more general type

$$\ell_j \leq x_j \leq u_j \tag{118}$$

arise quite naturally in many LP applications, and as explained in Section 2, LINPROG is able to deal with such constraints. The lower bound constraints $\ell_j \leq x_j$ are simply disposed of by a substitution

$$x'_j = x_j - \ell_j, \tag{119}$$

where the bounds for x'_j are

$$\ell'_j = 0 \tag{120}$$

and

$$u'_j = u_j - \ell_j. \tag{121}$$

Assuming that the translation (119) is already carried out, we may drop the primes in x'_j, ℓ'_j, and u'_j. Henceforth we therefore assume $\ell_j = 0$ and consider upper bound constraints only:

$$0 \leq x_j \leq u_j \quad (j = 1, \ldots, n). \tag{122}$$

Formally we could postulate a restriction (122) for all j, even when x_j is unbounded above, in which case we let $u_j = \infty$. With this convention we can state (122) in vector form:

$$\mathbf{0} \leq \mathbf{x} \leq \mathbf{u}. \tag{123}$$

In analogy with (8) we order the elements of the bound vector $\mathbf{u}$ according to the current partition in basic and nonbasic variables:

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}_B \\ \mathbf{u}_N \end{pmatrix}, \tag{124}$$

such that

$$(\mathbf{u}_B)_i = u_i \quad (i = 1, \ldots, m), \qquad (\mathbf{u}_N)_j = u_{m+j} \quad (j = 1, \ldots, n-m). \tag{125}$$

Of course, constraints like (122) could be realized by adding extra rows to the constraint matrix $\mathbf{A}$ in (5). But it is inefficient to do so, because one of the critical size parameters of an LP is the number of rows m. Instead we extend the simplex idea by allowing a nonbasic variable x_j to be equal to *either* of its bounds $x_j = 0$ or (if $u_j < \infty$) $x_j = u_j$.

In the bounded simplex procedure the formulas (7) - (15) are still valid. The restriction (16) should be modified to

$$\mathbf{0} \leq \mathbf{x}_B \leq \mathbf{u}_B, \quad \mathbf{0} \leq \mathbf{x}_N \leq \mathbf{u}_N. \tag{126}$$

Also the expression (17) for the current basic solution must be altered. Let us divide the total nonbasic index set $\{1, \ldots, n-m\}$ in the lower bound set

$$L = \{j \mid x_{m+j} \text{ nonbasic at zero}\} \tag{127}$$

and the upper bound set

$$U = \{j \mid x_{m+j} \text{ nonbasic at } u_{m+j}\}. \tag{128}$$

Then (17) should be replaced by

$$\mathbf{x}_B = \tilde{\boldsymbol{\beta}}, \tag{129}$$

where we have introduced the "effective" $\boldsymbol{\beta}$-vector

$$\tilde{\boldsymbol{\beta}} = \{\tilde{\beta}_i\} = \boldsymbol{\beta} - \sum_{j \in U} u_{m+j} \boldsymbol{\alpha}_j, \tag{130}$$

with $\boldsymbol{\beta}$ defined in (12) and $\boldsymbol{\alpha}_j$ in (20). Using L and U, the objective function z in (14) can be written

$$z = \mathbf{c}_B^T \boldsymbol{\beta} + \sum_{j \in L} d_j x_{m+j} + \sum_{j \in U} d_j x_{m+j}, \tag{131}$$

where $\mathbf{d} = \{d_j\}$ is the reduced-cost vector defined by (18). Suppose now that we have a basic feasible solution for which the condition

$$\forall j \in L \quad d_j \geq 0 \quad \text{and} \quad \forall j \in U \quad d_j \leq 0 \tag{132}$$

holds. Then (131) shows that z is at its minimum; it will not decrease, if we increase x_{m+j} from zero ($j \in L$), or if we decrease x_{m+j} from u_{m+j} ($j \in U$). Thus (132) is a sufficient condition for optimality. If (132) does not hold, then either

$$\exists j \in L \quad d_j < 0, \tag{133}$$

or

$$\exists j \in U \quad d_j > 0. \tag{134}$$

In both cases we may improve on z by introducing a nonbasic variable x_{m+j} into the basis; in the former case it should be increased from zero, and in the latter case decreased from u_{m+j}.

Also the selection of the leaving variable becomes somewhat more complicated than in the algorithm with no upper bounds. This is because a basic variable $(\mathbf{x}_B)_i = x_i$ may reach either its lower bound 0 or its upper bound $(\mathbf{u}_B)_i = u_i$. Moreover, we must consider the possibility of the entering variable reaching its opposite bound and thus becoming nonbasic anew. Which of these events will first take place depends on the respective step lengths θ_1, θ_2, and θ_3, of the entering variable x_{m+q}. We see from (15) and (130) that the critical steps should be determined from the expression

$$\mathbf{x}_B = \tilde{\boldsymbol{\beta}} - x_{m+q}\boldsymbol{\alpha}_q, \tag{135}$$

if $q \in L$, and from

$$\mathbf{x}_B = \tilde{\boldsymbol{\beta}} + (u_{m+q} - x_{m+q})\boldsymbol{\alpha}_q, \tag{136}$$

if $q \in U$. We then compute θ_1 as follows:

$$q \in L: \quad \theta_1 = \min\left\{ \tilde{\beta}_i/\alpha_{iq} : i = 1, \ldots, m \text{ and } \alpha_{iq} > 0 \right\}, \tag{137}$$

$$q \in U: \quad \theta_1 = \min\left\{ -\tilde{\beta}_i/\alpha_{iq} : i = 1, \ldots, m \text{ and } \alpha_{iq} < 0 \right\}; \tag{138}$$

the step θ_2 is similarly computed as:

$$q \in L: \quad \theta_2 = \min\left\{ (\tilde{\beta}_i - u_i)/\alpha_{iq} : i = 1, \ldots, m \text{ and } \alpha_{iq} < 0 \right\}, \tag{139}$$

$$q \in U: \quad \theta_2 = \min\left\{ (u_i - \tilde{\beta}_i)/\alpha_{iq} : i = 1, \ldots, m \text{ and } \alpha_{iq} > 0 \right\}. \tag{140}$$

In (137) – (140) we use the convention

$$\min\{\emptyset\} = \infty. \tag{141}$$

We see that (137) corresponds to (21) in the standard algorithm. Because we have a basic feasible solution,

$$0 \leq x_i \leq u_i, \tag{142}$$

the two steps θ_1 and θ_2 cannot be negative. Defining

$$\theta_3 = u_{m+q}, \tag{143}$$

it is clear that the x_{m+q}-step for the first event to happen will be given by

$$\theta = \min\{\theta_1, \theta_2, \theta_3\}. \tag{144}$$

As in the standard algorithm, $\theta = \infty$ means an unbounded solution. In the two cases $\theta = \theta_1$ and $\theta = \theta_2$ we carry out a pivot step, which comprises a basis exchange operation (Forrest-Tomlin). In addition we must compute the new basic solution $\bar{\mathbf{x}}_B$. For the standard algorithm we have

$\bar{\mathbf{x}}_B = \overline{\boldsymbol{\beta}}$, where $\overline{\boldsymbol{\beta}}$ was given by (116). From (135) - (140) it can be shown that with the upper bound algorithm we should use the following more general form for $\bar{\mathbf{x}}_B$:

$$\bar{\mathbf{x}}_B = \overline{\boldsymbol{\beta}}_0 + \begin{pmatrix} 1 & & -\alpha_{1q}/\alpha_{pq} & & \\ & \ddots & \vdots & & \\ & & 1/\alpha_{pq} & & \\ & & \vdots & \ddots & \\ & & -\alpha_{mq}/\alpha_{pq} & & 1 \end{pmatrix} (\tilde{\boldsymbol{\beta}} - \tilde{\boldsymbol{\beta}}_0), \tag{145}$$

where

$$\tilde{\boldsymbol{\beta}}_0 = \begin{cases} \mathbf{0} & \text{if } x_p \text{ hits zero} \\ u_p \mathbf{e}_p & \text{if } x_p \text{ hits } u_p \end{cases} \tag{146}$$

and

$$\overline{\boldsymbol{\beta}}_0 = \begin{cases} \mathbf{0} & \text{if } q \in L \\ u_{m+q} \mathbf{e}_p & \text{if } q \in U \end{cases} \tag{147}$$

In the third case, $\theta = \theta_3$, the nonbasic variable x_{m+q} goes to its opposite bound. This must be recorded, as must also the induced change in the current basic solution; the new solution becomes (cf. (135) - (136)):

$$\bar{\mathbf{x}}_B = \tilde{\boldsymbol{\beta}} - u_{m+q} \boldsymbol{\alpha}_q \quad \text{for } q \in L, \tag{148}$$

$$\bar{\mathbf{x}}_B = \tilde{\boldsymbol{\beta}} + u_{m+q} \boldsymbol{\alpha}_q \quad \text{for } q \in U. \tag{149}$$

We can now summarize the upper bound simplex procedure in algorithmic form. Compared to the standard algorithm in Section 3.3, the only steps to modify are CHUZC, CHUZR, and the β-updating part of PIVOT:

- CHUZC Step — Choose the entering nonbasic variable ($j = q$) as one from L with negative reduced cost d_j, or one from U with positive reduced cost d_j. (In LINPROG we choose one with maximum $|d_j|$.) If no candidate q can be found, the current solution is optimal.
- CHUZR Step — Use one of the ratios (137) or (138) to compute a step θ_1 for x_{m+q} taking a basic variable x_i to zero. Also we use one of the ratio rules (139) or (140) to compute a step θ_2 taking x_i to its upper bound u_i. Put $\theta_3 = u_{m+q}$; the critical step θ is then the minimum of θ_1, θ_2, and θ_3. If $\theta = \theta_3$, we record the bound shift of the nonbasic variable and the shift of the basic solution as expressed by (148) or (149), and then we return to the CHUZC Step. If $\theta = \infty$, we have an unbounded solution. Otherwise we proceed to the PIVOT Step.
- PIVOT Step — Make the basis matrix exchange. Update the current basic solution by using (145). Return to the BTRAN Step.

Recall from Section 2, that LINPROG is able to handle a BOUNDS section with bound types UP, LO, FX, FR, PL, and MI. We have so far concentrated on the UP-facility, and this is in fact the only one which took some difficulty to implement. As already mentioned, LO is taken care of by substitution. A variable of type FX is permanently locked out from the basis, and a FR variable is locked into the basis. PL is obtained by default, and MI is realized by negating the variable.

When implementing RANGES constraints of the type given in (3) in Section 2, LINPROG takes advantage of the upper bound framework outlined previously. It transforms (3) to an equality restriction

$$\sum a_j x_j - y = 0 \tag{150}$$

by introducing a new variable y, which might be thought of as a "slack" variable associated with range constraints. The variable y is bounded below and above,

$$\ell \leq y \leq u, \tag{151}$$

and the way such a constraint is handled in LINPROG has already been discussed at length.

8.2 Simplex initialization: CRASH and zero-slack elimination

Before the standard simplex algorithm can work, we must have a basic feasible solution. LINPROG achieves this goal in several stages. Here we shall describe the first two of these: the "CRASH" procedure for setting up an initial basis, and the "Phase 0" procedure for removal of zero slacks from the basis. In Section 8.3 we will describe how the infeasibilities are driven to zero by a special pass of simplex called "Phase 1".

In the description of the ROWS Section output in Section 2.3 we saw that not only will each column in the constraint matrix be associated with a variable, but so also will each row. LINPROG uses the same principle in its internal organization of the computations. To discern the row variables from the physical variables (the latter being the "structural" variables), the former are often called "logical variables" or more preferably "slack variables".

In the LP formulation (1) we may assume that the restrictions R_i are either $\leq$ or $=$, because, as mentioned in Section 3.1, the $\geq$-restrictions are converted to $\leq$-restrictions, and the free rows are deleted from the constraint matrix. To each $\leq$-restriction there corresponds a nonnegative slack variable, and to each equality restriction a "zero-slack" variable.

Adopting this convention, LINPROG augments the original constraint matrix with a unit matrix $\mathbf{I}$, which comes from the slack variables. It is this augmented matrix that corresponds to the $m \times n$ constraint matrix $\mathbf{A}$ in the formulation (4) - (6). Rather than (6) we use the constraint (123) for $\mathbf{x}$. Notice that zero slacks may be considered as bounded variables with zero as the common lower and upper bound. Basic zero slacks will normally be infeasible.

Our first task will be to select a subset of m independent columns from $\mathbf{A}$ to form the initial basis matrix $\mathbf{B} = \mathbf{B}_0$. Assuming that all nonbasic variables start at zero, equations (17) and (12) give us the first basic solution. It might be infeasible, but we are willing to accept this in the initial stage.

Of course, an easy choice would be to select the all-slack initial basis matrix $\mathbf{B}_0 = \mathbf{I}$. The inversion in (12) would be trivial, and we would get the initial solution $\mathbf{x}_B = \mathrm{b}$. But this solution might very well contain many infeasible elements, which afterwards would entail much work in Phase 1. Fortunately we can do better than this. Following common practice in LP computations we choose a starting basis matrix $\mathbf{B}_0$ that is as close to feasibility as possible and is also (permuted) triangular. This is the CRASH procedure. A triangular basis matrix is still easy to invert. After experimenting with some CRASH variants we found that for a broad class of test problems a good strategy is to choose as many feasible slacks and structurals as possible. We do this by a recursive scanning of $\mathbf{A}$ for triangularity, starting with the unit vectors from the feasible slacks. When no more feasible slacks or structurals are available for a triangular $\mathbf{B}_0$, we are forced to fill the remaining places with infeasible slacks.

After setting up various indicators for bound status and for the nonbasic variables, LINPROG performs the initial inversion. After this the code is ready to enter the next stage, which is the elimination of zero slacks from the basis (Phase 0). One can hope that the CRASH procedure already has removed a good deal of the zero slacks, but in general there will be some left. Phase 0 resembles the other simplex phases in many respects. The basis exchange mechanism is the Forrest-Tomlin update scheme described in Section 7. The difference lies in the selection criteria for departing and entering variables. In Phase 0 we first choose the departing variable. The criteria for selecting the entering variable are first that it must not be an already eliminated zero slack (or any other fixed variable), and next that the prospective pivot element ($\overline{u}_{pp}$ in (81)) be nonzero (otherwise the new column would be linearly dependent on those already in

the basis). In practice we scan each available column until the condition

$$|\overline{u}_{pp}| \geq \epsilon_0 \tag{152}$$

is fulfilled, in which case we accept the column; if (152) is not satisfied for any column, LINPROG uses the column with greatest $|\overline{u}_{pp}|$ (ϵ_0 corresponds to the parameter ZERPIV mentioned in Section 8.5).

Suppose now that a successful pivot has been found. We then mark the eliminated zero slack as nnavailable for future re-entrance into the basis and proceed to the Forrest-Tomlin exchange procedure. On the other hand, assume that no pivot element $\overline{u}_{pp} \neq 0$ could be found. Depending on the corresponding component of the transformed right-hand side β, there are two possibilities: If the component is $\neq 0$, we have detected an infeasibility. If it is zero, the corresponding restriction is redundant. In that case it is impossible to remove the zero slack from the basis, and it must stay locked into the basis throughout the simplex process.

Also in Phase 0 we invoke re-inversions at regular intervals of the iteration counter.

8.3 Finding a first feasible solution: Phase 1

Let us recapitulate the main passes of simplex used in LINPROG:

- CRASH: Provide an initial triangular basis.
- Phase 0: Try to remove zero slacks from the basis.
- Phase 1: Establish feasibility (or demonstrate infeasibility).
- Phase 2: Establish optimality.

After CRASH and Phase 0, LINPROG enters Phase 1, in which the infeasibilities are sought removed. There are several variants of Phase 1 in use in different LP codes. For example, the zero slack eliminations, which we segregated out as a special Phase 0, could instead be integrated in Phase I. We included Phase 0 to improve the efficiency on LP problems with many equality restrictions; notice in this context that LINPROG can he used to solve n linear equations in n unknowns.

A central feature in Phase 1 is the use of a special objective function, called the "sum of infeasibilities", or SINF, instead of the objective function $\mathbf{c}^T\mathbf{x}$ in (4). We minimize SINF by the simplex technique; if the minimum is zero, feasibility is achieved. Otherwise the LP is infeasible. In this way Phase 1 is just a variant of the simplex procedure discussed in Sections 3 and 8.1. The main difference is that in Phase 1 we allow basic solutions to be infeasible. In other implementations the basic feasibility is maintained formally by introducing so-called "artificial variables". Such variables are not used in LINPROG.

When explaining our Phase 1 technique, which is inspired by the code of Bartels *et al.* [16], and which also resembles the method described by Nazareth [1], our starting point will be the bounded simplex procedure given in Section 8.1. The first step is to give a precise definition of SINF. This definition is dynamical in the sense that it reflects the infeasibility status of the current basic solution $\mathbf{x}_B = \beta$ given by (17) and (12). (Admittedly, the upper-bounding algorithm may induce a correction to β (cf. (130)), but for notational convenience we assume that this correction is already included in β.) Let us partition the basic index set

$$B = \{1, \ldots, m\} \tag{153}$$

in index sets for "feasibility", "infeasibility below lower bonnd", and "infeasibility above upper bound":

$$B = F \cup I^- \cup I^+. \tag{154}$$

To define the sets in (154) formally, assume that the ith basic variable $x_i = \beta_i$ is bounded below by ℓ_i and above by u_i (Though LINPROG works with the lower bound $\ell_i = 0$ internally, we

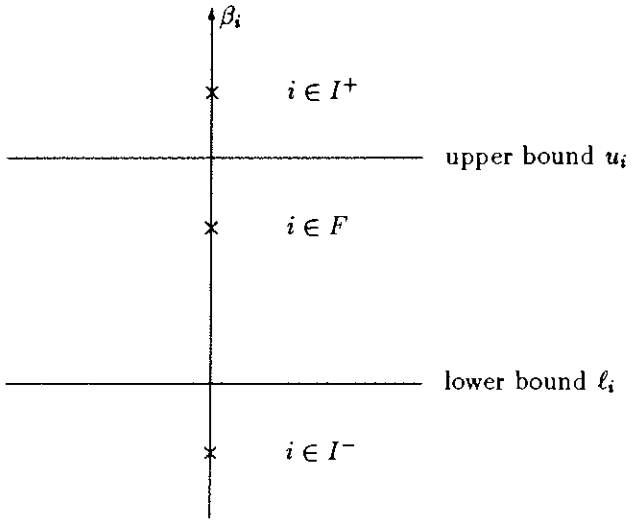

Figure 5. Possible feasibility states for basic variable $x_i = \beta_i$

keep ℓ_i here as a general parameter to make the exposition clearer.) We then have

$$F = \{i \in B | \ell_i \leq \beta_i \leq u_i\}, \tag{155}$$

$$I^- = \{i \in B | \beta_i < \ell_i\}, \tag{156}$$

and

$$I^+ = \{i \in B | \beta_i > u_i\}. \tag{157}$$

We can depict the possible feasibility states in a diagram as shown in Figure 5. The three cross marks represent the typical states for a basic variable. As a special case we may have $u_i = +\infty$. With the notation introduced we define

$$\text{SINF} = \sum_{i \in I^-} (\ell_i - x_i) + \sum_{i \in I^+} (x_i - u_i), \tag{158}$$

justifying the phrase "sum of infeasibilities" for SINF. We may also write SINF as a linear function similar to (4). To do this, we introduce the "infeasibility cost vector" $\mathbf{c}_I$. This n-vector is partitioned in a basic m-vector and a nonbasic $(n-m)$-vector as in (8):

$$\mathbf{c}_I = \begin{pmatrix} \mathbf{c}_{BI} \\ \mathbf{c}_{NI} \end{pmatrix}. \tag{159}$$

As nonbasic variables are always feasible we let

$$\mathbf{c}_{NI} = \mathbf{0}. \tag{160}$$

The ith component of $\mathbf{c}_{BI}$ is set to 0, -1, or $+1$, according to the following rules:

$$i \in F \quad \Rightarrow \quad (\mathbf{c}_{BI})_i = \ 0, \tag{161}$$

$$i \in I^- \quad \Rightarrow \quad (\mathbf{c}_{BI})_i = -1, \tag{162}$$

$$i \in I^+ \quad \Rightarrow \quad (\mathbf{c}_{BI})_i = +1. \tag{163}$$

We see that apart from a constant, SINF is the inner prodnct of $\mathbf{c}_I$ with $\mathbf{x}$:

$$\text{SINF} = \mathbf{c}_I^T\mathbf{x} + \sum_{i\in I^-} \ell_i - \sum_{i\in I^+} u_i. \tag{164}$$

In contrast to (4), the linear form (164) is not fixed, but changes from iteration to iteration. Nevertheless, the main simplex principle of reducing the objective fnnction from one iteration to the next still holds, such that we expect Phase 1 to be finished in a finite nnmber of steps, ignoring the possibility of cycling.

We may also speak of the *number* of infeasibilities

$$\text{NINF} = |I^- \cup I^+|. \tag{165}$$

When feasibility is attained, then I^- and I^+ are empty; hence NINF $= 0$, while (158) gives SINF $= 0$. Also $B = F$, and by (160) and (161) $\mathbf{c}_I = \mathbf{0}$. We therefore deduce that

$$\mathbf{c}_I = \mathbf{0} \quad \Longleftrightarrow \quad \text{NINF} = 0 \quad \Longleftrightarrow \quad \text{SINF} = 0, \tag{166}$$

such that the fnlfillment of any of these three conditions means feasibility, in which case Phase 1 terminates. Otherwise we proceed by computing the "infeasibility pricing vector"

$$\boldsymbol{\pi}_I^T = \mathbf{c}_{BI}^T\mathbf{B}^{-1} \tag{167}$$

by the same BTRAN operation as used for the normal pricing vector (13); after this the vector of "reduced infeasibility costs"

$$\mathbf{d}_I^T = \mathbf{c}_{NI}^T - \boldsymbol{\pi}_I^T\mathbf{N} = -\boldsymbol{\pi}_I^T\mathbf{N} \tag{168}$$

is computed in analogy with (18). We can now argue in precisely the same way as in Sections 3.2 and 8.1 to obtain conditions for SINF to be minimal, and, if it is not minimal, to find an entering variable to reduce it. We find that if $(\mathbf{d}_I)_j \geq 0$ for all nonbasic variables at lower bound and $(\mathbf{d}_I)_j \leq 0$ for all nonbasic variables at npper bonnd, then SINF is minimal. Formally this condition can be stated as in (132):

$$\forall j \in L \;\; (\mathbf{d}_I)_j \geq 0 \quad \text{and} \quad \forall j \in U \;\; (\mathbf{d}_I)_j \leq 0. \tag{169}$$

We know that SINF > 0, and if (169) holds, our LP must be infeasible. If not we use exactly the same CHUZC procedure as in Section 8.1 to select a variable x_{m+q} to be introduced into the basis.

The more difficult part of Phase 1 is to devise a strategy for selecting a variable to leave the basis (CHUZR). Again a number of variants are in use in different LP codes. The method in LINPROG forms a natnral generalization of the bounded simplex algorithm for feasible solutions given in Section 8.1. It works well for the test problems we have studied.

We assnme that the CHUZC procedure by now has selected the variable x_{m+q} to be introduced into the basis. We mnst then choose an index $i = p$ from the set B in (153) identifying a variable to leave the basis. Consider the updated incoming vector (cf. (115) and (34)):

$$\boldsymbol{\alpha}_q = \{\alpha_{iq}\}_{i=1,\ldots,m} = \mathbf{B}^{-1}\mathbf{a}_{m+q}. \tag{170}$$

First we notice that a leaving candidate $i = p$ must have its pivot element $\alpha_{pq} \neq 0$. We therefore introdnce the index set B_1 of candidates available for pivoting:

$$B_1 = \{i \in B | \, \alpha_{iq} \neq 0\}, \tag{171}$$

and in the following discussions we rnle out all candidates $i \notin B_1$. We can partition B_1 in the same way as we did for B in (154):

$$B_1 = F_1 \cup I_1^- \cup I_1^+, \tag{172}$$

where

$$F_1 = F \cap B_1 = \{i \in B_1 | \ell_i \leq \beta_i \leq u_i\}, \tag{173}$$

$$I_1^- = I^- \cap B_1 = \{i \in B_1 | \beta_i < \ell_i\}, \tag{174}$$

and

$$I_1^+ = I^+ \cap B_1 = \{i \in B_1 | \beta_i > u_i\}. \tag{175}$$

Thus F_1 is the feasible part of B_1. The infeasible part is

$$I_1 = I_1^- \cup I_1^+. \tag{176}$$

Making x_{m+q} basic means that either it rises from its lower bound ℓ, or it decreases from its upper bound u. If θ is the absolute value of the x_{m+q}-step in both cases, we can calculate the change in the ith basic solution component using (19), which holds for small enough $\theta \geq 0$:

$$x_i = \beta_i \mp \theta \alpha_{iq}, \tag{177}$$

where the upper sign applies if x_{m+q} rises, and the lower if it decreases. As we have already excluded the case $\alpha_{iq} = 0$, we know that the basic variable x_i will move from one of the typical cross positions for β_i in Figure 5, when θ increases from 0. Whether (177) represents an increase or a decrease of x_i from β_i depends on the bound of x_{m+q} and the sign of α_{iq}. For each candidate $i \in B_1$ let us define a *direction indicator* D_i to be $\uparrow$ ("up") or $\downarrow$ ("down"), by the following rule:

$$D_i = \begin{cases} \downarrow & \text{if} \quad (\alpha_{iq} > 0 \wedge x_{m+q} = \ell) \vee (\alpha_{iq} < 0 \wedge x_{m+q} = u) \\ \uparrow & \text{if} \quad (\alpha_{iq} < 0 \wedge x_{m+q} = \ell) \vee (\alpha_{iq} > 0 \wedge x_{m+q} = u). \end{cases} \tag{178}$$

Using this direction indicator, we partition the infeasible set I_1 in (176) in another way:

$$I_1 = I_{\text{better}} \cup I_{\text{worse}}, \tag{179}$$

where

$$I_{\text{better}} = \{i \in I_1^+ | D_i = \downarrow\} \cup \{i \in I_1^- | D_i = \uparrow\}, \tag{180}$$

and

$$I_{\text{worse}} = \{i \in I_1^+ | D_i = \uparrow\} \cup \{i \in I_1^- | D_i = \downarrow\}. \tag{181}$$

The set I_{better} represents infeasible candidates moving towards feasibility, while I_{worse} represents infeasible candidates moving further away from feasibility. It is clear that we can never accept a candidate from I_{worse}:

$$p \notin I_{\text{worse}}. \tag{182}$$

We therefore concentrate on the candidate set

$$C = F_1 \cup I_{\text{better}}, \tag{183}$$

and for each $i \in C$ we compute certain critical step lengths for x_{m+q}, taking the corresponding basic variable x_i to either of its bounds ℓ_i or u_i:

$$i \in F_1: \qquad \theta_i = \begin{cases} (u_i - \beta_i)/|\alpha_{iq}| & \text{for } D_i = \uparrow \\ (\beta_i - \ell_i)/|\alpha_{iq}| & \text{for } D_i = \downarrow, \end{cases} \tag{184}$$

$$i \in I_{\text{better}}: \qquad \theta_{in} = \begin{cases} (\beta_i - u_i)/|\alpha_{iq}| & \text{for } D_i = \downarrow \\ (\ell_i - \beta_i)/|\alpha_{iq}| & \text{for } D_i = \uparrow, \end{cases} \tag{185}$$

and

$$i \in I_{\text{better}}: \qquad \theta_{if} = \begin{cases} (\beta_i - \ell_i)/|\alpha_{iq}| & \text{for } D_i = \downarrow \\ (u_i - \beta_i)/|\alpha_{iq}| & \text{for } D_i = \uparrow. \end{cases} \tag{186}$$

Next we calculate

$$\theta_1' = \min\{\theta_i : i \in F_1\}, \tag{187}$$

$$\theta_1'' = \min\{\theta_{iJ} : i \in I_{\text{better}}\}, \tag{188}$$

and from this

$$\theta_1 = \min\{\theta_1', \theta_1''\}. \tag{189}$$

We also compute

$$\theta_2 = \max\{\theta_{in} : i \in I_{\text{better}}\}. \tag{190}$$

In these formulas, θ_i takes a feasible x_i to its bound, θ_{in} takes an infeasible x_i to its nearest bound, whereas θ_{iJ} takes an infeasible x_i to its farthest bound. It may happen that $F_1 = \emptyset$. Also I_{better} may be empty, but this happens only when the current solution is feasible. To cover these cases formally, we use the conventions (141) and

$$\max\{\emptyset\} = \infty. \tag{191}$$

Moreover we let

$$\theta_3 = u - \ell, \tag{192}$$

i.e. the difference between the upper and lower bounds for the entering variable. Then finally, we use the step

$$\theta = \min\{\theta_1, \theta_2, \theta_3\}. \tag{193}$$

If θ_3 becomes blocking, the leaving and entering variables coincide.

To state it in another way, we can characterize the properties of our strategy as follows: The leaving variable goes to one of its bounds. No feasible variable goes infeasible, hence the number of infeasibilities NINF is monotonically decreasing. No infeasible variable can cross the feasible range and again go infeasible. Within these constraints we eliminate as many infeasibilities as we possibly can. This Phase 1 strategy does not guarantee that the sum of infeasibilities SINF be monotonic decreasing, too. It would be so if we always were to stop at the nearest boundary; this is a consequence of the way we select the entering variable from the "reduced infeasibility costs" in CHUZC. But when we allow an infeasible variable x_i to go to its farthest bound, there exists a possibility that SINF might increase: the initial decrease in SINF could be outweighed by a subsequent increase, because when x_i moves into the feasible range between the bounds, we suddenly lose its contribution to the decrease in SINF. Notice however, that when this happens, NINF certainly decreases, which is important for the convergence of the procedure.

Let us summarize the Phase 1 CHUZR in compressed algorithmic form, as we did it for the Phase 2 CHUZR in Section 8.1:

- CHUZR Step in Phase 1 — Compute the least step θ_1' taking a feasible basic variable to its bound by (187). Compute the least step θ_1'' taking an infeasible variable to its farthest bound by (188). Find the minimum θ_1 of these two steps. Compute the largest step θ_2 taking an infeasible basic variable to its nearest bound by (190). Compute the difference θ_3 between the upper and lower bounds for the entering variable. Take the least of the three steps θ_1, θ_2, and θ_3, to be the resulting step θ of the entering variable. If $\theta = \theta_3$ make the entering variable nonbasic at its opposite end. Otherwise identify the leaving variable and proceed to the PIVOT Step.

We have not paid attention to problems arising from degeneracies, which in connection with rounding errors may introduce serious troubles in practical computations. A common means of trying to improve the robustness of the CHUZR procedure is to use a two-pass version of the procedure, where the bounds are slightly perturbed in the first pass, but treated exactly in the second. This technique, which dates back to Harris [17], has also been tried in LINPROG, but we did not find the results sufficiently rewarding to justify the coding complications. Instead, we use feasibility tolerances in a much simpler way (Section 8.5).

Degeneracies of the type $u_i = \ell_i$ for basic variables will not occur in Phase 1. Like zero slacks such variables are considered as fixed variables by LINPROG, and no fixed variable can be basic and available for pivoting in Phase 1.

When Phase 1 terminates with a feasible solution, we drop the infeasibility cost function and return to the genuine objective function of (4). This will be minimized in "Phase 2" of simplex, where we use the mechanisms described in Section 3 together with the upper-bounding amendments in Section 8.1. In fact the above description of the Phase 1 CHUZR algorithm also holds for Phase 2, where now all the "infeasible" sets are empty.

8.4 Pricing strategies. Cycling.

As described in Sections 3.2 and 8.1, LINPROG uses a simple criterion for selecting the entering variable: First the reduced costs d_j are calculated by (18) (operation PRICE), and next the code seeks out a column with maximum $|d_j|$ among the candidates with the proper sign of d_j (operation CHUZC). Taken together, PRICE and CHUZC form our pricing strategy. Many refinements of this strategy are in use in commercial LP software. Some of them will be mentioned here with comments on their potential usefulness for enhancing the performance of LINPROG.

The advanced pricing methods can be divided in two classes. In the first class of methods an attempt is made to reduce the number of simplex iterations by modifying the CHUZC selection criterion based on d_j. This requires an increase of the work per iteration, however. The second class of methods seeks to reduce the average work per iteration at the expense of some increase of the iteration count.

To the first class belong the *steepest-edge* pricing methods, of which the DEVEX scheme by Harris [17] and the method of Goldfarb and Reid [18] are the best known. The simple LINPROG pricing based on the d_j-criterion corresponds to finding an edge of the polytope (Section 3.2) with maximum absolute value of the gradient of the objective function, measnred in the *reference space* of the current nonbasic variables. This reference space changes at each iteration. Steepest-edge pricing, on the other hand, uses gradients in a *fixed* reference space. Harris uses a reference space spanned by the structural variables of the LP. She computes the gradients in this space by attaching proper weight parameters w_j to the reduced costs d_j, such that d_j/w_j rather than d_j are used for selecting the incoming variable. The weights w_j, being costly to compute afresh, are estimated from previous iterates using a heuristic recursion formula. Goldfarb and Reid use a similar approach. Their method differs from that of Harris by the use of a reference space spanned by all the LP variables, and the use of an exact recursion formula for the weights. Both methods affect a dynamic scaling of the columns. We have experimented with the Goldfarb and Reid method in LINPROG for selected sample problems. As expected, the result was a significant reduction of the iteration count (typically about 40 per cent), but this gain was offset by the extra work involved in the pricing operation, so that no overall improvement could be measured. Moreover, the recurrence formula seemed to induce numerical instability. We have not tried Harris's scheme, but this is generally rated equal to that of Goldfarb and Reid in efficiency. Of course, such comparisons depend a lot on the computer environment. Our conclusion was that it is not worthwhile to use steepest-edge pricing in LINPROG.

Another way to reduce the number of simplex iterations is by "multiple-target" pricing, where more than one objective function is used concurrently. For example, in Phase 1 we could compute reduced costs for *both* the sum of infeasibilities *and* the "genuine" objective function and find an incoming variable which decreases both of them. (The so-called "Big-M" method [6, 1] forms a variant of this principle; its use, however, is discouraged for numerical reasons.) It is also possible to select a secondary objective function in such a way that degeneracies are impeded. This technique, which is described by Nazareth [1], has been tried in LINPROG, but without sufficiently encouraging results. Otherwise, multiple-target pricing methods have not been tried by us.

Let us briefly consider the other class of pricing methods, in which one seeks to lessen the

average work in the pricing process. The motivation for such methods is that the full product $\pi^T \mathbf{N}$ in (18) represents $n-m$ inner-product operations and may therefore be costly to compute for big problems with many nonbasic variables.

With *partial pricing* only a portion of the nonbasic columns in $\mathbf{N}$ is scanned at each iteration to find an incoming candidate. At the next iteration another portion is scanned, for example by starting where the last scan ended. Experiments with partial pricing in LINPROG did not show significant improvement of performance, but the reason could be that few of our large test cases have $n \gg m$.

With *multiple pricing* one seeks out a number of promising candidates, say five, with "good" reduced costs, and processes them together in extended FTRAN, CHUZR, and PIVOT operations. Only one candidate, of course, is selected to be introduced into the basis, but for the next iteration only the remaining four columns are considered. The process continues as long as the reduced costs admit it. In this way a subproblem is defined, within which *minor iterations* take place. In the next *major iteration* we define a new subproblem with five new candidates, and so on. This technique should reduce the total simplex work. It is also possible to make a local CHUZR optimization within the subproblem by finding a step θ which reduces the objective function as much as possible. Preliminary experiments with multiple pricing indicate that this technique might indeed significantly improve the efficiency in LINPROG. Maybe the best strategy would be to combine multiple pricing with partial pricing. The present version of LINPROG contains no multiple pricing.

We shall also comment on *cycling*. It was mentioned in the description of the standard simplex method in Section 3.2 that basic solutions may become degenerate such that one or more basic variables are zero (or more generally, equal to one of their bounds). Then there exists a possibility, that the step size θ in (21) equals zero, with the consequence that the objective function is unaltered. After a series of snch exchanges between degenerate solntions we might again return to the same set of basic variables, and we have an instance of cycling.

It is generally accepted that cycling is an unlikely phenomenon in the standard simplex algorithm. However, the various simplex modifications we have described, viz. the nse of a dynamical snm-of-infeasibility form SINF and of the upper-bonnding algorithm, increase the risk of cycling. In ill-conditioned or badly scaled LP problems the step length θ determined in the CHUZR procedure is often so small that it may be difficult to discern it from zero, due to the influence of the simplex tolerances (Section 8.5). In fact, there have been certain examples of cycling in LINPROG and its predecessors. We were able to protect ourselves against some of these instances, but not all, so the possibility still exists that cycling may strike unexpectedly (this happened in the PILOT.JA test case, see Section 9.3).

In LINPROG the criterion for choosing a θ_3-step (193) seems to be a potential source for starting a cycling. We have seen examples of indefinitely alternating selections between θ_3 and θ_1; here the cure was a systematic tiebreaking rule (see Section 8.5). In other cases θ_3 was selected every time, such that the state of a nonbasic variable oscillated between its two bounds. This situation could be avoided by forbidding an immediate re-selection of θ_3.

Apart from these precautions we have not bothered to build a "cycle-protection" device into the code. Such protection methods are given in the literature. For example, Garfinkel and Nemhauser [19] describe a technique based on a lexicographic ranking rule for vectors; Gill *et al.* [20] discuss practical experiences from using an "anti-cycling procedure" in LP computations.

8.5 Scaling, LP tolerances, and numerical stability.

When an LP is solved by a computer, the finite precision arithmetic will inevitably induce rounding errors. Such errors are well-known from solution of equations, but their impact is mnch more unpredictable when ineqnalities are considered. Following common practice, LINPROG uses a set of *tolerances* to provide margins of error when comparisons are made between two numbers or between a number and zero, as required in the simplex procedure.

To make the use of such tolerances meaningful, the LP problem should first be brought on a

suitable standard form. Therefore, before starting the simplex algorithm, LINPROG normally makes a *scaling* of the left-hand side of the LP system (1). We have seen in Section 2.1, that there are several possible scaling options, governed by the numerical keyword MSCALE. By default (MSCALE = 1), LINPROG makes only a *row scaling*, which has the effect that the maximum norm of each scaled row becomes unity:

$$\max\{|a_{ij}| : j = 1, \ldots, n_s\} = 1. \tag{194}$$

The objective row is not scaled. Experiments show that this type of scaling is adequate for a major class of LP problems. Sometimes, however, it might be better to use *column scaling*. This is achieved in LINPROG by specifying MSCALE = 2. Then the maximum norm of each scaled column becomes unity:

$$\max\{|a_{ij}| : i = 1, \ldots, m\} = 1. \tag{195}$$

As a third possibility (MSCALE = 3) LINPROG can scale rows *and* columns, such that the scaled coefficients satisfy (194) as well as (195). A simple code for this purpose was found in Bartels *et al.* [16]. We use their method and compute the row and column scales, r_i and s_j, by the following algorithm (given in pseudo-PASCAL):

FOR $i := 1$ **TO** m **DO** $r_i := 0$;
FOR $j := 1$ **TO** n_s **DO**
  BEGIN
    $h := 0$;
    FOR $i := 1$ **TO** m **DO** $h := \max(h, |a_{ij}|)$;
    $s_j := 1/h$;
    FOR $i := 1$ **TO** m **DO** $r_i := \max(r_i, s_j|a_{ij}|)$
  END;

The matrix elements can now be normalized by r_i and s_j as follows:

FOR $j := 1$ **TO** n_s **DO**
  FOR $i := 1$ **TO** m **DO** $a_{ij} := s_j/r_i \times a_{ij}$;

It is easy to see that after this operation both (194) and (195) are fulfilled.

Of course, row scaling affects the right-hand side as well. Likewise, column scaling induces a scaling of the structural variables.

Other LP systems may offer more elaborate scaling procedures, though the benefit from them is sometimes questioned. In some systems it is possible to specify column scale numbers s_j directly in the input. A survey of scaling methods for LP is given by Tomlin [21].

It should be noticed that the LINPROG user can suppress the scaling completely by letting MSCALE = 0. This could be appropriate when the original formulation (1) is already well scaled.

LINPROG uses nine fundamental tolerance parameters. The number of parameters as well as their default values are subject to possible future changes. Using the FORTRAN names, we list in Table 3 the present setting and use of the parameters. The default values were chosen after much trial work with our test problems; they apply to computers with (at least) 8 bytes working precision. The choice of tolerances should be suitable for a wide class of problem types. As explained in Section 2.1, the default settings can be modified by use of numerical keywords in the Control File.

The parameter BIG is a computer substitute for infinity. The other parameters are used to "kill' small numbers, to perturb feasibility ranges, resolve "ties", or prevent use of small pivots. We give a short account of each of them in the following.

The parameter EPSCHC is used in the CHUZC procedure for deciding when a reduced cost d_j is insignificant. We use a relative test:

$$|d_j| \leq \text{EPSCHC} \, \|\boldsymbol{\pi}\|_1 \quad \Rightarrow \quad d_j \text{ insignificant}, \tag{196}$$

Parameter	Value	Usage
BIG	$1.0\ 10^{+30}$	"Approximation" of $+\infty$
EPSCHC	$1.0\ 10^{-10}$	Reduced cost tolerance
EPSCHR	$1.0\ 10^{-10}$	CHUZR tolerance
EPSFEA	$1.0\ 10^{-10}$	Feasibility tolerance
EPSINA	$1.0\ 10^{-08}$	Tolerance for nonzero input coefficients
EPSLU	$1.0\ 10^{-13}$	Tolerance for new elements in eta vectors
EPSPIV	$1.0\ 10^{-09}$	Minimum pivot size in CHUZR
EPSRIN	$1.0\ 10^{-02}$	Minimum relative pivot size in inversion
ZERPIV	$1.0\ 10^{+00}$	Minimum pivot size in Phase 0

Table 3. Fundamental tolerance parameters in LINPROG

where π is the pricing vector defined in (13) (or (167) for Phase 1); $\|\cdot\|_1$ stands for the"1-norm", which is the sum of the absolute values of the vector components. The test (196) is also used to set insignificant dual activities to zero in the output (these are reduced costs for the slack variables.)

The parameter EPSCHR is used in CHUZR. It works by making a perturbation of the feasible range, thus permitting very small infeasibilities. Given a basic solution $\mathbf{x}_B = \beta = \{\beta_i\}$ $(i = 1, \ldots, m)$, LINPROG first computes the perturbation

$$\text{TOLCHR} = N(\beta)\ \text{EPSCHR}, \tag{197}$$

where N is a norm-like function defined by

$$N(\beta) = \frac{1}{\sqrt{m}} \max\{\|\beta\|_1, m\}, \tag{198}$$

and then replaces the feasibility test in (155) by

$$\ell_i - \text{TOLCHR} \leq \beta_i \leq u_i + \text{TOLCHR}. \tag{199}$$

The heuristic formula (198) was chosen after theoretical and practical considerations combined with numerical experiments. Experience shows that the $\sqrt{m}$ divisor, which also occurs in tests in other LP codes, e.g. [4], provides sensible feasibility tolerances TOLCHR over a wide span of matrix sizes and sparsities. Notice that N is not a genuine norm function, as $N(\mathbf{0}) = \sqrt{m} > 0$. We made this amendment to prevent TOLCHR to become exactly zero, should β be zero.

Consider now the important default case $u_i = +\infty$. As LINPROG internally sets $\ell_i = 0$, (199) reduces to

$$0 \leq \beta_i + \text{TOLCHR}; \tag{200}$$

in the ratio test (21) we therefore replace the numerator β_i by $\beta_i +$ TOLCHR. This "perturbed" test has the advantage that should two ratios in (21) be equal, then priority is given to the larger pivot size $|\alpha_{iq}|$, thus enhancing the numerical stability.

Another use of EPSCHR in CHUZR is as a tool for *tie breaking*. Often in simplex a candidate is selected from some minimum or maximum criterion, and when several candidates are equal, we must break the tie. This can usually be done arbitrarily, by taking the first candidate in the list. Sometimes we use a secondary criterion, say pivot size, as we did in the inversion procedure (Section 6.4).

It was pointed out in Section 8.4, that the choosing process for θ_3 in (193) might have a critical influence on cycling. When a tie occurs, cycling is counteracted by a systematic favouring of one of the possibilities. We do this by replacing θ_3 in (193) with $\theta_3/(1 + \text{EPSCHR})$.

The use of EPSFEA is similar to EPSCHR. It is used to derive a tolerance

$$\text{TOLFEA} = N(\beta)\ \text{EPSFEA}, \tag{201}$$

which in turn is applied in the evaluation of the infeasibility indicators NINF and SINF (Section 8.2). We could have used a common parameter for TOLCHR and TOLFEA, but two parameters give more flexibility in monitoring the feasibility of the LP solutions.

The tolerance parameter EPSINA is used as a filter for small input elements in the constraint matrix, in the following simple way:

$$|a_{ij}| \leq \text{EPSINA} \quad \Rightarrow \quad a_{ij} := 0. \tag{202}$$

The parameter EPSLU is used in the same way as a filter for small elements in the newly created eta vectors in the Forrest-Tomlin updating process. For the column vector (cf. (109)) we make the test

$$|v_i| \leq \text{EPSLU}\,|\overline{u}_{pp}| \quad \Rightarrow \quad v_i := 0. \tag{203}$$

For the row vector in (74) we make a similar test, only we do not multiply EPSLU by $|\overline{u}_{pp}|$ in that case.

The parameter EPSPIV is the minimum pivot size we are willing to accept in a simplex iteration. It is used in CHUZR and in Phase 0.

The threshold parameter EPSRIN corresponds to ϵ in Section 6.4 and serves, as described there, to ensure the numerical stability of our re-inversion when choosing pivots.

Finally ZERPIV is a threshold parameter for the pivot size in Phase 0. It corresponds to ϵ_0 in Section 8.2, cf. the test (152). Numerical experiments indicate that the value of ZERPIV should be rather high.

If we compare the Forrest-Tomlin scheme in LINPROG by the Bartels-Golub update procedure as used e.g. in [16], we must admit that inherently the latter is the more stable of the two. With the Forrest-Tomlin method we may sometimes get rather small pivots, and if this happens too often, the representation of $\mathbf{B}^{-1}$ becomes inaccurate. As a consequence of this, we monitor the basic solution for feasibility also in Phase 2 using the TOLFEA tolerance of (201). To save work, however, we check the solution only after a re-inversion and again by the end of Phase 2. Should unexpected infeasibilities then be detected, the computations are thrown back to Phase I, where feasibility normally is restored after a few iterations. This "repair" procedure seems to work well in practice. The idea behind the strategy may be characterized as "cure is better than prevention".

Rounding errors set ultimate bounds on how large LP problems we can solve. "Large" means here both large dimensions of the constraint matrix and many nonzero elements. A positive impact of the rounding errors is, however, that they help resolve degeneracies and thereby avoid cycling in practical calculations.

8.6 Sparse-matrix and other programming techniques

As mentioned in the Introduction the application of sparse-matrix technique is essential for handling large LP problems. There are several ways to represent sparse matrices and vectors in a computer. The techniques in LINPROG are based on the so-called "ordered lists". Other codes may use "linked lists". A survey of sparse-matrix methods can be found in the book by Duff *et al.* [22].

The constraint matrix is conveniently stored by columns with row numbers associated with each non-zero element. Only the non-zeroes are stored, and there are no gaps in the lists. Thus the matrix is stored as a collection of packed sparse vectors. As the computer is supposed to have virtual memory, it is practical to let the lists be two arrays in the program. In addition we use a pointer array holding the addresses of the last non-zeroes in each column. With these lists it is easy to perform the necessary operations involving the constraint matrix.

In the Forrest-Tomlin updating scheme we may also represent the column eta vectors by a string of packed sparse vectors, and similarly for the row eta vectors.

Such a simple compact representation is not possible when fill-in occurs as a result of an updating process. In LINPROG this happens during the re-inversion of the basis matrix. We

described in Section 6.5 how we used double ordered lists with elbow room as temporary storage for the "Boolean" part of the inversion.

Before LINPROG can set up its sparse representation of the constraint matrix, an internal matrix generator must interpret the contents of the Matrix File discussed in Section 2.2. We recall that each row was given an 8-character name in the ROWS Section. In the COLUMNS Section the column names were supplied successively, finishing one column before going to the next. Alongside, with each column name a set of pairs was given (row name, element value) corresponding to the nonzero coefficients in the column. The row names could come in any order within a column. When converting this structure to an LP matrix, it would be expensive to identify the row names in the COLUMNS Section by scanning the total input list of row names each time. Instead LINPROG starts to sort the row names in the ROWS Section input in alphabetic order. To do this the names are mapped into pairs of integers corresponding to character positions 1–4 and 5–8 via the ASCII standard collating sequence defining the bit patterns of the characters. The same sorting procedure is executed for each column in the COLUMNS Section input. This means that the row names in a column can be identified in a single merge-scan with the total sorted list of row names. The sorting algorithms applied in LINPROG are variants of the quicksort method of van Emden [23].

9 Test of LINPROG

LINPROG has undergone an extensive testing procedure involving 165 individual test problems. These test problems span in size from LP matrices with only a couple of rows and columns for the smallest, to many thousands of rows and columns for the largest.

9.1 Sources of test problems

The test problems are collected from many sources. We have taken small examples from a textbook in numerical analysis by Fröberg [24], from Cohn's book on linear algebra [25], from the books on linear programming by Murtagh [6] and by Nazareth [1], from Garfinkel and Nemhauser's book on integer programming [19], and from the user guides for two other LP systems, MPSX [2] and MINOS [4].

We have also constructed sample problems ourselves in order to test certain special features. For example, we have constructed an example of a solution of 200 equations in 200 unknowns with a random sparsity pattern and random values of the nonzeroes.

Many of the medium and larger test cases originate from various LP applications at Risø. Among these some of the earliest were contributed by Steen Weber (personal communication), who used LP as a tool for predicting optimal fuel management schemes for nuclear reactors. Weber's test cases are not particularly large, but they are quite dense and may present some numerical difficulties.

Later, J. Munksgaard Pedersen in a Ph.D. thesis work [26] used LP as a requisite for planning economic resources under strict limitations of supply. His cases are small to medium (from 27 to about 1400 rows in the LP matrix). They contain many redundancies and lead typically to highly degenerate solutions. We have made variants of his cases in which the restrictions are defined in BOUNDS Sections whenever possible.

The single project, that provided the largest test problems and presented the greatest challenges to the further development of LINPROG is, however, the so-called EFOM project. EFOM is a computer modelling package [27] developed by the European Communities. It is intended for nationwide optimization of energy systems; it generates an LP problem which is supposed to be solved by extraneous software. Denmark has used the EFOM model since 1979, where the LP was solved by an early version of LINPROG. J. Fenhann, Risø, had been operating

the EFOM system during that time, and we have saved a number of his computations as test problems. More recently, from about 1987, a new series of EFOM computations was initiated at Risø by O. Gravgård Pedersen, and now carried on by P. E. Grohnheit. The EFOM software has improved much since 1979, and so has LINPROG as a result of the demands to the LP solver to cope with bigger and bigger models.

Recently we acquired a valuable collection of LP test cases from the NETLIB service [3]. Via electronic mail we received 65 test problems, which are recognized worldwide as benchmark examples for solving LP problems. The NETLIB cases proved to be of utmost importance for the consolidation of LINPROG, and several bugs were located when we tried these examples.

9.2 List of test results

In the following we list the results of our tests. To save space, we show only the output summary line produced by LINPROG for each problem. Comments on the tests are given in Section 9.3.

CASE	ROWS	COLS	ELEM	PH.0	PH.1	ITNS	VIOL	OPTIMUM	CPU-SEC
========== FILE SMALLCAS (VARIOUS SMALL CASES) RUN 900315 ==========									
GARFIN37	3	3	8	0	1	3	0.0E+00	7.7500000000000E+00	0.03
GARFIN47	4	2	8	0	0	2	0.0E+00	-7.6666666666667E+00	0.02
GARFINMI	3	3	8	0	1	3	0.0E+00	7.7500000000000E+00	0.03
MURTARED	3	3	6	1	0	1	0.0E+00	0.0000000000000E+00	0.01
FULLFOUR	5	4	16	4	0	4	2.2E-16	0.0000000000000E+00	0.03
FULLFIVE	6	5	25	5	0	5	2.2E-16	0.0000000000000E+00	0.05
PENDLING	8	7	13	0	0	0	0.0E+00	0.0000000000000E+00	0.04
MINIDATA	8	7	16	0	0	1	0.0E+00	7.2000000000000E+02	0.01
DIETPROB	4	6	24	0	1	2	0.0E+00	9.2500000000000E+01	0.03
ZERTHREE	4	3	6	3	0	3	0.0E+00	0.0000000000000E+00	0.02
BLENDING	8	4	20	2	1	4	0.0E+00	5.1340000000000E+01	0.03
CRISIS1	4	3	12	3	0	3	2.2E-16	-1.4454545454545E+01	0.02
LIQUOR	7	5	15	0	2	4	0.0E+00	-2.4500000000000E+02	0.04
========== FILE COHNDATA (EXAMPLES FROM COHN) RUN 900315 ==========									
COHNEXP1	3	4	10	0	0	1	0.0E+00	-1.5000000000000E+01	0.04
COHNEXP2	4	4	16	3	0	3	2.2E-16	1.2000000000000E+01	0.02
COHNEXP3	4	3	12	0	0	2	0.0E+00	-3.9411764705882E+00	0.03
COHNEXC1	4	4	14	3	0	3	8.9E-16	6.0000000000000E+00	0.03
COHNEXCF	4	4	14	3	0	4	2.2E-15	7.1772575250836E+00	0.03
COHNEXC2	4	3	11	0	0	2	0.0E+00	-2.0625000000000E+01	0.03
COHNEXC3	4	3	11	0	1	2	0.0E+00	2.0625000000000E+01	0.02
======= FILE DIETNAZA (NAZARETH DIET PROBLEM, UNIT 100 G) RUN 900315 =========									
DIETNAZA	6	3	16	0	1	3	1.8E-15	1.7470817120623E+02	0.03
==== FILE DIETMODI (NAZARETH DIET PROBLEM, UNIT 100 G, BOUNDS) RUN 900315 ====									
DIETMODI	6	3	16	0	1	2	1.8E-15	2.0500000000000E+02	0.02
======== FILE DIETNAZX (NAZARETH DIET PROBLEM, UNIT 1 G) RUN 900315 ==========									
DIETNAZX	6	3	16	0	1	3	0.0E+00	1.7470817120623E+02	0.04
===== FILE DIETMODX (NAZARETH DIET PROBLEM, UNIT 1 G, BOUNDS) RUN 900315 =====									
DIETMODX	6	3	16	0	2	2	0.0E+00	2.0500000000000E+02	0.05
========== FILE FROEBERG (EXAMPLES FROM FROEBERG) RUN 900315 ==========									
FROEP382	4	3	12	0	0	2	0.0E+00	-6.6000000000000E+01	0.06
FROEP384	4	3	12	2	0	2	2.2E-16	7.1666666666667E+00	0.02
FROEEX01	3	4	10	0	0	3	0.0E+00	-4.4000000000000E+00	0.02
FROEEX02	4	3	12	0	1	3	4.4E-16	-2.0000000000000E+00	0.03
FROEEX04	3	3	7	0	0	1	0.0E+00	-1.0000000000000E+00	0.04
FROEEX05	5	4	13	0	0	2	0.0E+00	0.0000000000000E+00	0.05
FROEEX06	3	3	8	0	0	0	0.0E+00	5.5833333333333E+00	0.02
FROEEX08	4	4	16	1	1	4	0.0E+00	3.0000000000000E+00	0.02
FROEEX09	4	4	15	0	0	2	0.0E+00	-5.3562500000000E+01	0.03
FROEEX10	4	3	11	0	0	2	0.0E+00	-2.8875000000000E+04	0.03
FROEEX11	4	2	8	0	1	3	0.0E+00	1.8000000000000E+01	0.02
FROEBOUN	3	2	6	0	0	1	0.0E+00	-2.0875000000000E+01	0.03
========= FILE LIQUORMX (SMALLCAS LIQUOR EXPL AS MAX) RUN 900315 ==========									

```
 LIQUORMX    7    5    15    0    2     4 0.0E+00  2.4500000000000E+02    0.04
======= FILE BEALECYC (BEALE'S EXPL FOR PROVOKING CYCLING) RUN 900315 ========
 BEALECYC    4    4    13    0    0     2 0.0E+00 -5.0000000000000E-02    0.04
================ FILE METALS (MPSX DEMO CASE) RUN 900315 ====================
 METALS      8    7    48    0    2     7 2.8E-14  2.9621660649819E+02    0.07
========== FILE EQSPARSE (RISOE SPARSE EQUATION SYSTEM) RUN 900315 ===========
 EQSPARSE  201  200   800  188    0   188 9.7E-13  0.0000000000000E+00    3.09
================ FILE WEBERDAT (WEBER'S EXAMPLES) RUN 900315 ================
 SW028064   28   63   373   14   25    49 2.1E-14  1.7297725316942E+08    0.37
 SW031084   31   83   519   14   18    60 2.1E-14  1.4772670909184E+08    0.52
 SW037099   37   98   652   17   23    70 2.8E-14  1.6805517101762E+08    0.68
 SW049129   49  128   918   23   62   147 3.6E-14  2.0663098647648E+08    1.58
 SW073189   73  188  1450   35  109   280 3.9E-13  2.7452894117283E+08    4.43
 SW120129  120  128  1024   74   61   207 1.4E-14  1.8312738480710E+08    2.59
 SW125167  125  166  1376   84  109   293 6.4E-11  2.8052050544962E+08    4.48
 SW129177  129  176  1468   88  132   382 3.8E-12  2.8162008879127E+08    6.23
 SW148217  148  216  1738   90  145   468 5.7E-14  1.9016667470627E+08    8.05
 SW202225  202  224  2026  126  105   319 7.1E-14  5.1990448656657E+08    6.38
 SW214231  214  230  2279  137  250   666 9.9E-14  6.9550906798347E+08   17.89
============ FILE WEBERDAT (WEBER'S EXAMPLES) RUN 900315 MSCALE=0 ============
 SW028064   28   63   373   14   16    42 1.9E-13  1.7297725316942E+08    0.31
 SW031084   31   83   519   14   15    49 2.1E-14  1.4772670909184E+08    0.55
 SW037099   37   98   652   17   20    58 3.9E-14  1.6805517101762E+08    0.65
 SW049129   49  128   918   23   37   109 7.1E-14  2.0663098647648E+08    1.54
 SW073189   73  188  1450   35   79   177 7.0E-11  2.7452894117283E+08    3.24
 SW120129  120  128  1024   74   43   148 2.7E-12  1.8312738480710E+08    1.78
 SW125167  125  166  1376   84   56   203 1.5E-13  2.8052050544962E+08    3.89
 SW129177  129  176  1468   88   62   217 1.4E-14  2.8162008879127E+08    4.03
 SW148217  148  216  1738   90   84   266 4.0E-11  1.9016667470626E+08    5.25
 SW202225  202  224  2026  126   74   285 1.4E-12  5.1990448656657E+08    5.86
 SW214231  214  230  2279  137   93   318 4.6E-14  6.9550906798347E+08    8.83
============ FILE MUNKFEAS (MUNKSGAARD'S EXAMPLES) RUN 900315 ================
 CRISECON   32   18    84    0    3     6 7.1E-15 -2.2875000000000E+02    0.17
 MULINRAT   60   42   206    0   14    27 2.4E-11 -2.7423201977225E+06    0.34
 M297ROWS  297  127   628    8  103   115 4.1E-12 -2.4259892314057E+06    2.70
 5750PTIM  575  267  1354   40  379   441 3.6E-12 -2.4259892314057E+06   14.94
 1020OPTI 1020  475  2846   11  601   645 8.4E-12 -2.4263973924262E+06   48.22
========= FILE MUNKFEAB (MUNKSGAARD'S EXPLS, UPPER BOUNDS) RUN 900315 ========
 CRISECOB   23   18    75    0    3     6 1.8E-15 -2.2875000000000E+02    0.13
 MULINRAB   37   42   183    0    8    20 1.8E-12 -2.7423201977225E+06    0.24
 M297ROWB  166  127   497    8   29    39 2.5E-10 -2.4259892314057E+06    1.17
 5750PTIB  442  267  1221   38  295   353 6.8E-12 -2.4259892314057E+06   10.22
 1020OPTB  788  475  2614   22  501   533 9.5E-12 -2.4263973924262E+06   32.44
====== FILE MUNKFEAL (MUNKSGAARD'S EXPLS, LOWER+UPPER BOUNDS) RUN 900315 =====
 CRISECOL   17   18    69    0    5     6 1.1E-14 -2.2875000000000E+02    0.14
 MULINRAL   23   42   169    0    9    16 3.6E-12 -2.7423201977225E+06    0.21
 M297ROWL   40  127   371    0   20    23 2.4E-10 -2.4259892314057E+06    0.64
 5750PTIL  315  267  1094   33  302   344 3.6E-12 -2.4259892314057E+06    8.83
 1020OPTL  562  475  2388    2  516   535 2.1E-11 -2.4263973924262E+06   28.81
============= FILE STANFORD (NETLIB COLLECTION 1-12) RUN 900315 =============
 AFIRO      28   32    88    0    4    11 5.8E-15 -4.6475314285714E+02    0.11
 ADLITTLE   57   97   465    4   11   109 2.8E-14  2.2549496316238E+05    1.00
 SHARE2B    97   79   730    0   82   130 3.2E-13 -4.1573224074142E+02    1.51
 SHARE1B   118  225  1182   71   93   315 7.4E-11 -7.6589318579186E+04    4.68
 BEACONFD  174  262  3476    0    0    32 7.6E-12  3.3592485807200E+04    2.80
 ISRAEL    175  142  2358    0    0   343 1.8E-11 -8.9664482186305E+05    9.96
 BRANDY    221  249  2150   70  102   299 1.7E-12  1.5185098964881E+03    7.69
 E226      224  282  2767    0   54   463 1.3E-13 -1.1638929066370E+01   13.04
 CAPRI     272  353  1786   82  290   459 3.3E-13  2.6900129137682E+03   10.97
 BANDM     306  472  2659    0  142   344 1.4E-13 -1.5862801845012E+02   12.89
 STAIR     357  467  3857   98  329   708 3.7E-13 -2.5126695119296E+02   48.41
 ETAMACRO  401  688  2489   17  550   969 4.8E-14 -7.5571523022120E+02   27.96
```

```
======= FILE STANFORD (NETLIB COLLECTION 1-12) RUN 900321 APOLLO DN10000 =====
 AFIRO       28   32    88    0    4    11 1.8E-14 -4.6475314285714E+02    0.10
 ADLITTLE    57   97   465    4   11   112 2.3E-13  2.2549496316238E+05    0.63
 SHARE2B     97   79   730    0   88   129 6.8E-13 -4.1573224074142E+02    0.93
 SHARE1B    118  225  1182   71   93   315 6.2E-10 -7.6589318579186E+04    2.71
 BEACONFD   174  262  3476    0    0    32 4.2E-11  3.3592485807200E+04    1.60
 ISRAEL     175  142  2358    0    0   361 6.9E-09 -8.9664482186305E+05    5.86
 BRANDY     221  249  2150   70  155   372 2.8E-10  1.5185098964881E+03    5.42
 E226       224  282  2767    0   54   459 5.0E-12 -1.1638929066370E+01    7.25
 CAPRI      272  353  1786   82  292   464 3.3E-12  2.6900129137682E+03    6.37
 BANDM      306  472  2659    0  141   344 1.3E-12 -1.5862801845012E+02    7.75
 STAIR      357  467  3857   98  325   700 8.5E-11 -2.5126695119296E+02   26.38
 ETAMACRO   401  688  2489   17  395   841 1.7E-13 -7.5571522988346E+02   14.85
========= FILE STANFORD (NETLIB COLLECTION 1-12) RUN 900322 UNISYS A6 =========
 AFIRO       28   32    88    0    4    11 1.7E-21 -4.6475314285714E+02    1.79
 ADLITTLE    57   97   465    4   11   109 3.4E-21  2.2549496316238E+05   27.20
 SHARE2B     97   79   730    0   82   128 1.9E-19 -4.1573224074142E+02   39.20
 SHARE1B    118  225  1182   71   93   315 2.6E-17 -7.6589318579186E+04  143.36
 BEACONFD   174  262  3476    0    0    32 2.8E-18  3.3592485807200E+04   52.52
 ISRAEL     175  142  2358    0    0   372 4.7E-17 -8.9664482186305E+05  285.81
 BRANDY     221  249  2150   70  103   301 9.5E-20  1.5185098964881E+03  230.12
 E226       224  282  2767    0   54   470 8.5E-21 -1.1638929066371E+01  366.95
 CAPRI      272  353  1786   82  315   494 4.0E-19  2.6900129137682E+03  328.35
 BANDM      306  472  2659    0  142   344 7.2E-20 -1.5862801845012E+02  389.81
 STAIR      357  467  3857   98  325   700 6.6E-18 -2.5126695119296E+02 1346.76
 ETAMACRO   401  688  2489   17  526   883 2.4E-20 -7.5571523022120E+02  753.21
========= FILE PILOT (NETLIB COLLECTION 13) RUN 900221 APOLLO DN10000 ========
 PILOTS A 1442 3652 43220    017003 33172 2.6E-10 -5.5748972928406E+0228786.08
============ FILE REID (NETLIB COLLECTION 14-17, REID) RUN 900320 ============
 25FV47     822 1571 11127    5 1032  4080 3.2E-12  5.5018458882867E+03  450.82
 CZPROB     930 3523 14173    0  865  1682 2.3E-13  2.1851966988566E+06  163.05
 FFFFF800   525  854  6235    0   39   307 3.6E-12  5.5567956481750E+05   19.51
 SHELL      537 1775  4900  533  103   857 0.0E+00  1.2088253460000E+09   24.90
========= FILE SCHRAGE (NETLIB COLLECTION 18,20, SCHRAGE) RUN 900320 =========
 GANGES    1310 1681  7021   12    0   436 7.5E-12 -1.0958573612928E+05   46.81
 SEBA       516 1028  4874    0   80   116 4.5E-13  1.5711600000000E+04   15.82
========== FILE BORE3D (NETLIB COLLECTION, FOURER 1F) RUN 900320 ===========
 BORE3D     234  315  1525    4   47    83 4.5E-10  1.3730803942085E+03    2.68
==== FILE 80BAU3B (NETLIB COLLECTION, FOURER 3F) RUN 900213 APOLLO DN10000 ===
 80BAU3B   2263 9799 29063    0  976  5729 2.6E-12  9.8722419240909E+05  963.62
========== FILE GFRD-PNC (NETLIB COLLECTION, FOURER 6F) RUN 900320 ==========
 GFRD-PNC   617 1092  3467    0  134   515 4.5E-13  6.9022359995488E+06   24.20
=== FILE GREENBEA (NETLIB COLLECTION, FOURER 7F) RUN 900222 APOLLO DN10000 ===
 GREENBEA  2393 5405 31499    2 4928 14816 6.6E-08 -7.2555238968172E+07 2905.31
=== FILE GREENBEB (NETLIB COLLECTION, FOURER 7BF) RUN 900223 APOLLO DN10000 ==
 GREENBEB  2393 5405 31499    2 3501  8666 1.5E-10 -4.3022602607851E+06 1742.55
========= FILE FOUSTAIR (NETLIB COLLECTION 8F-10F, FOURER) RUN 900320 ========
 GROW7      141  301  2633    0    0   195 7.3E-08 -4.7787811814712E+07    7.54
 GROW15     301  645  5665    0    0   561 7.5E-09 -1.0687094129358E+08   39.98
 GROW22     441  946  8318    0    0   917 3.5E-10 -1.6083433648256E+08   94.05
========== FILE PILOT4 (NETLIB COLLECTION, FOURER 11F) RUN 900209 ==========
 PILOT4     411 1000  5145    0  726  4152 3.0E-06 -2.5811375541301E+03  297.98
====== FILE PILOT4 (NETLIB COLLECTION, FOURER 11F) RUN 900308 MSCALE 2 =======
 PILOT4     411 1000  5145    0  243  2472 4.0E-07 -2.5811392612808E+03  180.41
====== FILE PILOT4 (NETLIB COLLECTION, FOURER 11F) RUN 900321 MSCALE 3 =======
 PILOT4     411 1000  5145    0  214  2006 4.0E-07 -2.5811392612808E+03  155.00
====== FILE PILOTJA (NETLIB COLLECTION, FOURER 12F) RUN 900319 MSCALE 3 ======
 PILOT.JA   941 1988 14706    0 2034 19104 1.5E-05 -6.1131371681635E+03 3230.29
========== FILE PILOTWE (NETLIB COLLECTION, FOURER 13F) RUN 900321 ==========
 PILOT.WE   723 2789  9218    0 1179 11373 2.1E-08 -2.7201073715840E+06 1641.90
====== FILE PILOTWE (NETLIB COLLECTION, FOURER 13F) RUN 900321 MSCALE 2=======
 PILOT.WE   723 2789  9218    0  813 10490 2.0E-05 -2.7201075794844E+06 1451.00
```

```
====== FILE PILOTWE (NETLIB COLLECTION, FOURER 13F) RUN 900321 MSCALE 3=======
 PILOT.WE  723 2789  9218    0  812 12071 2.0E-05 -2.7201076865223E+06 1585.62
======== FILE FOUR1421 (NETLIB COLLECTION 14F-21F, FOURER) RUN 900320 ========
 RECIPE     92  180   752    3    7    27 0.0E+00 -2.6661600000000E+02    0.69
 SC205     206  203   552    0    0   153 1.7E-13 -5.2202061211707E+01    3.07
 SCAGR25   472  500  2029    0  471   723 1.2E-12 -1.4753433060769E+07   26.64
 SCAGR7    130  140   553    0   45    94 4.4E-12 -2.3313898243310E+06    1.34
 SCFXM1    331  457  2612   11  142   298 1.1E-12  1.8416759028349E+04    9.19
 SCFXM2    661  914  5229   22  409   749 1.3E-12  3.6660261564999E+04   41.53
 SCFXM3    991 1371  7846   33  659  1204 5.7E-13  5.4901254549751E+04   98.65
 SCORPION  389  358  1708  168   39   272 3.1E-16  1.8781248227381E+03    5.72
======== FILE FOUR2225 (NETLIB COLLECTION 22F-25F, FOURER) RUN 900320 ========
 SCRS8     491 1169  4029    0   46   572 2.7E-13  9.0429695380079E+02   25.19
 SCSD1      78  760  3148    0   62   282 7.2E-16  8.6666666743334E+00    7.50
 SCSD6     148 1350  5666    0  127   490 1.5E-16  5.0500000077144E+01   20.00
 SCSD8     398 2750 11334    0  388  1235 4.4E-16  9.0499999992546E+02   95.45
======== FILE FOUR2628 (NETLIB COLLECTION 26F-28F, FOURER) RUN 900320 ========
 SCTAP1    301  480  2052    0  169   222 2.2E-15  1.4122500000000E+03    6.92
 SCTAP2   1091 1880  8124    0  222   570 8.9E-16  1.7248071428571E+03   53.50
 SCTAP3   1481 2480 10734    0  286   704 2.2E-16  1.4240000000000E+03   91.04
======== FILE FOUR2930 (NETLIB COLLECTION 29F-30F, FOURER) RUN 900321 ========
 SHIP04L   403 2118  8450    0  427   523 1.9E-15  1.7933245379704E+06   30.15
 SHIP04S   403 1458  5810    0  183   255 1.9E-15  1.7987147004454E+06   14.01
======== FILE FOUR3132 (NETLIB COLLECTION 31F-32F, FOURER) RUN 900321 ========
 SHIP08L   779 4283 17085    0  439   849 5.1E-15  1.9090552113891E+06  100.34
 SHIP08S   779 2387  9501    0  172   440 5.6E-15  1.9200982105346E+06   40.09
======== FILE FOUR3334 (NETLIB COLLECTION 33F-34F, FOURER) RUN 900321 ========
 SHIP12L  1152 5427 21597    0 1566  1863 3.2E-15  1.4701879193293E+06  256.97
 SHIP12S  1152 2763 10941    0  536   757 5.8E-15  1.4892361344061E+06   78.78
======== FILE FOUR3538 (NETLIB COLLECTION 35F-38F, FOURER) RUN 900321 ========
 SIERRA   1228 2036  9252  505  143   908 5.7E-14  1.5394362183632E+07   66.21
 STANDATA  360 1075  3038    0   44    57 3.0E-14  1.2576995000000E+03    4.76
 VTP.BASE  199  203   914    0  349   380 5.7E-12  1.2983146246136E+05    5.87
======== FILE FOUR4243 (NETLIB COLLECTION 42F-43F, FOURER) RUN 900321 ========
 NESM      663 2923 13988    0 1172  4998 3.9E-13  1.4076037620771E+07  487.72
 FORPLAN1  162  421  4916    0   62   177 4.0E-13 -6.6421896127220E+02    7.53
========= FILE PILOTNOV (NETLIB COLLECTION, FOURER 44F) RUN 900302 ==========
 PILOTS B  976 2172 13129    011329 11445 6.7E-10 -4.4972761882189E+03 2031.55
======= FILE PILOTNOV (NETLIB COLLECTION, FOURER 44F) RUN 900223 APOLLO ======
 PILOTS B  976 2172 13129    0 8808 12795 1.4E-05 -4.4972761882189E+03 1376.42
===== FILE PILOTNOV (NETLIB COLLECTION, FOURER 44F) RUN 900308 MSCALE 2 ======
 PILOTS B  976 2172 13129    0 6107  6577 1.5E-05 -4.4972761882189E+03  999.47
========= FILE STANDMPS (NETLIB COLLECTION, FOURER 46F) RUN 900321 ==========
 STANDMPS  468 1075  3686    0   96   178 4.7E-14  1.4060175000000E+03    9.61
========= FILE LEACHMAN (NETLIB COLLECTION 60-62, LEACHMAN) RUN 900321 =======
 AGG       489  163  2541    0   39    86 6.5E-11 -3.5991767286577E+07    4.95
 AGG2      517  302  4515    0   31   113 2.9E-11 -2.0239252355977E+07    8.40
 AGG3      517  302  4531    0   23   118 2.9E-11  1.0312115935089E+07    8.49
========= FILE GASSMANN (NETLIB COLLECTION 63-64, GASSMANN) RUN 900321 =======
 STOCFOR1  118  111   474    0    0    17 6.3E-13 -4.1131976219436E+04    0.53
 STOCFOR2 2158 2031  9492    0    0   991 1.2E-12 -3.9024408537882E+04  162.32
========== FILE BELGEFOM (EFOM BELGIAN SYSTEM BY FENHANN) RUN 900213 =========
 BGTESTRD   68   97   336    0   57    84 1.1E-11  1.6762180373051E+10    0.72
 BGTEST    121  149   463    0   37    83 1.8E-12  1.6762259183117E+10    0.90
 ELECOORD  894 1290  5178    0  378   930 4.1E-12  2.5927333441134E+11   52.27
========= FILE FENHANRD (EFOM TEST BY FENHANN WITH REDUCE) RUN 900213 ========
 FENHRD78  164  278   965    0   58   143 1.8E-12  1.6681370450240E+10    2.09
 FENHRD81  342  556  2043    0  121   326 1.1E-12  9.6056755229249E+10    8.16
 FENHRD85  536  854  3268    0  160   506 9.1E-13  1.4350551377207E+11   19.04
 FENHRD90  732 1158  4592    0  186   742 1.1E-12  1.8317183787769E+11   36.66
 FENHRD10  925 1463  5786    0  289  1066 1.6E-12  2.3168128725701E+11   65.08
======= FILE FENHANNR (EFOM TEST BY FENHANN WITHOUT REDUCE) RUN 900213 =======
```

```
 FENHNR78  628  722  1908    0   89   171 2.7E-12  1.6681241473300E+10    7.39
 FENHNR81 1256 1444  3903    0  180   407 1.4E-12  9.6056267897658E+10   32.77
 FENHNR85 1881 2166  5971    0  230   615 1.8E-12  1.4350388959082E+11   74.51
 FENHNR90 2505 2888  8122    0  302   963 2.3E-12  1.8316997907451E+11  155.04
 FENHNR10 3125 3610 10134    0  397  1111 3.7E-12  2.3167866675635E+11  241.05
============ FILE GRAV2877 (EFOM CASE BY GRAVGAARD) RUN 900209 ===============
 GRAV2877 2877 3274 18680   84  368   898 1.4E-14  6.4694205935941E+11  192.99
======== FILE BIGMAT  (EFOM CASE BY GRAVGAARD, DO NOTHING) RUN 900220 ========
 ON0000 A 4211 4923 26778   87  528  2067 7.9E-13  6.4697464199287E+11  689.26
======= FILE TOTALMAT (EFOM CASE BY GRAVGAARD, DO NOTHING) RUN 900219 ========
 ON0000 B 5140 5985 31714   86  667  2448 3.4E-13  6.4801085423833E+11 1065.06
======= FILE GROHES9 (EFOM CASE BY GROHNHEIT, 90% SO2 RED) RUN 900213 ========
 EFFS00 A 5645 5962 36521  126 2142  5132 2.3E-13  8.0652524279469E+08 2623.37
======== FILE GROHLEG (EFOM CASE BY GROHNHEIT, LEGAL CASE) RUN 900213 ========
 LEG000 B 4657 4619 26132   19 1805  4621 1.7E-13  8.3323179739657E+08 1785.77
========= FILE GROHCEC (EFOM CASE BY GROHNHEIT, CEC CASE) RUN 900209 =========
 CEC000   5340 5690 34263  120 1646  4020 3.2E-13  7.9953386244257E+08 1852.12
======== FILE GROHDAT (EFOM CASE BY GROHNHEIT, DO NOTHING) RUN 900209 ========
 ON0000 C 5280 6024 32654   85 1171  5477 2.0E-11  2.1413651189930E+09 2530.64
==== FILE GROHEFF (EFOM CASE BY GROHNHEIT, EFFICIENCY, 35% SO2) RUN 900209 ===
 EFF000   5645 5962 36521  126 2033  6417 9.7E-10  8.0697728098242E+08 3280.09
======= FILE GROHEN3 (EFOM CASE BY GROHNHEIT, 30% NOX RED) RUN 900209 ========
 EFFN00 A 5645 5962 36521  126 2103  6269 5.8E-10  8.0397982342634E+08 3219.41
======= FILE GROHES3 (EFOM CASE BY GROHNHEIT, 30% SO2 RED) RUN 900212 ========
 EFFS00 B 5645 5962 36521  126 1771  4307 3.1E-13  7.9787362407735E+08 2194.69
======== FILE GROHOLD (EFOM CASE BY GROHNHEIT, LEGAL CASE) RUN 900321 ========
 LEG000 A 5361 5836 33571   97 1265  3774 1.1E-12  6.4090368474316E+08 1799.71
===== FILE GROHOLD (EFOM CASE BY GROHNHEIT, LEGAL CASE) RUN 900321 APOLLO ====
 LEG000 A 5361 5836 33571   97 1371  4111 3.5E-12  6.4090368474316E+08 1337.27
======== FILE GROHLEGA (EFOM CASE BY GROHNHEIT, LEGAL CASE) RUN 900212 ========
 LEG000 C 5319 5650 33942  120 1711  4389 2.3E-13  8.0491919521062E+08 2002.22
==== FILE GROHDEXT (EFOM CASES BY GROHNHEIT, ELECTRICITY ONLY) RUN 900220 ====
 DEXT00 A 1266 1105  6175    0   73   218 8.9E-16  7.5377729837411E+06   23.40
 DEXT00 B 2004 2271 14303   96   75   436 8.9E-16  7.5377729837411E+06   72.87
 DEXT00 C 2004 2271 14555   96   75   730 1.8E-15  4.6240997061265E+07  106.82
 DEXT00 D 2004 2271 14779   96   75   528 1.9E-15  5.5265338937865E+07   83.73
======= FILE GROHDLEG (EFOM CASE BY GROHNHEIT, LEGAL CASE) RUN 900214 ========
 DLEG00   5319 5610 33862  120 1730  4660 8.0E-13  8.0767014936027E+08 2159.74
======= FILE GROHS70 (EFOM CASE BY GROHNHEIT, 70% SO2 RED) RUN 900214 ========
 S70000   5645 5962 36521  126 1826  4902 2.3E-13  8.0166821337604E+08 2502.40
======= FILE GROHN25 (EFOM CASE BY GROHNHEIT, 25% NOX RED) RUN 900214 ========
 EFFN00 B 5645 5962 36521  126 2148  5973 2.3E-13  8.0217524599260E+08 3186.16
======= FILE GROHN35 (EFOM CASE BY GROHNHEIT, 35% NOX RED) RUN 900214 ========
 EFFN00 C 5645 5962 36521  126 2066  6620 1.8E-09  8.0665957598331E+08 3437.88
====== FILE GROHR25 (EFOM CASE BY GROHNHEIT, 25% SO2+NOX RED) RUN 900321 =====
 EFFN00 D 5645 5962 36521  126 2155  6211 5.7E-13  8.0217888758760E+08 3318.74
== FILE GROHR25 (EFOM CASE BY GROHNHEIT, 25% SO2+NOX RED) RUN 900223 APOLLO ==
 EFFN00 D 5645 5962 36521  126 2201  6355 2.1E-12  8.0217888758760E+08 2414.75
====== FILE GROHR35 (EFOM CASE BY GROHNHEIT, 35% SO2+NOX RED) RUN 900214 =====
 R35000   5645 5962 36521  126 2062  5773 4.4E-07  8.0741081695290E+08 2959.63
======== FILE GROHREF (EFOM CASE BY GROHNHEIT, DO NOTHING) RUN 900214 ========
 REF000   4355 4233 23090   24 1447  3626 3.5E-13  7.9745041720658E+08 1263.05
```

9.3 Comments on the test problems

If nothing else is stated in the interspersed "headlines" of the list, the problems were solved on the VAX-8700 installation at Risø's Computer Section, with default settings of LINPROG's parameters and options.

Some of the computations were made on an Apollo DN10000 machine. The twelve Stanford cases from NETLIB (# 1 – 12) were run on both VAX-8700 and DN10000, and it appears from

our time measurements that DN10000 is almost twice as fast as VAX-8700. The Stanford cases were also run on a UNISYS A6 machine, in which the inherent (double) precision corresponds to about 22 decimal digits. (We have also made successful tests on a SUN 3/75 workstation, but the results are not given in the list.)

To illustrate the influence of scaling, a few computations were made with values of MSCALE other than the default value 1.

Most test cases were solved with the current version of LINPROG. Some, however, were solved with slightly older versions. Dates for running are given in the headlines. (In some of the older runs the iteration counts for Phase 1 might be misleadingly high, because they refer to the latest "repair" of infeasibility (Section 8.5).)

By now, we have solved all the NETLIB cases. However, the two problems PILOT (NETLIB # 13) and PILOTJA (NETLIB # 12f) were quite troublesome and for a long time they resisted our efforts to make LINPROG solve them. PILOT could be solved by standard settings on the Apollo DN10000, but required a lot of computer time.

When we tried to solve PILOTJA with the default option MSCALE = 1, LINPROG could not locate the optimum: new infeasibilities showed up continually in Phase 2, and eventually the code went into cycling. We have not yet been able to alleviate this problem, but we have noticed that PILOTJA has caused similar troubles for others (Lustig, [28]). With MSCALE = 3 it was possible to get a solution on VAX-8700, but still not on DN10000, even after use of coarser tolerance settings (which resulted in a singular basis matrix.)

It is apparent, that a high working precision on the computer is necessary to overcome the ill-conditioning of PILOJA. In fact we did never succeed in solving it on DN10000, which has slightly less precision than VAX-8700 (with "D-floating"). On the other hand, when we tried PILOTJA with a special program version LINPROG4 working in quadruple (16 bytes) precision with sharpened tolerances, we could obtain perfect solutions for all the scaling modes MSCALE = 0, 1, 2, and 3, provided we restarted from an optimal dump produced by standard LINPROG with MSCALE = 3. In all four cases we found the optimum

$$z_{\text{opt}} = -6.1131364655813 \times 10^3,$$

and we therefore consider this figure as the correct "mathematical" answer with the given digits.

Unlike the situation for the PILOT cases, use of MSCALE = 3 had an adverse effect on LINPROG's performance on Grohnheit's cases.

All the test problems in this list have feasible solutions, because we decided not to report LINPROG performance on infeasible cases. In this connection let us mention, that we have met LP problems which come very close to the borderline between feasibility and infeasibility. Such problems are troublesome to handle for any LP solver. A particularly difficult example of this kind was an EFOM case (GRAV2676) with $m = 2676$, where LINPROG was virtually unable to decide feasibility within the usual 8 bytes working precision. Only by using the 16-byte version LINPROG4 mentioned above, could we, beyond any reasonable doubt, deem the problem to be mathematically infeasible. Sometimes the user wants the LP solver to treat such borderline cases as if they were feasible, so we may speak of "numerical feasibility". The tool for obtaining this in LINPROG would be to try to modify one or more of the tolerance parameters in the Control File (Section 2.1).

10 Summary and Conclusions

Our goal was to write a FORTRAN-based LP code which was able to produce efficient and reliable solutions to most of the LP problems encountered at Risø. Priority was given to a simple structure and to robustness, rather than to sophisticated LP extensions. We considered it to be important that the code be portable across several computer systems.

LINPROG uses the simplex method. It is well-known that simplex faces increasing competition from a class of projection methods, called interior-point methods. Karmarkar's algorithm

[29], being of this type, has attracted much attention. It was claimed that its efficiency would outperform simplex for big problems by quite large factors. However, in a comparative study, Tomlin [30] has pointed out that there are still a lot of practical LP problems for which simplex compares favorably with Karmarkar's method.

Acknowledgements

As mentioned in Section 9.1, the LINPROG project has benefited from our co-operation with several groups and individuals at Risø. In this connection we wish to thank Steen Weber, Jørgen Fenhann, Jesper Munksgaard Pedersen, Ole Gravgård Pedersen, and Poul Erik Grohnheit, who all contributed interesting test problems. Being the result of many types of computer modelling, these test cases show a good deal of diversity. They have been very valuable for us as a debugging tool for the code, and for improving its robustness and numerical stability. They have also guided us in selecting new features and enhancements for implementation. We want in particular to acknowledge P. E. Grohnheit, who not ouly provided the largest of all the test problems, but also made a valuable contribution in promoting the use of LINPROG.

Moreover, we thank the System Manager for VAX-8700 at Risø, Anne Margrethe Larsen, who helped us with setting up the proper VAX-environment for LINPROG. She also assisted with the use of the TEX and LATEX typesetting software, by trouble shooting and by making useful auxiliary software components available, related to TEX. Also in the Computer Section, Bjarne Wallin has guided us in making LINPROG operable at the Apollo DN10000 computer.

Outside Risø we are indebted to Morten Norby Larsen from the Technical University in Lyngby for his professional help during the installation and test of LINPROG at the SUN workstation at his laboratory. We also thank Claude Thonet, who works for the European Communities in Brussels, for his assistance in making comparative test runs with other LP codes.

References

[1] J. L. Nazareth, *Computer Solution of Linear Programs.* New York and Oxford: Oxford University Press, 1987.

[2] *IBM Mathematical Programming System Extended/370 (MPSX/370), Primer.*, 1979. GH19-1091-1, File No. S370-82.

[3] J. J. Dongarra and E. Grosse, "Distribution of mathematical software via electronic mail," *Comm. ACM*, vol. 30, pp. 403–407, 1987.

[4] B. A. Murtagh and M. A. Saunders, "MINOS 5.1 USER'S GUIDE," Tech. Rep. SOL 83-20R, Systems Optimization Laboratory, Stanford University, California 94305-4022, 1987.

[5] G. B. Dantzig, *Linear Programming and Extensions.* New Jersey: Princeton University Press, 1963.

[6] B. A. Murtagh, *Advanced Linear Programming: Computation and Practice.* New York: McGraw-Hill, 1981.

[7] G. H. Golub and C. F. van Loan, *Matrix Computations.* Baltimore, Maryland: The John Hopkins University Press, 1984.

[8] G. E. Forsythe and C. B. Moler, *Computer Solution of Linear Algebraic Systems.* Englewood Cliffs, N.J.: Prentice-Hall, 1967.

[9] W. Orchard-Hays, *Advanced Linear-Programming Computing Techniques.* New York: McGraw-Hill, 1968.

[10] M. Benichou, J. M. Gauthier, G. Hentges, and G. Ribiere, "The efficient solution of large-scale linear programming problems — some algorithmic techniques and computational results," *Mathematical Programming*, vol. 13, pp. 280–322, 1977.

[11] I. S. Duff and J. K. Reid, "An implementation of Tarjan's algorithm for the block triangularization of a matrix," *ACM Transactions on Mathematical Software*, vol. 4, pp. 137–147, 1978.

[12] I. S. Duff, "On algorithms for obtaining a maximum transversal," *ACM Transactions on Mathematical Software*, vol. 7, pp. 315–330, 1981.

[13] E. Hellerman and D. C. Rarick, "The partitioned preassigned pivot procedure (P^4)," in *Sparse matrices and their applications, Proceedings of Conference in Yorktown Heights, September 9-10, 1971* (D. J. Rose and R. A. Willoughby, eds.), pp. 67–76, Plenum Press, 1972.

[14] J. A. Tomlin, "Modifying triangular factors of the basis in the simplex method," in *Sparse matrices and their applications, Proceedings of Conference in Yorktown Heights, September 9-10, 1971* (D. J. Rose and R. A. Willoughby, eds.), pp. 77–85, Plenum Press, 1972.

[15] J. J. H. Forrest and J. A. Tomlin, "Updated triangular factors of the basis to maintain sparsity in the product form simplex method," *Mathematical Programming*, vol. 2, pp. 263–278, 1972.

[16] R. H. Bartels, J. Stoer, and C. Zenger, "A Realization of the Simplex Method based on Triangular Decompositions," in *Handbook for Automatic Computation - Linear Algebra* (J. H. Wilkinson and C. Reinsch, eds.), pp. 152–190, Springer-Verlag, 1971.

[17] P. M. J. Harris, "Pivot selection methods of the Devex LP code," *Mathematical Programming*, vol. 5, pp. 1–28, 1973.

[18] D. Goldfarb and J. K. Reid, "A practicable steepest-edge simplex algorithm," *Mathematical Programming*, vol. 12, pp. 361–371, 1977.

[19] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming.* New York: John Wiley and Sons, 1972.

[20] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, "A practical anti-cycling procedure for linear and nonlinear programming," Tech. Rep. SOL 88-4, Systems Optimization Laboratory, Stanford University, California 94305-4022, 1988.

[21] J. A. Tomlin, "On scaling linear programming problems," *Mathematical Programming Study*, vol. 4, pp. 146–166, 1975.

[22] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices.* Oxford: Clarendon Press, 1986.

[23] M. H. van Emden, "Increasing the efficiency of quicksort," *Comm. ACM*, vol. 13, pp. 563–567, 1970.

[24] C.-E. Fröberg, *Lärobok i numerisk analys* (In Sweedish). Copenhagen and Stockholm: Scandinavian University Books, Svenska Bokförlaget/Bonniers, 1962.

[25] C. F. Cohn, *Algebra, Vol. 1.* Bedford: Bedford College, 1981.

[26] J. M. Pedersen, "LINRAT - en energirationeringsmodel for Danmark," Tech. Rep. Risø-M-2611 (in Danish), Risø National Laboratory, DK-4000 Roskilde, Denmark, 1986.

[27] E. van der Voort, E. Donni, C. Thonet, E. B. d'Enghien, C. Dechamps, and J. F. Guilmot, *ENERGY SUPPLY Modelling Package EFOM-12 C Mark I - Mathematical description.* Louvain-la-Neuve, Belgium: CABAY (for the Commision of the European Communities), 1984.

[28] I. J. Lustig, "An analysis of an available set of linear programming test problems," Tech. Rep. SOL 87-11, Systems Optimization Laboratory, Stanford University, California 94305-4022, 1987.

[29] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, pp. 373–395, 1984.

[30] J. A. Tomlin, "A Note on Comparing Simplex and Interior Methods for Linear Programming," in *Progress in Mathematical Programming. Interior-Point and Related Methods* (N. Megiddo, ed.), pp. 91–103, Springer-Verlag, 1987.

APPENDIX

A1: Report writers in FORTRAN and PASCAL

```
      PROGRAM REPORT
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C
C THIS PROGRAM READS THE LP SOLUTION FROM THE BINARY COMMUNICATION
C FILE PRODUCED BY LINPROG, AND PRODUCES A CORRESPONDING FORMATTED
C SOLUTION FILE. THIS FILE WILL CONTAIN KEY DATA FOR THE PROBLEM
C AND A TABULATION OF THE ROW AND COLUMN SECTIONS OF THE SOLUTION.
C MOREOVER, THE PROGRAM STORES ROW AND COLUMN NAMES AND ACTIVITIES,
C DUAL ACTIVITIES AND REDUCED COSTS, IN ARRAYS FOR LATER REFERENCE,
C IF YOU WANT A PARAGON FOR A FULL-FLEDGED REPORT WRITER
C
      PARAMETER (MR=8191, MC=9799)
      DIMENSION ACTIVR(MR), DUALAC(MR), ACTIVC(MC), REDUCO(MC)
C THESE FOUR ARRAYS WILL HOLD THE FOLLOWING INFORMATION:
C     ACTIVR(I)  =  ACTIVITY OF ROW NO. I
C     DUALAC(I)  =  DUAL ACTIVITY OF ROW NO. I
C     ACTIVC(J)  =  ACTIVITY OF COLUMN NO. J
C     REDUCO(J)  =  REDUCED COST FOR COLUMN NO. J
C
      CHARACTER ROWNAM(MR)*8, COLNAM(MC)*8
C ROWNAM(I) WILL CONTAIN THE 8-CHARACTER-NAME OF ROW NO. I
C COLNAM(J) WILL CONTAIN THE 8-CHARACTER-NAME OF COLUMN NO. J
      CHARACTER YYMMDD*6, VERS*4, TARGET*8, PBSTAT, MARK, STATUS*10
     1, SETNAM*8, OBJNAM*8, RHSNAM*8, RNGNAM*8, BDSNAM*8, STATUR*2
     2, STATUC*2
C OPEN FILES
      OPEN (UNIT=1, STATUS='OLD', ACCESS='SEQUENTIAL'
     1, FORM='UNFORMATTED')
      REWIND 1
      OPEN (UNIT=2, STATUS='NEW', ACCESS='SEQUENTIAL', FORM='FORMATTED')
      REWIND 2
C READ AND WRITE IDENTIFICATION SECTION
  100 READ (1,END=900) YYMMDD, VERS, TARGET
      WRITE (2,101) YYMMDD, VERS, TARGET
  101 FORMAT (//' PROBLEM DATE', T32, A/' LINPROG VERSION', T32, A/
     1' TARGET', T32, A/)
      READ (1) SETNAM, OBJNAM, RHSNAM, RNGNAM, BDSNAM
      WRITE (2,102) SETNAM, OBJNAM, RHSNAM, RNGNAM, BDSNAM
  102 FORMAT (' NAME OF DATA SET', T32, A/' NAME OF OBJECTIVE ROW', T32
     1, A/' NAME OF RIGHT-HAND SIDE', T32, A/' NAME OF RANGES', T32, A/
     2, ' NAME OF BOUNDS', T32, A/)
      READ (1) PBSTAT, L, NS, ICNTR, FUNVAL
      STATUS = 'INFEASIBLE'
      IF (PBSTAT .EQ. '0') STATUS = 'OPTIMAL'
      WRITE (2,103) STATUS, L, NS, ICNTR, FUNVAL
  103 FORMAT (' PROBLEM STATUS', T32, A/' NUMBER OF ROWS', T32, I4/
     1' NUMBER OF COLUMNS', T32, I4/' NUMBER OF ITERATIONS', T32, I4/
     2' OBJECTIVE VALUE', T32, 1PE20.13)
C READ AND WRITE ROW SECTION OF SOLUTION
      WRITE (2,104)
  104 FORMAT (/' TABULATION OF THE FILED ROW SECTION'/T3, 'NAME', T11
     1, 'NUMBER', T18, 'STATUS', T27, 'ACTIVITY', T43, 'SLACK', T54
     2, 'DUAL ACTIVITY', T70, 'MARK')
      DO 1, I = 1, L
      READ (1) MARK, ROWNAM(I), STATUR, ACTIVR(I), SLACAC, DUALAC(I)
    1 WRITE (2,105) ROWNAM(I), I, STATUR, ACTIVR(I), SLACAC, DUALAC(I)
     1, MARK
  105 FORMAT (T2, A8, T12, I4, T20, A2, T25, 1PE13.6, T39, 1PE13.6, T54
     1, 1PE13.6, T71, A1)
C READ AND WRITE COLUMN SECTION OF THE SOLUTION
      WRITE (2,106)
  106 FORMAT (/' TABULATION OF THE FILED COLUMN SECTION'/T3, 'NAME', T11
     1, 'NUMBER', T18, 'STATUS', T27, 'ACTIVITY', T41, 'INPUT COST', T55
     2, 'REDUCED COST', T70, 'MARK')
      DO 2, J = 1, NS
```

```
      READ (1) MARK, COLNAM(J), STATUC, ACTIVC(J), COST, REDUCO(J)
    2 WRITE (2,105) COLNAM(J), J+L, STATUC, ACTIVC(J), COST, REDUCO(J)
     1, MARK
C NOW YOU MAY PROCEED WITH YOUR REPORT WRITER, USING THE SIX ARRAYS
C ACTIVR(), DUALAC(), ACTIVC(), REDUCO(), ROWNAM(), AND COLNAM():
*           ..........................
*           ..........................
*           ..........................
C GO BACK AND READ MORE SOLUTION SETS, IF ANY
      GO TO 100
C CLOSE COMMUNICATION FILE
  900 CLOSE (1)
C
      END


PROGRAM REPORT(F1, F2, OUTPUT);
(*
                    PASCAL program REPORT.
  This program reads the LP solution from the binary communication
  file produced by LINPROG, and produces a corresponding formatted
  solution file. This file will contain key data for the problem
  and a tabulation of the row and column sections of the solution.
  Moreover, the program stores row and column names and activities,
  dual activities and reduced costs, in arrays for later reference,
  if you want a paragon for a full-fledged report writer.
*)
CONST
  MR=8191;
  MC=9799;
TYPE
  INPSTR = PACKED ARRAY[1..47] OF CHAR;
  C1     = PACKED ARRAY[1..1] OF CHAR;
  C2     = PACKED ARRAY[1..2] OF CHAR;
  C4     = PACKED ARRAY[1..4] OF CHAR;
  C6     = PACKED ARRAY[1..6] OF CHAR;
  C8     = PACKED ARRAY[1..8] OF CHAR;
  C10    = PACKED ARRAY[1..10] OF CHAR;
  A1 =ARRAY[0..MR] OF DOUBLE;
  A2 =ARRAY[0..MC] OF DOUBLE;
  A3 =ARRAY[0..MR] OF C8;
  A4 =ARRAY[0..MC] OF C8;
  REC = RECORD
          CASE I:INTEGER OF
           1: (S:INPSTR);
           2: (DUMMY2:C2;
               YYMMDD:C6;
               VERS  :C4;
               TARGET:C8);
           3: (DUMMY3:C2;
               SETNAM,OBJNAM,RHSNAM,RNGNAM,BDSNAM:C8);
           4: (DUMMY4:C2;
               PBSTAT:C1;
               L,MS,ICNTR:INTEGER;
               FUNVAL:DOUBLE);
           5: (DUMMY5:C2;
               MARK:C1;
               NAME:C8;
               STAT:C2;
               ACTI,COST,REDU:DOUBLE);
        END(*OF RECORD*);
VAR
  F1 :FILE OF INPSTR;
  F2 :TEXT;
  R  :REC;
  (*
    These four arrays will hold the following information:
        ACTIVR[I]  =  activity of row no. I
        DUALAC[I]  =  dual activity of row no. I
        ACTIVC[J]  =  activity of column no. J
        REDUCO[J]  =  reduced cost for column no. J
  *)
```

```
  ACTIVR, DUALAC : A1;
  ACTIVC, REDUCO : A2;

  I, J, L, NS, ICNTR  : INTEGER;
  FUNVAL, SLACAC, COST: DOUBLE;

  (*
    ROWNAM[I] will contain the 8-character-name of row no. I
    COLNAM[J] will contain the 8-character-name of column no. I
  *)
  ROWNAM              : A3;
  COLNAM              : A4;

  PBSTAT, MARK                                    :C1;
  DUMMY, STATUR, STATUC                           :C2;
  VERS                                            :C4;
  YYMMDD                                          :C6;
  TARGET, SETNAM, OBJNAM, RHSNAM, RNGNAM, BDSNAM:C8;
  STATUS                                          :C10;
BEGIN
        (* Open and reset files F1 and F2 *)
  OPEN(FILE_VARIABLE:=F1,FILE_NAME:='LIBINO.DAT',HISTORY:=READONLY,
            ERROR:=MESSAGE);
  RESET(F1);
  OPEN(FILE_VARIABLE:=F2,FILE_NAME:='REPORT.OUT',HISTORY:=NEW,
            ERROR:=MESSAGE);
  REWRITE(F2);
         (* Read and write identification section *)
  WHILE NOT EOF(F1) DO
  BEGIN
    R.S:=F1^; GET(F1); (* Read a packed array from file F1 *)
    WRITELN(F2); WRITELN(F2);
    WITH R DO
    BEGIN
      WRITELN(F2,' PROBLEM DATE                ',YYMMDD:6);
      WRITELN(F2,' LINPROG VERSION             ',VERS:4);
      WRITELN(F2,' TARGET                      ',TARGET:8);
    END;
    WRITELN(F2);

    R.S:=F1^; GET(F1); (* Read a packed array from file F1 *)
    WITH R DO
    BEGIN
      WRITELN(F2,' NAME OF DATA SET            ',SETNAM:8);
      WRITELN(F2,' NAME OF OBJECTIVE ROW       ',OBJNAM:8);
      WRITELN(F2,' NAME OF RIGHT-HAND SIDE     ',RHSNAM:8);
      WRITELN(F2,' NAME OF RANGES              ',RNGNAM:8);
      WRITELN(F2,' NAME OF BOUNDS              ',BDSNAM:8);
    END;
    WRITELN(F2);

    R.S:=F1^; GET(F1); (* Read a packed array from file F1 *)
    IF R.PBSTAT = 'O' THEN STATUS:='OPTIMAL'
                      ELSE STATUS:='INFEASIBLE';
    L:=R.L; NS:=R.NS;
    WITH R DO
    BEGIN
      WRITELN(F2,' PROBLEM STATUS              ',STATUS:10);
      WRITELN(F2,' NUMBER OF ROWS              ',L:4);
      WRITELN(F2,' NUMBER OF COLUMNS           ',NS:4);
      WRITELN(F2,' NUMBER OF ITERATIONS        ',ICNTR:4);
      WRITELN(F2,' OBJECTIVE VALUE             ',FUNVAL:20);
    END;
    WRITELN(F2);
          (* Read and write row section of solution *)
    WRITELN(F2,' TABULATION OF THE FILED ROW SECTION');
    WRITELN(F2,'  NAME    NUMBER STATUS   ACTIVITY        ',
              'SLACK       DUAL ACTIVITY   MARK');
    FOR I:=1 TO L DO
    BEGIN
      R.S:=F1^; GET(F1); (* Read a packed array from file F1 *)
      ROWNAM[I]:=R.NAME; STATUR:=R.STAT; ACTIVR[I]:=R.ACTI;
```

```
      SLACAC:=R.COST; DUALAC[I]:=R.REDU;
      WRITELN(F2,' ',R.NAME:8,I:6, R.STAT:6,
                 '   ',R.ACTI:13,' ',R.COST:13,'  ',R.REDU:13,
                 '    ',R.MARK:1);
    END(*for I loop*);
    WRITELN(F2);
          (* Read and write column section of the solution *)
    WRITELN(F2,' TABULATION OF THE FILED COLUMN SECTION');
    WRITELN(F2,'  NAME    NUMBER STATUS   ACTIVITY      ',
              'INPUT COST    REDUCED COST   MARK');
    FOR J:=1 TO NS DO
    BEGIN
      R.S:=F1^; GET(F1); (* Read a packed array from file F1 *)
      COLNAM[J]:=R.NAME; STATUC:=R.STAT; ACTIVC[J]:=R.ACTI;
      REDUCO[J]:=R.REDU;
      WRITELN(F2,' ',R.NAME:8,(J+L):6, R.STAT:6,
                 '   ',R.ACTI:13,' ',R.COST:13,'  ',R.REDU:13,
                 '    ',R.MARK:1);
    END(*for J loop*);
    (*
      Now you may proceed with your report writer, using the six arrays
      ACTIVR[], DUALAC[], ACTIVC[], REDUCO[], ROWNAM[], AND COLNAM[]:
                 .........................
                 .........................
                 .........................
      Go back and read more solution sets, if any.
    *)
  END(*while loop*);
  CLOSE(F1); (* Close communication file *)
END.
```

A2: Algorithmic description of LINPROG

Below we summarize the total LINPROG algorithm in annotated form.

- Initiate another LP. Compile keywords from Control File.
- Read first part of Matrix File, including row section.
- Read column section from Matrix File and build lists for constraint matrix. Sort-merge row names for each column.
- Read RHS section from Matrix File.
- Read RANGES section (if any) from Matrix File.
- Read BOUNDS section (if any) from Matrix File.
- Adjust constraint matrix and initialize eta lists.
- Print LP matrix statistics.
- If PICTURE was specified, then print matrix picture.
- Set indicators and upper bounds for slacks.
- Compute effective upper bounds (Section 8.1) and mark zero-range variables fixed.
- If RESTART was specified, then read restart file.
- Set up initial triangular basis (CRASH procedure, Section 8.2).
- Initialize arrays of nonbasic pointers and bound indicators for variables available for pivoting.
- Initialize arrays of nonbasic pointers and bound indicators for variables unavailable for pivoting.
- Make an initial inversion (Section 6).

- Try to clear basis for zero slacks (Phase 0, Section 8.2).
- If LOGFRQ < 0 was specified, then print a map of initial basic vectors.
- Execute Phase 1 of simplex.
- Set up cost vector c for Phase 2 and make a BTRAN.
- Execute Phase 2 of simplex.
- Print optimal value.
- If DUMP was specified, then produce a dump file.
- Construct column section output.
- Construct row section output.
- If SOLUTION was specified, then print row section output.
- If SOLUTION was specified, then print column section output.
- If BINOUT was specified, then produce a binary solution file.
- Check solution for feasibility.
- Produce a summary output line.
- Return to top to see if more problems should be solved.

After this we give more details for the simplex part of the LINPROG algorithm. Unless explicitly stated, they are valid for both Phase 1 and 2:

- (Phase 1 only) Check if current solution is feasible. If not, set up the infeasibility cost function (Section 8.3).
- If iteration count since last re-inversion equals MITRE, then make another inversion (Section 6).
- (Phase 2 only) After an inversion, check the feasibility. If infeasibilities are detected, then transfer control back to Phase 1 ("Repair" procedure, Section 8.5).
- Make an extended BTRAN step (Forrest-Tomlin procedure, Section 7.4).
- Compute reduced costs.
- Select entering variable.
- (Phase 1 only) If no entering variable could be found, then the problem is infeasible.
- (Phase 2 only) If no entering variable could be found, then optimality is established. Check feasibility.
- If an entering variable was found, then proceed to compute the updated pivot column by an FTRAN step.
- Select leaving variable by the CHUZR procedure and see if entering candidate goes to its opposite bound. (Sections 8.1 and 8.3).
- Update the transformed rhs vector β.
- Update indicators.
- Simplex iteration step finished. Increase iteration counter, and print iteration results if requested by LOGFRQ. Return to top of simplex.

A3: Subroutines in LINPROG

In the following we list, in tabular form, the FORTRAN subprograms making up LINPROG itself and the machine-dependent set of subroutines LPAUX, together with a short description of their tasks. Apart from the main program, the lists are in alphabetic order.

Name	Task
MAIN	LINPROG driver program.
BNDINP	Reads BOUNDS Section input.
BTRAN	Makes extended BTRAN with concurrent Forrest-Tomlin updating.
CHECK	Checks the solution for feasibility.
CHUZC	Selects entering variable in Phase 1 or 2.
CHUZR	Selects leaving variable in Phase I or 2.
COLINP	Reads COLUMNS Section input and generates input matrix lists..
COLLAT	Computes integer equivalents of the two halves of an 8-byte character word.
COLOUT	Produces column section output.
COLPRT	Prints column section output.
COMPIL	Compiles keywords in Command File for one problem.
CRASH	Selects an initial triangular basis.
EFFLEN	Finds the effective length of a string ignoring trailing blanks.
ERRPRT	Prints error messages.
FILSOL	Produces a binary solution file.
FIND	Searches for matching string of a text.
FMONIT	Monitors the infeasibilities.
FTRAN	Makes an FTRAN step.
INVERT	Makes an LU re-inversion.
LPMAST	Produces LP matrix statistics.
LPSOLV	Governs the solution of the LP.
LUDOT	Computes scalar product of sparse eta vectors for the LU inversion.
MANCOL	Performs memory management in column-ordered list during inversion.
MANROW	Performs memory management in row-ordered list during inversion.
MDATE	Computes the day number in the year from the date.
NUMEDI	Formatting routine for numbers.
NUMVAR	Returns external variable number, given its internal number.
OBJECT	Computes the objective function.
PIVCOL	Scans pivot column candidate to produce a column eta vector and a pivot element.
PIVROW	Scans pivot row to produce a row eta vector during inversion.
PRICE	Computes reduced costs.

RECBET	Reconstructs updated right-hand side β.
RHSINP	Reads RHS Section input.
RNGINP	Reads RANGES Section input.
ROWINP	Reads Matrix File heading and ROWS Section input.
ROWOUT	Produces row section output.
ROWPRT	Prints row section output.
SIGNUM	Makes a structural picture of the LP matrix.
SISPA0	Tries to eliminate zero slacks (Phase 0).
SISPAR	Performs Phase 1 or 2 of sparse revised simplex.
TABLIS	Prepares column ordered list for constraint matrix.
TRIN1	Sorting routine for one integer vector.
TRIN2	Sorting routine for two integer vectors.
UPDATE	Makes a single step in the updating of ordered lists for inversion.

Finally, a BLOCK DATA subprogram provides default settings of the FORTRAN unit numbers for the program files.

There are three auxiliary routines collected in the file LPAUX. The contents of the LPAUX subroutines depend on the actual type of computer and the FORTRAN compiler.

Name	**Task**
DATJOB	Returns current date.
MCLOCK	Timing routine.
OPENLP	Performs proper opening of LINPROG files.

Index

— A —

A, indicator for alternative solutions, 15
Activity, 7
Activity, Output, 14
Adjacent basis matrix, 21
Algorithmic description of LINPROG, 71
Alternative optimal solutions, 15
Anti-cycling procedure by Gill *et al.*, 54
Apollo DN10000, 6, 63
Artificial variables, 48
ASCII Result File, 17
ASCII standard collating sequence, 58
Assignment of pivots in re-inversion, 32
Augmented constraint matrix, 47

— B —

Bartels, R. H., 48, 55
Bartels-Golub update procedure, 57
Basic feasible solution, 21
Basic index set, 7, 16, 48
Basic variables, 7, 21
Basis, 7
Basis matrix, 7, 21
Basis-inverse, 23
Benichou, M., 32
BIG keyword, 55
Big-M method, 53
Binary output (BINOUT), 8, 17
BINOUT keyword, 8, 17
Block triangular rearrangement of basis matrix, 35
Boolean inversion, 33
Borderline problems between feasibility and infeasibility, 64
Bounded simplex method, 44
  Selection of leaving variable, 45
  Updating basic solution, 46
Bounds, 6, 11, 43
  Lower bounds, 11, 44
  Multiple set of bounds, 11
  Translation of bounds, 44
  Upper bounds, 11
BOUNDS instruction, 9, 58
BOUNDS Section, Input, 11
BTRAN operation, 23, 41, 50
Bump, 31

— C —

Cancellation of matrix elements, 31
Chronological numbering of rows and columns in basis matrix, 34
CHUZC operation, 23, 41, 46, 50, 53, 55
CHUZR operation, 23, 42, 46, 50, 54, 56–57
Cohn, C. F., 58
Collating sequence, 58
Column replacement updating, 25
Column singletons, 31–32
Column-eta vectors, 30
COLUMNS instruction, 9
COLUMNS Section, Input, 10, 58
COLUMNS Section, Output, 14ff
Communication file, 8, 16–17
Condition for minimal SINF, 50
Condition for optimality, 45
Constraint matrix, 7, 21
Constraint type, *see*
  • Restriction type
Constraints, 5
Control File, 7ff, 16
CPU time consumption, 15–16
CRASH procedure, 47–48
Crout's method, 31
Current tableau, 23
Cyclic permutation, in Forrest-Tomlin updating, 37
Cycling, 21, 23, **54**, 56–57, 64

— D —

Dantzig, G. B., 20
Decomposition of triangular matrices, 26
Degeneracies and degenerate solutions, 21, 52–54, 57–58
DEVEX scheme of Harris, 53
Diet problem, 5
Direct simplex, 23
Direction indicator, 51
Double precision, 18, 20
Dual activities, 15
Duff, I. S., 35, 57
Dummy restrictions, 21
Dump facility in LINPROG, 16
Dump File, 16
DUMP keyword, 8, 17

— E —

ECHO keyword, 7
Effective β-vector, 44
EFOM project, 58, 64
Elbow room, for sparse-matrix inversion, 33, 35, 58
Electronic mail, 59
Elementary column matrix, 24, 38–40, 42
Elementary matrix, 24, 38
Elementary product forms, 27
Elementary row matrix, 25, 36, 38, 40
ELTAB keyword, 7
ENDATA instruction, 9
Entering the basis, 21
EPSCHC keyword, 55
EPSCHR keyword, 56
EPSFEA keyword, 56
EPSINA keyword, 57
EPSLU keyword, 57
EPSPIV keyword, 57
EPSRIN keyword, 33, 57
Error messages from LINPROG, 16
Eta lists and eta files, 29–30
Eta vectors, 29, 57
European Communities, 58
Exchange of variables, 21
EXEC keyword, 7, 9

— F —

Feasibility, 7, 14, 48
Feasibility test, 56
Feasible point, 7
Feasible solution, 7
Fenhann, J., 58
Files used in LINPROG, 16
Fill-in, 31–33, 35, 57
Filter for small elements, 57
Fixed variables, 6, 11
Forrest-Tomlin method, 23, 28–29, 35, 57
  Concurrent scanning of lists, 43
  Implementation in LINPROG, 42
  Interface with simplex, 41ff
  Scanning the L-file, 41
  Scanning the U-file, 41
  Zeroing of elements, 40–41
Forsythe, G. E., 26
FORTRAN unit no.s for files, 16
FR instruction, 11, 46
Free variables, 6, 11
Frobenius matrix, 24
Fröberg, C.-E., 58
FTRAN operation, 23, 41, 54
FX instruction, 11, 46

— G —

Garbage collection, 33
Garfinkel, R. S., 54, 58
Geometrical interpretation of simplex, 23
Gill, P. E., 54
Goldfarb, D., 53
Golub, G. H., 24, 57
Grohnheit, P. E., 59, 64

— H —

Harris, P. M. J., 52–53
Hellerman, E., 35

— I —

IEEE floating-point standard, 20
Infeasibility, 7
Infeasibility above upper bound, 48
Infeasibility below lower bound, 48
Infeasibility cost vector, 49
Infeasibility pricing vector, 50
Infeasible "solution", 15, 21, 48
Inhomogeneous objective functions, 21
Initial basic feasible solution, 21, 47
Initial basis, 47
Initialization of simplex, 23, 47
Input cost, Output, 15
Input to LINPROG, 7ff
Installation of LINPROG, 20
Integer programming, 6, 58
Interior-point methods, 64
Inversion, 7, 30ff; *see also*
  • Re-inversion
  algorithmic description, 34
Iteration count, 16
Iteration printout, 8

— K —

Karmarkar's method, 64
Karmarkar, N., 64
Keywords
  Action keywords, 7
  Descriptive keywords, 7
  Numerical keywords, 7

— L —

Leaving the basis, 21
Lexicographical vector ranking, 54
Linear equations, 48, 58
Linear program, 5
Linear programming, 5
Linked lists, 35, 57
LO instruction, 11, 46
LOGFRQ keyword, 8
Logical variables, 47
Lower limit, Output, 15
LP extensions, 6
LU product form, 28
LU-factorization, 26, 28, 30
Lustig, I. J., 64

— M —

Major iterations, 54
Matrix File, 9ff, 16, 58
Matrix format, 6
Matrix generator, 17, 58
Matrix inversion in product form, 29
Matrix relations, 24ff
MAX keyword, 8
MAXCPM keyword, 8, 17
Maximizing, 21
MAXITS keyword, 8, 17
Memory management for sparse-matrix inversion, 33, 35
MI instruction, 11, 46
Minor iterations, 54
MINOS, 15, 58
MITRE keyword, 8, 30
Moler, C. B., 26
MPS format, 9
MPSX, 6, 9, 14–15, 58
MSCALE keyword, 8, 64
Multiple pricing, 54
Multiple right-hand sides, 10
Multiple-target pricing, 53
Murtagh, B. A., 20, 24, 58

— N —

NAME instruction, 9
Nazareth, J. L., 20, 48, 53, 58
Nemhauser, G. L., 54, 58
NETLIB collection of test problems, 6, **59**, 64
NINF, 50, 57
Nonbasic variables, 21
    at a bound, 44
    Shift between bounds, 46, 54
Norms , *see*
    • Vector norms
Nucleus, 31–32
Number of infeasibilities (NINF), 50
Numerical feasibility, 64
Numerical stability of re-inversion, 33

— O —

Object(ive) function, 5, 10, 21, 44
Objective row, 10
Optimization
    Economics, 5, 58
    Energy systems, 5, 58
    Nuclear reactors, 5, 58
Orchard-Hays, W., 31
Ordered lists, 57
    for sparse-matrix inversion, 33
Output from LINPROG, 12ff
Output summary, 15

— P —

Packed arrays, 27
Packed sparse vectors, 57
PARAMETER in FORTRAN 77, 20
Parametric programming, 6
Partial pricing, 54
Partially updated incoming vector, 36
PC-version of LINPROG, 6
Pedersen, J. Munksgaard, 58
Pedersen, O. Gravgård, 59
PEND keyword, 7, 9
Permutation matrices, 25
Permutation of columns of a matrix, 26
Permutation of rows of a matrix, 26
Perturbation analysis, 16
Perturbation in feasibility test, 56
PFI method, 35
Phase 0, 47–48, 57
Phase 1, 47ff
Phase 2, 48
PICTURE keyword, 8
PILOT test case, 64
PILOTJA test case, 54, 64
Pivot column, 25, 40
Pivot element, 23, 25, 31, 36
Pivot index, 23, 38
PIVOT operation, 23, 42, 46, 54
Pivot selection, 35
    Strategy in re-inversion, 31

Pivotal factor, 40
PL instruction, 11, 46
Pointer arrays, 57
Polytope, 23, 53
Pre-ordering of rows and columns of basis matrix, 31
Pre-pivotal factors, 40
PRICE operation, 23, 41, 53
Pricing strategies, 53
Pricing vector, 22
Printer file, 16
Product representation, 23
Protection against cycling, 54

— Q —

Quadruple-precision version of LINPROG, 64
Quicksort method, 58

— R —

Range for a right-hand side, 10–11, 47
Range value, 11
Ranges, 43
RANGES instruction, 9, 46
RANGES Section, Input, 10
Rank of matrix, 21
Rank-deficient constraint matrix, 21
Rank-one updated matrices, 24
Rarick, D. C., 35
Ratio test, 22
Re-factorization, 30
Re-inversion, 23, 27–28, 30ff, 57
  algorithmic description, 34
Rearrangement of basis matrix before inversion, 32
Reduced costs, 15, 22, 55
Reduced infeasibility costs, 50
Redundancies, 48, 58
Reference space, 53
Reid, J. K., 35, 53
Removal of zero slacks, 47
Repair procedure for feasibility restoration, 57
Report writers, 17
  REPORT program in FORTRAN, 18
  REPORT program in MODULA-2, 18
  REPORT program in PASCAL, 18
  Source codes in FORTRAN and PASCAL, 68
Restart facility in LINPROG, 16
Restart File, 16
RESTART keyword, 8, 16
Restraints, 5
Restriction type, 5, 10
Restrictions, 5
Result File, 12, 16
Revised simplex, 23
Rhs, 7
RHS instruction, 9
RHS Section, Input, 10
Rounding errors, 54, 57
Row singletons, 31–32
Row-eta vectors, 30
ROWS instruction, 9
ROWS Section, Input, 10, 58
ROWS Section, Output, 14ff

— S —

Sample output, 12ff
Sample problem, 5–6, 9
Scalar products used in matrix inversion, 33
Scaling, 55
  Column scaling, 55
  Options, 8
  Row scaling, 55
Selection of leaving variable in Phase 1, 50
Shadow prices, 15
Sherman-Morrison identity, 24
Simplex algorithm, 23
Simplex method, 6, 20ff
Simplex multipliers, 15, 22
SINF, 48ff, 57
Slack activity, Output, 15
Slack variables, 21, 47
SOLUTION keyword, 8
Sorting, 58
  in matrix inversion, 34
Sparse-matrix techniques, 6, 57
  in re-inversion, 33
Sparsity preservation, 31–32
Spikes, 35
SPUT matrix, 37–38, 41
Standard form of LP, 21
Standard product form, 27
Stanford test cases, 63
Steepest-edge pricing, 53
Step size in simplex, 22, 45
  Phase 1, 52
Strategy for CHUZR in Phase 1, 52
Structural variables, 5, 47
Subroutines in LINPROG, 73
Sum of infeasibilities (SINF), 48ff

Summary line, Output, 15
SUN workstation, 6, 64
Symmetric permutations, 26, 37

— T —

Test of LINPROG, 58ff
Tie breaking, 54, 56
Tolerances, 8, 54, 64
    default values, 55
Tomlin, J. A., 55, 65
Transformed right-hand vector, 22
Triangular factorization of basis matrix, 28
Two-pass CHUZR, 52

— U —

Unbounded solution, 23, 45
Unformatted output file, 17
UNISYS A6 computer, 6, 64
UP instruction, 11, 46
Updated incoming vector, 41, 50
Updating of basis matrix factorization , *see*
        • Forrest-Tomlin method
Upper limit, Output, 15
User's Guide for LINPROG, 7ff

— V —

van Emden, M. H., 58
van Loan, C. F., 24
VAX, 6, 19, 63
    COM files for job programs, 19
    D-floating, 20, 64
    DCL, 19
    Dump example, 19
    EXE-file, 20
    G-floating, 20
    Job programs, 19
    Pagefile quota, 20
    Restart example, 19
    SUBMIT command, 19
    WSextent, 20
Vector norms
    1-norm, 56
    Maximum norm, 55
    $N(\beta)$, norm-like function, 56
Vector-matrix notation, 21
Violation of constraints, 16
Virtual memory, 20, 57

— W —

Weber, S., 58

— Z —

Zero-slack variable, 47
ZERPIV keyword, 48, 57

Bibliographic Data Sheet **Risø–M–2797**

Title and author(s)

LINPROG: A Linear-Programming Code Developed at Risø

Peter Kirkegaard and Ole Lang Rasmussen

ISBN
87–550–1541–7

ISSN
0418–6435

Dept. or group
Computer Section

Date
March 1990

Groups own reg. number(s)

Project/contract no.

Pages	Tables	Illustrations	References
79	3	5	30

Abstract (Max. 2000 char.)

A computer code LINPROG written in Standard FORTRAN 77 has been developed at Risø for solving medium- to large-scale linear programming problems. It runs primarily on a VAX-8700 computer, but also on other systems where virtual memory is available. LINPROG uses the revised simplex method with the Forrest-Tomlin updating scheme of the inverse basis. Sparse-matrix techniques are applied throughout. A comprehensive test and verification study has been performed with data sets provided by local users and with data sets available in the literature.

Descriptors INIS/EDB

Available on request from:
Risø Library, Risø National Laboratory (Risø Bibliotek, Forskningscenter Risø)
P.O. Box 49, DK–4000 Roskilde, Denmark
Phone +45 42 37 12 12, ext. 2268/2269 · Telex 43 116 · Telefax +45 46 75 56 27

Addendum to the LINPROG documentation

by

Peter Kirkegaard and Poul Erik Grohnheit
Risoe National Laboratory
DK-4000 Roskilde
Denmark

Fax: +45 42 37 39 93

January 1992

Scope

Since the issue of the LINPROG report [1], the code has been extended and improved in various ways. The aim of this note is to document these enhancements. The current LINPROG version is given the release identification 9201. Backward compatibility is maintained, such that the main documentation [1] is still perfectly valid as a user manual also for the present release of LINPROG, apart from the new features.

Larger problem capacity

The problem capacity was increased in the new version of LINPROG. The maximum number of rows and columns are now MR=12209 and MC=11845, respectively.

The reduction module in LINPROG

The most important new facility added to LINPROG is the reduction module. This is able to decrease the effective size of the LP problem by recursive application of an elimination technique to the original constraint matrix. By this process redundant variables and constraints can be removed. Moreover nodal balance equations are identified and eliminated. Such equations are equality constraints in which one variable is a positive linear combination of other variables. With this form the positivity is guaranteed automatically after the elimination.

The reduction facility is activated by using the new keyword REDUCE. For example, the following control file:

```
REDUCE
BINOUT
EXEC
PEND
```

solves an LP using the reduction module, and directs the solution to a binary output file.

When REDUCE is on, LINPROG will print a short summary of the reduction statistics. This might look as follows:

```
====================== REPORT ON MATRIX REDUCTION ======================
SIZE OF UNREDUCED LP MATRIX: 1953 X  1899
SIZE OF   REDUCED LP MATRIX:  841 X   851

  SCANS    SINGLE-  IMPLIED  IMPLIED  IMPLIED   SIGN     ELIMINA-  FINAL
 IN ROW     TON     LOWER    UPPER    FIXED    REDUN-  TED TREE LENGTH
 SEARCH     ROWS    BOUNDS   BOUNDS   BOUNDS   DANCIES   EQNS.   OF LIST
    4       289       18       1       270       37       776     8685
========================================================================
```

DUMP/RESTART works properly together with REDUCE, provided the user takes care of restarting only REDUCE problem from REDUCE dumps (otherwize a conflict in the matrix size may occur).

All vectors and matrices in LINPROG are physically reduced to smaller sizes (in contrast to e. g. the reduction technique proposed by Tomlin and Welch [3]). After the reduced system is optimized, a recreation procedure must be invoked. In LINPROG this is implemented by taking advantage of the capability of the code to restart from any basic index set.

Among several possible elimination strategies we have chosen the so-called fixed Markowitz method, which is capable to keep matrix fill-in and arithmetics at a low level during the reduction process, using fairly simple coding. The elimination process is facilitated by a double set of linked lists with coefficient values only in the row list.

To illustrate the power of our reduction module, we have tried it on a so-called EFOM problem. EFOM [2] is an energy flow optimization model which is characterized by many topological network constraints resulting in nodal balance equations, though it is not a pure network model. Specifically, we have solved a problem with 5645 rows and 5962 columns without reduction in 2623 cpu seconds on our VAX-8700. With REDUCE activated the time dropped to 1340 seconds, and this time includes reduction as well as recreation.

Miscellaneous new features

Apart from the reduction module, certain minor improvements are incorporated in the present version of LINPROG. They are listed below:

1) We have taken measures to decrease the probability of cycling and "stalling" (this means that virtually no progress is made during many consecutive simplex iterations).

2) In the solution output, violation of activity ranges are flagged with ** in the same way as violation of constraints.

3) Not only the date, but also the time of the day, is given in the output.

4) In the matrix file, it is now permitted to let the restriction key (N, L, E, or G) be in either column 2 or 3.

5) Header and trailer records are added to the dump/restart files for problem identification. This permits a compatibility check of the restart file.

Program structure

The extensions in LINPROG resulted in several new subroutines. The table below is an addendum to the list in Appendix A3 in [1]:

Name	Task
DELETE	Delete an entry from double linked list for reduction
ELIMIN	Eliminates variable from constraint matrix during reduction
LOAD	Loads information from restart file
NEW	Adds a new element to double linked list for reduction
REDLNK	Makes double linked lists for reduction constraint matrix
REDMOD	Modifies lists and arrays for reduced matrix
REDSAV	Saves original lists for recreation after the reduction
REDSIG	Searches for redundant inequalities by sign test
REDSRW	Eliminates singleton rows in reduction process
REDSTA	Writes statistics for reduction process
REDTRE	Eliminates nodal balance equations
RESTOR	Restores original lists to be used in recreation
RHSOFF	Modifies rhs to accomodate for offset in a variable

We have decided to use INCLUDE files for COMMON declarations in the new LINPROG version, although this adds to the very few LINPROG violations of Standard FORTRAN 77. Also an included file with problem size parameters (set by FORTRAN PARAMETER statements) is used. These included files limit the repetitions in the source code and make the maintenance of the code both safer and easier. In particular it is now very easy to increase parameters related to problem size in LINPROG. This INCLUDE design of the code organization was patterned after the EFOM software [2].

The file table (cf. Table 2 p. 16 in [1]) should be extended by one more file, which is used when REDUCE is activated:

Unit No.	File	Usage	Mode
12	Temporary for reduction	Scratch	Binary

LINPROG running on personal computers

The list of computers where LINPROG is running covers Digital VAX, UNISYS A6, Apollo DN 10000, and SUN workstations, see p.6 in [1]. In 1991 we also succeeded with getting LINPROG to run on personal computers of the 386 type under DOS. Specifically, we compiled LINPROG with the FTN77 Fortran-77 compiler developed by Salford University, UK, with virtual

storage provided by the DBOS run-time system. The computer used for our tests was an Olivetti PC/M380-XP1 with Cyrix FasMath 83D87 Math Processor under DOS 5.0. It turned out that any problem that could be solved on our VAX-8700, could also be solved on this PC, only the execution time was about five times longer. Thus the "small" computer environment is an important target for LINPROG.

References

[1] Kirkegaard, P.; Lang Rasmussen, O., LINPROG: A Linear-Programming Code Developed at Risoe. (Risoe-M-2797, Risoe National Laboratory, Roskilde, Denmark), 1990.

[2] Van der Voort, E.; Donni, E.; Thonet, C.; Bois d'Enghien, E.; Dechamps, C.; Guilmot, J. F., Energy Supply Modelling Package EFOM-12 C Mark I - Mathematical Description. CABAY, Louvain-la-Neuve, for the Commission of the European Communities, 1984).

[3] Tomlin, J. A.; Welch, J. S., Integration of a primal simplex network algorithm with a large-scale mathematical programming system (ACM Trans. Math. Software, vol. 11, pp. 1-11, 1985).