FortMP Manual

Last Update
23 April 2008

OptiRisk SYSTEMS

INVESTOR IN PEOPLE
BS EN ISO 9000 : 2000


FortMP

Manual Prepared by

E F D Ellison
M Hajian
H Jones
R Levkovitz
I Maros
G Mitra
D Sayers (NAG Ltd)

.

Preface to Release 2

A) Changes Made Since Release 1

Many improvements have been made throughout the system and new features have been introduced. The major new features introduced are in the input, in Mixed Integer Programming, and in the Callable Library (external data interface) as follows:

Input Data

Standard data input is extended by new features permitting free-form layout and long data names, enlarged from 8 to 16 characters.

Mixed Integer Programming

MIP has been substantially revised in order to permit addition of advanced new methods and to solve models irrespective of the limitations imposed by maximum storage constraints. A system for recovery of 'dead' space enables large search trees to be developed with comparatively little memory.

A powerful algorithm for MIP pre-processing has been added with substantial time improvement in the solution of many MIP problems.

A feature has been added for the users to classify the MIP constraint-types in their models.

Callable Library and External Data Interface

Many users of External Data Interface - module SUBMP1 - have been obliged to write their own data input procedures based on MPS-form data layout. The detailed checking and more complex features are usually omitted as a consequence. Now there is a new input feature available which reads from MPS-form or from any other FortMP data form and makes the data available to a user via the external data interface in the same layout as that used for passing data to SUBMP1. Features are added to enable users to look up names or classes of name so that they can program according to the data structure.

Miscellaneous additions

In the PRIMAL algorithm a weighted column-selection procedure (DEVEX) has been added as an option (good for some models).

In the DUAL algorithm a pivot-choice to counter degeneracy has been added as an option.

In the IPM algorithm the chances of instability occurring are greatly reduced by Conjugate Gradient iterations added as a refinement to the main iterative process.

The crossover from IPM to SSX (basis recovery or BASREC) has been substantially improved and no longer extends solution-time by a large amount.

Internal data interfacing routines are augmented by facilities to access the updated tableau.

Research programs

Newly developed algorithms include Quadratic Programming, Stochastic Programming, and a suite of CUT generating procedures for MIP. These are not yet a part of the standard FortMP release but may be made available on application to the developers.

B) Changes Made In This Manual

A variety of minor corrections have been put in which are not described here. Significant changes are described - chapter by chapter

Chapter 1:- Introduction and background

This chapter is not changed

Chapter 2:- Elementary use of FortMP

This chapter is not changed

Chapter 3:- Overview of the Stand-Alone FortMP System

3.1 Introduction and overview

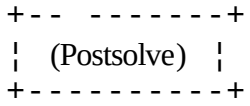
3.1.1 The algorithms employed

Mention is added of supporting algorithms:

BASREC (IPM-SSX crossover)
MIP Pre-processor

3.1.5 Overall structure of the system

The diagram, Figure 3, is amended to include:



following 'Primal'.

3.2 Data Preparation

The existing section 3.2.3 is renumbered as 3.2.4 and a new section 3.2.3 is added

3.2.3 Free Format MPS and Long Data Names

This newly added section introduces commands:

INPUT TYPE FREE
INPUT LONG NAMES

and describes the corresponding changes in MPS-form input.

3.2.4 Other types of input data

Previously this was section 3.2.3

3.3 Running the System

3.3.4 Running Mixed Integer

The following new commands are described here:

MIP PREPROCESS <ON/OFF>
MIP PRIORITY UP <ON/OFF>

introducing new features together with:

MAXIMUM MIP NODES = n
MAXIMUM MIP TIME = v

introducing new limits, and

NODE LOG FREQUENCY = n

to provide for reduced volume of node logging

3.3.5 Using the PRESOLVE algorithm

Presolve level maximum is changed to 5 (new default) and the following command is added:

PRESOLVE LOG LEVEL = n

3.4 Output Descriptions

3.4.3 Log File Output

Minimum log level is now zero instead of one. Level zero suppresses all logs except for error messages.

The following new command is described:

LOG DISPLAY LEVEL = n

and the description of 'LOG DISPLAY' commands is clarified.

3.4.4 Suppressing Output

This is a new section describing how to avoid unnecessary output.

The following new commands are described:

OUTPUT <ON/OFF>
OUTPUT SUPPRESS ZERO

3.5 Further Topics

3.5.1 Saving and Restarting

Default frequencies are changed. Frequency 0 (zero) has the effect of cancelling SAVE's.

3.6 Input and Output Files in FortMP

The list of internal files and their extensions is amended

3.7 Errors and Recovery

3.7.1 Data Errors

Up to 50 data errors before halting.

3.7.3 Numerical Difficulties and Instability

Dual infeasibility or unrecoverable numerical error in the DUAL algorithm result in a revert to the primal algorithm.

3.8 Summary of SPECS Commands

The new commands mentioned above are added to this section.

Chapter 4:- Sparse Simplex (SSX) Solver

4.6 SSX Algorithm Controls

4.6.1 The Principal controls

The following enhancements are described

- DEVEX algorithm (description in 4.3.2)
- Automatic Forrest-Tomlin activation
- Dual anti-degeneracy procedures (description in 4.4)

The following commands are added:-

PRIMAL DEVEX <ON/OFF/SINGLE/DOUBLE>
FORTOM AUTO
DUAL ADEGEN = n

together with descriptions.

4.6.3 Tolerances and other parameters

The title of this section is extended to include 'parameters'

The following commands to change tolerances and parameters are specified:

PRIMAL DEVEX RATIO = v
FORTOM ACTIVATE PERCENT = n
FORTOM ACTIVATE GROWTH = n
DPROGRESS CRITERION = v
DPROGRESS FREQUENCY = n
DUAL PIVOT THRESHOLD = v

4.6.6 Miscellaneous Controls

Log levels are now 0-4, default is 1.

4.7 Summary of SPECS Commands

This section is reduced to summarise only the commands described in chapter 4. Reference is made to 3.8 for the previous summary and to appendix B for the complete list.

Chapter 5:- The Interior Point Method

5.2 Controls on the IPM Algorithms

The existing section 5.2.4 is deleted and replaced by the new section 5.2.4 described below

5.2.1 Using the Algorithms

The default for 'IPM RELATIVE EPSILON' is changed to 1.0e-9.

5.2.3 Choice of Solution Algorithm for the Equations

The default for 'IPM SOLVER' is changed to 'XSUPERNODE'. Also the following new command is described:

IPM TOFIX = v

5.2.4 Refinement by Conjugate Gradient Iterations

Describes how CG iterations are used to improve the accuracy of each iteration. The following controls are specified:

CHOLESKY CG TOLERANCE = v
CHOLESKY ERROR TOLERANCE = v
MAXIMUM CG ITERATIONS = n

5.2.5 IPM Save and Restart: Iteration Limit

Default SAVE frequency is increased

5.2.6 IPM-SSX Crossover Options: BASREC

The following new command is described:

DUAL PUSH <ON/OFF>

5.2.7 Miscellaneous IPM Commands

The following new commands are described:

PUSH LOG FREQUENCY = n
PUSH LOG LEVEL = n

Save-files model.a/b/c/d are described for graphical output display.

5.3 Summary of SPECS Commands

This section is reduced to summarise only the commands described in chapter 5. Reference is made to 3.8 for the previous summary and to appendix B for the complete list.

Chapter 6:- Mixed Integer Programming (MIP)

This chapter has been extensively re-written and the text cannot realistically be correlated with chapter 6 of release 1. The general description of MIP remains with certain corrections to clarify the meaning and the data input is unchanged in outline but with the example presented differently.

Section 6.6 described the new version of FIXMIX and PRIORITY features - now changed and made more flexible. The AUTO-ROUND feature is also changed - see 6.8.1

Elsewhere the reader will find additions as follows:

- UP node-choice priority feature (6.5.3)
- AGENDA data given by name (6.6.3)
- FIXMIX output feature (6.6.4)
- Advanced algorithms (6.7)

- Cutoff controls (6.8.2)
- New search limits (6.8.3)

Because of memory savings made, the MIP DISK feature is no longer used.

Chapter 7:- FortMP Subroutine Library and External Data Interface

The existing release 1 features described in Chapter 7 are retained as before (7.1 and 7.2) with certain improvements to the text.

Readers should note the change in specification for REAL-type arguments which become DOUBLE PRECISION in the new release standard version (see 7.2.3). Special versions using single precision (REAL) arguments can be delivered to customers who need compatibility with their existing programs.

The new interface features are described in sections 7.3 to 7.6 and a summary is given in 7.7.

Chapter 8:- Data Interfacing Service Utilities

Most of the material in this chapter is unchanged from release 1. Readers should note the change in specification for REAL-type arguments which become DOUBLE PRECISION in the new release standard version (see 8.1.6). With double precision the criterion for infinite bounds is increased (see 8.3.7 notes 4. and 5.)

Subroutines GTABLR and GTABLC are added to enable user to obtain vectors from the updated tableau after execution of PRIMAL or DUAL algorithm (see 8.1.2, 8.1.3, 8.1.4, 8.2.4 and 8.3.5 with arguments added to 8.3.1).

A change is added to mixed integer type-codes to enable users to recognise the beginning of each SOS (see description of MIT in 8.3.1)

Appendices

Chapter 9 of release 1 has been re-cast as appendices to allow addition of new chapters in the future.

Included in Appendix A (Input/Output Data Layouts) is the detailed specification of newly added free-format input and long data names, together with the layout of MIP Agenda files.

Appendix B (SPECS Commands - formerly section 9.3) is thoroughly revised and brought up to date with all commands, including minor commands not described before in the manual. This appendix acts as an index, giving references to those sections of the manual where a command is described.

Preface to Release 3

A) Changes Made Since Release 2

General Improvements

Release 3 of FortmP (March 1999) is provided with means to adapt its memory usage dynamically to the size of the problem being solved. It will no longer require user to specify machine-size or problem-size in order to have a version specially tailored.

Two versions now cover the needs of virtually all users:

DS Double precision, Short Indexing for problems with no more than 32k constraints

DL Double precision, Long Indexing for larger constraint-size (virtually unlimited).

Single precision versions are also available but their use is not recommended in view of the loss of accuracy.

Memory adaptation is now available also for parallel versions and for calling as a DLL from other systems such as MPL. Standard subroutine libraries are delivered replacing separate modules.

Input Data Modelling

A better integration with the MPL modelling system has been developed permitting direct, internal communication of files, and with a dialogue developed for the entry of SPECS commands (Appendix D).

Primal Algorithm

New procedures are added to facilitate research, particularly in connection with parallel execution (Appendices F, H).

Dual Algorithm

Dual Phase I has been added together with better control of numerical accuracy and additional SPECS controls (Sections 4.4, 4.6.7).

Mixed Integer Programming

An auto-rounding heuristic feature has been developed with consequent changes to its previous description in Release 2 (Section 6.8.1).

Extensions have been added to the cutoff controls (Section 6.8.3).

Internal Specs Communication

New features are now available for the subroutine library user to simplify the preparation of SPECS commands by including common sections that are read by all calls to the solver (Section 7.4.4).

Quadratic Programming

The chapter on QP is added to this manual (Chapter 9). FortMP-QP is an IPM based solver for convex quadratic objectives (Q-matrix positive semi-definite). It has been extended to solve integer QP using a branch and bound algorithm.

'CRASH' Features

The special CRASH(SOR) feature is added in this manual together with other CRASH extensions (Chapter 10).

New Appendices

New appendices are added as follows:

- Appendix C on installation procedures
- Appendix D on modelling systems
- Appendix E on C-language usage
- Appendix F on parallel execution
- Appendix G on memory management
- Appendix H on special testing and other miscellaneous extras.

B) Changes Made In This Manual

A variety of minor corrections have been put in which are not described here. Significant changes are described - chapter by chapter

Chapter 1:- Introduction and background

A statement of the QP problem is added (Section 1.7) with introduction (Section 1.1) and reference to the author (Section 1.2). New platforms are mentioned (Section 1.6).

Chapter 2:- Elementary use of FortMP

This chapter is not changed

Chapter 3:- Overview of the Stand-Alone FortMP System

3.7.4 Running Out of Memory

This section is re-written

Chapter 4:- Sparse Simplex (SSX) Solver

4.3 PRIMAL Algorithm

4.3.2 Column Selection

Additional column selection procedures are mentioned

4.4 DUAL Algorithm

This section has been re-written and considerabley enlarged, with sub-sections corresponding to those of the PRIMAL algorithm - section 4.3.

4.6 SSX Algorithm Controls

4.6.1 The Principal controls

'DUAL DEVEX' controls are added and the 'DOUBLE' option for primal devex now has a new meaning:- apply DEVEX in phase 1.

4.6.3 Tolerances and Parameters

The pivot-control commands are changed so as to be less confusing and to provide separate PRIMAL, DUAL and INVERT application. New commands specified in section 4.6.7 complete the picture.

4.6.7 Special Pivoting and Update Commands

This section is added so as to describe additional new commands.

4.7 Summary of SPECS Commands

Revised to incorporate the earlier corrections and additions.

Chapter 5:- The Interior Point Method

This chapter is not changed.

Chapter 6:- Mixed Integer Programming (MIP)

6.8 Miscellaneous MIP Controls

6.8.1 Automatic Rounding Heuristics

This section is re-written. Auto-rounding is now re-designed as a heuristic for finding integer solutions quickly, before the main tree is developed. 'PROBE' rounding is discontinued.

6.8.2 Cutoff and Tolerance Control

New controls on the bound and the cutoff tolerance are added and a more complete description is given.

6.8.3 Placing Limits on the Tree Search

'MAXIMUM' controls are added for the auto-rounding heuristic.

6.11 Summary of MIP SPECS Commands

Revised to incorporate the earlier corrections and additions.

Chapter 7:- FortMP Subroutine Library and External Data Interface

7.2 External Data Interface

7.2.5 An Example

'MAXIMIZE' is added to the sample SPECS-commands. This was an error in the previous version.

7.4 Internal SPECS Commands

7.4.2 Once-off Entry of SPECS Commands

The specification of 'CALL SPCINT' is corrected.

7.4.3 Default Initialization

The wording of this section is clarified

7.4.4 Common Sections in the SPECS file

This is a newly added section describing the use of *'ALL'* and *'DEFAULT'* sections.

Chapter 8:- Data Interfacing Service Utilities

This chapter is not changed

Chapter 9:- The Interior Point Method for Quadratic Programming

Newly added chapter.

Chapter 10:- Advanced Starting Bases

Newly added chapter.

Appendix A:- Input/Output Data Layouts

This appendix is not changed

Appendix B:- SPECS Commands

B1.3 BEGIN and END Commands

Revised to include description of *'ALL'* and *'DEFAULT'* sections

B2.4 Maximum Limits

Controls limiting the auto-rounded tree search added

B2.6 SSX Controls: Algorithmic

Control `'DUAL DEVEX <ON/OFF>'` added:

B2.7 SSX Controls: Parameters

Pivot tolerance controls changed - separate versions introduced for Primal, Dual and Invert.

New controls added for numerical accuracy and checking.

B2.10 MIP Controls: Algorithmic

New control `'MIP AROUND SOLVER <SSX/IPM>'` added. `'MIP PROBE ROUNDING'` is deleted.

B2.11 MIP Controls: Parameters

Bound and Cutoff controls extended.

New Appendices Added

- Appendix C on installation procedures
- Appendix D on modelling systems
- Appendix E on C-language usage
- Appendix F on parallel execution
- Appendix G on memory management
- Appendix H on special testing and other miscellaneous extras.

Contents

PART 4

APPENDICES

(This page intentionally left blank)

1. Introduction and Background

Contents

1.1 Introduction

FortMP is a Mathematical Programming system designed to solve large scale Linear Programming (LP), Quadratic Programming (QP), Integer Programming (IP) and variable separable programming including special ordered sets of Type 1 and Type 2 (SOS1 and SOS2) problems. It has been developed at Brunel University by the Mathematical Programming Research Group headed by Professor G Mitra, and is marketed by NAG Ltd.

FortMP is used in Management Science or Operations Research and covers applications such as transportation, chemical engineering product blending, economic modelling, energy systems and networks. In short, most industrial applications or research problems involving linear or discrete optimisation are handled by this system.

The main algorithms used by FortMP are the Sparse Simplex (SSX) PRIMAL and DUAL algorithms. These are supplemented for large problems or QP problems by alternative Interior Point Method (IPM) algorithms (Barrier, Affine and Predictor-Corrector) based on the Primal-Dual Logarithmic Barrier Method. Mixed Integer Programming (MIP) problems (including SOS1 and SOS2) are solved using a Branch and Bound tree search algorithm.

Back to Chapter contents

1.2 Background

FortMP is an update of the FortLP system and has been developed through continuing research at the Department of Mathematics and Statistics, Brunel University, under the leadership of Professor Gautam Mitra. A number of funding bodies and industrial partners have supported this research. The Science and Engineering Research Council of the UK and the Department of Trade and Industry UK have funded some of the research and development work. NAG Ltd has been an active collaborator on this project since 1985. Other industrial organisations whose funding and contributions have helped this development include Digital Equipment Corporation's European research initiative, British Gas plc, Parsytec Ltd and the US Army's European Research Office.

A number of people have worked at various stages of this project. Some of the work on inversion was started by Dr K Darby-Dowman [10] and was extended by Dr M Tamiz in his PhD as well as his postdoctoral work [11], [12]. The Interior Point Method was developed by Dr R Levkovitz [13] during and after his PhD work. Dr R Levkovitz has also helped in various vectorisation and parallelisation studies [14], [19]. Most of the integer programming development is due to Dr M Hajian [15] and the work on parallel Branch and Bound has been undertaken by M Hajian and I Hai [16] on a cluster of workstations using PVM (Parallel Virtual Machine). Many recent algorithmic extensions to SSX are due to Professor I Maros [17], [18] who has worked closely with the development team. Dr H Jones has been responsible for extending the IPM algorithm to QP [25]. In integrating the interior point method (IPM) and Sparse Simplex (SSX) and MIP, E F D Ellison has put in considerable effort. The substantial system development effort put in by him, with the able support of all the developers, has turned the research codes into an integrated system covering SSX, IPM and MIP running on many platforms. Dr D Sayers of NAG has worked on many areas of the system and provided enormous

support in all aspects of system production and testing. D Winstanley at University of London Computer Centre has implemented and tuned many versions of the system.

Back to Chapter contents

1.3 Scope and Purpose

FortMP is designed to fulfill the needs of users at different levels of sophistication: beginner, intermediate and advanced.

This manual is therefore subdivided into three parts:

Part 1: Introductory and simple use of FortMP

This part provides a simple introductory tutorial on the use of FortMP. It is intended for users who are not interested in the mechanics of the solution and who simply require a black box to solve their problems.

Part 2: Intermediate level use of FortMP

In this part the major algorithmic capabilities of FortMP are explained. The material in this section addresses the requirements of users with very large or very difficult problems who need the stand-alone system and in addition the power to use controls that overcome the difficulties.

Knowledge of LP and IP techniques would be useful to readers in understanding this manual and in using the stand-alone system, but is not a prerequisite.

Part 3: Advanced use of FortMP

This part describes the advanced use of the system by analysts who wish to

(i) either construct a specialist optimisation application,
or (ii) use the system as a research tool.

A typical user may have multiple problems and may wish to integrate the solver within a system of its own in order to improve processing efficiency. For example a series of problems can be handled making use of the solution to one problem as the starting point for the next.

In such an integration provision is made to communicate the problem data and solution by an internal interface rather than by external files (although external files can be used). Controls can also be supplied internally, externally or be defaulted. Research workers interested in developing algorithms for themselves who need to make use of the internal subroutines of the FortMP system will find this part valuable.

Back to Chapter contents

1.4 Related Documents

Users of the FortMP system are assumed to possess a basic knowledge of linear programming. In order to access all the relevant information covering the capability, performance, installation and usage of FortMP system, the following supporting documents should be consulted as appropriate:

1. FortMP brochure
2. This User Manual
3. Installation Guide
4. Technical Reports and Research Papers

The FortMP brochure defines the functionality and scope of use in summary form, whilst the manual is intended to serve both as a User Guide and as a Reference Manual. The Installation Guide is intended to help installers to implement and test the FortMP system.

A number of Technical Reports and Research Papers are listed in the reference section. These may be valuable for investigators who wish to use FortMP as a development and research tool.

Back to Chapter contents

1.5 Outline

Part 1 of this manual comprises only Chapter 2 and explains the elementary use of FortMP. In this part most of the concepts and controls are introduced through examples. Examples are given for solving bounded Linear Programming (LP) and Integer Programming (IP) problems where the integer variables have binary values or have general integer values. This is all that is needed for most users at the elementary level.

Chapter 3 through to Chapter 6 make up Part 2, the intermediate level. Chapter 3 of this manual introduces the solution algorithms and the advanced use of controls and also gives a description of the printable output files, solution output and log. This introduction is expanded in Chapters 4, 5 and 6 into a fuller description of how to use the three main algorithm groups which are Sparse Simplex (SSX), Interior Point Method (IPM) and Mixed Integer Programming (MIP).

Chapter 7 and Chapter 8 as well as the material in the Appendix make up Part 3 of the manual.

This part is designed for experienced users such as an application developer or a research investigator who wishes to use FortMP as a development or a research tool.

Although the definitive model is introduced in Section 1.7, it is also revisited in Part 2 of the manual. FortMP employs an extended MPS data format; this is introduced by example in both Part 1 and Part 2 of the manual. A complete specification of the industry standard MPS data formats and also extensions to cover SOS1, SOS2 and integer programming, is supplied in the Appendix. The emerging sparse matrix data structure standard is also set out in this Appendix.

Back to Chapter contents

1.6 Platforms

The FortMP system is developed in a way such that it is readily adaptable to a wide range of hardware and software environments, both serial and parallel. ANSI FORTRAN 77 is used for the widest possible portability between systems. The following are environments for which a particular version has been developed:

PC386/486/586 using the Salford FTN77 FORTRAN system
PC386/486/585 using the Watcom FORTRAN compiler
PC Win32 using the MS Development Studio with C compiler and Digital Fortran.
Miscellaneous UNIX based operating systems
DEC VAX systems using VMS and the VAX FORTRAN compiler
Intel I860 hardware using Microway I860 FORTRAN with automatically vectorised subroutines
CONVEX hardware using special optimising FORTRAN compiler with vectorised subroutines
CRAY hardware using the CRAY FORTRAN compiler

The distributed material may be an executable program (for the stand-alone system) or a mixture of compiled and source text material subroutine library. With the latter the user has the option of adapting the memory size available to FortMP. See the Installation Guide for further details.

Back to Chapter contents

1.7 Statement of the Definitive Problem

The general linear programming problem with simple upper bounds may be stated as follows.

Given a set of n variables $x_1, x_2, \ldots, x_n$ and a set of n corresponding constants $c_1, c_2, \ldots, c_n$, minimise (or maximise) the linear function

$$c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

subject to the following m linear constraints:

$$a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{in} x_n :: b_i$$

for $i = 1, 2, \ldots, m$

where $a_{11}, \ldots, a_{mn}$ is an $m \times n$ matrix of constant constraint coefficients and $b_1, b_2, \ldots, b_m$ are constant right-hand sides and where the symbol '::' in the above represents any of the relations:

≤	less than or equal to
≥	greater than or equal to
=	equal to
~	a non-binding constraint

and also subject to the following bounds:

$$l_j \le x_j \le u_j \text{ for } j = 1,2,\ldots,n .$$

This form of problem statement is illustrated in Figure 1 below.

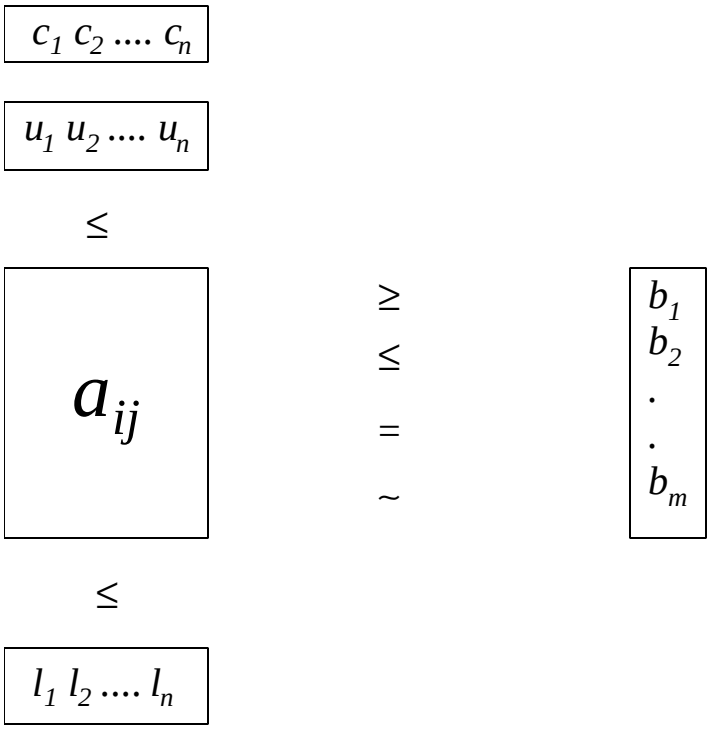


Figure 1. General LP problem with simple upper bounds

In a problem with 'two-sided linear' constraints the problem statement is slightly different, as follows.

> Given variables x_j and constants c_j, a_{ij}, l_j and u_j as before, minimise (or maximise) the same objective as before, subject to the following constraints:
>
> $$L_i \le a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \le U_i$$
>
> for $i = 1,2,\ldots,m$
>
> where $L_1,\ldots,L_m$ and $U_i,\ldots,U_m$ are constant lower and upper bounds respectively for each constraint row. The bounds on variables x_j remain as before.

This is illustrated in Figure 2 below.

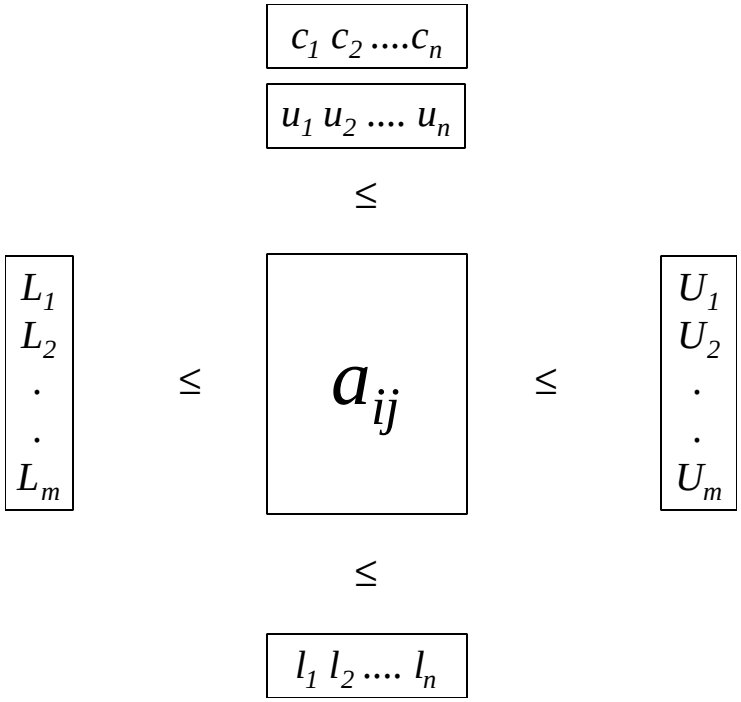


Figure 2. General LP problem with Simple Upper Bounds and RHS ranges

It is implicit in both the above statements that wherever a bound does not exist the corresponding bound values (u_j, U_i, l_j or L_i) is $\pm\infty$ as appropriate. The user signifies each case by the use of type codes in the input data.

In a Mixed Integer problem the following classes of nonlinear variables are allowed.

Binary variables

A binary variable has only two legal values, zero and one.

Integer variables

An integer variable may take only integer values within its given bound range, $l_j \le x_j \le u_j$.

Special Ordered Sets, type 1 (SOS1)

An SOS1 is a consecutive subset of variables in the problem of which only one can be non-zero.

Special Ordered Sets, type 2 (SOS2)

An SOS2 is a consecutive subset of variables in the problem of which at most two can be non-zero and when non-zero these two variables must be adjacent.

Semi-continuous variables

A semi-continuous variable may take the value zero or lie in the range $l_j \le x_j \le u_j$ where l_j and u_j are both positive.

The general statement of the Convex QP problem is similar to that of the general LP problem with the following change permitting the objective to be a quadratic function

Minimize $Sc_j.x_j + ½SSq_{jk}.x_j.x_k$

Or:

Maximize $Sc_j.x_j - ½SSq_{jk}.x_j.x_k$

Summed for *j* and *k* over the range 1 to *n*, where the 'q_{jk}' are elements of a symmetric matrix Q having *n* rows and columns that is either positive definite or positive semi-definite.

The QP solver is also capable of solving problems having binary variables by applying a simplified form of Branch and Bound.

Back to Chapter contents

1.8 References

[1] MITRA, G
Theory and Application of Mathematical Programming
Academic Press, 1976.

[2] ORCHARD-HAYES, W
Advanced Linear Programming Computing Techniques
McGraw-Hill, 1968

[3] BEALE, E M L
Mathematical Programming in Practice
Pitman, 1968

[4] *MPSX – Mathematical Programming System*.
Extended (MPSX), program number 5734 XM4
IBM Trade Corporation, New York, 1971.

[5] MITRA, G
Investigation of some Branch and Bound Strategies for the Solution of Mixed Integer

Linear Programs
Mathematical Programming, **4**, 155–710, 1973.

[6] BENICHOU, B M et al.
Experiments in Mixed Integer Linear Programming
Mathematical Programming, **1**, 76–94, 1971.

[7] FORREST, J J H and TOMLIN, J A
Updated Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method
Mathematical Programming, **3**, 263–278, 1972.

[8] MITRA, G and TAMIZ, M
Alternative Methods for Representing the Inverse of Linear Programming Basis Matrices
Recent Developments in Mathematical Programming, 273–302, Edited by Santosh Kumar, Gordon & Breach, 1989.

[9] MURTAGH, B A and SAUNDERS, M A
MINOS 5.1 User's Guide
Technical Report SOL 83-20R, Department of Operational Research, Stanford University, 1983.

[10] DARBY-DOWMAN, K
An Investigation of Algorithms used in the Restructuring of Linear Programming Basis Matrices Prior to Inversion
PhD Thesis, Brunel University, 1979.

[11] TAMIZ, M
Design, Implementation and Testing of a General Linear Programming System Exploiting Sparsity
PhD Thesis, Brunel University, December 1986.

[12] TAMIZ, M and MITRA, G
FortLP: A Linear and Integer Programming System
NAG Newsletter, October 1989.

[13] LEVKOVITZ, R
An Investigation of Interior Point Methods for Large Scale Linear Programs: Theory and Computational Algorithms
PhD Thesis, Brunel University 1992.

[14] LEVKOVITZ, R
Solving Large Scale Linear Programming Problems using an Interior Point Method on Vector Processors
Departmental Report, Brunel University, 1993.

[15] HAJIAN, M
Computational Methods for Discrete Programming Problems
PhD Thesis, Brunel University, 1992.

[16] HAI, I, HAJIAN, M and MITRA, G
A Distributed Processing Algorithm for Solving Integer Programs using a Cluster of Workstations
To appear in Parallel Computing, Elsevier Press, 1997

[17] MAROS, I
A General Phase-I Method in Linear Programming
European Journal of Operational Research, **23**, 64–77, 1986.

[18] MAROS, I
Adaptivity in Linear Programming, II, (in Hungarian),
Alkalmazott Matematikai Lapok (Journal of Applied Mathematics, Budapest) **7**, 1–71, 1981.

[19] LEVKOVITZ, R and MITRA, G
Solution of Large-scale Linear Programs: A Review of Hardware, Software and Algorithmic Issues
Optimisation in Industry, 1993.

[20] MAROS, I and MITRA, G
Recent advances in Linear and Integer Programming
From 'Simplex Algorithms', Chapter 1 (Oxford University Press, editor: J.BEASLEY).

[21] MAROS, I and MITRA, G
Strategies for Creating Advanced Bases for Large Scale Linear Programming Problems
INFORMS Journal on Computing, USA (submitted, now under revision).

[22] FORREST, J J H, HIRST, J P H and TOMLIN, J A
Practical Solutions of Large Mixed Integer Programming Problems with UPMPIRE
Management Science, 20(5), 1974, pp736-773.

[23] GAUTHIER, J M and RIBIERE, G
Experiments in Mixed Integer Programming using Pseudo-Costs
Mathematical Programming, 12, 1977, pp26-47.

[24] HAJIAN, M T and MITRA, G
Design Implementation and Testing of an Integrated Branch and Bound Algorithm for Piece-wise Linear and Discrete Programming Problems within an LP Framework
Departmental Report, Brunel University 1991.

[25] Jones, H
IPM Solution of the Convex QP Problem, Chapter 8 of: *A computational Investigation of the Solution of Large Scale Optimization Problems*
PhD Thesis, Brunel University 1997.

Back to Chapter contents

2. Elementary Use of FortMP

Contents

2.1 Initial Tutorial

2.1.1 LP Modelling

The following is a simple example of a problem that has been modelled for solution by an LP system.

Minimize the cost function:

$$4x_1 + 6x_2 + 5x_3 + 16x_4 + 2x_5 + 5x_6 + x_7$$

subject to the following constraints:

$$x_1 + x_6 = 2$$
$$x_1 + 3x_2 = 5$$
$$2x_3 + 3x_4 = 4$$
$$x_1 + x_2 + x_3 + 4x_4 + x_5 = 11$$
$$x_1 + 2x_2 + 2x_3 + 3x_4 + x_5 + 2x_6 + x_7 = 14$$

and to the following bounds:

$$0.5 \leq x_1$$
$$0.5 \leq x_2$$
$$0.0 \leq x_3 \leq 1.0$$
$$x_4 = 1.0$$
$$0.0 \leq x_5 \leq 6.0$$
$$1.0 \leq x_7$$

Back to Chapter contents

2.1.2 Elementary Data Preparation

Data such as the above is entered by using a standardised layout, widely accepted as the norm for LP systems, which is referred to as MPS format. In Appendix A1.1 there is given a proforma layout for MPS data entry forms and these forms provide the simplest way to code problem data such as the above.

Data is in five sections: ROWS section, COLUMNS section, RHS section, RANGES section (optional, not used in the example) and BOUNDS section (optional).

In the ROWS section each constraint row including the objective row is named and its type is specified. The ROWS section of the example may be encoded as in Table 1 below.

Field 1 2—3	Field 2 5—12	Field 3 15—22	Field 4 25—36	Field 5 40—47	Field 6 50—61
N	COST	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
EQ	R1	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
EQ	R2	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
EQ	R3	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
EQ	R4	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
EQ	R5	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦

Field 1, Row type:

LE = Less than or Equal GE = Greater than or Equal
EQ = Equal N = Non-binding (objective)

Field 2, Row name

Table 1. The ROWS Section

In the COLUMNS section the coefficient values on the left-hand side of the constraints is given plus the objective row coefficients. Only non-zero values need to be entered. The sequence of entry is column-wise, that is all the values for one column (or variable) must be kept together. The COLUMNS section of the example may be encoded as in Table 2 below.

Field 1 2—3	Field 2 5—12	Field 3 15—22	Field 4 25—36	Field 5 40—47	Field 6 50—61
¦¦¦¦¦¦¦	X1	COST	4.0	R1	1.0
¦¦¦¦¦¦¦	X1	R2	1.0	R4	1.0
¦¦¦¦¦¦¦	X1	R5	1.0		
¦¦¦¦¦¦¦	X2	COST	6.0	R2	3.0
¦¦¦¦¦¦¦	X2	R4	1.0	R5	2.0
¦¦¦¦¦¦¦	X3	COST	5.0	R3	2.0
¦¦¦¦¦¦¦	X3	R4	1.0	R5	2.0
¦¦¦¦¦¦¦	X4	COST	16.0	R3	3.0
¦¦¦¦¦¦¦	X4	R4	4.0	R5	3.0
¦¦¦¦¦¦¦	X5	COST	2.0	R4	1.0
¦¦¦¦¦¦¦	X5	R5	1.0		
¦¦¦¦¦¦¦	X6	COST	5.0	R1	1.0
¦¦¦¦¦¦¦	X6	R5	2.0		
¦¦¦¦¦¦¦	X7	COST	1.0	R5	1.0

Field 2, Column name
Field 3, Row name Field 4, Value
Field 5, Row name Field 6, Value

Table 2. The COLUMNS Section

Note that a column is never split up with data from another column in between. Records may hold one or two entries at will; if there is only one entry then fields 5 and 6 are left blank. The order of rows within a column is immaterial.

In the RHS section right-hand side values are entered in the same way as the values of the COLUMNS section. The column name in field 2 is replaced by a name for the RHS itself. The RHS section of the example may be encoded as in Table 3 below.

Field 1 2—3	Field 2 5—12	Field 3 15—22	Field 4 25—36	Field 5 40—47	Field 6 50—61
¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦	RHS	R1	2.0	R2	5.0
¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦	RHS	R3	4.0	R4	11.0
¦ ¦ ¦ ¦ ¦ ¦ ¦ ¦	RHS	R5	14.0		

Field 2, RHS set name
Field 3, Row name Field 4, Value
Field 5, Row name Field 6, Value

Table 3. The RHS Section

In the BOUNDS section each non-zero bound value must be given singly. Field 3 here names the column (variable) to which the bound applies and field 2 names the bound-set. In field 1 a code is entered to specify the bound type. The BOUNDS section of the example may be encoded as in Table 4.

Field 1 2—3	Field 2 5—12	Field 3 15—22	Field 4 25—36	Field 5 40—47	Field 6 50—61
LO	BND	X1	0.5	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
LO	BND	X2	0.5	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
UP	BND	X3	1.0	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
FX	BND	X4	1.0	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
UP	BND	X5	6.0	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
LO	BND	X7	1.0	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦

Field 1, Bound type code

LO = Lower bound	UP = Upper bound
FX = Fixed value	FR = Free variable
MI = Minus type	PL = Plus type (default)
LI = Integer variable, lower bound	UI = Integer variable, upper bound
BV = Binary variable	SC = Semi-continuous, upper bound

Field 2, Bound set name

Field 3, Column name

Field 4, Value

Table 4. The BOUNDS Section

Note that in all the above tables, names are entered at the left of their fields while values are entered at the right. In fact values need not be coded to the right so long as the decimal point appears but it is better to stick to this convention anyway.

At the beginning of the data there is a ‘NAME’ record which has the keyword ‘NAME’ in positions 1–4 and a model name of up to 8 characters in field 3.

Each section is headed by an indicator record comprising the section name beginning at position 1.

After the last data record in the last section there is an indicator record having 'ENDATA' in positions 1–6.

The complete file of input data in MPS format is illustrated in Table 5.

Back to Chapter contents

2.1.3 Simple Use of FortMP

To execute the stand-alone FortMP system and solve this example the following preparations are made.

> The input data is placed in a file named 'MODEL.MPS'.
> A set of control commands known as 'SPECS controls' or as 'SPECS commands' is placed in a file named 'FORTMP.SPC'.

No special controls are needed to solve the simple example but the file FORTMP.SPC should contain the following two commands as a minimum:

```
BEGIN
END
```

With this all controls are set to default values and the input and output files are named by default to be

MODEL.MPS	The input data file in MPS format.
MODEL.RES	The solution output file.

With the two input files in place it is only necessary to execute the stand-alone FortMP program in order to solve the problem and obtain the outputs.

Back to Chapter contents

2.1.4 The Complete Example

The input, the SPECS controls and the solution output for the complete example set out in Section 2.1.1 are given in Table 5**,** Table 6 and Table 7 below.

```
NAME          TESTIN
ROWS
 N  COST
 EQ R1
 EQ R2
 EQ R3
 EQ R4
 EQ R5
COLUMNS
    X1        COST              4.0   R1                1.0
    X1        R2                1.0   R4                1.0
    X1        R5                1.0
    X2        COST              6.0   R2                3.0
    X2        R4                1.0   R5                2.0
    X3        COST              5.0   R3                2.0
    X3        R4                1.0   R5                2.0
```

```
    X4        COST              16.0   R3                 3.0
    X4        R4                 4.0   R5                 3.0
    X5        COST               2.0   R4                 1.0
    X5        R5                 1.0
    X6        COST               5.0   R1                 1.0
    X6        R5                 2.0
    X7        COST               1.0   R5                 1.0
RHS
    RHS       R1                 2.0   R2                 5.0
    RHS       R3                 4.0   R4                11.0
    RHS       R5                14.0
BOUNDS
 LO BND       X1                 0.5
 LO BND       X2                 0.5
 UP BND       X3                 1.0
 FX BND       X4                 1.0
 UP BND       X5                 6.0
 LO BND       X7                 1.0
ENDATA
```

Table 5. Simple example: Problem data in file MODEL.MPS

```
BEGIN
END
```

Table 6. Simple example: SPECS controls in file FORTMP.SPC

```
FORTMP SOLUTION REPORT: (TOTAL ITERATIONS=      2)
 LP: OPTIMAL
 IP: NONE
 PROBLEM NAME   = TESTIN
 OBJECTIVE NAME = COST
 RHS NAME       = RHS
 BOUNDS NAME    = BND
 MROW   =       6
 NCOL   =       7
 LP OPTIMUM VALUE =    0.420000D+02


 COLUMNS.......STRUCTURAL VARIABLES

  NO  STATE  NAME          VALUE        LOWER BND     UPPER BND    REDUCED
COST
    1   B   X1              2.             0.5         NONE           0.
    2   B   X2              1.             0.5         NONE           0.
    3   B   X3              0.5            0.           1.            0.
    4   F   X4              1.             1.           1.            6.
    5   B   X5              3.5            0.           6.            0.
    6   L   X6              0.             0.          NONE           2.
    7   B   X7              2.5            1.          NONE           0.

 ROWS.........LOGICAL VARIABLES

  NO  STATE  NAME        ROW VALUE      LOWER RHS     UPPER RHS   SHADOW PRICE
    1   B   COST           42.            NONE         NONE           1.
    2   F   R1              2.             2.           2.           -1.
```

3	F	R2	5.	5.	5.	-1.
4	F	R3	4.	4.	4.	-1.
5	F	R4	11.	11.	11.	-1.
6	F	R5	14.	14.	14.	-1.

Table 7. Simple example: Solution output

Back to Chapter contents

2.2 Simple Controls

2.2.1 The SPECS Controls

The file named FORTMP.SPC provides the FortMP system with the controls that it needs to execute a run. In principle no controls are strictly necessary provided that the input data is in MPS format and resides in the file MODEL.MPS. However, the user will most probably wish to change the filenames for a run and may need to change other defaults as well.

Controls which change the default settings are referred to as SPECS commands or as SPECS controls. Each command is presented on one line comprising one or more keywords followed, where needed, by the appropriate data. Examples are:

```
MODEL NAME (testin)
MAXIMIZE
PRESOLVE ON
FEASIBILITY TOLERANCE = 1.0d-6
INVERT FREQUENCY = 75
```

The SPECS commands are listed on the FORTMP.SPC file between the indicator lines

```
BEGIN
END
```

which delimit the active part of the file.

Back to Chapter contents

2.2.2 Controlling the Input and the Output

The input and output file names can be changed by the following commands:

```
INPUT FILE NAME (filename)
OUTPUT FILE NAME (filename)
LOG FILE NAME (filename)
```

with the desired name for each file placed between the parentheses. However, it is not usually necessary to use these commands because it is easier to use default names.

By default these filenames are

```
model.mps
```

```
model.res
model.log
```

where 'model' is a name of up to 8 characters which can itself be changed by the following command:

```
MODEL NAME (modname)
```

With this one command default names are assigned to all input and output files by adding the appropriate extension after '*modname*'. Specific names for individual files need to be changed only when necessary.

Back to Chapter contents

2.2.3 Additional Useful Controls

The following commands set the direction of optimisation for the objective function:

```
MAXIMIZE
MINIMIZE
```

The default is `MINIMIZE`

The commands given below alter the default algorithms applied in the solution.

```
PRESOLVE ON
```

The PRESOLVE algorithm examines for any possible reductions and simplifications that may be applied directly to the problem. It should be fairly easy for the user to experiment with its use in order to decide whether solution times can be improved in this way.

```
SCALE OFF
```

SCALE is an algorithm that applies factors to the rows and columns of the constraint matrix in order to reduce the extremes of variation in the values of coefficients which may cause numerical difficulties in solution algorithms.

```
ALGORITHM IPM
```

The IPM algorithm will almost always be found useful for the larger problems (1000 rows or more). However, it may still require a good understanding for setting the controls in the most efficient way depending on the problem structure and the user is referred to Chapter 5 of this manual.

```
IPM BASREC OFF
```

If IPM is used and is successful then the solution obtained is employed in a crossover procedure termed 'BASREC' (basis recovery) for the final optimisation since IPM itself does not provide a basic solution to the original problem. This can be stopped, and the IPM solution printed as the final solution, with the command 'IPM BASREC OFF'.

```
ALGORITHM PRIMAL
ALGORITHM DUAL
```

If IPM is not used then PRIMAL is the normal solution algorithm. The DUAL algorithm is employed normally to solve the subproblems created in the Branch and Bound algorithm for MIP (in an MIP problem the LP solution is obtained as a first step before Branch and Bound is called). DUAL can also

be used as an alternative to PRIMAL for the LP solution. Problems that are 'DUAL feasible' or that have a large number of rows in proportion to the number of columns benefit from the use of DUAL. After DUAL the system always reverts to PRIMAL for a final check on optimality.

The following SPECS commands are useful in modifying the output written to the logfile

```
LOG DISPLAY
```

This command causes the log (which includes progress information and statistics) to be duplicated on the standard display.

```
SIMPLEX LOG LEVEL = n
```

Where 'n' is a number 0, 1, 2, 3 or 4. The SIMPLEX iteration log is normally produced at level 1, only listing the important events such as re-inversions. At higher levels an iteration log is listed in progressively more and more detail.

```
SIMPLEX LOG FREQUENCY = n
```

The SIMPLEX iteration log is normally produced at every iteration (levels 2 or above). This would produce far too much for a large problem and so the user may reduce the frequency of output with this instruction.

Back to Chapter contents

2.3 Additional Data Preparation Features

An example: Bounded variables and RHS ranges

The following is a simple example of a problem having both bounds on the variables and ranges on the RHS.

Minimize the cost function:

$$4x_1 + 3x_2 + 2x_3$$

subject to the following constraints:

$$2 \le x_1 + 2x_2 + x_4 \le 6$$
$$8 \ge x_1 + 3x_3 \ge 4$$
$$3x_2 + x_4 = 4$$

and to the following bounds:

$$0.5 \le x_1 \le 5.0$$
$$x_2 = 2.0$$

x_3 is unbounded (free)

$x_4 \leq 0.0$ (i.e. is negative)

The ROWS section of the example may be encoded as in Table 8 below.

Field 1 **2—3**	**Field 2** **5—12**	**Field 3** **15—22**	**Field 4** **25—36**	**Field 5** **40—47**	**Field 6** **50—61**
N	COST	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
LE	R1	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
GE	R2	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
EQ	R3	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦

Field 1, Row-type:

LE = Less than or Equal GE = Greater than or Equal

EQ = Equal N = Non-binding (objective)

Field 2, Row name

Table 8. Example of ROWS Section

The COLUMNS section of the example may be encoded as in Table 9 below.

Field 1 2—3	Field 2 5—12	Field 3 15—22	Field 4 25—36	Field 5 40—47	Field 6 50—61
¦¦¦¦¦¦¦	X1	COST	4.0	R1	1.0
¦¦¦¦¦¦¦	X1	R2	1.0		
¦¦¦¦¦¦¦	X2	COST	3.0	R1	2.0
¦¦¦¦¦¦¦	X2	R3	3.0		
¦¦¦¦¦¦¦	X3	COST	2.0	R2	3.0
¦¦¦¦¦¦¦	X4	R1	1.0	R3	1.0

Field 2, Column name
Field 3, Row name Field 4, Value
Field 5, Row name Field 6, Value

Table 9. Example of COLUMNS Section

The RHS section of the example may be encoded as in Table 10 below.

Field 1 2—3	Field 2 5—12	Field 3 15—22	Field 4 25—36	Field 5 40—47	Field 6 50—61
¦¦¦¦¦¦¦	RHS	R1	6.0		
¦¦¦¦¦¦¦	RHS	R2	4.0		
¦¦¦¦¦¦¦	RHS	R3	4.0		

Field 2, RHS set name
Field 3, Row name Field 4, Value
Field 5, Row name Field 6, Value

Table 10. Example of RHS Section

The RANGES section that follows is similar in form to the right-hand side section. The value entered is

the difference between the upper and lower values for each constraint that has a range. In the example it would be encoded as in Table 11 below.

Field 1 2—3	Field 2 5—12	Field 3 15—22	Field 4 25—36	Field 5 40—47	Field 6 50—61
¦¦¦¦¦¦¦	R1	4.0			
¦¦¦¦¦¦¦	R2	4.0			

Field 2, Range set name
Field 3, Row name Field 4, Value
Field 5, Row name Field 6, Value

Table 11. Example of Ranges Section

The BOUNDS section of the example may be encoded as in Table 12 below.

Field 1 2—3	Field 2 5—12	Field 3 15—22	Field 4 25—36	Field 5 40—47	Field 6 50—61
LO	BND	X1	0.5	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
UP	BND	X1	5.0	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
FX	BND	X2	2.0	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
FR	BND	X3		¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
MI	BND	X4		¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦

Field 1, Bound type code

LO = Lower bound	UP = Upper bound
FX = Fixed value	FR = Free variable
MI = Minus type	PL = Plus type (default)
LI = Integer variable, lower bound	UI = Integer variable, upper bound
BV = Binary variable	SC = Semi-continuous, upper bound

Field 2, Bound set name

Field 3, Column name

Field 4, Value

Table 12. Example of BOUNDS Section

Back to Chapter contents

2.4 An Example: Binary and Integer Variables

For this example we shall largely repeat the example of Section 2.1.1 but with additional binary and integer constraints also considered. The problem statement becomes the following.

Minimize the cost function:

$$4x_1 + 6x_2 + 5x_3 + 16x_4 + 2x_5 + 5x_6 + x_7$$

subject to the following constraints:

$$x_1 + x_6 = 2$$
$$x_1 + 3x_2 = 5$$
$$2x_3 + 3x_4 \geq 4$$
$$x_1 + x_2 + x_3 + 4x_4 + x_5 = 11$$
$$x_1 + 2x_2 + 2x_3 + 3x_4 + x_5 + 2x_6 + x_7 = 14$$

and to the following bounds and discrete value constraints:

$0.5 \leq x_1$
$0.5 \leq x_2$
x_3 is a binary variable (zero or one)
$x_4 = 1.0$
x_5 is an integer variable in the range $0.0 \leq x_5 \leq 6.0$
x_7 is an integer variable with lower bound 1.0

Readers will note that row R3 has become an inequality — it is relaxed because otherwise the problem does not have any solution that satisfies the binary and integer constraints.

The ROWS section of the example may be encoded as in Table 13 below.

Field 1 2—3	Field 2 5—12	Field 3 15—22	Field 4 25—36	Field 5 40—47	Field 6 50—61
N	COST	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
EQ	R1	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
EQ	R2	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
GE	R3	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
EQ	R4	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
EQ	R5	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦

Field 1, Row type:

LE = Less than or Equal GE = Greater than or Equal
EQ = Equal N = Non-binding (objective)

Field 2, Row name

Table 13. Example of ROWS Section

The COLUMNS and RHS sections have not been changed in any way and remain exactly as shown in Table 9 and Table 10. There is no RANGES section.

The new BOUNDS section of the example may be encoded as in Table 14 below.

Field 1 2—3	Field 2 5—12	Field 3 15—22	Field 4 25—36	Field 5 40—47	Field 6 50—61
LO	BND	X1	0.5	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
LO	BND	X2	0.5	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
BV	BND	X3		¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
FX	BND	X4	1.0	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
UI	BND	X5	6.0	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦
LI	BND	X7	1.0	¦¦¦¦¦¦¦¦	¦¦¦¦¦¦¦¦

Field 1, Bound type code

LO = Lower bound	UP = Upper bound
FX = Fixed value	FR = Free variable
MI = Minus type	PL = Plus type (default)
LI = Integer variable, lower bound	UI = Integer variable, upper bound
BV = Binary variable	SC = Semi-continuous, upper bound

Field 2, Bound set name

Field 3, Column name

Field 4, Value

Table 14. Example of BOUNDS Section

Having prepared the data there are no SPECS controls to consider other than giving a name to the model with the 'MODEL NAME' command. We shall use the name BININT for this example.

To execute the stand-alone FortMP system and solve this example the following preparations are made.

The input data is placed in a file named 'BININT.MPS'.
'SPECS controls' are placed in the file named 'FORTMP.SPC' as before.

No special controls are necessary because the Mixed Integer algorithm is invoked automatically by the presence of discrete value constraints in the problem. The file FORTMP.SPC contains the three commands

```
BEGIN
MODEL NAME (BININT)
END
```

All other controls are set to default values and the input and output files are named as:

BININT.MPS	The input data file in MPS format.
BININT.RES	The solution output file.

With these files in place it is only necessary to call execution of the stand-alone FortMP program in order to solve the problem and obtain the outputs. The input, the SPECS controls and the solution output for this example are given in Table 15, Table 16 and Table 17 below.

```
NAME          BININT
ROWS
 N  COST
 EQ R1
 EQ R2
 GE R3
 EQ R4
 EQ R5
COLUMNS
    X1        COST              4.0   R1                1.0
    X1        R2                1.0   R4                1.0
    X1        R5                1.0
    X2        COST              6.0   R2                3.0
    X2        R4                1.0   R5                2.0
    X3        COST              5.0   R3                2.0
    X3        R4                1.0   R5                2.0
    X4        COST             16.0   R3                3.0
    X4        R4                4.0   R5                3.0
    X5        COST              2.0   R4                1.0
    X5        R5                1.0
    X6        COST              5.0   R1                1.0
    X6        R5                2.0
    X7        COST              1.0   R5                1.0
RHS
    RHS       R1                2.0   R2                5.0
    RHS       R3                4.0   R4               11.0
    RHS       R5               14.0
BOUNDS
 LO BND       X1                0.5
 LO BND       X2                0.5
 BV BND       X3                1.0
 FX BND       X4                1.0
 UI BND       X5                6.0
 LI BND       X7                1.0
ENDATA
```

Table 15. Problem Data File

```
      BEGIN
      MODEL NAME (BININT)
      END
```

Table 16. Problem SPECS File

```
FORTMP SOLUTION REPORT: (TOTAL ITERATIONS=      3)
 LP: OPTIMAL
 IP: OPTIMAL
 PROBLEM NAME   = BININT
 OBJECTIVE NAME = COST
 RHS NAME       = RHS
 BOUNDS NAME    = BND
 MROW   =       6
 NCOL   =       7
 LP OPTIMUM VALUE =    0.420000D+02
 IP OPTIMUM VALUE =    0.430000D+02

 COLUMNS.......STRUCTURAL VARIABLES

  NO  STATE  NAME          VALUE        LOWER BND     UPPER BND    REDUCED
COST
    1   B   X1               2.             0.5         NONE           0.
    2   B   X2               1.             0.5         NONE           0.
    3   F   X3               1.             1.           1.            2.
    4   F   X4               1.             1.           1.            9.
    5   B   X5               3.             0.           6.            0.

    6   L   X6               0.             0.          NONE           2.
    7   B   X7               2.             1.          NONE           0.

 ROWS..........LOGICAL VARIABLES

  NO  STATE  NAME        ROW VALUE      LOWER RHS     UPPER RHS   SHADOW PRICE
    1   B   COST            43.            NONE         NONE           1.
    2   F   R1               2.             2.           2.           -1.
    3   F   R2               5.             5.           5.           -1.
    4   B   R3               5.             4.          NONE           0.
    5   F   R4              11.            11.          11.           -1.
    6   F   R5              14.            14.          14.           -1.
```

Table 17 ‘Problem Output’

Back to Chapter contents

2.5 Summary of SPECS File Controls

The following SPECS commands have been introduced so far in this manual. Those introduced in this chapter are underlined thus

BEGIN

`BEGIN`

This must be the first command line.

`MODEL NAME (`*`modname`*`)`

This command supplies a default for the names of input files and output files in the run which are built by adding an extension as follows.

MODNAME.MPS	Input data.
MODNAME.RES	Output results.
MODNAME.LOG	Log file and messages.

The default model name is 'MODEL'.

`INPUT FILE NAME (`*`filename`*`)`

This command assigns a special name to the problem data input file.

`OUTPUT FILE NAME (`*`filename`*`)`

This command assigns a special name for the output file.

`LOG FILE NAME (`*`filename`*`)`

This command assigns a special name for the log file.

`MINIMIZE`

This command specifies that the objective is to be minimised. NB: 'MINIMIZE' is the default so this command is not actually needed.

`MAXIMIZE`

This command specifies that the objective is to be maximised.

`PRESOLVE ON`

This command activates a 'PRESOLVE' operation prior to optimisation.

`SCALE OFF`

This command cancels initial scaling of the data.

`ALGORITHM PRIMAL`
`ALGORITHM DUAL`
`ALGORITHM IPM`

One of these commands is used to specify the primary solution algorithm for the continuous LP problem. 'PRIMAL' is the default and need not be specified.

`FEASIBILITY TOLERANCE =` *`value`*

This command assigns a tolerance used in the algorithms. The default for 'value' is 1.0d-5. Other tolerances are introduced in Chapter **Error! Reference source not found.** of this manual.

`INVERT FREQUENCY =` *`n`*

This command assigns a frequency, in terms of the iteration count, for reinversion during SIMPLEX algorithms. The default for '*n*' is 50.

`SIMPLEX LOG LEVEL = n`

This command determines in what level of detail to print the iteration log during SIMPLEX algorithms. The default for '*n*' is 1 giving only outstanding events, but higher levels also give details of each iteration

`SIMPLEX LOG FREQUENCY = n`

This command assigns a frequency for printing the iteration log during SIMPLEX algorithms. The default for '*n*' is 1.

`IPM BASREC OFF`

When 'ALGORITHM IPM' has been selected the user can use this command to halt execution and print the output immediately on reaching the IPM solution.

`LOG DISPLAY`

This command causes all logged output, not just important diagnostic messages, to be duplicated on the online display (or written to the standard output stream).

`END`

This command terminates the SPECS commands. Anything beyond it is ignored. \

Back to Chapter contents

3. Overview of the Stand-alone FortMP System

Contents

3.1 Introduction and Overview

3.1.1 The Algorithms Employed by FortMP

FortMP is a highly advanced system for the solution of Linear Programming (LP) and related problems. It can solve problems of all sizes and has special features which enable it to find a solution even in the case of degeneracy and numerical difficulties.

There are three main classes of algorithm employed by FortMP.

> The Sparse Simplex (SSX) algorithms comprises PRIMAL and DUAL. Both of these algorithms employ INVERT.
>
> The Interior Point Method (IPM) algorithms are based on the Primal Dual logarithmic barrier method.
>
> A Branch and Bound technique for the solution of Mixed Integer (MIP) problems allows Binary, Integer, SOS1, SOS2, and Semi-continuous variable types and a variety of choice strategies.

In addition there is a full supporting set of minor algorithms and procedures enabling user to achieve the most efficient solution possible. Examples are:

> SCALE
> PRESOLVE
> CRASH
> BASREC (IPM-SSX crossover)
> MIP pre-processor
> SAVE and RESTART features
> BASIS input and output

Input data is normally presented in the standardised MPS format with extensions for MIP variable types. Other formats are also available.

The stand-alone version of FortMP begins by reading a set of control commands (SPECS) supplied by the user to define which algorithms are to be used and to supply any special tolerances, weights or other necessary parameters needed to control the run. However, the user does not need to supply all such controls as the system will use defaults that have been judged as best for a variety of problem types.

Back to Chapter contents

3.1.2 Sparse Simplex (SSX): PRIMAL, DUAL and INVERT

The revised SSX algorithms comprising PRIMAL and DUAL together with the INVERT procedure, which is used in both algorithms, form the central core of the system. By default PRIMAL is used for the solution and PRIMAL is always used to complete the solution process and verify that any other solution (e.g. that supplied by DUAL) is indeed a valid solution before terminating.

A rather simplified view of the SSX algorithms is given here; the reader will find a fuller discussion in Chapter 4.

SSX proceeds via a progression of 'basic' solutions. In a basic solution there is a set of *M* 'basic' variables (where *M* is the row size of the problem including the objective row) and all other variables are fixed either to the lower bound or to the upper bound.

One step in the progression, or one 'iteration', exchanges one basic for one non-basic variable and this proceeds until one of the following three outcomes takes place.

The problem is shown to have no feasible solution (infeasible).
It is shown that the objective can be improved without limit (unbounded).
An optimum solution is found.

Knowledge of the basis enables procedure INVERT to provide a set of factors termed 'ETA vectors' with which the current solution is calculated. During PRIMAL or DUAL iterations the list of ETA vectors is extended, zero, one or two being added for each iteration. This growth of the ETAs requires that INVERT be repeated periodically to reduce the storage used. INVERT also reduces the volume of calculation and improves accuracy.

Back to Chapter contents

3.1.3 Interior Point Method (IPM) — When and When Not to Use

IPM is an entirely different approach from SSX which does not proceed via a progression of 'basic' solutions. Instead it proceeds via a progression of 'interior' points – an interior point being a set of values that satisfy the bounds and ranges but are not necessarily feasible in the sense of satisfying the equations.

One iterative step proceeds by choosing a direction, a set of changes to apply to the variable values, and going a certain distance in that direction without causing any bound or range to be violated so that the point remains 'interior'. The direction is chosen in such a way as to ensure that the total infeasibility is reduced and that there is progress towards an optimal objective value.

In most industrial optimisation problems the optimum solution is by no means unique. The optimum solution obtained by IPM usually differs from one obtained by SSX in not being a 'basic' solution. Only in rare text book examples does the IPM optimum have exactly *M* variables with intermediate values between lower and upper bound.

This is a disadvantage to many users because a non-basic solution cannot be used for analysis, nor can it provide an advanced basis for the solution of another problem. Hence in FortMP there is an algorithm called BASREC which does the job of recovering a basic optimum solution from the non-basic IPM solution. As with the DUAL algorithm, PRIMAL is called to apply the final verification before the solution is accepted.

The other main difference between SSX and IPM is that with IPM the number of iterations required is a function of the problem's complexity rather than its size. Experience shows that, for similar problems, IPM solution times grow in a nearly linear fashion with problem size while SSX solution times grow quadratically because both the number of iterations and the time per iteration increase linearly with

problem size. Thus there is a certain size beyond which it usually pays the user to employ IPM. Problems with 1000 or more constraint rows can probably be solved more quickly with IPM although users may need to experiment in order to find the best set of control parameters to suit their model structures.

If you do not need a basic solution and can be satisfied with the IPM solution then the system can be halted at that point reducing the solution time even further.

Back to Chapter contents

3.1.4 Mixed Integer (MIP) with the Branch and Bound Method

Before entering the MIP solver (Branch and Bound) the system must obtain the basic optimum solution to the continuous problem — that is the original problem with all integer and other discrete constraints ignored.

The continuous optimum forms the root node of a branching tree which is then developed by the MIP algorithm. At each node (starting with the root node) some MIP variable is chosen which as yet does not have a legal discrete value. Two branches are formed with the variable restricted one way on the first branch and the opposite way on the second branch. This restriction works differently for different MIP variable types.

A binary variable is restricted to zero on one branch, and to one on the other branch.

An integer variable having a value between two integers K and $K+1$ is restricted to an upper bound of K on one branch and a lower bound of $K+1$ on the other branch.

An SOS is divided in two contiguous subsets. On one branch the lower set is restricted all to zero and on the other branch the upper set is restricted all to zero.

A semi-continuous variable is restricted to zero on one branch and to its continuous range on the other branch.

It is clear that by progressively branching in this manner and solving the subproblems so obtained to derive new nodes on which to branch again, the algorithm can eventually reach all possible integer solutions and select the best. However, this would be impossibly long. The algorithm succeeds in a reasonable length of time by 'bounding', that is by cutting off any branch with a solution value worse than the current best available integer solution. No bounding is possible until an integer solution is known and so the initial policy is to continue down one branch to the first integer solution in order to find a bound quickly.

Each subproblem is solved by using the SSX solver. The bounding process works because no subproblem can have a better solution value than its parent node since an extra restriction has been added.

Back to Chapter contents

3.1.5 Overall Structure of the System

Figure 1. below illustrates in simplified form the block structure of the system.

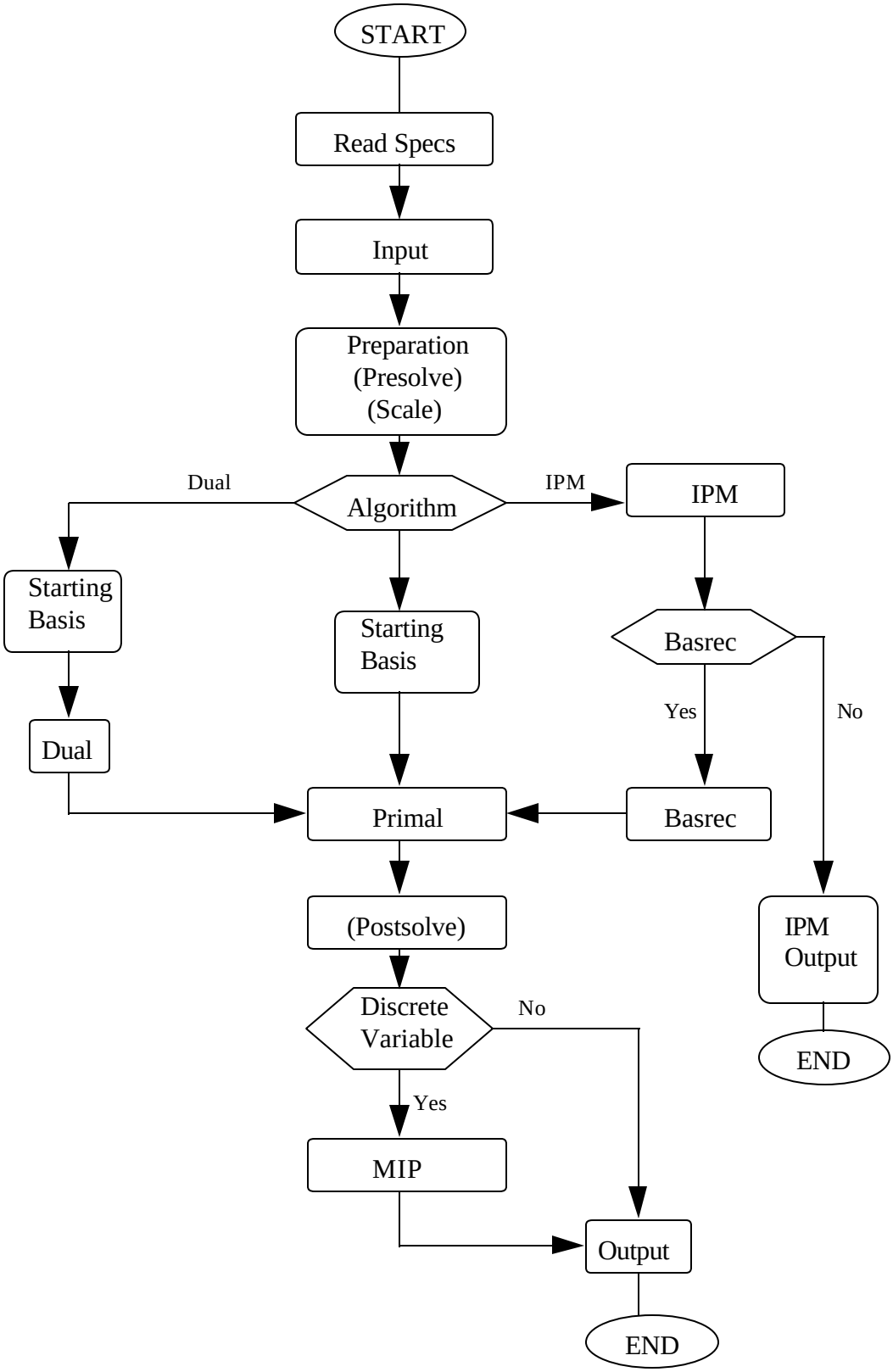


Figure 1. Block structure of the system

As can be seen, the execution path through the system is controlled by three main switches.

The ALGORITHM switch selects one of the three main procedures PRIMAL, DUAL or IPM to employ for the LP solution.

The BASREC switch is only active after IPM and causes BASREC to be executed followed by PRIMAL. If it is BASREC is OFF then IPM is followed immediately by IPM OUTPUT and the run terminates.

The 'Discrete Variables' or MIP switch causes the MIP algorithm, Branch and Bound, to be executed. The MIP switch is always set OFF if the problem has no MIP variables.

Back to Chapter contents

3.1.6 Providing Controls on the SPECS File

At the outset FortMP reads all of the control commands that it needs for the run from the file named 'FORTMP.SPC'. Every control has a default value so that in the simplest case the user need not provide SPECS commands.

Each SPECS command is presented on one line and consists of one or more keywords followed by a value which, depending on the command, may be either:

a name or text string enclosed in parentheses,
a numerical value, optionally after an '='sign, or
a switch ON or OFF (if omitted then ON is assumed).

Examples are:

```
MODEL NAME (testin)
MAXIMIZE
PRESOLVE ON
INVERT FREQUENCY = 75
```

Additional examples have been given in Chapter 2, Section 2.2.

Details of the syntax of every SPECS command are given in the Appendix B.

All keywords may be abbreviated to the first four or more letters and the first keyword (only) may be abbreviated to the first three letters. The asterisk character (*) acts as a line terminator so that the user may include comments to the right. Users may also include whole comment lines by putting an asterisk in column 1. The above commands could be written as:

```
MOD NAME (testin)
* Filenames are testin.mps, testin.res, testin.log
MAX
PRE
INV FREQ 75                    * NB: default is 50
```

However, for the sake of clarity it is better not to abbreviate too much.

The system does not use every line in the SPECS file regardless, but only that part of the file which is delimited between the two indicator commands 'BEGIN' and 'END'. The complete example without abbreviation would be

```
BEGIN
MODEL NAME (testin)
MAXIMIZE
PRESOLVE ON
INVERT FREQUENCY = 75
END
```

A comment is added to show the range of allowable values (if needed) and the default (where it applies). In Section 3.8 there is a summary of all the commands introduced so far.

Back to Chapter contents

3.2 Data Preparation

3.2.1 MPS Format

The standard means for encoding external problem data is MPS format which has already been described in Chapter 2, Section 2.1.2 with further examples in Sections 2.3 and 2.4. Here we give some further rules and examples. A fully detailed specification is given in Appendix A.

MPS format data is invoked by the SPECS command

```
INPUT TYPE MPS
```

However, this is the standard default and need not be given.

Comment Lines

Any line beginning with an asterisk (*) in column 1 is ignored by the input.

The Format of a Name

A name is 8 characters long and may contain any letter, digit or special character, upper and lower case being considered different. Leading blanks are ignored but internal blanks are counted – for example the following names are all different:

```
ROW1 11
ROW111
row1 11.
```

Alternative Row Type Codes

Alternative codes, such as 'G' or '>' in the place of 'GE' may be used for row types in the ROWS section. See Appendix A, section A1.1 for details.

Rhs Range Coding

In the RANGES section all GE type and LE type rows have only positive or zero range values. A zero range value on these row types implies that the row is changed in type to become an EQ type row.

Rows defined as EQ type may have positive or negative range values which effectively change the row type as follows.

A positive range implies row type GE
A negative range implies row type LE

Consider the following example row with an RHS range:

$$10.0 \leq \sum a_{ij} x_j \leq 20.0$$

This can be coded in four alternative ways.

1) ROWS : Use type LE
RHS : value 20.0
RANGES : value +10.0

2) ROWS : Use type GE
RHS : value 10.0
RANGES : value +10.0

3) ROWS : Use type EQ
RHS : value 10.0
RANGES : value +10.0

4) ROWS : Use type EQ
RHS : value 20.0
RANGES : value -10.0

Multiple Rhs Sets, Range Sets and Bound Sets

An important option for the user is to provide several alternative models in the same data file. This is done by making use of multiple sets in the RHS, RANGES and BOUNDS sections of the data with distinct names in field 2 of the input records.

The following SPECS commands specify the selection of particular sets:

```
RHS NAME (rhsname)
RANGES NAME (rngname)
BOUNDS NAME (bndname)
```

By default the first set to appear will be selected for any section where the command is not given.

These sets follow the same rule as for columns in the COLUMNS section; a set may not be split. All the data for any one set must be presented consecutively. In addition no set name should be entirely blank because 'blank' is used internally to select the default. It is possible for a set to be empty, i.e. to have no data records at all with that set name.

Objective Row Selection

The following SPECS command names the row to be used as objective row in the problem data:

```
OBJECTIVE NAME (objname)
```

where '*objname*' represents the name of a free row in the model. By default the system selects as objective row the first row to be defined with type FREE (code 'N') in the ROWS section.

Back to Chapter contents

3.2.2 Matrix Generator Format

Problem data prepared for input to FortMP by a matrix generator (MG) such as MPL is quite different from MPS format data. The objective of such data is to achieve a more efficient interfacing between FortMP and the MG in which external matters, such as names, are handled by the MG and not by FortMP.

The MG is expected to act in conjunction with a Report Writer (RW) which reads a specially formatted solution output from FortMP and identifies the rows and columns on the basis of index number.

Detailed layouts for MG input and RW output are given in Appendix A, section A3.1. To invoke this form of data input the following SPECS command is used:

```
INPUT TYPE MG
```

Back to Chapter contents

3.2.3 Free Format MPS and Long Data Names

The standard MPS format of data is based upon 'Fixed form' data entry under which every field is positioned at a given place in the input data line. Ultimately this derives from data entry on punched cards where the fixed form is natural.

In modern data entry it is not natural for an operator to use the keyboard in this way. Rather the data is seen as a series of items or 'tokens' to be entered in a particular order, where the tokens correspond to the fields of fixed form data entry. Data 'lines' replace 'cards', these being separated by new-line entries on the keyboard, and the tokens are separated from each other by using certain special characters that cannot be used in a token.

In Free Format MPS data input for FortMP the first field (positions 2-3) remains in fixed form, completely unchanged. Fields 2 to 6 from position 5 to the end of a line are replaced by tokens with blank spaces acting as separator. Any number of spaces are equivalent to a single space so that the data may be 'padded' out with blanks to arrange in columns for better viewing. In fact most ordinary fixed-form MPS data files are also quite valid as free format data files provided the unused gaps between fields are blank and provided there are no blank names or names with interior blank spaces.

Free format data is invoked with the SPECS command:

```
INPUT TYPE FREE
```

which must be given as the default type is 'MPS'. Some additional points are to be noted:

Comment Lines

Any line beginning with asterisk (*) in position 1 is ignored by the input.

Row and Column Names

Row and column names are normally restricted to 8 characters. This can be enlarged to 16 characters with the following SPECS command:

```
INPUT LONG NAMES ON
```

where 'OFF' is the default. Other types of name such as problem name, RHS/Range/Bound set names etc. remain limited to 8 characters.

Heading Keywords

The following are special heading keywords as in Fixed-form MPS:

NAME	RHS	ENDATA
ROWS	RANGES	
COLUMNS	BOUNDS	

and these must always be entered starting at position 1 of a line with no preceding blanks.

Input type with Long Names

When INPUT LONG NAMES ON is specified the type of input data is necessarily free format MPS. Long names must therefore conform to the rules for tokens (i.e. no interior blank spaces).

Back to Chapter contents

3.2.4 Other Types of Input Data

Various alternative input features have been provided in the FortMP system. Details of these mechanisms are given in Appendix A (or for special reasons may be supplied in separate documents).

Back to Chapter contents

3.3 Running the System

3.3.1 Using SSX

The SSX algorithms are invoked for LP with the SPECS commands

```
ALGORITHM PRIMAL
ALGORITHM DUAL
```

PRIMAL is the default so 'ALGORITHM PRIMAL' need not be specified. In fact DUAL is followed anyway by PRIMAL to verify the optimal solution.

A full description of SSX is given in Chapter 4, but the user may find the following commands useful without reading further.

```
SIMPLEX LOG LEVEL = n          * n = 0-4, default 1
INVERT LOG LEVEL = n           * n = 0-4, default 1
SIMPLEX LOG FREQUENCY = n      * default n=1
```

These commands determine the nature and frequency of outputs to the log file (frequency refers to the iteration log). Log level zero is limited to error messages only.

```
INVERT FREQUENCY = n           * default n=50
```

This sets a maximum to the number of iterations allowed before an automatic reinvert is invoked. However this applies at the start only, as the system increases the number when Forrest-Tomlin update begins (see Chapter 4, section 4.6.1).

```
MAXIMUM SIMPLEX ITERATIONS = n     * default n=50000
```

After this number of iterations the SSX algorithms save a basis for restart and then terminate (see Sections 3.5.1 and 4.6.4 on saving and restarting).

Back to Chapter contents

3.3.2 Setting up a Starting Basis for SSX

As already stated, a starting basis must exist before either PRIMAL or DUAL can execute. The following alternative mechanisms are available for this.

The starting basis can be built by the CRASH algorithm.
The starting basis can be read from an external file.
The user can use RESTART, which employs SAVE information from a previous run. A basis is included as part of the saved data.
The user can start from the all logical basis, also known as the UNIT basis.

The method chosen is according to a 'SIMPLEX START' command which may be one of the following:

```
SIMPLEX START CRASH            * This is the default
SIMPLEX START INPUT BASIS
SIMPLEX START RESTART
SIMPLEX START UNIT BASIS
```

A description of the INPUT BASIS mechanism is given in Section 3.5.3 on BASIS input and output. Details of the MPS format layout of an external basis are given in Appendix A, section A1.3.

The SAVE/RESTART mechanism is described in Section 3.5.1.

When the starting basis has been installed the FortMP system calls INVERT and calculates the basic solution so that PRIMAL or DUAL can then execute directly.

Back to Chapter contents

3.3.3 Using IPM

The IPM algorithms are invoked with the command:

```
ALGORITHM IPM
```

A variety of controls direct which specific IPM algorithms are to be used and supply parameters. To use these controls the user should read the descriptions given in Chapter 5 of this manual.

The following SPECS commands determine the log level of IPM and the execution limit in the same manner as for PRIMAL and DUAL.

```
IPM LOG LEVEL = n              * n = 1-4, default 1
MAXIMUM IPM ITERATIONS = n     * default n=80
```

After this number of iterations the IPM algorithm saves the current solution for restart and terminates (see Section 3.5.1 on saving and restarting).

The following SPECS command is only available in certain versions of FortMP. It invokes a graphical display which shows the density pattern of the matrices and the progress of the iterations. Copies of the display will be left in files '*model*.a', '*model*.b', '*model*.c' and '*model*.d' if this command is successfully activated.

```
IPM GRAPHICAL DISPLAY ON
IPM GRAPHICAL DISPLAY OFF      * This is the default
```

After execution of IPM the system has a choice: to stop there and print the solution or to recover a basis from the IPM solution and use it as an advanced starting basis to the SSX solver.

If the basis recovery option is taken then the printed solution will be a basic optimum (if that is possible) whereas the IPM solution is only rarely basic. Basis recovery is the default option of the system; to stop with the IPM solution the following command is used:

```
IPM BASREC OFF
```

Back to Chapter contents

3.3.4 Running Mixed Integer

Before starting the Branch and Bound algorithm for MIP problems, the system must obtain a basic optimum solution to the LP problem, that is the problem with all MIP variables allowed their continuous range. Either SSX or IPM can be used for this purpose; if IPM is used the BASREC switch must be ON (which is the default).

Entry to Branch and Bound is normally automatic, dependent only on whether the problem contains any MIP variable types. However, the user can prevent the system from entering Branch and Bound, for example to give time for examining the LP optimum, by setting the MIP switch to OFF as follows:

```
MIP OFF
```

An important aid to finding the solution for MIP problems is the pre-processor which is invoked with the following SPECS command:

```
MIP PREPROCESS ON
```

(default is OFF). Not every problem is improved with the pre-processor, however, and it is most effective when the problem has binary or integer variables. It is worthwhile for the user to experiment in order to determine whether to use it as a normal method for all his problems.

Another useful command, important for many MIP problems is the following:

```
MIP PRIORITY UP ON
```

(default is OFF). This is very useful for problems related to scheduling and resource allocation for reasons explained later in chapter 6 of this manual.

A variety of other controls direct the branching choices in MIP and supply parameters. To find out about these controls the user should read the descriptions given in Chapter 6 of this manual.

The following SPECS commands determine the log level of MIP and the execution limit in the same manner as for PRIMAL and DUAL.

```
MIP LOG LEVEL = n              *  n = 0-4, default 1
NODE LOG FREQUENCY = n         *  default 1
MAXIMUM MIP INTEGER = nnn      *  default 300
MAXIMUM MIP INTSOL = nnn       *  default 300
MAXIMUM MIP NODES = nnn        *  default 50000
MAXIMUM MIP TIME = vvv         *  default 50000.0
```

Limits are applied to the number of integer solutions reached ('INTEGER' and 'INTSOL' are the same), to the number of nodes (i.e. sub-problems) to be solved, and to the total time (given in seconds) that may be spent executing the MIP algorithm. At each limit the MIP algorithm saves the current tree for restart and terminates (see Section 3.5.1 on saving and restarting).

Back to Chapter contents

3.3.5 Using the PRESOLVE Algorithm

PRESOLVE is an algorithm which can often simplify the problem internally and reduce the time needed to reach the solution. If employed, it takes place after input and before execution of any other algorithm.

The following SPECS commands invoke or cancel execution of PRESOLVE:

```
PRESOLVE ON
PRESOLVE OFF                   * This is the default
```

PRESOLVE seeks any simplifications that can be made to the problem by looking for the obvious solutions implicit in the constraints as they stand. Elimination of redundant rows and fixed variables reduces the size of the problem to be solved and thereby improves efficiency.

For example, if all fixed variable values are substituted in the constraints one or more equations may be left with only one variable. These variables are then fixed, the equations become redundant and their row types are made free. The substitution of the newly fixed variables may then lead to more simplifications of the same kind.

This is a simple 'Level 1' technique; other more complex techniques designated as 'Level 2', 'Level 3' and so on are available. The user can select a maximum level to use, once the PRESOLVE switch has been set ON, by the following command:

```
PRESOLVE LEVEL = n              *   n=1-5, default is 5
```

At solution time there is an option to print the presolved solution as it stands or to apply POSTSOLVE which expands back to the original problem in order to obtain a basic solution (similar to using BASREC after IPM). This expansion is invoked or cancelled with the following SPECS commands:

```
POSTSOLVE ON                    * This is the default
POSTSOLVE OFF
```

The amount of information sent to the log during execution of PRESOLVE and POSTSOLVE can be controlled with the following command:

```
PRESOLVE LOG LEVEL = n          *   n=0-4, default is 1
```

Back to Chapter contents

3.3.6 Using the SCALE Algorithm

The SCALE algorithm finds a set of factors, one per row and one per column, which reduce the variation in data values so that the solution process is numerically more stable. The resulting scaled solution values are unscaled before the final output.

SCALE normally takes place after INPUT (or after PRESOLVE) and before execution of any algorithm. It is invoked or cancelled with the following SPECS commands:

```
SCALE ON                        * This is the default
SCALE OFF
```

Two further SPECS commands control the operation of SCALE when it is switched ON:

```
SCALE PASSES = n                * n=1-4,  Default is 4
SCALE VARIANCE = v              * Default v = 10.0
```

Each 'pass' of SCALE is an attempt to reduce the variation of the matrix coefficients, starting from the previous results. If the given target variance is reached SCALE ends immediately, otherwise it continues for the given maximum number of passes.

Quite often the variance does not reduce after the first pass and may on the contrary increase again. Results are given on the log file and the user may use these to determine the best values for the commands.

Back to Chapter contents

3.4 Output Descriptions

3.4.1 Standard Output Description

Table 17 on page II-19 shows the standard output from the simple example of Chapter 2. There the heading gives details of the model and of the solution that follows. This is followed by a table of column values showing each structural variable in the problem. Outputs under each heading are:

NO	Index number of the variable
STATE	Codes for the primal and dual status ('B' means BASIC; 'F' means FIXED; 'L' means at Lower bound; 'U' means at Upper bound)
NAME	Name of variable as given in the input
VALUE	Solution value
LOWER BOUND	Lower bound as given in the data
UPPER BOUND	Upper bound as given in the data
REDUCED COST	Reduced cost or 'DJ' of the variable

When an integer (IP) solution is printed the lower and upper bound values that are printed reflect the branching applied to each variable and not necessarily the original bounds.

The next section is a table of row values showing each constraint (or logical variable) in the problem. Outputs under each heading are:

NO	Index number of the row
STATE	Codes for the status of the slack variables ('B' means BASIC; 'F' means FIXED; 'L' means at Lower bound; 'U' means at Upper bound)
NAME	Name of the row as given in the input
ROW VALUE	Value of the constraint left-hand side i.e. the value of $\sum a_{ij}x_j$
LOWER RHS	Lower bound of $\sum a_{ij}x_j$
UPPER RHS	Upper bound of $\sum a_{ij}x_j$
SHADOW PRICE	Shadow price of the row (equal to the reduced cost of the logical)

Standard output is invoked by the following SPECS command:

```
OUTPUT TYPE STD
```

However, this command is the default and need not be given.

Back to Chapter contents

3.4.2 Report Writer Output

Output for a Report Writer such as MPL is a simplified form of standard output. Headings are not printed and the initial heading material is compressed to a single record.

Names are not printed, only the index number of each column and row.

The layout comprises FORTRAN formatted records but is designed for easy input when using C. For details see Appendix A, section A3.2.

The RW format of output is invoked with the following SPECS command:

```
OUTPUT TYPE RW
```

Back to Chapter contents

3.4.3 Log File Output

Each major algorithm of FortMP is associated with a log 'level' which controls the information written to the log file. There are five levels which can be applied in each case :

Level 0	Error messages only
Level 1	Errors and important messages only.
Level 2	Important messages and a primary log (e.g. iteration log).
Level 3	A more detailed log (if relevant) and useful general statistics.
Level 4	Level 3 plus more detailed statistics.

Variations on this general scheme and other special features are described in the following chapters devoted to the individual algorithms. Log file examples are given with explanations if needed.

The following SPECS commands specify the log levels for each algorithm:

```
SIMPLEX LOG LEVEL = n          * n = 0-4, default is 1
INVERT LOG LEVEL = n           * n = 0-4, default is 1
IPM LOG LEVEL = n              * n = 0-4, default is 1
MIP LOG LEVEL = n              * n = 0-4, default is 1
```

A duplicate log can also be displayed when running on-line in interactive mode. This is invoked with the following commands:

```
LOG DISPLAY
LOG DISPLAY LEVEL = n          * n = 0-4, default is 1
LOG DISPLAY ONLY
```

The system copies some of the logged messages onto the standard output (or the display screen when running on-line). In default the levels copied are 0 and 1 (errors and main events). With the LOG DISPLAY command the maximum level copied in this way can be changed to a specific level so as either to display more or to reduce the display (errors cannot be suppressed). If 'LEVEL = n' is omitted then all messages are copied - this is equivalent to 'LEVEL = 4'. If 'ONLY' is specified then the log file itself is suppressed and all logs are written solely to the standard output. These commands do not change the level of messages actually issued in the first place.

Back to Chapter contents

3.4.4 Suppressing Output

It frequently happens that the entire output is not needed by the user and for large problems in particular it can be useful to reduce it by eliminating all variables with solution value zero. If the user does not need to know the dual solution values this means that output is reduced simply to those variables with a non-zero primal solution value.

This is achieved with the following SPECS command:

```
OUTPUT SUPPRESS ZERO          *  default: not suppressed.
```

In addition it is possible to suppress the output altogether - for example to restart and print in a later run - with the following command

```
OUTPUT OFF                    * default ON
```

(ON may also be given).

Back to Chapter contents

3.5 Further Topics

3.5.1 Saving and Restarting

It is inevitable that some users will need to experiment or will have some difficulties in determining how best to apply the system to their problems. In some cases a run may exceed its time or memory allocation so the user has to make adjustments and rerun the problem.

FortMP is provided with several save and restart features which enable a rerun to begin from a point close to the termination of the previous run, thus avoiding a waste of computer time. The save features are as follows.

INPUT save	The input data is saved after an initial pass in which all names are identified and indexed. The internal data file has all names replaced by indices so that a much faster INPUT can be used in a restart.
SIMPLEX save	A periodic save of the basis takes place during the SSX algorithms. The saved basis can form an advanced starting basis in a restart.
IPM save	Periodically during IPM iterations the solution is saved and this solution can form an advanced initial solution point in a restart.
MIP save	Periodically during Branch and Bound the tree is saved and can then be used to bypass previous node development in a restart.

The associated SPECS commands comprise controls to specify the frequency of saving and restart switches to specify where any restart is to apply. These commands are:

```
SIMPLEX SAVE FREQUENCY = n  * default 10, each 10th
reinvert
SIMPLEX START RESTART
```

```
IPM SAVE FREQUENCY = n      * default 10, each 10th
iteration
IPM RESTART ON
MIP SAVE FREQUENCY = n      * default n=500, each 500th
node.
MIP RESTART ON
```

By setting the frequency n to zero in one of the above SAVE commands the corresponding SAVE feature is switched off entirely.

Users must be aware that certain internal files from the previous run are needed in a restart, not just the file created by the particular kind of restarts invoked. The original controls from the previous run are always reread and apply as defaults in the restart run.

Back to Chapter contents

3.5.2 Bypassing the Initial MPS Input

It is not necessary to specify any 'SAVE' for the original MPS data. Input data is saved in an internal form during an initial pass when reading MPS format data. With the command

```
INPUT RESTART ON
```

the initial input pass which looks up all names and creates the data indexing is bypassed.

Back to Chapter contents

3.5.3 BASIS Input and Output

There is a mechanism whereby the user can supply an initial basis in MPS format, using the external names for the variables.

In this format the input basis data comprises:

NAME record
BASIS header record (optional)
BASIS data section
ENDATA record

The BASIS data section has records of type XL, XU, LL and UL. The rationale behind this is that each record of type XL or XU gives an exchange between a column variable (structural) which becomes basic and a row variable (logical) which becomes non-basic. Initially all row variables are basic and all column variables are non-basic at lower bound. The UL and LL records simply name non-basic columns and only UL records actually need to be entered.

An external basis is intended for use even when the input problem has been revised so that a variety of inconsistencies in MPS format basis data are allowed. For example if a name is not recognised then a warning message is given on the log but the system carries on by simply ignoring that record.

Precise details of the layout are given in Appendix A, section A1.3.

The following SPECS command (already introduced in Section 3.3.2) is used to input the external basis and set it up as the starting basis for PRIMAL or DUAL.

```
SIMPLEX START INPUT BASIS
```

There is a SPECS command to create an output MPS format basis when the LP optimum is reached after completing the SSX algorithm. This command is an ON/OFF switch as follows.

```
OUTPUT BASIS ON
OUTPUT BASIS OFF              * This is the default
```

The switch ON creates output to a file with extension BAS (model.bas) and this file can subsequently be used for the 'SIMPLEX START BASIS INPUT' command. Note however that both BASIS INPUT and OUTPUT BASIS can co-exist is the same run and that OUTPUT BASIS may overwrite the previous input file. To overcome this problem, the user should copy the file between runs, or rename the file and use a BASIS FILE NAME command as described in Section 3.6.

Back to Chapter contents

3.6 Input and Output Files in FortMP

For the convenience of UNIX and other operating systems which distinguish lowercase letters from uppercase, FortMP uses lowercase lettering for all default filenames and extensions in the system. We will follow this convention in this section.

As already explained, the MPS format data is presented to the program on a file named

model.mps

where 'model' is the name of the data model. You can specify the name of the model with the following SPECS command:

```
MODEL NAME (mdlname)
```

where 'mdlname' is a name of 8 characters or less that must be acceptable as a file name to the operating system after the addition of an extension. The default for 'mdlname' is 'model'.

This convention is followed for all other inputs and outputs, including the internal files, with the exception of the SPECS file which is named simply 'FORTMP.SPC'. The following is a list of these files:

model.mps	MPS format input data file
model.bas	MPS format basis data file
model.res	Standard output file
model.log	Standard log file
model.mat	Internal matrix file
model.nam	Internal names file
model.sav	Internal save file
model.bbf	SIMPLEX save (binary basis) file
model.isv	IPM solution save file
model.msv	MIP Branch and Bound tree save file

In addition to providing the model name you can change the file naming structure by using a directory name and by setting individual file names.

The DIRECTORY or equivalent PATH command is as follows.

```
DIRECTORY NAME (dirname)
PATH NAME (dirname)
```

where ‘dirname’ is a text string of maximum length 50 characters which is used as a prefix to every filename with the exception of ‘specs’ (The SPECS file cannot of course define its own prefix). Any special separator such as ‘\’ or ‘/’ to appear between ‘dirname’ and the model name must be included by the user in the ‘dirname’ argument.

Finally some of the input and output files can be given individual names with one of the following SPECS commands:

```
INPUT FILE NAME (filename)
OUTPUT FILE NAME (filename)
LOG FILE NAME (filename)
BASIS FILE NAME (filename)
```

The filename supplied in this case may be up to 70 characters long and will not be prefixed by the directory name (if any).

Back to Chapter contents

3.7 Errors and Recovery

3.7.1 Data Errors

During the data input stage the system checks for errors in the data and will provide you with diagnostic messages, including the input line number, text or other available information to assist in locating an error. When one has been found, the system may continue checking to find more errors (up to 50) but unless the errors found have an obvious remedy the system then halts without attempting a solution.

Back to Chapter contents

3.7.2 Maximum Iterations Reached or Other Limit Exceeded

The following SPECS commands have already been introduced (see Section 3.3) and control the bounds upon the algorithm’s execution.

```
MAXIMUM SIMPLEX ITERATIONS = n          * default 50000
MAXIMUM IPM ITERATIONS = n          * default 80
MAXIMUM MIP <INTEGER/INTSOL> = n    * default 300
MAXIMUM MIP NODES = n               * default 50000
MAXIMUM MIP TIME = v                * default 50000.0
```

On reaching such a limit the FortMP system makes a SAVE using the type of SAVE relevant to the current algorithm and terminates with an error message.

The user will need to check whether an adjustment to parameters is necessary to avoid looping or stalling before rerunning the problem. This is done by using the appropriate 'RESTART' command, one of the following:

```
SIMPLEX START RESTART
IPM RESTART ON
MIP RESTART ON
```

Input time can also be saved with the command:

```
INPUT RESTART ON
```

Back to Chapter contents

3.7.3 Numerical Difficulties and Instability

With the advantage of coefficient scaling provided by the SCALE algorithm (see Section 3.3.6) numerical difficulties and instability should arise only rarely in FortMP and then they are usually overcome without troubling the user.

Some intractable cases of numerical difficulty are detectable and cause premature termination of the solvers. An explanatory message is written to the log and to the standard output (e.g. displayed on the screen). In the case of the Dual algorithm unrecoverable errors are overcome by reverting to Primal in order to continue the optimisation process.

Certain cases may, in very rare situations, cause the system to cycle, without any progress in the iterations, or with only very slow progress towards the optimum.

When the system is halted the user's main technique for overcoming these problems is to change the controls in some way and restart the run. A control is available to allow user to watch the iteration log online so that the run can be halted before excessive time is wasted. In any case there is a maximum iteration limit for both SSX and IPM solvers.

When the maximum iteration limit is reached the system always saves a restart point before exit and the user can also set controls to save restart points periodically to guard against a crash or to allow a run to be halted without wasting the time used so far.

Back to Chapter contents

3.7.4 Running Out of Memory

There are few computer installations where users can completely disregard the amount of memory they use, even with virtual memory available. Many users will attempt the largest possible problems with the memory available whilst others will wish to limit their memory usage for operational reasons.

In order to supprt these requirements FortMP is loaded for execution in a minimum size memory region and will use the system to allocate additional memory according to the problem size.

However in some rare cases the required memory is impossible to predict, while in other cases too little memory can cause serious congestion and inefficiency because of frequent data compression. Hence

the system is provided with several emergency controls to assist with memory management. These are described in Appendix G.

Back to Chapter contents

3.7.5 Software Errors

The user is protected by a great deal of defensive programming so as to detect internal inconsistencies and halt as soon as they occur. In such a case there is an explanatory diagnostic and a 'FATAL ERROR' message written to the log and to the standard output (display).

In this unlikely event you should contact your supplier and provide details so that the fault can be corrected.

Back to Chapter contents

3.8 Summary of SPECS Commands

The following SPECS commands have been introduced to date in this manual.

```
BEGIN
```

This must be the first command line.

```
MODEL NAME (modname)
```

This command supplies a default for the names of input files and output files in the run which are built by adding an extension as follows:

`modname.mps`	MPS format input data file
`modname.bas`	MPS format basis data file
`modname.res`	Standard output file
`modname.log`	Standard log file
`modname.mat`	Internal matrix file
`modname.nam`	Internal names file
`modname.sav`	Internal save file
`modname.bbf`	Simplex save file
`modname.isv`	IPM solution save file
`modname.msv`	MIP Branch and Bound tree save file

The default model name is 'MODEL'.

```
INPUT FILE NAME (filename)
OUTPUT FILE NAME (filename)
LOG FILE NAME (filename)
BASIS FILE NAME (filename)
```

These commands assign a special name to the various input and output files.

```
DIRECTORY NAME (dirname)
PATH NAME (dirname)
```

These two alternative commands prefix all filenames other than those introduced with an explicit 'FILE NAME' command.

```
INPUT TYPE MPS
INPUT TYPE MG
INPUT TYPE FREE
INPUT LONG NAMES
```

These commands select the type of input data.

```
RHS NAME (rhsname)
RANGES NAME (rngname)
BOUNDS NAME (bndname)
OBJECTIVE NAME (objname)
```

These commands select appropriate items in the MPS format input data.

```
MINIMIZE
MAXIMIZE
```

These two commands specify the direction of optimisation. ‘MINIMIZE’ is the default so this command is not actually needed.

```
SCALE ON
SCALE OFF
```

These two commands set the SCALE switch. The default is ON

```
SCALE PASSES = n
SCALE VARIANCE = v
```

These commands set parameters for scaling, to apply when the SCALE switch is ON. The range for ‘Passes’ is 1–4 and the default value is 4. The default target variance is 10.0.

```
PRESOLVE ON
PRESOLVE OFF
```

These two commands set the ‘PRESOLVE’ switch. The default is OFF.

```
PRESOLVE LEVEL = n
```

This command specifies the PRESOLVE level when the PRESOLVE switch has been set ON. The range is 1–5 with default value 5.

```
POSTSOLVE ON
POSTSOLVE OFF
```

These two commands set the ‘POSTSOLVE’ switch (active when PRESOLVE is ON). The default is ON.

```
PRESOLVE LOG LEVEL = n
```

This command specifies the log level for PRESOLVE and POSTSOLVE. The range is 0-4 with default value 1.

```
ALGORITHM PRIMAL
ALGORITHM DUAL
ALGORITHM IPM
```

One of these commands is used to specify the primary solution algorithm for the continuous LP problem. ‘PRIMAL’ is the default and need not be specified.

```
MAXIMUM SIMPLEX ITERATIONS = nnn   *  default 50000
MAXIMUM IPM ITERATIONS = nnn       *  default 80
MAXIMUM MIP <INTEGER/INTSOL> = nnn *  default 300
MAXIMUM MIP NODES = nnn            *  default 50000
MAXIMUM MIP TIME = vvv             *  default 50000.0
```

These commands set the termination limits for each major algorithm. A SAVE is made before exit at termination.

```
SIMPLEX LOG LEVEL = n
INVERT LOG LEVEL = n
```

```
IPM LOG LEVEL = n
MIP LOG LEVEL = n
```

These commands specify the 'level' of the output to be sent to the log file in each major algorithm. The range is 0–4 with default value 1 in each case.

```
SIMPLEX SAVE FREQUENCY = n    * default 10, each 10th reinvert
IPM SAVE FREQUENCY = n        * default 10, each 10th iteration
MIP SAVE FREQUENCY = n        * default 10, each 500th node
```

These commands specify the frequency for making a SAVE in each major algorithm.

```
SIMPLEX START RESTART
IPM RESTART ON
MIP RESTART ON
IPM RESTART OFF
MIP RESTART OFF
```

These commands specify 'RESTART' in each major algorithm. For the SIMPLEX algorithms 'RESTART' is an alternative to other starting basis options. Otherwise there is an ON/OFF switch (default OFF).

```
SIMPLEX START CRASH
SIMPLEX START INPUT BASIS
SIMPLEX START RESTART
SIMPLEX START UNIT BASIS
```

These commands select the mechanism for setting up the starting basis when the main algorithm is PRIMAL or DUAL. CRASH is the default.

```
FEASIBILITY TOLERANCE = value
```

This command assigns a tolerance used in the algorithms. The default for 'value' is 1.0d-5. Other tolerances are introduced later in this manual.

```
INVERT FREQUENCY = n
```

This command assigns a frequency, in terms of the iteration count, for reinversion during SIMPLEX algorithms. The default for '*n*' is 50.

```
SIMPLEX LOG FREQUENCY = n
NODE LOG FREQUENCY = n
```

These commands assign a frequency for printing the iteration log during SIMPLEX algorithms or the node log during MIP. Default for '*n*' is 1.

```
IPM BASREC ON
IPM BASREC OFF
```

When 'ALGORITHM IPM' has been selected the user can set this switch OFF in order to halt execution and print the output immediately on reaching the IPM solution. The default is ON.

```
MIP ON
MIP OFF
```

If there are no discrete variables this switch is always OFF. When there are discrete variables the default is ON and setting to OFF cancels execution of MIP so that user can examine the continuous LP solution.

```
MIP PREPROCESS ON
MIP PREPROCESS OFF
MIP PRIORITY UP ON
MIP PRIORITY UP OFF
```

These commands control important algorithmic features of MIP. Both switches are OFF by default.

```
OUTPUT TYPE STD
OUTPUT TYPE RW
```

These commands specify the type of solution written to the output file. The default is 'STD'

```
OUTPUT ON
OUTPUT OFF
OUTPUT SUPPRESS ZERO
```

These commands control the amount written to the output file. Default is 'ON' with zeros not suppressed.

```
LOG DISPLAY
LOG DISPLAY ONLY
```

This command causes all logged output, not just important diagnostic messages of level 1, to be displayed online (or written to the standard output-stream). With the keyword 'ONLY' the normal log file is suppressed.

```
LOG DISPLAY LEVEL = n
```

This command causes logged output up to level *n* to be displayed online. Default for *n* is 1.

```
END
```

This command terminates the SPECS commands. Anything beyond it is ignored.

Back to Chapter contents

4. Sparse Simplex (SSX) Solver

Contents

4.1 Internal Problem Statement

We begin this chapter by studying the FortMP SSX internal representation of LP problems.

Let m denote the number of constraints and n the number of original variables. All the constraints are converted into equality form by adding appropriate variables to each constraint. This will result in the following LP problem.

Minimise (or Maximise) the objective function

$$\sum c_j x_j$$

subject to the constraints:

$$y_i + \sum a_{ij} x_j = b_i \text{ for } i = 1,\ldots,m$$

and bound constraints (if any) on y_i and x_j.

The x_j variables (that were obtained from the original problem by linear transformations) are referred to as structural variables while the y_i variables added to the rows are called logical variables. Thus, the total number of variables the algorithm works with is $m+n$.

As a result of the linear transformations (done during input), all finite lower bounds of variables will be zero. In the above formulation, all the variables (x_j and y_i alike) can be handled in a uniform way.

Both the x_j and the y_i conform to one of the following types:

Fixed	(variable = 0.0)
Bounded	(0.0 ≤ variable ≤ upper bound)
Plus	(0 ≤ variable)
Free	(variable can take any value)

The simplified structure of the problem is illustrated in Figure 5 below, where the upper bounds U_i and u_j exist only for bounded variables.

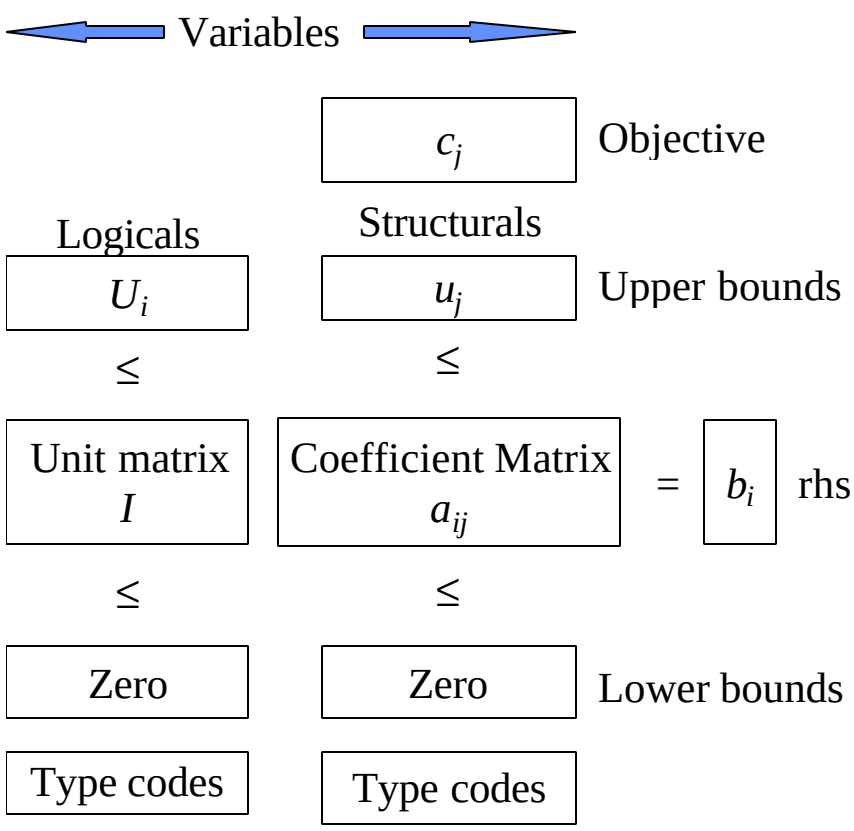


Figure 5 'Internal SSX problem structure'

In practice, the c_j of the objective row appear as coefficients of a row (namely, the objective row) in the MPS form of the input data and this idea is retained internally so that no separate c_j vector is actually needed.

A set of values of the variables is called a solution if it satisfies the constraints. A solution is called a feasible solution if, additionally, it satisfies the bound constraints too. Using this terminology, the purpose of the simplex method is to find a/the best feasible solution if such exists.

In the simplex method, the variables are divided into two separate categories: there are m basic and n non-basic variables. The main distinction is that the value of a non-basic variable may be temporarily fixed (usually at zero, or upper bound) while the value of a basic variable may change at each iteration. The set of values of the actual non-basic variables uniquely determines the values of the basic variables in such a way that they together form a solution. This set of values is called a basic solution. A basic solution can be computed by the inverse of the matrix formed from the columns corresponding to the basic variables. Sometimes this matrix is referred to as a 'basis matrix' or as simply a 'basis'.

An iteration of the simplex method makes either a basis change or a bound change. In the first case one basic variable is replaced by one non-basic variable, while in the second case the basis remains unchanged but a non-basic variable moves from its lower bound to the upper bound or the other way

around. If there is a basis change then the value of the objective function will improve or remain the same, while a bound change always results in an improvement of the objective value.

Back to Chapter contents

4.2 Introduction to the Algorithms

Large scale LP problems tend to have relatively very few non-zero coefficients in the matrix of constraints. There is an observation that the average number of non-zeros per column is between 5 and 10, independent of the size of the problem. This means that as problem sizes increase the density (the ratio of non-zeros to the total number of possible entries) decreases. For problems with around 1000 constraints the density is typically less than 1%. The low density is often referred to as sparsity.

Sparsity is the main feature that enables the solution of very large LP problems. The solution algorithms take full advantage of sparsity of the original problem and also keep the creation of new non-zeros during the solution under control.

The Sparse Simplex algorithm of FortMP is a state-of-the-art implementation of the most advanced sparse LP techniques. This makes FortMP capable of solving large scale LP problems efficiently, reliably and accurately.

Sparsity is exploited throughout all the algorithms of SSX. This requires the application of sophisticated data structures and manipulations which are transparent to the user of the stand-alone version of FortMP.

The main algorithm in SSX of FortMP is PRIMAL. This addresses any LP problem. There is a dual algorithm also available, called DUAL. It requires the presence of a dual feasible solution. This is always available at the nodes of the MIP algorithm and the main intended usage of DUAL is to assist the efficient solution of the subproblems arising in MIP.

The proper operation of both of these algorithms is supported by some additional procedures (such as SCALE, CRASH and INVERT) and these will also be discussed briefly.

Back to Chapter contents

4.3 PRIMAL Algorithm

The simplex method is an iterative procedure making basis/bound changes at each step. The primal simplex first attempts to find a feasible solution (Phase I), i.e. one that satisfies all constraints and bounds. This is achieved by simplex iterations (basis/bound changes). After a feasible solution has been found, it makes further iterations that maintain feasibility and result in a better (or, in case of degeneracy, not worse) objective value (Phase II) until an optimal solution is encountered or the problem is found unbounded.

Optimality is declared if the solution is feasible and no further improving variable can be found. Similarly, infeasibility of the problem is declared if no improving variable can be found while the solution is in Phase I.

The primal algorithm of FortMP is a version of the revised simplex method. The inverse of the basis matrix is kept as a product of a lower and an upper triangular matrix (elimination form of the inverse). This form is generated by a re-factorization (reinversion) step and can optionally be maintained during the simplex iterations.

Back to Chapter contents

4.3.1 Starting Procedures

A solution procedure can start from the all logical (UNIT) basis, crash basis, or supplied basis.

The UNIT basis is created in such a way that a logical variable and a corresponding unit vector are added to every constraint in the LP problem. They, together, form a unit matrix which is chosen as a first basis with all the structural variables set to zero. The basic solution for this basis is equal to the right-hand-side vector. This is the simplest starting basis and the fastest to obtain. Another advantage is that the subsequent first simplex iterations are very fast. The disadvantage is that it does not contain any of the structural variables that must be introduced into the basis one by one during the simplex iterations. Another disadvantage is that the starting solution may be infeasible, requiring many iterations to find a feasible solution (if one exists).

The crash basis is created with some work but this is carried out relatively very quickly since the major part of it is done symbolically. This crash procedure attempts to replace as many logical vectors as possible by structural ones. The vectors are selected so that they can symbolically be rearranged to form a triangular basis (i.e. they can enter the basis without transforming them). Additionally, if there is a tie in the selection of the incoming structural variables (column tie), variables (vectors) with larger feasibility range are given priority. In the case of row tie, logical variables with smaller feasibility range are selected to be replaced. The resulting triangular basis is easy and efficient to handle and has a greater chance to being feasible or near feasible. Experience shows that using this basis requires fewer simplex iterations to solve a wide range of problems. Consequently it has been chosen as a default.

By supplied basis we mean one of two possibilities. One is the case when the basis is given in MPS format as a text (ASCII) file prepared by the user or by an earlier run of FortMP. The basis is interpreted and amended by logical variables as necessary. The other case is a binary basis created by FortMP earlier. The quality of such starting bases is completely dependent on factors known by the user. Algorithmically they can be extremely good (e.g. to re-optimize a slightly modified problem starting from the previous optimum) or just poor. In any case, when users are prepared to supply a basis they may have some knowledge about the problem and the basis and, therefore, can judge the usefulness of this starting procedure.

Back to Chapter contents

4.3.2 Column Selection

Column selection refers to the choice of an improving variable to bring into the basis or to change its bound setting. The selection is made from among non-basic vectors that show a rate of improvement (candidate vectors). The rate of improvement is reflected by the sign and magnitude of the reduced cost (d_j) of the variable corresponding to a vector. The positivity (or in some cases the negativity) of the reduced cost is interpreted with a tolerance (DJ TOLERANCE, default value is 1.0d-5). It means that a

computed d_j value is considered negative if it is less than -(DJ TOLERANCE), i.e. the negative of the tolerance.

The ultimate goal would be to select the best candidate. By best we mean the one by which optimality can be achieved in the least number of iterations. Unfortunately, it is theoretically impossible to identify such a candidate. At column selection only the rate of improvement can be determined but not the amount of improvement in the objective function. A good rate does not necessarily give large improvement. The only thing that can be done is to increase the probability of selecting good candidates.

However, even this modified goal is hard to achieve. Column selection is based on a step called pricing the non-basic variables. It involves a computation of a dot product of each non-basic column and a pricing vector (simplex multiplier). If the size of the problem is large there can be many non-basic columns having many non-zero elements and the pricing operation can be very expensive in terms of computation (and time). Thus, economizing column selection is not an easy exercise.

PRIMAL applies a sophisticated column selection procedure that attempts to compromise in finding good candidates and reduce the work per selection. The procedure can be characterized by the following. Usually, only a part of the non-basic variables is scanned (partial pricing). In a pricing step, scanning starts from where it was previously terminated. Having reached the end of the matrix, pricing continues at the beginning (cyclic pricing). The number of variables to be scanned is determined after each reinversion (dynamic partial pricing). During one scanning pass, several candidates are selected (multiple pricing). The number of them is controlled by the SPECS command

```
PRIMAL MSUB = n
```

where *n* is given by the user if needed. The default value for MSUB is 4. Since 4 is satisfactory for a large number of cases this command can simply be omitted from the SPECS file. The selected candidate columns are updated (expressed in terms of the current basis) and are stored in explicit (dense) form. This operation is called FTRAN.

PRIMAL temporarily considers a smaller LP problem consisting of the selected variables and the current basic variables and finds the optimal solution of it (sub-optimization). The variable to enter the basis is selected by the criterion of greatest improvement. This is expensive for the whole problem but moderate for the sub-optimization. At the same time, it results in a very good iteration from amongst the selected variables.

Selection of candidates for sub-optimization is called major iteration, while iterations in the sub-optimization are called minor iterations. Minor iterations are the actual simplex iterations.

The user may want to experiment with the value of MSUB. He should be warned that increasing its value results in increased memory requirement and steps in sub-optimization tend to slow down.

PRIMAL is also equipped with a more sophisticated column selection strategy which is widely known as DEVEX. This technique attempts to estimate the progress that can be achieved by a potential candidate variable. It is based on an approximation of the column norms which requires considerable extra computations that has to be traded off against the possible reduction in number of iterations needed. At every iteration the quality of the approximation is checked and when it becomes too inaccurate, column norms are reinstated. Such an event is triggered when the ratio of the accurately computed to the estimated column norm of the entering column falls below a tolerance (DEVEX

RATIO) which has a default value is 0.4. This value can be anything between 0 and 1 but values outside the range (0.2 - 0.6) are not recommended. Higher value will result in a more frequent recalculation of the norms which is costly.

Certain additional column selection procedures are provided for use in parallel processing and in research studies:

- Sectionalised partial pricing
- Dynamic scaling
- Adaptive composite phase 1 weighting
- Anti-degenerate wighting

These are described in Appendices F and G.

Back to Chapter contents

4.3.3 Phase I

PRIMAL is equipped with a particularly efficient algorithm to find a first feasible solution or to detect that one does not exist (problem infeasible). This algorithm takes full advantage of the features of the basis created by the crash procedure outlined above. Namely, it can take more efficient steps if the basis contains variables with relatively large feasibility ranges. The important characteristic feature of this procedure is that it allows feasible basic variables to become infeasible but the sum of infeasibilities is kept monotonic decreasing in modulus. This increased freedom of the basic variables usually results in much more efficient basis changes reducing the sum of infeasibilities to a maximum possible extent per iteration.

Feasibility is interpreted with a tolerance. It means that if a variable violates one of its bounds by more than FEASIBILITY TOLERANCE (user-accessible parameter, with default value of 1.0d-5) then it is treated as infeasible.

Before each iteration in Phase I, before each major iteration in Phase II, and after each reinversion, the feasibility of the actual solution is checked. If it is found infeasible, the Phase I procedure is activated, otherwise a Phase II iteration is executed. The main difference between these two types of steps lies in setting up the simplex multiplier and determining the variable to leave the basis (outgoing variable).

With numerically difficult problems, it may happen occasionally that after a Phase II reinversion the newly computed solution becomes infeasible. Checking feasibility detects such a case and handles it automatically.

Back to Chapter contents

4.3.4 Handling Degeneracy

Degeneracy in linear programming has long been known as a phenomenon that can cause troubles for the simplex algorithm. A solution is degenerate if at least one basic variable is at its lower or upper

bound. An iteration is degenerate if the value of the objective function does not change. Degeneracy of an intermediate solution may (but does not necessarily) result in a degenerate iteration. We talk about stalling if degenerate (non-improving) steps are made for a large number of iterations. In the worst case, stalling can become cycling if the procedure returns to an earlier basis and this sequence of iterations is repeated an infinite number of times.

Since large scale real-life problems – even after PRESOLVE – tend to be rather degenerate, proper handling of degeneracy is an important issue.

PRIMAL is well prepared to solve degenerate LP problems efficiently. Its Phase I procedure provides some automatism to reduce the chance of degenerate steps by maximizing the progress towards feasibility. In the case of MSUB > 1 when the principle of greatest change is applied this chance is further increased. This latter remark is valid for Phase II, as well, i.e. whenever a non-degenerate step is possible with the candidate vectors, PRIMAL will find it.

In Phase II, PRIMAL applies a special technique to avoid the repetition of a previous basis. It makes arrangements to improve the objective at each iteration, at least by a very small amount even if it may result in the slight violation of feasibility. Such small violations, if they arise, are kept under control and are completely eliminated before termination. This anti-degeneracy technique is very cheap (hardly requires any additional operations per iteration), contributes to an increased numerical stability of the solution and does not require the identification of the presence of degeneracy.

The anti-degeneracy strategy is automatic and is transparent to the user.

Back to Chapter contents

4.3.5 Numerical Features

The revised simplex method works with real (floating point) numbers. During operations with such numbers, different types of numerical errors may occur. The two main types are cancellation error (in subtraction or addition) and rounding error (losing least significant digits in any of the operations). At one extreme, these errors may remain unnoticed but, on the other, they may cause serious troubles.

Good numerical procedures try to minimize the occurrence of numerical errors and, if they still occur, can efficiently cope with the situation. In the case of the simplex method, numerical problems can have quite serious effects on the solution. They can slow down progress towards feasibility/optimality, or, in the worst case, can prevent convergence.

FortMP, and within it PRIMAL, is equipped with a wide range of techniques aiming at guaranteeing smooth progress, without numerical problems.

The first is scaling. The adaptive scaling algorithm of FortMP reduces the range of magnitude of the coefficients.

The second is the careful use of tolerances. This means that whenever a numerical value is to be tested against 0 (zero) a small tolerance is used instead of 0. The appropriate value of a tolerance is situation dependent. This is the reason PRIMAL uses several different tolerances. (Note that some tolerances may have identical default values.)

In addition to the above, PRIMAL regularly checks accuracy at some well-defined points and takes appropriate actions (usually carries out a reinversion to restore accuracy) to avoid serious consequences. The algorithm works with the assumption that the INVERT procedure of PRIMAL produces the most accurate numerical values.

Back to Chapter contents

4.3.6 Issues of Efficiency

The total computational effort needed to solve an LP problem defines the efficiency. Since computational effort cannot easily be measured, the CPU time can be used as a measure of efficiency.

Since PRIMAL works entirely in the CPU (it does not use disk communication during optimization), its performance can be traced. There are several factors that affect performance.

The number of candidates in sub-optimization (MSUB) is one of them. If MSUB is 1 then only one vector is selected at each pricing operation (single pricing). Since pricing is expensive, using single pricing will lengthen the overall solution time. If we go to the other extreme, namely increase MSUB beyond 6, the memory requirement goes up considerably (each selected column must be stored explicitly in double precision) and later minor iterations in the sub-optimization tend to be locally good and globally less valuable. This usually results in an increase of the total number of iterations that may not be offset by the increased iteration speed. At the same time, it also may happen that only a smaller part of the selected and updated vectors can finally enter the basis rendering a large part of the computational work useless. MSUB = 4 seems to be a good compromise.

Back to Chapter contents

4.4 DUAL Algorithm

The DUAL algorithm aims to solve the DUAL problem that is associated with the (primal) problem to be solved. However it does not simply construct this DUAL problem and use the primal algorithm to solve it, instead it retains the primal data structure internally and adopts the DUAL argument for every step of its procedure. DUAL lacks any equivalent of the major/minor iteration structure and sub-optimization in PRIMAL, but is otherwise entirely equivalent and is, almost, equally capable as a solver for LP problems.

DUAL is of particular use for solving MIP sub-problems since each sub-problem has an initial basic solution that is DUAL-feasible but PRIMAL-infeasible. Hence DUAL can begin immediately in Phase 2 and often requires only a few iterations to reach the optimum. As a further advantage the objective value progresses monotonically from a better to a worse value, so that MIP cutoff considerations apply, and we may never need to complete the solution process.

DUAL suffers the disadvantage that its calculations may be less accurate than with PRIMAL and it may abort through stalling or through unrecoverable numerical error. In this case the FortMP system always reverts to PRIMAL to overcome the difficulty and solve the problem with its greater reliablility.

Back to Chapter contents

4.4.1 Starting Procedures

For solving an original LP problem from scratch DUAL has the same starting procedures as those of PRIMAL (see section 4.3.1).

Back to Chapter contents

4.4.2 Row Selection

This is analogous to Column Selection in the primal algorithm. The selection is based on primal infeasibility and can be modified by applying DUAL DEVEX, which is entirely analogous to DEVEX in the primal algorithm (see 4.3.2). DUAL DEVEX is a much better choice for DUAL than PRIMAL DEVEX is for PRIMAL. This is because the calculation overhead is much less, and so the option is recommended for all larger or more difficult MIP problems.

DUAL DEVEX employs the same DEVEX RATIO as for the primal case (see 4.3.2).

Back to Chapter contents

4.4.3 DUAL Phase 1

Since DUAL is usually employed when the starting basic solution is dual-feasible, any DUAL phase 1 may not be necessary. However as with PRIMAL the phase sometimes switches back after a re-invert, particularly in harder problems. The ability to recover with phase 1 is therefore an important feature, and much time may be saved, especially in MIP problems.

Back to Chapter contents

4.4.4 Handling Degeneracy

FortMP DUAL is provided with an anti-degenracy option based on nested reduction of the degenerate positions so as to make a unique pivot-choice. There is also a progress-check on the objective to counter persistant stalling. These techniques are controlled by SPECS commands described below.

Back to Chapter contents

4.4.5 Numerical Features

Several SPECS commands are available to adjust the controls on pivot tolerance, updating accuracy, and accuracy check by comparing results evaluated separately by the forward and backward tansformations. The standard default setting for these controls have been chosen to balance the demands of efficiency and reliablilityfor the great majority of problems. In cases of special difficulty it may be possible to obtain better performance by changing them.

Back to Chapter contents

4.5 INVERT

At each basis change the representation of the inverse of the basis ('ETA file', which is not really a file but a set of arrays in the main memory) gets appended by some newly created vector(s). After several iterations this representation can grow quite large which may become a problem in itself because of the limited memory available. However, there are two other problems arising here. The operations with the inverse become more and more time consuming because of the growing number of non-zero elements in the inverse. Furthermore, the newly created components are computed from earlier ones and therefore inherit their inaccuracies in addition to their own. As a result, as we make more iterations after a reinversion, operations become slower and less accurate. When this reaches a critical level, a reinversion must be carried out.

The main purpose of INVERT is to 'refresh' the representation of the inverse of the basis. This results in a shorter and more accurate ETA file. Thus, right after a reinversion, iterations are faster and more accurate. However, reinversion itself is just a reproduction step and is not for free. Its time complexity is problem dependent but never negligible. It means that reinversion should not be performed too frequently.

In PRIMAL, the need for reinversion depends also on the way the new vector(s) of a basis change are created. There are two possibilities. The first is the Product Form (PF) update. This usually generates a relatively large number of new non-zero elements in the ETA file but the computations themselves do not require additional memory. If PF update is used then a good value for reinversion frequency is 50, i.e. make a reinversion after every 50 basis changes. The other possibility is to use the Forrest–Tomlin (FORTOM) update scheme which usually results in a very modest growth of non-zeros in the ETA file. However, it requires the explicit storage of MSUB vectors (increased memory requirement) and involves some additional computation. Usually the system begins with FP-update and switches at some stage to FORTOM when ETA-growth becomes large. At the same time the re-inversion frequency is increased by half - from 50 to 75 - so as to take advantage of the reduced growth. This can be controlled with SPECS commands as described below.

Back to Chapter contents

4.6 SSX Algorithm Controls

4.6.1 The Principal Controls

A description of the syntax of SPECS commands which control the run has been given in Chapter 3, Section 3.1.6.

The following commands state the direction of optimisation, i.e. how the objective row is to be treated:

```
MINIMIZE
MAXIMIZE
```

The default is 'MINIMIZE'.

The following commands invoke either PRIMAL or DUAL as the initial solver for LP problems:

```
ALGORITHM PRIMAL
```

```
ALGORITHM DUAL
```

The default is PRIMAL.

The following command specifies the maximum number of candidate columns used by PRIMAL in sub-optimisation and extracted by multiple pricing:

```
PRIMAL MSUB = n
```

The default is MSUB = 4 with possible values 1 to 10. Single pricing (MSUB = 1) will lengthen the overall solution time because only one minor iteration takes place to every major iteration. If we go to the other extreme, namely increase MSUB beyond 6, the memory requirement goes up considerably (with FORTOM ON each selected column is stored twice in double precision) and later minor iterations in the sub-optimization tend to have very little value. The total number of iterations increases but without any corresponding increase in iteration speed. At the same time, it also may happen that only a smaller part of the selected and updated vectors can finally enter the basis rendering a large part of the computational work useless. MSUB = 4 seems to be a good compromise.

The following commands control the use of DEVEX column selection procedure in the PRIMAL algorithm:

```
PRIMAL DEVEX OFF
PRIMAL DEVEX ON
PRIMAL DEVEX SINGLE
PRIMAL DEVEX DOUBLE
```

the default being 'OFF'. 'ON' and 'SINGLE' are the same and indicate that DEVEX should apply only in Phase 2 (i.e. when feasible). 'DOUBLE' indicates that DEVEX should apply in both Phase 1 and Phase 2.

The corresponding commands for DUAL are:

```
DUAL DEVEX OFF
DUAL DEVEX ON
```

the default being 'OFF'. 'ON' indicates that DEVEX should apply only in Phase 2 (i.e. when DUAL feasible).

The following commands set or cancel Forrest–Tomlin update of the ETA file:

```
FORTOM OFF
FORTOM ON
FORTOM AUTO
```

When Forrest–Tomlin is cancelled with 'OFF' the simple product form update is used in its place. The default is 'FORTOM AUTO' under which Forrest-Tomlin is initially OFF but will be set to ON - i.e. become active - when certain criteria are satisfied concerning the proportion of structurals in the basis and growth of the ETA-file during SSX iterations. At the same time when Forrest-Tomlin is AUTO-activated the invert frequency is increased by one half (actually from 50 to 75 in default). Thus 'ON' is really unnecessary except for experimental reasons. It should not be necessary to use 'OFF' except for very dense problems in which the extra calculations outweigh the savings. Users may suspect the

Forrest–Tomlin update as being of no real benefit if growth figures of 200% or more show up on the INVERT log (with the standard INVERT frequency increased to 76 iterations).

The following command sets a maximum to the number of SSX iterations between re-INVERTs.

```
INVERT FREQUENCY = n
```

The default is 50 which will increase to 76 at Forrest-Tomlin auto-activation. Reinversion will take place at earlier intervals if a demand arises for some other reason, e.g. lack of memory or signs that the inaccuracy is growing too large. The user should use a lower value if the switch FORTOM is set OFF.

For the DUAL algorithm the anti-degeneracy procedure can be employed with the following SPECS command:

```
DUAL ADEGEN = n                    * Default is n=0 implying OFF
```

Values 1 and 2 can be given both specifying ON. Option 2 is faster. When this option is OFF there is a progress-check applied during DUAL iterations which stops DUAL if it becomes very slow. The system reverts to PRIMAL in this case.

Back to Chapter contents

4.6.2 Starting Basis Controls

The control switches for the starting basis are set by the following commands:

```
SIMPLEX START CRASH
SIMPLEX START INPUT BASIS
SIMPLEX START RESTART
SIMPLEX START UNIT BASIS
```

'CRASH' is the default if no command is given. 'RESTART' is described in Section 4.6.4 and 'INPUT BASIS' is described in Section 4.6.5. 'UNIT BASIS' may be used to set up an all logical starting basis rather than 'CRASH'.

When IPM is used the starting basis for the final check by PRIMAL is provided by BASREC as will be described in the next chapter.

Back to Chapter contents

4.6.3 Tolerances and parameters

A tolerance is a small numerical range. Two numbers whose difference lies within this range are considered to be equal. Tolerances are used throughout FortMP to overcome the effects of computer arithmetic, which, for calculations involving real and double precision quantities, is imprecise.

Various tolerances can be set by SPECS commands; however, the user should be aware that the default values have been the subject of careful research and should not be lightly changed. The defaults are expected to be effective provided that the problem coefficients have been normalised with the SCALE procedure.

The following tolerances affect feasibility control:

```
FEASIBILITY TOLERANCE = value      * Default is 1.0d-5
RHS TOLERANCE = value              * Default is 1.0d-5
DJ TOLERANCE = value               * Default is 1.0d-5
```

where the default value is given in each case. 'FEASIBILITY' and 'RHS' tolerances are in fact synonyms for the same tolerance value which defines how far any primal variable solution value may go outside its range without being deemed infeasible. DJ tolerance is the equivalent for any dual solution value. This means in effect how far a d_j may become negative if the primal variable is at lower bound or how far positive if the primal is at upper bound.

The following controls affect pivot selection, which is vital in maintaining stability and restraining the growth of non-zeros on the ETA file[1].

```
PRIMAL PIVOT ADMIT THRESHOLD = v   * Default v = 1.0d-8
DUAL PIVOT ADMIT THRESHOLD = v         * Default v = 1.0d-
8
INVERT PIVOT ZERO TOLERANCE = v        * Default v = 1.0d-
7
INVERT PIVOT ADMIT RELATIVE = v        * Default v = 1.0d-
2
```

The default values are as given above. Pivot control always avoids selecting any pivot less than the 'ADMIT' threshold, and 'ZERO' tolerance refers to the level considered indistinguishable from zero.

INVERT pivoting is protected both by a large zero tolerance - the criterion for singularity - and also by the 'ADMIT' threshold, given as 'RELATIVE' which in fact is more important. This is a factor that, when multiplied into the largest absolute value on the current row or column being considered for pivoting, defines the minimum admissible pivot size. Relative thresholds between 0.001 and 0.1 may be suitable. Larger values lead to a higher growth of non-zeros on the ETA file but lower values may lead to instability or inaccuracy.

A wider choice of pivoting controls is described below in section 4.6.7.

The following tolerance is used generally to test for zero or to test whether two values are equal:

```
ZERO TOLERANCE = value             * Default is 1.0d-15
```

The following command sets the parameter used by the DEVEX algorithm (both in PRIMAL and in DUAL):

```
DEVEX RATIO = value                * Default is 0.4
```

[1] The four commands given here have been changed from their forms in previous versions of FortMP. This is to clarify their meaning and name the algorithm to which each applies. Old forms are included in Appendix B as alternates and remain usable, however users should revise them if possible.

which determines when to trigger re-calculation of column-norms (see 4.3.2 above) for PRIMAL, or row-norms for DUAL. Values outside the range (0.2 - 0.6) are not recommended.

Forrest-Tomlin auto-activation is controlled by the following SPECS commands:

```
FORTOM ACTIVATE PERCENT = n        * Default is 40

FORTOM ACTIVATE GROWTH = n              * Default is 100
```

The 'PERCENT' in the first of these commands refers to percentage of structural variables in the basis which must reach this level for activation. In addition 'GROWTH', the amount of increase during SSX iterations after an INVERT, must be this percentage of the ETA-size immediately after the invert. The default value 100 implies that ETA-size must double before auto-activation can take place.

The DUAL algorithm is subject to a progress-check for which controls may be entered with the following SPECS commands:

```
DPROGRESS FREQUENCY = n            * Default is 1

DPROGRESS CRITERION = value        * Default = 1.0d-25
```

Checks are made after re-inversion and the 'FREQUENCY' refers to the number of inversions made between checks. If the progress made in the dual objective does not exceed the criterion given in this time then DUAL is considered as stalled and the system reverts to PRIMAL. The check is necessarily slow to operate, even with comparisons at every re-invert. Hence if dual degeneracy occurs frequently it is better to invoke the anti-degeneracy procedure (DUAL ADEGEN = 1 or 2) which should improve progress anyway. However anti-degeneracy is itself rather slower than normal iterating.

Back to Chapter contents

4.6.4 Save and Restart

The following command provides necessary SAVEs for re-starting a subsequent run.

```
SIMPLEX SAVE FREQUENCY = n         * Default is 10
```

SAVEs take place during reinversion. The given frequency defines how many reinversions take place between successive SAVEs.

The restart procedure using such a save operates by providing the saved basis as a starting basis for the next run. The necessary command is

```
SIMPLEX START RESTART
```

which has already been described in Section 4.6.2

The following command specifies the limiting number of SSX iterations.

```
MAXIMUM SIMPLEX ITERATIONS = n          * Default is 10000
```

When this limit is reached both PRIMAL and DUAL will terminate after doing a SAVE operation to provide for a restart. The given limit is added to any previous iterations in a restart so that you do not need to change the limit. Iteration numbering continues from the previous run.

Back to Chapter contents

4.6.5 Processing an External Basis

An external basis in MPS form can be used to provide the starting basis. The command is

```
SIMPLEX START INPUT BASIS
```

which has already been described in Section 4.6.2

The following commands set or clear a switch to provide an output basis for starting a subsequent run.

```
OUTPUT BASIS ON
OUTPUT BASIS OFF
```

The 'OUTPUT BASIS ON' command invokes output of an MPS format basis when the SSX solver (whether PRIMAL or DUAL) has completed normally. This basis can be used as a starting basis for similar problems with the same names for rows and columns – the identity being on the basis of names rather than indices.

For the layout of MPS format basis records see Section 3.5.1 and Appendix A1.3. The corresponding 'BASIS INPUT' procedure does not require every name to be matched but it ignores any record which cannot be matched and in the case of upper bound status it ignores any record where the new variable no longer has an upper bound.

Back to Chapter contents

4.6.6 Log Lising Controls

Logged output from SSX algorithms is controlled with the following commands.

```
SIMPLEX LOG LEVEL = n              * n=0-4, Default is 1
INVERT LOG LEVEL = n               * n=0-4, Default is 1
SIMPLEX LOG FREQUENCY = n          * Default is 1
```

where the defaults are as shown. The 'LOG FREQUENCY' defines at what iteration frequency the primal log is issued (levels 2 and above). This log is also issued by DUAL and shows the Phase I and Phase II objective values plus other items of interest.

The INVERT log gives details of the ETA counts and non-zero counts both before and after each reinversion.

Back to Chapter contents

4.6.7 Special Pivoting and Update Controls

Some of the controls on pivoting have already been described in section 4.6.3 above. It has been found convenient to extend these controls to cover all three SSX algorithms, and to make additional controls available for research testing and for 'tuning' the most difficult problems. In what follows a SPECS command is prescribed for each operation, however not all of these are, in fact, available for use. Those not available are fixed at a default level, which is applied as a constant, or if zero, is not applied at all (in the current release). They are indicated by the annotation 'Fixed' for the value in the command.

The following tolerances relate to pivoting:

```
PRIMAL PIVOT ZERO TOLERANCE = v        * Fixed v = 1.0d-35
DUAL PIVOT ZERO TOLERANCE = v      * Default v = 1.0d-14
INVERT PIVOT ZERO TOLERANCE = v        * Default v = 1.0d-
7
```

These commands all refer to the level considered as zero by the corresponding algorithm.

The following commands specify an acceptance level for pivoting:

```
PRIMAL PIVOT ADMIT THRESHOLD = v   * Default v = 1.0d-8
DUAL PIVOT ADMIT THRESHOLD = v         * Default v = 1.0d-
8
INVERT PIVOT ADMIT THRESHOLD = v   * Fixed v = 0
PRIMAL PIVOT ADMIT RELATIVE = v        * Fixed v = 0
DUAL PIVOT ADMIT RELATIVE = v      * Default v = 0.0
INVERT PIVOT ADMIT RELATIVE = v        * Default v = 1.0d-
2
```

(Keyword 'THRESHOLD' may be added after 'RELATIVE'). Pivots falling between the 'ADMIT' level and the 'ZERO' level are postponed, hopefully so as to avoid their use completely. In the case of INVERT these pivots cannot be selected until no other alternative remains - the ZERO level then defines the criterion for singularity.

In the case of PRIMAL and DUAL postponed pivots are *flagged* - that is the corresponding row or column selection is temporarily ruled out until a fresh re-invert occurs. When all possible selections are flagged out in this way an early re-inversion can be invoked so as to correct the numerical error as far as possible before trying again. However, if all selections become flagged immediately after re-invert, then the algorithm is blocked and is halted. DUAL then reverts to PRIMAL, but such an error in PRIMAL is not resolvable.

The following tolerance applies to the forward/backward pivot comparisons made as a check on numerical accuracy:

```
PIVOT DIFFERENCE EPSILON = v           * Default v = 1.0d-
2
```

The tolerance is relative - that is to say the difference should not be greater in magnitude than this multiple of the original calculations (using forward transform or backward transform). In case of violation the system may re-invert directly or flag the selection as with pivots below the admit level.

The following zero tolerances apply to update calculations:

```
PRIMAL RELATIVE EPSILON = v             * Fixed v = 0.0
DUAL RELATIVE EPSILON = v          * Default v = 0.0
INVERT RELATIVE EPSILON = v             * Fixed v = 0.0
```

Which specify that the result of a subtraction equates to zero if its magnitude is less than a certain fraction of the original magnitudes subtracted.

Back to Chapter contents

4.7 Summary of SPECS Commands

The following SPECS commands relate to the SSX algorithm. Those introduced in this chapter are highlighted thus (e.g.) '`FORTOM ON`'. Some commands, which are not actually available yet, are written in parentheses and have 'Fixed' values.

```
MINIMIZE
MAXIMIZE
```

These two commands specify the direction of optimisation. 'MINIMIZE' is the default so this command is not actually needed.

```
ALGORITHM PRIMAL
ALGORITHM DUAL
```

One of these commands is used to specify which SSX algorithm is to be used. 'PRIMAL' is the default and need not be specified.

```
MAXIMUM SIMPLEX ITERATIONS = nnn   *  default 10000
```

This command sets the termination limit for SSX algorithms. A SAVE is made before exit at termination.

```
SIMPLEX LOG LEVEL = n
INVERT LOG LEVEL = n
```

These commands specify the 'level' of the output to be sent to the log file in each major algorithm. The range is 1–4, with default value 1 in each case.

```
SIMPLEX SAVE FREQUENCY = n         * default n=10,
                                     each 10th reinvert
```

This command specifies the frequency for making a SAVE in SSX algorithms.

```
SIMPLEX START RESTART
```

This command specifies 'RESTART' for the SIMPLEX algorithms. 'RESTART' is an alternative to other starting basis options.

```
SIMPLEX START CRASH
SIMPLEX START INPUT BASIS
SIMPLEX START RESTART
SIMPLEX START UNIT BASIS
```

These commands select the mechanism for setting up the starting basis when the main algorithm is PRIMAL or DUAL. CRASH is the default.

```
PRIMAL MSUB = n
```

This command sets the number of priced candidates in PRIMAL sub-optimisation. The range for '*n*' is 1–10, with default value 4.

```
PRIMAL DEVEX OFF
PRIMAL DEVEX ON
PRIMAL DEVEX SINGLE
PRIMAL DEVEX DOUBLE
DUAL DEVEX OFF
DUAL DEVEX ON
```

The above commands control use of DEVEX algorithm, the default being 'OFF' for both PRIMAL and DUAL. 'DOUBLE' causes DEVEX to be active in both phases of the PRIMAL algorithm. 'ON' and 'SINGLE' are equivalent.

```
DEVEX RATIO = value                    * Default = 0.4
```

This command sets the ratio-criterion for recalculating column-norms or row-norms in DEVEX.

```
INVERT FREQUENCY = n
```

This command assigns a frequency, in terms of the iteration count, for reinversion during SIMPLEX algorithms. The default for '*n*' is 50.

```
FORTOM ON
FORTOM OFF
FORTOM AUTO
```

The above commands set or cancel Forrest-Tomlin update of the ETA file. Default is AUTO implying initially OFF and subsequently becomes ON when the auto-activate creteria are satisfied.

```
FORTOM ACTIVATE PERCENT = n              * Default = 40
FORTOM ACTIVATE GROWTH = n          * Default = 100
```

These commands control the auto-activation of Forrest-Tomlin update. 'PERCENT' is the required percentage of structurals in the basis and 'GROWTH' is the (percent) increase to ETAs during SSX iterations between inverts. Both criteria must be satisfied before auto-activation can occur.

```
DPROGRESS CRITERION = value              * Default = 1.0d-25
DPROGRESS FREQUENCY = n             * Default = 1
```

These commands control the progress-check made in the DUAL algorithm when there is no anti-degeneracy procedure specified (default DUAL ADEGEN = 0). 'FREQUENCY' refers to the separation betweencomparisons, counted in number of re-inverts, and 'CRITERION' is the minimum progress to be made - otherwise DUAL is stopped and the system reverts to PRIMAL.

```
DUAL ADEGEN = n                        * Default is 0
```

Controls use of anti-degeneracy in the DUAL algorithm. Zero specifies progress-checking (the default). Values 1 and 2 specify anti-degenerate pivoting.

```
FEASIBILITY TOLERANCE = value      * Default is 1.0d05
RHS TOLERANCE = value              * Default is 1.0d-5
DJ TOLERANCE = value               * Default is 1.0d-5
ZERO TOLERANCE = value             * Default is 1.0d-15
(PRIMAL PIVOT ZERO TOLERANCE = v   * Fixed v = 1.0d-35)
DUAL PIVOT ZERO TOLERANCE = v      * Default v = 1.0d-14
INVERT PIVOT ZERO TOLERANCE = v        * Default v = 1.0d-
7
```

The above commands assign tolerances used in the algorithms.

```
PRIMAL PIVOT ADMIT THRESHOLD = v   * Default v = 1.0d-8
DUAL PIVOT ADMIT THRESHOLD = v         * Default v = 1.0d-
8
(INVERT PIVOT ADMIT THRESHOLD = v  * Fixed v = 0)
(PRIMAL PIVOT ADMIT RELATIVE = v   * Fixed v = 0)
DUAL PIVOT ADMIT RELATIVE = v      * Default v = 0.0
INVERT PIVOT ADMIT RELATIVE = v        * Default v = 1.0d-
2
```

The above commands specify an acceptance level for pivoting:

```
PIVOT DIFFERENCE EPSILON = v       * Default v = 1.0d-2
```

The above command controls the checking by forward/backward comparison of pivot values.

```
(PRIMAL RELATIVE EPSILON = v       * Fixed v = 0.0)
DUAL RELATIVE EPSILON = v          * Default v = 0.0
(INVERT RELATIVE EPSILON = v       * Fixed v = 0.0)
```

The above commands specify relative zero levels in subtractions performed during update or calculation of vectors.

```
SIMPLEX LOG FREQUENCY = n
```

This command assigns a frequency for printing the iteration log during SIMPLEX algorithms. The default for '*n*' is 1.

Back to Chapter contents

5. The Interior Point Method

Contents

5.1 Introduction to the IPM Algorithm

5.1.1 IPM Problem Statement

FortMP IPM is a primal dual approach to the solution of an LP problem; that is to say it takes note of the original problem (primal) and of the corresponding dual problem in which the coefficient matrix is transposed (rows become columns and columns rows), the right-hand sides become objective coefficients and the objective coefficients become right-hand sides.

The primal problem may be stated at follows.

Minimise the expression $\sum c_j x_j$ *subject to the conditions:*

$$\sum a_{ij} x_j = b_i \text{ for } i = 1,2,\ldots,m$$

$$x_j + s_j = u_j \text{ for } j = 1,2,\ldots,n$$

where the x_j are the structural variable and the variables s_j $(j = 1,2,\ldots,n)$ are complements of the x_j with respect to the upper bounds u_j.

We state the equations this way although naturally variables s_j only exist for those x_j which are bounded in both senses. For the sake of simplicity this complication is ignored in all the equations that follow.

These equations are derived from the original problem statement (see Chapter 1, Section 1.7) by dropping all fixed variables and free rows and by adding slack variables to convert inequalities into equalities.

The dual problem may be stated as follows.

Maximise the expression $\sum b_i y_i - \sum u_j w_j$ *subject to the conditions:*

$$\sum a_{ij} y_i + z_j - w_j = c_j \text{ for } j = 1,2,\ldots,n$$

where the variables y_i $(i = 1,\cdots,m)$ *are dual variables complementary to the original rows of the primal statement above,* z_j $(j = 1,\ldots,n)$ *are dual variables complementary to the* x_j *and* w_j $(j = 1,\ldots,n)$ *are dual variables complementary to the* s_j.

All of the variables x_j, s_j, z_j and w_j $(j = 1,2,\ldots,n)$ must be non-negative; this criterion defines an interior point for the system.

According to LP theory any feasible solution to the above equations will also be optimal if the duality gap (i.e. the difference between primal and dual objective functions) is zero. This duality gap is given by the sum:

$$\sum_{j=1}^{n} x_j z_j + \sum_{j=1}^{n} s_j w_j$$

in which every term must be separately zero because all the variables are positive or zero.

In the logarithmic barrier method these conditions are restated as:

$$x_j z_j = m \text{ for } j = 1,2,\cdots,n$$

$$s_j w_j = m \text{ for } j = 1,2,\cdots,n$$

where *m.* is a positive constant that must reduce to zero at the final stage of the iterative solution process. *m* is referred to as the 'Barrier parameter'.

Back to Chapter contents

5.1.2 Introduction to the Solution Procedures

In each of the solution algorithms one iteration considers an interior point to be a set of fixed values and looks for a set of 'changes' or 'directions' as follows.

Δx_j, changes to the x_j for $j = 1,2,\cdots,n$
Δs_j, changes to the s_j for $j = 1,2,\cdots,n$
Δy_j, changes to the y_j for $j = 1,2,\cdots,m$
Δz_j, changes to the z_j for $j = 1,2,\cdots,n$
Δw_j, changes to the w_j for $j = 1,2,\cdots,n$

If we now substitute $(x_j + \Delta x_j)$ for x_j in the equations above, and similarly substitute $(s_j + \Delta s_j)$ for s_j, $(y_j + \Delta y_j)$ for y_j, $(z_j + \Delta z_j)$ for z_j and $(w_j + \Delta w_j)$ for w_j then new equations are derived. These equations are now rearranged with the delta terms on the left and everything else (constant terms) on the right. The equations then become:

$$\sum a_{ij}\Delta x_j = b_i - \sum a_{ij} x_j \text{ for } i = 1,2,\cdots,m$$

$$\Delta x_j + \Delta s_j = u_j - x_j - s_j \text{ for } j = 1,2,\cdots,n$$

$$\sum a_{ij}\Delta y_i + \Delta z_j - \Delta w_j = c_j - \sum a_{ij} y_i - z_j + w_j \text{ for } j = 1,2,\cdots,n$$

$$\Delta x_j \Delta z_j + z_j \Delta x_j + x_j \Delta z_j = m - x_j z_j \text{ for } j = 1,2,\cdots,n$$

$$\Delta s_j \Delta w_j + w_j \Delta s_j + s_j \Delta w_j = m - s_j w_j \text{ for } j - 1,2,\cdots,n$$

The constant *m* (barrier parameter) is estimated by taking a certain proportion of the duality gap. The precise formula for this proportion can be controlled by the user.

There are various means for approximating these equations so as to obtain a viable iterative procedure. Three alternatives are available in FortMP IPM as follows:

Affine algorithm
Barrier algorithm
Predictor-corrector algorithm

Each is described below.

Once the delta terms have been determined then the new interior point for the next iteration is obtained as follows:

*New point = Previous point + Delta * Factor*

where the *Factor* is made as large as possible without taking any variable outside its bound range so that the new point remains an interior point. Actually the new point is only allowed to approach the boundary and never quite to reach it since to be on a boundary could make the whole process fail.

As will be seen, the essential step in the solution process involves finding the inverse of a large, symmetric matrix and this in fact is much the most time consuming part of the whole IPM procedure.

Remembering that at each iteration the current values of the x_j, s_j, y_j, z_j and w_j must be *interior*, an obvious requirement is to find a good interior starting point for the first iteration. Different methods are available for this and the user can specify which method should be used.

Back to Chapter contents

5.1.3 Affine, Barrier and Predictor–Corrector Algorithms

Affine Algorithm

In the affine algorithm we approximate by setting m to zero in the equations so as to get the following:

$$\sum a_{ij}\Delta x_j = b_i - \sum a_{ij}x_j \text{ for } i = 1,2,\cdots,m$$

$$\Delta x_j + \Delta s_j = u_j - x_j - s_j \text{ for } j = 1,2,\cdots,n$$

$$\sum a_{ij}\Delta y_i + \Delta z_j - \Delta w_j = c_j - \sum a_{ij}y_i - z_j + w_j \text{ for } j = 1,2,\cdots,n$$

$$\Delta x_j \Delta z_j + z_j \Delta x_j + x_j \Delta z_j = -x_j z_j \text{ for } j = 1,2,\cdots,n$$

$$\Delta s_j \Delta w_j + w_j \Delta s_j + s_j \Delta w_j = -s_j w_j \text{ for } j - 1,2,\cdots,n$$

In the 4th and 5th equations the 2nd order, quadratic terms have been dropped – this is a common approximation used in all the algorithms.

This may be thought of as attempting to go all the way – force m to zero in one step – although afterwards of course the related duality gap is recalculated on the next iteration to evaluate actual progress made.

A starting point is derived by one of the three standard methods which can be selected by the user.

Barrier Algorithm

In the barrier algorithm the equations for Δx_j, Δs_j, Δy_i, Δz_j and Δw_j are approximated as follows:

$$\sum a_{ij}\Delta x_j = 0 \text{ for } i = 1,2,\cdots,m$$

$$\Delta x_j + \Delta s_j = 0 \text{ for } j = 1,2,\cdots,n$$

$$\sum a_{ij}\Delta y_i + \Delta z_j - \Delta w_j = 0 \text{ for } j = 1,2,\cdots,n$$

$$z_j\Delta x_j + x_j\Delta z_j = m - x_j z_j \text{ for } j = 1,2,\cdots,n$$

$$w_j\Delta s_j + s_j\Delta w_j = m - s_j w_j \text{ for } j - 1,2,\cdots,n$$

Here the point $(x_j, s_j, y_j, z_j, w_j)$ is assumed to be so far feasible as to satisfy the original equations so that the first three right-hand sides above are all zero. This is guaranteed by a suitable choice of starting point so that the barrier algorithm has a starting point method of its own – different from the affine and predictor–corrector algorithms.

As before the quadratic terms have been dropped in the 4th and 5th equations.

Predictor–Corrector Algorithm

The predictor–corrector algorithm computes the direction of change, the Δ, twice. For the first computation the same equations are used as for the Affine algorithm. This gives the 'predictor' direction which may be denoted $(\Delta x_j^p, \Delta s_j^p, \Delta y_j^p, \Delta z_j^p, \Delta w_j^p)$. When this is substituted in the original and subtracted from the equations above we get the following equations for the 'corrector' direction.

$$\sum a_{ij}\Delta x_j = 0 \text{ for } i = 1,2,\cdots,m$$

$$\Delta x_j + \Delta s_j = 0 \text{ for } j = 1,2,\cdots,n$$

$$\sum a_{ij}\Delta y_i + \Delta z_j - \Delta w_j = 0 \text{ for } j = 1,2,\cdots,n$$

$$z_j\Delta x_j + x_j\Delta z_j = m - \Delta x_j^p\Delta z_j^p \text{ for } j = 1,2,\cdots,n$$

$$w_j\Delta s_j + s_j\Delta w_j = m - \Delta s_j^p\Delta w_j^p \text{ for } j = 1,2,\cdots,n$$

m is calculated after taking the full affine step on an experimental basis. Then when the corrector direction has been calculated the predictor and corrector directions are added and a fresh departure is made from the original point using the composite direction.

Fortunately these new equations do not require a new factorisation so that the extra time involved is relatively modest and the improvement in number of iterations will usually compensate for the extra calculations.

The user is recommended for the most part to employ the predictor-corrector algorithm as it will generally solve in fewer iterations than either the affine or barrier method. However certain problems, particularly those that are unusually sparse, may be speeded up by the affine algorithm sufficiently for the user to consider this a better alternative. Barrier should not be used except as a back up in case the other algorithms fail for one reason or another.

Back to Chapter contents

5.1.4 Solving the System of Equations

To show how the equations are solved we first restate them using matrices.

The following vectors and matrices are defined.

A The $m \times n$ matrix of coefficients a_{ij}.

A′ The $n \times m$ matrix of transposed coefficients $a'_{ji} (= a_{ij}$, rows become columns and columns become rows).

X The $n \times n$ diagonal matrix comprising solution point values $x_1, x_2, \cdots, x_n$ on the diagonal and zeros elsewhere.

Z, S and *W*
Diagonal $n \times n$ matrices comprising dual solution point values from the z_j, the s_j and the w_j on the diagonal and zeros elsewhere.

Δx n-vector of variables $\Delta x_1, \Delta x_2, \cdots, \Delta x_n$.

Δy m-vector of variables $\Delta y_1, \Delta y_2, \cdots, \Delta y_m$.

$\Delta z, \Delta s$ and Δw
n-vectors of variables constructed from the Δz_j, the Δs_j and the Δw_j in the same way.

R_1, R_2, R_3, R_4 and R_5
Constant right-hand side vectors of size n except for R_2 which is of size m.

Then the equations become:

$$A\Delta x = R_1 \quad (1)$$

$$\Delta x + \Delta s = R_2 \quad (2)$$

$$A'\Delta y + \Delta z - \Delta w = R_3 \quad (3)$$

$$Z\Delta x + X\Delta z = R_4 \quad (4)$$

$$W\Delta s + S\Delta w = R_5 \quad (5)$$

where R_1, R_2, R_3, R_4 and R_5 vary according to the algorithm used.

To solve these equations we may begin by eliminating all the variables $\Delta s, \Delta z$ and Δw from equations (2), (3), (4) and (5). When this is done the system is reduced to two equations as follows:

$$-(X^{-1}Z + S^{-1}W)\Delta x + A'\Delta y = R_3' \quad \text{(new RHS for 3rd equation)}$$

$$A\Delta x = R_1$$

This system could now be solved by finding the inverse of the matrix:

$$\begin{pmatrix} -D & A' \\ A & 0 \end{pmatrix}$$

where D is the diagonal matrix $(X^{-1}Z + S^{-1}W)$.

However, the number of equations can be reduced still further by eliminating the variables Δx across these two equations. This will result in the following:

$$(AD^{-1}A')Dy = R \quad \text{(new RHS)}.$$

Here we have to invert the matrix $AD^{-1}A'$, where A' denotes A transpose. The matrix is symmetric positive-definite matrix and so is inverted by the Cholesky procedure.

Back to Chapter contents

5.1.5 Determining the Starting Point

It is vital in the IPM algorithm to have a good starting interior point from which to begin iterations. Failure in this will increase the number of iterations necessary and may indeed lead to divergence or cycling so that a solution can never be found.

In the barrier algorithm a special procedure is used since the starting point, in addition to being 'interior', must also satisfy feasibility in the primal equations. To ensure this a centralised starting point is chosen and one extra row and column are added to the model with large coefficients based on a 'Big M' value designed to make the extended starting point feasible. The extra variable must converge to zero in the final solution. Users can control this procedure by weighting the Big M value as described in Section 5.2.2.

Affine and predictor–corrector algorithms can employ one of three starting point methods based on a quadratic formula to maximise the distance from boundaries in the initial interior point. The choice of method is described in Section 5.2.2.

Back to Chapter contents

5.2 Controls on the IPM Algorithms

5.2.1 Using the Algorithms

IPM is invoked in the first place with the SPECS command.

```
ALGORITHM IPM
```

which has already been described in previous chapters.

The primary control for selecting an algorithm within IPM is by one of the following SPECS commands:

```
IPM ALGORITHM AFFINE
IPM ALGORITHM BARRIER
IPM ALGORITHM PDC
```

(where 'PDC' is for 'Predictor–Corrector')

Three general features of the algorithms are controllable:

The calculation of barrier parameter, m .

The approach to a boundary when a step is made after calculating the change directions.

The feasibility and termination criteria.

The Barrier Parameter – m

m is determined by one of two alternative formulae at each iteration step.

The first is a simple formula.

$$m = \text{DGAP}/\phi(N)$$

where DGAP is the current duality gap and $\phi(N)$ is a value dependent on a user controllable parameter PHI and the column size N of the problem. For large N (greater than 5000 columns) we use:

$$\phi(N) = \text{PHI} \times N^2$$

and for smaller N we use

$$\phi(N) = \text{PHI} \times N \times \sqrt{N}$$

This simple formula is always used by the barrier algorithm. Users can give the value of PHI with the following SPECS command:

```
IPM PHI = v                    *default v = 10.0
```

In the affine algorithm m is taken to be zero and its implicit value is never used.

In the predictor–corrector algorithm the user can enforce the use of the simple formula above (controllable as before) or can specify a more complicated, adjustable formula as follows:

$$(\text{PGAP}^{\text{P}})/(\text{M} \times \text{IGAP}^2)$$

where:

IGAP represents the duality gap at the start of the iteration, that is $\sum x_j z_j + \sum s_j w_j$

PGAP represents the predicted gap after the first step has been taken with the variables incremented as they would be in a completed affine iteration step.

M is the row size of the problem

P is a power that can be supplied by the user.

The user can specify the power P with the following SPECS command:

```
IPM POWER = n                    * n=0-4   default is 4
```

If the power is specified as zero then the simple formula is always used in any case. The more complicated formula will also be replaced by the simpler formula when the duality gap reaches a low value.

With the value 3 for the power the system is in effect placing relatively less importance on reducing infeasibility, and this value has been found generally suitable. However, it may be necessary to use 2 or 1 for a problem if infeasibility takes a large number of iterations to be eliminated (or indeed is never eliminated within the maximum number of iterations allowed).

Option 4 is a modified formula, similar to option 3 but found usually to perform the best.

Power 1 should be used only if the model is numerically stable.

Approach to the Boundary

After calculating the direction of change, the system calculates how much change can be applied without having any individual variable go outside the interior point region. One change factor applies to all primal variables and another change factor to all dual variables. The factor in each case is derived from the variable which first reaches the boundary.

However the system never goes all the way using these factors, but only a fixed proportion of the distance which is given by the DARE parameter.

Users can specify DARE with the following SPECS command.

```
IPM DARE = v                     *  0  <  v  <1.0.  Default  is
0.9995
```

Lower values of DARE will result in more iterations, but a value that is too high results in instability. The process no longer converges normally and may diverge instead.

Feasibility and Termination Criteria.

There are two tolerances (EPSILON values) which the user can set to control termination.

```
IPM RELATIVE EPSILON = v       * Default is 1.0e-9
IPM FEASIBILITY EPSILON = v        * Default is 1.0e-4
```

The relative epsilon value applies to the relative duality gap – that is to say actual duality gap normalised by the following formula.

$$RGAP = DGAP/(|DOBJ| + 1)$$

where:

RGAP	is the relative duality gap
DGAP	is the actual duality gap (difference between primal and dual objective values)
DOBJ	is the dual objective value

The value of feasibility epsilon specifies the feasibility tolerance used in testing the equations.

The coarser the values specified, the earlier the termination. However, the solution may become rather inaccurate. The defaults given have been found achievable without special difficulties in most cases.

Back to Chapter contents

5.2.2 Control and Choice of the Starting Point Methods

In the barrier algorithm a single standard starting point method is always used. This is the so-called 'Big M' procedure for choosing a starting point (described in Section 5.1.5) to satisfy the primal equations.

The Big M value to be used is controlled by the user with the parameter WEIGHT (which is a simple multiplier in the formula). WEIGHT can be set by the user with the SPECS command

```
IPM BIGM WEIGHT = value
```

where 'value' is the value to be used. The default value is 0.1 and recommended values are from 0.1 to 10.0. Larger values result in greater numerical stability but also lead to increased number of iterations.

In the affine and predictor–corrector algorithms one of three starting point methods can be used as follows.

(1) A standard procedure seeks to minimize the infeasibility in the primal equations.
(2) A simplified procedure has a separate treatment for upper bounds.
(3) A similar to method (2) has scaling applied to balance the primal and dual values.

The method can be specified with the SPECS command

```
IPM STARTING POINT METHOD = n
```

where '*n*' is 1, 2 or 3. The default is method 1 and this should be retained unless you find the result unsatisfactory or you wish to experiment.

Back to Chapter contents

5.2.3 Choice of Solution Algorithm for the Equations

Since the matrix inversion procedure that is used to solve the equations occupies much the largest part of the execution time, more than 90% in many cases, you should consider carefully which version to use.

There are 3 versions available:

Standard Cholesky procedure
Cholesky with supernodes
Cholesky with extended supernodes

'Supernodes' is a procedure whereby dense patterns in the matrix $AD^{-1}A'$ (see Section 5.1.4) can be exploited so as to take advantage of special hardware – for example vector registers.

The extended supernodes version carries these ideas further and can run significantly faster by making better use of vector registers in the hardware. Speed gains up to five times faster have been observed. However, it has the disadvantage of needing significantly more memory in order to run.

The version to use is specified with the following SPECS commands:

```
IPM SOLVER CHOLESKY
IPM SOLVER SUPERNODE
IPM SOLVER XSUPERNODE
```

The default is XSUPERNODE.

Although the matrix to be inverted is positive definite, numerical stability can be a problem. It is controlled by checking the pivot-size at each step with a criterion that is set by the following SPECS command:

```
IPM TOFIX = n                    * Default = 1.0e-12
```

This is similar to the pivot tolerances used in SSX algorithms but is not quite the same because here there is no other choice available if the pivot is too small. Instead the pivot is simply changed to 1.0. This of course results in a somewhat inaccurate solution and at a later stage the solution is refined with Conjugate Gradient iterations - see the next section below.

Back to Chapter contents

5.2.4 Refinement by Conjugate Gradient iterations

The solution to the equations at each iteration, once found, is subject to a degree of error that may be calculated by substitution and comparing with the RHS. If the error is not too large it can usually be reduced by Conjugate Gradient (CG) iterations and two levels are defined:

- CG tolerance level
- Error tolerance level

The lower tolerance level is the criterion for sufficient accuracy, while the upper error level is the highest level at which CG iterations may be tried. Between these levels CG iterations are performed up to a certain maximum. The system is controlled by the following SPECS commands:

```
CHOLESKY CG TOLERANCE = v              * default = 1.0e-4
```

which sets the lower level,

```
CHOLESKY ERROR TOLERANCE = v           * default = 10.0
```

which sets the upper level, and

```
MAXIMUM CG ITERATIONS = n              * 0-3, default is 3
```

which limits how many CG steps can be used for each major IPM iteration. Zero necessarily implies OFF.

Back to Chapter contents

5.2.5 IPM Save and Restart: Iteration Limit.

IPM restart is similar to binary basis restart in the SSX algorithm. In other words the IPM algorithm periodically saves its current solution to an output file that can then be used in a subsequent run to restart the process at an advanced starting point.

With the IPM RESTART switch set ON the procedure bypasses calculation of the starting point and uses instead the solution supplied from a previous run. The following SPECS commands control this switch:

```
IPM RESTART ON
IPM RESTART OFF
```

The default is OFF.

The normal frequency for making SAVEs of the solution is at every 10 iterations. This may be modified with the command

```
IPM SAVE FREQUENCY = n
```

where '*n*' is a positive integer (default 10). The system also performs a SAVE when the IPM iteration limit is reached. The iteration limit may be changed with the command

```
MAXIMUM IPM ITERATIONS = n
```

where '*n*' is a positive integer, the default being 80. In a restart the previous iteration count is added to the limit of the new run so that there is no need to increase it when a restart is made.

When a previous solution is to be used in a restart you must remember to duplicate exactly the data preparation from the previous run. The user must repeat SCALE and/or PRESOLVE if these were used and must repeat the setting of the IPM PREFIX switch. A normal rule to follow would be to repeat all the previous controls exactly and simply add the necessary RESTART commands.

Back to Chapter contents

5.2.6 IPM-SSX Crossover Option: BASREC

The user has the option either to terminate immediately after IPM or to develop the IPM solution into a basic solution with the BASREC procedure. This recovered basis then forms an advanced starting basis for the SSX procedure – it is usually near optimal anyway.

The option is exercised via the IPM BASREC switch as follows.

```
IPM BASREC ON
IPM BASREC OFF
```

The default is ON. When the switch is set OFF the system writes the IPM solution to the output unless OUTPUT is cancelled with the OUTPUT OFF switch.

The BASREC algorithm itself comprises primal and dual 'PUSH' iterations preceded by a 'CRASH' to set up a starting basis. Normal CRASH controls apply here and in addition the user can avoid use of dual PUSH with the following command:

```
DUAL PUSH OFF                    * Default is ON
```

(ON may also be specified). It is not recommended to use OFF here because the final basis must be corrected with the PRIMAL algorithm which can take many more iterations. Dual PUSH iterations are slower but limited in number.

The following command may be used to restart BASREC from the IPM solution of a previous run.

```
BASREC RESTART ON
BASREC RESTART OFF
```

the switch being OFF by default. If set to ON the whole IPM procedure is skipped, including the rather lengthy initialisations.

FortMP IPM carries out an automatic IPM SAVE when it terminates so that the BASREC RESTART feature can be used in a later run. This will be found very useful if the user wishes to stop after IPM and then do the BASREC and optimisation with SSX in a subsequent run.

Back to Chapter contents

5.2.7 Miscellaneous IPM and BASREC Commands

The following SPECS command determines the log level of IPM in the same manner as for PRIMAL and DUAL.

```
IPM LOG LEVEL = n                * n = 0-4, default 1
```

For the BASREC (crossover to SSX) stage the following commands are used:

```
PUSH LOG LEVEL = n               * n = 0-4, default 1
PUSH LOG FREQUENCY = n           * default: n = 10
```

these commands being similar to ‘PRIMAL LOG LEVEL’ and ‘PRIMAL LOG FREQUENCY’.

The following SPECS commands are only available in certain versions of FortMP. (Please see the Implementation Guide for details.) A switch is set to invoke a graphical display which shows the density pattern of the matrices and the progress of the iterations. Copies of the display will be left in files ‘*model*.a’, ‘*model*.b’, ‘*model*.c’ and ‘*model*.d’ if this command is successfully activated.

```
IPM GRAPHICAL DISPLAY ON
IPM GRAPHICAL DISPLAY OFF
```

The default is OFF.

Back to Chapter contents

5.3 Summary of SPECS Commands

The following SPECS commands have been introduced so far in this manual. Those introduced in this chapter are highlighted thus:

```
IPM ALGORITHM AFFINE
```

```
ALGORITHM IPM
```

Thise command is used to specify IPM as the primary solution algorithm for the continuous LP problem..

```
IPM ALGORITHM AFFINE
IPM ALGORITHM BARRIER
IPM ALGORITHM PDC
```

These commands invoke the algorithm to be used within IPM. The default is PDC (Predictor-Corrector). See 5.2.1

```
IPM PHI = v                    * Default is 10.0
```

This command sets the PHI control in the elementary formula for calculating *m*.. See 5.2.1.

```
IPM POWER = n                  * n=0-3. Default is 4
```

This parameter sets the 'POWER' applied to the numerator in the more advanced formula for μ, which is used in the earlier phase of the predictor-corrector algorithm. See 5.2.1.

```
IPM DARE = v                   * Default is 0.9995
```

‘DARE’ is a fraction between 0 and 1 controlling how closely to approach the nearest boundary when moving from one interior point to the next. See 5.2.1.

```
IPM RELATIVE EPSILON = v       * Default is 1.0e-7
```

The relative epsilon is the tolerance within which the duality gap can be considered to be zero. Thus the optimum solution is reached provided the point is feasible. See 5.2.1.

```
IPM FEASIBILITY EPSILON = v        * Default is 1.0e-4
```

The feasibility epsilon is the tolerance governing feasibility of the current primal and dual solutions. See 5.2.1.

```
IPM BIGM WEIGHT = v              * Default is 0.1
```

This value is the BIGM weight used in the starting point method for the barrier algorithm. See 5.2.2.

```
IPM STARTING POINT METHOD = n * Default is 3
```

This command selects one of three starting point methods for the predictor–corrector algorithm. See 5.2.2.

```
IPM SOLVER CHOLESKY
IPM SOLVER SUPERNODE
IPM SOLVER XSUPERNODE
```

These commands select the solution mechanism to be used. The default is 'XSUPERNODE'. See 5.2.3.

```
IPM TOFIX = n                          * default = 1.0e-12
```

This command sets the criterion for minimum pivot size in the Cholesky factorisation. See 5.2.3.

```
CHOLESKY CG TOLERANCE = v          * default = 1.0e-4
CHOLESKY ERROR TOLERANCE = v             * default = 10.0
```

These commands set lower and upper levels to the solution error between which CG iterations are used to refine the major IPM iterations. See 5.2.4.

```
MAXIMUM CG ITERATIONS = n                * 0-3, default is 3
```

This command limits the number of CG steps taken at each major IPM iteration. See 5.2.4.

```
IPM RESTART ON
IPM RESTART OFF
```

This command specifies whether to 'RESTART' the IPM algorithm. Default is OFF. See 5.2.5.

```
IPM SAVE FREQUENCY = n                    * default n=10
```

This command specifies the frequency for making a SAVE in IPM. See 5.2.5.

```
MAXIMUM IPM ITERATIONS = nnn            *  default 80
```

This command sets the termination limit for IPM. A SAVE is made before exit at termination. See 5.2.5.

```
IPM BASREC ON
IPM BASREC OFF
```

When 'ALGORITHM IPM' has been selected the user can set this switch OFF in order to halt execution and print the output immediately on reaching the IPM solution. The default is ON. See 5.2.6.

```
DUAL PUSH OFF                          * default ON
DUAL PUSH ON
```

The OFF option cancels execution of DUAL PUSH (not recommended). See 5.2.6

```
BASREC RESTART ON                    * default OFF
BASREC RESTART OFF
```

Provided that the IPM algorithm has completed with an optimum solution then BASREC can be restarted in a subsequent run without repeating the IPM calculations again. See 5.2.6.

```
IPM LOG LEVEL = n
```

These commands specify the 'level' of the output to be sent to the log file in each major algorithm. Range is 1–4, default is 1 in each case. See 5.2.7.

```
PUSH LOG LEVEL = n                   * 0-4, default is 1
PUSH LOG FREQUENCY = n               * default = 10
```

These commands control the log level and frequency of 'PUSH' iterations during the BASREC (crossover to SSX) algorithm. See 5.2.7

```
IPM GRAPHICAL DISPLAY ON
IPM GRAPHICAL DISPLAY OFF
```

Certain implementations of FortMP have the graphical feature which displays the pattern of non-zeros in the matrices followed by a progress display of the iterations. The default is OFF. See 5.2.7.

Back to Chapter contents

Chapter 6 : Mixed Integer Programming (MIP)

Contents

6.1 Introduction to MIP

In many real-world problems, it is often impossible to represent certain features of a problem using only linear constraints and continuous variables. In modelling a real world problem it is often necessary to represent discrete activities by variables which are restricted to take only integer values. Moreover, there are non-linear variable separable functions of one variable which can be approximated by a piecewise linear function. Within Mathematical Programming, these models are often given the generic name of Mixed/Integer Programming (MIP/IP).

The diversity of applicable MIP/IP models stems from the fact that in many practical problems, activities and resources, like machines and operators are indivisible. Many problems require the determination of yes-no decisions, which can modelled by introducing binary variables which are integer variables restricted to the values zero and one representing 'no' and 'yes' respectively. Also, many optimisation problems of a combinatorial nature can be formulated as IPs. The range of applications includes problems such as the distribution of goods, production scheduling and machine sequencing. They also include planning problems such as capital budgeting, facility location, crew and aircraft scheduling, design problems such as communications and transportation network design, very large scale integrated circuit (VLSI) design and the design of automated production systems.

Back to Chapter contents

6.2 MIP Problem, data types, and problem definition

An MIP is a mathematical program which in addition to its usual linear restrictions includes some discrete (integer) restrictions on some or all the variables of the model. These restrictions can be categorised as one of the following:

(i)Zero-One or Binary variables,	$x_j \in \{0,1\}$,
(ii)General Integer variables,	$x_j \in Z$,
(iii)Semi-Continuous variables,	$x_j = 0$ or $0 < l_j \leq x_j \leq u_j$,
(iv)Special Ordered Sets of type One,	*See 6.2.3*
(v)Special Ordered Sets of type Two.	*See 6.2.4*

The definition and the modelling applications of each group of variables and sets are given separately in the following sections.

Back to Chapter contents

6.2.1 Binary and Integer Variables

A *zero-one or binary variable* (BV) is a variable which can take either the value zero or one in a given model. An important and common use of zero-one variables is to represent binary, yes or no choices. Consider a situation where one has to make a decision whether or no to perform a number of activities and suppose that the problem (or at least part of the problem) is to decide which activities to select. To represent such a condition in a model we use a binary variable x_j for each activity and let

$x_j = 1$ if activity j is performed,
$x_j = 0$ if activity j is not performed.

Knapsack, Facility location, Network flow, Set covering, Set packing, Set partitioning, Travelling salesman, Scheduling and Logic programming provide instances of models which include binary variables.

General Integer variables are used to model entities where non-integer values are not meaningful. For example, the production of items in batches of a fixed size can only be realistically modelled if the model excludes the possibility of non-integer numbers of batches being produced.

Back to Chapter contents

6.2.2 Semi-Continuous Variables

A *Semi-Continuous variable* (SC) is a variable which may take the value zero or any value within a certain range bounded by positive finite lower and upper bounds (figure 1). Thus, these variables are used to represent activities which, if used at all, must have a usage above a specified minimum level.

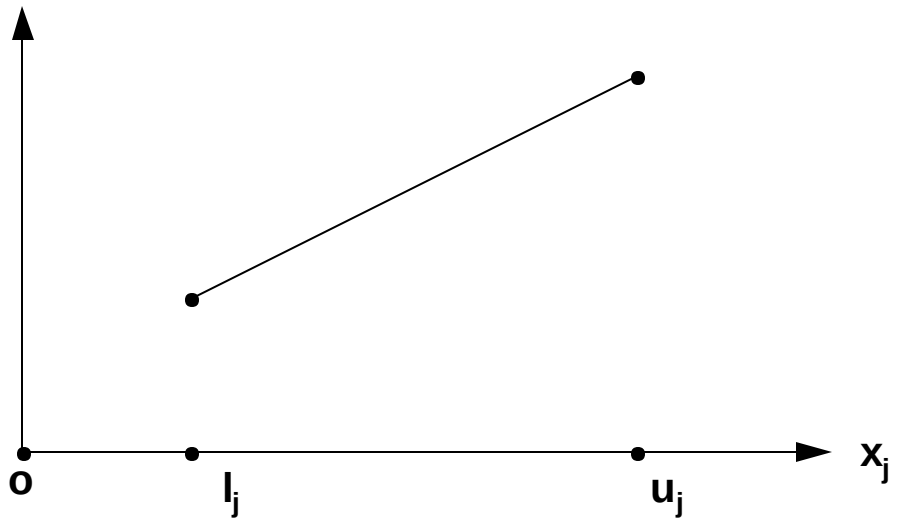


Figure 1

The situation arises (for example) in certain blending problems where materials must be excluded from the blend if they cannot be used in significant quantities. It has been modelled by the introduction of a decision-variables to represent whether or not a material is to be included.

Consider a blending problem which has the requirement that, if the j^{th} material is used at all then it must be used at a level between l_j and u_j. Define a zero-one variable d_j in connection with each material amount x_j. The semi-continuous requirement is established by the two constraints:

$$l_j d_j \pounds x_j$$
$$x_j \pounds u_j d_j$$

of a type known respectively as variable lower bound and variable upper bound. However by defining x_j as a semi-continuous variable this complexity can be avoided and the solution process is more efficient.

In the present version of FortMP the continuous-range lower bound must be 1.0. This is not a real restriction however as the modeller may easily scale x_j by the lower bound reciprocal $1.0/l_j$ ($l_j \neq 0$) so that this is achieved.

6.2.3 Special Ordered Sets of Type One

A *Special Ordered Set of type One* (SOS1) is defined to be a set of variables for which not more than one member from the set may be non-zero in a feasible solution. All such sets are mutually exclusive of each other, the members are not subject to any other discrete conditions and are grouped together consecutively in the data.

The normal use of an SOS1 is to represent a set of mutually exclusive alternatives ordered in increasing values of size, cost or some other suitable units appropriate to the context of the model. This representation is a discrete programming extension of the separable programming model. There is a strong implied assumption that a nonlinear function represented in this way is single valued over the range of its argument.

Consider a function g(y) represented by the points $P_1,...,P_K$ as shown in figure 2.

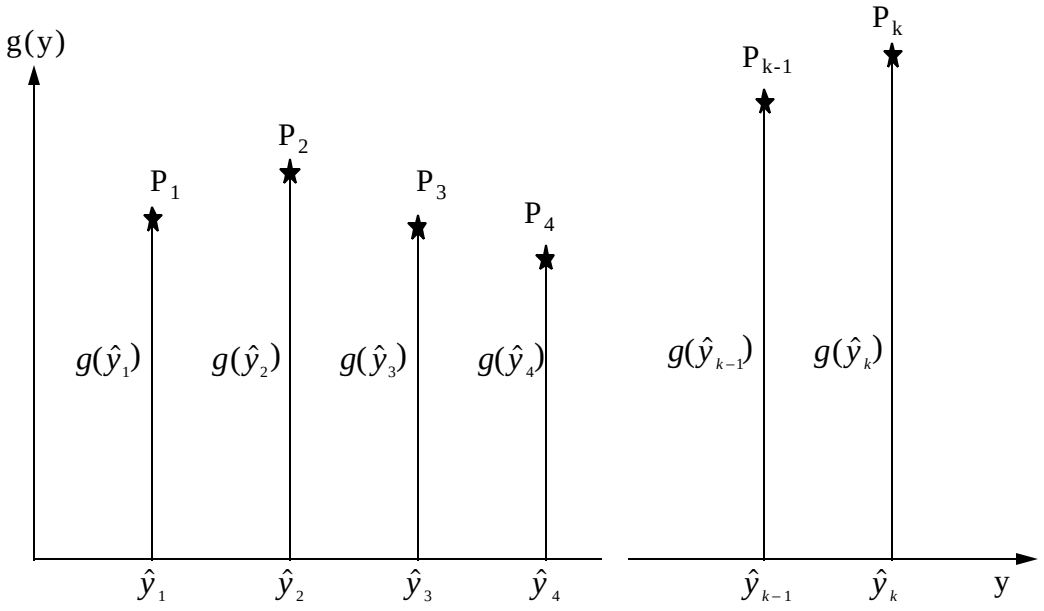


Figure 2

Given the tabulated coordinates $(\hat{y}_k, g(\hat{y}_k))$ $k=1,\ldots,K$, the function *g(y)* may be represented as

$$g(y) = g(\hat{y}_1)x_1 + g(\hat{y}_2)x_2 + \cdots + g(\hat{y}_K)x_K \quad (1)$$

where

$$\hat{y}_1x_1 + \hat{y}_2x_2 + \cdots + \hat{y}_Kx_K - y = 0, \quad y \geq 0 \quad (2)$$

$$x_1 + x_2 + \cdots + x_K = 1, \quad x_k \geq 0, \quad k = 1,2,\ldots,K \quad (3)$$

The discrete function can take only one of the K possible values weighted by the variables x_k, of which only one can be non-zero, and that must have the value one.

This requirement could be expressed by restricting each x_k to be a binary variable but the alternative of defining them collectively as a special ordered set of type one, which is a direct statement of their nature, leads to a more efficient solution process.

The weighting variables x_k are called *special ordered set type one variables* and the rows (1), (2), and (3) are called *function rows*, *reference rows* and *convexity rows* respectively. Should the SOS1's not represent a modelling of discrete, separable variables then none of these rows need actually exist, but there is an advantage to the system if it is aware of the reference rows at least.

Back to Chapter contents

6.2.4 Special Ordered Sets of Type Two

A *Special Ordered Set of type Two* (SOS2) is a set of consecutive variables in which not more than two adjacent members may be non-zero in a feasible solution. All such sets are mutually exclusive of each other, the members are not subject to any other discrete conditions and each set is grouped together consecutively in the data.

SOS2s were introduced to make it easier to find global optimum solutions to problems containing piecewise linear approximations to a nonlinear function of a single argument (as in classical Separable Programming). The overall problem has an otherwise LP or an IP structure except for such nonlinear functions.

Consider the function $f(y)$ illustrated in Figure 3 as a piecewise linear function in one variable defined over the closed intervals $[\hat{y}_k, \hat{y}_{k+1}]$, $k = 1,\ldots,K-1$, where the coordinates $(\hat{y}_k, f(\hat{y}_k))$, $k = 1,\ldots,K$, represent points $P_1,\ldots,P_K$.

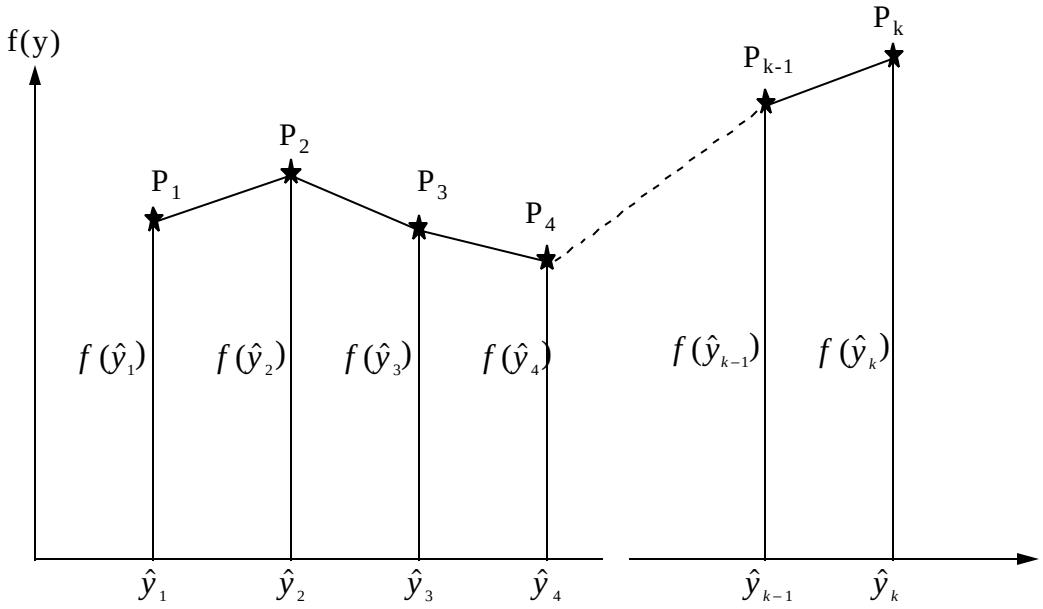


Figure 3

Any point y in the closed interval $[\hat{y}_k, \hat{y}_{k+1}]$ may be written as

$$y = x_k \hat{y}_k + x_{k+1} \hat{y}_{k+1}$$

where

$$x_k + x_{k+1} = 1 \text{ and } x_k,\ x_{k+1} \geq 0 .$$

Similarly, as $f(y)$ is linear in the interval, it can be written as

$$f(y) = f(\hat{y}_k) x_k + f(\hat{y}_{k+1}) x_{k+1} .$$

This leads to the representation of $f(y)$ using a set of weighting variables, x_k, $k = 1,\ldots,K$, by the equality

$$f(y) = f(\hat{y}_1) x_1 + f(\hat{y}_2) x_2 + \cdots + f(\hat{y}_K) x_K \tag{4}$$

where

$$\hat{y}_1 x_1 + \hat{y}_2 x_2 + \cdots + \hat{y}_K x_K - y = 0, \quad y \geq 0 \tag{5}$$

$$x_1 + x_2 + \cdots + x_K = 1, \quad x_k \geq 0, \quad k = 1,2,\ldots,K . \tag{6}$$

Plus the added condition that not more than two adjacent variables can be non-zero at any one time.

The weighting variables x_k are called the *special ordered set type two variables* and the rows (4), (5), and (6) are called *function rows, reference rows* and the *convexity rows* respectively, as in equations (1), (2) and (3) of section 6.2.3. Should the SOS2's not represent separable functions then none of these rows need actually exist, but there is an advantage to the system if it is aware of the reference rows at least.

Back to Chapter contents

6.2.5 MIP model definition

The classes of discrete variables are described in the previous sections, as Binary, General integer, Semi-Continuous variables and Special Ordered Set variables of type One and type Two. A statement of the general discrete programming problem (DPP), which contains the above types of variables and sets as well as continuous variables is set out below. Consider the index sets N1,..., N7 which are used to specify the different variable types.

N_1 = set of indices of bounded variables

$$l_j \le x_j \le u_j, \quad j \in N_1 \qquad (7)$$

where either l_j or u_j may be infinite but not both

N_2 = set of indices of free variables

$$-\infty < x_j < +\infty, \quad j \in N_2. \qquad (8)$$

N_3 = set of indices of binary variables

$$x_j = 0 \text{ or } 1, \quad j \in N_3. \qquad (9)$$

N_4 = set of indices of general integer variables

$$l_j \le x_j \le u_j, \quad \textit{and} \quad x_j \text{ integer}, \quad j \in N_4. \qquad (10)$$

N_5 = set of indices of semi-continuous variables

$$\begin{array}{l} \textit{either} \begin{cases} 0 < 1 \le x_j \le u_j \\ \textit{or} \quad x_j = 0 \end{cases}, \quad j \in N_5 \end{array} \qquad (11)$$

$N_6 = \bigcup_l N_{6l}$ where N_{6l} is the set of indices of the l_{th} SOS1–type variable-set, these sets being mutually exclusive of each other,

$$x_j \ge 0, \quad j \in N_{6l}, \quad l = 1, \ldots, L' \qquad (12)$$

L' = *Number of SOS1s*.

and only one x_j can be non-zero in each set.

$N_7 = \bigcup_l N_{7l}$ where N_{7l} is the set of indices of the l_{th} SOS2 type variable-set - these sets being mutually exclusive of each other.

$$x_j \geq 0\,,\ j \in N_{7k}\,,\ l = 1,\ldots,L'' \qquad (13)$$
L'' = *Number of SOS2s*.

and at most two adjacent x_j's can be non-zero in each set.

These index sets are mutually exclusive, that is $N_p \bigcap N_q = \varnothing$ for $p \neq q,\quad p = 1,..,7$ 1 and thus the total numbet of variables defined is n where:

$$n = \sum_{p=1}^{7} |N_p| \qquad (14)$$

where $|N_p|$ is the cardinality of the set N_p.

Using these set definitions a general discrete programming model may be presented as

$$\text{Minimise } \sum_{j=1}^{n} c_j x_j \qquad (15)$$

subject to constraints:

$$\sum_{j=1}^{n} a_{ij} x_j \begin{Bmatrix} \leq \\ \geq \\ = \end{Bmatrix} b_i\,,\ \ i = 1,\ldots,P \qquad (16)$$

where:

$$\begin{array}{ll} l_j \leq x_j \leq u_j & \\ -\infty < l_j, u_j < +\infty & ,\ j \in N_1 \bigcup N_4 \\ -\infty < x_j < +\infty & ,\ j \in N_2 \\ 0 \leq x_j \leq 1 & ,\ j \in N_3 \\ 0 \leq x_j \leq u_j & ,\ j \in N_5 \\ x_j \geq 0 & ,\ j \in N_6 \bigcup N_7 \end{array} \qquad (17)$$

with further discrete restrictions

$$
\begin{aligned}
& x_j = 0 \text{ or } 1 \quad , \text{ if } j \in N_3 \\
& x_j \text{ integer} \quad , \text{ if } j \in N_4 \\
& \text{either } \begin{cases} 0 < l_j \le x_j \le u_j \\ \text{or} \quad x_j = 0 \end{cases} \text{ if } j \in N_5 . \qquad (18) \\
& \textit{Only one } x_j \textit{ can be non-zero if } j \in N_{6l}, \textit{ and at most} \\
& \textit{two adjacent } x_j \textit{ can be non-zero if } j \in N_{7l}
\end{aligned}
$$

The model therefore has *m* rows given by (16) and *n* columns given by (14)

In the normal usage of special ordered sets to represent separable variables (discrete or piecewise linear) the constraints (16) include function rows, reference rows and convexity rows as defined in 6.2.3 and 6.2.4

Back to Chapter contents

6.3 MIP Data Preparation; Marker lines

Integer, binary and semi-continuous variables may be specified in the MPS form input data by using codes in the BOUNDS section as illustrated in table 14 of Chapter 2.

As an alternative the user may employ 'MARKER' lines in the COLUMNS section to specify a list of consecutively sequenced binary or integer variables. 'MARKER' lines in the COLUMNS section are also the means whereby special ordered sets of type SOS1 and SOS2 are defined. Their usage in FortMP corresponds to IBM's extension to MPS format in their MPSX system. Certain variations are also available to conform with usage by other systems.

Back to Chapter contents

6.3.1 Defining Binary, Integer and Semi-continuous variables in the BOUNDS section

Binary, Integer and Semi-continuous variables can be defined in the BOUNDS section of the MPS file using the following type codes in field 1:

BV indicates that the variable in Field 3 is a binary variable restricted to discrete values zero and one,

UI indicates that the variable defined in Field 3 is an integer variable with upper bound given by the value in field 4 rounded down to the nearest integer. The lower bound is 0, unless explicitly defined with a record that has LO or LI as the type code in field one.

LI indicates that the variable defined in Field 3 is an integer variable with lower bound given by the value in field 4 rounded up to the nearest integer. The upper bound is infinite, unless explicitly defined with a record that has UP or UI as the type code in field one.

SC Indicates that the variable named in Field 3 is semi-continuous with upper bound of the continuous range given by the value in Field 4. An SC specification cannot be combined with any other BOUNDS specification for that variable. Lower bound of the continuous range is implicitly set to 1.0 (see section 6.2.2).

Back to Chapter contents

6.3.2 Marker lines

Marker lines are used in the COLUMNS section of the input data to define a list of consecutively sequenced variables. Four types of marker line can be used as follows:

'INTORG'	Indicates the beginning of a consecutive sequence of Integer or Binary variables
'INTEND'	Indicates the end of a consecutive sequence of Integer or Binary variables
'SOSORG'	Indicates the beginning of a special ordered set (type SOS1 or SOS2)
'SOSEND'	Indicates the end of a special ordered set (type SOS1 or SOS2)

A marker line has the keyword 'MARKER' (quotes included) in Field 3 of the standard MPS-form layout and the type keyword 'xxxORG' or 'xxxEND' either in Field 4 or in Field 5 (with quotes included).

The full layout is as follows:

Field 1 (2-3)	Field 2 (5-12)	Field 3 (15-22)	Field 4 (25-32)	Field 5 (40-47)	Field 6 (50-61)
Blank	Label	'MARKER'	'INTORG'	blank	blank
Blank	Label	'MARKER'	'INTEND'	blank	blank
SOS type	Label	'MARKER'	'SOSORG'	REF-row	blank
Blank	Label	'MARKER'	'SOSEND'	blank	blank

Each marker line must appear between one column and another - a column may not be split by a marker.

It is not allowed to overlap sections of the data with marker lines since all sets must be mutually exclusive. Thus markers are always to be in pairs with the 'ORG' type marker followed by its corresponding 'END' type marker before any other marker can appear.

Fields 4 and 5 are interchangeable - the marker type keyword may be placed in field 5 and when 'SOSORG' is placed in field 5 the 'REF-row' data is placed in field 4.

Blank fields should be left blank (but may contain unused material).

The 'Label' field is intended for the user to attach a name or label to a set. For this purpose the 'END' marker should have the same label as its corresponding 'ORG' marker. However this is quite optional.

'SOS type' in Field 1 is described below.

Back to Chapter contents

6.3.3 Defining Integer and Binary Variables with Markers

'INTORG' and 'INTEND' markers define a list of consecutively sequenced binary or integer variables. The integer bounds may be defined with 'LO' and 'UP' records in the BOUNDS section of the data, and the system distinguishes binary variables as those with a bound range zero to one.

If no records in the BOUNDS section are provided for one or more of the variables outlined by 'INTORG' and 'INTEND' then these variables receive a default bound specification LO-value zero, UP-value infinite. Thus they become simple non-negative variables restricted to integer values in any feasible solution.

The default bound can be changed with the following alternative SPECS-commands:

INPUT INTORG UPPER BOUND = n (default infinity)
INPUT INTORG BOUND = n (default infinity)

which in effect supplies an implicit UP record in the BOUNDS section. By setting n = 1 with the command

INPUT INTORG BOUND = 1

the default specification becomes binary (BV) rather than integer (UI).

Back to Chapter contents

6.3.4 Defining a Special Ordered Set

'SOSORG' and 'SOSEND' markers are used to outline each Special Ordered Set in the model.

In Field 1 of the 'SOSORG' marker there is a 2-character code which specifies the SOS-type as follows:

S1 indicates type 1
S2 indicates type 2

In Field 5 (or in Field 4 if Field 5 contains 'SOSORG') the user can enter the name of the reference row of the SOS. If it is left blank the system assumes an implicit reference row having whole number coefficients 1, 2, 3,.... for the set members in their order.

Back to Chapter contents

6.3.5 An Example

A small Discrete Programming model is set out here to show how binary, integer, semi-continuous and SOS type variables are presented in MPSX format. Consider the following model M(i):

Minimise:

$$3x_1 + 4x_2 + 5x_3 - x_4 + f_1(x_5) + f_2(x_6) + f_3(x_7)$$

subject to:

$$2x_1 + 3x_2 - 4x_3 + x_4 + x_5 + 2x_6 + 3x_7 \leq 25$$
$$5x_2 + 3x_3 - x_4 + 3x_6 \geq 50$$
$$6x_1 + 3x_2 + 2x_5 + x_7 = 100$$
$$x_1 - 100x_4 \leq 0$$

where:

- $0 \leq x_1 \leq 100$
- x_2 is integer, and $0 \leq x_2 \leq 20$
- x_3 is semi-continuous: $x_3 = 0$ *or* $1 \leq x_3 \leq 10$
- x_4 is binary: $x_4 = 0$ *or* 1
- $f_1(x_5)$ is a discrete separable function comprising the following co-ordinates

x_5=	0	1	2	3	4
f_1=	2	3	4	3	2

  undefined otherwise

- $f_2(x_6)$ is a piecewise linear form connecting the following set of co-ordinates:

x_6=	0	2	3
f_2=	0	2	1

  undefined for $x_6 < 0$ and for $x_6 > 3$

- $f_3(x_7)$ is a piecewise linear form connecting the following set of co-ordinates:

x_7=	1	5	10	15
f_3=	1	10	25	-7.5

  undefined for $x_7 < 1$ and for $x_7 > 15$

Using SOS type variables $f_1(x_5)$, $f_2(x_6)$ and $f_3(x_7)$ are individually modelled as follows:

- Let x_{11}, x_{12}, x_{13}, x_{14} and x_{15} be a special ordered set of type SOS1 where:

$$\begin{aligned} 1 &= x_{11} + x_{12} + x_{13} + x_{14} + x_{15} \\ x_5 &= x_{12} + 2x_{13} + 3x_{14} + 4x_{15} \\ f_1(x_5) &= 2x_{11} + 3x_{12} + 4x_{13} + 3x_{14} + 2x_{15} \end{aligned}$$

- Let x_{21}, x_{22} and x_{23} be a special ordered set of type SOS2 where:

$$\begin{aligned} 1 &= x_{21} + x_{22} + x_{23} \\ x_6 &= 2x_{22} + 3x_{23} \\ f_2(x_6) &= 2x_{22} + x_{23} \end{aligned}$$

- Let x_{31}, x_{32}, x_{33} and x_{34} be a special ordered set of type SOS2 where:

$$\begin{aligned} 1 &= x_{31} + x_{32} + x_{33} + x_{34} \\ x_7 &= x_{31} + 5x_{32} + 10x_{33} + 15x_{34} \\ f_3(x_7) &= x_{31} + 10x_{32} + 25x_{33} - 7.5x_{34} \end{aligned}$$

By appending the above SOS formulation to M(i) the DPP model can be stated as M(ii):

Minimise:

$$\begin{aligned} & 3x_1 + 4x_2 + 5x_3 - x_4 \\ & \quad + 2x_{11} + 3x_{12} + 4x_{13} + 3x_{14} + 2x_{15} && \textit{(function row)} \\ & \quad + 2x_{22} + x_{23} && \textit{(function row)} \\ & \quad + x_{31} + 10x_{32} + 25x_{33} - 7.5x_{34} && \textit{(function row)} \end{aligned}$$

subject to:

$$\begin{aligned} & 2x_1 + 3x_2 - 4x_3 + x_4 + x_5 + 2x_6 + 3x_7 \leq 25 \\ & 5x_2 + 3x_3 - x_4 + 3x_6 \geq 50 \\ & 6x_1 + 3x_2 + 2x_5 + x_7 = 100 \\ & x_1 - 100x_4 \leq 0 \\ & x_{11} + x_{12} + x_{13} + x_{14} + x_{15} = 1 && \textit{(convexity row)} \\ & x_{12} + 2x_{13} + 3x_{14} + 4x_{15} - x_5 = 0 && \textit{(reference row)} \\ & x_{21} + x_{22} + x_{23} = 1 && \textit{(convexity row)} \\ & 2x_{22} + 3x_{23} - x_6 = 0 && \textit{(reference row)} \\ & x_{31} + x_{32} + x_{33} + x_{34} = 1 && \textit{(convexity row)} \\ & x_{31} + 5x_{32} + 10x_{33} + 15x_{34} - x_7 = 0 && \textit{(reference row)} \end{aligned}$$

where:

- $0 \pounds x_1 \pounds 100$
- x_2 is integer, and $0 \pounds x_2 \pounds 20$
- x_3 is semi-continuous: $x_3 = 0$ or $1 \pounds x_3 \pounds 10$
- x_4 is binary: $x_4 = 0$ or 1
- x_{11}, x_{12}, x_{13}, x_{14} and x_{15} are a set of type SOS1
- x_{21}, x_{22} and x_{23} are a set of type SOS2
- x_{31}, x_{32}, x_{33} and x_{34} are a set of type SOS2

The input data for the above model M(ii) is shown below,

```
NAME          MGINT
ROWS
 N  OBJ
 N  'MARKER'
 L  COS1
 G  COS2
 E  COS3
 L  COS4
 E  CON1
 E  CON2
 E  CON3
 E  REF1
 E  REF2
 E  REF3
COLUMNS
    X1        OBJ       3.0            COS1      2.0
    X1        COS3      6.0            COS4      1.0
    X2        OBJ       4.0            COS1      3.0
    X2        COS2      5.0            COS3      3.0
    X3        OBJ       5.0            COS2      3.0
    X3        COS1      -4.0
    X4        OBJ       -1.0           COS1      1.0
    X4        COS2      -1.0           COS4      -100.0
    X5        COS1      1.0            COS3      2.0
    X5        REF1      -1.0
    X6        COS1      2.0            COS2      3.0
    X6        REF2      -1.0
    X7        COS1      3.0            COS3      1.0
    X7        REF3      -1.0
 S1 S1SET1    'MARKER'  'SETORG'       REF1      1.0
    X11       OBJ       2.0
    X11       CON1      1.0
    X12       OBJ       3.0            REF1      1.0
    X12       CON1      1.0
    X13       OBJ       4.0            REF1      2.0
    X13       CON1      1.0
    X14       OBJ       3.0            REF1      3.0
    X14       CON1      1.0
    X15       OBJ       2.0            REF1      4.0
    X15       CON1      1.0
```

```
    S1SET1E   'MARKER'  'SETEND'
 S2 S1SET2    'MARKER'  'SETORG'       REF2      1.0
    X21       CON2      1.0
    X22       OBJ       2.0            REF2      2.0
    X22       CON2      1.0
    X23       OBJ       1.0            REF2      3.0
    X23       CON2      1.0
    S1SET2E   'MARKER'  'SETEND'                 'E'
 S2 S1SET3    'MARKER'  'SETORG'       REF3      1.0
    X31       OBJ       1.0            REF3      1.0
    X31       CON3      1.0
    X32       OBJ       10.0           REF3      5.0
    X32       CON3      1.0
    X33       OBJ       25.0           REF3      10.0
    X33       CON3      1.0
    X34       OBJ       -7.5           REF3      15.0
    X34       CON3      1.0
    S1SET3E   'MARKER'  'SETEND'                 'E'
RHS
    MOZ1      COS1      25.0           CON1       1.0
    MOZ1      COS2      50.0           CON2       1.0
    MOZ1      COS3      100.0          CON3       1.0
BOUNDS
 UP MOZ2      X1        100.0
 UI MOZ2      X2        20.0
 SC MOZ2      X3        10.0
 BV MOZ2      X4        1.0
ENDATA
```

6.4 Branch and Bound algorithm

6.4.1 Branch and Bound - the Background

Branch and Bound (B&B) is a technique for solving certain constrained optimisation problems. It is particularly important for solving those problems whose solution by complete enumeration would be prohibitively expensive. A wide variety of problems of this kind arise in Operational Research, Combinatorial Optimisation and Artificial Intelligence. Taking into account practical computational experiences, B&B is considered to be the most robust and widely used technique for solving MIP/IP problems.

The B&B algorithm carries out a progressive partitioned search of the solution space of a given problem. This is done by exploring a search tree and usually only a small proportion of the solution space is searched. The remainder of the search tree can be partly eliminated using the bound derived from a good discrete/integer feasible solution. Such a solution is proven to be optimal when the rest of the tree is fully eliminated.

Back to Chapter contents

6.4.2 Branch and Bound - the Algorithm

In order to describe the B&B algorithm we first introduce a few definitions. We consider an MIP problem defined by relations (16), (17) and (18) in section 6.2.5 and note that it is a minimisation problem. The objective function is always specified as minimisation for the purpose of these discussions.

A solution is said to be *LP feasible* if it satisfies all the linear constraints (16, and 17 of section 6.2.5). The problem with only these constraints and all discrete restrictions ignored is referred to as the *Relaxed LP* or *LPR*. The objective value of the LPR is denoted z^{l}_{LPR}.

A solution is said to be *integer feasible* if it satisfies all the discrete restrictions (18 of section 6.2.5) as well as all the linear constraints (16, 17 of section 6.2.5).

An *IP optimum solution* is an integer feasible solution of the problem (if one exists) which in respect of the objective function value, is either better or at least as good as all other integer feasible solutions of that problem.

The term *discrete entity* is used to represent any binary, general integer or semi-continuous variable and any set of type SOS1 and SOS2 (not an individual set member). If the solution does not satisfy the discrete restrictions of an entity then that entity is considered *violated* in the solution.

z^{l}_{IP} is called the *cut off value*. It is initially fixed to a large positive number and during the tree search it is updated by the objective function value of the best integer feasible solution found so far.

The steps of the B&B algorithm can be stated as follows:

Step 0) Initial Step:

If an optimum solution to the LPR does not exist then go to Exit, else if the solution is integer feasible go to Exit. Otherwise set the cutoff value z^l_{IP} *to a large positive value, prepare a space for the list of sub-problems (nodes) of the search tree and continue.*

Step 1) Variable selection:

Select a discrete entity that is violated in the solution of the current node P_k.

Step 2) Branching:

Partition the continuous range of this entity and create two new sub-problems, nodes P_{k+1} *and* P_{k+2} *of the tree linked to the parent* P_k. *Store the current basis as starting point for the solution of* P_{k+1} *and* P_{k+2}.

Step 3) Node Selection:

If the list of unsolved sub-problems is empty then go to Exit, else select an unsolved sub-problem P_k *from the list of nodes.*

Step 4) Solving:

Recover the starting basis for the selected node and solve the selected sub-problem with the Primal or the Dual algorithm. If Dual is used the sub-problem may be discontinued should its objective function value become greater than or equal to z^l_{IP}; *in this case repeat from step 3. Denote the eventual optimum solution* z^l_P.

Step 5) Bounding:

If the sub-problem has no feasible solution or if z^l_P *is greater than or equal to* z^l_{IP} *then delete the node and repeat from step 3. If the solution is not integer feasible then repeat from step 1.*

Step 6) Integer Solution:

Set integer solution marker. If z^l_P *is less than* z^l_{IP} *then update* z^l_{IP} *and save the current problem bounds and basis for the final solution. If* z^l_P *is equal to* z^l_{LPR} *then go to Exit. Otherwise the node is deleted and we repeat from step 3.*

Exit)

If the integer solution marker is not set to any integer feasible solution and the list is empty then no feasible (integer) solution exists to the problem. Otherwise output the best integer solution.

Back to Chapter contents

6.4.3 The Branching Mechanism, UP and DOWN branching

Branching (Step 2 above) partitions the data space by adding a constraint or constraints in such a way that the selected discrete entity is forced to a feasible state. In some cases two or more branches may be needed for the same discrete entity.

When branching is performed the two sub-nodes correspond to an UP branch and a DOWN branch. For the different types of discrete entity these branches are formed as follows:

Binary variable branching

A binary variable is fixed to one on the UP branch, to zero on the DOWN branch.

Integer variable branching

The fractional solution-value to an integer variable forms a division-point. On the UP branch the lower bound is increased to the next integer above, on the DOWN branch the upper bound is decreased to the next integer below.

Semi-continuous branching

On the UP branch the lower bound is set to 1.0. On the DOWN branch the variable is fixed to zero.

SOS branching

The reference row function is evaluated which determines a point between two set members. On the UP branch lower-numbered set members are fixed to zero and on the DOWN branch higher-numbered set members are fixed to zero. For SOS2 the member nearest to the evaluation is allowed to be non-zero on both branches.

If no reference row is given in the data then a series 1,2,...k is assumed for the reference row coefficients.

Back to Chapter contents

6.5 Controlling the Tree Development

6.5.1 Definition of Tree Search Heuristics

In the Branch and Bound algorithm described above there are two major steps where a choice has to be made

Step 1 in which a global entity is selected for partitioning the solution space - this is termed **variable choice**

Step 2 in which an unsolved node is selected for solution - this is termed **node choice**

We may consider a solution 'strategy' to be the combination of criteria whereby these two choices are made. Experiments on real life problems using different strategies show that each individual strategy behaves differently on different problems. FortMP-MIP therefore provides a number of alternative criteria for variable and node choice. Moreover, the node choice criterion can change after the first integer solution has been found.

In principle, the node choice may be considered separately from the variable choice criterion. But not all such choices can be carried out independently of the information provided by the variable choice procedure. However, the node choice criterion is usually changed after the first integer feasible solution is achieved in order to provide a more flexible tree search. In FortMP-MIP the following parameters are used to set the variable and node choice criteria. These parameters are as follows:

MIP VARCHOICE = n
MIP FNODCHOICE = n
MIP SNODCHOICE = n

where n is an integer specifying alternative choices

Back to Chapter contents

6.5.2 Provision of choice criteria by the user

A complete tree search strategy is defined by a unique combination of variable and node choice criteria. Variable choice criterion is selected by the following SPECS command:

```
MIP VARCHOICE = n              *1-5, default n = 1
```

In order to specify the options available we define the following terms:

Fractionality Measure of non-discreteness associated with an entity.

Cost Evaluation: Absolute value of variable cost multiplied by fractionality.

where the 'measure of non-discreteness' is a formula for each type of global entity which is zero when the entity is integer feasible and increases with the degree of violation.

Now the 'VARCHOICE' command has the following options:

n = 1 Select the entity with minimum fractionality. This is the default.
n = 2 Select the entity with maximum fractionality
n = 3 Not currently available (reserved for future enhancement)
n = 4 Select the entity with maximum cost evaluation
n = 5 Select the entity with minimum cost evaluation.

Node choice criterion is selected by the following two commands:

```
MIP FNODCHOICE = n              * n=1-7,  default n = 1
MIP SNODCHOICE = n              * n=1-7,  default n = 7
```

where 'FNODCHOICE' governs the initial search up to the first integer solution and 'SNODCHOICE' governs the search thereafter.

Both commands have the same options and in order to specify them we define the following terms relating to each node:

Number Non-integer Number of violated global entities in the solution.

Fractional Sum Sum of fractionalities associated with all the violated entities in the solution

Deterioration Difference between the objective values of the node solution and the solution at the root node ($z^l_{LPR} \sim z^l_P$).

Projection The predicted integer solution (based on a heuristic) when the current branch is pursued to its end.

The projection heuristic assumes that fractional sum will decrease uniformly with further deterioration in the objective as the branching proceeds.

Now the options for 'FNODCHOICE' and 'SNODCHOICE' are:

n = 1 Last in first out. This is the default for 'FNODCHOICE'
n = 2 First in first out
n = 3 Choose a node with minimum deterioration
n = 4 Choose a node with maximum deterioration

n = 5 Choose a node with minimum number non-integer
n = 6 Choose a node with minimum fractionality
n = 7 'Best projection criterion': that is choose node on a branch predicted to reach the best integer solution. This is the default for 'SNODCHOICE'

The user is referred to references [5], [15], [16], [22], [23] and [24] given in Chapter 1, section 1.8 for definitions and theoretical discussion of choice strategies.

Back to Chapter contents

6.5.3 UP-direction priority option

In a large number of problems there is an advantage to selecting the UP direction branch for sub-problem solution in preference to the opposite DOWN branch. This may have the effect of forcing other variables to zero and thus reaching an integer solution quicker than if the choice is open.

Problems that benefit in this way arise in allocation and scheduling models, and elsewhere. They may, for example, have a large number of constraints similar to the convexity row of an SOS, that is:

$$\sum_{j \in B_i} x_j \leq 1$$

where B_i is a subset of the binary variables appearing in row i.

To enforce this priority the following SPECS command is used

```
MIP PRIORITY UP ON              * default OFF
```

(OFF may also be given). With PRIORITY UP activated (ON) the variable choice options become modified so as to consider fractional values only in the UP sense. This prevents selection of a binary close to zero in preference to one that is close to one which would have a damaging effect.

UP priority is mainly intended for use with binary variables and has little relevance to general integer variables or special ordered sets.

Back to Chapter contents

6.6 Detailed User-control of the Tree Search

6.6.1 User control of variable choice

Very often a modeller will have information about the variables that can be used in variable selection - step 1 of the algorithm described above in 6.4.2. For example some binary variables may represent strategic decisions and others represent minor decisions - meaningful only when the strategic decisions have been made. Although the algorithm is probably unable to distinguish the two, if it can be made to branch on strategic decisions first the search time can be greatly reduced.

In general the modeller is not expected to specify a total ordering of the variables and so to force a specific choice at every node. All the same the modeller may be able to classify variables according to their importance and give every class of variable a corresponding priority.

These user-priorities are used in step 1 as the primary choice criterion. If multiple choices are available at the highest priority then the standard choice criterion, as set up by 'MIP VARCHOICE' specs-command, is used to discriminate between them.

User priorities are input via a MIP 'AGENDA' file for which the layout is given below in section 6.6.3.

Back to Chapter contents

6.6.2 Control of node choice - Fixing an Integer Solution

The choice of next node (sub-problem) for solution is step 3 of the branch and bound algorithm. The user can direct this choice towards an integer solution which is provided in the data.

An advance integer solution can be known in various ways, for instance:

- from a previous execution of FortMP with the same, or very similar, problem data,
- from a known situation in the real world,
- from executing an independent, heuristic program that does not necessarily determine the integer optimum.

If the solution is good, that is close to the integer optimum, it is desirable for the system to establish it as quickly as possible so as to provide a cutoff point which limits the tree search.

There are two ways to do this:

FIXTRY

Under the 'FIXTRY' option the system sets up the given integer solution immediately after entry as a special sub-problem entirely separate from the tree. The sub-problem is solved and if it does indeed prove to be feasible (and satisfy all the discrete restrictions) then its solution value provides the initial cutoff.

`FIXMIX`

Under the 'FIXMIX' option the system gives absolute priority at step 3 (node selection) to any node in the tree for which the established constraints at that stage of the branching do not conflict with the given integer solution. When every discrete entity is given a value (set of values for an SOS) there can be only one such node, unsolved, in the tree and the tree development must therefore follow a direct path to the solution if it is in fact feasible.

Once the 'FIXMIX' solution has been established (or proved infeasible) then this priority ceases to have effect and the normal criteria for node choice are followed (with priority 'UP' if specified).

FIXMIX is preferable to FIXTRY if the given integer solution is known to be a good one - that is both feasible and either optimal or close to optimal. In this case it creates an advanced tree leading to reduced search time.

Data for FIXTRY and FIXMIX is provided in a MIP 'AGENDA' file described below.

Back to Chapter contents

6.6.3 AGENDA data for variable priorities or 'FIX' solutions

A MIP 'AGENDA' file provides the data for variable priorities or a 'FIX' solution. The outline of this file is as follows:

```
AGENDA
PRIORITY
. . . variable priorities . . .
FIXMIX or FIXTRY
. . . 'FIX' solution . . .
ENDATA
```

where the header records begin in position 1 (as with other headers).

The file must have the following name:

```
modname.agn
```

where 'modname' is the model name in use for the run.

The two data sections are each optional and may appear in either order. Once the MIP AGENDA ON command has activated the input of the file it is the header record that activates the corresponding procedure or priority select-criterion as described in the two previous sections.

The data layout in either section is the same, namely:

Field 1 (2-3)	Field 2 (5-12)	Field 4 (15-22)	Field 5 (25-36)
	Column name	Column name	Value

with similar fields to MPS data input.

Fields 2 and 3 specify a start-column and an end-column in the data. The value in Field 4 is assigned to every variable in this range inclusive of the start and end column. All values are initially assigned to be zero and these defaults are over-written when a record provides data. The data of one record may also be over-written by a later record. These arrangements permit the modeller to specify a default (or use default zero) for a wide range and simply enter exceptions. For convenience the column range may include ordinary continuous variables which are just ignored.

The following is an example of agenda data for the model example of section 6.3:

```
AGENDA
FIXMIX
    X2        X3        5.0
    X4        X4        1.0
    X15       X15       1.0
    X22       X23       0.5
    X33       X34       0.5
PRIORITY
    X4        X4        3
    X2        X2        2
    X3        X3        1
ENDATA
```

In this example the binary variable has first priority, the integer variable has second and the semi-continuous variable has third. The special ordered sets have zero priority and will not be branched if

a choice of another kind is available.

Also in this solution the FIX solution corresponds to the integer optimum. Precise values for the adjacent pair to be non-zero in each SOS2 is not significant and need not be given.

Back to Chapter contents

6.6.4 SPECS commands for AGENDA input

The agenda file will be input when the following SPECS command is given:

```
MIP AGENDA INPUT ON            * default OFF
```

(OFF may also be used).

Back to Chapter contents

6.6.5 SPECS commands for AGENDA output

An agenda file for the best integer solution found during a run may be created with the SPECS command:

```
MIP AGENDA OUTPUT ON           * default OFF
```

(OFF may also be used). Each time an integer solution is found the system creates an AGENDA file with no PRIORITY section and with a FIXMIX section containing the solution. The previous AGENDA file (if any) is over-written.

Another option is provided by the following command:

```
MIP AGENDA OUTPUT ALL
```

With this option multiple integer solutions are all recorded in the output AGENDA file, later solutions in successive sections separated by FIXMIX header lines. This is not a valid file for input in a later run using MIP AGENDA ON. In order to use it for this purpose the modeller needs to select the section of interest (usually the last one which is the best) and delete the remaining sections.

AGENDA OUTPUT commands provide a convenient way to record integer solutions when the more detailed information of the standard output is not needed.

Back to Chapter contents

6.6.6 Un-named Agenda Files

Agenda files can be used with the variables identified by index rather than by name. This can be necessary, for example, when tabular form data is supplied or when using the external data interface.

The data layout is very similar to named agenda files, it has the same header lines and a difference only in the layout of data lines which is as follows:

Field 1 (1-10)	Field 2 (11-20)	Field 3 (21-30)
Column index	Column index	Integer Value

where fields 1 and 2 specify start and end column and field 3 specifies the priority class or 'FIX' value as before. The numerical data must be entered with right justification, this is because any trailing blanks in a field are treated as zeros.

The associated SPECS commands are:

```
MIP LIST INPUT ON              * default OFF
MIP LIST OUTPUT ON             * default OFF
MIP LIST OUTPUT ALL
```

('OFF' may also be used). These commands have the same meaning as the corresponding 'AGENDA' commands.

A 'LIST' agenda file must have the following name:

```
modname.agl
```

where 'modname' is the model name in use for the run.

6.7 Advanced Algorithms for MIP

6.7.1 Recent advances

Recently there have been considerable improvements to MIP solution procedures through the use of features such as Pre-processing and Cut generation.

The aim of these techniques is to determine additional constraints that must be satisfied by any integer solution but are not satisfied by the optimum solution to the relaxed LP or by the optimum solution to an intermediate sub-problem of the Branch and Bound tree having partial LP relaxation.

If such a constraint is added to the relaxed LP at the root node the effect may well be to degrade its optimum objective value and thus reduce the gap between that and the integer optimum with evident beneficial effect on the search procedure. The same is true for any intermediate node in the tree, where the optimum solution is LP feasible but not fully integer feasible.

The pre-processing procedures have another aim, namely, to release constraints whenever they are found to be redundant. As a simple example consider the common 'Clique'-type constraint which takes the form:

$$\sum_{j \in B_i} x_j \leq 1$$

where B_i is a subset of the binary variables applying in the i'th row. If we discover one of its members to have a lower bound greater than zero then that member must be one and the remaining members must be zero. Once these fixes have been applied the constraint itself is redundant and may be released, which improves solution time by reducing both the time and the number of Sparse Simplex steps needed to solve sub-problems.

Back to Chapter contents

6.7.2 Pre-processing - Variable Fixing and Constraint Relaxing

The most direct technique to apply is that of PRESOLVE where the implications of primal constraints:

$$\sum a_{ij} x_j \; :: \; b_i$$

are studied, whatever the variable types involved. This is a general and comprehensive technique for fixing variables and relaxing constraints.

In addition if a tightened bound can be discovered for an integer or binary variable, giving it a fractional bound in one sense or the other, then the fraction can be truncated which further tightens the bound and gives an improved effect.

In FortMP this technique is activated with the following SPECS command:

```
MIP PREPROCESS ON              * default OFF
```

(MIP PREPROCESS OFF may also be used). This causes MIP pre-processing to be carried out at the outset (before the relaxed LP is solved) and again before solving each sub-problem in Branch and Bound.

Additional memory areas are required for this algorithm so that the fixing of variables and relaxing of constraints can be replicated at every point on the tree allowing full freedom to branch in any direction without requiring the MIP pre-processor execution to be repeated un-necessarily.

These areas are called FIX tables and are assigned together with the node tables in the ratio 10 to 1 (10 fixes to every node as an average). For some models this ratio is not enough and it can be altered with the SPECS command:

```
MIP FIX QUOTA = n              * default = 10
```

While 10 is usually suficient a ratio of 20 to 1 has been found necessary for some problems. Extremely high values should be avoided so as to allow space for a maximum of nodes.

The following commands correspond to the same features in the Presolve algorithm:

```
MIP PREPROCESS LEVEL = n       * 1-3, default = 3
MIP PREPROCESS LOG LEVEL = n       * 0-4, default = 1
```

Three levels are available in the pre-processor and the level of information logged can be controlled as before. Note that log level refers to the root node only, the level for logging during branch and bound is reduced by one.

In addition the MIP pre-processor can be used at the root node, and then be switched off during branch and bound. The command for this is:

```
MIP PREPROCESS ROOT ONLY
```

With this command there is no need for any FIX quota.

Back to Chapter contents

6.7.3 Cut Generation

Cuts, like fixes, are additional constraints that may be applied to the problem. Unlike fixes, however, they cannot be applied to the problem by a simple change to variable types and the

bound-set.

A cut is a constraint, an inequality (or an equality) just like any other constraint in the problem. It can arise in many different ways:

- as Maximal Clique or Minimal Cover obtained by processing knapsack constraints in the model (constraints that have non-zero coefficients only on binary variables),
- Gomory cuts,
- Optimality cuts and Infeasibility cuts,

and many others.

Cuts are never added directly to the problem - rather they are stored separately in an area called the Cut Pool which holds cut coefficients and cut right-hand-sides. From the cut pool a cut may be applied to the matrix whenever it is advantageous to do so. It is not advantageous to apply a cut when the optimal solution to the current sub-problem (including the relaxed LP as a sub-problem in this context) actually satisfies that cut-constraint anyway. Consequently after every optimal solution has been found a search is made for 'Strong' cuts - that is cuts which are violated by the current solution. These cuts are added to the main problem constraint-set and the problem is then re-solved.

The following SPECS command must be given in order to make cut-pool space available and activate the FortMP cut-generator procedures:

```
GENERATE CUTS ON                * default OFF
```

(GENERATE CUTS OFF may also be specified).

The following SPECS commands are similar to the 'MIP FIX QUOTA' command in that they govern the amount of space that the system reserves for the cut pool and for applying cuts to the problem:

```
CUT QUOTA = n                   * default = 10
```

This governs the amount of cut pool space provided in relation to the model size. By default the system assigns space for 10 cut-constraints per constraint-row in the original model and for 10 cut non-zero coefficients per non-zero coefficient in the original matrix.

```
ACT QUOTA = n                   * default = 5
```

This governs the extra space allowed in the problem storage for applying cuts. It applies in the same way as before:- The system (by default) assigns 5 extra constraint-rows per original constraint-row, 5 extra spare locations per original non-zero for expanding the matrix and also the same amount spread over all columns as extra row-space in each column so that rows can be added with a

minimum of delay. This extra spare space in the matrix is essential to ensure that the process is efficient.

ACT quota also governs the amount of space needed per node to record applied cuts - enabling them to be replicated and not require re-calculation. Clearly the user may need to raise the quotas for his problem but should avoid raising them too high in order to leave enough room for Branch and Bound itself and for the other algorithms.

Back to Chapter contents

6.7.4 Fixing Variables by Dual Solution Analysis

When selecting a discrete entity for branching only basic variables are considered (SOSs apart). Non-basic variables, valued at zero or at bound, should not be selected because they may never need it. In general only a small proportion of the discrete entities need to be branched on in order to reach an integer solution.

However the dual value associated with a non-basic, discrete variable (binary, integer or semi-continuous at zero) can be used to find a minimum change that would apply to the objective if the variable shifted to another discrete value. If this minimum change increases the current optimum above the cutoff level then clearly it will be unnecessary. The variable may be fixed at its current value.

To invoke the procedure the following SPECS command is used:

```
MIP ANALYSE DUAL ON            * default OFF
```

(OFF may also be given). The user should employ this option with caution as not all problems benefit and the calculations require extra time.

6.8 Miscellaneous MIP controls

6.8.1 Automatic rounding heuristics

At the relaxed LP optimum solution it may be expected that some of the discrete variables are already feasible or very near feasible in value. Automatic Rounding allows user to fix these variables immediately at the nearest discrete value and consider the restricted branch and bound tree that results. Given the likelihood of finding an integer solution in this restricted sub-tree the time required to find a suitable solution to the problem may be very greatly reduced.

MIP can operate in two phases, a search of the restricted tree (auto-rounded) followed by a search of the full tree and each tree may be controlled by its own independent limits.

In order to activate auto-rounding the following command is given:

```
MIP AUTO ROUNDING ON          * Default OFF
```

Control over what variables will be rounded is provided by the following command:

```
MIP ROUNDING FRACTION = v     * Default v = 0.001
```

where the default is in fact set from the integer tolerance. Any value up to 0.5 can be specified - the auto-round procedure will fix any binary or integer variable that is closer than this amount to the nearest integer value. Fixing covers both basic and non-basic variables, so implicitly all binary and integer variables that are non-basic in the relaxed LP optimum will be fixed.

Finally the algorithm used to solve the auto-rounded sub-problem, which is used as the root node for the restricted tree, may be governed with the following commands:

```
MIP AROUND SOLVER SSX
MIP AROUND SOLVER IPM         * Default SSX
```

where 'SSX' specifies the Dual algorithm (reverting to Primal if it fails).

Back to Chapter contents

6.8.2 Bound, Cutoff and Tolerance control

The following SPECS commands are useful to avoid searching unwanted parts of the tree or searching in unnecessary detail.

```
INTEGER TOLERANCE = v         * default v=0.001
```

This tolerance supplies the largest deviation from an integer value for which a discrete entity is deemed to be integer feasible. It should represent the maximum fraction considered by the user to

be tolerable in a solution. Very often it is found that most or all fractions drop to zero anyway below the default level.

Two concepts, the *BOUND* and the *CUTOFF TOLERANCE*, permit user to reduce unnecessary searching of branches during MIP. They operate by controlling the *TREE CUTOFF* or maximum *GAP* beyond the RLP optimum that is the region where solutions to MIP sub-problems may be found. Once the solution value of a branch deteriorates beyond the limit set by this gap then that branch is discontinued. Initially the gap is set by the starting *BOUND*. This may be controlled by the user with the following SPECS commands:

```
MIP BOUND = v                    * default: High value
MIP BOUND RELATIVE = v           * default: High value
```

'MIP BOUND = v' may also be written as *'MIP BOUND ABSOLUTE = v'*, and it specifies an actual initial gap. *'MIP BOUND RELATIVE = v'* specifies gap size relative to the RLP optimum - that is, the gap size is the RLP optimum value multiplied by the factor supplied. If the user supplies both commands then the gap size used will be the smaller of the two.

CUTOFF TOLERANCE becomes relevant once an integer solution has been found. From there on a maximum gap is determined by the Best Integer Solution found so far (the *BIS*), and this gap is simply the difference between the BIS and the RLP optimum. User can modify the gap by applying a 'tolerance', which reduces the gap, thereby reducing search time. This may prevent the genuine Optimum Integer Solution from being found, but ensures that the eventual BIS differs from the optimum by no more than the tolerance.

Cutoff Tolerance can be controlled with the following SPECS commands:

```
MIP CUTOFF TOLERANCE = v         * default v=1.0e-12
MIP CUTOFF RELATIVE = v          * default v=0.0
MIP CUTOFF RELISOL = v           * default v=0.0
```

Here *'CUTOFF TOLERANCE'* specifies the actual amount of gap-reduction. *'CUTOFF RELATIVE'* specifies the reduction as a fraction of the gap itself, so that a large reduction will be applied for poor solutions and the reduction will get less progressively as the BIS improves. *'CUTOFF RELISOL'* specifies the reduction as a fraction of the BIS value.

Note that when the reduced gap becomes zero or negative the tree search is immediately concluded.

Back to Chapter contents

6.8.3 Placing Limits on the Tree Search

The SPECS commands given below limit the total extent of the tree search. When the limit is exceeded there is a SAVE made of the tree (unless SAVEs are cancelled) and a continued search can be restarted in the following run.

```
MAXIMUM MIP NODES = n          * default = 50,000
```

Specifies the maximum number of nodes, i.e. sub-problem solutions, that can be attempted.

```
MAXIMUM MIP TIME = v           * default = 50000.0
```

Gives the maximum number of seconds for which the MIP algorithm may run.

```
MAXIMUM MIP INTEGER SOLUTIONS = n
MAXIMUM MIP INTSOL = n         * default = 300
```

Either of the above commands sets a limit to the number of integer feasible solutions, and the run halts when this number is reached.

There may be separate limits given for the restricted AUTO ROUND tree developed first in the event that user specifies auto-rounding. During auto-rounding no tree-saves are made. The commands are:

```
MAXIMUM AROUND NODES = n       * default = 5000
```

Specifies the maximum number of nodes, i.e. sub-problem solutions, that can be attempted. Auto-round nodes do not count towards the MIP limit.

```
MAXIMUM AROUND TIME = v        * default = 5000.0
```

Gives the maximum number of seconds for which the auto-round tree-search may run. Auto-round search time does not count towards the MIP limit.

```
MAXIMUM AROUND INTEGER SOLUTIONS = n
MAXIMUM AROUND INTSOL = n      * default = 1
```

Either of the above commands sets a limit to the number of integer feasible solutions, and the auto-rounded tree-search halts when this number is reached. The integer solutions found by auto-rounding do count towards the maximum for MIP as a whole (unlike the other auto-round limits).

It is also possible to place a limit on the total storage used with the following command:

```
MAXIMUM MIP SPACE = n          * default = 10,000 nodes
```

but this does not necessarily limit the search because the space for deleted nodes is recovered and used over again any number of times. However the system may even so run out of node space.

The system in any case reduces this maximum if there is not enough room by calculating how many nodes the store can hold. There must be room enough for at least one tenth the specified maximum and otherwise MIP halts at the outset. Default minimum storage is therefore 1000 nodes when the command is not given.

Back to Chapter contents

6.8.4 Saving the tree and restarting MIP

FortMP-MIP carries out an automatic save of the search tree at a frequency given in terms of the number of nodes built. The SPECS command is:

```
MIP SAVE FREQUENCY = n          * Default = 500
```

By setting n to zero the facility is switched off.

If for any reason MIP fails to complete the search in one run then it is possible to restart the search at or very close to the stopping point. To restart the search the following SPECS command is used:

```
MIP RESTART ON
```

(OFF may also be given). For a successful restart the files built in the previous run must be present on the working directory, in particular the following file:

modname.msv

where 'modname' is the model name.

It is of course obvious that in order to continue running in a restart by this means the original limit that halted the previous run must be relaxed. A new time limit is set up anyway but the Maximum NODES, INTEGER SOLUTIONS or SPACE may need to be increased (see the previous section).

In the case of node storage limit exceeded the user may be able to increase the maximum but this will not help if the maximum is already restricted by the store size. The user can attempt to run with reduced quotas (FIX, CUT and ACT) but if this also fails then a larger memory is needed.

Back to Chapter contents

6.8.5 Bypassing Mixed Integer

In many cases users prefer to see the solution to the linear programming relaxation of a MIP model before the beginning of the branch and bound process. In this case the user can bypass the call to MIP module with the following SPECS command

```
MIP OFF                         * Default ON
```

(ON can also be specified). Alternatively the user may limit the MIP run, for example with the command `'MAXIMUM MIP NODES = 0'`.

Note that it is possible to restart MIP after restarting the LPR stage with the saved optimal basis. Commands to use are:

```
INPUT RESTART
SIMPLEX START RESTART
```

with the same *SCALE* and *MIP PREPROCESS* option as before but without using *PRESOLVE* or *IPM*.

Back to Chapter contents

6.8.6 Making use of the PRIMAL algorithm

DUAL is employed by default for solving sub-problems. The following SPECS command invokes primal as an alternative:

```
MIP DUAL OFF                    * default ON
```

(MIP DUAL ON can also be given). DUAL is numerically less reliable than PRIMAL in certain cases which may make this option desirable. However failure of DUAL due to numerical difficulty causes the system to revert to PRIMAL in any case.

Back to Chapter contents

6.9 Logged Output and Screen Display

As in other FortMP procedures the output written to the log and the screen display can be varied to suit the degree of detail required. The log output is controlled with the following SPECS command:

```
MIP LOG LEVEL = n              * n=0-4, default=1
```

In default, log level 1, the screen display and log file record outstanding items, including the occurrence of each integer solution.

With level 2 or higher a log is shown of every node. Nodes are numbered following selection for sub-problem solution and so will appear in the log numbered in order 1,2,3,.... Users may find this log much too long and it can be reduced to appear at node intervals given by the following SPECS command:

```
NODE LOG FREQUENCY = n         * default n=1
```

The various headings of the level 2 log are shown in the following table:

Log Heading	Screen Text	Description
Node	N=	Node number for identification
Parnt	P=	Parent node number
Obj.ve	LP=	Objective value of sub-problem solution
Fractn	FR=	'Fractionality' - or measure of non-discreteness. Here is also given a text indicating whether node is infeasible or cutoff
Bvar		'Branch Variable':- index of the selected discrete entity (or set member)
Typ		Type of discrete entity partitioned:- BV, UI, SC, S1 or S2
Dir		Branch direction:- UP or DWN
NNI	NI=	Number non-integer:- remaining number of violated discrete entities
NCH	CH=	Node choice:- Current number of unsolved sub-problems in the tree
Dpth		Depth of the node in the tree

IPbest	BI=	Best integer solution - the current cutoff (high value if no integer solution found so far)
SET#	SET#	Index of selected SOS (when 'Typ' is S1 or S2)
FROM/ TO	FROM/ TO	Index range of selected SOS members allowed to be non-zero (all other members of the set fixed to zero at this node)

Other information is given with self-explanatory text. The following is the output to log file at level 2 for the example problem given earlier:

```
---------------------------------------
           IN THIS SEARCH
FIRST NODE CHOICE STRATEGY =  1
SECOND NODE CHOICE STRATEGY =  7
VARIABLE CHOICE STRATEGY =  1
---------------------------------------
 optimum lp solution is  0.9344E+02
Dual infeasibility:       3  -51.109671      Iter#     13
  Node  Parnt    Obj.ve    Fractn       Bvar Typ Dir  NNI  NCH Dpth  Ipbest
     2      1  223.417    No feas sol      4 BV  DWN    2    1    1  0.1000E+11
     3      1  93.8205     0.2720          4 BV  UP     2    0    1  0.1000E+11
     4      3  94.3363     0.1775          2 UI  DWN    1    1    2  0.1000E+11
     5      4  107.112     0.5070E-01     16 S2  UP     1    2    3  0.1000E+11
                SET#    3,  FROM/TO   17/   19,  Iterno=    21,  Prty=   0
     6      5  138.369    No feas sol     17 S2  UP     2    3    4  0.1000E+11
                SET#    3,  FROM/TO   18/   19,  Iterno=    21,  Prty=   0
     7      5  108.000     0.3970         17 S2  DWN    2    2    4  0.1000E+11
                SET#    3,  FROM/TO   17/   18,  Iterno=    22,  Prty=   0
 $$$$ INTEGER SOLUTION:  0.108000E+03,  FOUND AT NODE#      7
          Time since B&B start:    0.05 secs,  Iter# =     22
  Node  Parnt    Obj.ve    Fractn       Bvar Typ Dir  NNI  NCH Dpth  Ipbest
     8      4  95.1667     0.5070E-01     16 S2  DWN    1    1    3   108.0
                SET#    3,  FROM/TO   16/   17,  Iterno=    23,  Prty=   0
 $$$$ INTEGER SOLUTION:  0.951667E+02,  FOUND AT NODE#      8
          Time since B&B start:    0.05 secs,  Iter# =     23
  Node  Parnt    Obj.ve    Fractn       Bvar Typ Dir  NNI  NCH Dpth  Ipbest
     9      3  94.5185     0.1775          2 UI  UP     1    0    2   95.17
    10      9  96.1667    Dual cutoff     14 S2  UP     1    1    3   95.17
                SET#    2,  FROM/TO   14/   15,  Iterno=    24,  Prty=   0
    11      9  95.7051    Dual cutoff     14 S2  DWN    1    0    3   95.17
                SET#    2,  FROM/TO   13/   14,  Iterno=    24,  Prty=   0
 ********** Search completed **************
 The IP optimum is    95.166667
 Total nodes: opened =     11  processed =     10
 Final iteration count =      24
        $$$$$$$$$ Saving the tree  $$$$$$$$$$
 No of nodes:    11 built    10 processed
 Storage used:- Nodes     8, Bases     3, Fixes:     0
```

```
  Integer solutions    2  best =   95.1667
 TIME TAKEN FOR INTEGER      =    0.16 SECS,  TOTAL SO FAR  =    0.71 SECS
```

Back to Chapter contents

6.10 MIP Constraint Classification

6.10.1 Introduction and SPECS command

A useful feature of FortMP leading to better understanding of model types and more effective decisions on how best to tackle them is constraint classification. Classification is used in the pre-processing and cut generation procedures of the system.

The user may ask to have this classification recorded in the output log with the following SPECS command:

```
MIP CLASSIFY ROWS ON          * default OFF
```

(OFF can also be given).

A description follows of all the various constraint classes distinguished by this command. In the definitions given the notation 'N_3' is used to represent a sub-set of the binary indices rather than the full set defined by 'N_3' in section 6.2.5. Similarly for other index sets.

Back to Chapter contents

6.10.2 Knapsack Constraint Classification

Classification literature refers to a Knapsack constraint as an inequality:

$$\sum_{j \in N_3} a_j x_j \leq b$$

where all the relevant variables x_j are binary and where the RHS b and all the coefficients a_j are positive and integer.

In fact we can consider any inequality constraint that involves only binary variables to be a knapsack constraint by a combination of scaling and complementing those variables that have negative coefficients. Let the constraint be:

$$\sum_{j \in N_3^+} a_j x_j + \sum_{j \in N_3^-} a_k x_k \leq b$$

wnere N^+_3 and N^-_3 are the sets for which a_j is positive and negative respectively. The complement variables (also binary) are defined by

$$\overline{x_k} = 1 - x_k$$

and when we substitute for terms with negative coefficients the constraint becomes:

$$\sum_{j \in N_3^+} a_j x_j + \sum_{k \in N_3^-} (-a_k) \overline{x_k} \leq b - \sum_{k \in N_3^-} a_k$$

with all the coefficients on the left hand side positive. Since all coefficients are rational this may now be scaled to make them all integers and the RHS-value, truncated if necessary, is integer also. Hence we have a knapsack constraint.

Bearing this in mind, we give a classification for constraint-types on the basis that any such necessary transformations are assumed to have been carried out, and the resulting positive/negative sub-division of the coefficients into N^+_3 and N^-_3 refers to their values after complements have been substituted in a way to obtain the prescribed form.

Knapsack constraints are classified into the following:

Knapsack (KNA)

This is the generic type defined above

Invariant Knapsack (INK)

Knapsack constraint with all coefficients 1.0

$$\sum_{j \in N_3} x_j \leq b$$

This constraint-type is also known as a *cover*.

Clique (CLQ)

Invariant knapsack with RHS equal to one.

$$\sum_{j \in N_3} x_j \leq 1$$

Set Covering (SCV)

Invariant Knapsack with RHS equal to one less than the number of LHS terms.

$$\sum_{j \in N_3} x_j \leq |N_3| - 1$$

When each x_j is complemented this becomes similar to the clique but with opposite sense to the inequality:

$$\sum_{j \in N_3} \overline{x_j} \geq 1$$

Bin Packing (BPK)

Knapsack constraint with RHS zero and exactly one negative coefficient.

$$\sum_{j \in N_3^+} a_j x_j - a_k x_k \leq 0, \quad k \in N_3^-(a_k > 0)$$

Plant Location (PLK)

Bin Packing constraint with all the positive coefficients equal to 1.0:

$$\sum_{j \in N_3^+} x_j - a_k x_k \leq 0, \quad k \in N_3^-(a_k > 0)$$

Reverse Plant Location (RPL)

Knapsack constraint with RHS zero, exactly one positive coefficient and all negative coefficients equal to -1.0:

$$a_k x_k - \sum_{j \in N_3^-} x_j \leq 0, \quad k \in N_3^+(a_k > 0)$$

Back to Chapter contents

6.10.3 Mixed Less or Equals Constraint Classification

Variable Upper Bound (VUB)* and *Variable Lower Bound (VLB)

Weak knapsack constraint with one binary term, one non-binary term, and zero RHS:

$$a_j x_j + a_k x_k \leq b, \quad j \in N_3, k \notin N_3$$

If $a_k > 0$ then we have VLB, else if $a_k < 0$ then we have VUB.

A combination of VUB and VLB for the same pairing of a decision variable with a continuous bounded variable is modelled in a simpler way by one semi-continuous variable.

Weak Knapsack (WKN)

If we have a mixed constraint:

$$\sum_{j \in N_3} a_j x_j + \sum_{k \notin N_3} a_k x_k \leq b$$

Then appropriate bounds on the non-binary variables may lead to a knapsack. If (l_k,u_k) is the bound-range of x_k and if:

l_k is finite for each $a_k > 0$
u_k is finite for each $a_k < 0$

Now let the finite bound so indicated be $[l|u]_k$. We can derive the following constraint:

$$\sum_{j \in N_3} a_j x_j \leq b - \sum_{k \notin N_3} a_k \left[l \middle| u \right]_k$$

which is a meaningful knapsack constraint provided that the RHS is not too large.

Weak knapsack constraints are further sub-classified into the following constraint types:

Weak Invariant Knapsack (WIK)

Weak Knapsack constraint with all binary terms having coefficient 1.0:

$$\sum_{j \in N_3^+} x_j + \sum_{k \notin N_3} a_k x_k \leq b$$

with the same bound conditions as before and assuming that the derived knapsack is meaningful.

Back to Chapter contents

6.10.4 Equality constraints

Although an equation might be considered as two inequalities, to be separately classified as in 6.10.2 or 6.10.3 this is not generally useful and equations are considered separately.

Various types of equation are highlighted as follows:

Diophantine Equations (DPQ)

An equation involving only binary and integer variables:

$$\sum_{j \in N_3} a_j x_j + \sum_{k \in N_4} a_k x_k = b$$

DPQ as originally defined should have all a_j , a_k and the RHS (b) integer-valued. This requirement is omitted on the same argument as for knapsack constraints - the constraint can be scaled up to

achieve this condition.

P-fold Alternative (PFLD)

Sum of binary variables equal to a constant p:

$$\sum_{j \in N_3^+} x_j = p$$

Exclusive OR (XOR)

P-fold alternative with $p = 1$:

$$\sum_{j \in N_3^+} x_j = 1$$

Note here that the convexity row of an SOS1 (if it exists) is an XOR constraint.

Discrete Goal-Oriented Equations (DGOQ)

$$\sum_{j \in N_3} a_j x_j + x_h - x_k = b$$

Where x_h and x_k are 'singleton' columns, not of binary type and not appearing in any other constraint. This consists of an equation involving binary variables augmented by logical additions that can specify a penalty cost attached to any failure to achieve a solution such that:

$$\sum_{j \in N_3} a_j x_j = b$$

It is not necessary to have both the positive and the negative additional terms. The following types of constraint are also classified as DGOQ:

$$\sum_{j \in N_3} a_j x_j + x_h = b$$

$$\sum_{j \in N_3} a_j x_j - x_k = b$$

where x_h and x_k are 'singleton' columns as before.

Back to Chapter contents

6.10.5 Full Classification Hierarchy

Certain general classes not yet mentioned may be named here:

LE	The class of all LE-type constraints (including GE-type which are converted on input to LE-type)
EQ	The class of all EQ-type constraints.
RNG	All constraints with RHS range
OBJ	Objectives and free rows
MLE	Mixed LE constraints, part binary and part non-binary.
OLE	Other LE constraints having no binary component
BDPQ	Diophantine equations involving only binary variables
IDPQ	Diophantine equations involving only Integer variables
MDPQ	Mixed Diophantine equations, part binary and part integer
NDPQ	Non-diophantine equations, i.e. incorporating one or more continuous variables.

We include also the following 'Other' types:

SUB	Simple Upper Bound:- where there is only one non-zero term giving a fixed upper bound to that variable.
SLB	Simple Lower Bound:- where there is only one non-zero term giving a fixed lower bound to that variable.

The following diagram illustrates the hierarchical relationship of all these classes and (on the right) lists the total sub-classification derived by the system. Note that the BGOQ class appears as a sub-set of both NDPQ and MDPQ classes. This is the only exception to the ordinary, hierarchical structuring.

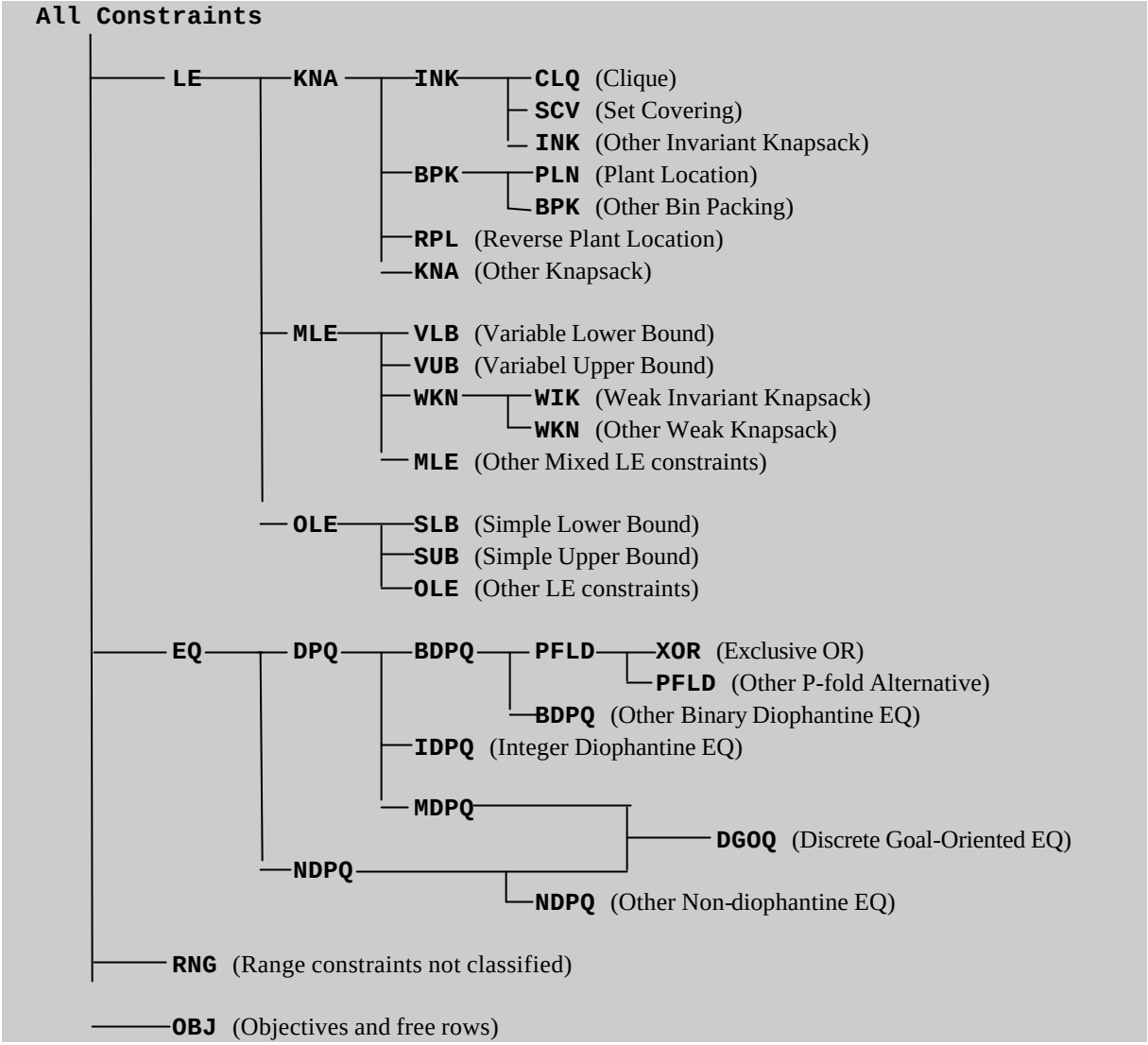


Diagram 1. Row Classification Hierarchy.

Back to Chapter contents

6.11 Summary of MIP SPECS Commands

The following SPECS-commands have been introduced in this chapter.

```
INPUT INTORG UPPER BOUND = n       * default n is
infinite
INPUT INTORG BOUND = n         * default n is infinite
```

Assigns a default upper bound to integer variables outlined by INTORG/INTEND markers. If n = 1 the variables are binary. See section 6.3.3

```
MIP VARCHOICE = n              * Default n = 1
```

Controls the choice of branching variable (or SOS) at each node. Values are:

n = 1 Select closest to integer value
n = 2 Select farthest from integer value
n = 3 Not in use (invalid)
n = 4 Select variable with highest cost
n = 5 Select on basis of absolute cost

See section 6.5.2.

```
MIP FNODECHOICE = n            * Default n = 1
MIP SNODECHOICE = n            * Default n = 7
```

Controls choice of node to be developed during the each phase of MIP. 'F'-choice is used prior to the first integer solution 'S'-choice thereafter. Possible choices are:

n = 1 Last in first out
n = 2 First in first out
n = 3 Choose minimum deterioration in the objective.
n = 4 Choose minimum percentage error
n = 5 Choose node with fewest non-discrete values.
n = 6 Choose node with minimum sum of fractions.
n = 7 Choose according to best projection criterion.

See section 6.5.2.

```
MIP PRIORITY UP ON             * default OFF
MIP PRIORITY UP OFF
```

'ON' invokes UP priority node-selection. See section 6.5.3.

```
MIP AGENDA INPUT ON            * default OFF
MIP AGENDA INPUT OFF
```

'ON' invokes input of MIP AGENDA data. See section 6.6.4

```
MIP AGENDA OUTPUT ON          * default OFF
MIP AGENDA OUTPUT OFF
MIP AGENDA OUTPUT ALL
```

'ON' and 'ALL' activate output of FIX data representing integer solutions. See section 6.6.5.

```
MIP LIST INPUT ON             * default OFF
MIP LIST INPUT OFF
```

'ON' invokes input of unnamed MIP AGENDA data. See section 6.6.6.

```
MIP LIST OUTPUT ON            * default OFF
MIP LIST OUTPUT OFF
MIP LIST OUTPUT ALL
```

'ON' and 'ALL' activate output of un-named FIX data representing integer solutions. See section 6.6.6.

```
MIP PREPROCESS ON             * default OFF
MIP PREPROCESS OFF
MIP PREPROCESS ROOT ONLY
```

'ON' activates the MIP preprocessor throughout, 'ROOT ONLY' for the relaxed LP and not for sub-problems during branch and bound. See section 6.7.2.

```
MIP PREPROCESS LEVEL = n      *  1-3, default n = 3
MIP PREPROCESS LOG LEVEL = n      *  0-4, default n = 1
```

These commands are similar to corresponding 'PRESOLVE' commands, See section 6.7.2.

```
MIP FIX QUOTA = n             * default n = 10
```

Assigns quota for number of stored 'fixes' in relation to the nodes. See section 6.7.2.

```
GENERATE CUTS ON              * default OFF
GENERATE CUTS OFF
```

'ON' activates the cut-generation procedures and the application of strong cuts before and during Branch and Bound. See section 6.7.3.

```
CUT QUOTA = n                 * default n = 10
ACT QUOTA = n                 * default n = 5
```

These commands assign quotas for cut-storage and applied cuts in relation to the model size. See section 6.7.3.

```
MIP ANALYSE DUAL ON           * default OFF
MIP ANALYSE DUAL OFF
```

'ON' activates the Dual Solution analysis and resultant fixing of non-basic discrete variables. See section 6.7.4.

```
MIP ROUNDING FRACTION = v      * default v = 0.0
```

Criterion for a discrete variable to be rounded in the 'AUTO ROUND' option. See section 6.8.1.

```
MIP AUTO ROUNDING ON           * default OFF
MIP AUTO ROUNDING OFF
```

'ON' activates the 'AUTO ROUNDING' option. See section 6.8.1.

```
MIP AROUND SOLVER IPM          * default SSX
MIP AROUND SOLVER SSX
```

These commands select which solver to use for the root node of the auto-rounded tree. See section 6.8.1.

```
INTEGER TOLERANCE = v          * default v = 0.001
```

Criterion for approximation to an integer value. See section 6.8.2.

```
MIP BOUND = v                  * default infinite
MIP BOUND RELATIVE = v         * default infinite
```

Assigns the starting gap beyond the RLP optimum to the initial tree cutoff bound. 'RELATIVE' implies relative to the RLP optimum value. See section 6.8.2.

```
MIP CUTOFF TOLERANCE = v       * default v = 1.0e-12
MIP CUTOFF RELATIVE = v        * default v = 0.0
MIP CUTOFF RELISOL = v         * default v = 0.0
```

Measure by which the gap from RLP to tree cutoff is reduced, once an integer solution has been found. 'RELATIVE' implies relative to the gap, and 'RELISOL' implies relative to the integer solution. See section 6.8.2

```
MAXIMUM MIP NODES = n          * default n = 50,000
MAXIMUM MIP TIME = v           * default v = 50,000.0
MAXIMUM MIP INTEGER SOLUTIONS = n
MAXIMUM MIP INTSOL = n         * default n = 300
```

Limits placed on the tree search. See section 6.8.3.

```
MAXIMUM AROUND NODES = n       * default n = 5000
MAXIMUM AROUND TIME = v        * default v = 5000.0
MAXIMUM AROUND INTEGER SOLUTIONS = n
MAXIMUM AROUND INTSOL = n      * default n = 1
```

Limits placed on the auto-rounded tree search. See section 6.8.3.

```
MAXIMUM MIP SPACE = n          * default n = 10,000
```

Limit on node storage (does not limit the execution). See section 6.8.3.

```
MIP SAVE FREQUENCY = n          * default n = 500
```

Save interval given in terms of the number of nodes built. All saving is cancelled if *n* is zero. See section 6.8.4.

```
MIP RESTART ON                  * default OFF
MIP RESTART OFF
```

'ON' activates a restart, using save-files from a previous run. See section 6.8.4.

```
MIP ON
MIP OFF                         * default ON
```

'OFF' causes Branch and Bound execution to be skipped. See section 6.8.5.

```
MIP DUAL ON
MIP DUAL OFF                    * default ON
```

'ON' causes Primal algorithm to be used in the place of Dual as solver for sub-problems. See section 6.8.6.

```
MIP LOG LEVEL = n               * 0-4, default n = 1
```

Level of output written to the log. Levels 0 to 4 available. Level 2 activates the node output log - also to the screen. See section 6.9.1.

```
NODE LOG FREQUENCY = 1          * default n = 1
```

Frequency of node output written to the log and displayed on screen with log level 2 or higher. See section 6.9.1.

```
MIP CLASSIFY ROWS ON            * default OFF
MIP CLASSIFY ROWS OFF
```

'ON' causes the MIP constraint classification to be carried out in full and written to the log. See section 6.10.1.

Back to Chapter contents

Chapter 7. FortMP Subroutine Library and External Data Interface

Contents

7.1 Using FortMP as a sub-system to solve linear problems

7.1.1 Incorporating FortMP as a sub-system

The main program of the FortMP system is a module named FORTMP and has been included for the users in source form so that they can adapt it to their own ends.

One simple idea is to make it into a subroutine without arguments so that it can be called with the following program statement:-

```
CALL FORTMP
```

This will now carry out exactly the function of the stand-alone FortMP system each time that the CALL is executed. The users build a complete program for execution by combining their own programming together with the modified 'FORTMP' module and the rest of the FortMP callable library.

This simple procedure will not, however, suit many users. A more powerful use can be made of subroutine FORTMP if its composition is studied. It can then be re-organised with other user-procedures interpolated.

The study will be found quite simple because FORTMP is simply a driving routine that calls a sequence of library subroutines in order to carry out the work.

Subroutines that can be called by FORTMP are:-

MPTIME	Routine to create time reports
INITAL	Initialises and reads the SPECS control file
INPUT	Reads the input data (including input basis, if any)
GETCTL	Obtains the control switches
SCALE	Performs scaling
PREMIP	MIP pre-processor
PRSLVE	Carries out PRESOLVE
IPMCTL	Controls execution of IPM and basis recovery
INPBAS	Sets up the input MPS-format basis
BBASIN	Reads and sets up the saved binary basis
CRASH	Carries out CRASH starting basis procedure
UNIBAS	Sets up the UNIT starting basis
DUAL	Executes the DUAL algorithm
PRIMAL	Executes the PRIMAL algorithm
PSTSLV	Carries out POSTSOLVE
BASOUT	Writes the output, MPS-format basis
BBASOT	Writes the output binary save basis

MIPCTL	Controls execution of MIP
OUTPUT	Writes the solution output
FINIAL	Finishes execution and closes any open files

Almost all the execution is conditional, controlled by the switches which the user can set to ON or OFF in the SPECS commands. Those switches that control the main FORTMP operation are recovered by subroutine GETCTL and their details may be clearly understood by reference to the source code of the main control routine, of which a documented copy is provided.

Back to Chapter contents

7.1.2 Parameter specifications

The following three parameters appear as arguments in the library subroutine calls:

SPID SPECS identity

This is a CHARACTER*8 variable (or constant) which selects a part of the SPECS file so that different SPECS-commands may be provided for different calls to the FortMP sub-system. Sections of the SPECS-file are delimited with BEGIN.....END commands and the identity of a section is named on the BEGIN line as follows:

```
BEGIN (identity)
```

where the default identity is blank if omitted. The following special values for SPID should be noted:

'NOSPECS'	Defaults all commands without any reference to the SPECS file.
Blank	Always selects the first section on the SPECS file even if the 'BEGIN' identity is non-blank..

SPID is an argument to the INITAL subroutine

STSL Status of the solution

This is an INTEGER variable, output from the solver routines DUAL, PRIMAL, IPMCTL and MIPCTL which give the status of the solution obtained. Standard values are:-

STSL = 0	No solution has been obtained
STSL = 1	The problem is infeasible
STSL = 2	The problem is unbounded
STSL = 3	The optimum solution has been obtained
STSL = 4	An integer solution has been obtained
STSL = 5	The optimum integer solution has been obtained

TCTN Terminate criterion

This is an INTEGER variable, output from all subroutines with the exception of FINIAL. It states whether the calculation can continue, or whether a fatal error took place and the run is aborted.

TCTN is set to zero by subroutine INITAL, which should never give a fatal error. Thereafter it is used as an input by the remaining subroutines which will halt immediately without executing anything further if they find that TCTN has a non-zero value. Thus it acts as a bypass to the final exit if a fatal error occurs at any point during the execution.

The following is a list of the switches controlling the FORTMP main routine (in each case 1=YES, 0=NO):

ISCL	Whether to scale the data
IPRE	Whether to PRESOLVE
IIPM	Whether to use IPM as the principal solver
IBIN	Whether to start SSX solvers from an MPS input basis
IBBN	Whether to re-start SSX solvers from a saved (binary) basis.
ICRS	Whether to crash to a starting basis for SSX solvers
IDUL	Whether to use DUAL as the principal solver
IPST	Whether to POSTSOLVE
IBOT	Whether to output an MPS-form basis of the (LP) optimal solution
IMIP	Whether Branch and Bound is to be applied after solving the relaxed LP for a problem having discrete variable-types
IOUT	Whether to output the final solution.

There are also two parameters associated with the timing routine as follows:

TTYP	Type:- Zero to start the timer at the outset, and One otherwise
TTEXT	12-character text to identify the execution that is timed.

Each call with TTYP=1 records the time taken since the previous call of the timer. In practice both ttyp and ttext are passed to the timer routine (MPTIME) as constants.

Back to Chapter contents

7.1.3 Simple FORTMP subroutine with parameters

In the main program FORTMP, the variables SPID, STSL and TCTN described in the previous section are declared as local variables. An effective way to use FortMP as a sub-system is to use these three variables as arguments in the main subroutine call. The user could write:-

```
SUBROUTINE MYLP(SPID, STSL, TCTN)
```

and follow this by copying all the declarations without change. The initialisation statement:

```
SPID = '          '
```

is omitted so that the external system can supply it for each CALL. The the rest of the FORTMP main program is then copied exactly as is.

The user may also simplify the FORTMP main program by omitting calls to procedures that are not needed and omitting unused switches. This will have the benefit of reducing the number of FortMP callable library routines that need to be linked for the user's final executable system.

The following is an example of a simple subroutine to solve using the SSX primal algorithm:-

```
      SUBROUTINE MYLP(SPID, STSL, TCTN)
C
      CHARACTER*8    SPID
      INTEGER        STSL, TCTN
      EXTERNAL       INITAL, INPUT, SCALE, CRASH
      EXTERNAL       PRIMAL, OUTPUT, FINIAL
C
      STSL = 0
      CALL INITAL(SPID, TCTN)
      CALL INPUT(TCTN)
      CALL SCALE(TCTN)
      CALL CRASH(TCTN)
      CALL PRIMAL(STSL, TCTN)
      CALL OUTPUT(TCTN)
      CALL FINIAL
C
      RETURN
      END
```

Back to Chapter contents

7.1.4 Subroutine call-library specifications

The full list of call-library specifications for those subroutines that are called from main FORTMP is as follows:

```
      CALL MPTIME(TTYP, TTEXT)
      CALL INITAL(SPID, TCTN)
      CALL INPUT(TCTN)
      CALL GETCTL(ISCL, IPRE, IIPM, IBIN, IBBN, ICRS, IDUL,
                  IPST, IOUT, IBOT, IMIP)
      CALL SCALE(TCTN)
      CALL PREMIP(TCTN)
      CALL PRSLVE(TCTN)
      CALL IPMCTL(STSL, TCTN)
      CALL INPBAS(TCTN)
      CALL BBASIN(TCTN)
      CALL CRASH(TCTN)
      CALL UNIBAS(TCTN)
      CALL DUAL(STSL, TCTN)
      CALL PRIMAL(STSL, TCTN)
      CALL PSTSLV(TCTN)
      CALL OUTBAS(TCTN)
      CALL BBASOT(TCTN)
      CALL MIPCTL(STSL, TCTN)
      CALL OUTPUT(TCTN)
      CALL FINIAL
```

where the arguments are as described above in 7.1.2.

Certain rules must be followed by the users in building their own LP-control subroutine:

1) The general sequence of CALLs as given above must be respected.

2) Just one of the five routines IPMCTL, INPBAS, BBASIN, CRASH and UNIBAS, which create a starting basis, must be called prior to executing DUAL or PRIMAL.

3) PRIMAL must be executed before MIPCTL (as implied by rule 1).

Back to Chapter contents

7.2 External Data Interface

7.2.1 Introduction

The external data interface is a supplement to the data input and solution output features of FortMP. Instead of using disk files for these items the data is entirely passed via the argument-lists of CALL-statements which is both faster and in many cases much easier for the user since MPS-form data does not have to be built.

The term 'external' is used here to contrast with 'internal' for the data interface and not to indicate use of a disk file (which could also be construed as an 'external' form of interface). The external data interface described in this chapter is the user's means to convey externally built problem data to and from the FortMP sub-system via arguments.

In the next chapter, library routines are introduced that directly access the internal stored form of the data which has been subject to various changes (sign-change, scaling, pre-processing etc). It is tempting to use internal access for modifications, e.g. to supply changes to the RHS or bound-set, before re-solving a problem. However, if this method is employed the user must be responsible for converting the modifications from the external form to the internal equivalent and this may not be easy. Facilities to do this may be provided in a future release of FortMP.

Arguments comprise input tables and output tables. The input tables are prepared in advance by the user's program and a simple protocol is used for the presentation of sparse matrix data. Once the system is informed that external data interface is being used, the OUTPUT module suppresses its disk-file output in favour of tables containing solution values etc.

There are two ways to employ the external data interface, by a single call or by separate input and output calls.

In the single call method the user does not prepare any 'MYLP' subroutine as above. He makes a call to subroutine called:

SUBMP1

with an argument list that includes both input tables and output tables

In the separate input and output call method the user prepares a 'MYLP' subroutine or simply interpolates calls to the FortMP subroutine library within his own main program (respecting the rules given in 7.1.4 above). The calls to INPUT and OUTPUT are replaced by calls to the following subroutines:

INPMP1
OUTMP1

In addition when the user supplies a starting basis via the external data interface the following subroutine is used:

IBSMP1

with appropriate argument lists in each case.

Back to Chapter contents

7.2.2 General description of the data interface

The general form of problem statement to be used is that of "two sided linear" constraints given in Part I, Chapter 1, Section 1.7. For clarity this is restated here as follows:

Given a set of n variables $x_1,x_2,...,x_n$ and a set of n corresponding constants $c_1,c_2,...,c_n$. Minimize (or maximize) the following linear function:

$$c_1x_1 + c_2x_2 + ... + c_nx_n$$

subject to the following m linear constraints:

$$L_i \pounds a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n \pounds U_i$$

for $i = 1,2,...,m$

where $a_{11},...,a_{mn}$ is an $m \times n$ matrix of constant constraint coefficients and $L_1,...,L_m$ and $U_1,...U_m$ are constant lower and upper bounds respectively for each constraint row.

Also subject to the following bounds:

$$l_j \pounds x_j \pounds u_j \text{ for } j = 1,2,...n$$

where $l_1,...,l_n$ and $u_1,...u_n$ are constant lower and upper bounds respectively for each variable.

This is illustrated in diagram 6-1 below

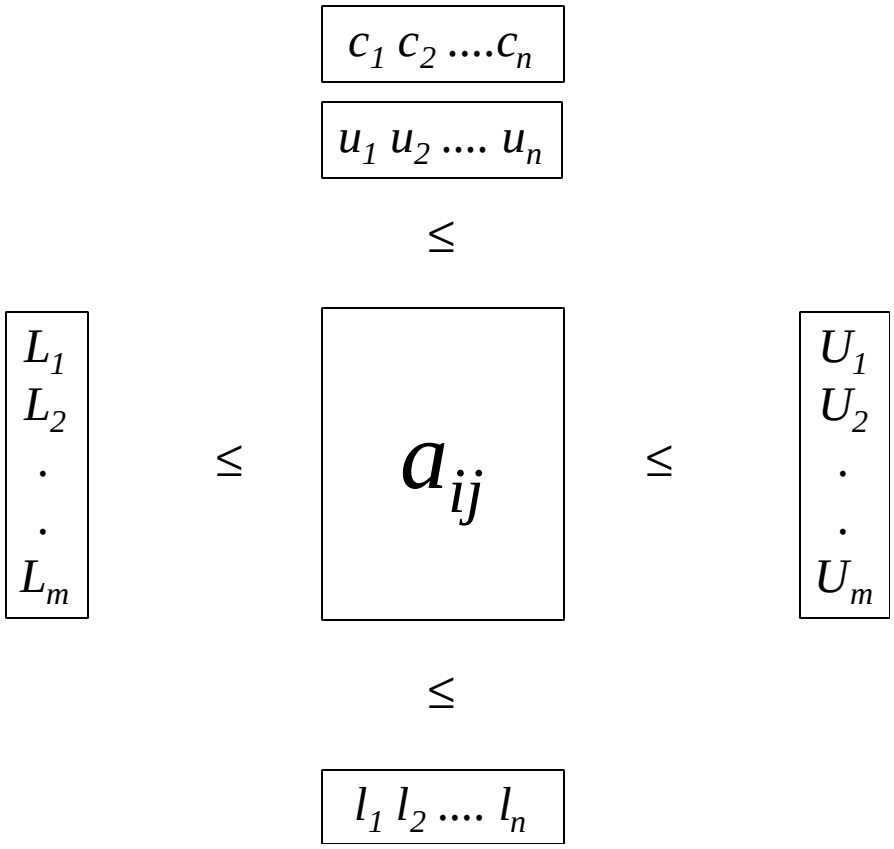


Diagram 6-1 General LP problem with Simple Upper Bounds and RHS ranges

Each of the bound values l_j, u_j, L_i and U_i in the above formulation may take an infinite value to indicate that that bound does not exist (negative infinite for a lower bound or lower RHS). Unlike the original MPS-form input where such cases are indicated by type codes (row-types or bound-types for columns) a standard 'big' value is defined and any value having this magnitude or larger is treated as infinite; that is the corresponding bound does not exist.

For Mixed Integer data, additional tables are provided in the interface which detail the specific mixed-integer-types of each variable and provide the structure of any special ordered sets that may be present. A description of these tables is given in 7.2.3 below.

Back to Chapter contents

7.2.3 Argument specifications

In this section, certain arguments have their type specified as:

DOUBLE PRECISION/REAL

These arguments are DOUBLE PRECISION in a standard double precision version of FortMP. However some users with a need to conserve memory may be provided with a single precision version and for these users the type is REAL

Arrays specified simply as 'DOUBLE PRECISION' are always of type double precision.

The following are INPUT arguments and will not be changed by a subroutine call:

SCALARS:-

MR is the INTEGER number of rows.

NC is the INTEGER number of columns.

NAIJ is the INTEGER number of non-zero entries in the A-matrix.

NSET is the INTEGER number of special ordered sets.

PNAME is the CHARACTER*8 model name.

SPID is the CHARACTER*(*) identity of the BEGIN-line in the SPECS file. If blank then the first BEGIN command is chosen whatever its identity.

TCTN is an INTEGER type variable (also an OUTPUT argument) holding the terminate criterion. A value zero indicates the execution can continue, otherwise no execution takes place.

ARRAYS:-

AIJ is a DOUBLE PRECISION/REAL type array of size NAIJ which holds the non-zero elements of the A-matrix.

ROWIN is an INTEGER type array of size NAIJ. This holds the row indices of the A_matrix belonging to the corresponding entries in AIJ.

COLIN is an INTEGER type array of size NAIJ. This holds the column indices of the A_matrix belonging to the corresponding entries in AIJ.

The arrays AIJ, ROWIN and COLIN do not need to be organised in any particular sequence so long as the entries correspond to each other.

UPB is a DOUBLE PRECISION/REAL type array of size NC. This holds the upper bound values on columns. Any value greater than or equal to 10^{31} indicates that no upper bound exists.

LOB is a DOUBLE PRECISION/REAL type array of size NC. This holds the lower bound values on columns. Any value less than or equal to -10^{31} indicates that no lower bound exists.

RHS is a DOUBLE PRECISION/REAL type array of size MR. This holds the upper bound values on rows (RHS). Any value greater than or equal to 10^{31} indicates that no upper bound exists.

LHS is a DOUBLE PRECISION/REAL type array of size MR. This holds the lower bound values on rows (LHS). Any value less than or equal to -10^{31} indicates that no lower bound exists.

COST is a DOUBLE PRECISION/REAL type array of size NC. This holds the cost coefficient values in the objective function.

MITYP is an INTEGER type array of size nc. This holds a variable type code for MIP as follows:-

CODE	MEANING
0	Continuous variable
1	Binary Variable
2	General Integer variable
3	Semi Continuous variable
4	Member of an SOS type one
5	Member of an SOS type two

SREF is an INTEGER type array of size NSET. This array holds the reference row numbers of each set.

SFUN is an INTEGER type array of size NSET. This array holds the function row number of the sets.

SBEG is an INTEGER type array of size NSET. This array holds the first column number in the set.

SEND is an INTEGER type array of size NSET. This array holds the last column number of the set.

In the tables which transfer output values back to the user, both logical and structural variables are represented in this order, plus in addition one extra position for the objective function at the beginning. Table-size is therefore

```
1 + MR + NC
```

where:-

Position 1 refers to the objective
Positions 2 to (1+MR) refer to logicals
Positions (2+MR) to (1+MR+NC) refer to structurals

The tables are:-

VCSOL is a DOUBLE PRECISION type array of size (1+MR+NC). This array holds the primal solution values.

BSTAT is an INTEGER array of size (1+MR+NC). This array holds a code value for the basis status of each variable in the final solution. Codes are:-

CODE	MEANING
0	basic variable
-1	Variable is at its lower bound
+1	Variable is at its upper bound

BSTAT may also be an INPUT argument - see note 5 of 7.2.4 below.

RSCOS is a DOUBLE PRECISION type array of size (1+MR+NC). This array holds the dual solution values, that is shadow prices and reduced costs.

In addition the following scalars are output:-

STSL is an INTEGER type variable holding solution status as already specified above in 7.1.2

TCTN is an INTEGER type variable (also an INPUT argument) holding the terminate criterion. A value zero indicates that the execution was successful, subject to the value of STSL.

Back to Chapter contents

7.2.4 Call specifications

Full call specifications for the data-interfacing subroutines are:-

```
      CALL SUBMP1(MR, NC, NAIJ, NSET, PNAME, SPID,
     *            AIJ, ROWIN, COLIN, UPB, LOB, RHS, LHS,
     *            COST, MITYPE, SREF, SFUN, SBEG, SEND,
     *            VCSOL, BSTAT, RSCOS, STSL, TCTN)
```

This call is for the complete solution process with both problem data inputs and solution outputs passed via the arguments (external interface).

```
      CALL INPMP1(MR, NC, NAIJ, NSET, PNAME,
     *           AIJ, ROWIN, COLIN, UPB, LOB, RHS, LHS,
     *             COST, MITYPE, SREF, SFUN, SBEG, SEND, TCTN)
```

This call converts users problem data to internal form.

```
      CALL IBSMP1(MR, NC, BSTAT, TCTN)
```

This call sets up a starting basis for SSX solvers PRIMAL and DUAL.

```
      CALL OUTMP1(MR, NC,
     *            VCSOL, BSTAT, RSCOS, TCTN)
```

This call converts the solution back to external form.

Note 1.

Within a users 'MYLP' routine (not SUBMP1) it is legal to employ INPMP1 in combination with the standard OUTPUT or to employ standard INPUT in combination with OUTMP1. However in the latter case the user must have precise knowledge of the row-size, MR, and column-size, NC, in order to supply correct array-sizes in the argument list (a requirement of FORTRAN). This is checked by the system.

Note 2.

A minor problem arises when the input data has no special ordered sets of either type. NSET would be zero in this case. However many FORTRAN compilers do not allow zero dimension sizes and so the user needs a compromise. He should supply dimension size:

```
NSET = 1
```

together with dummy arrays SREF, SFUN, SBEG and SEND which will not be overwritten. The system computes the correct value for NSET in any case from the input data.

Note 3.

The table MITYPE must be supplied whether or not the data has any MIP-type variables. If it has none then MITYPE must be filled with zeros.

Note 4.

It is not necessary to specify input type and/or output type in the SPECS file. The system will enforce correct settings for the combination of interface input and/or interface output after the SPECS commands have been read in.

Note 5.

For the IBSMP1 subroutine BSTAT is an input argument. Its execution is invoked within SUBMP1 by the SPECS command:

```
SIMPLEX START INPUT BASIS
```

and in this case BSTAT is also an input argument for SUBMP1.

Back to Chapter contents

7.2.5 An example

The following simple example is provided as part of the delivered software:

Model: Sample

Maximize $x_1 + 2x_5 - x_8$

Subject to row bounds,

$$
\begin{array}{lll}
2.5 \leq & 3x_1 + x_2 - 2x_4 - x_5 - x_8 & \\
& 2x_2 + 1.1x_3 & \leq 2.1 \\
& x_3 + x_6 & = 4.0 \\
1.8 \leq & 2.8x_4 - 1.2x_7 & \leq 5.0 \\
3.0 \leq & 5.6x_1 + x_5 + 1.9x_8 & \leq 15.0
\end{array}
$$

and column bounds:

$$
\begin{array}{l}
`2.5 \leq x_1 \\
0 \leq x_2 \leq 4.1 \\
0 \leq x_3 \\
0 \leq x_4 \\
0.5 \leq x_5 \leq 4.0 \\
0 \leq x_6 \\
0 \leq x_7 \\
0 \leq x_8 \leq 4.3
\end{array}
$$

The input scalar arguments to be supplied are as follows:

MR	=	5	(number of rows)
NC	=	8	(number of columns)
NAIJ	=	14	(number of non-zeros)
NSET	=	1	(dummy in the absence of any special ordered sets)
PNAME	=	'SAMPLE '	(problem name)
SPID	=	' '	(1st 'BEGIN' line in the SPECS command file)
TCTN	=	0	(terminate criterion - also an output)

The input array arguments to be supplied are as follows:

1) Matrix non-zeros

	ROWIN Row indices	COLIN Column indices	AIJ Non-zeros
1	1	1	3.0

2	5	1	5.6
3	1	2	1.0
4	2	2	2.0
5	2	3	1.1
6	3	3	1.0
7	1	4	-2.0
8	4	4	2.8
9	1	5	-1.0
10	5	5	1.0
11	3	6	1.0
12	4	7	-1.2
13	1	8	-1.0
14	5	8	1.9

2) Column-vectors

	LOB Lower bounds	UPB Upper bounds	COST Objective	MITYPE MIP types
1	2.5	1.0e31	1.0	0
2	0.0	4.1	0.0	0
3	0.0	1.0e31	0.0	0
4	0.0	1.0e31	0.0	0
5	0.5	4.0	2.0	0
6	0.0	1.0e31	0.0	0
7	0.0	1.0e31	0.0	0
8	0.0	4.3	-1.0	0

3) Row-vectors

	LHS Row lower bounds	RHS Row upper bounds

1	2.5	1.0e31
2	-1.0e31	2.1
3	4.0	4.0
4	1.8	5.0
5	3.0	15.0

The remaining inputs are dummy SOS tables that may be single-cell integer arrays or integer scalars whose contents are not material.

The above inputs can be presented either via calling SUBMP1 or via calling INPMP1. When this is done the outputs returned via SUBMP1 or via OUTMP1 will be as follows:

	VCSOL Primal solution	BSTAT Basis status	RSCOS Dual solution
1	4.50	0	-1.0
2	6.26	0	0.00
3	2.10	0	0.00
4	4.00	-1	0.00
5	1.80	+1	0.00
6	15.00	-1	2.00
7	2.50	-1	10.20
8	1.05	-1	0.00
9	0.00	-1	0.00
10	0.64	0	0.00
11	1.00	0	0.00
12	4.00	0	0.00
13	0.00	-1	0.00
14	0.00	-1	4.80

The following scalar values are also returned:

STSL	=	3	(Solution status code: Optimum)
TCTN	=	0	(Termination: OK)

Note that for the row-positions 1 to 6 (including objective in position 1) VCSOL represents the value of the row itself ($\Sigma a_{ij}x_j$) whereas the code in BSTAT represents the attached logical variable. When the row equals its lower RHS value the slack is at upper bound and vice-versa. The exception to this is a GE-type row with no upper RHS value which the system negates internally in order to avoid minus-type logicals. For both GE-type rows and LE-type rows the only non-basic status code is -1.

The associated file of SPECS-commands to be provided in file 'fortmp.spc' is as follows:

```
BEGIN
MODEL NAME (sample)
MAXIMIZE
END
```

Back to Chapter contents

7.3 Standard Data Input to the Interface

7.3.1 Calling the standard data input

In using the external data interface, the user is obliged to build the problem data in his own data space with his own coding. Frequently it happens that the original data for this is provided in MPS-form anyway so that the user would be obliged to write an input routine which exactly parallels the input routine of FortMP itself.

Naturally it is desirable to avoid this duplication of effort and FortMP provides a sub-routine MP1INP in the library to do this. The specification is as follows:

```
      CALL MP1INP(MR, NC, NAIJ, NSET, PNAME, SPID,
     *            AIJ, ROWIN, COLIN, UPB, LOB, RHS, LHS,
     *            COST, MITYPE, SREF, SFUN, SBEG, SEND,
     *            BSTAT, TCTN)
```

The arguments here include all the arguments of INPMP1 and of IBSMP1 with a reversal of roll:- Input-type arguments of INPMP1 and IBSMP1 become output-type arguments of MP1INP. In addition, the following argument:

SPID SPECS identity (up to 8 characters)

enables the user to select a specific section of the SPECS file to control the input process.

The actions of MP1INP are as follows:

- Call INITAL to initialise and read the SPECS commands
- Call INPUT to read the problem data and store it internally
- Call one of the basis start-up routines (INPBAS, BBASIN, CRASH or UNIBAS), as indicated by the SPECS. The default is UNIBAS (unit, all-logical basis).
- Transfer the data now stored internally to the layout of the external interface and return it via the arguments to the user.

Subroutine MP1INP is coded in the same module as SUBMP1 and the user is provided with documented source code.

Back to Chapter contents

7.3.2 Access to row-names and column-names

The INPUT subroutine of FortMP is the means whereby row and column names are entered and become known to the system. Since the external interface itself does not provide any communication of names the user-access to names is based on the use of INPUT, or rather on the use of MP1INP which calls INPUT.

The following SPECS command is to be used:

```
INPUT SAVE NAMES ON            * default OFF
```

(OFF may also be given). When this feature is ON the system takes steps to ensure that not only the names themselves but also the associated look-up tables remain available.

In subroutine MP1INP there is an additional step executed after creating the interface which re-loads the names and look-up tables (over-writing the internal stored form of the model) so that the user can employ name look-up utilities provided in the subroutine library. This is conditional on the use of the above SPECS command.

Back to Chapter contents

7.3.3 Looking up the index of a named variable

Given that the user knows the exact name of some variable, the corresponding row or column index in the interface can be established with one of the following two functions:

```
IROW = FINDIR(RNAME)
JCOL = FINDJC(CNAME)
```

where:

RNAME	is a row name (8 characters)
IROW	is the corresponding row index (zero if not found)
CNAME	is a column name (8 characters)
JCOL	is the corresponding column index (zero if not found)

Note that these operations can only take place after execution of MP1INP, the indices refer to the row and column indices of the interface data in the arguments, and the command 'INPUT SAVE NAMES ON' must have been given. It is also possible to execute SUBMP1 with the same SPECS command given and this will retrieve the same names and look-up tables as before, but will not be able to recognize any change to the indexing that the user may have programmed between the calls to MP1INP and SUBMP1. If two or more models have been entered with MP1INP the different models must be distinguished by providing the model name in a SPECS 'MODEL NAME' command to SUBMP1.

In the case of long names (16 characters) the following two functions are to be used in the place of FINDIR and FINDJC:

```
IROW = FNDLIR(LRNAME)
JCOL = FNDLJC(LCNAME)
```

where:

LRNAME	is a row name (16 characters)
LCNAME	is a column name (16 characters)

Back to Chapter contents

7.3.4 Name pattern matching

In cases where the users do not know the exact name to be looked up they may nevertheless wish to know the index-range corresponding to some pattern in order to program action for a particular type of variable or constraint.

A pattern is a text of the same length as a name containing the characters '?', '&' or '#' in positions that do not need to be matched exactly. The significance of these special characters is as follows:

'?'	matches any character
'&'	matches any alphabetic character
'#'	matches any digit

Pattern matching is invoked by one of the following subroutine calls:

```
CALL MTCHIR(RPTTRN, IROW)
CALL MTCHJC(CPTTRN, JCOL)
```

where:

RPTTRN	is a row-name pattern (8 characters)
IROW	indicates the start point for the search and returns the next following row-index of a matched position (zero if not matched)
CPTTRN	is a column-name pattern (8 characters)
JCOL	indicates the start point for the search and returns the next following column-index of a matched position (zero if not matched)

Search begins at the next index-position after the start-index and continues forward until a match is reached. To start at position 1 initialise the index to zero.

Note that pattern matching is likely to be slow compared to exact name look-up. The same conditions apply, namely that MP1INP must have been executed and the SPECS command 'INPUT SAVE NAMES ON' must have been given.

In the case of long (16 character) names the following subroutine-calls are to be used in the place of MTCHIR, MTCHJC:

CALL MTCLIR(LRPTRN, IROW)
CALL MTCLJC(LCPTRN, JCOL)

where:

RPTTRN	is a row-name pattern (8 characters)
CPTTRN	is a column-name pattern (8 characters)

Back to Chapter contents

7.3.5 Looking up the name of an indexed variable

In order to look up the name corresponding to a row index or column index one of the following functions is employed:

```
RNAME = GIRNAM(IROW)
CNAME = GJCNAM(JCOL)
```

for ordinary 8-character names, or

```
LRNAME = GIRLNM(IROW)
LCNAME = GJCLNM(JCOL)
```

for long 16-character names. These routines are simply the reverse of the name look-up functions specified in 7.3.3 above.

The same conditions apply, namely that MP1INP must have been executed and the SPECS command 'INPUT SAVE NAMES ON' must have been given.

Back to Chapter contents

7.3.6 Managing the constant term in the objective

Following the use of the input subroutines MP1INP it is expected that the user will employ SUBMP1 to solve the model. The entire model is then passed to the solver with the single exception of the constant term in the objective.

The reason for this is that MPS-form data, by using an ordinary matrix row for the objective, actually allows a more general linear form:

$$\sum_{j=1}^{N} c_j x_j \; - \; K$$

to be minimised or maximised where K is a constant. K is given as the right hand side value for the objective row.

On the other hand the arguments for SUBMP1 have no such provision since the objective is given in a separate array from the matrix.

One way to overcome this problem is to use the following SPECS command in the call to SUBMP1:

```
OBJECTIVE OFFSET ON            * default OFF
```

(OFF may also be given). When ON is specified the offset K that was stored by MP1INP is used as it stands instead of being zeroed as will happen in the default 'OFF' situation.

The above will be satisfactory whenever the call to SUBMP1 has been preceded by a corresponding call to MP1INP which provides K. However if the user is alternately solving two or

more different models, then K must be provided explicitly for each call to SUBMP1. The user will also need to retrieve the different values of K after each call to MP1INP.

Access to read or write K is provided by the following subroutine calls:

```
CALL GTKOBJ(KOBJ)
CALL PTKOBJ(KOBJ)
```

where `'KOBJ'` is the objective offset value (RHS constant term) K. Subroutine GTKOBJ 'gets' K and PTKOBJ 'puts' K.

The use of PTKOBJ does not change the default for SPECS command 'OBJECTIVE OFFSET ON' which remains OFF. If omitted the value of K is set to zero.

Back to Chapter contents

7.4 Internal SPECS Commands

7.4.1 The need for faster command entry

During a single run that employs SUBMP1 to solve sub-problems, or uses some combination of FortMP library calls including a solver, a very large number of sub-problems may be solved in a very short space of time. In one stochastic programming example with sub-problems averaging 100 rows, 500 columns and 5000 non-zeros the solution rate achieved was approximately 100 sub-problems per second.

Such rates are impossible without eliminating all the repeated opening and closing of files that could take place on every sub-problem, and in particular the I/O required for SPECS commands. Ways to achieve this are described below - techniques to avoid the use of miscellaneous internal files are given later in section 7.5

Back to Chapter contents

7.4.2 Once-off entry of SPECS commands

In order to avoid repeated I/O from the SPECS-file (fortmp.spc), it is necessary to read the file and store it internally. This read-in is executed in the user's main system as part of the overall initialization before any other FortMP library call is made.

Internal SPECS-command entry is invoked with the following subroutine call:

```
CALL SPCINT(TCTN)
```

Non-zero is returned by TCTN if there is any error.

Once SPCINT has been called the internal procedure INITAL automatically avoids file input for the SPECS commands and uses the internal store, with no other change on the user's part.

Back to Chapter contents

7.4.3 Default initialisation

Defaults for all SPECS commands are stored in a memory region that is initialized when the system is loaded. A library routine is available for the user to change these defaults, and by this means the necessity for a SPECS file may be entirely avoided in many cases.

The library routine can only change defaults and not to actual parameters. Interference with the actual control parameters would be dangerous because there are several inter-dependencies checked at the time of SPECS command entry. However changes to the defaults cannot cause any harm provided that in every case the user respects the maximum and minimum limits given for each separate control.

In order to do this the following call is made:

```
CALL SPCDFT(COMMND, TCTN)
```

where 'COMMND' is a text of up to 50 characters holding the actual SPECS-command, If the command cannot be recognized then TCTN is returned with a non-zero value indicating the error as follows:

1-4	Corresponding keyword not recognized
5	Value out of range.

If this system is employed as a way to avoid use of a SPECS file then the parameter 'SPID' (8 characters) must be assigned as follows:

```
SPID = 'NOSPECS'
```

before each call to library routines SUBMP1, MP1INP or INITAL.

One warning however:- If different controls are used for different entries then do not forget to reset the standard defaults for unused controls as well as setting fresh defaults for each new entry to the FortMP sub-system.

Back to Chapter contents

7.4.4 Common Sections in the SPECS file

When the SPECS-file (fortmp.spc) is read in by the subroutine SPCINT there is a reserved section-heading for setting default values to the commands as follows:

```
BEGIN (DEFAULT)
```

This section is not stored for later interpretation but is acted on at once by invoking 'CALL SPCDFT' for each command in the section (see section 7.4.3 of the manual). It is therefore useful for reducing the number of repetitions of the same command in situations where the SPECS file must have several sections.

The ‘DEFAULT’ section is useful only when SPCINT is used to load the specs commands internally. There is also the possibility of having one section used in all entries to the solver (submp1) in addition to the section designated by ‘SPID’ for each entry. The reserved heading for this section is as follows:

```
BEGIN (ALL)
```

The ‘ALL’ section must appear first, or preceded only by the ‘DEFAULT’ section if that appears. 'ALL' section commands are listed on the log at the beginning, followed by the commands of the section selected by parameter SPID (‘DEFAULT’ section commands are not listed).

Back to Chapter contents

7.5 How to Avoid Miscellaneous I/O

7.5.1 Log channel

In FortMP all I/O channels are initialised to negative numbers indicating that there is no corresponding open file on that channel. To open a file FortMP must first negate the channel, making it positive, and it will be negated again after closing.

With the following SPECS command:

```
LOG CHANNEL = n
```

where 'n' is a positive, valid I/O unit-number, FortMP is forced to consider the file already open and it will not be closed upon exit by FINIAL routine.

Users must make sure that the file is opened before making the first call to a FortMP sub-system and should keep it open throughout the run.

As a result of this the logged output for one sub-problem is never over-written by that of a later sub-problem and the log can usefully be studied after the run. The users can also present the log of their own operations on the same file.

It can be useful to set the log channel by default as described in 7.4.3 above. This avoids the need to set the log channel in every section of the SPECS file.

Back to Chapter contents

7.5.2 Controlling the log

In FortMP the default levels of logging are as follows:

```
SIMPLEX LOG LEVEL = 1
INVERT LOG LEVEL = 1
IPM LOG LEVEL = 1
MIP LOG LEVEL = 1
MIP PREPROCESS LOG LEVEL = 1
PRESOLVE LOG LEVEL = 1
PUSH LOG LEVEL = 1
```

The output produced is generally much more than is needed by the user of the external interface, particularly when a large number of calls are made to solve sub-problems. In order to reduce it the levels can be set to zero which eliminates all output except for error reports. Thus the user should employ the following SPECS commands:

```
SIMPLEX LOG LEVEL = 0
INVERT LOG LEVEL = 0
IPM LOG LEVEL = 0
MIP LOG LEVEL = 0
MIP PREPROCESS LOG LEVEL = 0
```

```
PRESOLVE LOG LEVEL = 0
PUSH LOG LEVEL = 0
```

Not all of these commands will be needed - only those for the FortMP algorithms actually employed.

As before it is possible to use default settings rather than SPECS commands - see section 7.4.3 above.

Back to Chapter contents

7.5.3 Avoiding the use of SAVE files

It is most unlikely that the user of the external interface will require any SAVE files. In the SUBMP1 environment there are initial default settings different from those of stand-alone FortMP which cancel all SAVEs and avoid any opening/closing of SAVE files. These settings are:

```
SIMPLEX SAVE FREQUENCY = 0
IPM SAVE FREQUENCY = 0
MIP SAVE FREQUENCY = 0
```

It is perfectly possible for the user to re-activate any SAVEs that he may need - either by command or by assigning the default settings as stated in 7.4.3.

If the user simply wishes for a final save at termination then a high value should be given for the frequency.

Back to Chapter contents

7.6 MPS-form Output

When a tabular form has been used for the input, or when input data is passed via the external interface, it may be desirable for the user to create an MPS-form equivalent on a disk file. An option to do this immediately after the input can be invoked by means of the following SPECS command:

```
OUTPUT MPS ON                    * Default: OFF
```

(OFF can also be specified). If the model has no names MPSOUT creates row and column names based on the indices with prefix 'R' or 'C'.

The standard file-name given to this output is

```
<probname>.mpo
```

where 'probname' refers to the problem-name. However the user may wish to distinguish outputs from different calls to SUBMP1 without using a different problem-name. This can be achieved by encoding a subroutine library call within a revised SUBMP1 routine, or within a revised FORTMP main program.

Another reason for doing this is that the model is subject to changes as a result of PRESOLVE, or if it has discrete variables as a result of MIP pre-processing and the Branch and Bound algorithm. As before the user may wish to preserve these changes in a form suitable for later re-entry and which can be compared with the original.

The library subroutine which does this task is as follows:

```
CALL MPSOUT(EXTEN)
```

where

EXTEN (CHARACTER*3) is the output file extension.

The file name for output is formed by appending '.exten' to the model name (and directory path if used).

It is expected that the user will incorporate the call to MPSOUT within the coding of FORTMP main program or within SUBMP1 sub-system control routine. An internal model must exist, created either by INPUT or by INPMP1. If INPUT was used and names have been stored then MPSOUT must occur before the final call to FINIAL which closes the names file.

The model written to the output incorporates all variable fixes or constraint relaxations current at the time MPS output is called. The effect of scaling is removed however and the original sign for data elements on MI-type rows and GE-type columns is restored (these are negated internally by applying a scale factor -1.0).

Back to Chapter contents

7.7 Summary of callable library, external data interface and associated commands

7.7.1 Summary of the callable library

The following subroutines are called from the main program of the stand-alone FortMP system. See section 7.1 for details.

```
CALL MPTIME(TTYP, TTEXT)
```

reports processing time (TTYP=0 initialises the timer).

```
CALL INITAL(SPID, TCTN)
```

initialises and reads the SPECS commands.

```
CALL INPUT(TCTN)
```

inputs model data from disk files and store it internally together with basis input data (if any).

```
CALL GETCTL(ISCL, IPRE, IIPM, IBIN, IBBN, ICRS, IDUL,
*            IPST, IOUT, IBOT, IMIP)
```

gets the principal algorithm controls.

```
CALL SCALE(TCTN)
```

scales the model.

```
CALL PREMIP(TCTN)
```

applies MIP pre-processor to the model.

```
CALL PRSLVE(TCTN)
```

applies PRESOLVE to the model and stores the operation sequence for a subsequent POSTSOLVE.

```
CALL IPMCTL(STSL, TCTN)
```

applies the IPM solver and basis recovery (crossover).

```
CALL INPBAS(TCTN)
```

sets up starting basis for PRIMAL or DUAL from the input basis (MPS-form).

```
CALL BBASIN(TCTN)
```

sets up starting basis from one saved in binary form by a previous run.

```
CALL CRASH(TCTN)
```

applies the CRASH algorithm to create a starting basis

```
CALL UNIBAS(TCTN)
```

creates the all-logical starting basis.

```
CALL DUAL(STSL, TCTN)
```

applies the DUAL solver algorithm (valid only of the basis is dual-feasible).

```
CALL PRIMAL(STSL, TCTN)
```

applies the PRIMAL solver algorithm.

```
CALL PSTSLV(TCTN)
```

applies POSTSOLVE to reverse the effect of PRESOLVE operations while maintaining an optimal basis.

```
CALL OUTBAS(TCTN)
```

outputs the current basis in MPS proforma.

```
CALL BBASOT(TCTN)
```

outputs the current basis in binary form (for subsequent restart).

```
CALL MIPCTL(STSL, TCTN)
```

applies the Branch and Bound algorithm to solve a model with discrete variables.

```
CALL OUTPUT(TCTN)
```

outputs the final solution.

```
CALL FINIAL
```

Finishes execution and closes all files before final exit.

The following subroutine calls relate to use of the external interface. See section 7.2 for details.

```
 CALL SUBMP1(MR, NC, NAIJ, NSET, PNAME, SPID,
*            AIJ, ROWIN, COLIN, UPB, LOB, RHS, LHS,
*            COST, MITYPE, SREF, SFUN, SBEG, SEND,
*            VCSOL, BSTAT, RSCOS, STSL, TCTN)
```

Principal sub-system to solve a problem passed via the external (argument) interface.

```
 CALL INPMP1(MR, NC, NAIJ, NSET, PNAME,
*            AIJ, ROWIN, COLIN, UPB, LOB, RHS, LHS,
*            COST, MITYPE, SREF, SFUN, SBEG, SEND, TCTN)
```

Sub-routine to input the external data interface and store the problem internally.

```
CALL IBSMP1(MR, NC, BSTAT, TCTN)
```

Set up starting basis for PRIMAL or DUAL

```
 CALL OUTMP1(MR, NC,
*            VCSOL, BSTAT, RSCOS, TCTN)
```

Output the solution and basis.

```
 CALL MP1INP(MR, NC, NAIJ, NSET, PNAME, SPID,
*            AIJ, ROWIN, COLIN, UPB, LOB, RHS, LHS,
```

```
*            COST, MITYPE, SREF, SFUN, SBEG, SEND,
*            BSTAT, TCTN)
```

Input a problem and convert it to the form of the external (argument) interface returned to the calling routine.

The following functions and subroutines enable access to names (after calling MP1INT). SPECS command 'INPUT SAVE NAMES ON' must be given.

```
IROW = FINDIR(RNAME)
JCOL = FINDJC(CNAME)
IROW = FNDLIR(LRNAME)
JCOL = FNDLJC(LCNAME)
```

These functions look up the index corresponding to a row or column name (8-character or 16-character). See section 7.3.3.

```
CALL MTCHIR(RPTTRN, IROW)
CALL MTCHJC(CPTTRN, JCOL)
CALL MTCLIR(LRPTRN, IROW)
CALL MTCLJC(LCPTRN, JCOL)
```

The above subroutines find the next row index or column index corresponding to a pattern - see section 7.3.4.

```
RNAME = GIRNAM(IROW)
CNAME = GJCNAM(JCOL)
LRNAME = GIRLNM(IROW)
LCNAME = GJCLNM(JCOL)
```

These functions look up the name (8-character or 16-character) that corresponds to a row index or column index. See section 7.3.5.

The following two functions enable read or write access to the objective constant term. SPECS command 'OBJECTIVE OFFSET ON' must be used.

```
CALL GTKOBJ(KOBJ)
CALL PTKOBJ(KOBJ)
```

GTKOBJ gets and PTKOBJ puts the objective constant term. See section 7.3.6.

The following subroutine is for once-off input of SPECS commands. Module SPCINT is to be loaded in the place of SPECFY.

```
CALL SPCINT(TCTN)
```

This subroutine reads the SPECS file and stores all the commands internally. Command sections are identified exactly as usual with the SPID parameter. See section 7.4.2.

The following subroutine sets up the default for a single SPECS command.

```
CALL SPCDFT(COMMND, TCTN)
```

See section 7.4.3.

The following subroutine generates an output MPS-form disk-file from the internally stored problem data with modifications.

```
CALL MPSOUT(EXTEN)
```

See section 7.6.

Back to Chapter contents

7.7.2 Summary of arguments and parameters

The following are arguments to main FortMP library calls. See section 7.1.2.

SPID SPECS identity (CHARACTER*8). Special values are:

Blank	for the first section on the SPECS file
'NOSPECS'	to cancel all input of SPECS commands and use defaults.

STSL Solution status (INTEGER). Values are:

STSL = 0	No solution has been obtained
STSL = 1	The problem is infeasible
STSL = 2	The problem is unbounded
STSL = 3	The optimum solution has been obtained
STSL = 4	An integer solution has been obtained
STSL = 5	The optimum integer solution has been obtained

TCTN Terminate criterion (INTEGER)

The following are parameters controlling the operations in FORTMP and SUBMP1. All are INTEGER, 1=YES, 0=NO. See section 7.1.2.

ISCL Whether to scale the data
IPRE Whether to PRESOLVE
IIPM Whether to use IPM as the principal solver
IBIN Whether to use an MPS input basis
IBBN Whether to use a saved (binary) basis
ICRS Whether to crash to a starting basis
IDUL Whether to use DUAL as the principal solver
IPST Whether to POSTSOLVE
IBOT Whether to output an MPS-form basis of the (LP) optimal solution
IMIP Whether Branch and Bound is to be applied
IOUT Whether to output the final solution.

The following two parameters are associated with the timing routine. See section 7.1.2.

TTYP Zero to start timer, One otherwise (INTEGER)
TTEXT Identification text (CHARACTER*12)

The following arguments constitute the external data interface. Those labelled as 'DOUBLE PRECISION/REAL' must be DOUBLE PRECISION or REAL according to version of FortMP provided (standard delivery is DOUBLE PRECISION). See section 7.2.3.

MR	Number of rows (INTEGER)
NC	Number of columns (INTEGER)
NAIJ	Number of A-matrix non-zeros (INTEGER)
NSET	Number of special ordered sets (INTEGER)
PNAME	Model name (CHARACTER*8)
SPID	SPECS identity (CHARACTER*(*))
TCTN	Terminate control number (INTEGER)
STSL	Solution status (see above) (INTEGER)
AIJ(NAIJ)	A-matrix non-zeros (DOUBLE PRECISION/REAL)
ROWIN(NAIJ)	A-matrix row indices (INTEGER)
COLIN(NAIJ)	A-matrix column indices (INTEGER)
UPB(NC)	Variable upper bounds (DOUBLE PRECISION/REAL)
LOB(NC)	Variable lower bounds (DOUBLE PRECISION/REAL)
RHS(MR)	Upper right hand sides (DOUBLE PRECISION /REAL)
LHS(MR)	Lower right hand sides (DOUBLE PRECISION /REAL)
COST(NC)	Objective coefficients (DOUBLE PRECISION /REAL)
MITYP(NC)	MIP type code as follows (INTEGER): 0 = Continuous 1 = Binary 2 = General integer 3 = Semi-continuous 4 = SOS1 member 5 = SOS2 member
SREF(NSET)	Reference-row indices of S.O.sets (INTEGER)
SFUN(NSET)	Function-row indices of S.O.sets (INTEGER)
SBEG(NSET)	Columns indices beginning each S.O.set (INTEGER)
SEND(NSET)	Columns indices ending each S.O.set (INTEGER)
VCSOL(1+MR+NC)	Primal solution values (DOUBLE PRECISION)
BSTAT(1+MR+NC)	Basis status indicators (INTEGER): 0 = Basic -1 = Non-basic at lower bound +1 = Non-basic at upper bound
RSCOS(1+MR+NC)	Dual solution values (DOUBLE PRECISION)

The following arguments and function outputs refer to the access features for model names. See sections 7.3.3 - 7.3.5.

IROW	Row index (INTEGER)
RNAME	Row name (CHARACTER*8)
LRNAME	Row name (CHARACTER*16)
RPTTRN	Row name pattern (CHARACTER*8)

LRPTRN	Row name pattern (CHARACTER*16)
JCOL	Column index (INTEGER)
CNAME	Column name (CHARACTER*8)
LCNAME	Column name (CHARACTER*16)
CPTTRN	Column name pattern (CHARACTER*8)
LCPTRN	Column name pattern (CHARACTER*16)

In a pattern the following characters are wild (see 7.3.4):

'?'	matches any character
'&'	matches any alphabetic character
'#'	matches any digit

The following are miscellaneous arguments.

KOBJ	Constant term in the objective (DOUBLE PRECISION) - see 7.3.6.
SPCFIL	SPECS file name (CHARACTER*n where n is any length up to maximum 70) - see 7.4.2.
COMMND	Command-line to set up a default value with subroutine SPCDFT - see 7.4.3
EXTEN	File extension used by MPSOUT (CHARACTER*3) - see 7.6.

Back to Chapter contents

7.7.3 Summary of relevant SPECS commands

The following SPECS commands have been introduced in this chapter.

- To enable access to names (see section 7.3.2):

```
INPUT SAVE NAMES ON                 * default OFF
INPUT SAVE NAMES OFF
```

'ON' specifies that names and name look-up tables are to be saved and re-loaded as the final operation of MP1INP, SUBMP1 or FORTMP.

- To manage the objective constant term (see section 7.3.6):

```
OBJECTIVE OFFSET ON                 * default OFF
OBJECTIVE OFFSET OFF
```

'ON' indicates that the objective constant term is not zeroed by the input of the external interface (subroutines SUBMP1 or INPMP1).

- To manage the log (see section 7.5.1):

```
LOG CHANNEL = n                     * default internal
```

This command assigns a channel number to the log file and indicates that it is already opened by the user and is not to be closed.

8. Internal Data Interfacing Service Utilities

Contents

8.1 Introduction to the Internal Data Interfacing Service Utilities

8.1.1 Objectives

The objective of the data interfacing service utilities – DISERV utilities in brief – is to provide those who employ FortMP as a subsystem with a means to access directly and modify the LP problem data as seen by the algorithms of the system.

Thus for example the writer of some cutting plane algorithm could use INPUT to read in a problem from an MPS format file, then solve that problem and then, perhaps in an overall iterative procedure, modify the constraints or add new ones and re-solve the problem several times.

As another example researchers into PRESOLVE procedures can gain access to the problem constraints, objective and bounds in order to execute their algorithms and can then apply any changes to variable type or bounds before using FortMP solution algorithms to test the efficiency of their procedures.

Back to Chapter contents

8.1.2 Data Description

It is important to note at the outset the distinction to be made between DISERV utilities described here and the external data interface described in the previous chapter. Whereas the external data interface relates to data which is external to FortMP, passed for example via arguments to subroutine SUBMP1, DISERV utilities relate to data which is internal to FortMP after processing by procedures such as INPUT, SCALE and PRESOLVE.

In both cases the general form of problem statement is 'two-sided linear' as stated in the previous chapter, Section 7.2.2. Some differences to be noted are as follows.

> The objective row is presented as a separate vector in the internal data interface but is an integral part of the A-matrix in DISERV utilities. DISERV users do not need to account for the extra row in their indexing.
>
> In DISERV utilities the corresponding row wise and column wise elements are combined in a single vector. Row positions are given first followed by column positions.
>
> In DISERV utilities the absence of any bound is not necessarily to be shown by an infinite magnitude in the corresponding bound vector. It is indicated by a type code in a separate vector of variable type codes.
>
> Only the following four variable types are recognised by data interfacing service routines.
>
> 1 = Plus type variable
> 2 = Bounded variable

3 = Fixed variable
4 = Free variable

This coding applies equally to structural and to logical variables and may be interpreted for logicals as follows.

1 = LE type row
2 = Row with RHS range
3 = EQ type row
4 = Free row

There is no internal recognition of a minus type structural or GE type row.

The following diagrams illustrate the data that can be accessed and modified with data interfacing service routines:

Logicals	Structurals

1. Matrix data:

I	Aij Objective row

2. RIM vectors

Upper RHS	Upper bounds

Lower RHS	Lower bounds

Variable type codes — 1,2,3,4

MIP type codes (cannot be modified)

3. Solution vectors (cannot be modified)

Row values	Structural values

Primal solution status codes

Shadow prices	Reduced costs

Dual solution status codes

4. Updated Tableau (cannot be modified)

$B^{-1}.[I,Aij\ ]$

Back to Chapter contents

8.1.3 Description of the Utilities

Each of the access utilities provided falls into one of four categories:

General utilities provide necessary housekeeping.

'GET' utilities retrieve data and present it to the user without making any change.

'CHG' utilities change certain specific items in the data.

'ADD' utilities for matrix rows and columns do not actually add new rows or columns to the matrix but rather 'activate' an existing spare row or column which has been provided in advance. See Section 8.1.5 below.

Utilities are provided in connection with the following data categories.

(1) Matrix data

Utilities are provided to 'GET' or 'CHG' by row or by column and individually for single elements. 'ADD' utilities are provided for a row or for a column.

(2) RIM data

Utilities are provided to 'GET' or 'CHG' all the RIM vectors, or simply to 'GET' or 'CHG' a single position within the RIM vectors. Exceptionally the MIP type code vector may be accessed with 'GET' but cannot be changed with 'CHG'

(3) PRIMAL and DUAL solutions

'GET' utilities are provided for both primal and dual solutions, together with associated status vectors.

(4) Updated Tableau

'GET'-type utilities are provided which calculate either a row-vector or a column-vector in the updated tableau.

Back to Chapter contents

8.1.4 How to Use the Utilities – Operating Modes

In any interface where the user has the ability to change aspects of the internal data there must be some necessary housekeeping to ensure that all of the data is mutually consistent. For example some changes will render the ETA file invalid and therefore require that INVERT is executed before any algorithms can be called.

However, it would be grossly inefficient for such housekeeping to be performed after every call in order to make some minor change. For this reason FortMP has two separate operating 'modes' which are as follows.

PROBLEM CHANGE mode

During this mode the user may apply changes to the problem and may add new rows and columns. Algorithms may not be executed.

ALGORITHM mode

During this mode algorithms may be executed but the user may not apply changes. Retrieval access to the primal or dual solution and to the updated tableau is available in this mode.

Retrieval access to 'GET' an aspect of the problem data might conceptually be carried out in either mode. For definiteness however there is a restriction to execute these 'GET's only in the problem change mode (this is because the coding would be different in the two modes).

Following execution of the algorithms 'INPUT', 'SCALE', 'CRASH' etc. the system is in algorithm mode. To change from there the user executes a utility 'BEGCHG' (begin change) to switch into problem change mode and enable all the access and changing utilities. Once these are completed the user executes a utility 'ENDCHG' (end change) to perform necessary housekeeping and enable all the algorithms to execute.

Back to Chapter contents

8.1.5 Necessary Preparation and Provision for Matrix Expansion

New rows and columns cannot physically be added to the data structure. Instead the system provides a feature to make advance provision in the form of 'inactive' (dormant) rows and columns. The following SPECS commands are used:

```
MAXIMUM EXTRA ROWS = nnn
MAXIMUM SPARE ROWSPACE = nnn
MAXIMUM EXTRA COLUMNS = nnn
MAXIMUM SPARE COLSPACE = nnn
```

Spare 'ROWSPACE' and 'COLSPACE' is provided when the matrix is initially built by leaving the former number of positions unused at the end of each column (spare rowspace) and leaving the latter number of positions unused at the end of the whole matrix (spare colspace).

Any update which causes the spare capacity to overflow will be rejected by the system and will cause a fatal error.

Back to Chapter contents

8.1.6 Single and Double precision versions of FortMP

With release 2 of FortMP there is a change from single precision to double precision for storing all problem data internally in order to give more reliability, especially for difficult problems. The user's interface with the internal data is accordingly also changed and REAL variables now become double precision in the standard version..

Some users may wish to retain compatibility with their existing programs, or may prefer the older system because it is more economical in storage. For these users a special version can be delivered having single precision for internal problem data storage and having corresponding arguments of type REAL rather than DOUBLE PRECISION. These arguments have their type specified as:

DOUBLE PRECISION/REAL

in section 8.3.1 below.

Back to Chapter contents

8.2 The Facilities Available

8.2.1 General Facilities

BEGCHG	Begins problem change mode execution.
ENDCHG	Ends problem change mode execution and carries out necessary housekeeping for re-entry into algorithm mode.
GETSIZ	Retrieves problem statistics for the user. This is an initial utility which the user must execute in order to be able to create arrays of the required size.

Back to Chapter contents

8.2.2 Matrix Facilities

ADDCOL	Adds a column by activating the next dormant column provided by the user's 'MAXIMUM EXTRA COLUMNS' command. Inserts into the matrix a set of coefficients supplied for that column by the user.
ADDROW	Adds a row by activating the next dormant row provided by the user's 'MAXIMUM EXTRA ROWS' command. Inserts into the matrix a set of coefficients supplied for that row by the user.
CHGAIJ	This may add or change one element of the A matrix and, by setting AVAL=0.0 may delete the element.
CHGCOL	Deletes the existing coefficients on one matrix column and replaces them by an alternative set supplied by the user.
CHGOBJ	Deletes the existing cost coefficients on the objective row and replaces them by an alternative set supplied by the user.
CHGROW	Deletes the existing coefficients on one matrix row and replaces them by an alternative set supplied by the user.
GETAIJ	Retrieves a single matrix element for the user.
GETCOL	Retrieves one column of the matrix and places the coefficients in a user supplied area.
GETOBJ	Retrieves the objective row of the matrix and places the cost coefficients in a user supplied area.
GETROW	Retrieves one row of the matrix and places the coefficients in a user supplied area.

There are no 'DELETE' facilities corresponding to the 'ADD' facilities for matrix rows and columns. It is not possible to delete any row or column physically from the data because the user would lose the relationship between row and column numbers and the external naming given for constraints and variables. However, the user can achieve the same result by changing row and column type codes. See note at the end of Section 8.2.3.

Back to Chapter contents

8.2.3 RIM Facilities

CHGCTP	(change column type). Changes the lower bound, upper bound and variable type of a single column.
CHGRIM	Replaces all three RIM vectors – Lower bounds, upper bounds and variable types.

CHGRTP	(change row type). Changes the lower RHS, upper RHS and row type of a single row.
GETCTP	(get column type). Retrieves the lower bound, upper bound, variable type and MIP variable type of a single column.
GETRIM	Retrieves all three RIM vectors — Lower bounds, upper bounds and variable types — together with the vector of MIP variable types and places them in user supplied areas.
GETRTP	(get row type). Retrieves the lower RHS, upper RHS and row type of a single row.

Note that the 'deletion' of a row or column can be accomplished in effect by using CHGRTP or CHGCTP. To delete a row give type 'Free' (code 4). To delete a column give type 'Fixed' (code 3) together with the associated fixed value stated both in lower bound and in upper bound.

Back to Chapter contents

8.2.4 Solution Facilities

GETDSL	Retrieves the dual solution vector together with associated status codes and places them in user supplied areas.
GETSOL	Retrieves the primal solution vector together with associated status codes and places them in user-supplied areas.

Back to Chapter contents

8.2.5 Tableau Facilities

Tableau facities enable user to obtain acces to the updated 'Tableau' or matrix after pre-multiplication by the inverse basis matrix.

GTABLC	Calculates one column of the updated tableau
GTABLR	Calculates one row of the updated tableau

Back to Chapter contents

8.3 Specifications

In order to avoid repeated specifications of the same item in this section, we will give argument specifications separately in Section 8.3.1. In the subsequent utility subroutine specifications, arguments will be listed with identical argument names.

8.3.1 Arguments

In this section certain arguments have their type specified as:

DOUBLE PRECISION/REAL

These arguments are DOUBLE PRECISION in a standard double-precision version of FortMP and REAL in a single-precision version of FortMP

The scalar arguments used are as follows.

AVAL	DOUBLE PRECISION/REAL. Matrix element value.
CTP	INTEGER. Column type code: 1 = Plus 2 = Bounded 3 = Fixed 4 = Free
IOBJ	INTEGER. Row number of the objective function.
IROW	INTEGER. Matrix row number.
JCOL	INTEGER. Matrix column number.
JVAR	INTEGER. Problem variable number. Variables comprise Logicals, one per row, followed by Structurals, one per column. For a logical JVAR equals the row number and for a structural JVAR equals MROW plus the column number.
LOV	DOUBLE PRECISION/REAL. Lower bound value of a column.
LRHS	DOUBLE PRECISION/REAL. Lower RHS value of a row.
MAIJ	INTEGER. Maximum available AIJ (matrix element) space.
MIT	INTEGER. Mixed integer type code of a structural variable. Coded as follows: 0 = Continuous variable 1 = Binary variable 2 = Integer variable 3 = Semi-continuous variable 4 = Member of SOS, type 1 (not the 1st) 5 = Member of SOS, type 2 (not the 1st) 12 = 1st member of an SOS, type 1 13 = 1st member of an SOS, type 2
MRACT	INTEGER. Number of active rows. This value increases by one each time a new row is added.
MROW	INTEGER. Total number of rows – including inactive rows and the objective row.

NAIJ — INTEGER. Total used AIJ space – including the spare unused space in each column. The difference (MAIJ – NAIJ) is the space available for new columns.

NANZ — INTEGER. Number of AIJ non-zero elements.

NCMAX — INTEGER. Maximum number of columns

NCNZ — INTEGER. Number of non-zeros in one column.

NCOL — INTEGER. Number of active columns. This value increases by one each time a new column is added.

NRNZ — INTEGER. Number of non-zeros in one row.

OBJ — DOUBLE PRECISION/REAL. Cost coefficient value on the objective row.

RTP — INTEGER. Row type code:

1 = Less than 2 = Bounded
3 = Equals 4 = Free

SLST — INTEGER. Solution status code, coded as:

0 = Solver did not terminate
1 = No feasible solution
2 = Unbounded
3 = Optimal solution found
4 = Integer solution found
5 = Optimum integer solution found

TCTN — INTEGER. This is an important argument used by ALL of the utility subroutines. It gives the 'Terminate Criterion' and so long as it is zero the execution continues normally. On entry to any utility if its value is non-zero then an immediate exit occurs without doing anything. Otherwise if a fatal error occurs it will be given a non-zero value in order to force a bypass through all subsequent stages straight to the final exit of the program. Users should respect this convention and test TCTN themselves before executing any algorithmic work of their own.

UPV — DOUBLE PRECISION/REAL. Upper bound value of a column.

URHS — DOUBLE PRECISION/REAL. Upper RHS value of a row.

The array arguments used are as follows.

BASTAB — INTEGER. Table of non-basic variable assignment codes as follows:

0 = Basic
−1 = Non-basic at lower bound (row is at upper RHS value)
+1 = Non-basic at upper bound (row is at lower RHS value)

Row entries come first, followed at position (MROW+1) by column entries. Dimension/size is (MROW+NCOL).

CVTAB — DOUBLE PRECISION/REAL. Table of non-zero element values in one matrix column. When supplied for an 'ADDCOL' or 'CHGCOL' operation this table has the dimension/size NCNZ. When supplied as an empty area to receive a column in 'GETCOL' it has dimension/size NCOL.

DSLVEC — DOUBLE PRECISION. Dual solution vector. Rows followed by columns as before with dimension/size (MROW+NCOL).

DSSTAB — INTEGER. Table of dual solution status codes to correspond with DSLVEC. Code values are as follows:

1 = Zero. The primal variable is basic
3 = Basic feasible
4 = Basic infeasible negative. The primal variable is at its lower bound
5 = Basic infeasible positive. The primal variable is at its upper bound

Row entries come first, followed at position (MROW+1) by column entries. Dimension/size is (MROW+NCOL).

IRTAB — INTEGER. Table of row index numbers corresponding to entries in CVTAB described above. Dimension/size is the same as CVTAB.

JCTAB — INTEGER. Table of column index numbers corresponding to entries in RVTAB. Dimension/size is the same as RVTAB.

LOTAB — DOUBLE PRECISION/REAL. Table of lower bound and lower RHS values. Row entries come first, followed at position (MROW+1) by column entries. Dimension/size is (MROW+NCOL).

MITTAB — INTEGER. Table of mixed integer type code values coded as for scalar MIT (see above). Dimension/size (NCOL)

OBJTAB — DOUBLE PRECISION/REAL. Table of objective values with one position for each column. Dimension/size is NCOL.

RVTAB — DOUBLE PRECISION/REAL. Table of non-zero element values in one matrix row. When supplied for an 'ADDROW' or 'CHGROW' operation this table has the dimension/size NRNZ. When supplied as an empty area to receive a column in 'GETCOL' it has dimension/size MROW.

SLSTAB — INTEGER. Table of solution status codes to correspond with SOLVEC. Code values are as follows:

1 = Non-basic at lower bound
2 = Non-basic at upper bound
3 = Basic feasible
4 = Basic infeasible, less than lower bound
5 = Basic infeasible, greater than upper bound

SOLVEC — DOUBLE PRECISION. Solution vector. Rows followed by columns with dimension/size (MROW+NCOL).

TCOL — DOUBLE PRECISION. Column vector of the updated Tableau. Dimension size is MROW.

TPTAB — INTEGER. Table of variable type codes as follows:
1 = Plus or less than
2 = Bounded
3 = Fixed or equals
4 = Free
Row entries come first, followed at position (MROW+1) by column entries. Dimension/size is (MROW+NCOL).

TROW — DOUBLE PRECISION. Row vector of the updated Tableau. Dimension size is (MROW+NCOL) with rows followed by columns.

UPTAB — DOUBLE PRECISION/REAL. Table of upper bound and upper RHS values. Row entries come first, followed at position (MROW+1) by column entries. Dimension/size is (MROW+NCOL).

Back to Chapter contents

8.3.2 General Utility Specifications

General utility subroutines are specified as follows.

```
      CALL GETSIZ(MROW, MRACT, NCOL, NCMAX, MAIJ, NAIJ, NANZ,
     *            IOBJ, TCTN)
```

This retrieves problem statistics for the user. It is an initial utility which the user must execute in order to be able to create arrays of the required size.

Input arguments are: TCTN
Output arguments are: MROW, MRACT, NCOL, NCMAX, MAIJ, NAIJ, NANZ, IOBJ, TCTN

```
      CALL BEGCHG (TCTN)
```

Begins problem change mode execution. TCTN is an input/output argument.

```
      CALL ENDCHG (TCTN)
```

Ends problem change mode execution and carries out necessary housekeeping for re-entry into algorithm mode. TCTN is an input/output argument.

Back to Chapter contents

8.3.3 Matrix Utility Specifications

Matrix utility subroutines are specified as follows.

```
CALL GETCOL (MROW, JCOL, CVTAB, IRTAB, NCNZ, OBJ, TCTN)
```

This retrieves one column of the matrix and places the coefficients in the user supplied areas CVTAB and IRTAB. GETCOL also extracts the column count (NCNZ) and objective value (OBJ). Note that OBJ will not be among the coefficients placed in the areas CVTAB and IRTAB.

Input arguments are: MROW, JCOL, TCTN
Output arguments are: CVTAB(MROW), IRTAB(MROW), NCNZ, OBJ, TCTN
Only the first NCNZ positions in CVTAB and IRTAB are filled. The remainder will not be used.

```
CALL GETROW (NCOL, IROW, RVTAB, JCTAB, NRNZ, TCTN)
```

This retrieves one row of the matrix and places the coefficients in the user supplied areas RVTAB and JCTAB. GETROW also retrieves the row count into the argument NRNZ.

Input arguments are: NCOL, IROW, TCTN
Output arguments are: RVTAB(NCOL), JCTAB(NCOL), NCNZ, OBJ, TCTN
Only the first NRNZ positions in RVTAB and JCTAB are filled. The remainder will not be used.

```
CALL GETAIJ (IROW, JCOL, AVAL, TCTN)
```

This retrieves the matrix element on row IROW column JCOL into the variable AVAL.

Input arguments are: IROW, JCOL, TCTN
Output arguments are: AVAL, TCTN

```
CALL GETOBJ (NCOL, OBJTAB, TCTN)
```

This retrieves the objective row of the matrix and places the cost coefficients in the user supplied area OBJTAB. The table is zeroed first and then the non-zero entries are copied in from the matrix at their correct index position.

Input arguments are: NCOL, TCTN
Output arguments are: OBJTAB(NCOL), TCTN

```
CALL ADDCOL (NCNZ, JCOL, CVTAB, IRTAB, OBJ, TCTN)
```

This adds a column by activating the next dormant column provided by the users 'MAXIMUM EXTRA COLUMNS' command. It inserts into the matrix a set of coefficients supplied for that column by the user in areas CVTAB and IRTAB and also inserts the objective value supplied in cell OBJ. The column number activated is returned in cell JCOL which is obtained by adding one to the number of columns NCOL.

Input arguments are: NCNZ, CVTAB(NCNZ), IRTAB(NCNZ), OBJ, TCTN
Output arguments are: JCOL, TCTN

```
CALL ADDROW (NRNZ, IROW, RVTAB, JCTAB, TCTN)
```

This adds a row by activating the next dormant row provided by the user's 'MAXIMUM EXTRA ROWS' command. It inserts into the matrix a set of coefficients supplied for that row in areas RVTAB

and JCTAB. The row number activated is returned in cell IROW which is obtained by adding one to the number of active rows MRACT.

Input arguments are: NRNZ, RVTAB(NRNZ), JCTAB(NRNZ), TCTN
Output arguments are: IROW, TCTN

```
CALL CHGCOL (NCNZ, JCOL, CVTAB, IRTAB, OBJ, TCTN)
```

This deletes the existing coefficients on matrix column JCOL and replaces them by an alternative set supplied by the user in areas CVTAB and IRTAB with the objective coefficient in cell OBJ.

Input arguments are: NCNZ, JCOL, CVTAB(NCNZ), IRTAB(NCNZ), OBJ, TCTN
Output arguments are: TCTN

```
CALL CHGROW (NRNZ, IROW, RVTAB, JCTAB, TCTN)
```

This deletes the existing coefficients in matrix row IROW and replaces them by an alternative set supplied in the areas RVTAB and JCTAB.

Input arguments are: NRNZ, IROW, RVTAB(NRNZ), JCTAB(NRNZ), TCTN
Output arguments are: TCTN

```
CALL CHGAIJ (IROW, JCOL, AVAL, TCTN)
```

This changes the A matrix element on row IROW, column JCOL to the value AVAL. This may add, change or, by setting AVAL=0.0, delete the element.

Input arguments are: IROW, JCOL, AVAL, TCTN
Output arguments are: TCTN

```
CALL CHGOBJ (NCOL, OBJTAB, TCTN)
```

This deletes the existing cost coefficients on the objective row and replaces them by an alternative set supplied by the user in table OBJTAB (only non-zero elements are used).

Input arguments are: NCOL, OBJTAB(NCOL), TCTN
Output arguments are: TCTN

Back to Chapter contents

8.3.4 RIM Utility Specifications

RIM utility subroutine specifications are as follows.

```
CALL GETRIM (MROW, NCOL, LOTAB, UPTAB, TPTAB, MITTAB,
TCTN)
```

Retrieves all the three RIM vectors – Lower bounds, upper bounds and variable types – and places them in user supplied areas LOTAB, UPTAB and TPTAB. See notes (2) and (3) in Section 8.3.7 below.

Input arguments are: MROW, NCOL, TCTN
Output arguments are: LOTAB(MROW+NCOL), UPTAB(MROW+NCOL),
TPTAB(MROW+NCOL), MITTAB(NCOL), TCTN

```
CALL GETCTP (JCOL, LOV, UPV, CTP, MIT, TCTN)
```

This retrieves the lower bound, upper bound and variable type of a single column and places them in cells LOV, UPV and CTP. See note (2) in Section 8.3.7 below.

Input arguments are: JCOL, TCTN
Output arguments are: LOV, UPV, CTP, MIT, TCTN

```
CALL GETRTP (IROW, LRHS, URHS, RTP, TCTN)
```

This retrieves the lower RHS, upper RHS and row type of a single row and places them in cells LRHS, URHS and RTP. See note (3) in Section 8.3.7 below.

Input arguments are: IROW, TCTN
Output arguments are: LRHS, URHS, RTP

```
CALL CHGRIM (MROW, NCOL, LOTAB, UPTAB, TPTAB, TCTN)
```

This replaces all three RIM vectors – Lower bounds, upper bounds and variable types – with the user supplied areas LOTAB, UPTAB and TPTAB. See notes (4), (5) and (6) in Section 8.3.7 below.

Input arguments are: MROW, NCOL, LOTAB(MROW+NCOL), UPTAB(MROW+NCOL),
TPTAB(MROW+NCOL), TCTN
Output arguments are: TCTN

```
CALL CHGCTP (JCOL, LOV, UPV, CTP, TCTN)
```

This changes the lower bound, upper bound and variable type of a single column, replacing them by cells LOV, UPV and CTP. See notes (4) and (6) in Section 8.3.7 below.

Input arguments are: JCOL, LOV, UPV, CTP, TCTN
Output arguments are: TCTN

```
CALL CHGRTP (IROW, LRHS, URHS, RTP, TCTN)
```

This changes the lower RHS, upper RHS and row type of a single row, replacing them by cells LRHS, URHS and RTP. See notes (5) and (6) in Section 8.3.7 below.

Input arguments are: IROW, LRHS, URHS, RTP, TCTN
Output arguments are: TCTN

Back to Chapter contents

8.3.5 Solution Utility Specifications

Solution utulities are specified as follows:

```
CALL GETSOL (MROW, NCOL, SLST, SOLVEC, SLSTAB, TCTN)
```

This retrieves the primal solution vector together with associated status codes and places them in user supplied areas SOLVEC and SLSTAB. This utility cannot execute in problem change mode – see note (7) in Section 8.3.7 below.

Input arguments are: MROW, NCOL, TCTN
Output arguments are: SLST, SOLVEC(MROW+NCOL), SLSTAB(MROW+NCOL), TCTN

```
CALL GETDSL (MROW, NCOL, DSLVEC, DSSTAB, TCTN)
```

This retrieves the dual solution vector together with associated status codes and places them in user supplied areas DSLVEC and DSSTAB. This utility cannot execute in problem change mode – see note (7) in Section 8.3.7 below.

Input arguments are: MROW, NCOL, TCTN
Output arguments are: DSLVEC(MROW+NCOL), DSSTAB(MROW+NCOL), TCTN

Back to Chapter contents

8.3.6 Tableau Utility Specifications

Solution utulities are specified as follows:

```
CALL GTABLC (MROW, JVAR, TCOL, TCTN)
```

This subroutine calculates and retrieves one column-vector from the updated tableau in the user-supplied area TCOL. It cannot be executed in problem change mode - see note (7) is Section 8.3.7 below.

Input arguments are: MROW, JVAR, TCTN
Output arguments are: TCOL(MROW), TCTN

```
CALL GTABLR (MROW, NCOL, IROW, TROW, TCTN)
```

This subroutine calculates and retrieves one column-vector from the updated tableau in the user-supplied area TCOL. It cannot be executed in problem change mode - see note (7) is Section 8.3.7 below.

Input arguments are: MROW, NCOL, IROW, TCTN
Output arguments are: TROW(MROW+NCOL), TCTN

Back to Chapter contents

8.3.7 Some Notes on the Specifications

(1) In several of the utilities the user must supply MROW or NCOL as an input argument in order to give the dimension/size of certain argument tables. Actually the system already knows the values of MROW and NCOL and this is required only because of restrictions in FORTRAN 77.The system checks to see that user has provided correct values and will return an error (TCTN > 0) if not.

(2) For variables given as MINUS type in external data, for example by code ‘MI’ in the MPS form bounds section, the quoted upper bound is negated (together with the matrix column) and presented as lower bound in the DISERV utilities. Type code is 1.

(3) The one-sided RHS value supplied externally, for example in the RHS section of MPS form data, is communicated in the upper RHS value after negating (together with the matrix row) if the original row type was GE. Both GE type and LE type rows become type coded as 1.

(4) Whenever the user supplies new variable bounds to the system the associated type coding must be consistent. A finite lower bound must be present for all type codes other than ‘Free’ (code

4). Lower and upper bounds must be equal for type 'Fixed' (code 3). Finite lower and upper bounds must be present for type 'Bounded' (code 2) with upper bound value greater than lower bound value. For type 'Plus' (code 1) the upper bound is ignored whatever its value.

A system constant with the value 10^{36} is used to check for finite bound values (10^{20} for the single-precision version).

Type coding and bounds supplied must also be consistent with the MIP data types (if non-zero).

(5) Whenever the user supplies new row bounds to the system the associated row type coding must be consistent. A finite upper RHS must be present for all type codes other than 'Free' (code 4). Lower and upper RHS must be equal for type 'Equal' (code 3). Finite lower and upper RHS must be present for type 'Ranged' (code 2) with upper RHS value greater than lower RHS value. For type 'LE' (code 1) the lower RHS is ignored whatever its value.

A system constant with the value 10^{36} is used to check for finite RHS values (10^{20} for the single-precision version).

(6) The 'deletion' of a row or column can be accomplished in effect by using CHGRTP or CHGCTP. To delete a row give type 'Free' (code 4). To delete a column give type 'Fixed' (code 3) with associated lower and upper bound values both equal.

It is not possible to delete any row or column physically from the data.

(7) The solution access utilities GETSOL and GETDSL and the updated tableau utilities GTABLC and GTABLR must execute in ALGORITHM mode and not in PROBLEM CHANGE mode. This is because they require use of the ETA file which in general is not valid during problem change mode.

The user who needs solution values or updated tableau as part of a series of GETs and CHGs etc. should execute GETSOL and GETDSL before making the call to BEGCHG.

(8) The solution vector SOLVEC (obtained by CALL GETSOL) supplies the values of logical variables in positions 1 to MROW. Position IOBJ (obtained by using GETSIZ) is therefore the logical value on the objective row which is equal to the objective value reversed in sign. No separate argument is needed to get the objective value.

Back to Chapter contents

Chapter 9.

The Interior Point Method for Quadratic Programming

Contents

9.1 Statement of the QP Problem

9.1.1 The QP Problem:- Symmetric *Q* form

FortMP IPM for QP is a primal dual approach to the solution of the separable convex QP problem.

The primal QP problem may be stated as follows:

$$\textit{Minimise} \quad c^T x + \frac{1}{2} x^T Q x$$

subject to the conditions:

$$Ax = b$$
$$x + s = u$$
$$x, s \geq 0$$

Where

A is an $m \times n$ matrix of coefficients a_{ij}.

c is an n vector of the cost coefficients $c_1, c_2, \ldots c_n$.

x is an n vector of the structural variables $x_1, x_2, \ldots, x_n$.

u is an n vector of upper bounds $u_1, u_2, \ldots, u_n$.

s is an n vector of variables $s_1, s_2, \ldots s_n$ compliment to x with respect to the upper bounds u.

Q is an $n \times n$ symmetric positive semi-definite matrix.

The dual problem may be stated as follows.

$$\textit{Maximise} \quad b^T y - \frac{1}{2} x^T Q x - u^T w$$

subject to the conditions:

$$A^T y + z - Qx - w = c$$
$$z, w \geq 0$$

Where the variables y_i $(i = 1,\cdots,m)$ are dual variables complementary to the original rows of the primal statement above, z_j $(j = 1,\ldots,n)$ are dual variables complementary to the x_j, and w_j $(j = 1,\ldots,n)$ are dual variables complementary to the s_j.

Back to Chapter contents

9.1.2 The QP problem:- Separable FF^T Form

FortMP currently only handles the case in which it is possible to reformulate Q as $Q = FF^T$. This leads to a separable formulation of the QP problem, which, in the case of the primal problem may be stated as:

$$\textit{Minimise} \quad c^T x + \frac{1}{2} p^T p$$

Subject to the conditions:

$$\begin{aligned} Ax &= b \\ F^T x - p &= 0 \\ x + s &= u \\ x, s &\geq 0 \\ -\infty < p &< \infty \end{aligned}$$

In order that the separable formulation be possible, FortMP requires that either:

1) Q is positive definite and so can be Cholesky factored.

or

2) The matrix F is supplied instead of Q.

Back to Chapter contents

9.1.3 Mixed Integer QP with Binary Variables

By using Branch and Bound we may extend the QP problem statement to include models where some elements of vector x are constrained to be binary variables ($x_j = 0$ or $x_j = 1$). At present the only discrete constraints handled by FortMP-QP are binary variables. Other discrete constraint-types may be added in a future release and in any case it is usually possible to model such constraints entirely with binary variables (suitably chosen).

The general Branch and Bound mechanism has been described in chapter 6. A simplified version is used for Mixed Integer QP (MIQP), which is described below in section 9.5.

Back to Chapter contents

9.2 IPM Solution Procedure

9.2.1 Formulation

The IPM solution procedure is a modification of the predictor-corrector IPM solution for LP (see Chapter 5). Each IPM iteration considers an interior point to be a set of fixed values and looks for a set of 'changes' or 'search directions':

Δx_j, changes to the x_j for $j = 1,2,\cdots,n$
Δs_j, changes to the s_j for $j = 1,2,\cdots,n$
Δy_j, changes to the y_j for $j = 1,2,\cdots,m$
Δz_j, changes to the z_j for $j = 1,2,\cdots,n$
Δw_j, changes to the w_j for $j = 1,2,\cdots,n$

These search directions define a new interior point:

$$(x + aDx, s + aDs, y + aDy, z + aDz, w + aDw)$$

Where a is made as large as possible without taking any variable outside its bound range so that the new point remains an interior point.

The predictor-corrector method first predicts the directions, and then corrector directions are computed. The actual directions are a combination of both the predicted and corrector directions.

If in addition to the matrices and vectors previously defined, we define

X The $n \times n$ diagonal matrix comprising solution point values $x_1, x_2, \cdots, x_n$ on the diagonal and zeros elsewhere.

Z, S and W
Diagonal $n \times n$ matrices comprising dual solution point values from the z_j, the s_j and the w_j on the diagonal and zeros elsewhere.

Δx n-vector of variables $Dx_1, Dx_2, \cdots, Dx_n$.

Δy m-vector of variables $Dy_1, Dy_2, \cdots, Dy_m$.

Δz, Δs and Δw
n-vectors of variables constructed from the Δz_j, the Δs_j and the Δw_j in the same way.

And let the 'predicted' directions be $(Dx^p, Ds^p, Dy^p, Dz^p, Dw^p)$, then solving the following system of equations gives the predicted search directions.

$$ADx^p = b - Ax$$
$$Dx^p + Ds^p = u - x - s$$
$$A^T Dy^p + Dz^p - Dw^p - QDx^p = c - A^T y - z + w + Qx$$
$$ZDx^p + XDz^p = -XZe$$
$$WDs^p + SDw^p = -SWe$$

The corrector search directions $(Dx^c, Ds^c, Dy^c, Dz^c, Dw^c)$ are found by solving

$$ADx^c = b - Ax$$
$$Dx^c + Ds^c = u - x - s$$
$$A^T Dy^c + Dz^c - Dw^c - QDx^c = c - A^T y - z + w + Qx$$
$$ZDx^c + XDz^c = me - XZe - DX^p DZ^p$$
$$WDs^c + SDw^c = me - SWe - DS^p DW^p$$

The search directions (Dx, Ds, Dy, Dz, Dw) are then computed as:

$$(Dx^p + Dx^c, Ds^p + Ds^c, Dy^p + Dy^c, Dz^p + Dz^c, Dw^p + Dw^c)$$

Back to Chapter contents

9.2.2 Solving the System of Equations

The set of equations defining the predictor and corrector search directions respectively differs from each other only in their right hand side. Hence only a single factorization is required for solving both systems.

If we let R_1, R_2, R_3, R_4 and R_5 define constant right-hand side vectors of size n except for R_2 which is of size m, then both of the above sets of equations are of the form:

$$ADx = R_1 \quad (1)$$
$$Dx + Ds = R_2 \quad (2)$$
$$A^T \Delta y + \Delta z - \Delta w - Q\Delta x = R_3 \quad (3)$$
$$Z\Delta x + X\Delta z = R_4 \quad (4)$$
$$W\Delta s + S\Delta w = R_5 \quad (5)$$

To solve these equations we may begin by eliminating all the variables $\Delta s, \Delta z$ and Δw from equations (2), (3), (4) and (5). When this is done the system is reduced to two equations as follows:

$$-(X^{-1}Z + S^{-1}W + Q)\Delta x + A^T \Delta y = R_3' \quad \text{(new RHS for 3rd equation)}$$
$$ADx = R_1$$

This system could now be solved by finding the inverse of the matrix:

$$\begin{pmatrix} -(D+Q) & A^T \\ A & 0 \end{pmatrix}$$

Where D is the diagonal matrix $(X^{-1}Z + S^{-1}W)$.

However, the number of equations can be reduced still further by eliminating the variables Δx across these two equations. This will result in the following:

$$(A(D+Q)^{-1}A^T)Dy = R \quad \text{(new RHS)}.$$

The matrix is a symmetric positive-definite matrix and so may be inverted by the Cholesky procedure.

Back to Chapter contents

9.2.3 Determining the Starting Point

It is vital in the IPM algorithm to have a good starting interior point from which to begin iterations. Failure in this will increase the number of iterations necessary and may indeed lead to divergence or cycling so that a solution can never be found.

The predictor–corrector algorithm for QP computes a starting point in exactly the same way as for LP. As with LP, the predictor-corrector algorithm can employ one of three starting point methods based on a quadratic formula to maximise the distance from boundaries in the initial interior point. The choice of method is described in Section 5.2.2.

Back to Chapter contents

9.2.4 Controls on the Predictor-Corrector Algorithm for QP

The controls for the predictor-corrector algorithm for QP are exactly the same as those for the LP case, except that there is no procedure to develop the IPM solution into a basic solution. On reaching the IPM solution, the output is written and execution halts. The reader is referred to Sections 5.2.1 - 5.2.5.

Back to Chapter contents

9.3 Input Data Layout

The data for the QP problem is entered by using an extension of the MPS format for LP, as described in detail in Appendix A. The MPS layout is extended to include the data definition of the Q or F matrix, (see Section 9.1.).

Back to Chapter contents

9.3.1 Matrix Input for the symmetric *Q* form

When the Q-matrix is to be entered, a separate 'QDATA' section is prepared by the user and presented in the MPS data-file after the 'COLUMNS' section and before the 'ENDATA' line. The header keyword for this section is 'QDATA' and the layout is similar to that of the normal COLUMNS data lines but with column names in the place of row names as follows:

Field 1 (cols 2-3)	Blank
Field 2 (cols 5-12)	Column name
Field 3 (cols 15-22)	Column name
Field 4 (cols 25-36)	Value
Field 5 (cols 15-22)	Column name (optional)
Field 6 (cols 25-36)	Value (if field 5 is entered)

The user enters only the diagonal and upper triangular coefficients of the Q matrix with the name that corresponds to the row in field 2 and names corresponding to columns in fields 3 and 5. It follows that no coupling of columns in fields 2 and 4 or in fields 2 and 5 can be repeated in the opposite order. The normal rule of continuity also applies, that is:

- Rows of the Q-matrix defined by field 2 may not be split

The following restriction also applies:

- Q-matrix rows, that is the columns named in field 2 must appear in the same order as in the COLUMNS section even though this may differ from the natural triangular ordering of the Q-matrix (field 3 and field 5 columns may be named in any order).

This restriction is only necessary to support the present release of FortMP and may be removed in the future.

Example

$$Minimise\, x_1 + x_2 + \frac{1}{2}(x_1, x_2)\begin{pmatrix} 5 & 5 \\ 5 & 10 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Subject to

$$x_1 + x_2 > 10$$
$$x_1, x_2 \geq 0$$

The input data for the above problem is:

```
NAME          PROB2
ROWS
 G  R01
 N  COST
COLUMNS
    X01       R01                1.0
    X02       R01                1.0
RHS
    RHS       R01               10.0
QDATA
    X01       X01                5.0
    X01       X02                5.0
    X02       X02               10.0
ENDATA
```

Back to Chapter contents

9.3.2 Matrix Input for the separable FF^T form

Given that $Q = FF^T$, the user enters the coefficients of the F^T matrix in the ROWS section of the MPS-form input data. The rows are designated as type 'F ' (or ' F').

Example

$$\textit{Minimise}\quad x_1 + x_2 + \frac{1}{2}(x_1, x_2)\begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}\begin{pmatrix} 1 & 3 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Subject to

$$x_1 + x_2 > 10$$
$$x_1, x_2 \geq 0$$

The input data for the above problem is:

```
NAME          PROB1
ROWS
 G  R01
 F  F01
 F  F02
 N  COST
COLUMNS
    X01       R01                1.0
    X01       F01                1.0
    X01       F02                2.0
    X02       R01                1.0
    X02       F01                3.0
    X02       F02                1.0
RHS
    RHS       R01               10.0
ENDATA
```

Back to Chapter contents

9.4 Worked Example

The example presented here is that of chapter 2, section 2.1.1 with the addition of quadratic terms in the objective function. The problem is stated as:

Minimize:

$$4x_1 + 6x_2 + 5x_3 + 16x_4 + 2x_5 + 5x_6 + x_7 + 1/2\{X^TQX\}$$

Subject to:

$$x_1 + x_6 = 2$$
$$x_1 + 3x_2 = 5$$
$$2x_3 + 3x_4 = 4$$
$$x_1 + x_2 + x_3 + 4x_4 + x_5 = 11$$
$$x_1 + 2x_2 + 2x_3 + 3x_4 + x_5 + 2x_6 + x_7 = 14$$
$$0.5 \pounds x_1$$
$$0.5 \pounds x_2$$
$$0.0 \pounds x_3 \pounds 1.0$$
$$x_4 = 1.0$$
$$0.0 \pounds x_5 \pounds 6.0$$
$$1.0 \pounds x_7$$

Where Q is a symmetric matrix comprising the quadratic cost factors, possibly expressible as FF^T.

Back to Chapter contents

9.4.1 Worked example using the *Q* form

In this example the quadratic terms of the objective are as follows:

$$2x_1^2 + x_1.x_2 + x_2^2 + x_3^2 + x_3.x_4 + x_4^2$$

The Q-matrix is as follows:

$$Q \equiv \begin{pmatrix} 4 & 1 & . & . & \dots \\ 1 & 2 & . & . & \dots \\ . & . & 2 & 1 & \dots \\ . & . & 1 & 2 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

With the remaining positions all zero.

Input data presented in a file 'TESTQP.QDT' is shown below in listing (1). Listing (2) shows the associated SPECS commands in file 'FORTMP.SPC'.

```
NAME          TESTQPQD
ROWS
 N  COST
 EQ R1
 EQ R2
 EQ R3
 EQ R4
 EQ R5
COLUMNS
    X1        COST                 4.0   R1                   1.0
    X1        R2                   1.0   R4                   1.0
    X1        R5                   1.0
    X2        COST                 6.0   R2                   3.0
    X2        R4                   1.0   R5                   2.0
    X3        COST                 5.0   R3                   2.0
    X3        R4                   1.0   R5                   2.0
    X4        COST                16.0   R3                   3.0
    X4        R4                   4.0   R5                   3.0
    X5        COST                 2.0   R4                   1.0
    X5        R5                   1.0
    X6        COST                 5.0   R1                   1.0
    X6        R5                   2.0
    X7        COST                 1.0   R5                   1.0
RHS
    RHS       R1                   2.0   R2                   5.0
    RHS       R3                   4.0   R4                  11.0
    RHS       R5                  14.0
BOUNDS
 LO BND       X1                   0.5
 LO BND       X2                   0.5
 UP BND       X3                   1.0
 FX BND       X4                   1.0
 UP BND       X5                   6.0
 LO BND       X7                   1.0
QDATA
    X1        X1                   4.0   X2                   1.0
    X2        X2                   2.0
    X3        X3                   2.0   X4                   1.0
    X4        X4                   2.0
ENDATA
```

Listing (1): Input data using *Q* form in file '`TESTQP.QDT`'

```
BEGIN
MODEL NAME (testqp)
INPUT FILE NAME (testqp.qdt)
OUTPUT FILE NAME (testqp.qrs)
LOG FILE NAME (testqp.qlg)
END
```

Listing (2): Specs commands for *Q*-form in file '`FORTMP.SPC`'.

Outputs comprising the solution in file 'TESTQP.RES' and the log in file 'TESTQP.LOG' are shown below in listings (3) and (4).

```
FORTMP SOLUTION REPORT
PROBLEM NAME   = TESTQPQD
OBJECTIVE NAME = COST
RHS NAME       = RHS
BOUNDS NAME    = BND
MROW   =       6
NCOL   =       7
OBJECTIVE VALUE =    0.479124D+02
LOB/FIX OFFSET  =    0.239717D+02

COLUMNS.......STRUCTURAL VARIABLES

NO *=INF  NAME          VALUE        LOWER BND     UPPER BND    REDUCED COST
  1       X1             1.56267         .5         NONE          -3.0999
  2       X2             1.14578         .5         NONE          -3.29971
  3       X3             0.5             0.          1.           -2.60342
  4       X4             1.              1.          1.            2.09487
  5       X5             3.79156         0.          6.            0.
  6       X6             0.43733         0.         NONE           0.
  7       X7             1.47956         1.         NONE           0.

ROWS.........LOGICAL VARIABLES

NO *=INF  NAME          VALUE        LOWER RHS     UPPER RHS   SHADOW PRICE
    1     COST          42.87467       NONE         NONE           0.
    2     R1             2.             2.          2.           -3.
    3     R2             5.             5.          5.           -2.0999
    4     R3             4.             4.          4.           -2.30171
    5     R4            11.            11.         11.           -1.
    6     R5            14.            14.         14.           -1.
```

Listing (3): Solution output for *Q*-form in file '`TESTQP.QRS`'.

```
FORTMP release version 2.03a, Mar 1997
       begin
       model name (testqp)
       input file name (testqp.qdt)
       output file name (testqp.qrs)
       log file name (testqp.qlg)
 MPS INPUT PASS FINISHED
 INPUT DATA STATISTICS:
 PROBLEM NAME IS TESTQPQD
 NUMBER OF ROWS (INCLUDING OBJECTIVE)=           6
 NUMBER OF COLUMNS =                             7
 NUMBER OF BOUNDED VAR. =                        6
 NUMBER OF NONZEROS =                           25
 AVERAGE NUMBER OF NONZEROS PER COLUMN =         3
 NUMBER OF ROWS IN Q-MATRIX          =           4
 NUMBER OF NON-ZEROS IN Q-MATRIX     =           6
 RUN TIME OPTIONS:
 MINIMIZE
TIME TAKEN FOR INPUT/SETUP  =    0.93 SECS,  TOTAL SO FAR  =    0.93 SECS
SCALING IN PROGRESS ...
SCALING COMPLETE
PRESOLVE . . . . . .
Matrix non-zeros reduced to      21   from      25
TIME TAKEN FOR SCALE/PRSLVE =    0.11 SECS,  TOTAL SO FAR  =    1.04 SECS
TIME FACTORIZING Q:    0.164835
 Q MATRIX POSITIVE DEFINITE
Q cholesky factor nonzeros:      4
 SOLVING SEPARABLE FORM - NORMAL SYSTEM
TIME TAKEN FOR QPM PREPROC  =    0.38 SECS,  TOTAL SO FAR  =    1.43 SECS
------------------------------------------------------------
       Symmetric Matrix U Non-Zeros        28
       Cholesky  Factor U Non-Zeros        28
 IT# P-PHASE   P-INFEZ  D-PHASE   D-INFEZ      P-OBJVAL      RELGAP
   1     1   0.68801        1   0.10588        35.495        1.9557
   2     1   0.43483E-03    1   0.69612E-02    24.605       0.18322
   3     2   0.21806E-06    1   0.15224E-02    24.548       0.22496E-01
   4     2   0.10791E-09    2   0.22964E-04    24.539       0.10774E-02
   5     2   0.63483E-13    2   0.11492E-07    24.536       0.35992E-04
   6     2   0.56903E-15    2   0.57466E-11    24.535       0.12290E-05
   7     2   0.44371E-15    2   0.29388E-14    24.535       0.41985E-07
***IPMABC: Terminated, Small Duality Gap ****
***        Optimum solution obtained     ****
 *** IPM starting point saved ***
  Objective Function =  0.4853534E+02
            (offset = -0.2400000E+02)
  Elapsed structuring time =     0.164835
  Elapsed solution time    =     0.384615
  IPM iteration count =     6
TIME TAKEN FOR QPM PDSUBS   =    0.55 SECS,  TOTAL SO FAR  =    1.98 SECS
TIME TAKEN FOR IPM SOLUTION =    0.16 SECS,  TOTAL SO FAR  =    2.14 SECS
TIME TAKEN FOR OUTPUT       =    0.11 SECS,  TOTAL SO FAR  =    2.25 SECS
```

Listing (4): Log output for *Q* form in file '`TESTQP.QLG`'.

Back to Chapter contents

9.4.2 Worked example using FF^T form

The quadratic terms in the objective take the form

$$(1/2).(p_1^2 + p_2^2 + p_3^2)$$

Where the variables p_1, p_2 and p_3 are defined by equations:

$$p1 = x1$$
$$p2 = x1 + x2$$
$$p3 = x3 + x4$$

The expressions x_1, x_1+x_2, and x_3+x_4 are thus to be entered as F-type rows in the data.

Input data presented in a file '`TESTQP.FDT`' is shown below in listing (5). Listing (6) shows the associated SPECS commands in file '`FORTMP.SPC`'.

```
NAME          TESTQPFF
ROWS
 N  COST
 EQ R1
 EQ R2
 EQ R3
 EQ R4
 EQ R5
 F  F1
 F  F2
 F  F3
COLUMNS
    X1        COST                 4.0   R1                   1.0
    X1        R2                   1.0   R4                   1.0
    X1        R5                   1.0
    X1        F1                   1.0   F2                   1.0
    X2        COST                 6.0   R2                   3.0
    X2        R4                   1.0   R5                   2.0
    X2        F2                   1.0
    X3        COST                 5.0   R3                   2.0
    X3        R4                   1.0   R5                   2.0
    X3        F3                   1.0
    X4        COST                16.0   R3                   3.0
    X4        R4                   4.0   R5                   3.0
    X4        F3                   1.0
    X5        COST                 2.0   R4                   1.0
    X5        R5                   1.0
    X6        COST                 5.0   R1                   1.0
    X6        R5                   2.0
    X7        COST                 1.0   R5                   1.0
RHS
    RHS       R1                   2.0   R2                   5.0
    RHS       R3                   4.0   R4                  11.0
    RHS       R5                  14.0
BOUNDS
 LO BND       X1                   0.5
 LO BND       X2                   0.5
 UP BND       X3                   1.0
 FX BND       X4                   1.0
 UP BND       X5                   6.0
 LO BND       X7                   1.0
ENDATA
```

Listing (5): Input data using FF^T form in file 'TESTQP.FDT'

```
BEGIN
MODEL NAME (testqp)
INPUT FILE NAME (testqp.fdt)
OUTPUT FILE NAME (testqp.frs)
LOG FILE NAME (testqp.flg)
END
```

Listing (6): Specs commands for FF^T form in file 'FORTMP.SPC'.

Outputs comprising the solution in file '`TESTQP.FRS`' and the log in file '`TESTQP.FLG`' are shown below in listings (7) and (8).

```
FORTMP SOLUTION REPORT
PROBLEM NAME   = TESTQPFF
OBJECTIVE NAME = COST
RHS NAME       = RHS
BOUNDS NAME    = BND
MROW   =       9
NCOL   =      10
OBJECTIVE VALUE =    0.468204D+02
LOB/FIX OFFSET  =    0.220000D+02

COLUMNS.......STRUCTURAL VARIABLES

NO *=INF  NAME          VALUE        LOWER BND     UPPER BND    REDUCED COST
  1       X1             1.62899         .5         NONE            0.
  2       X2             1.12367         .5         NONE            0.
  3       X3             0.5             0.          1.             0.
  4       X4             1.              1.          1.             5.25
  5       X5             3.74734         0.          6.             0.
  6       X6             0.37101         0.         NONE            0.
  7       X7             1.63432         1.         NONE            0.
  8       Q 100001       1.62899       NONE         NONE           -0.9405
  9       Q 100002       2.75266       NONE         NONE           -1.58925
 10       Q 100003       1.5           NONE         NONE           -1.5

ROWS.........LOGICAL VARIABLES

NO *=INF  NAME          VALUE        LOWER RHS     UPPER RHS   SHADOW PRICE
  1       COST         42.74201       NONE         NONE            0.
  2       R1             2.             2.          2.            -3.
  3       R2             5.             5.          5.            -1.52975
  4       R3             4.             4.          4.            -1.75
  5       R4            11.            11.         11.            -1.
  6       R5            14.            14.         14.            -1.
  7       F1             0.              .           .             0.9405
  8       F2             0.              .           .             1.58925
  9       F3             0.              .           .             1.5
```

Listing (7): Solution output for FF^T form in file '`TESTQP.FRS`'.

```
FORTMP release version 2.03a, Mar 1997
       begin
       model name (testqp)
       input file name (testqp.fdt)
       output file name (testqp.frs)
       log file name (testqp.flg)
 MPS INPUT PASS FINISHED
 INPUT DATA STATISTICS:
 PROBLEM NAME IS TESTQPFF
 NUMBER OF ROWS (INCLUDING OBJECTIVE)=           9
 NUMBER OF COLUMNS =                            10
 NUMBER OF BOUNDED VAR. =                        6
 NUMBER OF NONZEROS =                           30
 AVERAGE NUMBER OF NONZEROS PER COLUMN =         3
 NUMBER OF F-TYPE ROWS               =           3
 RUN TIME OPTIONS:
 MINIMIZE
TIME TAKEN FOR INPUT/SETUP  =    0.38 SECS,  TOTAL SO FAR  =    0.38 SECS
SCALING IN PROGRESS ...
SCALING COMPLETE
PRESOLVE . . . . . .
Matrix non-zeros reduced to      28   from      33
TIME TAKEN FOR SCALE/PRSLVE =    0.05 SECS,  TOTAL SO FAR  =    0.44 SECS
TIME TAKEN FOR QPM PREPROC  =    0.00 SECS,  TOTAL SO FAR  =    0.44 SECS
------------------------------------------------------------
       Symmetric Matrix U Non-Zeros        28
       Cholesky  Factor U Non-Zeros        28
 IT# P-PHASE   P-INFEZ  D-PHASE   D-INFEZ      P-OBJVAL      RELGAP
   1     1   0.57416        1   0.12061        35.770        1.9567
   2     1   0.35614E-03    1   0.10059E-01    24.933       0.20156
   3     2   0.17924E-06    1   0.14675E-02    24.837       0.21546E-01
   4     2   0.88048E-10    1   0.26460E-03    24.825       0.20145E-02
   5     2   0.37038E-13    2   0.13203E-06    24.821       0.61993E-04
   6     2   0.15094E-15    2   0.66014E-10    24.820       0.21119E-05
   7     2   0.80189E-16    2   0.33094E-13    24.820       0.72143E-07
***IPMABC: Terminated, Small Duality Gap ****
***        Optimum solution obtained     ****
 *** IPM starting point saved ***
  Objective Function =  0.4682038E+02
            (offset = -0.2200000E+02)
  Elapsed structuring time =     0.329670
  Elapsed solution time    =     0.054945
  IPM iteration count =     6
TIME TAKEN FOR QPM PDSUBS   =    0.38 SECS,  TOTAL SO FAR  =    0.82 SECS
TIME TAKEN FOR IPM SOLUTION =    0.05 SECS,  TOTAL SO FAR  =    0.88 SECS
TIME TAKEN FOR OUTPUT       =    0.00 SECS,  TOTAL SO FAR  =    0.88 SECS
```

Listing (8): Log output for FF^T form in file '`TESTQP.FLG`'.

Back to Chapter contents

9.5 Branch and Bound Algorithm for MIQP

9.5.1 Sub-problem Solution

The Branch and Bound technique is no different in principle when a quadratic objective is added from the normal LP case (see chapter 6, section 6.4). Each sub-problem arising from the tree development is solved by using IPM-QP (as described above) rather than by using sparse simplex (Dual or Primal).

Since the addition of a fresh constraint at each branch must lead to a sub-node solution no better than its parent, the same bounding techniques apply to limit the tree development.

Back to Chapter contents

9.5.2 Simplified Tree

IPM solution requires considerably more memory than sparse simplex and this consideration leads us to consider a simplified Branch and Bound that itself uses less memory and can accommodate IPM for the sub-problems.

The two most important simplifications are:

- Discrete constraint-types limited to binary variables (which can be used for other constraint-types by special modelling)

- Tree development strategy limited to LIFO (Last In First Out).

The latter simplification means that node storage never exceeds the maximum tree-depth, which is usually much less than the number of binary variables because integer solutions (or infeasible nodes) appear well before all the binary variables have been fixed.

Back to Chapter contents

9.5.3 User Controls

The main user-control available for tree development is the following:

```
MIP PRIORITY UP ON            * Default OFF
```

(OFF may also be used). When 'ON' is specified the first sub-branch of each node to be developed is always the UP branch and variable selection considers the UP fraction of the candidate binary variables (see chapter 6, section 6.5.3). If the model contains many CLQ-type ($\Sigma x_j \leq 1$) or XOR-type ($\Sigma x_j = 1$) constraints - the x_j in each case being binary - then this option will have an important effect.

Other elements of the tree development strategy are fixed as follows:

Variable choice = 1 (minimum fraction)
First node choice = 1 (LIFO)
Second node choice = 1 (LIFO)

User-control for these elements may be added in a future release.

Control of the Bound and Cutoff can be obtained with the following commands:

```
MIP BOUND = v                  * Default high value
MIP CUTOFF TOLERANCE = v       * Default 1.0e-12
```

Which are described in Chapter 6, section 6.8.2.

The criterion for fractional part considered as zero in a binary variable is set with the following command:

```
INTEGER TOLERANCE = v          * Default 0.001
```

The level of log-messages issued may be controlled with:

```
MIP LOG LEVEL = n              * 0-4, default 1
```

It may also be convenient to set the log level for IPM to zero in order to prevent excessive output on the log.

Other outputs can also be obtained with the following commands:

```
MIP AGENDA OUTPUT ON           * Default OFF
MIP AGENDA OUTPUT ALL
MIP INTSOL OUTPUT ON
```

Agenda outputs have already been described - see Chapter 6, sections 6.6.3 and 6.6.5. However the corresponding input features are not available (as yet). Integer Solution output causes the complete set of primal solution values to be listed in the log whenever an integer solution is found.

Back to Chapter contents

9.6 Summary of SPECS Commands

9.6.1 Controls for IPM

There are no SPECS commands specific to IPM for QP. On reading in the data, FortMP automatically detects the problem is QP, and solves using predictor-corrector IPM. The following are all the IPM specific commands that are also applicable to QP. They have already been described in Chapter 5.

```
IPM PHI = v                          * Default is 10.0
```

This command sets the PHI control in the elementary formula for calculating *m*. See 5.2.1.

```
IPM POWER = n                        * n=0-3. Default is 4
```

This parameter sets the 'POWER' applied to the numerator in the more advanced formula for μ, which is used in the earlier phase of the predictor-corrector algorithm. See 5.2.1.

```
IPM DARE = v                         * Default is 0.9995
```

'DARE' is a fraction between 0 and 1 controlling how closely to approach the nearest boundary when moving from one interior point to the next. See 5.2.1.

```
IPM RELATIVE EPSILON = v             * Default is 1.0e-7
```

The relative epsilon is the tolerance within which the duality gap can be considered to be zero. Thus the optimum solution is reached provided the point is feasible. See 5.2.1.

```
IPM FEASIBILITY EPSILON = v              * Default is 1.0e-4
```

The feasibility epsilon is the tolerance governing feasibility of the current primal and dual

```
IPM STARTING POINT METHOD = n        * Default is 3
```

This command selects one of three starting point methods for the predictor–corrector algorithm.

```
IPM SOLVER CHOLESKY
IPM SOLVER SUPERNODE
IPM SOLVER XSUPERNODE
```

These commands select the solution mechanism to be used. The default is 'XSUPERNODE'. See 5.2.3.

```
IPM TOFIX = n                        * default = 1.0e-12
```

This command sets the criterion for minimum pivot size in the Cholesky factorisation. See 5.2.3.

```
CHOLESKY CG TOLERANCE = v            * default = 1.0e-4
CHOLESKY ERROR TOLERANCE = v             * default = 10.0
```

These commands set lower and upper levels to the solution error between which CG iterations are used to refine the major IPM iterations. See 5.2.4.

```
MAXIMUM CG ITERATIONS = n            * 0-3, default is 3
```

This command limits the number of CG steps taken at each major IPM iteration. See 5.2.4.

```
IPM RESTART ON
IPM RESTART OFF
```

This command specifies whether to 'RESTART' the IPM algorithm. Default is OFF. See 5.2.5.

```
IPM SAVE FREQUENCY = n                    * default n=10
```

This command specifies the frequency for making a SAVE in IPM. See 5.2.5.

```
MAXIMUM IPM ITERATIONS = n          * default 80
```

This command sets the termination limit for IPM. A SAVE is made before exit at termination. See 5.2.5.

```
IPM LOG LEVEL = n                   * 1-4, default is 1
```

This command specifies the 'level' of the output to be sent to the log file. See 5.2.7.

```
IPM GRAPHICAL DISPLAY ON
IPM GRAPHICAL DISPLAY OFF
```

Certain implementations of FortMP have the graphical feature, which displays the pattern of non-zeros in the matrices followed by a progress display of the iterations. The default is OFF. See 5.2.7.

Back to Chapter contents

9.6.2 Controls for Branch and Bound

Only certain of the MIP controls described in Chapter 6 are available in the simplified Branch and Bound (others may be added in a future release). The commands available are summarised below (see section 9.5.3 above for more details).

```
MIP PRIORITY UP ON              * Default OFF
MIP PRIORITY UP OFF
```

When 'ON' is specified the first sub-branch developed takes the UP direction (see chapter 6, section 6.5.3).

```
MIP BOUND = v                   * Default high value
MIP CUTOFF TOLERANCE = v        * Default 1.0e-12
```

These commands control Bound and Cutoff as described in Chapter 6, section 6.8.2.

The criterion for fractional part considered as zero in a binary variable is set with the following command:

```
INTEGER TOLERANCE = v           * Default 0.001
```

Sets the criterion for fractional part considered as zero in a binary variable.

```
MIP LOG LEVEL = n               * 0-4, default 1
```

Sets the level for messages to the log file.

```
MIP AGENDA OUTPUT ON            * Default OFF
MIP AGENDA OUTPUT ALL
MIP INTSOL OUTPUT ON
```

These commands control out of integer solutions. Agenda output is described in Chapter 6, sections 6.6.3 and 6.6.5. Integer Solution output causes the complete set of primal solution values to be listed in the log.

Back to Chapter contents

Chapter 10.

Advanced Starting Bases

Contents

10.1 Introduction

Two main kinds of solver for LP problems have been described and are available in the FortMP system:

- Simplex algorithms (Primal and Dual)
- Interior Point Method (IPM)

Both can obtain optimal solutions but only Simplex can obtain a basic, optimal solution - that is one with exactly *m* basic and *n* non-basic variables (see chapter 4, section 4.1).

Because post-optimal work (including MIP) always requires the solver to find a basic solution, Simplex is the preferred solver and occasions where the end user can be satisfied with an IPM solution are exceptional.

However, Simplex requires a basic solution before it even starts (of course not necessarily optimal or even feasible). One such is the UNIT basis obtained by fixing all structural variables to be non-basic at lower bound, and the logical variables are basic. The unit basis leaves Simplex with all the work still to do, and a procedure that creates a more advanced starting basis quickly may improve the overall solution time. Such a procedure may be termed 'CRASH'.

There are two kinds of CRASH: first the direct kind that starts from the unit basis and performs rapid basis exchanges without the usual Simplex paraphernalia. These methods are described below in section 10.2

Then there is the indirect kind, which requires that an initial set of solution values be obtained. With this solution to start from an iterative procedure is used to generate a basic solution without either increasing the infeasibility or degrading the objective value. The better the initial solution, the better is the overall effect in crashing. Iterating to obtain a basic solution is referred to variously as *CROSSOVER, BASREC* or sometimes *PUSH.* The method is described in section 10.3 below.

In a sense we may consider IPM followed by BASREC to be a highly advanced form of CRASH - particularly as the final solving with Simplex is usually completed without further iterations. Most of the time is spent in the IPM algorithm.

FortMP also includes a supplemental version that applies '*SUCCESSIVE OVER RELAXATION*' (SOR) to develop a set of solution values, and then by using the *CROSSOVER* technique creates a starting basis. This is termed *CRASH(SOR)* and is described below in section 10.4. SOR is not so effective as IPM for large problems, but nevertheless can produce an improvement over other methods in certain cases.

Back to Chapter contents

10.2 Primary CRASH Algorithms

10.2.1 Basic CRASH Procedure

Before starting any CRASH procedure an initial basis is always set up that comprises all the logicals, with the structural variables set to zero (i.e. set to lower bound or to zero in the case of a free variable). This is termed the 'Unit' basis. Then the CRASH procedure performs a succession of exchanges, each of which replaces a basic logical with a non-basic structural.

In any such exchange the pivot element - that is the intersection of row (logical) with column (structural) in the updated matrix must be non-zero. Updating the matrix in order to verify this requirement is time-consuming and therefore the large majority of pivots are chosen in a way that avoids any need for updating. This can be done if the column selected at every step has nothing but zeros on the pivotal rows of previous steps. The process is called triangular selection because it leads to a basis matrix that has a lower triangular form when the rows and columns are re-ordered in the sequence of their selection.

Further steps that require updates before a pivot can be chosen take place, if at all, after a maximum number of triangular pivots have been chosen. In any case, a final INVERT is performed as the quickest way to generate an Eta-file so that other SSX algorithms can follow.

Back to Chapter contents

10.2.2 CRASH(LTSF)

The 'LTSF' designation symbolizes "*Lower Triangular, Symbolic, designed for Feasibility*". These attributes may be described as follows:

Lower Triangular: As described above in 10.2.1.

Symbolic: Actual coefficient values are not relevant and Eta-creation is unnecessary thanks to the triangularity.

Feasibility design: Selection priorities are used to promote feasibility in the eventual basic solution.

In order to achieve a maximum of lower triangular selection, an over-riding priority for row-selection is given to sparsity - the row should be selected that has the fewest elements on those columns which are candidates for selection. Then a pivot column is selected and every other candidate column intersecting the selected row is ruled out as a pivot for later selection. It is also important to keep the basis as sparse as possible and therefore the column selection itself is similarly governed by sparsity.

In practice there are many ties for selection based solely on sparsity - both the row selection and the column selection. The way in which ties are broken has a major influence on the eventual feasibility. For row-selection secondary priority is given to EQ-type rows in order to remove artificials from the basis, and thereafter to other row-types according to the degree of restriction. Free rows of course are never selected. Ties in the column selection are

broken by a similar consideration of the feasibility-range; secondary priority being given to free columns and so on with fixed columns never selected in any case.

Studies have shown that a good measure of feasibility is achieved in most cases with fewer iterations needed in phase 1 of a following Primal solve-algorithm.

Back to Chapter contents

10.2.3 CRASH(ART)

This CRASH procedure is a follow-up to CRASH(LTSF) designed specifically to exclude artificials (that is logicals on EQ-type rows) from the basis, in so far as this is possible. Here of course the selection is no longer triangular or purely symbolic. Each column must be 'tried out' before selection and updated by the previous pivot-steps in order to ensure that the selected basis will be non-singular.

Priority for row-selection is absolute - only artificials can be selected. Column-selection is based on a heuristic designed to eliminate any obviously singular choices and secure the stability of the eventual inverse by choosing pivots of a reasonable size.

Back to Chapter contents

10.2.4 CRASH(ADG)

This CRASH procedure is designed to counter degeneracy in the eventual basis (Anti-degeneracy). It is a modification of CRASH(LTSF) in which the secondary criterion for row-selection is modified so as to promote a reduction in the degeneracy that often appears at the outset of the Primal algorithm.

Such degeneracy is simply the result of the initial basic solution having a large proportion of zero values. In order to overcome it priority must be given to selection of pivot rows where the solution value is non-zero. As a result the non-zeros are propagated elsewhere when the basis exchange is applied.

In order to apply this priority the RHS must be updated at each basis-exchange. However the procedure remains 'Symbolic' because actual values are not needed, only a set of markers to show where the non-zeros were originally and where fill-in took place.

CRASH(ADG) is controlled by the 'ADG Factor', which is a code expressing at what level the ADG-criterion is to take priority over the Feasibility-criterion in the row-selection. By default EQ-type rows take priority, then come the selected rows of the ADG-criterion and then come the other row-types. However, this can be changed by means of a SPECS command, or the ADG-criterion can be ignored altogether.

Back to Chapter contents

10.2.5 User Controls

Although CRASH(LTSF) has been described as purely symbolic, in practice this is not so because values can sometimes be important. In certain cases a basis can be chosen which, although triangular and hence strictly non-singular, nevertheless results in numerical

difficulty. If a large number of the selected pivots have small values then the application of the inverse can lead to a progressively expanding numerical error in the calculation.

For this reason a control is applied to prevent selection of any pivot whose size is less than a certain fraction of the maximum element size in that column. The user may set this fraction with the following SPECS command:

```
CRASH ADMIT THRESHOLD = v
```

Where the default value for 'v' is 0.001.

In order to control the use of CRASH(ART) after CRASH(LTSF) the following command may be used:

```
CRASH ART = n
```

Where 'n' is 0, 1 or 2. The significance of 'n' is:

0 = OFF. CRASH(ART) is not used.

1 = ON, using the normal heuristic for the selection of trial columns. This is the default

2 = ON, using an extended heuristic for the selection of trial columns.

Finally the use of CRASH(ADG) is controlled with the command:

```
CRASH ADG FACTOR = n
```

Where 'n' is 0, 1, 2 or 3. The significance of 'n' is:

0 = Lowest priority. The ADG-criterion is considered only as the final tie-breaker, behind any feasibility criterion. This is the default.

1 = ADG-criterion takes priority over selection of inequality constraints (LE and GE)

2 = ADG-criterion takes priority over selection of inequality and range-constraints.

3 = ADG-criterion takes priority over selection of any constraint-type. In this case the ADG-criterion entirely supersedes the feasibility criterion, taking second priority after sparsity.

Back to Chapter contents

10.2.6 Logged Output from CRASH

The normal logged output from CRASH (IPM algorithm not used), based on the simple example of chapter 7, section 7.2.5, is shown below:

```
  CRASH(LTSF) ENDED.  VARIABLE TYPES:-  PLUS    BNDD     FIX    FREE

     LOGICALS REMOVED FROM BASIS:-         1       2       1       0
     STRUCTURALS ENTERED IN BASIS:-        3       1       0       0
  CRASH(ART) ENDED:   1 PASSES:      0 ARTIFICIALS,       0 PIVOTED OUT
 TIME TAKEN FOR CRASHING    =     0.07 SECS,  TOTAL SO FAR =     0.33 SECS
```

Listing (1): Logged output from the CRASH

This shows that the LTSF-stage exchanged four logical variables in the initial UNIT basis for four structurals, and in the ART-stage no further exchanges were made.

A higher log level can be set with command '`SIMPLEX LOG LEVEL = 3`' and this will cause every individual exchange to be listed.

Back to Chapter contents

10.3 Crossover Algorithms: Purify and Basis Recovery

10.3.1 Introduction to Crossover

Both CRASH procedures and CROSSOVER procedures aim to produce an advanced starting basis so as to shorten the work needed by an SSX solver algorithm (Primal or Dual). In the case of CROSSOVER as opposed to CRASH there exists an initial solution, either provided by the user or obtained by other means, and this solution is employed in a way to ensure that none of the work done in pre-solving is lost.

For example consider the case of IPM. This algorithm provides a solution that is both feasible and optimal (to a certain degree of tolerance) but is not 'basic', which means that it cannot be used for further analysis or for integer programming. In the Crossover we develop a basis without either creating infeasibility or degrading the objective value.

An initial starting basis can be created by using the CRASH tools described above in 10.2.

This basis now leaves the solution in an anomalous situation because a certain number of the non-basic variables are neither equal to lower bound nor equal to upper bound. They are called 'Super-basic'. In the case of IPM which provides solutions to both the primal and the dual problems there exist both primal super-basic variables and dual super-basic variables. We now modify the basis with iterative 'Purify' algorithms, which eliminate the super-basic variables one by one.

There are two 'Purify' algorithms - also known as 'PUSH' algorithms as follows:

Primal Push which eliminates primal super-basics
Dual Push which eliminates dual super-basics

The Dual Push algorithm itself can start with a fast, crashing stage that makes exchanges without updating the Eta-file. This is referred to as *Dual Crash Push*.

If there is only a primal solution known at the outset then only the primal push can be employed.

In the case of IPM both primal and dual solutions are known and are additionally both feasible and optimal. As shown by Megiddo, who described the push algorithms, in such a case primal push, followed by dual push should yield a basic solution that is immediately an optimal solution not requiring any further iterations in the primal algorithm. The number of Push steps needed is no more than the number of super-basic values in each solution.

Back to Chapter contents

10.3.2 The Push Algorithms

In a Push iteration we evaluate the changes to solution values of basic variables resulting from a change to one of the super-basic variables. As the size of the change is allowed to grow one of two things may happen, either:

- A basic variable may reach a bound, or:

- The super-basic variable may reach its bound in the sense of its change.

If the latter event occurs first then it is only necessary to record all the changes and one super-basic variable has been rendered non-basic. If the former event occurs first then we make a change to the basis, replacing the basic with the super-basic and once again the number of super-basic variables is reduced by one. The overall procedure is the same for both primal and dual push with re-inversions occurring at the usual intervals.

Back to Chapter contents

10.3.3 The starting CRASH

The starting CRASH is modified for use before doing Purify. The changes are designed to meet the needs of the primal push algorithm that follows.

All those variables with a primal solution value equal to lower or upper bound are temporarily considered as fixed. Logical slacks at zero or at range are now considered as artificials with high priority for removal from the basis in the CRASH. Structurals at lower or at upper bound are fixed so as never to enter the basis.

In fact the row-selection priorities of the CRASH are altered so as to concentrate entirely on removal of artificials since otherwise much extra work will be needed by the dual push. Default selection-heuristic for CRASH(ART) is the advanced version since this improves the chances of removing artificials.

Variables fixed must of course be released once again when the primal push has terminated.

Back to Chapter contents

10.3.4 User Controls

The initial starting basis may be controlled with the following SPECS command:

```
CRASH ADMIT THRESHOLD = v        * Default v = 0.001
```

as before with the primary CRASH algorithm. CRASH(LTSF) is followed by an enlarged CRASH(ART) subject to pivoting controls as in the INVERT algorithm.

The Primal and Dual PUSH algorithms are subject to an 'ADMIT' threshold, which is the minimum super-basic value considered for applying a PUSH step. The following SPECS command may be used:

```
PUSH ADMIT THRESHOLD = v         * Default v = 1.0e-4
```

Another control command available is:

```
PUSH CALSOL FREQUENCY = n        * Default n = 10
```

Which specifies that a new solution must be calculated after 'n' re-inversions. Note that in default 'n' is 10 although in standard Primal and Dual re-calculating always takes place after every re-invert. However frequent re-calculation loses time and is very rarely needed.

It is also possible to skip over using the Dual push stage, or just its CRASH preliminary with the following commands:

```
DUAL PUSH OFF                    * Default ON
DUAL CRASH PUSH OFF              * Default ON
```

(ON may also be given). The reliability and efficiency of Dual Push has now greatly improved so that these commands should not be necessary.

Output of messages to the log during the Push algorithms is controlled with the following SPECS commands:

```
PUSH LOG LEVEL = n               * Default n = 0
PUSH LOG FREQUENCY = n           * Default n = 10
```

Where level 1 produces log messages only at each re-invert, level 2 at each 10th iteration or as specified by '`PUSH LOG FREQUENCY`'.

Back to Chapter contents

10.3.5 Logged Output from Crossover Algorithms

The following is a normal log from the crossover stage:

```
 *** No of direct P-push steps =     0
 *** No of direct D-push steps =     3
 CRASH(LTSF) ENDED.  VARIABLE TYPES:-  PLUS    BNDD     FIX    FREE
    LOGICALS REMOVED FROM BASIS:-         0       0     478       0
    STRUCTURALS ENTERED IN BASIS:-      478       0       0       0
 *** Time triangular =     0.020
 *** There are    478 initial selections,    478 triangular
 *** Time TRpivots =     0.000
         650 M-targets         172 P-targets
           0 C-tried             0 C-selected
 *** Time symbolic =     0.000
 CRASH(ART) ENDED:   1 PASSES:    172 ARTIFICIALS,       0 PIVOTED OUT
  *** Elapsed time in basrec crash =     0.05
FORREST-TOMLIN ACTIVATED
PRIMAL PUSH ENDED AT ITER#    112  PUSH COUNT =      10
  *** Elapsed time in primal push =     0.10
 DUAL C-PUSH ENDED:-     56 Exchanges,      3 Push-steps
  *** Elapsed time in dcr-push    =     0.05
DUAL PUSH ENDED AT ITER#    118  PUSH COUNT =       0
  *** Elapsed time in dual push   =     0.01
 FEASIBLE BASIS REACHED AFTER ITERATION    118
 Invert demand:  Obj = 5501.85      Suminf = 0.00000      ITER#    120
 STATUS = 3  -- OPTIMUM SOLUTION FOUND.     5501.85      ITER#    120
TIME TAKEN FOR BASREC      =     0.27 SECS,  TOTAL SO FAR =     3.79 SECS
```

Listing (2): Logged output from the CROSSOVER

It comprises:

- Two lines relating to 'direct' push-steps that can be made immediately without any change of basis (using the UNIT basis to start off).
- CRASH log: a normal CRASH(LTSF) log followed by CRASH(ART) details, the CRASH(ART) summary and total CRASH time.

- Primal Push log: in this case 122 steps of which 112 involved basis exchange and 10 were simple solution-updates
- Dual Crash Push log: 59 steps of which 56 involved basis exchange (implied only - there is no Eta-file update).
- Dual Push log: After re-invert a further 6 steps, all requiring basis exchange.
- Primal log: Final optimisation and completion of the basis recovery.

Back to Chapter contents

10.4 Iterative Crash Algorithm

10.4.1 Introduction to CRASH(SOR)

SOR is an iterative procedure designed to produce a good starting primal solution. Its parameters can be set to concentrate on reducing infeasibility only, or reducing infeasibility while improving the objective function value. Options exist for the user to limit the process and prevent it from consuming an excessive amount of time. For certain classes of problems, such as set covering, set packing and set partitioning problems, the SOR procedure can be very effective.

The iterative crash algorithm has three stages:

Stage I - an initial solution is calculated using CRASH(LTSF)

Stage II - a feasible non-basic solution is sought using an adapted Successive Over-Relaxation method **(SOR**).

Stage III - a basic solution is retrieved by applying the crossover procedure.

Back to Chapter contents

10.4.2 The iterative SOR procedure

The basic iterative procedure is to cycle through the constraints and if a constraint is violated, then the current solution vector x^k is modified as follows:

$$x^{k+1} = x^k + a\,\frac{b_i - a_i^T x^k}{a_i^T a_i}\,.\,a_i$$

Where α is a relaxation parameter and $a_i^T = (a_{i1}, \ldots, a_{in})$. This is an orthogonal projection of x^k onto the constraint $a_i^T x^k = b_i$. If $a = 1$, then $a_i^T x^{k+1} = b_i$. If $a < 1$ then $a_i^T x^{k+1} > b_i$ and the step is said to be under relaxed. If $a > 1$ then $a_i^T x^{k+1} < b_i$ and the step is said to be over relaxed. Typically $1 < a < 2$, hence the term Successive Over-Relaxation.

The objective function may be included as an additional constraint of the form

$$c^T x = z / b\,,$$

Where z is the objective function value and b is a 'reduction' parameter. This constraint is included if the option to improve the objective function value is chosen.

The starting point solution x^0 is calculated in the simplest way possible by using CRASH(LTSF) which provides a basis for Invert and then a starting solution is calculated in the usual way.

Back to Chapter contents

10.4.3 Crossover

Since there cannot be any dual push it will be most unlikely that the crossover can produce an immediate optimal basis as should be the case with a primal-dual optimum after IPM. However certain claims can be made.

In the first place, if the SOR solution is entirely feasible, the primal push algorithm will not generate any infeasibility. Infeasibilities in the basis may indeed grow but equally they may disappear and once removed cannot become infeasible again. If the non-basic infeasibilities are quite small there is a good chance they will disappear once the solution is reset at the end of the primal push.

In the second place the change-directions for the push-steps are taken so as never to degrade the objective value. Hence the objective will remain as good or better than the objective value for the SOR solution.

Back to Chapter contents

10.4.4 User Controls

The type of SOR algorithm to be used is controlled with the following commands:

```
SOR OBJECTIVE ON                    * Default OFF
SOR OBJECTIVE OFF
```

Where

OFF	means reduce infeasibilty only. This is the default.
ON	means reduce infeasibility while improving the objective function value.

If the option to improve the objective function value is chosen then the reduction parameter b is controlled with the following command:

```
SOR OBJECTIVE PARAMETER = v
```

Where the default value for 'v' is 1.1.

The SOR relaxation parameter a is controlled with the command:

```
SOR RELAX PARAMETER = v
```

Where the default value for 'v' is 1.5.

An SOR iteration is a complete cycle through all the constraints. The maximum number of iterations is controlled with the command:

```
SOR MAXIMUM ITERATIONS = n
```

Where the default value for 'n' is 2.

Feasibility of a constraint is governed by the violation tolerance. A constraint is considered violated if $|b_i - a_i^T x^k| >$ *violation tolerance*. The violation tolerance parameter is controlled with the command:

```
SOR VIOLATION TOLERANCE = v
```

Where the default value for 'v' is 1×10^{-7}.

An SOR solution is considered feasible if $\|Ax^{k+1}-b\|<$*feasibility tolerance.* The feasibility tolerance parameter is controlled with the command:

```
SOR FEASIBILITY TOLERANCE = v
```

Where the default value for 'v' is 1×10^{-3}.

Convergence of SOR is considered to have occurred when the infeasibilty of the current solution is not reduced significantly in the next iteration i.e. convergence is considered to have occurred if $||Ax^{k+1}-b|| - ||Ax^{k}-b|| <$ *converge tolerance*. The converge tolerance parameter is controlled with the command:

```
SOR CONVERGE TOLERANCE = v
```

Where the default value for 'v' is 1×10^{-2}.

Back to Chapter contents

10.4.5 Logged Output from the SOR algorithm

An example of the log messages listed by SOR is as follows:

```
TIME TAKEN FOR SCALE/PRSLVE=     0.24 SECS,  TOTAL SO FAR =     5.19 SECS
  CRASH(LTSF) ENDED.  VARIABLE TYPES:-  PLUS    BNDD     FIX    FREE
     LOGICALS REMOVED FROM BASIS:-         1       0     125       0
     STRUCTURALS ENTERED IN BASIS:-      125       1       0       0
 TIME TAKEN FOR CRASHT SOLN =     0.11 SECS,  TOTAL SO FAR =     5.30 SECS
 TIME TAKEN FOR KCR PREPROC =     0.05 SECS,  TOTAL SO FAR =     5.35 SECS
 CRASH(SOR):- CONVTOL .10000D-01
              TERMTOL .10000D-02
                 FREQ         10
  PUSH FOR FEASIBILITY ONLY
 CRASH(SOR):- ITERATIONS =      3
                  |Ax-b| =     0.8467392312D+01
                     OBJ =     0.1165835523D+05
 TIME TAKEN FOR K-CRASH ALG =     0.03 SECS,  TOTAL SO FAR =     5.38 SECS
 TIME TAKEN FOR KCR SOLUTION=     0.07 SECS,  TOTAL SO FAR =     5.45 SECS
```

Listing (3): Logged output from SOR

Back to Chapter contents

10.5 Summary of SPECS Commands

10.5.1 Primary Crash Commands

The following commands relate to a primary CRASH:

```
CRASH ADMIT THRESHOLD = v        * Default v = 0.001
```

This command applies a limit to admissible pivots in the LTSF-stage (relative to row-wise maxima).

In order to control the use of CRASH(ART) after CRASH(LTSF) the following command may be used:

```
CRASH ART = n                    * Default n = 1
```

Where ‘n’ is 0, 1 or 2. The significance of ‘n’ is:

0 = CRASH(ART) is not used.
1 = Normal heuristic
2 = Extended heuristic

```
CRASH ADG FACTOR = n
```

Where ‘n’ is 0, 1, 2 or 3 giving the priority to assign the ADG-criterion.

Back to Chapter contents

10.5.2 Crossover Commands

The following commands relate to a primary CROSSOVER:

```
CRASH ADMIT THRESHOLD = v        * Default v = 0.001
```

This command limits the admissible pivots in the CRASH(LTSF) stage (as in a primary CRASH).

```
PUSH ADMIT THRESHOLD = v         * Default v = 1.0e-4
```

This controls the minimum super-basic value considered for applying a PUSH step.

```
PUSH CALSOL FREQUENCY = n        * Default n = 10
```

This control specifies the frequency of re-calculating super-basic solutions, in number of re-inversions.

```
DUAL PUSH OFF                    * Default ON
DUAL PUSH ON
DUAL CRASH PUSH OFF              * Default ON
DUAL CRASH PUSH ON
```

OFF instructions cancel the Dual Push stages. However, these commands should not be necessary.

```
PUSH LOG LEVEL = n               * Default n = 0
```

```
PUSH LOG FREQUENCY = n          * Default n = 10
```

These commands control logged output during PUSH.

Back to Chapter contents

10.5.3 SOR Commands

The following commands relate to SOR:

```
SOR OBJECTIVE ON                * Default OFF
SOR OBJECTIVE OFF
```

These commands control whether SOR considers both objective and feasibility criteria.

```
SOR OBJECTIVE PARAMETER = v     * Default v = 1.1
```

Controls the reduction parameter b when objective criterion is considered.

```
SOR RELAX PARAMETER = v         * Default v = 1.5
```

Controls the relaxation parameter a.

```
SOR MAXIMUM ITERATIONS = n      * Default n = 2
```

Controls the number of complete cycles through all the constraints.

```
SOR VIOLATION TOLERANCE = v     * Default 1.0d-7
SOR FEASIBILITY TOLERANCE = v   * Default 1.0d-3
SOR CONVERGE TOLERANCE = v      * Default 1.0d-2
```

These commands control tolerances and limit the algorithm.

Back to Chapter contents

APPENDIX A: Input/Output Data Layouts

Contents

A1. MPS-form data layouts

A1.1 MPS Layout for LP and MIP problem data

In MPS format the data is divided into five sections as follows:

Section name	Purpose
ROWS	To specify the name and type of each constraint row in the problem. This includes the objective row to be minimised (or maximised).
COLUMNS	To specify the name of each variable and give the associated constraint and objective coefficients (only non-zeros need be entered).
RHS	To specify a name for the right hand side and give the RHS values.
RANGES	To specify a name for any range-set and give RHS range values. If there are no RHS ranges in the data then this section is omitted.
BOUNDS	To specify a name for any bound-set and give the lower and upper bounds of the variables. The BOUNDS section is also used to specify binary and integer variables for MIP. If there are no bounds in the data then this section is omitted.

These five sections appear in this order in the input data, each section headed by an indicator record which consists of section name beginning in position 1.

The data-file begins with a 'NAME' record and ends with an 'ENDATA' record so that the complete layout is as follows:

- NAME record
- ROWS header record
- ROWS data section
- COLUMNS header record
- COLUMNS data section
- RHS header record
- RHS data section
- RANGES header record
- RANGES data section
- BOUNDS header record
- BOUNDS data section
- ENDATA record

In the optional RANGES and BOUNDS sections the header may be omitted if there is no data to follow. Note however that the RHS header record must appear even if there is no RHS data.

Comment lines bearing an asterisk (*) in position 1 may appear anywhere and will be ignored.

Data records in all five sections have a common fixed layout of six fields as follows:

Field	Positions	Contents
1	2-3	A type code (depending on the section)
2	5-12	A name
3	15-22	A name
4	25-36	Value corresponding to the name in field 3
5	40-47	A name of the same type as field 3
6	50-61	Value corresponding to the name in field 5

The contents of each field in each section is described below.

ROWS section:-

Field	Entry	Meaning
1	GE, G, >= or >	Greater than or equal to
	LE, L, <= or <	Less than or equal to
	EQ, E or =	Equal to
	N	No constraint (i.e. free row or objective)
2	Row name	The name of the row

Fields 3, 4 and 5 are not used

A row name may contain any characters including blanks after the leading character. Leading blanks are ignored, however the user is always advised to enter names left-justified to avoid confusion. Note that the name 'R1 11' is different from the name 'R 111' for example.

COLUMNS section:-

Field	Contents
1	Blank
2	Column name
3	Row name
4	Value on the column/row given on fields 2 and 3
5	Row name (optional)
6	Value on the column/row given in fields 2 and 5 (blank if field 5 is not used)

Fields 5 and 6 may be omitted.

Column names have the same description as row names given above.

Values must be entered at the right in fields 4 and 6 unless the decimal point is coded (any trailing blanks are treated as zeros).

All the data for any one column must be collected together (including the objective row) and appear on consecutive records. The order of rows within a column is insignificant, however the same row should not appear more than once.

RHS section:-

Field	Contents
1	Blank
2	RHS set name
3	Row name
4	Value on the RHS/row given on fields 2 and 3
5	Row name (optional)

6	Value on the RHS/row given in fields 2 and 5 (blank if field 5 is not used)

Fields 5 and 6 may be omitted.

The user may provide for alternative problems in one data-file by including multiple RHS-sets. By default the system selects only the first RHS-set and ignores the remainder. RHS-set names have the same description and values are entered in the same way as given above.

All the data for any one RHS-set must be collected together and appear on consecutive records. The order of rows within an RHS-set is insignificant, however the same row should not appear more than once.

An empty RHS-set (values all zero) can be built with one record assigning the RHS-set name and specifying zero on some row. This is not necessary if the RHS-set is to be the only RHS-set provided.

RANGES section:-

Field	Contents
1	Blank
2	RANGE set name
3	Row name
4	Value on the RANGE/row given on fields 2 and 3
5	Row name (optional)
6	Value on the RANGE/row given in fields 2 and 5 (blank if field 5 is not used)

Fields 5 and 6 may be omitted.

Range values apply in a manner dependant on the row-type to form lower and upper bounds on the LHS expression $(Sa_{ij}x_j)$ as follows:

For an LE-type constraint:

RHS-RANGE <= $Sa_{ij}x_j$ <= RHS

For a GE-type constraint:

RHS+RANGE >= $Sa_{ij}x_j$ >= RHS

For an EQ-type constraint with positive range-value:

RHS+RANGE >= $Sa_{ij}x_j$ >= RHS

For an EQ-type constraint with negative range-value:

RHS+RANGE <= $Sa_{ij}x_j$ <= RHS

No other possibilities exist. The range-value for a GE-type or LE-type constraint must be positive or zero - if range value is zero then the constraint becomes an equality.

The user may provide for alternative problems in one data-file by including multiple RANGE-sets. By default the system selects only the first RANGE-set and ignores the remainder. RANGE-set names have the same description and values are entered in the same way as given above.

All the data for any one RANGE-set must be collected together and appear on consecutive records. The order of rows within a RANGE-set is insignificant, however the same row should not appear more than once.

There is no such thing as an empty RANGE-set. Zero values, indicating an equality constraint, must be explicitly entered.

BOUNDS section:-

Field	Contents
1	Bound-type code as specified below
2	Bound set name
3	Column name
4	Value of the bound whenever relevant

Fields 5 and 6 are not used

The bounds section is used not only to specify bounds on continuous variables, but also to specify binary and integer variable types in MIP.

Each variable is given a default type of PL, meaning a continuous variable with upper bound plus infinity and lower bound zero. This is modified by bounds data as follows:-

Type code	Bound description
UP	Upper bound - value is given in field 4
LO	Lower bound - value is given in field 4
FX	Fixed value - upper and lower bound both equal and given in field 4

FR	Free variable - upper bound plus infinity and lower bound minus infinity.
PL	Plus-type (this is allowed but redundant)
MI	Minus-type - lower bound minus infinity
BV	Binary variable. Only legal values are zero and one.
UI	Integer variable with upper bound given by the value in field 4
LI	Integer variable with lower bound given by the value in field 4
SC	Semi-continuous variable. Either zero or lies in the range from 1.0 to the value given in field 4.

A variable may require two records to specify it in full and the possible combinations allowed for the same variable are:-

Combine	Meaning
LO and UP	Bounded variable
LO and PL	Plus-type variable with given lower bound
MI and UP	Minus-type variable with given upper bound
LI and UI	Integer variable with lower and upper bounds both given.
LO and UI	Same as LI/UI (not recommended)
LI and UP	Same as LI/UI (not recommended)
LI and PL	Same as LI on its own (no upper bound)
MI and UI	Integer variable with upper bound and no lower bound

The user may provide for alternative problems in one data-file by including multiple BOUND-sets. By default the system selects only the first BOUND-set and ignores the remainder. BOUND-set names have the same description and values are entered in the same way as given above.

All the data for any one BOUND-set must be collected together and appear on consecutive records. The order of columns within a BOUND-set is insignificant, however the same column should not appear more than once except in one of the legal combinations given above.

NAME record:-

The NAME record at the beginning of the data-file has the following layout:

Position	Contents
1-4	NAME
15-22 (field 3)	Problem-name which may comprise any characters.

ENDATA record:-

The ENDATA record at the end of the data-file has the following layout:

Position	Contents
1-6	ENDATA

The proformas given on the following pages are intended as a convenience for the user to copy and use as data-entry forms.

ROWS proforma

Field 1 2-3	Field 2 5-12	Field 3 15-22	Field 4 26-36	Field 5 40-47	Field 6 50-61

Field 1, Row-type:- LE = Less than or Equal GE = Greater than or Equal
EQ = Equal N = Non-binding (objective)

Field 2, Row name

COLUMNS proforma

Field 1 2-3	Field 2 5-12	Field 3 15-22	Field 4 26-36	Field 5 40-47	Field 6 50-61

Field 2, Column name	
Field 3, Row name	Field 4, Value
Field 5, Row name	Field 6, Value

RHS proforma

Field 1 2-3	Field 2 5-12	Field 3 15-22	Field 4 26-36	Field 5 40-47	Field 6 50-61

Field 2, RHS-set name
Field 3, Row name Field 4, Value
Field 5, Row name Field 6, Value

RANGES proforma

Field 1 2-3	Field 2 5-12	Field 3 15-22	Field 4 26-36	Field 5 40-47	Field 6 50-61

Field 2, RANGE-set name
Field 3, Row name Field 4, Value
Field 5, Row name Field 6, Value

BOUNDS proforma

Field 1 2-3	Field 2 5-12	Field 3 15-22	Field 4 26-36	Field 5 40-47	Field 6 50-61

Field 1, Bound type code

LO = Lower bound
FX = Fixed value
MI = Minus-type
LI = Integer variable, lower bound
UP = Upper bound
FR = Free variable
PL = Plus-type (default)
UI = Integer variable, upper bound

BV = Binary variable SC = Semi-continuous, upper bound

Field 2, Bound-set name
Field 3, Column name Field 4, Value

Back to Chapter contents

A1.2 Marker data for Binary, Integer and SOS specification

Marker lines are used in the COLUMNS section of the MPS-form input data to define a list of consecutively sequenced variables. Four types of marker line can be used as follows:

Marker	Meaning
INTORG	Beginning of a consecutive sequence of Integer or Binary variables
INTEND	End of a consecutive sequence of Integer or Binary variables
SOSORG	Beginning of a special ordered set (type SOS1 or SOS2)
SOSEND	End of a special ordered set (type SOS1 or SOS2)

A marker line has the keyword 'MARKER' (quotes included) in Field 3 of the standard MPS-form layout and the type keyword 'xxxORG' or 'xxxEND' either in Field 4 or in Field 5 (quotes included).

The full layout is as follows:

Field 1 (2-3)	Field 2 (5-12)	Field 3 (15-22)	Field 4 (25-32)	Field 5 (40-47)	Field 6 (50-61)
blank	Label	'MARKER'	'INTORG'	blank	blank
blank	Label	'MARKER'	'INTEND'	blank	blank
SOS type	Label	'MARKER'	'SOSORG'	REF-row	blank
blank	Label	'MARKER'	'SOSEND'	blank	blank

Each marker line must appear between one column and another - a column may not be split by a marker.

It is not allowed to overlap sections of the data with marker lines since all sets must be mutually exclusive. Thus markers are always to be in pairs with the 'ORG' type marker followed by its corresponding 'END' type marker before any other marker can appear.

Fields 4 and 5 are interchangeable - the marker type keyword may be placed in field 5 and when 'SOSORG' is placed in field 5 the 'REF-row' data is placed in field 4.

Blank fields should be left blank (but may contain unused material).

The 'Label' field is intended for the user to attach a name or label to a set. For this purpose the 'END' marker should have the same label as its corresponding 'ORG' marker. However this is quite optional.

'SOS type' in Field 1 is either 'S1' or 'S2' meaning type SOS1 or type SOS2 respectively

'REF-row' in field 5 (or field 4) of a SOSORG-type marker line is optional. If left blank then a default row with coefficients 1,2,3,... is assumed.

Further information on defining binary or integer variables with markers is given in chapter 6, section 6.3.3.

Back to Chapter contents

A1.3 MPS-format of external BASIS data (input or output)

In this format the input basis data comprises:

NAME record
BASIS header record (optional)
BASIS data section
ENDATA record

In the BASIS data section the layout employs fields as before described as follows:-

Field 1 Code for the type of record as follows:
XL, XU introducing a basic variable
UL, LL introducing a non-basic variable
Field 2 Name of a structural variable which becomes basic for codes XL, XU or remains non-basic for codes UL, LL.
Field 3 Name of a logical variable which becomes non-basic for codes XL, XU. Not used for codes UL, LL

The letter U or L in the code indicates whether the non-basic variable is to receive upper bound or lower bound status.

The rationale behind this layout is that an initial starting basis is defined and each record defines an exchange to this basis. Initially all row variables are basic and all column variables are non-basic at lower bound. Each record of type XL or XU defines an exchange between a column variable (structural) which becomes basic and a row variable (logical) which becomes non-basic. The UL records simply name non-basic columns which are to receive at-bound status and LL records are actually redundant.

An external basis is intended for use even when the input problem has been revised so that a variety of inconsistencies in MPS format basis data are allowed. For example if a name is not recognised then a warning message is given on the log but the system carries on by simply ignoring that record.

Back to Chapter contents

A2. Free-form Layout and Long Names

A2.1 Free-form input layout

The following SPECS commands:

```
INPUT FREE FORMAT ON
INPUT LONG NAMES ON
```

are both associated with Free-format input. Free-format input is based upon and is very similar to MPS-form input with just one difference that there is no fixed field layout beyond position 5 in free-form data lines.

A Free-form data line consists of the fixed-form MPS field 1 in positions 2-3 and from positions 5 onwards a number of 'tokens' separated by one or more blank spaces, where a 'token' is a string of 1 to 16 characters that are not blank. For each type of MPS-form data-line there is a precise free-form equivalent with blank-separated tokens replacing the fixed fields from positions 5-62 in the same order from left to right. Indeed an MPS-form data file can perfectly well be read in as free-form data, provided that no names are blank or have interior embedded blanks, and provided that no material exists in the unused part of the fixed layout. For 'MARKER' lines the label is treated as a name and must be non-blank.

Free-form data is sectioned by header lines:

ROWS
COLUMNS
RHS
RANGES
BOUNDS

with the keyword beginning in position 1, and the last section is terminated by:

ENDATA

again beginning in position 1.

Any line beginning with '*' in position 1 is treated as commentary and ignored.

Apart from comment-lines the first line in the free-form input data must have 2 tokens:

NAME	beginning in position 1
PROBLEM	Problem name located freely

The remaining data lines within each section have positions 1-4 fixed exactly as in MPS-form, that is to say with a type-code (if any) in positions 2-3 and blank otherwise. The first name in the line begins at or after position 5, optionally preceded by one or more blanks.

Since a token must inherently obey the following:

- First character is non-blank
- There are no embedded blanks

these are rules that must apply to all names - that is to:

- Row names
- Column names
- RHS-set names
- RANGE-set names
- BOUND-set names
- Problem name
- Label in a 'marker' line

Free-form layouts are summarised in the following tables:

Field	Token no	ROWS section	COLUMNS section	MARKER line
2-3		Row type		SOS-type
5-end	1	Row name	Column name	Label
	2		Row name 1	**'MARKER'**
	3		Value 1	Marker type in quotes ('xxxORG', 'xxxEND')
	4		Row name 2	Ref-row name
	5		Value 2	

Field	Token no	RHS section	RANGES section	BOUNDS section
2-3				Bound-type code
5-end	1	RHS-set name	RANGE-set name	BOUND-set name
	2	Row name 1	Row name 1	Column name
	3	Value 1	Value 1	Value

	4	Row name 2	Row name 2	
	5	Value 2	Value 2	

where the details of type-codes and other meanings, options etc are exactly as described for corresponding fields in MPS-form data - see Appendix A1 above.

In a MARKER line the SOS-type in positions 2-3 and the reference-row in token 4 apply only to marker type SOSORG and are omitted for other marker-types. Token 4 may also be omitted for the default reference-row. It is also permitted to switch the order of tokens 3 and 4, putting a dummy as token 3 if the REF-row name is not given.

Back to Chapter contents

A2.2 Name length

Long names, up to 16 characters, can be employed with the SPECS command:

INPUT LONG NAMES ON

This applies to Row names and Column names only, not to RHS-set, RANGE-set, Bound-set and Problem names which remain of maximum length 8 characters.

If long names apply then the form of input layout used is free-format - there is no fixed layout for long-name input. Therefore the rules given above in section A2.1 for free-format input necessarily apply.

Back to Chapter contents

A3. Tabular Layouts (MG/RW interface)

A3.1 Tabular input layout

Tabular input is an alternative to the normal MPS-form of input layout in which the rows and columns are not identified by name but rather by index. This layout is invoked with the SPECS-command 'INPUT TYPE MG'

Problem data is presented on the standard input channel (file name 'model.mps') in a FORTRAN formatted file which is described below. The data is essentially the same as for the internally callable interface - see chapter 6, section 6.2. In this description the names used are those of section 6.2.3 which gives a more detailed description.

Data is organised in sections as follows:-

- Header section
- Coefficient section
- Columns section
- Rows section
- Special ordered set section

Each section is described below.

Input Header Section

The header section comprises two records as follows:-

Record 1 holds the problem name - PNAME - in positions 1 to 8

Record 2 holds the following

Name	Description	Format
NC	Number of columns (structural variables)	I10
MR	Number of rows (logical variables)	I10
NAIJ	Number of non-zero coefficients in the matrix	I10
NSET	Number of special ordered sets	I10

If there are no special ordered sets then NSET must be zero.

Input Coefficient Section

The coefficient section contains NAIJ records (i = 1,2,...,NAIJ) as follows:-

Name	Description	Format
COLIN(i)	Column number	I10
ROWIN(i)	Row number	I10
AIJ(i)	Coefficient	3X,D20.13

The sequence of data presentation is not material. Although normally to be presented either column-wise (as in MPS format) or row-wise, the system accepts any sequence and re-orders the data internally.

The data is subject to consistency conditions as in MPS format:

- Every column must contain at least one entry.
  There may not be any duplicate entries

Input Columns section

One record for each structural variable (j = 1,2,...,NC) as follows:-

Name	Description	Format
j	Column number	I10
LOB(j)	Lower bound	3X,D20.13
UPB(j)	Upper bound	3X,D20.13
COST(j)	Cost (objective value)	3X,D20.13
MITYP(j)	Mixed Integer type code as follows: 0 = Continuous variable 1 = Binary variable 2 = Integer variable	I10

	3 = Semi-continuous 4 = Member of SOS1 5 = Member of SOS2	

Input Rows section

One record for each logical variables (i = 1,2,...,MR) as follows:-

Name	Description	Format
i	Row number	I10
LHS(i)	Left hand side, or lower bound of the expression	3X,D20.13
RHS(i)	Right hand side, or upper bound of the expression	3X,D20.13

Input Special Ordered Set section

If the model has special ordered sets then a further section is added with one record for each set (i = 1,2,...,NSET) as follows:

Name	Description	Format
i	Set number	I10
SREF(i)	Reference row number	I10
SFUN(i)	Function row number	I10
SBEG(i)	Column number of 1st member	I10
SEND(i)	Column number of last member	I10

The type of each SOS is indicated by the corresponding entries in array MITYP.

A reference row and a function row must exist for each set and sets may not overlap each other.

Back to Chapter contents

A3.2 Tabular output layouts

Tabular output is an alternative to the standard form of output layout in which the rows and columns are not identified by name but rather by index. This layout is invoked with the SPECS-command 'OUTPUT TYPE RW'

Problem data is written to the standard output channel (file name 'model.res') in a FORTRAN formatted file which is described below. The data written is essentially unchanged.

Data is organised in sections as follows:-

- Header section
- Columns section
- Rows section

Each section is described below.

Output Header Section

The header section comprises one record as follows:-

Name	Description	Format
PROBNM	Problem name	A8
MROW	Number of rows	I8
NCOL	Number of columns	I8
OBJVAL	Objective value	E14.6
STCODE	A 3-character code for the solution status. Values are given below.	2X,A3

Possible values for the code STCODE are as follows:

'OPT' indicates the LP optimum solution
'INF' indicates there is no feasible solution (the output solution is infeasible)
'UNB' indicates that solutions exist with unbounded objective value.
'***' This code is given for any other case

Output Columns section

One record for each structural variable (j = 1,2,...,NCOL) as follows:-

Name	Description	Format
JCOL	'X' followed by the column number	A8
STAT	A 1-character status code for the column. Values are given below.	2X,A1
XVAL	Solution value	E14.6
DJ	Reduced cost (dual solution value)	E14.6
LOB	Lower bound	E14.6
UPB	Upper bound	E14.6
MARK	A 1-character marker for infeasibility	2X,A1

The column number JCOL is written with full leading zeros after the 'X'. Example: column 1 is written 'X0000001'

Possible values for the status code STAT are:

'B' Indicates a basic variable
'L' Variable is at lower bound
'U' Variable is at upper bound
'F' The variable has a fixed value (this code overrides any possible alternatives)

Possible values for the feasibility marker MARK are:

'F' Feasible in both the primal and the dual
'P' Primal infeasible
'D' Dual infeasible
'B' Infeasible in both the primal and the dual

Output Rows section

One record for each logical variable (i = 1,2,...,MROW) as follows:-

Name	Description	Format
ICOL	'R' followed by the row number	A8

STAT	A 1-character status code for the row. See description below.	2X,A1
RVAL	Value of the row expression ΣAijXj.	E14.6
SP	Shadow price (dual solution value)	E14.6
LRHS	Lower bound to the RHS	E14.6
URHS	Upper bound to the RHS	E14.6
MARK	A 1-character marker for infeasibility	2X,A1

The row number JCOL is written with full leading zeros after the 'R'. Example: column 1 is written 'R0000001'.

The solution value RVAL represents the value of the row-expression ΣAijXj and not the value of the logical variable. This is according to the usage of the standard output layout and the values LRHS and URHS give the lower and upper bounds of this expression.

Possible values for the status code STAT are the same as for columns given above. Note that the status refers to the logical variable (or slack) and in non-basic situations is opposite to the status of the row-expression.

Possible values for the feasibility marker MARK are the same as for columns.

Back to Chapter contents

A4. MIP AGENDA Layouts

A4.1 Standard, named agenda layout

MIP agenda is divided in two sections as follows:

Section name	Purpose
FIX (FIXMIX or FIXTRY)	To specify integer-feasible values in order to obtain an initial integer solution rapidly
PRIORITY	To specify priority classes for selection of branch-entity

Either or both sections may appear in either order (once only). The FIX section follows a header FIXMIX or FIXTRY according to whether to create tree nodes normally during FIX processing or simply to probe for the indicated solution. The PRIORITY section follows a header - PRIORITY.

The following keywords are used to begin and end the file

AGENDA
ENDATA

Comment lines beginning with '*' (asterisk) in position 1 may appear anywhere and will be ignored.

The fields used are the same as those of MPS-format, or free-format with either of the SPECS commands:

INPUT FREE FORMAT ON
INPUT LONG NAMES ON

The layout of both sections is the same, as follows:

Field no (MPS-form)	Token no (Free-form)	Contents
2 (5-12)	1	Name of 1st column to be assigned
3 (15-22)	2	Name of last column to be assigned
4 (25-36)	3	Value to assign

The default value zero is assigned to every discrete variable before assignment begins. It is permitted to overwrite one assignment with a later one in the same data section.

Standard agenda data is input/output via disk-file named:

modname.agn

where 'modname' is the model nmae.

Back to Chapter contents

A4.2 Listed, un-named agenda layout

Listed agenda data employs the same overall structure as standard agenda data with the same keywords as before:

AGENDA
FIXMIX
FIXTRY
PRIORITY
ENDATA

In addition the same default value zero is used for every assignment and assignments can be overwritten as before. The data layout is fixed as follows:

Field	Contents
1-10	Index of 1st column to be assigned
11-20	Index of last column to be assigned
21-30	Integer value to assign

with data right-justified in each field.

The input/output file name is as follows:

modname.agl

where 'modname' is the model name as before.

Note that the use of standard agenda data causes equivalent listed form agenda to be created internally as scratch files.

Back to Chapter contents

APPENDIX B: SPECS Commands

This appendix lists all the SPECS commands available in the current version (V2.3) of FORTMP. Those commands which have already been specified in this manual are listed with reference to the chapter/section concerned. Other commands are described and the general syntax is given in full.

Contents

B1. Syntax

B1.1 Meta-syntax

In what follows the syntax of a command is indicated in bold lettering.

Keywords are given in capitals (exceptionally in lower case with duplicate commands), user provided information is given in lower case. An integer value is indicated by the conventions 'n', 'nn', 'nnn'. Real values are indicated by the convention 'vvv'.

Square brackets are used for an optional part of the syntax, for example 'MODEL [NAME] (name)' indicates a command that may be written 'MODEL (name)'. Note that ordinary parentheses are a part of the syntax and may not be omitted.

Angle brackets are used for alternatives, separated by a slash, for example 'INPUT TYPE <MPS/FREE/MG/MPL>' implies four possible alternative commands 'INPUT TYPE MPS', 'INPUT TYPE FREE', 'INPUT TYPE MG' and 'INPUT TYPE MPL'.

Extra blank spacing may be added before or between syntactic components. Blank spacing must not be embedded within a component.

Back to Chapter contents

B1.2 Simplification and abbreviation

It is possible to omit the SPECS file altogether when all commands are to be assigned to their defaults. Input problem data is in MPS format on the file named 'model.mps'.

Any command may be omitted and in such a case the system assumes a default setting.

All keywords may be abbreviated to the first 4 or more letters. The first keyword (only) may be abbreviated to the first 3 letters so long as no ambiguity arises. For example 'MAXIMIZE' could be written as, 'MAXIM', 'MAXI' or as 'MAX'. No confusion arises because 'MAXIMUM' is always followed by a further keyword.

The syntax '<ON/OFF>' may be omitted in all cases signifying 'ON'. In principal it should be given as '[<ON/OFF>]' but for simplicity this has not been done. For example the command 'SCALE' is the same as 'SCALE ON' and 'CRASH' is the same as 'CRASH ON' although these two commands have opposite defaults.

Back to Chapter contents

B1.3 BEGIN and END commands

The SPECS command file 'fortmp.spc' is split into *sections* permitting different commands to apply in multiple calls to the solver (e.g. via SUBMP1). A section is delimited by the following two lines:

```
BEGIN [(sectid)]

END
```

where `'sectid'` is a string of up to 8 characters, blank if omitted, that gives an identifier to each section. 'END' is not necessarily the end of the file but merely terminates the section, and any lines between 'END' and the next 'BEGIN' (or end of file) are ignored.

In a stand-alone FortMP execution there is usually only one relevant section, and this starts at the first 'BEGIN' line on the file, whatever section-id (if any) is given for it. However the special name 'ALL' - command `'BEGIN (ALL)'` - can change this, as explained below.

In a run where FortMP is used as a callable sub-system (e.g. with SUBMP1) the several invocations may well require different SPECS so that two or more lists may be needed. The list to be used for one call is selected by the 'SPID' parameter - see chapter 7, section 7.1.2. The following special values for 'SPID' should be noted:

SPID	Meaning
Blank	Select the first section. If the first section is 'ALL' then select the first and second sections.
`'NOSPECS'`	Bypass SPECS-command input altogether. All controls remain at default settings.
`'DEFAULT'`	Has a special meaning and should never be used for SPID
`'ALL'`	Has a special meaning and should never be used for SPID

Note that if a section named 'ALL' is to be used it should come first on the SPECS file. After reading and interpreting this section the system continues to search for another section corresponding to the SPID parameter. Note also that in a stand-alone FortMP execution the parameter SPID is blank.

If the command `'CALL SPCINT(TCTN)'` (see section 7.4.2) is used so as to load the SPECS file for internal communication each time the callable sub-system is invoked, then a section with the name 'DEFAULT' is interpreted specially. It is not stored as a section to be communicated but rather it modifies the actual control defaults on the spot. The effect is the same as for an 'ALL' section but 'DEFAULT' commands are applied even when the string `'NOSPECS'` is used for SPID. 'DEFAULT' commands are not listed in the log. 'DEFAULT' as a section name has no special meaning for a stand-alone execution, nor if `'CALL SPCINT(TCTN)'` is not used to load the SPECS file.

Back to Chapter contents

B1.4 Commentary

The character asterisk (*) may be used to terminate the command on any line. Material that follows is ignored and can be used as commentary for that command.

Any line with asterisk (*) as the first non-blank character is ignored. This provides a means to cancel a command without removing it from the SPECS file.

Another way to introduce commentary is to place the 'END' command after the last active command and all lines following 'END' are ignored.

Back to Chapter contents

B2. Command descriptions

In the tables that follow the range of possible values is given for each command, together with the coding and default value. This is intended as a reference for the user to find the syntax of any command quickly. The tables also give the reference in this manual to the chapter and section where each command is specified and references to other relevant sections.

In addition to commands already described in the manual there are additional commands for specialised purposes. The description of these commands is given with manual reference to 'Appendix'.

Also included in the tables are several commands shown in lower case lettering. These are alternative forms for other commands (mentioned in upper case above the lower-case command) which have been retained for compatibility with earlier versions of FortMP.

Back to Chapter contents

B2.1 Model name and data file names

Command	Manual ref	Lowest value	Highest Value	Default value
MODEL NAME (modnam)	3.6			'model'
INPUT FILE NAME (fnam)	3.6			' .mps'
OUTPUT FILE NAME (fnam)	3.6			' .res'
LOG FILE NAME (fnam)	3.6			' .log'
BASIS FILE NAME (fnam)	3.6			' .bsf'
BBASIS FILE NAME (fnam)	Below			' .bbf'
AGENDA FILE NAME (fnam)	Below			' .agn'

DIRECTORY NAME (pthnam) **PATH NAME (pthnam)**	3.6			blank

Default for all file names is initially blank and unless entered by command is then set to 'modname.xtn' where:

modname is the model name

xtn is a 3-character extension

The following additional commands are available:

AGENDA FILE NAME (fnam)
BBASIS FILE NAME (fnam)

These commands assign names to the corresponding input files (the output files always receive the default file-name).

Back to Chapter contents

B2.2 Input type and problem selection

Command	Manual ref	Lowest value	Highest value	Default value
MAXIMIZE	4.6.1	0=MIN	1=MAX	0 (MIN)
MINIMIZE	4.6.1			
INPUT TYPE <MPS/FREE/MG/MPL> **input type inf**	3.2.1 3.2.2 3.2.3	Code:- 1=MPS 2=FREE 3=MG/MPL/inf		MPS
INPUT LONG NAMES <ON/OFF>	3.2.3	0=OFF	1=ON	OFF
INPUT INTORG [UPPER] BOUND = v	6.3.3	1	-	0 (See below)
OBJECTIVE NAME (objname)	3.2.1			blank
RHS NAME (rhsname)	3.2.1			blank
RANGES NAME (rngname)	3.2.1			blank
BOUNDS NAME (bndname)	3.2.1			blank
INPUT SAVE NAMES <ON/OFF>	7.3.2	0=OFF	1=ON	OFF

OBJECTIVE OFFSET <ON/OFF>	7.3.6	0=OFF	1=ON	OFF
INPUT PERTURBATION = n	Below	Code:- 0=OFF 1=Fixed 2=Variable		0
MAXIMUM INPUT ERRORS = n	Below	1	-	50
INPUT ERRORS ACCEPTED	Below	0=Not acc.ptd	1= acc.ptd	Not acc.ptd

The default INPUT INTORG BOUND = 0 is translated to a high value - i.e. variables specified as integer by 'INTORG' markers have no upper bound.

The following additional commands are available:

INPUT PERTURBATION = n (Default: 0)

The above command may invoke perturbations applied to the RHS immediately after INPUT (occasionally useful to resolve problems).

MAXIMUM INPUT ERRORS = n (Default: 50)
INPUT ERRORS ACCEPTED (Default: Not accepted)

These commands control the handling of non-trivial errors in the data input. 'MAXIMUM INPUT ERRORS' controls how many can be accepted after which an abort will occur whether 'ACCEPTED' or not. 'INPUT ERRORS ACCEPTED' allows the calculations to proceed provided that less than 'MAXIMUM' errors have occurred.

Back to Chapter contents

B2.3 Choice of main algorithm

Command	Manual ref	Lowest value	Highest value	Default value
SCALE <ON/OFF>	3.3.6	0=OFF	1=ON	ON
PRESOLVE <ON/OFF>	3.3.5	0=OFF	1=ON	OFF
ALGORITHM PRIMAL	3.3.1 4.6.1	0=OFF	1=ON	=0 (PRIM)
ALGORITHM DUAL **dual <on/off>**	3.3.1 4.6.1	0=OFF	1=ON	=1 (DUAL) (dflt:OFF)
ALGORITHM IPM **ipm <on/off>**	3.3.3 5.2.1	0=OFF	1=ON	=1 (IPM) (dflt:OFF)
IPM SCALE <ON/OFF>	Below	0=OFF	1=ON	ON

IPM BASREC <ON/OFF>	3.3.3 5.2.6	0=OFF	1=ON	ON
POSTSOLVE <ON/OFF>	3.3.5	0=OFF	1=ON	ON
MIP <ON/OFF>	3.3.4 6.8.5	0=OFF	1=ON	ON

The following command:

IPM SCALE <ON/OFF> (Default: ON)

controls scaling when the IPM algorithm is selected.

Back to Chapter contents

B2.4 Maximum limits

Command	Manual ref	Lowest value	Highest value	Default value
MAXIMUM SIMPLEX ITERATIONS =n **maximum primal iterations = n**	3.3.1 3.7.2 4.6.4	1	-	10000
MAXIMUM IPM ITERATIONS = n	3.3.3 3.7.2 5.2.5	1	-	80
MAXIMUM MIP INTEGER [SOLUTIONS] = n **MAXIMUM MIP INTSOL = n**	3.3.4 3.7.2 6.8.3	1	-	300
MAXIMUM MIP NODES = n **maximum mip subproblems = n**	3.3.4 3.7.2 6.8.3	1	-	50000
MAXIMUM MIP TIME = v	3.3.4 6.8.3	0.0	-	50000.0
MAXIMUM AROUND INTEGER [SOLUTIONS] = n **MAXIMUM AROUND INTSOL = n**	6.8.3	1	-	1
MAXIMUM AROUND NODES = n **maximum AROUND subproblems = n**	6.8.3	1	-	5000
MAXIMUM AROUND TIME = v	6.8.3	0.0	-	5000.0
MAXIMUM MIP SPACE = n	6.8.3	1	-	10000

Back to Chapter contents

B2.5 Save and restart commands

Command	Manual ref	Lowest value	Highest value	Default value
SIMPLEX SAVE FREQUENCY = n	3.5.1 4.6.4 7.5.3	0	-	10
IPM SAVE FREQUENCY = n	3.5.1 5.2.5 7.5.3	0	-	10
MIP SAVE FREQUENCY = n	3.5.1 6.8.4 7.5.3	0	-	500
INPUT RESTART <ON/OFF> **input mps skip**	3.5.2 3.7.2	0=OFF	1=ON	OFF
SIMPLEX START RESTART **bbasis input <on/off>** **bbasis <on/off>**	3.3.2 3.5.1 3.7.2 4.6.2 4.6.4	0=OFF	1=ON	OFF
IPM RESTART <ON/OFF>	3.5.1 3.7.2 5.2.5	0=OFF	1=ON	OFF
BASREC RESTART <ON/OFF> **restart basrec** **ipm skip <on/off>**	5.2.6	0=OFF	1=ON	OFF
MIP RESTART <ON/OFF>	3.5.1 3.7.2 6.8.4	0=OFF	1=ON	OFF

Back to Chapter contents

B2.6 SSX controls:- algorithmic

Command	Manual ref	Lowest value	Highest value	Default value
SIMPLEX START INPUT BASIS	3.3.2 3.5.3 4.6.2 4.6.5	0=OFF	1=ON	OFF
SIMPLEX START CRASH **crash <on/off>**	3.3.2 4.6.2	0=OFF	1=ON	ON
SIMPLEX START UNIT BASIS	3.3.2 4.6.2	0=ON	1=OFF	CRASH
PRIMAL DEVEX <ON/OFF/ **SINGLE/DOUBLE>**	4.6.1	Code:- 0=OFF 1=ON/SINGLE 2=DOUBLE		OFF(=0)
DUAL DEVEX <ON/OFF>	4.6.1	Code:- 0=OFF 1=ON		OFF(=0)
DUAL ADEGEN = n	4.6.1	Code:- 0=OFF 1=ON (simple) 2=ON (advanced)		OFF(=0)
FORTOM <ON/OFF/AUTO>	4.6.1	Code:- 0=OFF 1=ON 2=AUTO		AUTO

Back to Chapter contents

B2.7 SSX controls:- parameters

Command	Manual ref	Lowest value	Highest value	Default value
Controls for all SSX algorithms				
RHS TOLERANCE = v **FEASIBILITY TOLERANCE = v**	4.6.3	1.0d-25	1.0	1.0e-5
DJ TOLERANCE = v	4.6.3	1.0d-25	1.0	1.0e-5
ZERO TOLERANCE = v	4.6.3	1.0d-25	1.0	1.0e-15
PIVOT DIFFERENCE EPSILON = V	4.6.7	1.0d-25	1.0	1.0d-2
INVERT FREQUENCY = n	3.3.1 4.6.1	1	-	50
FORTOM ACTIVATE PERCENT = n	4.6.3	0	101	40
FORTOM ACTIVATE GROWTH = n	4.6.3	0	1000	100
PRIMAL controls				
PRIMAL DEVEX RATIO = v	4.6.3	>0.0	<1.0	0.4
PRIMAL MSUB = n	4.3.2 4.6.1	1	10	4
PRIMAL PIVOT ZERO TOLERANCE = v	4.6.7	Fixed		1.0D-35
PRIMAL PIVOT ADMIT THRESHOLD = v **pivot tolerance = v**	4.6.3 4.6.7	1.0d-25	1.0	1.0d-8
PRIMAL PIVOT ADMIT RELATIVE = v	4.6.7	Fixed		0.0
PRIMAL RELATIVE EPSILON = v	4.6.7	Fixed		0.0
DUAL controls				
DPROGRESS CRITERION = v	4.6.3	1.0d-25	1.0	1.0e-25
DPROGRESS FREQUENCY = n	4.6.3	1	5	1
DUAL PIVOT ZERO TOLERANCE = v	4.6.7	1.0d-25	1.0	1.0D-14
DUAL PIVOT ADMIT THRESHOLD = v	4.6.3 4.6.7	1.0d-25	1.0	1.0e-8
DUAL PIVOT ADMIT RELATIVE = v	4.6.7	1.0d-25	1.0	0.0
DUAL RELATIVE EPSILON = v	4.6.7	1.0d-25	1.0	0.0
DFTRAN ROWWISE <ON/OFF>	Below	0=OFF	1=ON	ON
INVERT controls				

INVERT PIVOT ZERO TOLERANCE = v **invert tolerance = v**	4.6.3 4.6.7	1.0d-25	1.0	1.0D-7
INVERT PIVOT ADMIT THRESHOLD = v	4.6.7	Fixed		0.0
INVERT PIVOT ADMIT RELATIVE = v **pivot threshold = v**	4.6.3 4.6.7	1.0d-25	1.0	1.0d-2
INVERT RELATIVE EPSILON = v	4.6.7	Fixed		0.0
INVERT MEDIUM DENSITY = v	Below	-0.1	100.1	10.0
INVERT RELEPS <ON/OFF>	Below	0=OFF	1=ON	1 (ON)
INVERT THRESHOLD COLWISE	Below	0=OFF	1=ON	0 (OFF)
PIVOT RATIO LIMIT = v	Below	10.0	1.0d+25	1.0d+12

The following additional commands are available:

DFTRAN ROWWISE <ON/OFF> (Default ON)

This controls the sequence of dot-product calculations in DUAL - normally faster row-wise than column-wise because the sparsity permits more skipping. Some cases may arise where this is not true - particularly if there is a large proportion of rows to columns.

INVERT MEDIUM DENSITY = v
INVERT RELEPS <ON/OFF>
INVERT THRESHOLD COLWISE
PIVOT RATIO LIMIT = V

These commands control specialised tuning of the INVERT algorithm.

Back to Chapter contents

B2.8 IPM controls:- algorithmic

Command	Manual ref	Lowest value	Highest value	Default value
IPM ALGORITHM <AFFINE/ BARRIER/PDC> **ipm algorithm = n**	5.2.1	Code:- 1=AFFINE 2=BARRIER 3=PDC		3 (=PDC)
IPM SOLVER <CHOLESKY/ SUPERNODE/ XSUPERNODE/ AUGMENTED> **ipm version = n**	5.2.3	Code:- 1=CHOLESKY 2=SUPERNODE 3=XSUPERNODE 4=AUGMENTED		3 (=XSUP)
IPM STARTING POINT METHOD =n **ipm method = n**	5.2.2	1	3	1

The following commands are reserved for future use:

```
IPM SOLVER AUGMENTED
ipm version = 4
```

The augmented solver is currently available in a specially licensed release.

Back to Chapter contents

B2.9 IPM controls:- parameters

Command	Manual ref	Lowest value	Highest value	Default value
IPM RELATIVE EPSILON = v **ipm releps = v**	5.2.1	1.0d-25	1.0	1.0e-7
IPM FEASIBILITY EPSILON = v **ipm feaseps = v**	5.2.1	1.0d-25	1.0	1.0e-4
CHOLESKY CG TOLERANCE = v	5.2.4	1.0d-8	10.0	1.0e-4
CHOLESKY ERROR TOLERANCE = v	5.2.4	0.1	1.0d+10	10.0
MAXIMUM CG ITERATIONS = n	5.2.4	0 (=OFF)	3	3
IPM BIGM WEIGHT = v **ipm weight = v**	5.2.2	>0.0	<1.0	0.1
IPM DARE = v	5.2.1	0.1	0.99995	0.9995
IPM PHI = v	5.2.1	0.0	-	10.0
IPM POWER = n	5.2.1	1	5	4
IPM TOFIX = v	5.2.3	1.0d-25	1.0	1.0e-12
IPM GRAPHICAL DISPLAY <ON/OFF> **ipm display = n**	3.3.3 5.2.7	0=OFF	1=ON	OFF

Back to Chapter contents

B2.10 MIP controls:- algorithmic

Command	Manual ref	Lowest value	Highest value	Default value
MIP PREPROCESS <ON/OFF/ ROOT ONLY>	3.3.4 6.7.2	Code:- 0=OFF 1=ON 2=ROOT ONLY		OFF
GENERATE CUTS <ON/OFF>	6.7.3	0=OFF	1=ON	OFF
MIP ANALYSE DUAL <ON/OFF>	6.7.4	0=OFF	1=ON	OFF
MIP DUAL <ON/OFF>	6.8.6	0=OFF	1=ON	ON
MIP AGENDA INPUT <ON/OFF>	6.6.4	0=OFF	1=ON	OFF
MIP LIST INPUT <ON/OFF> **mip priority list <on/off>**	6.6.6	0=OFF	1=ON	OFF
MIP AUTO ROUNDING <ON/OFF>	6.8.1	0=OFF	1=ON	OFF
MIP AROUND SOLVER <SSX/IPM>	6.8.1	Code:- 0=SSX 1=IPM		SSX
MIP CLASSIFIY ROWS <ON/OFF>	6.10.1	0=OFF	1=ON	OFF

Back to Chapter contents

B2.11 MIP controls:- parameters

Command	Manual ref	Lowest value	Highest value	Default value
MIP FNODECHOICE = n	6.5.2	Code:- See below		1
MIP SNODECHOICE = n	6.5.2	Code:- See below		7
MIP VARCHOICE = n	6.5.2	Code:- See below		1
MIP PRIORITY UP <ON/OFF>	3.3.4 6.5.3	0=OFF	1=ON	OFF
INTEGER TOLERANCE = v	6.8.2	1.0d-25	<0.5	1.e-3
MIP BOUND = v **mip cutoff bound = v**	6.8.2	-	-	High value
MIP BOUND RELATIVE = v	6.8.2	-	-	High value
MIP CUTOFF TOLERANCE = v	6.8.2	0.0	1.0d+5	1.0e-12
MIP CUTOFF RELATIVE = v	6.8.2	0.0	-	0.0
MIP CUTOFF RELISOL = v	6.8.2	0.0	-	0.0
ACT QUOTA = n	6.7.3	1	-	2
CUT QUOTA = n	6.7.3	1	-	5
MIP FIX QUOTA = n	6.7.2	1	-	10
MIP ROUNDING FRACTION = v	6.8.1	0.0	<0.5	0.0
MIP PREPROCESS LEVEL = n	6.7.2	1	3	3

Codes for MIP FNODCH/SNODCH are:-

1 = LIFO
2 = FIFO
3 = Best objective value
4 = Worst objective value
5 = Minimum number non-integer
6 = Minimum sum of fractions
7 = Best projection heuristic

Codes for MIP VARCH are:-

1 = Minimum fraction
2 = Maximum fraction
3 :- Invalid
4 = Maximum cost
5 = Minimum cost times fraction

Back to Chapter contents

B2.12 Minor algorithm controls

Command	Manual ref	Lowest value	Highest value	Default value
SCALE controls				
SCALE PASSES = n	3.3.6	1	4	4
SCALE VARIANCE = v	3.3.6	0.1	1000.0	10.0
SCALE OBJECTIVE <ON/OFF>	Appendix	0=OFF	1=ON	OFF (ON=2)
SCALE USING OBJECTIVE <ON/OFF>	Appendix	0=OFF	1=ON	OFF
PRESOLVE controls				
PRESOLVE LEVEL = n	3.3.5	1	5	5
PRESOLVE PASSES = n	Appendix	1	4	4
PRESOLVE SAFE LEVEL = n	Appendix	1	5	5
CRASH controls				
CRASH ADMIT THRESHOLD = v	Appendix	1.00d-12	10.0	0.001
CRASHY PIVOTS = n	Appendix	0	50	1
CRASH SELECTION LIMIT = n	Appendix	1	10	2
CRASH TOLERANCE FACTOR = v	Appendix	1.0	1.0d+8	10000.0
BASREC (push) controls				
PUSH CALSOL FREQUENCY = n	Appendix	1	-	10
PUSH IGNORE THRESHOLD = n	Appendix	1.0d-25	10.0	1.0e-4
DUAL CRASH PUSH <ON/OFF>	Appendix	0=OFF	1=ON	OFF
DUAL PUSH <ON/OFF>	5.2.6	0=OFF	1=ON	ON

The following commands are also available:

```
SCALE OBJECTIVE <ON/OFF>                (Default OFF)
SCALE USING OBJECTIVE <ON/OFF>          (Default OFF)
```

By default the SCALE algorithm does not scale the objective row and does not include objective values in the calculations that derive column-scales. Either or both features can be set ON to handle special cases with a very wide range of objective values in the model.

```
PRESOLVE PASSES = n                     (Default = 4)
PRESOLVE SAFE LEVEL = n                 (Default = 5)
```

The limit to PRESOLVE passes prevents over-running which is generally not profitable after 4 repeats. 'SAFE LEVEL' can be used to define a lower than maximum level at which to re-try if PRESOLVE finds infeasibility. PRESOLVE may find spurious infeasibility (particularly at higher levels) if the model has poor scaling.

```
CRASH ADMIT THRESHOLD = v               (Default = 0.001)
CRASHY PIVOTS = n                       (Default = 1)
CRASH SELECTION LIMIT= n                (Default = 2)
CRASH TOLERANCE FACTOR = v              (Default = 10000.0)
```

These commands refer to experimental CRASH procedures which are present in the code for tuning difficult problems. They should not be employed by the end-user without expert advice.

```
PUSH CALSOL FREQUENCY = n               (Default = 10)
PUSH IGNORE THRESHOLD = n               (Default = 1.0e-4)
DUAL CRASH PUSH <ON/OFF>                (Default OFF)
```

These commands refer to experimental BASREC procedures which are present in the code for tuning difficult problems. They should not be employed by the end-user without expert advice.

Back to Chapter contents

B2.13 Log level and frequency

Command	Manual ref	Lowest value	Highest value	Default value
PRIMAL LOG FREQUENCY = n **simplex log frequency = n**	3.3.1 4.6.6	1	-	1
SIMPLEX LOG LEVEL = n **primal log level = n**	3.3.1 3.4.3 4.6.6 (7.5.2)	0	4	1
INVERT LOG LEVEL = n	3.3.1 3.4.3 4.6.6 (7.5.2)	0	4	1
IPM LOG LEVEL = n	3.3.3 3.4.3 5.2.7 (7.5.2)	0	4	1
MIP LOG LEVEL = n	3.3.4 3.4.3 6.9 (7.5.2)	0	4	1
NODE LOG FREQUENCY = n	3.3.4 6.9	1	-	1
MIP PREPROCESS LOG LEVEL = n	6.7.2 (7.5.2)			0
MIP SIMPLEX LOG LEVEL = n **mip ssx log level = n**	Appendix	0	4	0
PRESOLVE LOG LEVEL = n	3.3.5 (7.5.2)	0	4	1
PUSH LOG FREQUENCY = n	5.2.7	1	-	10
PUSH LOG LEVEL = n	5.2.7 (7.5.2)	0	4	1
LOG DISPLAY	3.4.3	0	5	1
LOG DISPLAY LEVEL = n	3.4.3			
LOG DISPLAY ONLY	3.4.3			OFF

The following commands are also available:

MIP SIMPLEX LOG LEVEL = n (Default: 0)
mip ssx log level = n (Default: 0)

During MIP the standard logs of PRIMAL, DUAL and INVERT are switched off by resetting the level to zero. However this can be countermanded with 'MIP <SIMPLEX/SSX> LOG [LEVEL] = n' giving a level which then applies to all three algorithms.

LOG DISPLAY LEVEL = n (Default: 1)

This command is used to restrict or expand the display of logged messages to any level. The command does not change the level of messages that are issued to the log in the first place.

Back to Chapter contents

B2.14 Output controls

Command	Manual ref	Lowest value	Highest value	Default value
OUTPUT <ON/OFF>	3.4.4	0=OFF	1=ON	ON
OUTPUT TYPE <MPL/RW/STD>	3.4.1 3.4.2	Code:- 1=STD 2=RW/MPL		STD
OUTPUT SUPPRESS ZERO	3.4.4	0=OFF	1=ON	OFF
OUTPUT CHANNEL = n	Appendix			-18
OUTPUT DISPLAY [ONLY]	Appendix			
OUTPUT BASIS <ON/OFF> **basis output <on/off>**	3.5.3 4.6.5	0=OFF	1=ON	OFF
LOG CHANNEL = n	7.5.1			-19
MIP AGENDA OUTPUT <ON/OFF/ALL>	6.6.5	Code:- 0=OFF 1=ON 2=ALL		OFF
MIP LIST OUTPUT <ON/OFF/ALL>	6.6.6	Code:- 0=OFF 1=ON 2=ALL		OFF
DISPLAY MEMORY	Appendix	0=OFF	1=ON	OFF

The following commands are also available:

OUTPUT DISPLAY [ONLY] (Default: No display)

'OUTPUT DISPLAY' (with or without 'ONLY') causes the normal output file to be suppressed and solution output to be displayed or written to the standard output.

OUTPUT CHANNEL = n (Default: -18)

As with the log channel a negative number indicates that the system must open the channel internally. A positive number indicates that the channel is opened externally. Using n = 6 (standard output unit) the solutions are written out online to the screen.

OUTPUT TYPE MPL (Default: STD)

This is an alternative for output type 'RW'.

DISPLAY MEMORY (Default: None)

This command causes a record of memory assignments to be issued to the log (and displayed if display-level is 2 or more).

Back to Chapter contents

B2.15 Limits for cut-generation and matrix extension

Command	Manual ref	Lowest value	Highest value	Default value
MAXIMUM CUT NONZEROS = n	Appendix	0	-	0 (See note below)
MAXIMUM CUTS = n	Appendix	0	-	
MAXIMUM EXTRA ROWS = n	8.1.5	0	-	
MAXIMUM EXTRA COLUMNS = n **maximum extra cols = n**	8.1.5	0	-	
MAXIMUM SPARE ROWSPACE = n	8.1.5	0	-	
MAXIMUM SPARE COLSPACE = n	8.1.5	0	-	

The following commands are also available:

MAXIMUM CUT NONZEROS = n (Default: 0
MAXIMUM CUTS = n (Default: 0

These commands specify user-given limits on the cut pool space.

To determine the maximum limits for cut generation and matrix extension the system takes the maximum of the user-given value and a quota-based calculation. Quotas are forced to zero when GENERATE CUTS is OFF.

Back to Chapter contents

B3. Alphabetical list of commands

Command	Default	References
ACT QUOTA = n	2	6.7.3
AGENDA FILE (fnam)	'model.agn'	B2.1
ALGORITHM DUAL	OFF	3.3.1/4.6.1
ALGORITHM IPM	OFF	3.3.3/5.2.1
ALGORITHM PRIMAL	Primal ON	3.3.1/4.6.1
BASREC RESTART <ON/OFF>	0 (norestart)	5.2.6
BASIS FILE (fnam)	'model.bas'	3.6
BASIS OUTPUT <ON/OFF>	OFF	B2.14
BBASIS FILE (fnam)	'model.bbf'	B2.1
BBASIS INPUT <ON/OFF>	OFF	B2.5
BBASIS <ON/OFF>	OFF	B2.5
BOUNDS NAME (bndname)	blank	3.2.1
CHOLESKY CG TOLERANCE = v	1.e-4	5.2.4
CHOLESKY ERROR TOLERANCE = v	10.0	5.2.4
CRASH ADMIT THRESHOLD = v	0.001	B2.12
CRASH <ON/OFF>	ON	B2.6
CRASH SELECTION LIMIT = n	2	B2.12
CRASH TOLERANCE FACTOR = v	10000.0	B2.12
CRASHY PIVOTS = n	1	B2.12
CUT QUOTA = n	5	6.7.3
DFTRAN ROWWISE <ON/OFF>	ON	B2.7
DIRECTORY NAME (pthnam)	none	3.6
DISPLAY MEMORY	0 (none)	B2.14
DJ TOLERANCE = v	1.0e-5	4.6.3
DPROGRESS CRITERION = v	1.0e-25	4.6.3
DPROGRESS FREQUENCY = n	1	4.6.3
DUAL ADEGEN = n	0 (OFF)	4.6.1
DUAL CRASH PUSH <ON/OFF>	OFF	B2.12
DUAL <ON/OFF>	OFF	B2.3
DUAL PIVOT THRESHOLD = v	1.0d-8	4.6.3
DUAL PUSH <ON/OFF>	ON	5.2.6
FEASIBILITY TOLERANCE = v	1.0e-5	B2.7
FORTOM ACTIVATE PERCENT = n	40	4.6.3
FORTOM ACTIVATE GROWTH = n	100	4.6.3
FORTOM <ON/OFF/AUTO>	OFF	4.6.1
GENERATE CUTS <ON/OFF>	OFF	6.7.3
INPUT ERRORS ACCEPTED	Not accepted	B2.2
INPUT FILE (fnam)	'model.mps'	3.6
INPUT INTORG BOUND = v	High value	6.3.3
INPUT LONG NAMES <ON/OFF>	OFF	3.2.3
INPUT MPS SKIP	0 (noskip)	B2.5
INPUT PERTURBATION = n	0	B2.2
INPUT RESTART <ON/OFF>	OFF	3.5.2/3.7.2/5.2.5
INPUT SAVE NAMES <ON/OFF>	OFF	7.3.2
INPUT TYPE <NGC/MPL/MG/MAG/ MPS/FREE/INF>	MPS	3.2.1/3.2.2/3.2.3

Back to Chapter contents

B3. Alphabetical list of commands (cont)

Command	Default	References
INVERT FREQUENCY = n	50	3.3.1/4.6.1
INVERT LOG LEVEL = n	1	3.3.1/3.4.3/7.5.2
INVERT MEDIUM DENSITY = v	10.0	B2.7
INVERT RELEPS <ON/OFF>	1 (ON)	B2.7
INVERT THRESHOLD COLWISE	0 (OFF)	B2.7
INVERT TOLERANCE = v	1.0e-7	4.6.3
IPM ALGORITHM = n	3 (PDC alg)	B2.8
IPM ALGORITHM AFFINE	=1	5.2.1
IPM ALGORITHM BARRIER	=2	5.2.1
IPM ALGORITHM PDC	=3	5.2.1
IPM BASREC <ON/OFF>	ON	3.3.3/5.2.6
IPM BIGM WEIGHT = v	0.1	5.2.2
IPM DARE = v	0.9995	5.2.1
IPM DISPLAY = n	0	B2.9
IPM FEASEPS = v	1.0e-4	B2.9
IPM FEASIBILITY EPSILON = v	1.0e-4	5.2.1
IPM GRAPHICAL DISPLAY <ON/OFF>	OFF	3.3.3/5.2.7
IPM LOG LEVEL = n	1	3.3.3/3.4.3/ 5.2.7/7.5.2
IPM METHOD = n	1	B2.8
IPM <ON/OFF>	OFF	B2.3
IPM PHI = v	10.0	5.2.1
IPM POWER = n	4	5.2.1
IPM RELATIVE EPSILON = v	1.0e-7	5.2.1
IPM RELEPS = v	1.0e-7	B2.9
IPM RESTART <ON/OFF>	OFF	3.5.1/3.7.2/5.2.5
IPM SAVE FREQUENCY = n	10	3.5.1/5.2.5/7.5.3
IPM SCALE <ON/OFF>	OFF	B2.3
IPM SKIP <ON/OFF>	OFF	B2.5
IPM SOLVER AUGMENTED	=4	B2.8
IPM SOLVER CHOLESKY	=1	5.2.3
IPM SOLVER SUPERNODE	=2	5.2.3
IPM SOLVER XSUPERNODE	=3	5.2.3
IPM STARTING POINT METHOD = n	1	5.2.2
IPM TOFIX = v	1.0e-12	5.2.3
IPM VERSION = n	1 (Cholesky)	B2.8
IPM WEIGHT = v	0.1	B2.9
LOG CHANNEL = n	-19	7.5.1
LOG DISPLAY	Level = 1	3.4.3
LOG DISPLAY LEVEL = n	1	3.4.3
LOG DISPLAY ONLY	OFF	3.4.3
LOG FILE (fnam)	'model.log'	3.6
MAXIMIZE	0 (min)	4.6.1
MAXIMUM CG ITERATIONS = n	3	5.2.4
MAXIMUM CUTS = n	Mrow*qta	B2.15
MAXIMUM CUT NONZEROS = n	Mnnz*qta	B2.15

Back to Chapter contents

B3. Alphabetical list of commands (cont)

Command	Default	References
MAXIMUM EXTRA COLS = n	0	B2.15
MAXIMUM EXTRA COLUMNS = n	0	8.1.5
MAXIMUM EXTRA ROWS = n	0	8.1.5
MAXIMUM INPUT ERRORS = n	50	B2.2
MAXIMUM IPM ITERATIONS = n	80	3.3.3/3.7.2/5.2.5
MAXIMUM MIP INTEGER <SOLUTIONS> = n	300	3.3.4/3.7.2/6.8.3
MAXIMUM MIP INTSOL = n	300	3.3.4/3.7.2/6.8.3
MAXIMUM MIP NODES = n	50000	3.3.4/3.7.2/6.8.3
MAXIMUM MIP SPACE = n	10000	6.8.3
MAXIMUM MIP SUBPROBLEMS = n	50000	B2.4
MAXIMUM MIP TIME = v	50000.0	3.3.4/6.8.3
MAXIMUM PRIMAL ITERATIONS = n	10000	3.3.1/3.7.2/4.6.4
MAXIMUM SIMPLEX ITERATIONS = n	10000	B2.4
MAXIMUM SPARE COLSPACE = n	0	8.1.5
MAXIMUM SPARE ROWSPACE = n	0	8.1.5
MINIMIZE	0 (min)	4.6.1
MIP ANALYSE DUAL <ON/OFF>	OFF	6.7.4
MIP AGENDA INPUT <ON/OFF>	OFF	6.6.4
MIP AGENDA OUTPUT <ON/OFF>	OFF	6.6.5
MIP AUTO ROUNDING <ON/OFF>	OFF	6.8.1
MIP BOUND = v	High value	B2.11
MIP CLASSIFY ROWS <ON/OFF>	OFF	6.10.1
MIP CUTOFF BOUND = v	High value	6.8.2
MIP CUTOFF TOLERANCE = v	1.0e-12	6.8.2
MIP DUAL <ON/OFF>	ON	6.8.6
MIP FIX QUOTA = n	10	6.7.2
MIP FIXMIX OUTPUT <ON/OFF/ALL>	OFF	B2.14
MIP FNODECHOICE = n	1	6.5.2
MIP LIST INPUT <ON/OFF>	OFF	6.6.6
MIP LIST OUTPUT <ON/OFF>	OFF	6.6.6
MIP LOG LEVEL = n	1	3.3.4/3.4.3/ 6.9/7.5.2
MIP <ON/OFF>	ON	3.3.4/6.8.5
MIP PREPROCESS LEVEL = n	3	6.7.2
MIP PREPROCESS LOG LEVEL = n	0	6.7.2/7.5.2
MIP PREPROCESS <ON/OFF>	OFF	3.3.4/6.7.2
MIP PREPROCESS ROOT ONLY	2	6.7.2
MIP PRIORITY LIST <ON/OFF>	OFF	B2.10
MIP PRIORITY UP <ON/OFF>	OFF	3.3.4/6.5.3
MIP PROBE ROUNDING = n	0	6.8.1
MIP PROBE ROUNDING <ON/OFF>	OFF	6.8.1
MIP RESTART <ON/OFF>	OFF	3.5.1/3.7.2/6.8.4

Back to Chapter contents

B3. Alphabetical list of commands (cont)

Command	Default	References
MIP ROUNDING FRACTION = v	0.0	6.8.1
MIP SAVE FREQUENCY = n	500	3.5.1/6.8.4/7.5.1
MIP SIMPLEX LOG LEVEL = n	0	B2.13
MIP SNODECHOICE = n	7	6.5.2
MIP SSX LOG [LEVEL] = n	0	B2.13
MIP VARCHOICE = n	1	6.5.2
MODEL NAME (modnam)	'model'	3.6
NODE LOG FREQUENCY = n	1	3.3.4/6.9
OBJECTIVE NAME (objname)	blank	3.2.1
OBJECTIVE OFFSET <ON/OFF>	OFF	7.3.6
OUTPUT BASIS <ON/OFF>	OFF	3.5.3/4.6.5
OUTPUT CHANNEL = N	-18	B2.14
OUTPUT DISPLAY [ONLY]	OFF	B2.14
OUTPUT FILE (fnam)	'result.dat'	3.6
OUTPUT <ON/OFF>	ON	3.4.4
OUTPUT SUPPRESS ZERO	OFF	3.4.4
OUTPUT TYPE <MPL/RW/STD>	STD	3.4.1/3.4.2
PATH NAME (pthnam)	none	B2.1
PIVOT RATIO LIMIT = v	1.e+12	B2.7
PIVOT THRESHOLD = v	1.e-2	4.6.3
PIVOT TOLERANCE = v	1.e-8	4.6.3
POSTSOLVE <ON/OFF>	ON	3.3.5
PRESOLVE LEVEL = n	5	3.3.5
PRESOLVE LOG LEVEL = n	1	3.3.5/7.5.2
PRESOLVE <ON/OFF>	ON	3.3.5
PRESOLVE PASSES = n	4	B2.12
PRESOLVE SAFE LEVEL = n	5	B2.12
PRIMAL DEVEX <ON/OFF/ SINGLE/DOUBLE>	OFF (=0)	4.6.1
PRIMAL DEVEX RATIO = v	0.4	4.6.3
PRIMAL LOG FREQUENCY = n	1	B2.13
PRIMAL LOG LEVEL = n	1	B2.13
PRIMAL MSUB = n	4	4.3.2/4.6.1
PUSH CALSOL FREQUENCY = n	10	B2.12
PUSH IGNORE THRESHOLD = n	1.0e-4	B2.12
PUSH LOG FREQUENCY = n	10	5.2.7
PUSH LOG LEVEL = n	1	5.2.7/7.5.2
RESTART BASREC	0 (norestart)	B2.5
RHS NAME (rhsname)	blank	3.2.1
RANGES NAME (rngname)	blank	3.2.1
SCALE OBJECTIVE <ON/OFF>	OFF (ON>=2)	B2.12
SCALE <ON/OFF>	ON	3.3.6
SCALE PASSES = n	4	3.3.6
SCALE USING OBJECTIVE <ON/OFF>	OFF	B2.12
SCALE VARIANCE = v	10.0	3.3.6
SIMPLEX LOG FREQUENCY = n	1	3.3.1/4.6.6

Back to Chapter contents

B3. Alphabetical list of commands (cont)

Command	Default	References
SIMPLEX LOG LEVEL = n	1	3.3.1/3.4.3/ 4.6.4/7.5.2
SIMPLEX SAVE FREQUENCY = n	10	3.5.1/4.6.4/7.5.3
SIMPLEX START INPUT BASIS	OFF	3.3.2/3.5.3/ 4.6.2/4.6.5
SIMPLEX START CRASH	ON	3.3.2/4.6.2
SIMPLEX START RESTART	OFF	3.3.2/3.5.1/3.7.2 4.6.2/4.6.4
SIMPLEX START UNIT BASIS	OFF (crash ON)	3.3.2/4.6.2
ZERO TOLERANCE = v	1.0e-15	4.6.3

The following commands have not been described. They are provided for research and testing purposes only.

Command	Default
ALGORITHM KCRASH	OFF
AUGMENTED LOG LEVEL = n	2
BASREC LGLFLAG <ON/OFF>	OFF
BASREC POSTSOLVE <ON/OFF>	ON
DUAL RESTRICTED PUSH <ON/OFF>	OFF
DUMP LENGTH = n	0
EXPLICIT OBJECTIVE <ON/OFF>	OFF
HINIT FREQUENCY = n	1
INDIRECT ETAS <ON/OFF>	OFF
INDIRECT UETAS (ON/OFF>	OFF (2=OFF)
INTERFACE SORT <ON/OFF>	ON
IPM INDICATE <ON/OFF>	OFF
IPM PREFIX <ON/OFF>	ON
IPM UPIV = v	1.0e-6
KCRASH EQONLY <ON/OFF>	OFF
KCRASH <ON/OFF>	OFF
MIP DISK <ON/OFF>	OFF
MIP FIXMIX = n	0
MIP FIXMIX OUTPUT LEVEL = n	0 (OFF)
MIP INFEAS OUTPUT	0 (OFF)
MIP NODS OUTPUT	0 (OFF)
MIP PRIORITY BRANCH <ON/OFF>	OFF
MIP ROUNDING = n	0
MIP SCOVER <ON/OFF>	OFF
PRESOLVE CANCEL FIXCOL	OFF
PRESOLVE CANCEL REDROW	OFF
PRESOLVE CANCEL REMBND	OFF
PRESOLVE CANCEL SFRCOL	OFF
PRESOLVE CANCEL SNGROW	OFF
PRESOLVE CANCEL SNGCOL	OFF

```
    PRIMAL IADC <ON/OFF>            OFF
    PRIMAL IAPR <ON/OFF>            ON
    PRIMAL IDEG <ON/OFF>            OFF
    PRIMAL ISTP <ON/OFF>            OFF
    PRIMAL KSEC = n                 1
    PRIMAL KVEC = n                 0
    PRIMAL TEST ETA = n             0
    PRIMAL TEST ITERATION = n       -1
    PUSH RESIDUAL CHECK = n         0 (OFF)
    SET Ixxxx = n  (x=A,...,J)      0
    SET Vxxxx = v  (x=A,...,J)      0.0
    SET RDGTOL = v                  1.0d-7
    SET ACGTOL = v                  1.0d-5
    SET RPFTOL = v                  1.0d-6
    SET RDFTOL = v                  1.0d-6
    SET APFTOL = v                  1.0d-1
    SET ADFTOL = v                  1.0d-1
    SET PINFS = v                   1.0d-5
    SET DINFS = v                   1.0d-5
    SET BARSET = v                  0.25
    SET BARGROW = v                 100.0
    ZCRASH = n                      1
```

Back to Chapter contents

APPENDIX E: C-Language Usage

This appendix is intended to provide users of C-language with the means to interpret and make calls to the subroutine library described in chapters 7 and 8.

Contents

E1. Mixed C/Fortran Usage on PC-WIN32 Platforms

E1.1 Entry-point Names

Fortran subroutines of the FortMP system are compiled using default standards. C-language subroutines that interface with Fortran subroutines are adapted to match Fortran defaults - never the other way around. This means that any Fortran subroutine-name or function-name is represented at the LINK-stage by an entry-name as follows:

`-<NAME>@<`*`n`*`>`

Where `<NAME>` is the source name in uppercase, and `<`*`n`*`>` is the number of bytes in the argument list.

In C-language the equivalent is to name the entry in uppercase and to apply the 'stdcall' attribute.

Given as example the FortMP entry:

```
Call Primal(stsl, tctn)
```

This would be represented by the following prototype in (say) a C header file:

```
extern void __stdcall PRIMAL(int *stsl, int *tctn);
```

and the call-statement in the executable code would be:

```
int stsl, tctn;
. . . .
PRIMAL(&stsl, &tctn);
. . . .
```

The linked entry-name for this is `'_PRIMAL@8'`.

Back to Chapter contents

E1.2 Argument lists

As exemplified above, arguments in Fortran are generally pointer-references to the data that is being passed. This holds for all numerical data - both scalars and arrays - the exception being only character data and strings. Characters or strings in Fortran always occupy a fixed number of bytes, and there is no concept of the 'null terminator' to allow variable length strings. In order to pass such an argument the length of the string is placed in the argument-list immediately after the pointer to the 1st character.

An example of a FortMP library entry with a character-type argument is:

```
call Spcdft(COMMND, TCTN)
```

Here `'COMMND'` is a variable-length string of up to 50 characters. The corresponding prototype in C-language would be:

```
 extern void __stdcall SPCDFT(char *commnd, int lng,
                              int *tctn);
```

Where `'lng'` is the actual length of the command string. In the code this routine could be called with the following statements:

```
char *commnd;
int tctn;
 . . .
SPCDFT(commnd, strlen(commnd), &tctn);
 . . .
```

The FortMP subroutine library does not have any numerical arguments that are multi-dimensioned arrays (two or more subscripts) or character arguments that are not scalars. Hence these possibilities are not considered here.

For the sake of a user who only needs the one entry - SUBMP1 - in the FortMP library we give here the prototype and an example of the calling statement:

```
void __stdcall SUBMP1(int *MR, int *NC, int *NAIJ, int *NSET,
            char *PNAME, int lpnm, char *SPID, int lspd,
            double *AIJ, int *ROWIN, int *COLIN,
            double *UPB, double *LOB, double *RHS, double
*LHS,
            double *COST, int *MITYPE,
            int *SREF, int *SFUN, int *SBEG, int *SEND,
            double *VCSOL, int *BSTAT, double *RSCOS,
            int *STSL, int *TCTN);
```

The following is an example of coding the call to SUBMP1:

```
int    MR, NC, NAIJ, NSET, STSL, TCTN;
double *AIJ, *UPB, *LOB, *RHS, *LHS;
double *COST, *VCSOL, *RSCOS;
int    *ROWIN, *COLIN, *BSTAT;
int    *SREF, *SFUN, *SBEG, *SEND;
char   *PNAME, *SPID;
. . .
. . ./* Fill values of MR, NC, NAIJ, NSET here */
. . .
MALLOC(*AIJ, NAIJ);
. . .
. . . /* Allocate all the arrays and strings */
. . .
. . .  /* Fill contents of input arrays here */
. . .  /* Fill the strings PNAME, SPID here */
. . .
SUBMP1(&MR, &NC, &NAIJ, &NSET,
             PNAME, (int)8, SPID, strlen(SPID),
             AIJ, ROWIN, COLIN, UPB, LOB, RHS, LHS,
             COST, MITYPE, SREF, SFUN, SBEG, SEND,
             VCSOL, BSTAT, RSCOS, &STSL, &TCTN);
. . .
```

The full set of C-prototypes for library entries is supplied in the file `'LIBHDR.H'`.

Back to Chapter contents

E1.3 Function Calls

So far the library entries considered have been subroutines - equivalent type 'void' in C - but there are also a few function-type entries. Complications are avoided by limiting these to functions with a numeric, scalar result. Such a function is simply declared in the C-prototype with the corresponding C-language type replacing 'void'.

The functions `'GIRNAM'`, `'GJCNAM'`, `'GIRLNM'` and `'GJCLNM'` (see chapter 7, section 7.3.5) are of character type and as such are not easily referenced in C-language. Because of this the following subroutine calls have been added to the Fortran library:

```
CALL AIRNAM(irow, rname)
CALL AJCNAM(jcol, cname)
CALL AIRLNM((irow, lrname)
CALL AJCLNM(jcol, lcname)
```

Where the original function-result is now the second argument. Equivalent C-prototypes are to be found in the file `'LIBHDR.H'`.

Back to Chapter contents