FortMP – Mex Manual

Last Update
23 April 2008

OptiRisk
SYSTEMS

INVESTOR IN PEOPLE
BS EN ISO 9000 : 2000

Contents

Chapter 1

Scope and Purpose

This document desribes how the optimization software system FortMP can be used as an add-on to MATLAB. FortMP is an industrial strength large scale optimization solver system developed by CARISMA, Brunel University as a research tool which is used also for teaching with many industrial applications. The purpose of this document is to explain to a user of MATLAB how through the MEX-file feature, the user can invoke FortMP dynamic link library(DLL) to create prototype models and applications of optimization within MATLAB. Although MATLAB has its own optimization toolbox , FortMP provides a powerful and desirable alternative

(a) for large scale sparse models and

(b) for discrete(integer) decision models

Chapter 2

Introduction and Background

2.1 Introduction

FortMP is an industrial strength large scale optimization solver system developed by CARISMA, Brunel University as a research tool which is used also for teaching with many industrial applications. FortMP runs on WINDOWS, UNIX and LINUX operating systems. FortMP is connected to modelling languages such as MPL [15] ,SPInE [6] and AMPL [1]. FortMP can also be executed by programs written in C,C^{++},Visual Basic,Fortran77 and Fortran90. FortMP is available

(a) as an executable stand alone optimization solver [3]

(b) as a static library and

(c) as a DLL

From MATLAB, FORTMP can be invoked by the C-mex dll feature and connecting to the MFortMP.dll.

2.2 The Problems Solved by FortMP

FortMP itself is designed to solve problems of the following type:-

LP	**Linear Programming**: with linear objective, linear constraints and continuous variables
MIP	**Mixed Integer Programming**: with linear objective, linear constraints and mixed variable types - discrete and integer
QP	**Quadratic Programming**: with quadratic objective, linear constraints and continuous variables
QMIP	**Quadratic Mixed Integer Programming**: with quadratic objective, linear constraints and mixed variable types - discrete and integer
SP	**Stochastic Programming:** Solves two stage and multistage stochastic programming problems with recourse

In addition to many instances of LP and MIP applications of FortMP and enhanced solution algorithms [7, 8, 9], it is also positioned as a leading QP and SP solver.

For QP and QMIP has been extensively used in financial applications [10] and acts as the solver engine of choice within the PAS system of UBS Warbug; this system has a global client base [5, 14]. The stochastic solver FortSP [13, 2, 4] is also used widely and has been been applied to supply chain planning and financial applications.

CARISMA and OptiRisk Systems have developed extensive training material for LP, MIP, QP and SP applications. For more information on training readers are referred to [12, 16, 11]

The library establishes a structure of scalar variables and arrays, referred to as the 'External Data Interface' in which the users specify a problem to be solved. The structure is passed as parameter arguments in the calls that solve a problem. Calls are available both to solve a problem stated in the external data interface form and to load a problem from an input data file to this form. Input data files may be in standard MPS (extended) layout or in a free-form equivalent.

2.3 Control of the Algorithms by SPECS Commands

A large number of parameter controls referred to as **'SPECS Commands'**, are available, both to tune the solver for difficult problems, and to specify action such as **'Save'** and **'Restart'** enabling a great deal of time-saving in many cases. Initial default values for all these controls are set up when the system is initiated and in the case of library use any control may be reset by a call that passes the appropriate SPECS command prior to calling the solver. The default values should serve for many users however.

In the case of the stand-alone FortMP program SPECS commands are entered via a special control-file named **'FORTMP.SPC'**. This method of changing the controls is also available to the library user, and is more flexible as controls can be changed without re-compiling the system. All SPECS commands are described in the FortMP manual and extensions (in particular file **'Commands.doc'**). In this document there is a description of those commands that are most useful for the general user.

2.4 Library Environment

The original environment for using the library is as a sub-system within a console-type program so that the system can display its progress from time to time to the user. In order to change this action for a Windows environment the user needs to write his own message output system, over-riding the mechanism provided in the library. Since it is not possible to over-ride a DLL subroutine, there is a system of 'Call-backs' to enable this to take place. The user codes certain routines in his own system to be invoked at relevant points during execution of library calls, and specifies their entry-point in the start-up calls made to the DLL prior to the main library-calls. Unfortunately this feature is not yet available for use in matlab.

At the moment the library is 'Single-threaded', which means that multi-processing of separate problems cannot be attempted. A lock is provided (Windows32 version only) to safeguard the system.

Chapter 3

Problem Statement

Given below are the mathematical statements of the kinds of problem that can be solved with the FortMP library.

3.1 LP and MIP problems

The LP or MIP problem may be stated as follows:

$$\begin{array}{rccc} \underset{x}{\text{min/Max}} & k & + & c^T x \\ & \text{s.t.} & & \\ & L \leq & Ax & \leq U \\ & l \leq & x & \leq u \end{array} \tag{3.1}$$

Where

k is a scalar constant offset to the objective
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 $x_1, x_2, \ldots x_n$.
L is an m-vector of lower right hand sides $L_1, L_2, \ldots L_m$.
A is an m x n matrix of coefficients a_{ij}.
U is an m-vector of upper right hand sides $U_1, U_2, \ldots Um$.
l is an n- vector of lower bounds $l_1, l_2, \ldots l_n$.
u is an n- vector of upper bounds $u_1, u_2, \ldots u_n$.

and in addition for MIP some or all of the variables xj may be of one of the following types:

- **Binary**, that is to say restricted to be either zero or one.
- **Integer**, that is to say restricted to integer values
- **Semi-continuous**, that is to say either zero, or continuous in the range 1.0 to an upper bound
- **SOS type 1**, member of set introduced for discrete separable programming
- **SOS type 2**, member of set introduced for continuous separable programming

3.2 QP and QMIP Problems

$$\begin{aligned}
\min_{\mathrm{x}} \quad & k + \quad c^T x \quad + \frac{1}{2} x^T Q x \\
& \text{s.t.} \\
& L \leq \quad Ax \quad \leq U \\
& l \leq \quad x \quad \leq u
\end{aligned} \tag{3.2}$$

k	is a scalar constant offset to the objective
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 $x_1, x_2, \ldots x_n$.
Q	is an symmetric positive semi-definite matrix.
L	is an m-vector of lower right hand sides $L_1, L_2, \ldots L_m$.
A	is an m x n matrix of coefficients a_{ij}.
U	is an m-vector of upper right hand sides $U_1, U_2, \ldots U_m$.
l	is an n- vector of lower bounds $l_1, l_2, \ldots l_n$.
u	is an n- vector of upper bounds $u_1, u_2, \ldots u_n$.

and in addition for QMIP some or all of the variables xj may be of one of the following types:

- **Binary**, that is to say restricted to be either zero or one.
- **Integer**, that is to say restricted to integer values

Chapter 4

The External Data Interface

The external data interface and certain other arguments are described here in order to avoid repeating parameter descriptions many different times. Note that solver inputs to FMP_SUBLP2C etc. are actually outputs of the data-read function FMP_LP2INPC etc.

4.1 Solver Input Scalar arguments

Variable	Type	Description
mr	INTEGER	number of rows -m
nc	INTEGER	number of columns -n
naij	INTEGER	number of non-zero entries in the A-matrix
nqij	INTEGER	number of non-zero entries in the Q-matrix
nset	INTEGER	number of special ordered sets (LP only)
koff	DOUBLE	constant offset to objective -k
pname	STRING	model name. Only the first 8 characters are used
spid	STRING	identity of the BEGIN-line in the SPECS file

4.2 Solver Input array arguments

Array(Length)	Type	Desciption
aij(naij)	DOUBLE	Non-zero elements of the A-matrix
arow(naij)	INTEGER	Row indices belonging to corresponding entries in **AIJ**
acol(naij)	INTEGER	Column indices of the A-matrix belonging to the corresponding entries in **AIJ**
upb(nc)	DOUBLE	Upper bound vector **-u**. Any value greater than or equal to 10^{31} indicates that no upper bound exists.
lob(nc)	DOUBLE	Lower bound vector **-l**. Any value less than or equal to -10^{31} indicates that no lower bound exists.
frhs(mr)	DOUBLE	Upper RHS vector **-U**. Any value greater than or equal to 10^{31} indicates that no upper bound exists.
flhs(mr)	DOUBLE	Lower RHS vector **-L**. Any value less than or equal to -10^{31} indicates that no lower bound exists.
cost(nc)	DOUBLE	Cost vector **-c**.
mitype(nc)	INTEGER	Variable type code for MIP - see below
qij(nqij)	DOUBLE	Non-zero elements of the **Q**-matrix.
qrow(nqij)	INTEGER	Row indices of the **Q**-matrix belonging to the corresponding entries in **qij**.
qcol(nqij)	INTEGER	Column indices of the **Q**-matrix belonging to the corresponding entries in **qij**.
sref(nset)	INTEGER	Reference row numbers of each special ordered set.
sfun(nset)	INTEGER	Function row numbers of each special ordered set.
sbeg(nset)	INTEGER	First column number in each special ordered set.
send(nset)	INTEGER	Last column number in each special ordered set.

Arrays **aij,arow,** and **acol** do not need to organized in any particular sequence so long as the entries correspond to each other.

Coding in array **MITYPE** as follows:

CODE	MEANING
0	Continuous variable
1	Binary Variable
2	Integer Variable
3	Semi-continuous Variable
4	Member of SOS type 1
5	Member of SOS type 2

Codes 3, 4 and 5 may not be used in a QP model.

Arrays **qij**,**qrow** and **qcol** do not need to be organised in any particular sequence so long as the entries correspond to each other. However the full **Q**-matrix should be supplied, and if user states the quadratic objective as an algebraic expression then its coefficients are to be doubled in **Q**. The transposed positions of the upper and lower triangle need not be identical, and may be added and put in as a single entry with indices in either order (FortMP does this operation in any case).

Special Ordered Sets are described in the manual sections 6.2.3 and 6.2.4.

4.3 Solver Output scalar arguments

Variable	Type	Description
obj	**DOUBLE**	Objective value of the solution.
stsl	**INTEGER**	Solution status. Standard values are shown below
tctn	**INTEGER**	Terminate criterion. A value zero indicates that the execution was successful, subject to the value of **stsl**.

Standard values for solution status **stsl** are:-

STSL = 0	No solution has been obtained
STSL = 1	The problem is infeasible
STSL = 2	The problem is unbounded
STSL = 3	Continuous optimum solution
STSL = 4	Integer feasible solution
STSL = 5	Integer optimum solution

4.4 Solver Output array arguments

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

$$\mathbf{1 + MR + NC}$$

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:-

Array(Length)	Type	Description
sol(1+MR+NC)	**DOUBLE**	Primal solution values.
dsl(1+MR+NC)	**DOUBLE**	Dual solution values.
bas(1+MR+NC)	**INTEGER**	Code value for the basis status of each variable in the final solution. Codes are shown below

Codes for each entry in array **BAS** are:-

CODE	MEANING
0	basic variable
-1	Variable is at its lower bound
+1	Variable is at its upper bound

bas may also be an INPUT argument supplying an advanced starting basis. This technique is often used when the user system calls FortMP several times to solve similar problems. It is also used when a basis has been read from the basis input file.

Chapter 5

FortMP-mex Library

The mex-library possesses functions in the following categories:

- functions to initiate and set up controls
- functions to read data from file to the external data interface
- functions to solve a problem and report the solution to the user

There are also many other functions (already described in the FortMP manual) enabling user to manage the solution process in more detail, and to perform many miscellaneous tasks. Most of these will be available in the next release of FortMP-mex library.

5.1 Functions to initiate execution and set-up controls

Entries include:

FMP_BLDFMPC	This function is used to initiate FortMP execution and set defaults for all controls. Only the message call back is set up enabling messages to be directed to matlab
FMP_SPECMDC	Enters a SPECS command over-riding the defaults set by FMP_BLDFMPC.
FMP_SPECINC	Reads the SPECS command-file (fortmp.spc) and stores it internally to save repeated re-reading when the solver has to be called a large number of times.

5.2 Functions to read data from file to the external data interface

FMP_MP2SIZC	Performs the initial stage of input, establishing the dimensions of the input model so that user can allocate space for the external interface.
FMP_LP2INPC	Reads in an LP model from standard input data and converts it to the external interface form - together with a starting basis if specified.
FMP_LPDINPC	Does the same as LP2INPC and, additionally, reads in the dictionary of row and column names together with look-up tables.
FMP_QP2INPC	reads in a QP model from standard input data and converts it to the external interface form -together with a starting basis if specified.

5.3 Functions to solve a problem and report the solution to the user

FMP_SUBLP2C	This function supplies all problem data for an LP or MIP model, solves the problem and returns the solution to the calling program - matlab-users.
FMP_SUBQP2C	This function supplies all problem data for a QP or QMIP model, solves the problem and returns the solution to the calling program - matlab-users.

5.4 Call Specifications

5.4.1 Start-up Entries

FMP_BLDFMPC

This function must be used before any other call to FortMP. It initialises FortMP execution and sets defaults for all controls. Messages are directed to Matlab.

```
Prototype:-
     function tctn = FMP_BLDFMPC()

     Where:- tctn Is a return code
     (zero if OK, non-zero locked or otherwise in error)
```

FMP_SPECMDC

This function passes one SPECS command in order to change the value of some control parameter, over-riding the default set by FMP_BLDFMPC.

```
Prototype:-
    function tctn = FMP_SPECMDC(spec)
```

```
Where:-spec is the text of a SPECS command
      - tctn   is the return-code
      (zero if OK, non-zero otherwise)
```

FMP_SPECINC

This function reads the SPECS command-file (fortmp.spc) and stores it internally to save repeated re-reading when the solver has to be called a large number of times.

```
Prototype:-
    function tctn = FMP_SPECINC()

    Where:- tctn  Is a return code
    (zero if OK, non-zero otherwise)
```

5.4.2 Functions to read data from file to the external interface

FMP_MP2SIZC

Performs the initial stage of input, establishing the dimensions of the input model so that user can allocate space for the external interface.

```
Prototype:-
    function [mr, nc, naij, nqij, nset, tctn] = FMP_MP2SIZC(spid)
```

FMP_LP2INPC

Reads in an LP model from standard input data and converts it to the external interface form - together with a starting basis if specified.

```
Prototype:-
    function [aij, arow ,acol, upb, lob ,frhs ,flhs,cost, mitype, ...
             sref ,sfun ,sbeg ,send ,koff, bas, tctn] = ...
             FMP_LP2INPC(mr,nc,naij,nqij,nset,pname,spid)
```

FMP_QP2INPC

Reads in a QP model from standard input data and converts it to the external interface form - together with a starting basis if specified.

```
Prototype:
    function [aij, arow, acol, qij ,qrow, qcol, upb ,lob ,frhs, flhs,...
             cost, mitype , koff ,bas, tctn] = ...
             FMP_QP2INPC(mr,nc,naij,nqij,pname,spid)
```

5.4.3 Functions to solve a problem and report solution to the user

Arguments used in this section are all described above in the External Data Interface. For all these entries, argument **BAS** becomes an input (as well as an output), supplying the starting basis when the SPECS command **'SSX START INPUT BASIS'** has been given.

FMP_SUBLP2C

This function supplies all problem data for an LP or MIP model, solves the problem and returns the solution to the calling program. This entry supplies all problem data for an LP or MIP model, solves the problem and returns the solution to the calling program.

```
prototype:-
    function [obj, sol ,dsl ,bas ,stsl, tctn] =...
        FMP_SUBLP2C(mr,nc,naij,nset,aij, pname,spid, arow, ...
        acol, upb,lob, frhs, flhs, cost ,mitype, sref, sfun,...
        sbeg, send, koff,bas)
```

Note that when the number of sets is actually zero, the size NSET should have the value 1 on input. This is because some FORTRAN systems do not accept zero as a valid dimension-size. Codes in MITYP determine if one set is actually present.

FMP_SUBQP2C

This entry supplies all problem data for a QP or QMIP model, solves the problem and returns the solution to the calling program.

```
prototype:-
    function [obj, sol ,dsl, bas ,stsl ,tctn] = ...
        FMP_SUBQP2C(mr,nc,naij,nqij,pname,spid,aij,arow, acol, qij ,...
        qrow, qcol, upb ,lob, frhs, flhs, cost,mitype ,koff,bas)
```

Chapter 6

High Level Library Functions

These intended to illustrate basic usage of FortMP in matlab and to provide a user with a high level interaction with FortMP

FMP_READLPMODEL

```
Usage:  [mr, nc, naij, nqij, nset,aij, ...
         arow, acol, upb ,lob ,frhs ,flhs, cost , ...
         mitype, sref, sfun ,sbeg,send ,koff,...
         bas ,tctn] = FMP_ReadLPModel(filename)

Description: Reads an LP model in an mps file filename.

Input: model input filename

Output: - see manual
```

FMP_READQPMODEL

```
Usage:  [mr, nc ,naij, nqij, nset ,aij, arow ,acol, qij ...
         qrow, qcol, upb, lob ,frhs,flhs ,cost, mitype ,...
         KOFF ,bas, tctn]  = FMP_ReadQPMODEL(filename)

Description: Reads a QP model

Input: model input filename

Output: see manual
```

FMP_SOLVEMODELMPS

```
Usage: [obj, sol ,dsl ,bas ,stsl, tctn] ...
        = FMP_SOLVEMODELMPS(filename,modelname,spid)
```

```
    Description: Solves(minimizes) the model in the mps file filename

     inputs :-
         filename : mps filename
         modelname : name of model
         spid : identity of begin line -see manual

     Outputs: see manual
```

FMP_SOLVE

```
 Usage : [obj, sol ,dsl ,bas ,stsl, tctn] = FMP_SOLVE(A,L,U,l,u,k,c)

 Description: Solves(minimizes) a model entered in canonical form

     Inputs:-
         k   is a scalar constant offset to the objective
         c   is an n- vector of the cost coefficients  c1,c2,...cn.
         x   is an n- vector of the structural variables x1,x2,...xn.
         L   is an m-vector of lower right hand sides L1,L2,...Lm.
         A   is an m x n matrix of coefficients aij.
         U   is an m-vector of upper right hand sides U1,U2,...Um.
         l   is an n- vector of lower bounds l1,l2,...ln.
         u   is an n- vector of upper bounds u1,u2,...un.
         mitype - see manual

Output - see manual
```

Chapter 7

Examples in Matlab

FMP_READQPMODEL

```
Usage:  [mr, nc ,naij, nqij, nset ,aij, arow ,acol, qij ...
         qrow, qcol, upb, lob ,frhs,flhs ,cost, mitype ,...
         KOFF ,bas, tctn]  = FMP_ReadQPMODEL(filename)

Description: Reads a QP model

Input: model input filename

Output: see manual

tctn = FMP_BLDFMPC;
if(tctn ~= 0)
    error('FMP_BLDFMPC:  FortMP failed to initialize');
end

specs = sprintf('INPUT FILE NAME(%s)',filename);
tctn = FMP_SPECMDC(specs);

specs = sprintf('MODEL NAME(%s)',filename);
tctn = FMP_SPECMDC(specs);

spid = 'NOSPECS'; [mr, nc, naij, nqij, nset, tctn] = FMP_MP2SIZC(spid);

if(nqij == 0)
    error('Not a QP model - exiting');
end

if(nset ~= 0)
    error('nset should be zero for QP - exiting');
end pname = 'model';


[aij, arow, acol, qij ,qrow, qcol, upb ,lob ,frhs, flhs, ...
cost, mitype ,KOFF ,bas, tctn] = FMP_QP2INPC(mr,nc,naij,nqij,pname,spid);
```

FMP_SOLVEMODELMPS

```
Usage: [obj, sol ,dsl ,bas ,stsl, tctn] ...
        = FMP_SOLVEMODELMPS(filename,modelname,spid)

Description: Solves(minimizes) the model in the mps file filename

 inputs :-
     filename : mps filename
     modelname : name of model
     spid : identity of begin line -see manual

 Outputs: see manual


 if length(filename) >8
    error('file name is too long  - maximum of 8 characters');
 end

 if exist(filename) ~=2
      errstring=  sprintf('File %s: does not exist',filename);
      error(errstring);
end

tctn = FMP_BLDFMPC;

if(tctn ~= 0)
    error('FMP_BLDFMPC:  FortMP failed to initialize');
end

specs = sprintf('INPUT FILE NAME(%s)',filename);
tctn = FMP_SPECMDC(specs);

 pname = modelname;
 specs = sprintf('MODEL NAME(%s)',pname);
 tctn = FMP_SPECMDC(specs);

 if nargin <3
    spid = 'NOSPECS';
 end

[mr, nc, naij, nqij, nset, tctn] = FMP_MP2SIZC(spid);

if (tctn ~= 0)
    error('TCTN is not zero');
end

if(nqij == 0)
    [mr, nc, naij, nqij, nset,aij, ...
```

```
        arow, acol, upb ,lob ,frhs ,flhs, cost , ...
        mitype, sref, sfun ,sbeg,send ,KOFF, bas ,tctn] = ...
        FMP_ReadLPMODEL(filename);

        FMP_SPECMDC('MIP DUAL OFF');
        FMP_SPECMDC('MIP PREP ON');
        FMP_SPECMDC('MIP PRIORITY UP');

         [obj, sol ,dsl ,bas ,stsl, tctn] = FMP_SUBLP2C(mr,nc,naij,nset,aij,...
          pname,spid,arow, acol, upb, lob, frhs, flhs, cost ,mitype, ...
         sref, sfun, sbeg, send, KOFF,bas);


    else

        [mr, nc ,naij, nqij, nset ,aij, arow ,acol, qij, ...
        qrow, qcol, upb, lob ,frhs,flhs ,cost, mitype ,...
        KOFF ,bas, tctn]  = FMP_ReadQPMODEL(filename);

        tctn =FMP_SPECMDC('INPUT QMATRIX FULL'); %SPECMDC


        if(length(find(cost))==0)
               % FortMP fix- Fallacy in reading QP when objective
               % linear vector is zero So shrink frhs and flhs
               % and reduce mr by 1
                frhs = frhs(2:mr);
                flhs = flhs(2:mr);
                mr = mr-1;
                arow = arow -1;

        end

        [obj, sol ,dsl, bas ,stsl ,tctn] = FMP_SUBQP2C(mr,nc,...
        naij,nqij,pname,spid,aij,arow, acol, qij ,qrow, qcol,...
        upb ,lob, frhs, flhs, cost,mitype ,KOFF,bas);
    end
```

FMP_SOLVE

```
 Usage : [obj, sol ,dsl ,bas ,stsl, tctn] = FMP_SOLVE(A,L,U,l,u,k,c)

 Description: Solves(minimizes) a model entered in canonical form

    Inputs:-
        k   is a scalar constant offset to the objective
        c   is an n- vector of the cost coefficients  c1,c2,...cn.
        x   is an n- vector of the structural variables x1,x2,...xn.
        L   is an m-vector of lower right hand sides L1,L2,...Lm.
        A   is an m x n matrix of coefficients aij.
```

```
         U   is an m-vector of upper right hand sides U1,U2,...Um.
         l   is an n- vector of lower bounds l1,l2,...ln.
         u   is an n- vector of upper bounds u1,u2,...un.
         mitype - see manual

Output - see manual


BIG = exp(40);
[mr nc] = size(A); [arow acol aij]=find(A); naij = length(arow);
upb = u; lob = l; flhs = L; frhs = U; cost = c; nset = 0; sref = [];
sfun = []; sbeg = []; send = []; bas = zeros(1,1+mr+nc);
KOFF =k; pname = 'model'; spid = 'NOSPECS';

tctn=FMP_BLDFMPC;

[obj, sol ,dsl ,bas ,stsl, tctn] = FMP_SUBLP2C(mr,nc,naij,nset,...
aij, pname,spid,arow, acol, upb, lob, frhs, flhs, cost ,mitype, ...
sref, sfun, sbeg, send, KOFF,bas);
```

7.1 Advanced examples

FMP_EFF

This consists of an application that uses four different models detailed below.

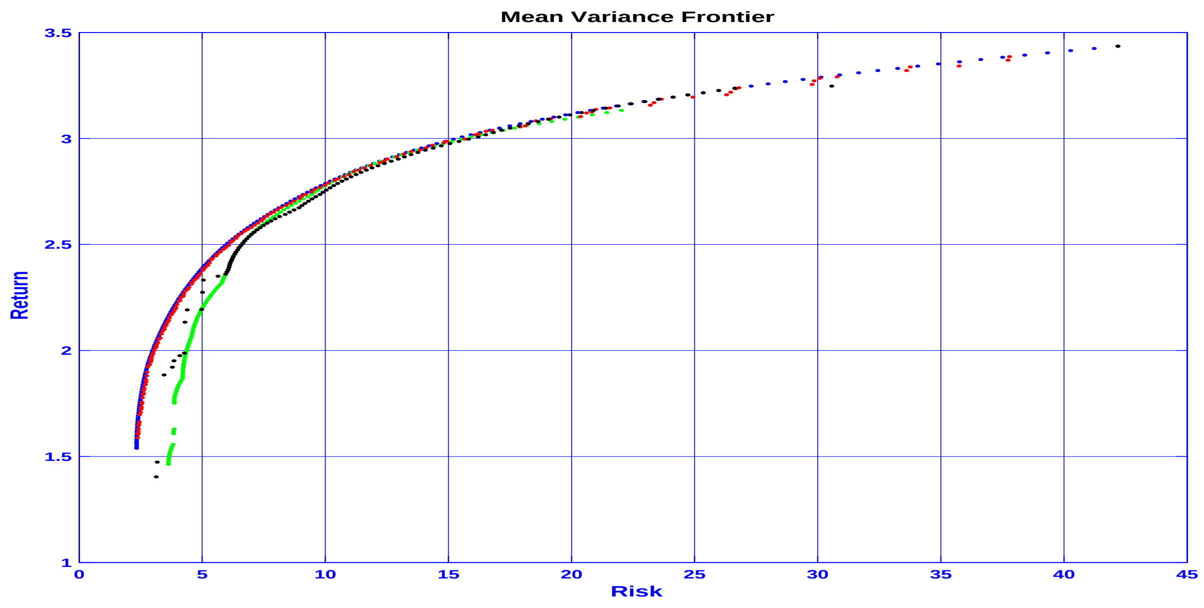

Figure 7.1: Efficient Frontier Comparing Four Models

```
Usage: FMP_EFF
```

```
    Description: Plots the effecient frontier for different models
     Inputs: Requires
         MEF.mps - continuous
        Card.mps - discrete- cardinality
        Lot.mps - Lot size constraints
        Buy.mps - Buy-In constraints
     For more information contact Manti Mendi - see address on manual

[Risk,Ret]=FMP_EFF_(1,'MEF.mps','MEF','NOSPECS');
plot(Risk,Ret,'b.'); clear Risk Ret

hold on; [Risk,Ret]=FMP_EFF_(2,'Card.mps','Card','NOSPECS');
plot(Risk,Ret,'g.'); clear Risk Ret

[Risk,Ret]=FMP_EFF_(3,'Lot.mps','Lot','NOSPECS');
plot(Risk,Ret,'r.'); clear Risk Ret

[Risk,Ret]=FMP_EFF_(4,'Buy.mps','Buy','NOSPECS');
plot(Risk,Ret,'k.'); clear Risk Ret hold off;


function  [Risk,RetOpt]= FMP_EFF_(ModelType,filename,modelname,spid)
%[Risk,RetOpt]= FMP_EFF(ModelType,filename,modelname,spid)
%ModelType = 1 for MEF
%            2 for CARD
%            3 for LOT
%            4 for Buy In

if length(filename) >8
    error('file name is too long  - maximum of 8 characters');
end if exist(filename) ~=2
    errstring=  sprintf('File %s: does not exist',filename);
    error(errstring);
end
tctn = FMP_BLDFMPC; %Initialize fortMP

if(tctn ~= 0)
    error('FMP_BLDFMPC:  FortMP failed to initialize');
end

specs = sprintf('INPUT FILE NAME(%s)',filename);
tctn = FMP_SPECMDC(specs);

pname = modelname;
specs = sprintf('MODEL NAME(%s)',pname);
tctn = FMP_SPECMDC(specs);

if nargin <3
    spid = 'NOSPECS';
end
```

```
[mr, nc, naij, nqij, nset, tctn] = FMP_MP2SIZC(spid);
if (tctn ~= 0)
    error('TCTN is not zero');
end

[mr, nc ,naij, nqij, nset ,aij, arow ,acol, qij, ...
    qrow, qcol, upb, lob ,frhs,flhs ,cost, mitype ,...
    KOFF ,bas, tctn]  = FMP_ReadQPMODEL(filename);

tctn =FMP_SPECMDC('INPUT QMATRIX FULL'); %SPECMDC

if(length(find(cost))==0)
    % FortMP fix- Fallacy in reading QP when objective linear vector is zero
    %So shrink frhs and flhs and reduce mr by 1

    frhs = frhs(2:mr);
    flhs = flhs(2:mr);
    mr = mr-1;
    arow = arow -1;
end

N=100; cnt =1;
switch ModelType
case 1

    Returns = aij(find(arow==1)); % coeffecients corresponding to one form
    %the return vector except for Lots !
    Ret=linspace(max(Returns),min(Returns),N);
    for u = 1:N
        flhs(1)= Ret(u);
        [obj, sol ,dsl, bas ,stsl ,tctn] = FMP_SUBQP2C(mr,nc,naij,nqij,...
        pname,spid,aij,arow, acol, qij ,...
            qrow, qcol, upb ,lob, frhs, flhs, cost,...
            mitype ,KOFF,bas);

        if (stsl ==3)
            RetOpt(cnt)=dot(sol(2+mr:1+mr+nc),Returns);  Risk(cnt)=obj;
            cnt = cnt +1;
        end

    end
case {2,4}
    Returns = aij(find(arow==1)); % coeffecients corresponding to one
    %form the return vector except for Lots !
    Ret=linspace(max(Returns),min(Returns),N);
    for u = 1:N
        flhs(1)= Ret(u);

        FMP_SPECMDC('IPM RESTART ON');
        FMP_SPECMDC('MIP RESTART ON');
```

```
         FMP_SPECMDC('MIP PREP ON');
         FMP_SPECMDC('MIP PRIORITY UP');
         [obj, sol ,dsl, bas ,stsl ,tctn] = FMP_SUBQP2C(mr,nc,naij,nqij,...
         pname,spid,aij,arow, acol, qij ,...
             qrow, qcol, upb ,lob, frhs, flhs, cost,...
             mitype ,KOFF,bas);



         if (stsl ==5)
             RetOpt(cnt)=dot(sol(2+mr:1+mr+nc/2),Returns);  Risk(cnt)=obj;
             cnt = cnt +1;
         end
     end

 case 3
     Lots = aij(find(arow==2));
     Returns = aij(find(arow==1))/Lots(1);
     Ret=linspace(max(Returns),min(Returns),N);

     for u = 1:N
         flhs(1)= Ret(u);

         [obj, sol ,dsl, bas ,stsl ,tctn] = FMP_SUBQP2C(mr,nc,naij,nqij,pname,...
         spid,aij,arow, acol, qij ,...
             qrow, qcol, upb ,lob, frhs, flhs, cost,...
             mitype ,KOFF,bas);

         if (stsl ==5)
             RetOpt(cnt)=dot(Lots(1)*sol(2+mr:1+mr+nc),Returns);
             Risk(cnt)=obj;
             cnt = cnt +1;
         end

     end
end
```

Chapter 8

SPECS Commands for Control of Solver and Data Input

The text-file **'fortmp.spc'** supplies run-time controls - that is SPECS commands - to the FortMP solver and data input subroutines. This section gives an introduction to SPECS command syntax and describes the most important and frequently used commands. Additional commands are described in the FortMP manual and extensions.

SPECS commands may be introduced by using the entries **SPECMD** and **SPECMDC**, which are described earlier. Commands introduced in this way are superseded by commands entered at run-time on the file **fortmp.spc**. There is no difference in syntax, and any command may be entered by either method.

In all the command syntax used below the symbols ' <', '>' are used to bracket an item supplied by the user. Character '/' is used to specify alternatives. Optional items are enclosed by square brackets '[' and ']'.

8.1 Run-time SPECS control file. The 'SPID' parameter

File '**fortmp.spc**' is split into sections permitting different commands to apply in multiple calls to the solver and data entry routines. 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 order to select a particular section the parameter SPID is employed when making a call to any solver or data entry routine. 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 previous settings.
'DEFAULT'	
'ALL'	Has a special meaning and should never be used for SPID

8.2 Most Important SPECS commands

Command	Default
BEGIN [(**<sectid>**)]	**Section ID is blank**
END	

The above commands begin and end each separate *section* of the SPECS file (fortmp.spc). Normally only one section is needed and section identity (with surrounding parentheses) is omitted. Multiple sections are used for systems in which FortMP is called as a sub-system to solve many different models.

Command	Default string
MODEL NAME (<modname>)	**Model**

This command supplies a default for the names of all input, output and local scratch files to be used in the run. An extension is added according to file type.

Command	Default String
INPUT FILE NAME (<filename>)	**<modname>.MPS**
BASIS FILE NAME (<filename>)	**<modname>.BAS**
OUTPUT FILE NAME (<filename>)	**<modname>.RES**
LOG FILE NAME (<filename>)	**<modname>.LOG**

These commands assign a special name to the indicated file, where ¡modname¿ represents the model name. In the case of '**BASIS** ' the filename applies to input only and not to output.

Command	Default String
DIRECTORY (<dirname>)	Empty
PATH NAME (<dirname>)	

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

Command	Default Action
MINIMIZE	**This is the default**
MAXIMIZE	

These commands specify the sense of optimsation.

Command	Default Action
QMATRIX <FULL—HALF>	**Full Q-matrix**

This command specifies whether the Q-matrix is supplied in full, or has only the diagonal entries and one half - the other half being deduced by symmetry. It applies only to library calls SUBQP2 and SUBQP2C, and will not be needed if the model is read previously with INPQP2. INPQP2 sets the command from the header of the quadratic objective section (QMATRIX implies FULL - QDATA and QUADS imply HALF).

Command	Default
Option SCALE [<ON/OFF>]	**ON**

This command determines if scaling is to be performed.

Command	Default
Option PRESOLVE [<ON/OFF>]	**OFF**

This command determines if PRESOLVE is to be executed.

Command	Default Action
ALGORITHM PRIMAL	**This is the default**
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.

Command	Default	Minimum
MAXIMUM SSX ITERATIONS = <intval>	**50000**	**1**
MAXIMUM IPM ITERATIONS = <intval>	**80**	**1**
MAXIMUM MIP <INTEGER SOLUTIONS> /INTSOL> = <intval>	**300**	**1**
MAXIMUM MIP NODES = <intval>	**50000**	**1**
MAXIMUM MIP TIME = <realval>	**50000.0**	**0.0**

These commands specify terminal limits on each major algorithm. When a limit is reached a 'SAVE' is made before exit in order to enable a restart when it is desired to continue the run.

Command	Default	Minimum
SSX SAVE FREQUENCY = <intval>	**10 (each 10th re-inversion) or 0**	**0**
IPM SAVE FREQUENCY = <intval>	**10 (each 10th iteration) or 0**	**0**
MIP SAVE FREQUENCY = <intval>	**500 (each 500th node) or 0**	**0**

These commands specify the frequency for making a SAVE in each major algorithm. Zero implies that no regular SAVE is to occur. Default settings are 10, 10, and 500 when FortMP is used as a stand-alone program, and zero (i.e. no SAVE) when using the library.

Command	Default Option
SSX START RESTART [<ON/OFF>]	**OFF**
IPM RESTART [<ON/OFF>]	**OFF**
MIP RESTART [<ON/OFF>]	**OFF**

These commands specify whether a RESTART is to be invoked for the input or for each major algorithm. The INPUT procedure always saves the lengthy 'MPS' stage output to permit rapid restart . For SSX algorithms 'RESTART' is an alternative to other starting basis options.

Command	Default Action
SSX START CRASH	**This is the default**
SSX START INPUT BASIS	
SSX START UNIT BASIS	

These commands select the mechanism for setting up the starting basis when the main algorithm is PRIMAL or DUAL. 'SSX START RESTART ' can also be used.

In the case of a library call to the solver 'SSX START INPUT BASIS' invokes use of array BAS to provide the starting basis.

Command	Default option
MIP PREPROCESS [< **ON/OFF/ROOT ONLY** >]	**OFF**

This command controls the use of MIP pre-processing. Option 'ON' invokes execution at the root and at every other node in the Branch and Bound tree.

Command	Default option
MIP PRIORITY UP [<**ON/OFF**>]	**OFF**

This command is an important control on the branching in MIP. Option 'ON' causes variable selection to consider fractional values and node selection to choose direction in the UP sense by preference to the DOWN sense. The feature is very useful for models such as allocation, with a large predominance of simple SOS-type constraints.

Command	Default option
GENERATE CUTS [<**ON/OFF/ROOT ONLY**>]	**OFF**

'ON' activates the cut-generation procedures and the application of strong cuts before and during Branch and Bound (See manual section 6.7.3). With ROOT ONLY the feature is invoked only for the root node of the branch and bound tree.

8.3 External Interface Save and Restart commands

FortMP provides a facility for the user to dump the external data interface, together with its model descriptors, onto a file for separate use. This file can act as an input for restarting (saving the time required for model generation), or it can be submitted for testing purposes to the supplier without compromising original source data. Should there be any difficulty in executing the solver, the dump file enables the precise situation to be duplicated, alternatives can be tried out, and the cause of any eventual bug can be investigated. The dump-file can be in binary or in ASCII form.

The dump-file is named <modname>.XFC for binary form, ¡modname¿.XFA for ASCII form, where <modname> is the model name.

The commands for this are given below.

Command	Default option
EXTERNAL SAVE [<**ON/OFF/ASCII**>]	**OFF**

This command activates the save-dump in library routines SUBLP2, SUBQP2, SUBLP2C and SUBQP2C. The file is written immediately after entry and reading the SPECS commands.

Command	Default option
EXTERNAL RESTART [<**ON/OFF/ASCII**>]	**OFF**

This command inputs the model from the save-dump in library routines INPLP2, INPQP2, INPLP2C and INPQP2C.

Bibliography

[1] AMPL. http://www.ampl. 2003.

[2] Gautam Mitra email: Gautam.Mitra@brunel.ac.uk. Lp,mip,qp,qmip,sp lectures and workshop notes. 2003.

[3] FortMP. www.optirisk-systems.com/documents.asp. 2003.

[4] FortSP. http://www.optirisk-systems.com/documents.asp. 2003.

[5] C Lucas. (B) G Mitra, T Kyriakis. *A Review of Portfolio Planning: Models and Systems, (2003) an invited chapter, in:Advances in Portfolio Construction and Implementation, S E Satchell, A E Scowcroft (Eds.) ,pp1-39.* Butterworth & Heinemann, Oxford.

[6] N Koutsoukis B Dominguez-Ballesteros G Mitra, C Lucas. *The SPInE stochastic programming system - To appear in special issue - Application of stochastic programming, editors: S.Wallace, W.Ziemba.* 2004.

[7] G. Mitra (B) I. Maros. *Simplex Algorithms,Chapter 1 in Recent Advances in Linear and Integer Programming,J. Beasley (editor),.* Oxford University Press, 1968.

[8] G. Mitra (J) I. Maros. Strategies for creating advanced bases for large scale linear programming problems. *INFORMS Journal on Computing, USA*, 10:248–260, 1998.

[9] B. Nygreen(J) K. Kularajan, G. Mitra. F.Ellison. Constraint classification, proprocessing and a branch and relax approach to solving mixed integer programming models. *International Journal of Mathematical Algorithms*, pages 1–45, 2000.

[10] C Lucas N Jobst, M Horniman and G Mitra. (J). Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints. *Quantitative Finance*, 1:1–13, 2001.

[11] OPTIRISK. http://www.optirisk-systems.com.

[12] OSP. http://osp-craft.com.

[13] C.A.Poojari G.Mitra. (J) S.A.MirHassani, C.Lucas. An application of lagrangean relaxation to a capacity planning problem under uncertainty. *Journal of Operational Research Society*, pages 1256–1266.

[14] Alan Scowcroft and James Sefton. *Enhanced indexation, (2003) chapter 4, in:Advances in Portfolio Construction and Implementation, S E Satchell, A E Scowcroft (Eds.) ,pp95-124.* Butterworth & Heinemann, Oxford.

[15] Maximal Software. http://www.maximal-usa.com. 2003.

[16] WEBOPT. www.webopt.org/default.asp.