＜新 Xpress8 シリーズ＞

Xpress8 入門書

Getting Started with Xpress

2019 年 4 月

MSI 株式会社
Email: xpress@msi-jp.com
WEB: http//msi-jp.com/xpress
Tel: 043-297-8841
FAX: 043-297-8836

目次 1

まえがき 6
この小冊子が対象とする読者 6
この小冊子を、どのように読むか 7
IVE で MOSEL 言語を使う 7
プログラミング言語環境の使い方 8

第 1 章 イントロダクション 9
1.1 数理計画法 9
1.2 Xpress Product Suite 10
1.2.1 製品バージョンについて 12

第 2 章 モデルを作成する 13
2.1 例題 13

I Getting Started with Mosel 16
第 3 章 線形計画法のモデルをインプットし、そして、解く 16
3.1 Xpress-IVE を起動し、新しいモデルを生成する 16
3.2 LP モデル 18
3.2.1 Mosel プログラムの一般的な構造 18
3.2.2 問題を解く 19
3.2.3 アウトプットの印刷 19
3.2.4 フォーマッティング 19
3.3 エラーを修正し、モデルをデバッグする 20
3.3.1 デバッキング 23
3.4 モデルを解き、最適化についての諸統計類を確認し、解を見る 24
3.4.1 文字列のインデックス付け 26

第 4 章 データ・ハンドリング 28
4.1 ファイルからのデータ・インプット 28
4.2 Xpress-IVE を起動し、新しいモデルを生成する 29
4.3 パラメータ 30
4.4 修正した後のモデル 31
第 5 章 ユーザ定義のグラフを描く 33

MSI 株式会社 Copy right©2015 All rights reserved

5.1 問題の拡張 33
5.2 最適化のループ 33
5.3 ユーザの求めるグラフを描く 34
5.4 これらのフィーチャーを持った最終的なモデルファイル foliodata.mos 36

第 6 章 混合整数計画(MIP) 39
6.1 問題の拡張 39
6.2 MIP モデル 1: 銘柄数を制限する 39
6.2.1 Mosel によるインプリメンテーション 40
6.2.2 解を分析する 41
6.3 MIP モデル 2: 各銘柄の最低投資額の条件を入れる 45
6.3.1 MOSEL で実行する 46

第 7 章 二次計画法 48
7.1 問題の説明 48
7.2 QP 49
7.2.1 Mosel で実行する 49
7.3 MIQP 52
7.3.1 Mosel によるインプリメンテーション 53
7.3.2 解を分析する 54

第 8 章 ヒューリスティックス 57
8.1 バイナリ変数を固定して行うヒューリスティック 57
8.2 Mosel で実行する 58
8.2.1 サブルーチン 60
8.2.2 Optimizer のパラメータと機能 61
8.2.3 トレランスを比較する 63

第 9 章 Mosel モデルをアプリケーションに埋め込む 64
9.1 deployment テンプレートをどのようにして生成するか 64
9.2 BIM ファイル 65
9.3 テンプレートを修正する 66
9.3.1 Mosel モデルの実行 66
9.3.2 パラメータ 66
9.3.3 VB アウトプットの出力先 67
9.4 マトリックス・ファイル 68

MSI 株式会社 Copy right©2015 All rights reserved

9.4.1 マトリックスのエクスポート 68
9.4.2 マトリックスのインポート 68
9.5 Optimization Modeler の開発 70
9.5.1 モデルファイルの準備 70
9.5.1.1 プロジェクト・アーカイブ 71
9.5.2 Optimization Modeler GUI を起動させる 71
9.5.2.1 新規の Optimization Modeler プロジェクトをスタートする 72
9.5.2.2 Optimization Modeler プロジェクトを調査する 73
9.5.2.3 モデルデータを修正・編集する 75
9.5.2.4 データシナリオを比較する 76

Ⅱ Getting started with BCL 78
第 10 章 線形計画問題に入力し、それを解く 78
10.1 BCL でのインプリメンテーション 78
10.1.1 初期化 79
10.1.2 構成 79
10.1.3 モデルを解く 80
10.1.4 アウトプットの印刷 80
10.2 プログラムのコンパイルと実行 80
10.3 ファイルからのデータ・インプット 82
10.4 アウトプット・ファンクションとエラーの扱い 85
10.5 マトリックスのエクスポート 86

第 11 章 混合整数計画法 87
11.1 拡張問題の説明 87
11.2 MIP モデル 1：様々な株の保有数を制限する 87
11.21 BCL によるインプリメンテーション 88
11.2.2 ソリューションを分析する 89
11.3 MIP2： 各銘柄の最低投資額の条件を入れる 91
11.3.1 BCL によるインプリメンテーション 91

第 12 章 二次計画法 94
12.1 問題の説明 94
12.2 QP 94
12.2.1 BCL で実行する 95
12.3 MIQP 98

MSI 株式会社 Copy right©2015 All rights reserved

12.3.1 BCL によるインプリメンテーション 98

第 13 章 ヒューリスティックス 101
13.1 バイナリ変数を固定して行うヒューリスティック 101
13.2 BCL で実行する 101

Ⅲ Getting started with the Optimizer 108
第 14 章 マトリックスのインプット 108
14.1 マトリックス・ファイル 108
14.2 インプリメンテーション 108
14.3 プログラムのコンパイルと実行 109

第 15 章 線形計画法(LP) の問題をインプットし、解く 111
15.1 マトリックス表示 111
15.2 Xpress-Optimizer によるインプリメンテーション 112
15.3 コンパイルと問題の実行 113

第 16 章 混合整数計画法 115
16.1 問題の拡張についての説明 115
16.2 MIP1: ポートフォリオに組み込む銘柄数を制限する 115
16.2.1 マトリックスによる表現 116
16.2.2 Xpress-Optimizer で実行する 116
16.3 MIP2: 各銘柄の最低投資額の条件を入れる 118
16.3.1 マトリックスによる表現 119
16.3.2 Xpress-Optimizer で実行する 119

第 17 章 二次計画法 121
17.1 問題の説明 121
17.2 QP 121
17.3 マトリックス表示 122
17.4 Xpress-Optimizer で実行する 122

Appendix A さらに理解を深めるには 126
A.1 Xpress のインストレーション、ライセンシング、トラブルシューティング 126
A.2 ユーザー・ガイド、参照マニュアル、その他の資料 126
A.2.1 モデル作成 126

MSI 株式会社 Copy right©2015 All rights reserved

A.2.2 Mosel 126
A.2.3 BCL 126
A.2.4 Optimizer 127
A.2.5 その他のソルバーおよび解法 127

Appendix B グロッサリー 128

MSI 株式会社 Copy right©2015 All rights reserved

まえがき

この『Xpress 入門書』という小冊子は、様々な最適化問題を解くために Xpress を使用してどのようにモデル化し、解くかを学ぶための理解しやすい入門書です。この小冊子では、線形問題 (Linear Programming)、混合整数問題 (Mixed Integer)二次計画問題 (Quadratic Programing)を、Mosel 言語を使用したモデリング方法や、Xpress-Optimizer を使用した解法を掲載しております。

上記の「Mosel モデル」を扱うとき、グラフィカル・ユーザ・インタフェース Xpress-IVE を使います。またこの高水準言語を使用する場合には、モデルを定義する方法として、下記に示す 2 通りの代替的な方法があります。一つは、プログラミング言語環境で、モデルビルダーライブラリ Xpress-BCL を使ってモデルを定義する方法とマトリックス形式で Optimizer に直接入力する方法です。

使用法を説明するにあたり、この小冊子の全体で様々な最適ポートフォリオ選択問題の例題として取り上げています。その他の最適化問題は、"Applications of Optimization with Xpress-MP" (Dash Optimization、2002)に掲載しております。

--*----*--*--*--*---*---*--*---*---*---*---*---*--*---*---*---*---*---*---*---*---*--

この小冊子は、Xpress を使用した様々な問題の定式化方法および、解法を掲載しております。小冊子の構成上、基本的な特定機能を取り上げているため Xpress の全機能を説明するものではありません。

<<Xpress 日本配給元 MSI の WEB サイトでは Xpress に関連するホワイトペーパーや各種参考資料、マニュアルを数多く掲載しております。是非、お役立てください>>

>>Xpress 日本配給元 MSI WEB サイト：http://www.msi-jp.com/xpress/

>>Xpress を初めてご使用の方へ：http://www.msi-jp.com/xpress/biginners.html

・Xpress ダウンロード、マニュアル、例題等を掲載したページです。

>>学習広場：http://www.msi-jp.com/xpress/learning/square/

・Xpress 学習に関する様々な資料を網羅したページです。

--*--*--*---*---*---*---*---*---*---*---*---*---*---*---*---*---*---*---*---*---*--

この小冊子が対象とする読者

この小冊子は、様々な Xpress 製品の概要やどのように、グラフィカル環境で使用する高水準言語を使用し定義したモデルの使用法を説明しています。本書はソフトウェアのご評価を希望される方にとって、評価の手始めに理想的な一冊となっております。

本書は、線形計画モデルの作成法から出発し、章を追うごとに新しい機能を追加的に実装していきます。Xpress を初めてお使いの方にとって、本書の説明文を確認しながら、少しずつ Xpress の機能や使用法を学び、モデルを数学的に表現することから始め、アプリケーションにモデルを組み込む

MSI 株式会社 Copy right©2015 All rights reserved

方法が掲載されている章(9 章)まで読み進めたり、ご自身が学びたい章まで必要に応じて本書をご活用頂ければ幸いです。また上級者向けの最適化技術ヒューリスティック法で解を得る方法も本書に掲載しております。(8 章)
この小冊子が取り上げているトピックは多岐にわたるため、Xpress の使用頻度が少ない方にとってもXpress の使用法などを素早く再確認する際に大変便利な内容となっております。

この小冊子を、どのように読むか

Xpress Product Suite を使用したモデル作成や、作成したモデルの解を得る方法のイントロダクションや概要をすべて把握するには、本書のすべてに目を通す必要がありますが、特定のトピックにご関心をお持ちの方は、下記のように本書の一部、または章を抜粋してご使用ください。

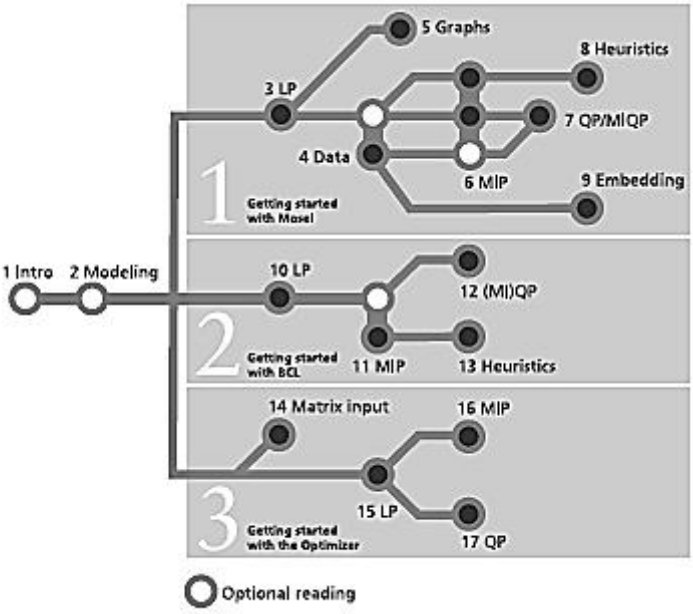


Figure 1: Suggested flow through the book

IVE で MOSEL 言語を使う

本書の最初で示しているアプローチは、問題と解の分析を視覚的に説明しているため、Xpress を初めてご使用の方、数理計画法の初心者や新しいモデルを迅速に開発して現実の場で活用したい方に推奨しているアプローチです。例えば、線形計画法(LP) モデルを開発し、それを何らかの既存アプリケーションに埋め込みたい場合、最初に 4 章を読み、次いで、Mosel モデルを、どのようにしてアプリケーションに埋め込むのかが説明されている 9 章をご確認ください。

Xpress を使用した二次計画(QP) 問題の作成法や解法については、少なくとも 1 章から 3 章、4 章の最初の部分、そして 7 章をご確認ください。また、混合整数二次計画(MIQP) の作成法、解法については、混合整数計画(MIP)を説明している 6 章も併せてご確認ください。ご自身のソリューションアルゴリズムの実行や、ヒューリスティックを Mosel 言語で実装する方法を知りたい場合は、1 章

MSI 株式会社 Copy right©2015 All rights reserved

から 3 章までを読み、次いで、4 章の最初を読んだ後、MIP に関する説明が掲載れている 6 章、それからヒューリスティック関する説明が掲載されている 8 章をご確認ください。

プログラミング言語環境の使い方

プログラミング言語環境でアプリケーション開発を行う場合、2 つのオプションが用意されています。モデルビルダーライブラリ BCL を使用する方法と問題を直接 Xpress-Optimizer に入力する方法があります。

モデル作成のサポートが必要かつツールを選択するにあたりモデルの実行スピードに重点を置く場合、モデルビルダーライブラリ BCL をご使用ください。BCL で定義されるモデルオブジェクトにより生成されるコードは、数学的なモデルと相対的に近いのでメンテナンスが容易になります。

BCL は、LP、MIP、および、QP 問題のモデリングをサポートします。(10- 12 章)BCL のモデルインプットは、13 章の例で示す通り Xpress-Optimizer を直接呼び出して、ソリューションアルゴリズムの定義と結合することができます。Optimizer への直接アクセスは、マトリックス生成ルーチン(LP 、MIP、および、QP 問題についての 15-17 章参照)を持つアプリケーションを使用した低レベルな統合が主となり、外部で生成した標準フォーマット(MPS または LP)(14 章)で与えられたマトリックスを解くために提供されています。Optimizer の特定機能に直接アクセスできるため、LP、MIP、QP 問題の解法に自身のアルゴリズムを使用する研究者や専門家に高い評価を頂いております。

MSI 株式会社 Copy right©2015 All rights reserved

第 1 章 イントロダクション

1.1 数理計画法

数理計画法は、数学的な最適化のテクニックです。製造業、輸送業、テレコミュニケーション、金融業、人員計画などの多様な分野で現実世界の多くの問題が「数理計画問題」として定式化され、利用されています。数理計画問題は、一組の変数、これらの変数の関係を示す制約式、そらに、最大化、または、最小化される目的関数から成っています。数理計画法の問題は、通常、変数、制約式、および、目的関数のタイプに従って分類されます。線形計画法（LP） 問題には、よく知られている効率的なアルゴリズム（シンプレックス法、内点法）が適用できます。このタイプの問題では、すべての制約式、および、目的関数は、変数の一次式で表現され、変数は、連続変数であり、通常、非負の実数なら、どのような値でも取れます。幸い、多くのアプリケーション問題が、このカテゴリーに含まれる問題です。そして、数十万または、数百万の変数と制約式からなる問題が、Xpress-Optimizer のような、市場で販売されている汎用的な数理計画法ソフトウェアによって、ルーチン的に解かれています。しかし、LP を使っている研究者や実務家は連続変数が離散的な性質（すなわち、yes/no、または、1、2、3、…）を表すには不十分であることにすぐに気付き、混合整数計画法（MIP） が開発されましたが、MIP では、LP と同じように、制約式と目的関数は一次式ですが、変数は、離散的な数値、または連続的な数値が取れます。このタイプの問題を解くために、LP のテクニックに、離散的な変数の実行可能値を調べる方法が（分枝限定法）が追加されました。このような枚挙型の方法は、相対的に小さい問題であっても、コンピュータの計算処理量が爆発的に増大するため、MIP 問題を解くとき、常に最適解を求めるのは現実的とはいえません。しかし、近年のコンピュータ・スピードの向上、さらに重要なのは、アルゴリズムが大幅に改善されたこと（例えば、カッティングプレーン・テクニックや特別な分枝法）により、従来よりも大きい問題に取り組むことができ、現実世界の状況を、より厳密にモデル化できるようになりました。比較的、効率的に処理できるもう一つの種類の問題は、二次計画法（QP） の問題です。これらの問題は、制約式は線形ですが、目的関数に二次項を持っています。変数は、連続変数でも離散的な変数でもかまいません。もし、変数が離散的な変数である場合、混合整数二次計画法（MIQP） の問題になります。本書の 7 章、および、12 章で、両方のケースの例を示します。もっとも難しいのは、非線形な制約式や目的関数を持つ問題です。この問題を非線形計画法（NLP） 問題と呼びます。このような問題の良い（部分最適）解を得るには、ヒューリスティックな方法や近似解を得る方法（approximation methods） がよく使われます。このタイプの問題を解く 1 つの方法が、逐次線形計画法（SLP: Successive Linear Programming） で、Xpress-MP suite にこのソルバーが入っていますが、本書では、このトピックについて詳しく説明致しません。モデルを作成し、モデルを解き、そして『解』を実行することは、通常一直線に進む工程ではありません。モデルを作成するときにミスを犯します。そして、ミスの多くは最適化プロセスを行い、アンバウンデッドな解を得たとき、実行不能解を得たとき、または私達の直観と一致しない答えを得たりすることで、初めて発覚します。このような解を得た場合、モデルを良く調べ、モデル内のミスを正し、モデルを再度解き、得た最適解をさらに詳しく分析するという繰返し作業が必要です。このプロセス

MSI 株式会社 Copy right©2015 All rights reserved

中に、制約式を追加したり、または誤って入れてしまった制約式を削除したり、誤ったデータを訂正したり、以前は必要と考えていなかった新しいデータを集めたる必要性も出てくるかもしれません。

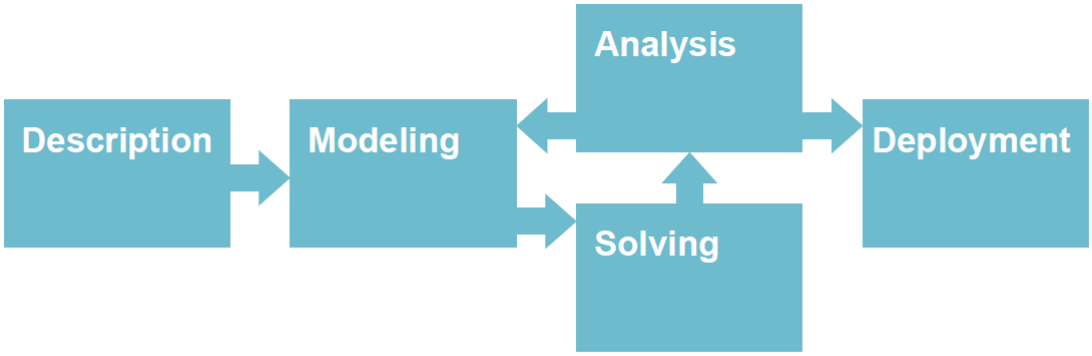


Figure 1.1: Scheme of an optimization project

この小冊子を読み進むと、読者はテキスト形式の記述から数理計画モデルを作成し、解を得るまでの全ステップを経験できます。以下の章では、結果の分析をサポートする方法のイントロダクションを含め、様々な改善方法やモデルへの追加、モデルの再定式化などについて学びます。一般的に数理計画法のアプリケーション開発とは、開発したモデルを他のアプリケーションに組込み、会社の情報システムの一部にする作業も含まれます。

1.2 Xpress Product Suite

様々なユーザの多様なニーズやお好みに対応するため、Xpress 製品 Suite には、モデルを作成し、最適化を行うためのツールがいくつかあります。

1. 高級言語

Xpress-Mosel 言語により、ユーザは、数学の数式表記に近い形式でモデルを定義でき、またそれを使って同一環境で解を得られます。Mosel のプログラミング機能であるこの高級言語の中で、ソリューションアルゴリズムを、直接実行することができます。Mosel は、スタンドアロンとして使えますしまたその他、多くのツール、特にグラフィカルなディスプレイを提供する Xpress-IVE 開発環境を通して使うこともできます。Mosel 環境は、「モジュール概念」に基づいて構築されているので、Mosel 環境への追加が認められています。 FICO 社から提供されているモジュールには、Xpress-Optimizer（LP 、MIP、QP）、Xpress-SLP、（例えば、ODBC 経由による）データ・ハンドリング機能、システム機能へのアクセスなどがあります。さらに、Mosel ネイティブインタフェース経由で、ユーザは必要に応じて（例えば、問題固有データの処理を行う、外部のソルバーや、ソリューションアルゴリズムと接続するなどの）ユーザ自身のモジュールを定義することで新しい機能を、Mosel 言語に追加することができます。

MSI 株式会社 Copy right©2015 All rights reserved

2. 埋め込みライブラリ

数理計画モデルを大規模なアプリケーションに埋め込む方法に、2 つのオプションがあります。最初のオプションは、Mosel 言語を使って開発したモデルを、Mosel ライブラリを通して、プログラミング言語環境（例えば、C、C++、Java など）から実行し、アクセスします。また、一部のモジュールでは、プログラミング言語環境から、それらの機能に、直接、アクセスすることもできます。2 番目のオプションは、モデルビルダーライブラリ Xpress-BCL のサポートを受け、プログラミング言語の中で、モデルを直接、開発する方法です。BCL により、ユーザは専門のモデリング言語と類似したオブジェクト（変数、制約式、インデックスセット）を使ってモデルを定式化できます。すべてのライブラリが、C、C++、Java、および、Visual Basic（VB）で利用可能です。

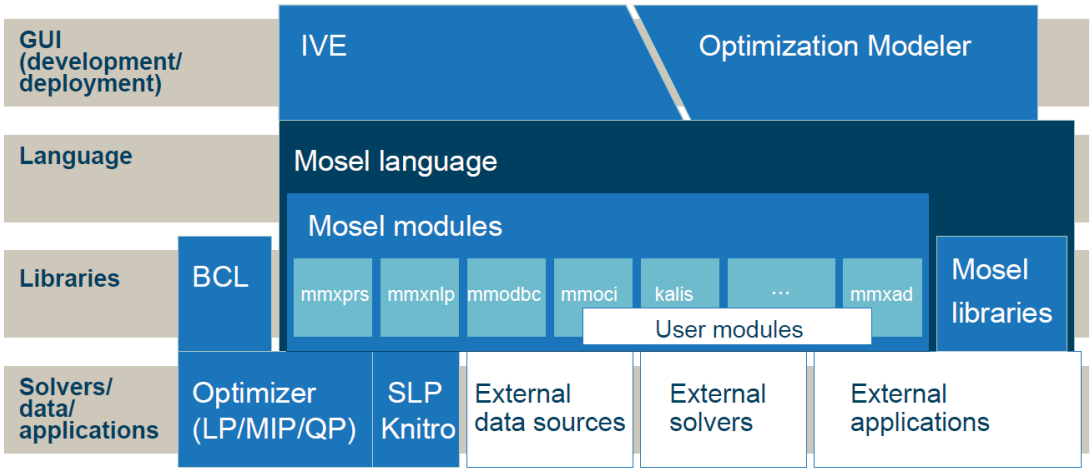


Figure 1.2: Xpress product suite

3. ソルバーへダイレクトにアクセス

一番低いレベル、すなわちもっとも直接的なレベルでは、ライブラリという形式、または、スタンドアロンで、Xpress-Optimizer、Xpress-SLP に、直接アクセスできます。この機能は、Optimizer を専門のマトリックス生成ルーチンを持つアプリケーションに埋め込む際に」便利です。Xpress の上級ユーザは、例えば、以前に Mosel や BCL で生成したマトリックスを使って、いろいろなインタフェースを通しては利用できない Optimizer の特別な機能を使いたい場合があるでしょう。

上記の 3 つアプローチで、高水準言語は、確かに数理計画法への最も理解しやすいアクセスでしょう。したがってこの小冊子では、Xpress-Mosel 言語を使った問題の定義方法、その問題の求解法や Mosel ライブラリを使い、作成されたモデルをアプリケーションに埋め込む方法を重点的に説明していきます。以下では、Mosel モデルをベースに、グラフィカル・ユーザ・インタフェース Xpress-IVE を使い、Xpress-IVE の持つ機能を有効活用してデバッグ、解の分析、ディスプレイする方法を説明します。

この小冊子では、直接プログラミング言語環境のなかで、どのように数理計画法問題を定式化し、そして解くかについても説明していきます。この説明は BCL、または直接 Xpress-Optimizer ライブラリ

MSI 株式会社 Copy right©2015 All rights reserved

を使用したモデル作成サポートを利用します。BCL では、モデルは、その数学的な表現に近い形式で定式化されるため理解しやすく、メンテナンスしやすいものとなります。この小冊子では、Mosel で使われるのと同じ問題を使って、BCL の使い方について説明します。この小冊子の最後の部分で、（例えば、Mosel や BCL などの、別のツールにより生成される）ファイルのマトリックス形式の問題やアプリケーションプログラムにあるマトリックスを、どのように、直接 Optimizer に入力するかを説明します。また、Optimizer ライブラリを直接、使う機能はモデルをアプリケーションに埋め込む方、Xpress の上級ユーザ向けです。数理計画法の初心者にはお勧めできません。

1.2.1 製品バージョンについて

この冊子に掲載されている Mosel の例題は、Xpress7.7 を使用しています。（Mosel 3.6.0）IVE の画像はリリース 7.0.1 を使用しています。（IVE 1.20.05）モデラーの例題は、Xpress7.8 で開発しています。BCL の例題は Xpress7.2 で提供している BCL4.40 で開発しています。Optimizer の例題は Xpress7.1(Optimizer 21.0.3)で開発しています。例題をその他製品バージョンで実行した場合、アウトプットが異なる場合があります。特に、アルゴリズムの改良や Xpress-Optimizer のデフォルト設定の修正が LP サーチの動きや MIP 分枝限定法の細部がが異なる場合があります。IVE インターフェースは次回のリリース版で新しい機能が追加されるため変更点が若干ありますがこの冊子に掲載されている動作には影響がありません。

MSI 株式会社 Copy right©2015 All rights reserved

第 2 章 モデルを作成する

この章では、記述的に書かれた現実の世界の問題を、どのように数学モデルに変換するかについて、詳細に説明します。ここで使用する例題：最適ポートフォリオ選択問題は、この小冊子全体で使用していく問題です。

数理計画法に関する経験値は必要ありませんが、読者が未知の量を表すのに使用する x や y などの記号の使用や、これらの変数を使う簡単な一次式や一次不等式に慣れていることを前提とします。例えば、

$$x + y \leq 6$$

は、「x で示されている量と y で示されている量の合計は、6 以下であるか、6 でなければならない」ということを示しています。また、読者は、「一組の変数の合計」という考え方にも慣れている必要があります。例えば、producei という表現で製品 i の生産量が示されるとするなら、ITEMS という集合のすべての品目の全体の生産量は、下記のように表現されます。

$$\sum_{i \in ITEMS} produce_i$$

これは、「集合 ITEMS の中のすべての製品 i の生産量を合計する」ということを意味しています。

この小冊子でよく使われるもう一つの数学記号は、∀という全称記号（「集合に属するすべてにたいして」と読む）です。例えば、仮に、ITEMS が要素 1、4、7、9 から成っているとすると、

$$\forall i \in ITEMS : produce_i \leq 100$$

上の表現は、下記の不等式全体の省略表現となります。

$$produce_1 \leq 100$$
$$produce_4 \leq 100$$
$$produce_7 \leq 100$$
$$produce_9 \leq 100$$

上記のような数学的表記法の使用を拒まないなら、モデル作成言語を使うことは簡単です。

2.1 例題

ある投資家は、一定の金額を投資したいと考えています。彼は、投資対象として 10 種類の証券（株式）を評価しています。彼は、1年間での投資リターンを評価しようとしています。下の表は、個々の株がどこの国の株か、またリスクカテゴリー(R：高いリスク、N： 低いリスク)を示し、さらに予想収益率（ROI）を示しています。投資家は投資についてある方針を持っています。リスク分散に、どの株に投資してもよい額は最大で資本金の 30%です。さらに彼は少なくとも資本金の 50%は北米の株式に投資し、最大三分の一はハイリスクの株に投資したいと考えています。最大の予想リターンを得るた

MSI 株式会社 Copy right©2015 All rights reserved

めには、資金をどの株式に、それぞれ、いくらずつ投資したらよいのでしょうか。

Table 2.1: List of shares with countries of origin and estimated return on investment

Number	Description	Origin	Risk	ROI
1	treasury	Canada	N	5
2	hardware	USA	R	17
3	theater	USA	R	26
4	telecom	USA	R	12
5	brewery	UK	N	8
6	highways	France	N	9
7	cars	Germany	N	7
8	bank	Luxemburg	N	6
9	software	India	R	31
10	electronics	Japan	R	21

数学的モデルを作成するために、まず、最初に、ソリューションを得るために必要な決定が何であるかを識別しましょう。このケースでは、各株式をいくらずつ、ポートフォリオに入れたらよいか、ということです。したがって、意思決定変数$frac_s$を定義します。この変数は、各株式に投資される資金を示します。ということは、これらの変数は 0 から 1 までの間の値を取ることを意味します（したがって、1 は投下資本の全体を意味します）。実際、すべての変数は、投資家が銘柄ごとに投資しようとしている最大金額で制限されています。

$$\forall s \in SHARES : 0 \leq frac_s \leq 0.3$$

すなわち、最大でも投資額の 30%です。こうして、下の制約式は、変数$frac_s$の取りうる値を制限します。（「SHARES のすべての s に関して･･･」と読んで下さい）。この問題の数学的な定式化の中で、SHARES により、投資家が投資の対象としている銘柄の集合を表し、RETs により、銘柄ごとの 1 株あたりの予想 ROI の集合を表します。また NA は、北米銘柄の部分集合を表し、RISK はハイリスク銘柄の集合を表します。投資家は、用意している資本を、すべて、投資したいと思っています。すなわち、いろいろな銘柄に投下される資金割合を合計すると 100%にならなければなりません。このことを下の等式で表現します。

$$\sum_{s \in SHARES} frac_s = 1$$

続いて、投資家が決めた二つの方針を表現する必要があります。すなわち、最大でも、ハイリスク銘柄に投資する金額は、全体の 1/3 と決めていますので、以下の制約式を得ます。

$$\sum_{s \in RISK} frac_s \leq 1 / 3$$

また、この投資家は、少なくとも投資する金額の 50%は北米銘柄に投資する方針があるので、以下の制約式を得ます。

$$\sum_{s \in NA} frac_s \geq 0.5$$

これら二つの制約式は不等式の制約式です。投資家の目的は、すべての銘柄への投資リターンを最大化することです。すなわち、下記の値を最大化することです。

$$\sum_{s \in SHARES} RET_s \cdot frac_s$$

MSI 株式会社 Copy right©2015 All rights reserved

これが、この数学モデルの目的関数です。上の制約式、および、目的関数をまとめると、下記のようなこの問題の数学的モデルの定式化が得られます。

$$\text{maximize} \sum_{s \in SHARES} RET_s \cdot frac_s$$
$$\sum_{s \in RISK} frac_s \leq 1/3$$
$$\sum_{s \in NA} frac_s \geq 0.5$$
$$\sum_{s \in SHARES} frac_s = 1$$
$$\forall s \in SHARES : 0 \leq frac_s \leq 0.3$$

次の章ではこの数学的なモデルを Mosel モデルに変換し、Xpress-Optimizer で解く方法を説明します。また、第 10 章では、この目的のための BCL の使用法、第 15 章では、モデル作成サポートを使わずに、このモデルを Optimizer に直接入力する方法を説明します。

MSI 株式会社 Copy right©2015 All rights reserved

Ⅰ Getting Started with Mosel

第 3 章 線形計画法のモデルをインプットし、そして、解く

この章では、第 2 章で定式化した例題の Mosel モデルへの変換方法、モデルを Optimizer に直接入力する方法を説明します。

- Xpress－IVE を起動する
- Mosel ファイルを作成し、保存する
- Mosel 言語を使い、モデルを入力する
- エラーを訂正し、モデルをデバッグする
- モデルを解き、IVE の LP 最適化ディスプレイを理解する
- 解を見て、確認し、現実世界の問題という観点から、その妥当性を検証して、正しいかどうかを理解する。

第 10 章では、同じ例題を、BCL を使ってどのように定式化し、解くかを、また、第 15 章では、Xpress-Optimizer に、直接、問題は入力して解く方法について説明します。

3.1 Xpress-IVE を起動し、新しいモデルを生成する

これから、Xpress-IVE のグラフィカル環境で、Mosel モデルを開発し、実行してみましょう。
Xpress-IVE の標準インストールが済んでいればデスクトップのアイコンをダブルクリックもしくは、
Start ≫ *Programs* ≫ *FICO* ≫ *Xpress* ≫ *Xpress-IVE* の順で進み、プログラムを起動させてください。また、DOS ウィンドウに ive とタイプ入力するか、または PC 内にモデルがある場合はモデルファイル（拡張子.mos を持つファイル）をダブルクリックするとプログラムが起動します。起動後、Xpress-IVE はいくつかの枠に分割されているウィンドウが表示されます。

MSI 株式会社 Copy right©2015 All rights reserved

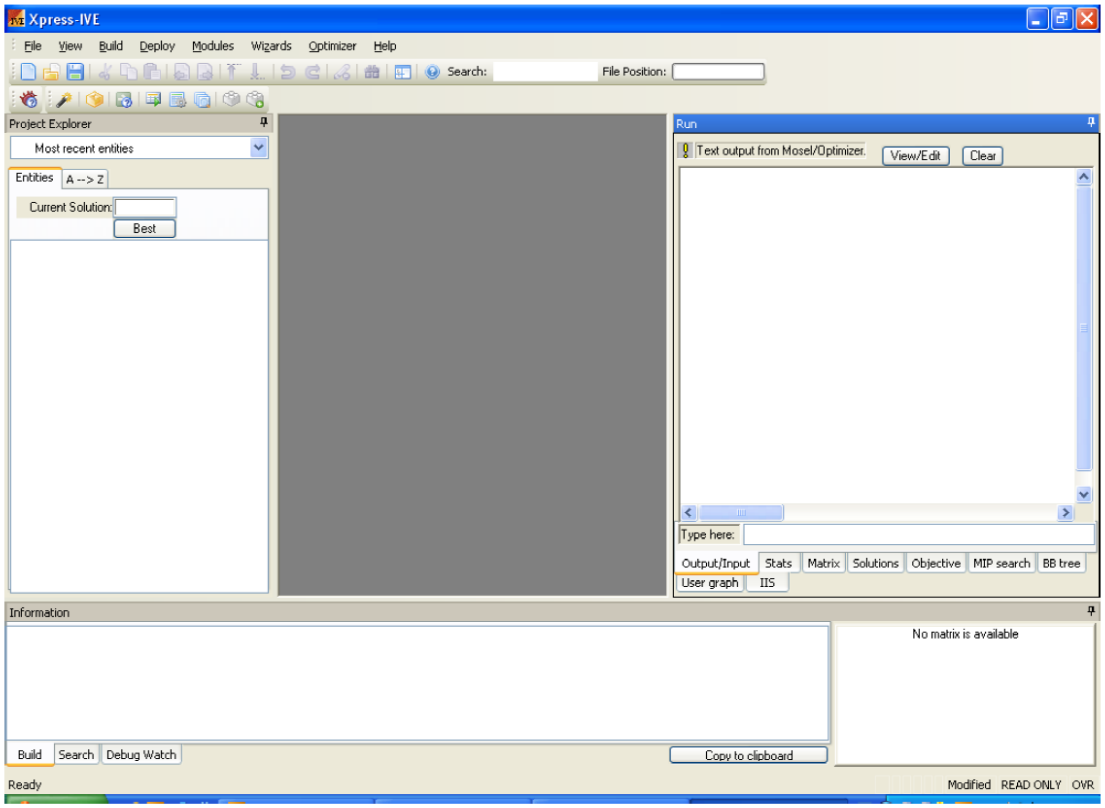


Figure 3.1: IVE at startup

このウィンドウの上部には、メニューバーとツールバーがあります。左側のウィンドウは poject bar であり、底部には info bar が、そして、右のウィンドウは run bar です。これらの bar は、View メニューを使ったり、トグルボタンを使ったりして調整できます。元の表示に戻るには View>>Repair Window Layout を選択するか restore ボタン : . を選択します。真ん中の灰色のウィンドウは、ワーキングファイルが表示される場所です。

新しいモデルファイルを作成するには、File >>New を選択するか、ツールバーの最初のボタン をクリックして下さい。これにより、ダイアログウィンドウが開き、ここでディレクトリを選び、新しいファイルの名前を入力します。ここでは、新しいファイルの名前として、foliolp という名前を入力しましょう。Mosel ファイルの拡張子.mos は自動的に付加されます。選択を確定するために、Save をクリックして下さい。こうすると、IVE の真ん中のウィンドウは、灰色から白に変わり、カーソルがその最上段に置かれ、モデルを入力する準備が整いました。

MSI 株式会社 Copy right©2015 All rights reserved

3.2 LP モデル

前章の数学的モデルは、IVE では、以下のような Mosel モデルに変換できるでしょう。

```
model "Portfolio optimization with LP"
 uses "mmxprs"                        ! Use Xpress-Optimizer

 declarations
  SHARES = 1..10                      ! Set of shares
  RISK = {2,3,4,9,10}                 ! Set of high-risk values among shares
  NA = {1,2,3,4}                      ! Set of shares issued in N.-America
  RET: array(SHARES) of real          ! Estimated return in investment

  frac: array(SHARES) of mpvar        ! Fraction of capital used per share

    end-declarations

    RET:: [5,17,26,12,8,9,7,6,31,21]

   ! Objective: total return
    Return:= sum(s in SHARES) RET(s)*frac(s)

   ! Limit the percentage of high-risk values
    sum(s in RISK) frac(s) <= 1/3

   ! Minimum amount of North-American values
    sum(s in NA) frac(s) >= 0.5

   ! Spend all the capital
    sum(s in SHARES) frac(s) = 1

   ! Upper bounds on the investment per share
    forall(s in SHARES) frac(s) <= 0.3

   ! Solve the problem
    maximize(Return)

   ! Solution printing
    writeln("Total return: ", getobjval)
    forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")

  end-model
```

ここでたった今、上に書いたことが「何を表現しているか」を理解しましょう。

3.2.1 Mosel プログラムの一般的な構造

どの Mosel プログラムも、キーワード model から始まり、その後ろに、ユーザが選択したモデル名が続きます。そして、Mosel プログラムは、キーワード end-model で終わります。

すべてのオブジェクトは、それらがアサインメント・ステートメントにより、明確に定義されない限り、declarations セクションで宣言されなければなりません。例えば、

```
Return:= sum(s in SHARES) RET(s)*frac(s)
```

は、Return が線形の制約式であり、それに、下の内容が割り当てられています。

MSI 株式会社 Copy right©2015 All rights reserved

```
sum(s in SHARES) RET(s)*frac(s)
```

- SHARES は、いわゆる range set－ すなわち、ここの連続する整数の集合（このケースでは、1 から 10 まで）
- RISK と NA は、簡単な整数の集合
- RET は、セット SHARES によってインデックスを付けた実数の配列であり、その値 は declarations で割り当てられる
- frac は、セット SHARES によってインデックスを付けられる mpvar という変数の配列。これらはこのモデルの決定変数。

モデルは、次いで、目的関数、2 つの線型の不等式の制約式、1 つの等式の制約式、および、変数の取り得る数値の上限（アッパー・バウンズ）を定義します。数学モデルの中で、forall ループを使用したのと同じように、集合 SHARES のインデックス付けに forall ループを使用します。

3.2.2 問題を解く

maximize というコマンドにより、線形の目的関数 Return の値を最大化するために、Xpress-Optimizer が呼ばれます。Mosel 自身はソルバーではないので、モデルの最初で、下記のステートメントにより、Xpress-Optimizer を使うことを指定します（モジュール mmxprs の説明は、「Mosel 言語の参照マニュアル」に掲載しています）。

```
uses "mmxprs"
```

目的関数 Return を定義する代わりに、次のように書いても同じです。

```
maximize(sum(s in SHARES) RET(s)*frac(s))
```

3.2.3 アウトプットの印刷

Mosel プログラムの最後の 2 行により、変数すべての最適解と解の値が印刷されます。
空のラインを追加プリントするには、単純に、（argument なしで）writeln とタイプして下さい。単一のラインにいくつかのアイテムを書くには、writeln ではなく、write を使って下さい。

3.2.4 フォーマッティング

より読みやすくするために、モデル内での字下げ、スペース、および、空行を追加しました。これらは Mosel では無視されます。
改行：下のように、セミコロンで分離することで、いくつかのステートメントを単一のラインに書くことができます。

```
RISK = {2,3,4,9,10}; NA = {1,2,3,4}
```


MSI 株式会社 Copy right©2015 All rights reserved

しかし、特別な「ライン終わり」を示したり、継続を示す文字はありませんので、複数の行に跨るステートメントのすべての行は、演算子（+ 、>、=など）、または、’ ,’のような文字で終わらせ、そのステートメントが終わっていないことを明確にします。また、上のプログラム例に見るように、Mosel でコメント行を入れるには、先頭に、！を入れます。複数行のコメント行を入れるには、！ではじめ！で終わらせます。

3.3 エラーを修正し、モデルをデバッグする

前のセクションでモデルを入力したので、それを実行しましょう。実行とは、すなわち,この最適化問題を解き、結果を取り出すことです。Build >>Run を選択するか、runbutton をクリックして下さい。最初にモデルのランを行うとき、おそらく、`Compilation failed. Please check for errors.’ というメッセージが出てくるでしょう。例えば、下の Figure 3.2 に示すように、一番下のウィンドウに、Mosel により生成されたエラーメッセージが表示されます。

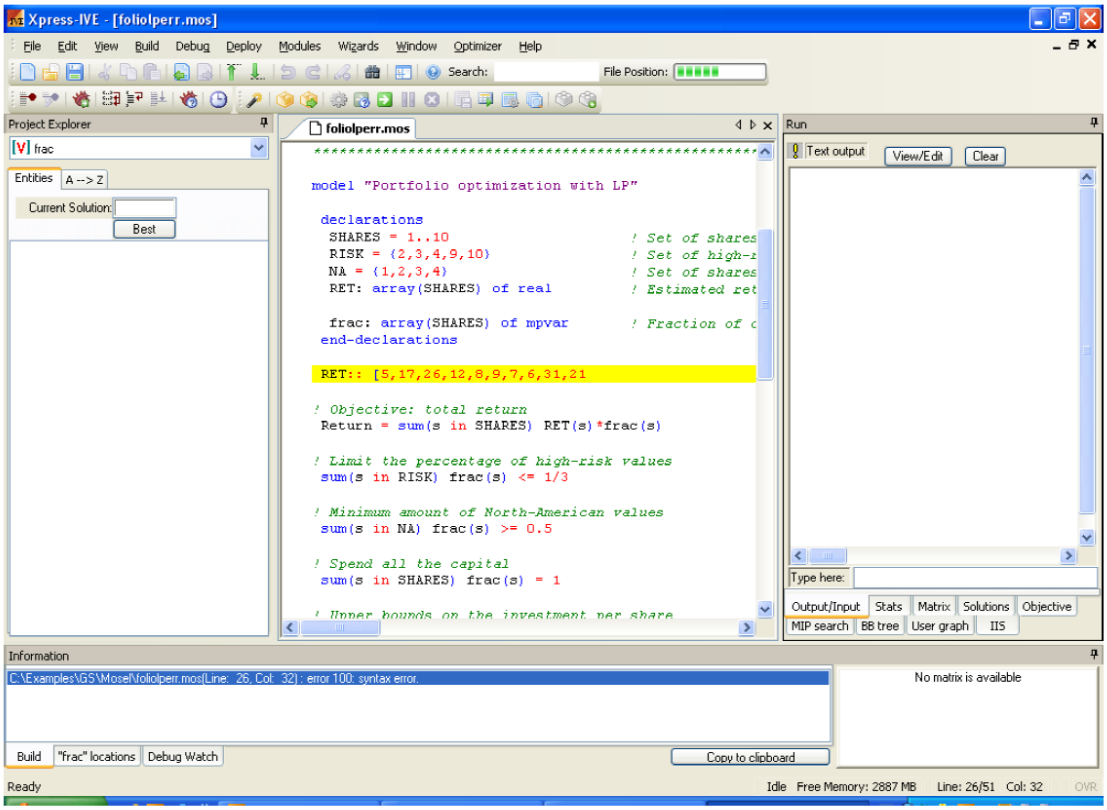


Figure 3.2: Info bar with error messages

前のセクションでリストされているモデルは、正しく入力されたものが表示されていますが、このモデルを入力する時、意識的に間違えて入力してみて下さい。例えば、RET から始まる行の最後の ‘] ‘ を入力しなかったとします。そうすると、それに対応するエラーメッセージが出てきますので、それをクリックすることにより、モデルに対応したラインがハイライトされます。この場合、出てくるエラーメッ

MSI 株式会社 Copy right©2015 All rights reserved

セージは下記の通りです。

```
error 100: syntax error
```

このメッセージをクリックすることで、シンタックス・エラーのある下記の行がハイライトされます。

```
RET:: [5,17,26,12,8,9,7,6,31,21
```

この例では、明らかに閉じ括弧を追加して、RET の定義を完結する必要があります。
定義が次の行に跨ぐ場合は、この行の最後の位置にコンマを追加し、定義が次の行に続くことを明示します。次に、下記のようなメッセージが出てきたとします。

```
warning 121: Statement with no effect
```

ここをクリックすると、

```
Return = sum(s in SHARES) RET(s)*frac(s)
```

という行がハイライトされます。
warning は、モデルの実行を妨げませんが、Mosel コンパイラは、このラインには意味がないと言っているので、なにかが間違っているにちがいありません。ここでエラーを見つけるのには、非常に注意深く書かれていることを調べる必要があります。そうすると、ここでは、‘ := ‘ ではなく、’ = ‘ が使われていることがわかります。Return は、右辺の合計に対応すると定義されなければならなかったので、このステートメントは意味がないと判定されたのです。このエラーを訂正した後に、再び、モデルを実行せようとした場合、次のエラーメッセージの修正を忘れていたので、下記のメッセージが出てきました。

```
error 123: ‘maximize’ is not defined
```


MSI 株式会社 Copy right©2015 All rights reserved

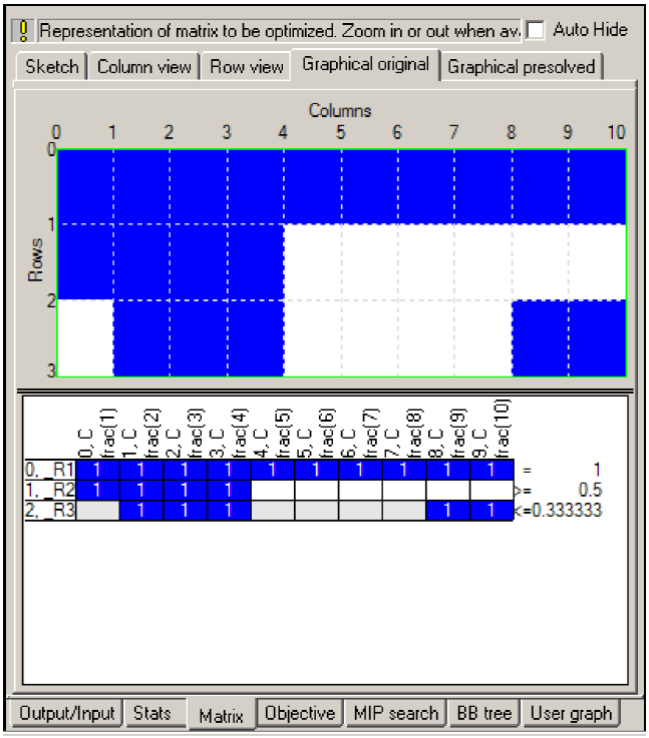


Figure 3.3: LP matrix display

上記のエラーは下記の行にあります。

```
maximize(Return)
```

maximizeという手順は、モジュールmmxprsによって定義されています。しかし、実は、Moselモデルの最初に、下記の行を入れるのを忘れていました。これを追加すると、モデルは正しくコンパイルされ、エラーメッセージは出てこなくなりました。

```
uses "mmxprs"
```

モジュールから何が提供されているかを簡単に知るために、モジュールブラウザという機能があります。これを利用するには、Modules >>List available modulesと選択するか、 ボタンをクリックして下さい。これにより、一つのウィンドウが開きますが、そこには、インストールされているXpressで利用可能なモジュールがリストされており、また個々のモジュールによりMosel言語に追加されている機能(サブルーチン、タイプ、定数など)を詳細にチェックすることができます。

ユーザは、モデルをタイプ入力しているときに、Moselキーワードの正しい名前を思い出せないときには、IVE エディタのコード完成(code completion)機能を使えます。これには、CRTL とスペースバーキーをホールドダウンして、キーワードのリストを出し、必要なものを選択して下さい。

MSI 株式会社 Copy right©2015 All rights reserved

3.3.1 デバッキング

ユーザがモデルに実行して欲しいことを、モデルが実際に実行してくれるか、どうかを保証することにはなりません。例えば、データの初期設定を忘れたとか、または変数と制約式が意仮に、モデルが Mosel の観点から文法的に正しいとしても、当然のことですがこのような理由から、Mosel により生成されたものが何であるかを、十分マトリックスをチェックする必要があります。こうして、モデルを実行して解く前に、一時、先に進むのを休止します。ここで、Build >>Options を選択するか、run options button をクリックして下さい。表示されるダイアログウィンドウの中で Pause のところの view matrix をチェックし、Apply で確認して下さい。こうしておくと、モデルのランを実行すると、ランを実行する前に、IVE workspace の右側の run bar で、マトリックスが表示されます。

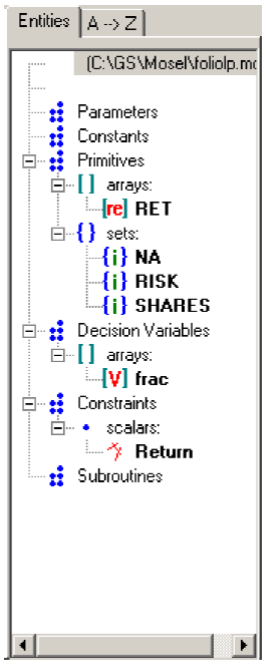


Figure 3.4: Entity display

上の部分に、マトリックスの一般的な構造が示され、そして、下の部分では、単一の係数にズームインでき、その値が示されます。このディスプレイは、正の係数を青で示し、負の係数は赤、0 の係数は、白、または、灰色で表示します。このモデルでは、3 つの制約式が示されていますね。ここでは、常にマトリックスの制約式は、モデルでの定義の順番の逆になっているので、気を付けて下さい。上で示したマトリックスディスプレイでは、そのままでは、Mosel モデルで与えた変数名は式のモデルで使った名前を Optimizer にロードすることができます。それには、Optimize を呼ぶ前に(すなわち、maximize (Return)の前に)、下記の行を追加します。モデルのランを実行する前に、モデルに小さな追加を行うと、変数と制約式のモデルで使った名前を Optimizer にロードすることができます。それには、Optimize を呼ぶ前に(すなわち、maximize (Return)の前に)、下記の行を追加します。
setparam("XPRS_LOADNAMES", true)

こうすると、マトリックスディスプレイは、モデルで使った名前を表示するようになります。こうすると、ス

MSI 株式会社 Copy right©2015 All rights reserved

クリーンディスプレイの中で、もう一つ、別の変化が起こります。今度は、workspace の左のウィンドウを見て下さい。そこには、モデルで定義されたすべてのエンティティが表示されています。そこで、’+‘ サインをクリックすると、すべての情報を見ることができます。
カーソルをいろいろなエンティティ上をゆっくり動かすと、その時点での、それらのエンティティの内容、およびステータスが表示されます。これにより、ユーザは、例えば、インデックスセットやデータ配列の内容をチェックし、配列 frac は、実際に、10 個の変数から成っているということが確認できます。
ユーザは、またエンティティをダブルクリックして新しいウィンドウを出し、エンティティの値の完全な内容を見ることもできます。これは、配列が大きいときに、特に便利です。さらにエンティティをクリックすると、infobar に、エディタで記述された文が示されます。
モデルの実行中にエンティティの値などを確認するには、IVE debugger を使います。
Debug>>Start/Continue を選択し、デバックのスタートや停止は ボタンをクリックしてください。
デバックメニュー内のデバックボタンでは標準的なデバック作業(例えば、ブレイクポイントの設定、カーソル位置でモデルを実行する、またはモデルをステップ・バイ・ステップで実行する)が行えます。
モデルの実行に進むには、pause button をクリックして下さい。また、モデルの実行を止めるためには、stop button をクリックして下さい。

3.4 モデルを解き、最適化についての諸統計類を確認し、解を見る

前のセクションで説明したように、モデルを実行するには、Build >>Run を選択するか、
run button をクリックする必要があります。実行が成功すると、スクリーンディスプレイは、Figure3.5 のように変わります。

```
Total return: 14.0667
1: 30%
2: 0%

3: 20%
4: 0%
5: 6.66667%
6: 30%
7: 0%
8: 0%
9: 13.3333%
10: 0%
```


MSI 株式会社 Copy right©2015 All rights reserved

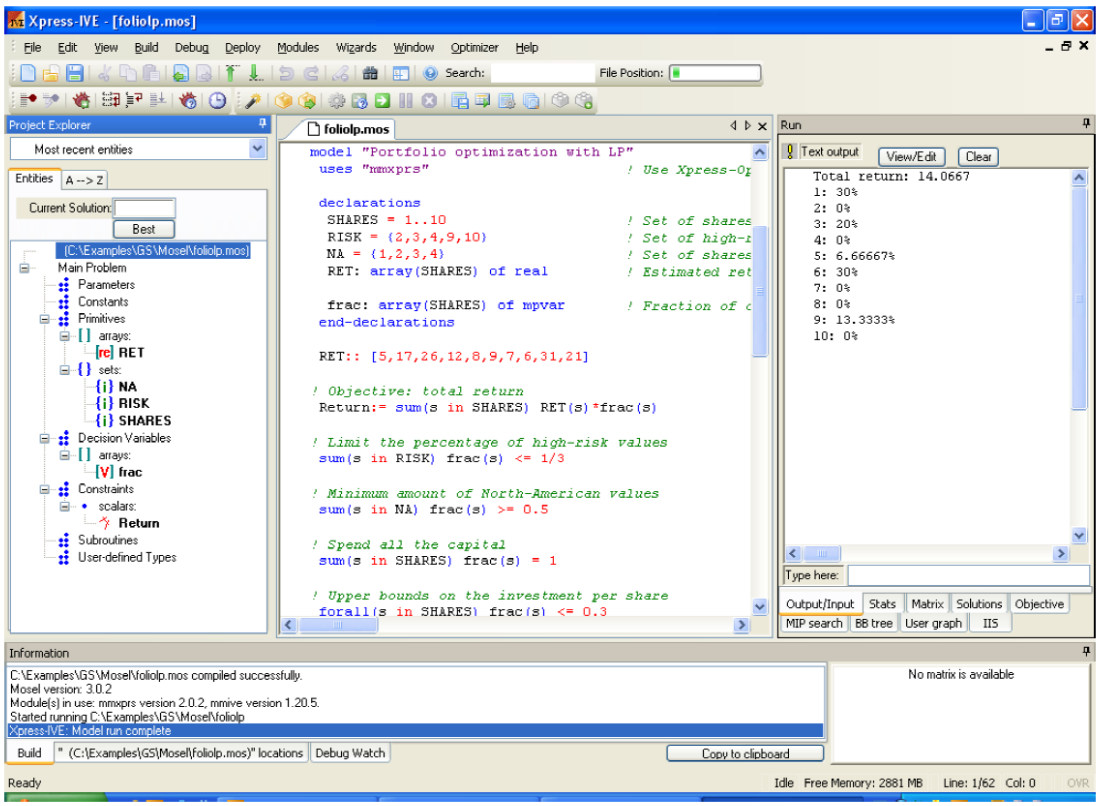


Figure 3.5: Display after model execution

この解は、ポートフォリオは銘柄 1、3、5、6、および、9 からなり、そのとき、最大のリターンと銘柄 6 に各 30%、銘柄 3 に 20%、銘柄 9 に 13.3333%、銘柄 5 に 6.6667%が使われます。ここで、すべての制約式が、実際に満たされていることは容易に確認できます。この解では、確かに 50%が北米銘柄に投資されており(銘柄 1 と銘柄 3)、33.33%がハイリスク銘柄です。(銘柄 3 と銘柄 9)

ここで、Stats タブを選び、Figure3.6 の詳細な LP optimization information を得ます。

ウィンドウの上の部分は、オリジナル、および、presolve 後のマトリックスについての統計が示されます。ここで、presolve とは、マトリックスを簡略化したり、変換したりするいくつかの数値的な操作を適用することを意味します。中央部分は、どの LP アルゴリズムが使われたかを(この例では、Simplex 法)、また、イタレーションの数、そして、このアルゴリズムが要した時間の長さを示します。下の部分は、IVE の様々なディスプレイにより必要とされたオーバーヘッド時間をリストします。この問題は非常に小さいので、ほぼ即座に解かれました。この問題では、「制約式」の項に目的関数のみがリストされていることに注意して下さい。

モデルによりプリントされるものよりも、さらに詳細なソリューション情報を見るには、ウィンドウのエンティティディスプレイに行き、’ + ‘サインをクリックして、変数と制約式を開いて下さい。いろいろなエンティティの上でカーソルをゆっくり動かすと、すべてのソリューション情報が表示されます(大きい配列の場合は、エンティティをダブルクリックして、エンティティ値の内容をすべて表示する新しいウィンドウを開いて下さい)これは、このモデルで名前を割り当てられた唯一の制約式であるからです。

MSI 株式会社 Copy right©2015 All rights reserved

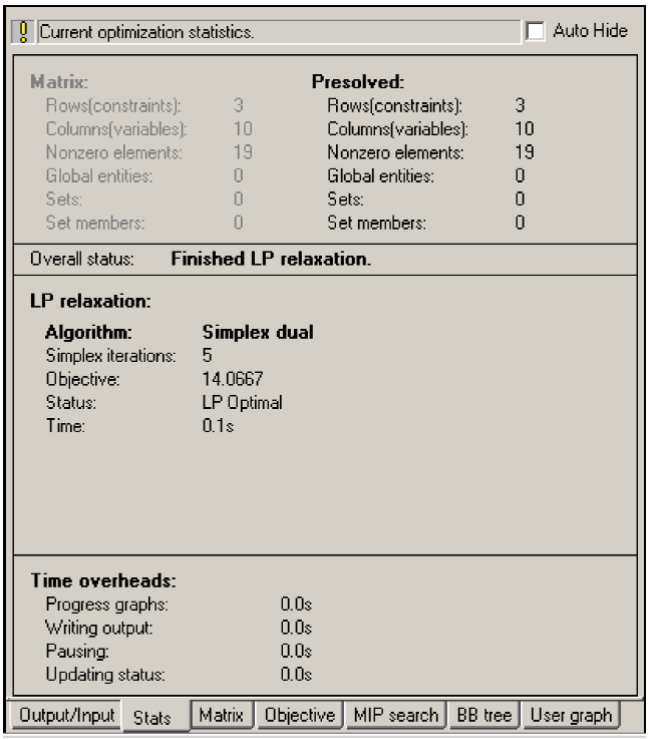


Figure 3.6: LP status display

3.4.1 文字列のインデックス付け

アウトプットを、もっと、読み易くするために、数字によるインデックスを、ストリング(文字列)によるインデックスで置き換えるのはよいアイディアです。この例題では、下記の 3 行の declaration line を

```
SHARES = 1..10
RISK = {2,3,4,9,10}
NA = {1,2,3,4}
```

下記の表現に置換します。

```
SHARES = {"treasury", "hardware", "theater", "telecom", "brewery",
          "highways", "cars", "bank", "software", "electronics"}
RISK = {"hardware", "theater", "telecom", "software", "electronics"}
NA = {"treasury", "hardware", "theater", "telecom"}
```

ここで、インデックスを使い RET 配列の初期化を行います。

```
 RET::([“treasury", "hardware", "theater", "telecom", "brewery",
        "highways", "cars", "bank", "software", "electronics"])[
5,17,26,12,8,9,7,6,31,21]
```

ここまで進んでくると、これ以上の変更はありません。

修正したモデルを foliolps.mos としてセーブします。ストリング・インデックスで置き換えたモデルを解き、プリントすると下記のようになり、この方が、結果が読みやすくなります。

MSI 株式会社 Copy right©2015 All rights reserved

```
Total return: 14.0667
treasury: 30%
hardware: 0%
theater: 20%
telecom: 0%

brewery: 6.66667%
highways: 30%
cars: 0%
bank: 0%
software: 13.3333%
electronics: 0%
```

もちろん、エンティティディスプレイにもまたこれらの文字列で表記されます。

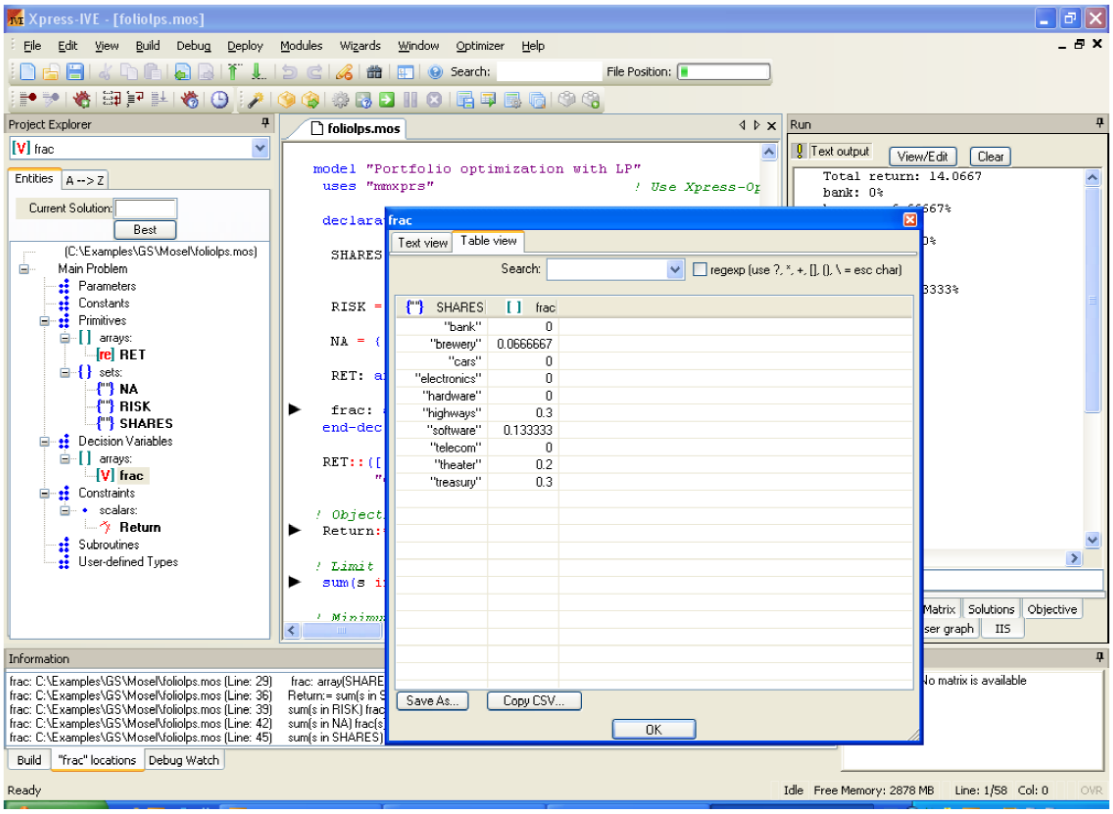


Figure 3.7: Entity display

MSI 株式会社 Copy right©2015 All rights reserved

第 4 章 データ・ハンドリング

この章では、Mosel のデータ・ハンドリングについての基本機能について説明します。

- Mosel 固有のフォーマットによるデータの読み込み、書き込みのための Initialization ブロック
- フリーフォーマットによるファイルへのデータアウトプット
- ファイル名、数値的制約式のパラメータ化（parameterization）
- アウトプットのフォーマティング

4.1 ファイルからのデータ・インプット

Mosel では、外部ファイルから、いくつかの方法でデータの読み込み、また書き出しができます。理解しやすくするためここでは、解説をテキストフォーマットのファイルに限定します。例えば Mosel は、ODBC 接続を使って、データをスプレッドシートやデータベースと交換するための特別なモジュールも持っていますが、これはこの本の範囲を越えていますので、興味のある読者はこれらのモジュールについてのドキュメントを参照して下さい。

これから使うデータファイル folio.dat は、以下の内容を持っています。

```
! Data file for 'folio*.mos'

RET: [("treasury") 5 ("hardware") 17 ("theater") 26 ("telecom") 12
      ("brewery") 8 ("highways") 9 ("cars") 7 ("bank") 6
      ("software") 31 ("electronics") 21 ]

RISK: ["hardware" "theater" "telecom" "software" "electronics"]

NA: ["treasury" "hardware" "theater" "telecom"]
```

モデルファイルと同じように、データファイルの中でも、'！' を前につけて、単一ラインのコメント行が挿入できます。すべてのデータエントリは、モデルの中の対応するエンティティに与えられた名前でレイベル (label)されます。データアイテムは、blank、tabulation、linebreak、または、comma で区切ります。ここで、第 3 章の Mosel モデルを次のように修正します。

```
declarations
 SHARES: set of string              ! Set of shares
 RISK: set of string                ! Set of high-risk values among shares
 NA: set of string                  ! Set of shares issued in N.-America
 RET: array(SHARES) of real         ! Estimated return in investment
end-declarations

initializations from "folio.dat"
 RISK RET NA
end-initializations

declarations
 frac: array(SHARES) of mpvar       ! Fraction of capital used per share
end-declarations
```

前のモデル foliolp.mos とは異なり、すべてのインデックスセットとデータ配列は、dynamic object とし

MSI 株式会社 Copy right©2015 All rights reserved

て作られています。すなわち、これらのサイズは、モデルの作成時点では未知です。後で、ファイル folio.dat から読み込まれるデータによって設定されます。オプションで、ファイルからの初期設定の後に、それらを静的にセットすることもできます。これにより、後で宣言され、これらのセットでインデックスを付けられる配列の扱いが効率的になります。さらに重要なことは、仮にセットが動的のために、検出できない'out of range'エラーを静的にセットする事により、Mosel がチェックすることができます。

```
finalize(SHARES); finalize(RISK); finalize(MA)
```

ここでは、セット SHARES を明示的に設定していないことに注意して下さい。これらは、配列 RET が読まれたとき、自動的に埋められます。さらに、ここでは、データを初期設定した後に、したがって、インデックスセットが知られた後に、変数を declare していることに注意して下さい。

4.2 Xpress-IVE を起動し、新しいモデルを生成する

前のセクションの initializations from と同様、Mosel には、initializations to もあります。
これは、標準フォーマットでデータを書き出すためのものです。しかし、ファイルに、IVE のアウトプット・ウィンドウに表示さているテキストを、そっくりそのまま、あるファイルに書き出したい場合は、次のようにして、手続き fopen と手続き fclose を call し、このテキストのプリントを囲むだけです。

```
fopen("result.dat", F_OUTPUT)
writeln("Total return: ", getobjval)
forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")
fclose(F_OUTPUT)
```

fopen の最初のアーギュメントは、アウトプットファイルの名前であり、2 番目は、それをどのモードで開くかを示します。上の設定では、モデルを再実行すると、アウトプットのファイルの内容は置換されます。既存のファイル内容に新しいアウトプットを付加するには、下記のようにします。

```
fopen("result.dat", F_OUTPUT+F_APPEND)
```

また、アウトプットの体裁をよくしたい場合には、例えば、下のようにします。

```
forall(s in SHARES)
 writeln(strfmt(s,-12), ": \t", strfmt(getsol(frac(s))*100,5,2), "%")
```

関数 strfmt は、ストリング、または、数をプリントするのに、最小のスペースを確保することを示します。2 番目のアーギュメントの負の値は、印刷が左詰で行われることを意味します。3 番目のアーギュメントはオプショナルなアーギュメントで、これで小数点の後の桁数を指定できます。このフォーマットされた方法で、アウトプットファイルは以下のようになります。

MSI 株式会社 Copy right©2015 All rights reserved

```
Total return: 14.0667
treasury    :  30.00%
hardware    :   0.00%
theater     :  20.00%
telecom     :   0.00%
brewery     :   6.67%
highways    :  30.00%
cars        :   0.00%
bank        :   0.00%
software    :  13.33%

electronics :   0.00%
```

4.3 パラメータ

一般に、良いモデル作成のスタイルと考えられていることは、できるだけ生の数字を直接、定義して、実行の際にそれらを外部のソースから読み込むようにします。そうすると、モデルは、用途が広くなり、いろいろな目的に使えるようになって、融通が利くモデルになり、容易に何度も再使用できるようになります。したがって、Mosel では、例えば、ファイル名や数字の定数をパラメータという形で定義することが可能です。これらのパラメータの値は、モデル自身に手を加えることなく、実行のときに修正できます。ここの例で言えば、インプット/アウトプットファイルをパラメータとして定義したり、また、制約式やバウンズの定数('RHS: right hand side') についても同様です。これらのパラメータ定義は、モデルファイルの最初(uses ステートメントのすぐ後)のに置く必要があります。

```
parameters
 DATAFILE= "folio.dat"              ! File with problem data
 OUTFILE= "result.dat"              ! Output file
 MAXRISK = 1/3                      ! Max. investment into high-risk values
 MAXVAL = 0.3                       ! Max. investment per share
 MINAM = 0.5                        ! Min. investment into N.-American values
end-parameters
```

勿論、この方式を使うときは、モデルの他の部分のファイル・ネームやデータの値もパラメータに置き換えなければなりません。IVE によってモデルを実行する時に、これらのパラメータの設定を修正するには、Build >>Run を選択するか、run button をクリックして下さい。開かれるダイアログボックスで、Usemodelparameters というフィールドにチェックを入れ、次の行で、パラメータの新しい値を入力します。例えば、Figure4.1 のように、結果の出力ファイルの名前を変更し、MAXRISK と MAXVAL の値を修正します。

MSI 株式会社 Copy right©2015 All rights reserved

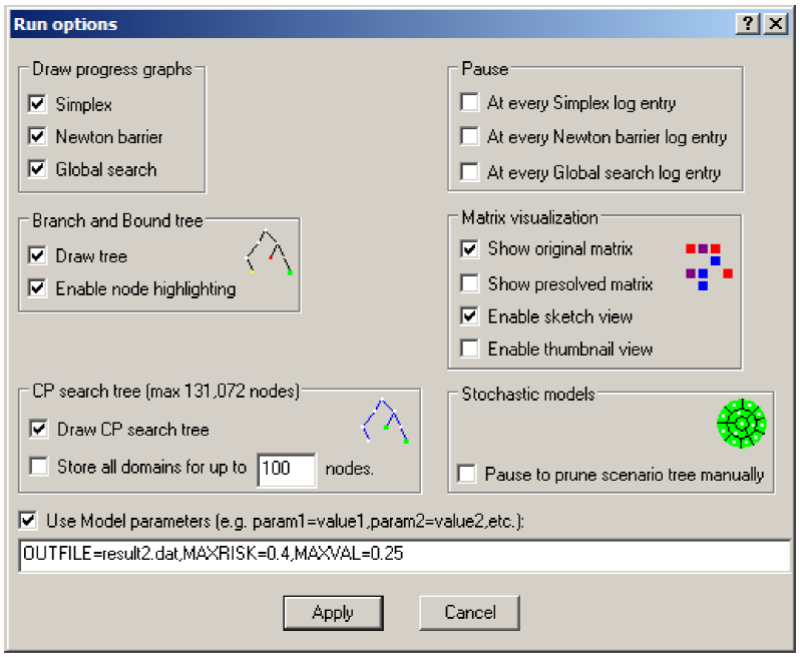


Figure 4.1: Changing model parameter settings

IVE の開発環境でモデルを単純にランするのではなく、テストや実験（バッチモード、コマンドラインインタフェースを使っているスクリプト）、最終的な開発（8 章を見てください）の場合には、パラメータの利用がとても重要であることを、是非、記憶に止めておいて下さい。例えば、種々のパラメータ設定でバッチファイルを書き、モデル foliodata.mos を繰り返し実行し、毎回、アウトプットを違うファイルに書きたいとします。そんな場合、単に以下の4行をバッチファイルに追加すればよいのです（そして、コマンド mosel で呼び出されるスタンドアロンバージョンの Mosel でモデルを実行します。）

```
mosel exec foliodata MAXRISK=0.1 OUTFILE='result1.dat'
mosel exec foliodata MAXRISK=0.2 OUTFILE='result2.dat'
mosel exec foliodata MAXRISK=0.3 OUTFILE='result3.dat'
mosel exec foliodata MAXRISK=0.4 OUTFILE='result4.dat'
```

パラメータ使用のもう一つの利点は、モデルが BIM ファイル（portable, compiled Binary Model files として配布される場合、それらのモデルをパラメータでランするようにしておけば、モデル自身を開示する必要がなく、したがって、知的所有権を保護できます。

4.4 修正した後のモデル

この章で解説したすべての機能をモデルファイル foliodata.mos は、次のようになります。

MSI 株式会社 Copy right©2015 All rights reserved

```
model "Portfolio optimization with LP"
 uses "mmxprs"                        ! Use Xpress-Optimizer

 parameters
  DATAFILE= "folio.dat"               ! File with problem data
  OUTFILE= "result.dat"               ! Output file
  MAXRISK = 1/3                       ! Max. investment into high-risk values
  MAXVAL = 0.3                        ! Max. investment per share
  MINAM = 0.5                         ! Min. investment into N.-American values
 end-parameters

 declarations
  SHARES: set of string               ! Set of shares
  RISK: set of string                 ! Set of high-risk values among shares
  NA: set of string                   ! Set of shares issued in N.-America
  RET: array(SHARES) of real          ! Estimated return in investment
 end-declarations

 initializations from DATAFILE
  RISK RET NA
 end-initializations

 declarations
  frac: array(SHARES) of mpvar        ! Fraction of capital used per share
 end-declarations

! Objective: total return
 Return:= sum(s in SHARES) RET(s)*frac(s)


   ! Limit the percentage of high-risk values
    sum(s in RISK) frac(s) <= MAXRISK

   ! Minimum amount of North-American values
    sum(s in NA) frac(s) >= MINAM



! Spend all the capital
 sum(s in SHARES) frac(s) = 1

! Upper bounds on the investment per share
 forall(s in SHARES) frac(s) <= MAXVAL

! Solve the problem
 maximize(Return)

! Solution printing to a file
 fopen(OUTFILE, F_OUTPUT)


 writeln("Total return: ", getobjval)
 forall(s in SHARES)
  writeln(strfmt(s,-12), ": \t", strfmt(getsol(frac(s))*100,2,3), "%")
 fclose(F_OUTPUT)

end-model
```


MSI 株式会社 Copy right©2015 All rights reserved

第 5 章 ユーザ定義のグラフを描く

この章では IVE を使って、どのようにしてユーザ定義のグラフを描くかについて説明します。表示したいグラフは、異なるパラメータ設定によって、モデルを繰り返し実行した結果をグラフにするものです。したがって、最初に、下記の事項について、Mosel 言語で簡単なアルゴリズムを書く例題を使って説明していきます。

- 制約式を定義し直す
- 最適化を繰り返す
- ソリューション情報を保存する
- ユーザーグラフを定義する：点、線、レイベル
- 簡単なプログラム作成（ループと選択）

5.1 問題の拡張

最適ポートフォリオ選択で考慮してきたデータ（第 2 章の表を参照）に加え、この投資家は、銘柄別の予想リターンの偏差の推定値を手に入れました。これらの逸脱の評価も手近に持っています（Table 5.1）。この投資家は、この追加的な情報も使って、ハイリスク銘柄への投資の最大値を変化させながら、LP モデルを実行させ、その結果をグラフとして表示したいと思っています。そこでは、リスクの尺度として偏差を使って、結果として得られる全体のリターンをプロットします。

Table 5.1: Estimated deviations

Number	Description	Deviation
1	treasury	0.1
2	hardware	19
3	theater	28
4	telecom	22
5	brewery	4
6	highways	3.5
7	cars	5
8	bank	0.5
9	software	25
10	electronics	16

5.2 最適化のループ

ここで、前から使ってきたモデル foliodata.mos を、銘柄別ハイリスク値（パーセンテージ）をベースに、修正して、問題を繰り返し最適化しようとしています。

具体的には、モデルは、以下のアルゴリズムを実施するように変換されます。

1. パラメータによって不変を保つモデルの部分の定義は変わります。
2. すべてのパラメータ値に対して、
   - ハイリスク値のパーセンテージを制限している制約式を再定義する

MSI 株式会社 Copy right©2015 All rights reserved

- その結果として得られる問題を解く
- もし、実現可能解が得られたら、解をセーブする

3. 結果をグラフに表示する

個々の最適化の後に、結果として得られる解の値、および、全体の推定偏差を蓄えるために、以下の 2 つの配列を宣言します。

```
declarations
 SOLRET: array(range) of real       ! Solution values (total return)
 SOLDEV: array(range) of real       ! Solution values (average deviation)
end-declarations
```

下のコーディングにより、ハイリスク銘柄の上限を制限している制約式の定義と問題を解く手順の部分のループが導入されます。このループを繰返すとき、前の定義を無効にすることができるように、この制約式に Risk とうい名前を付けます。もし制約式に名前がないときは、ループを実行すると、毎回、新しい制約式が追加されますが、既存の制約式は置換されません。

```
ct:=0
forall(r in 0..20) do
 ! Limit the percentage of high-risk values
  Risk:= sum(s in RISK) frac(s) <= r/20

  maximize(Return)                  ! Solve the problem
  if (getprobstat = XPRS_OPT) then  ! Save the optimal solution value
   ct+=1
   SOLRET(ct):= getobjval
   SOLDEV(ct):= getsol(sum(s in SHARES) DEV(s)*frac(s))
  else
   writeln("No solution for high-risk values <= ", 100*r/20, "%")
  end-if
end-do
```

上では、forall ループの 2 番目のフォーム、すなわち forall／do を使いました。いくつかのステートメントがループに含められているときには、このフォームは使われなければなりません。このループは、end-do で終わります。上のコードでの、もう一つの新しいフィーチャーは、if/then/else/end-if ステートメントです。ここでは、最適解が見つかった場合にのみ、その問題の値をセーブしたいと考えています。ここで、解のステータスは関数 getprobstat によって得られ、それが、「最適解であるかどうか」は、定数 XPRS_OPT でテストされます。選択ステートメントは、if/then/end-if、 および、if/then/elif/then/else/end-if というもう 2 つの別の形式も持っています。ここで、elif/then は、数回、繰り返えすこともできます。もっと多くの例、および、Mosel で利用可能なループや選択ステートメントに関する網羅的な説明については、「Mosel ユーザーガイド」を参照して下さい。

5.3 ユーザの求めるグラフを描く

これで、グラフを描くために必要なすべてのデータを集めました。グラフ機能はモジュール mmive により提供されます（詳しくは、「Mosel 言語参照マニュアル」を参照ください）。したがって、これは、モデルの最初でロードする必要があります。それには、下記の行を追加します。

MSI 株式会社 Copy right©2015 All rights reserved

```
uses "mmive"
```

続いて下記の行により、グラフを描きます。

```
declarations
 plot1: integer
end-declarations

plot1 := IVEaddplot("Solution values", IVE_BLACK)
forall(r in 1..ct) IVEdrawpoint(plot1, SOLRET(r), SOLDEV(r));
forall(r in 2..ct)
 IVEdrawline(plot1, SOLRET(r-1), SOLDEV(r-1), SOLRET(r), SOLDEV(r))
```

ユーザのグラフは、IVE の workspace の右のウィンドウに表示されます。タブ Usergraph を選んで、それをフォアグラウンドに移動させて下さい。以上の操作で、以下のアウトプットを得ます（いろいろな制約式の相互作用のため、予想に反して、結果として生じているグラフは直線ではありません）。

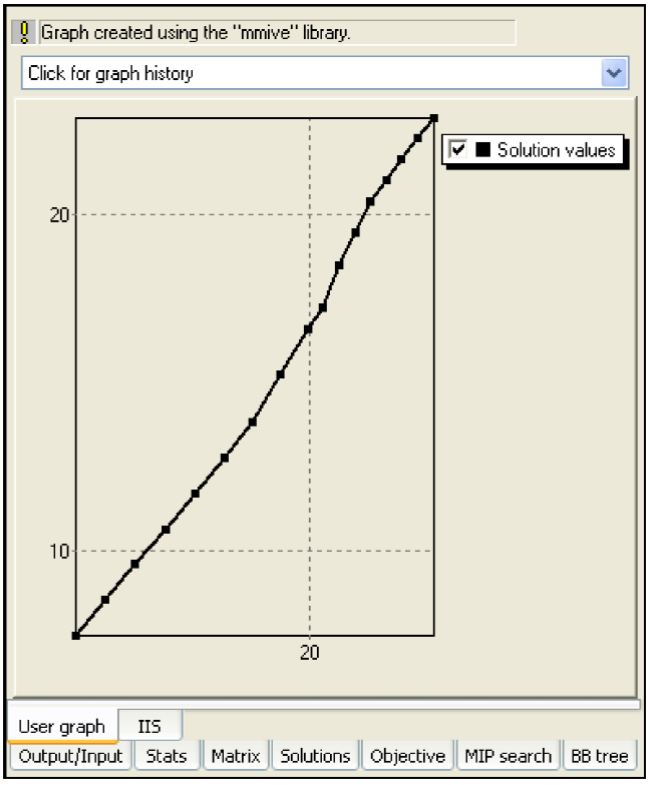


Figure 5.1: Plot of the result graph

このグラフだけでなく、入力データを示すレイベルを付けたポイントも表示できます（plot2 は低リスク銘柄、高リスク銘柄は plot3）。

MSI 株式会社 Copy right©2015 All rights reserved

```
declarations
 plot2, plot3: integer
end-declarations

plot2 := IVEaddplot("Low risk", IVE_YELLOW)
forall (s in SHARES - RISK) do
 IVEdrawpoint(plot2, RET(s), DEV(s))
 IVEdrawlabel(plot2, RET(s)+3.4, 1.3*(DEV(s)-1), s)
end-do


plot3 := IVEaddplot("High risk", IVE_RED)
forall (s in RISK) do
 IVEdrawpoint(plot3, RET(s), DEV(s))
 IVEdrawlabel(plot3, RET(s)-2.5, DEV(s)-2, s)


end-do
```

セット SHARES - RISK は、「RISK に含まれていない、SHARES のすべてのエレメント」を意味することに注意して下さい。こうすると、アウトプットは下記のようになります。

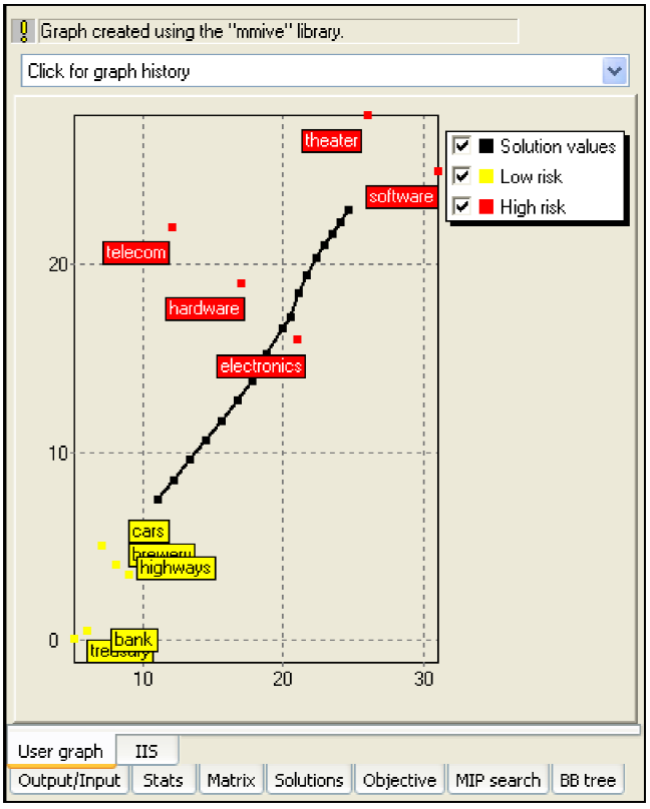


Figure 5.2: Plot of result graph and data

5.4 これらのフィーチャーを持った最終的なモデルファイル foliodata.mos

この章で議論したすべてのフィーチャーを持つ完全なモデルファイル foliodata.mos は、次のように

MSI 株式会社 Copy right©2015 All rights reserved

なります。ここで、2 つのモジュール mmxprs と mmive は、単一の use ステートメントにより、ロードできることに注意して下さい。偏差データは、オリジナルのデータ・ファイルに追加してもよいですし、ここに示されているように、他のファイルから読むこともできます。

```
model "Portfolio optimization with LP"
 uses "mmxprs", "mmive"             ! Use Xpress-Optimizer with IVE graphing

 parameters
  DATAFILE= "folio.dat"             ! File with problem data
  DEVDATA= "foliodev.dat"           ! File with deviation data
  MAXVAL = 0.3                      ! Max. investment per share
  MINAM = 0.5                       ! Min. investment into N.-American values
 end-parameters

 declarations
  SHARES: set of string             ! Set of shares
  RISK: set of string               ! Set of high-risk values among shares
  NA: set of string                 ! Set of shares issued in N.-America
  RET: array(SHARES) of real        ! Estimated return in investment
  DEV: array(SHARES) of real        ! Standard deviation
  SOLRET: array(range) of real      ! Solution values (total return)
  SOLDEV: array(range) of real      ! Solution values (average deviation)
 end-declarations


 initializations from DATAFILE
  RISK RET NA
 end-initializations

 initializations from DEVDATA
  DEV
 end-initializations

 declarations
  frac: array(SHARES) of mpvar      ! Fraction of capital used per share
  Return, Risk: linctr              ! Constraint declaration (optional)
 end-declarations

! Objective: total return
 Return:= sum(s in SHARES) RET(s)*frac(s)

! Minimum amount of North-American values
 sum(s in NA) frac(s) >= MINAM

! Spend all the capital
 sum(s in SHARES) frac(s) = 1

! Upper bounds on the investment per share
 forall(s in SHARES) frac(s) <= MAXVAL

! Solve the problem for different limits on high-risk shares
 ct:=0
 forall(r in 0..20) do
  ! Limit the percentage of high-risk values
   Risk:= sum(s in RISK) frac(s) <= r/20
```


MSI 株式会社 Copy right©2015 All rights reserved

```
maximize(Return)                    ! Solve the problem

if (getprobstat = XPRS_OPT) then  ! Save the optimal solution value
 ct+=1
 SOLRET(ct):= getobjval
 SOLDEV(ct):= getsol(sum(s in SHARES) DEV(s)*frac(s))
else
     writeln("No solution for high-risk values <= ", 100*r/20, "%")
    end-if
  end-do

 ! Drawing a graph to represent results ('plot1') and data ('plot2' & 'plot3')
  declarations
   plot1, plot2, plot3: integer
  end-declarations

  plot1 := IVEaddplot("Solution values", IVE_BLACK)
  plot2 := IVEaddplot("Low risk", IVE_YELLOW)
  plot3 := IVEaddplot("High risk", IVE_RED)

  forall(r in 1..ct) IVEdrawpoint(plot1, SOLRET(r), SOLDEV(r));

  forall(r in 2..ct)
   IVEdrawline(plot1, SOLRET(r-1), SOLDEV(r-1), SOLRET(r), SOLDEV(r))

  forall (s in SHARES-RISK) do
   IVEdrawpoint(plot2, RET(s), DEV(s))
   IVEdrawlabel(plot2, RET(s)+3.4, 1.3*(DEV(s)-1), s)
  end-do

  forall (s in RISK) do
   IVEdrawpoint(plot3, RET(s), DEV(s))
   IVEdrawlabel(plot3, RET(s)-2.5, DEV(s)-2, s)
  end-do

 end-model
```

この問題は、制約式 Risk の小さい限界値ではインフィージブルです。したがって、このグラフに加え、以下のテキストアウトプットが表示されます。

```
No solution for high-risk values <= 0%
No solution for high-risk values <= 5%
No solution for high-risk values <= 10%
No solution for high-risk values <= 15%
```


MSI 株式会社 Copy right©2015 All rights reserved

第 6 章 混合整数計画(MIP)

この章では、第 3 章で作成したモデルを混合整数計画(MIP) に拡張します。ここでは、下記について説明します。

- 離散的な変数を定義する
- IVE の MIP オプティマイザーのディスプレイを理解し、使いこなす

6.1 問題の拡張

この投資家は、少量の株式保有をしたくないと考えています。したがって、銘柄別の保有について、次のようなことができないかと考えています。

1. ポートフォリオに入れる銘柄数を制限する
2. ある銘柄がポートフォリオに入るなら、その銘柄の保有金額が総投資額に占める割合をあるレベル以上に設定(MINVAL) したい(投資資本の、少なくとも 10%)。

以下では、これらの二つの制約式を、二つの別個のモデルを使って、どのように扱うかを示します。

6.2 MIP モデル 1: 銘柄数を制限する

投資の対象となるいろいろな銘柄から、ポートフォリオに入れる銘柄数を制限するためには、銘柄数をカウントできなければなりません。そのために、第 2 章で作成したモデルに、二つ目の、buy という変数のセットを導入します。これらの変数はインディケータ変数(indicator variable) 、すなわち、バイナリ変数(binary variable)です。変数 buy_s は、ある銘柄がポートフォリオに入ると1という値を取り、入らない場合は 0 という値をとります。ここで、次の制約式を導入し、銘柄数の最大数を MAXNUM に制限します。この制約式で、セット変数 buy_s は、解の中で、最大でも、MAXNUM 個の変数 buy_s のみが値 1 を同時に取ることができるということを表現しています。

$$\sum_{s \in SHARES} buy_s \leq MAXNUM$$

ここで、新しいバイナリ変数 buy_s を、ポートフォリオに選ばれたすべての銘柄の数量を表す変数 fracs に結びつける必要があります。表現したい関係は、「もし、ある銘柄がポートフォリオに選ばれるならば、それは、銘柄数の一つとして、全体の中でカウントされる」、換言すると、'if fracs >0, then buy_s =1' ということです。以下の不等式により、このことを定式化します。

$$\forall s \in SHARES : frac_s \leq buy_s$$

もし、ある s で、$frac_s$ がノンゼロであるならば、buy_s は 0 より大きくなければならず、したがって、buy_s は1という値になります。逆に言えば、もし、buy_s が0であるなら、そうすると、$frac_s$ も 0 であり、銘柄 はポートフォリオに全たく取り入れられないことを意味します。ここで、これらの制約式が、buy_s が1という値を取り、かつ、$frac_s$ が0という値を取る可能性を排除しないことに注意して下さい。

MSI 株式会社 Copy right©2015 All rights reserved

しかし、このケースでは、これは問題ありません。なぜならば、解で、buy_s が 1 という値を取り、かつ、$frac_s$ が 0 という値をとっても、これらの両方の変数が 0 であるのと同じ効力を持つからです。

6.2.1 Mosel によるインプリメンテーション

第 3 章で開発された LP モデルを、(第 4 章で導入したファイルから、データの初期設定を行い)、新しい変数と制約式で拡張します。新しい変数がバイナリであるという事実(すなわち、それらが 0 という値と 1 という値のみを取る変数であること)は、Mosel の中で、'is_ binary' という表現を持った制約式で定義します。もう一つの、よく見られる普通のタイプの離散的な変数は、整数変数(integer variable)です。これらの変数は、与えられた下限と上限の間の整数値のみを取れます。このタイプの変数は、Mosel の中で、'is_ integer' という表現を持った制約式で定義します。以下のセクション(MIP model2) では、もう一つの離散的な変数、すなわち、半連続変数(semi-continuous variable)の例を見ます。

```
model "Portfolio optimization with MIP"
 uses "mmxprs"                      ! Use Xpress-Optimizer

 parameters
  MAXRISK = 1/3                     ! Max. investment into high-risk values
  MAXVAL = 0.3                      ! Max. investment per share
  MINAM = 0.5                       ! Min. investment into N.-American values
  MAXNUM = 4                        ! Max. number of different assets
 end-parameters

 declarations
  SHARES: set of string             ! Set of shares
  RISK: set of string               ! Set of high-risk values among shares
  NA: set of string                 ! Set of shares issued in N.-America
  RET: array(SHARES) of real        ! Estimated return in investment
 end-declarations

 initializations from "folio.dat"
  RISK RET NA
 end-initializations

 declarations
  frac: array(SHARES) of mpvar      ! Fraction of capital used per share
  buy: array(SHARES) of mpvar       ! 1 if asset is in portfolio, 0 otherwise
 end-declarations

! Objective: total return
 Return:= sum(s in SHARES) RET(s)*frac(s)

! Limit the percentage of high-risk values
 sum(s in RISK) frac(s) <= MAXRISK

! Minimum amount of North-American values
 sum(s in NA) frac(s) >= MINAM

! Spend all the capital
 sum(s in SHARES) frac(s) = 1
```


MSI 株式会社 Copy right©2015 All rights reserved

```
! Upper bounds on the investment per share
 forall(s in SHARES) frac(s) <= MAXVAL

! Limit the total number of assets
 sum(s in SHARES) buy(s) <= MAXNUM


forall(s in SHARES) do
 buy(s) is_binary                  ! Turn variables into binaries
 frac(s) <= buy(s)                 ! Linking the variables


  end-do

 ! Solve the problem
  maximize(Return)

 ! Solution printing
  writeln("Total return: ", getobjval)
  forall(s in SHARES)
   writeln(s, ": ", getsol(frac(s))*100, "% (", getsol(buy(s)), ")")

 end-model
```

上のモデル、`foliomip1.mos`でforall ループの2番目の形式、すなわち、forall/do を使いました。ループがいくつかのステートメントをまたがる場合には、この形式のループを使う必要があります。同等のことを以下のように書くこともできます。

```
forall(s in SHARES) buy(s) is_binary
forall(s in SHARES) frac(s) <= buy(s)
```

6.2.2 解を分析する

問題の実行を、ひとまず、ポーズし（Build >> Options を使うか、または、run button をクリックする）、マトリックスを視覚化すると、以下のディスプレイを得ます。

MSI 株式会社 Copy right©2015 All rights reserved

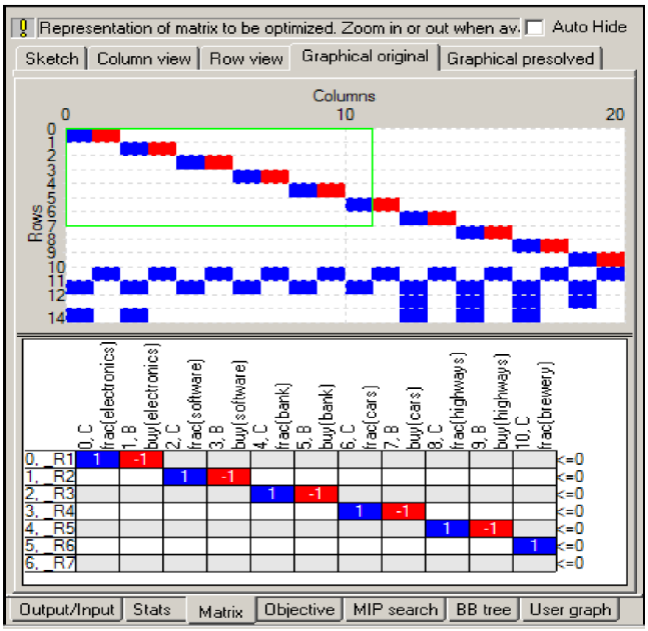


Figure 6.1: Matrix of the MIP problem

これまでの章の LP マトリックスより、現在のマトリックスには、もっと多くの行（制約式）と列（変数）があります。ここで、$frac_s \leq buy_s$という制約式は、$frac_s - buy_s \leq 0$という形式に変換されていることに注意して下さい。これは、マトリックスでは、すべての変数は、=、 、 の左側（左辺）にあるように表示するからです（Mosel でも、制約式をこの標準型で保存します）。

モデルを実行すると、その結果として、以下のアウトプットを得ます（ソリューションの information ウィンドウのタブ’ Output/input’を選ぶ）。

```
Total return: 13.1
treasury: 20% (1)

hardware: 0% (0)
theater: 30% (1)
telecom: 0% (0)
brewery: 20% (1)
highways: 30% (1)
cars: 0% (0)
bank: 0% (0)
software: 0% (0)
electronics: 0% (0)
```

この解で得られる最大のリターンは、もともとの LP 問題のそれより小さくなっていますが、これは、制約式を追加したためです。要求されたように、このポートフォリオは 4 つの銘柄から構成されています。ここで、ソリューション情報を詳細に見ましょう（タブ Stats） 。

MSI 株式会社 Copy right©2015 All rights reserved

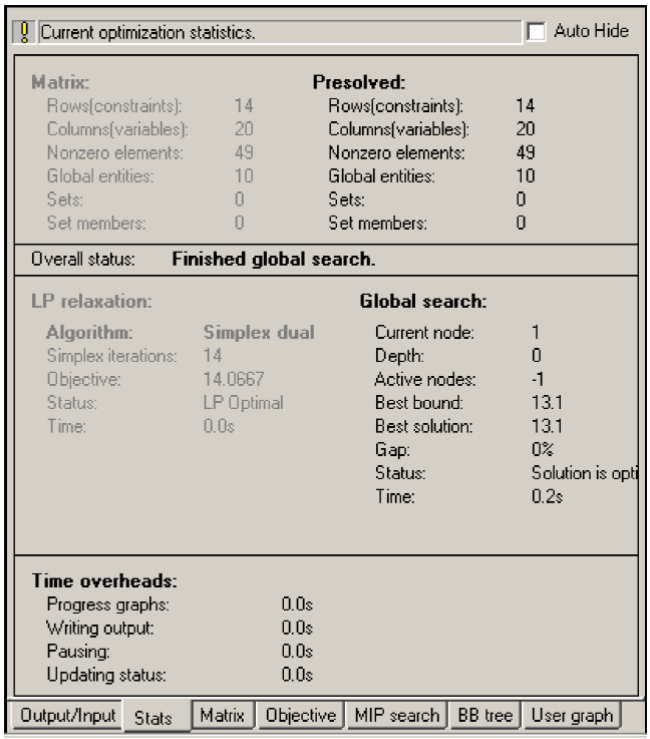


Figure 6.2: Detailed MIP solution information

ここには、Global search という、MIP-固有の情報が、ディスプレイの中央部に含まれています。ここで経験したように、いくつか(またはすべて)の変数に整数条件を追加して、LP モデルを MIP モデルに変えることは、比較的に容易です。しかし、MIP を解くためのアルゴリズムについては、同じことは言えません。なぜならば、MIP 問題は、繰り返し、何度も、何度も、LP 問題を解くことで解かれるからです。このとき、最初は、整数条件を無視して、問題は LP として解かれます(LP 緩和)。そして、現在の解では整数条件を満たしていないある離散的な変数を選び、この変数の新しい上限や下限を追加して、この変数の取る値が整数値になるようにして行きます。ここで、各 LP 解をノード(node) で示し、これらのノードをバウンドの変更や追加された制約式で結んで行くと、「木」のような形をした構造が得られます。これが分枝限定法のツリー(Branch- and-Bound tree) です。分枝についての情報は、問題を解くのに、いくつの分枝限定のノードが必要であったかを示します。この問題では、解くのに必要であったノードは 1 個ですみ、次々とノードを探って行く必要はありませんでした。Xpress-Optimizer では、デフォルトで、MIP の前処理のために、あるアルゴリズムを実行します(アルゴリズムの設定の詳細については、「Optimizer Reference Manual」を参照)。これには、MIP の解が存在しない LP の実行可能領域の一部を自動的にカットするための制約式の追加をする「カットの自動生成」が含まれています。この問題のサイズは非常に小さいので、前処理により、容易な問題になり、したがって、この問題は、即座に、解かれました。maximize を呼び、再実行を行う前に、モデルに下記の行を付け加えると、MIP の前処理の一部、すなわち、自動カット生成機能がスイッチ・オフされます。

MSI 株式会社 Copy right©2015 All rights reserved

```
setparam("XPRS_CUTSTRATEGY",0)
setparam("XPRS_HEURSTRATEGY",0)
setparam("XPRS_PRESOLVE",0)
```

この状態で、同じ問題を解くと、今度は、解を得るまでに、ノードを 25 個も調べているのが判ります

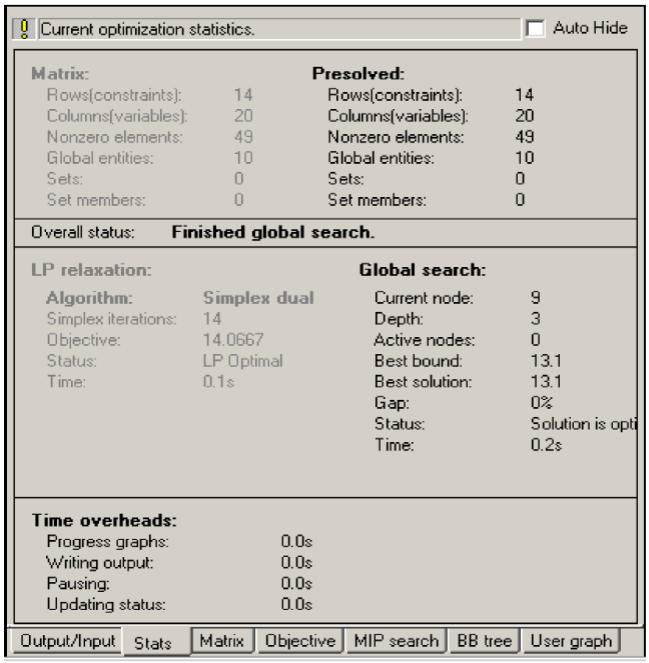


Figure 6.3: Detailed MIP solution information (after disabling cuts)

したがって、今度は、タブ BB tree を選択すると、分枝限定法のツリーを表示できます

MSI 株式会社 Copy right©2015 All rights reserved

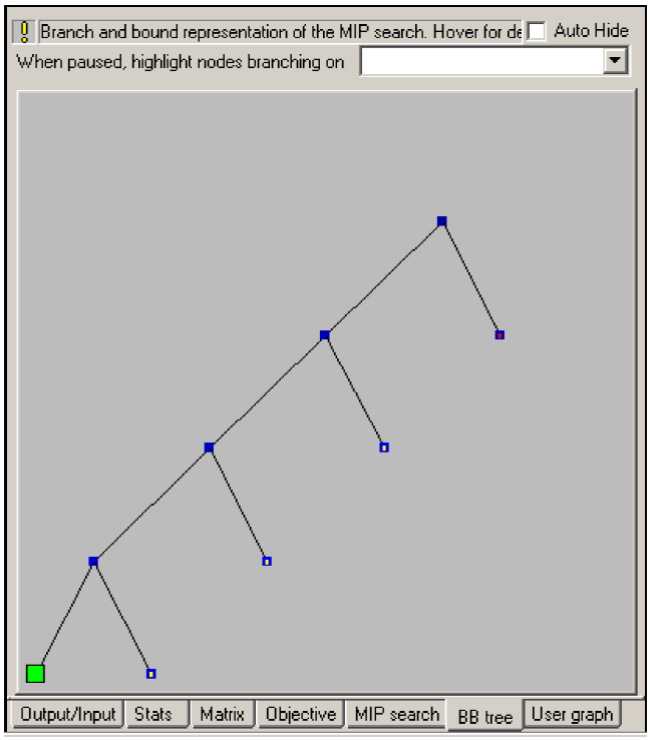


Figure 6.4: Branch-and-Bound tree

ツリーのノードは、異なるカラーコードで表示されます。ここで、整数解が見つかったノードは、緑色の四角形で表示され、最適解は大きな緑色の四角形で表示されます。赤い小さなドットは、LP 緩和の実行不能解を示しています。ツリーのノードの上でカーソルを動かすことによって、ノード番号、分枝のために選ばれた変数などの追加的な情報が表示されます。マトリックスのディスプレイと同様に、Branch-and Bound アルゴリズムを休止することができます。こうして、探索ツリーの構造を詳細に調べることができます。

そして、(Build >> Options を選択するか、または、run options button をクリック global search entry をチェックし、Apply で確認して下さい。こうすると、モデルを実行すると、ノードごとに実行を休止するので、ユーザは、探索ツリーの構築状況をフォローできます。実行を継続するには、pause button をクリックして下さい。実行を中止する場合は、stop button をクリックします。

6.3 MIP モデル 2: 各銘柄の最低投資額の条件を入れる

2 番目の MIP モデルを定式化するために、ここで、再び、第 2 章、および、第 3 章の LP モデルから始めます。定式化したい新しい制約式は、「もし、ある銘柄がポートフォリオの中に組み込まれるならば、少なくとも、その銘柄の保有金額が総投資額に占める割合をあるレベル以上に設定し(MINVAL) たい(ここでは、投資資本の少なくとも 10%)」ということです。今度は、セット変数 fracs のすべてを、単純に、0 と MAXVAL の間の値を取るように制約する代わりに、それらを MINVAL と MAXVAL の間の数値を取るか、または、0 という値を取るように制約します。このタイプの変数は半連続変数(semi-continuous variable)と呼ばれます。新しいモデルでは、fracs のバウンズを、以

MSI 株式会社 Copy right©2015 All rights reserved

下の制約式によって置き換えます。

$$\forall s \in SHARES : frac_s = 0 \text{ or } MINVAL \leq frac_s \leq MAXVAL$$

6.3.1 MOSEL で実行する

下のモデル`foliomip2.mos`は、第 3 章で開発された LP モデルを、(第 4 章で導入したファイルから、データの初期設定行い)、新しい変数と制約式で拡張したものもで、MIP モデル 2 を実行します。モデル内で、セミ連続変数は、'is_ semcont ' 制約式で定義されます。

MOSEL で扱えるもう一つの、類似したタイプの整数変数は、0 という値を取るか、与えられた下限と上限の間の整数値を取る変数です。この変数は、いわゆる、半連続整数変数(semi- continuous integer)と呼ばれています。モデル内で、セミ連続整数変数は、'is_ semint ' 制約式で定義されます。3 番目のタイプは、部分整数変数(partial integer)で、与えられた下限からある限界値までは整数値を取り、この値を越えると連続変数となるものです。これは、'is_ partint ' 制約式で定義されます。

```
 model "Portfolio optimization with MIP"
  uses "mmxprs"                      ! Use Xpress-Optimizer

  parameters
   MAXRISK = 1/3                     ! Max. investment into high-risk values
   MINAM = 0.5                       ! Min. investment into N.-American values
   MAXVAL = 0.3                      ! Max. investment per share
   MINVAL = 0.1                      ! Min. investment per share
  end-parameters

  declarations
   SHARES: set of string             ! Set of shares
   RISK: set of string               ! Set of high-risk values among shares
   NA: set of string                 ! Set of shares issued in N.-America
   RET: array(SHARES) of real        ! Estimated return in investment
  end-declarations

  initializations from "folio.dat"
   RISK RET NA
  end-initializations

 initializations from "folio.dat"
  RISK RET NA
 end-initializations

 declarations
  frac: array(SHARES) of mpvar       ! Fraction of capital used per share
 end-declarations

! Objective: total return
 Return:= sum(s in SHARES) RET(s)*frac(s)

! Limit the percentage of high-risk values
 sum(s in RISK) frac(s) <= MAXRISK
```


MSI 株式会社 Copy right©2015 All rights reserved

```
 ! Minimum amount of North-American values
  sum(s in NA) frac(s) >= MINAM

 ! Spend all the capital
  sum(s in SHARES) frac(s) = 1

 ! Upper and lower bounds on the investment per share
  forall(s in SHARES) do
   frac(s) <= MAXVAL
   frac(s) is_semcont MINVAL
  end-do

 ! Solve the problem
  maximize(Return)

 ! Solution printing
  writeln("Total return: ", getobjval)
  forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")

 end-model
```

このモデルを実行すると、以下のアウトプットが得られます（solution information ウィンドウの Output/input を選択して下さい）。

```
Total return: 14.0333
treasury: 30%
hardware: 0%
theater: 20%
telecom: 0%
brewery: 10%
highways: 26.6667%
cars: 0%


bank: 0%
software: 13.3333%
electronics: 0%
```

ポートフォリオには 5 つの銘柄が選ばれます。そして、各銘柄は、総投資金額の最低 10%、最大 30%の間にあります。追加された制約式のため、最適な MIP 解の値は、この場合も、最初の LP 解の値よりも小さくなります。

MSI 株式会社 Copy right©2015 All rights reserved

第 7 章　二次計画法

この章では、第 3 章で作成したモデルを二次計画法(Quadratic Programming)のモデルに、そして、第 6 章の最初の MIP モデルを混合整数二次計画法(Mixed Integer Quadratic Programming)のモデルに拡張します。この章は、下記について説明します。

・二次式の目的関数を定義する

・問題を少しずつ定義して、問題を解く

・ IVE の MIP optimization display を理解して、活用する

第 12 章では BCL でこれまで使ってきた例題の定式化および解く方法を解説し、第 17 章では、QP 問題を直接 Xpress-Optimizer でインプットし解く方法を解説します。

7.1 問題の説明

この投資家は、もっと、別の角度から、このポートフォリオ選択問題を見ることもできます。すなわち、予想リターンを最大化し、ハイリスクの投資部分を制限するのではなく、あるレベルのリターンを達成しながら、リスクを最小化することです。この投資家は、証券投資の予想リターンの分散/共分散マトリックスの推定値を得る、という Markowitz のアイデアを使おうと思っています。（例えば、ハードウェア銘柄とソフトウェア銘柄の会社の価値は連動する傾向があるが、人々は、新しいコンピュータやコンピュータゲームで遊ぶことに飽きるとさらに劇場に行くようになるので、演劇プロダクションの成功とは、逆の相関がある、と考えるのがこの考え方の一つの例です。）カナダの財務省証券の利回りは確実なのに対して、劇場プロダクション銘柄のリターンは大幅に変動します。予想リターン、および、分散／共分散マトリックスは、下表に示されています。

Table 7.1: Variance/covariance matrix

	treasury	hardw.	theater	telecom	brewery	highways	cars	bank	softw.	electr.
treasury	0.1	0	0	0	0	0	0	0	0	0
hardware	0	19	-2	4	1	1	1	0.5	10	5
theater	0	-2	28	1	2	1	1	0	-2	-1
telecom	0	4	1	22	0	1	2	0	3	4
brewery	0	1	2	0	4	-1.5	-2	-1	1	1
highways	0	1	1	1	-1.5	3.5	2	0.5	1	1.5
cars	0	1	1	2	-2	2	5	0.5	1	2.5
bank	0	0.5	0	0	-1	0.5	0.5	1	0.5	0.5
software	0	10	-2	3	1	1	1	0.5	25	8
electronics	0	5	-1	4	1	1.5	2.5	0.5	8	16

質問 1：ある特定された最小の目標利回りを得ることを前提にして、分散を最小化するには、投資家はどのような戦略を採用するべきか。

質問 2：ある特定された最小の目標利回りを得ることを前提にして、仮に、投資家が、最大でも 4 つの銘柄を選びたいとするならば、どれが、最小の分散を持つ投資戦略だろうか。最初の質問により、問題は、二次計画法問題、すなわち目的関数が二次項を持ち、制約式が一次制約式である数理計画問題になります。2 番目の質問により、銘柄の数を数えるために、離散的な変数の導入が必要と

MSI 株式会社 Copy right©2015 All rights reserved

なり、したがって、問題は、混合整数二次計画問題になります。これらの 2 つのケースは、以下の 2 つのセクションで別々に議論します。

7.2 QP

第 2 章で作成したモデルを、問題を新しい観点から見ると、下記の変更を行う必要があります。

・新しい目的関数は、リターンの合計ではなく、平均分散(meanvariance)となる

・ リスクに関係する制約式が消える

・目標利回りという新しい制約式を追加する

新しい目的関数は、下記で表されるポートフォリオの平均分散です。

$$\sum_{s,t \in SHARES} VAR_{st} \cdot frac_s \cdot frac_t$$

ここで、VARst は、すべての銘柄の分散/共分散マトリックスです。これは、二次の目的関数です。目的関数に、自乗された変数、例えば $frac_1^2$ や二つの変数が掛け合わされる、例えば、$frac_1 \cdot frac_2$ のような項があると、その目的関数は「二次の目的関数」であると言われます。

目標利回りの制約式は、下記のようになります。

$$\sum_{s \in SHARES} RET_s \cdot frac_s \geq TARGET$$

北米銘柄への投資の限度、すべての資金を投資すること、および各銘柄への投資の上限は、このモデルにも適用されます。したがってこのモデルの数学的モデルは以下のようになります。

$$\text{minimize} \sum_{s,t \in SHARES} VAR_{st} \cdot frac_s \cdot frac_t$$

$$\sum_{s \in NA} frac_s \geq MINAM$$

$$\sum_{s \in SHARES} frac_s = 1$$

$$\sum_{s \in SHARES} RET_s \cdot frac_s \geq TARGET$$

$$\forall s \in SHARES : 0 \leq frac_s \leq MAXVAL$$

7.2.1 Mosel で実行する

モデルの実行には、Xpress-Optimizer モジュール mmxprs だけでなく、QP モジュール mmquad をロードする必要があります。QP モジュール mmquad は、二次形式の定義に必要な機能を Mosel 言語に追加します（mmquad については、「Mosel 言語参照マニュアル（Mosel Language Reference Manual）」を参照）。QP モジュール mmquad をロードすると、二次形式の目的関数の最適化機能 maximize や minimize を使えるようになります。このモデルは、これまでのモデルとは異

MSI 株式会社 Copy right©2015 All rights reserved

なるデータファイル(folioqp.dat)を使います。

```
            ! trs  haw  thr  tel  brw  hgw  car  bnk  sof  elc
RET: [   (1)   5   17   26   12    8    9    7    6   31   21]

VAR: [ (1 1) 0.1    0    0    0    0    0    0    0    0    0 ! treasury
       (2 1)   0   19   -2    4    1    1    1  0.5   10    5 ! hardware
       (3 1)   0   -2   28    1    2    1    1    0   -2   -1 ! theater
       (4 1)   0    4    1   22    0    1    2    0    3    4 ! telecom
       (5 1)   0    1    2    0    4 -1.5   -2   -1    1    1 ! brewery
       (6 1)   0    1    1    1 -1.5  3.5    2  0.5    1  1.5 ! highways
       (7 1)   0    1    1    2   -2    2    5  0.5    1  2.5 ! cars
       (8 1)   0  0.5    0    0   -1  0.5  0.5    1  0.5  0.5 ! bank
       (9 1)   0   10   -2    3    1    1    1  0.5   25    8 ! software
      (10 1)   0    5   -1    4    1  1.5  2.5  0.5    8   16 ! electronics
     ]

RISK: [2 3 4 9 10]
NA: [1 2 3 4]
```

ここでは、ストリングインデックス(string index［文字列のインデックス］)ではなく、数字のインデックスを使っていることに注意してください。モデルでセット SHARES が定義されているので、ファイル内のすべてのデータエントリにインデックスタプルをリストする必要はありません。リストされているタプルは、判りやすくするためだけのものです。

```
model "Portfolio optimization with QP/MIQP"
 uses "mmxprs", "mmquad"            ! Use Xpress-Optimizer with QP solver

 parameters
  MAXVAL = 0.3                      ! Max. investment per share
  MINAM = 0.5                       ! Min. investment into N.-American values
  MAXNUM = 4                        ! Max. number of different assets
  TARGET = 9.0                      ! Minimum target yield
 end-parameters

 declarations
  SHARES = 1..10                    ! Set of shares
  RISK: set of integer              ! Set of high-risk values among shares
  NA: set of integer                ! Set of shares issued in N.-America
  RET: array(SHARES) of real        ! Estimated return in investment
  VAR: array(SHARES,SHARES) of real ! Variance/covariance matrix of
                                    ! estimated returns
 end-declarations


initializations from "folioqp.dat"
 RISK RET NA VAR
end-initializations

declarations
 frac: array(SHARES) of mpvar      ! Fraction of capital used per share
end-declarations
```


MSI 株式会社 Copy right©2015 All rights reserved

```
 ! Objective: mean variance
  Variance:= sum(s,t in SHARES) VAR(s,t)*frac(s)*frac(t)

 ! Minimum amount of North-American values
  sum(s in NA) frac(s) >= MINAM

 ! Spend all the capital
  sum(s in SHARES) frac(s) = 1

 ! Target yield
  sum(s in SHARES) RET(s)*frac(s) >=  TARGET


! Upper bounds on the investment per share
 forall(s in SHARES) frac(s) <= MAXVAL

! Solve the problem
 minimize(Variance)

! Solution printing
 writeln("With a target of ", TARGET, " minimum variance is ", getobjval)
 forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")

end-model
```

このモデルを解くと、以下のようなソリューションアウトプットが出てきます（solution information window の Output/input）。

```
With a target of 9 minimum variance is 0.557393
1: 30%
2: 7.15391%
3: 7.38246%
4: 5.46363%
5: 12.6554%
6: 5.91228%
7: 0.332458%
8: 30%
9: 1.09983%
10: 0%
```

第 5 章で示されたアルゴリズムと同じように、TARGET にいろいろな値を設定し、この問題を何度か解いて、その結果を目標リターン/標準偏差グラフにプロットして、「効率的なフロンティア」を知ることができます。(model file `folioqpgraph.mos`):

MSI 株式会社 Copy right©2015 All rights reserved

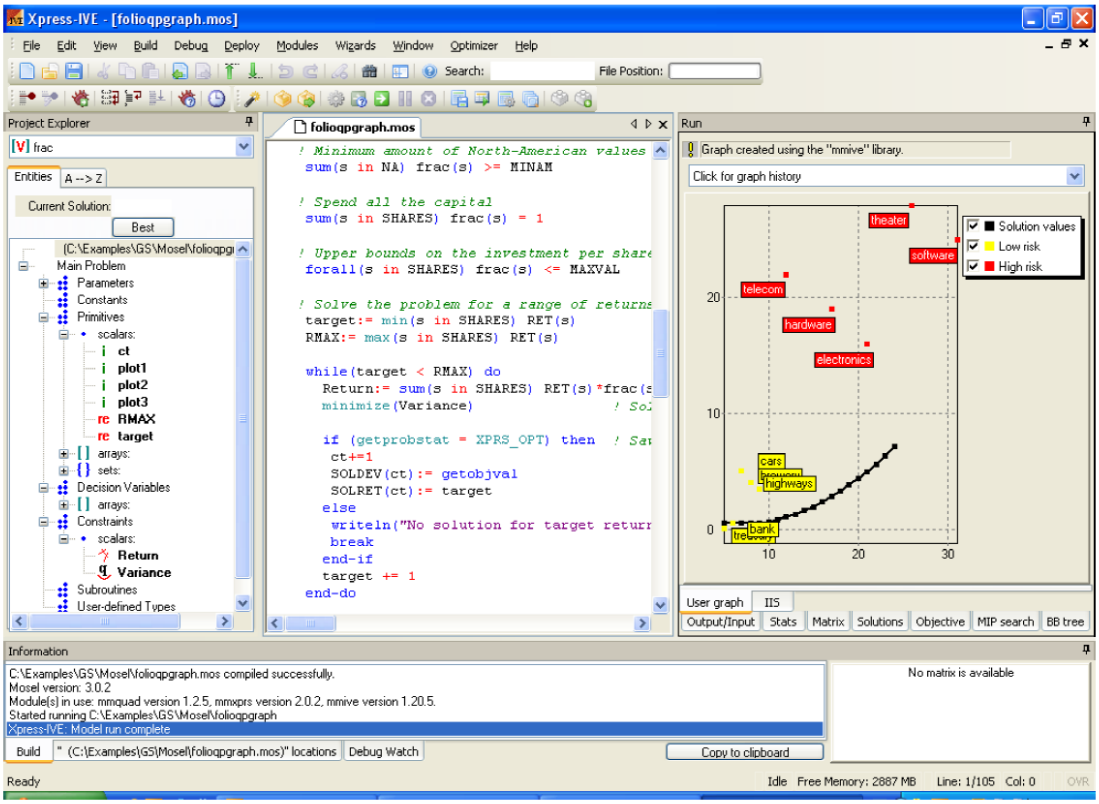


Figure 7.1: Graph of the efficient frontier

7.3 MIQP

ここで、前の QP モデルのすべての制約式の下で、最大でも MAXNUM という数の銘柄のみポートフォリオに入れる条件をモデルに入れましょう。すでに、第 6 章で、どのようにこれを行えばよいかを見ました。すなわち、バイナリ変数のセット buy_s を追加することにより、下記のように連続変数に論理的にリンクすることでした。

$$\forall s \in SHARES : frac_s \leq buy_s$$

この関係により、もし、$frac_s$ >0 ならば、すなわち、$frac_s$ がポートフォリオに選ばれるならば、変数 buy_s は 1 になり、また、buy_s が 0 であるならば、$frac_s$ は 0 でなければならないということになります。

$$\sum_{s \in SHARES} buy_s \leq MAXNUM$$

MSI 株式会社 Copy right©2015 All rights reserved

7.3.1 Mosel によるインプリメンテーション

前の QP モデルを修正する、すなわち、以下の行を、前節の QP モデルの終わりに追加します。その後この問題を、1 回の実行で、まず QP モデルとして解き、続けて、MIQP モデルとして解きます。

```
declarations
 buy: array(SHARES) of mpvar       ! 1 if asset is in portfolio, 0 otherwise
end-declarations

! Limit the total number of assets
sum(s in SHARES) buy(s) <= MAXNUM

forall(s in SHARES) do
 buy(s) is_binary
 frac(s) <= buy(s)
end-do

! Solve the problem
minimize(Variance)

writeln("With a target of ", TARGET," and at most ", MAXNUM,
        " assets, minimum variance is ", getobjval)
forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")
```

MIQP モデルを実行すると、下記のような解を得ます。

```
With a target of 9 and at most 4 assets,
 minimum variance is 1.24876
1: 30%
2: 20%
3: 0%
4: 0%
5: 23.8095%
6: 26.1905%
7: 0%
8: 0%
9: 0%
10: 0%
```

ポートフォリオの中の銘柄数についての制約式を追加したので、最小分散は、QP 問題の解のそれよりも、2 倍以上大きくなります。

MSI 株式会社 Copy right©2015 All rights reserved

7.3.2 解を分析する

それではここで、ソリューション・ディスプレイを見てみましょう。Stats ウィンドウを選択すると、以下の情報が見られます。

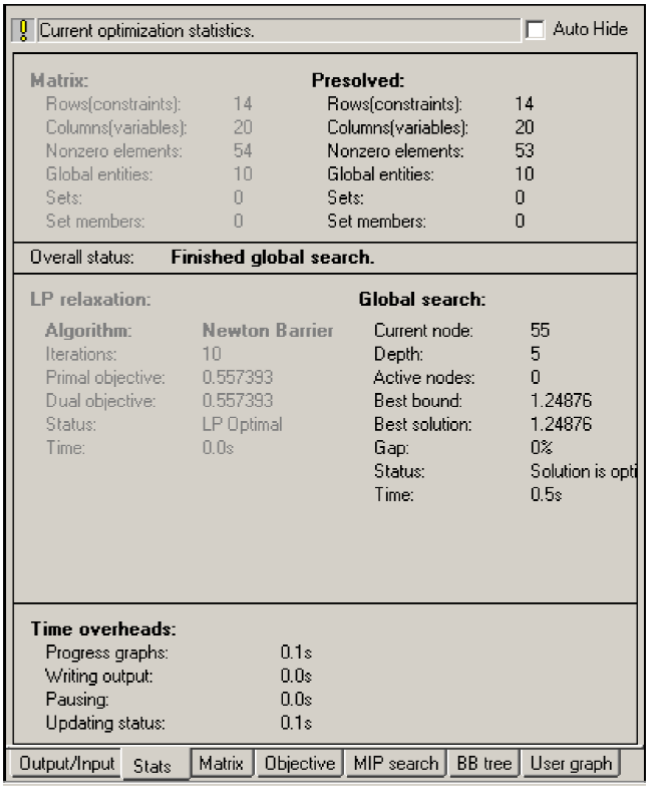


Figure 7.2: Detailed MIQP solution information

これは、おそらく LP ソリューション・アルゴリズムを除くと、MIP 統計と、非常によく似ています。このケースでは、この問題の LP 緩和は、Newton-Barrier algorithm で解かれています。Branch-and-Bound ツリーが、複数のノードを持っているので、結果として得られた探索ツリーを見ることもできます（ウィンドウ BB tree）。

MSI 株式会社 Copy right©2015 All rights reserved

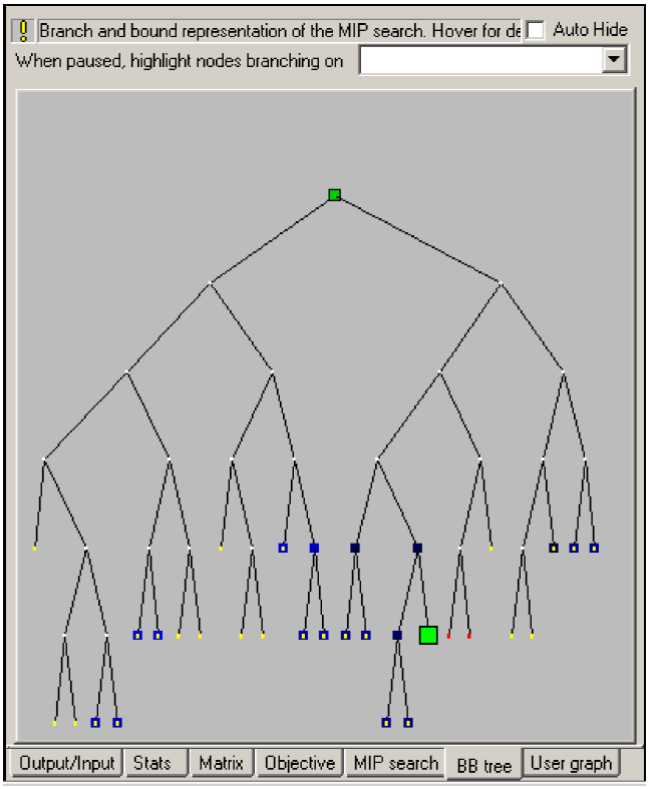


Figure 7.3: MIQP Branch-and-Bound search tree

探索の中で、2 つの整数実行可能解が見つかりました（緑色の四角形）。最もよい解は、少し大きめのサイズの四角形でハイライトされています。ウィンドウの Objective を開くと、見つかった 2 つの解の詳細が見られます（Figure 7.4）

MSI 株式会社 Copy right©2015 All rights reserved

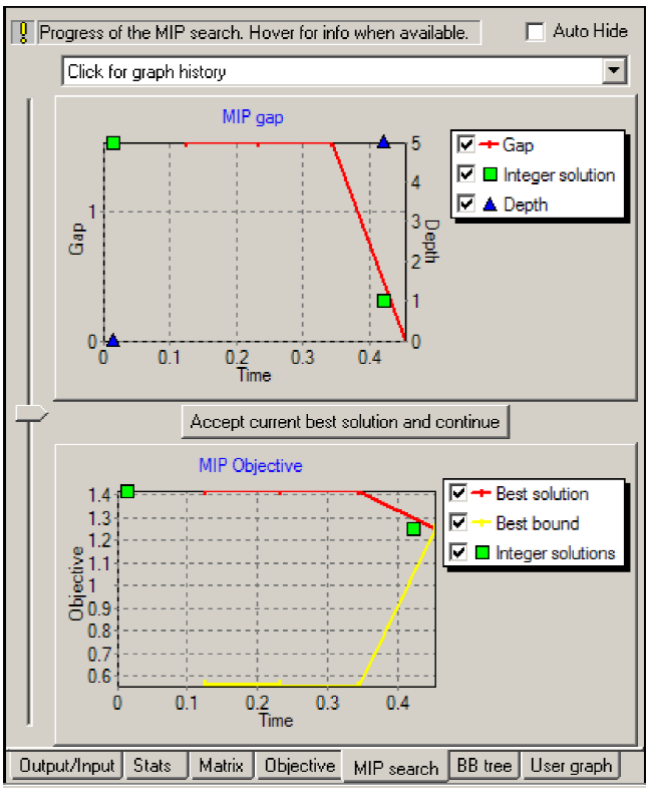


Figure 7.4: MIQP solutions

このウィンドウの上半分は、得られた MIP 解と LP 緩和解の間のギャップを示しています。このウィンドウの下半分は、見つけられた解の絶対値、および、残余のオープン・ノードの LP 緩和から得られた「最良の下限バウンド曲線」のグラフを示しています。この曲線は、終わりの部分で、最良解の値に繋がっています。これは、この解の optimality が証明されたことを意味しています。optimality の証明や最良解を見つけることはできませんが、探索を例えば、あるノード数だけ行った後、中止するという選択を行うこともできます。

MSI 株式会社 Copy right©2015 All rights reserved

第 8 章 ヒューリスティックス

この章では、バイナリ変数を固定して、ヒューリスティックな方法で解を得ることについて説明します。説明の中には下記の事柄が含まれています。

- サブルーチンを持つ Mosel モデルを構造化する
- パラメータを設定する、ベイシス(基底)をセーブし、ベイシスを戻す、変数のバウンド(bound)を修正することにより、Xpress-Optimizer とインタラクトして、ヒューリスティックな方法で解を得る手順

第 13 章で、BCL を使って、同じヒューリスティックをどのように実行するかについて説明します。

8.1 バイナリ変数を固定して行うヒューリスティック

ここで行おうとするヒューリスティックは、下記のステップが必要です。

1. LP 緩和を解き、最適解のベイシスをセーブする。
2. 丸めのヒューリスティックス(rounding heuristic):もし、対応する変数 frac の値が 0 に近ければ、バイナリ変数 buy を 0 に固定する。また、対応する変数 frac の値が、比較的に大きければ、バイナリ変数 buy を 1 に固定する。
3. こうして得られた MIP 問題を解く。
4. ここで、整数実行可能解が得られたら、その中の最良解をセーブする。
5. すべての変数のバウンドをもともとのバウンドにリセットして、もともとの問題を コンピュータにリストアして、セーブしておいたベイシスをロードする。
6. もともとの MIP 問題を解くが、このとき、ヒューリスティック・ソリューションの最良解の値を「カットオフ値」として使う。

上のステップ 2

割合を示す変数 frac は 0.3 という上限を持つので、「比較的大きな値」として、例えば、0.2 を使います。普通、一般に、個々のアプリケーションでは、バイナリ変数にたいしては、$1-\varepsilon$, を使います。ここで、ε は10^{-5} のような、非常に小さな値です。

ステップ 6

「カットオフ値」を設定する意味は、解の探索を行うときに、この値よりも良い値を持つソリューションを見つけようとすればよく、したがって、あるノードの LP 緩和の解が、この値よりも悪ければ、そのノードは切れます。なぜなら、このノード、および、その先に繋がる子孫のノードの整数実行可能解は、このノードの LP 緩和解よりも悪くなるからです。

MSI 株式会社 Copy right©2015 All rights reserved

8.2 Mosel で実行する

第 2 章で作成したモデル (file folioheur.mos) を、問題を新しい観点から見ると、下記の変更を行う必要があります。

ここで、第 6 章の MIP 1 モデルを使い、バイナリ変数を固定して、ヒューリスティックな方法で解を得る方法を実行します。サブルーチン（より正確に proedure）という形式で定義されたヒューリスティックにより、モデル自体には、最小限の変更を行います。ここでは、最初の部分で、forwardというキーワードを使ってこの手続を宣言し、標準のコールで maximization 機能を呼び出して問題を解く前に、このヒューリスティック・ソリューションを実行します。ソリューションの印刷も、この変化に対応して変更されています。

```
model "Portfolio optimization solved heuristically"
 uses "mmxprs"                      ! Use Xpress-Optimizer

 parameters
  MAXRISK = 1/3                     ! Max. investment into high-risk values
  MAXVAL = 0.3                      ! Max. investment per share
  MINAM = 0.5                       ! Min. investment into N.-American values
  MAXNUM = 4                        ! Max. number of assets
 end-parameters

 forward procedure solve_heur       ! Heuristic solution procedure

 declarations
  SHARES: set of string             ! Set of shares
  RISK: set of string               ! Set of high-risk values among shares
  NA: set of string                 ! Set of shares issued in N.-America
  RET: array(SHARES) of real        ! Estimated return in investment
 end-declarations

 initializations from "folio.dat"
  RISK RET NA
 end-initializations

 declarations
  frac: array(SHARES) of mpvar      ! Fraction of capital used per share
  buy: array(SHARES) of mpvar       ! 1 if asset is in portfolio, 0 otherwise
 end-declarations
```


MSI 株式会社 Copy right©2015 All rights reserved

```
! Objective: total return
 Return:= sum(s in SHARES) RET(s)*frac(s)

! Limit the percentage of high-risk values
 sum(s in RISK) frac(s) <= MAXRISK

! Minimum amount of North-American values
 sum(s in NA) frac(s) >= MINAM

! Spend all the capital
 sum(s in SHARES) frac(s) = 1

! Upper bounds on the investment per share
 forall(s in SHARES) frac(s) <= MAXVAL

! Limit the total number of assets
 sum(s in SHARES) buy(s) <= MAXNUM




 forall(s in SHARES) do
  buy(s) is_binary
  frac(s) <= buy(s)
 end-do

! Solve problem heuristically
 solve_heur

! Solve the problem
 maximize(Return)

! Solution printing
 if getprobstat=XPRS_OPT then
  writeln("Exact solution: Total return: ", getobjval)
  forall(s in SHARES) writeln(s, ": ", getsol(frac(s))*100, "%")
 else
  writeln("Heuristic solution is optimal.")
 end-if
```


MSI 株式会社 Copy right©2015 All rights reserved

```
!-------------------------------------------------------------------

 procedure solve_heur
  declarations
   TOL: real                        ! Solution feasibility tolerance
   fsol: array(SHARES) of real      ! Solution values for 'frac' variables
   bas: basis                       ! LP basis
  end-declarations

  setparam("XPRS_VERBOSE",true)     ! Enable message printing in mmxprs
  setparam("XPRS_CUTSTRATEGY",0)    ! Disable automatic cuts
  setparam("XPRS_HEURSTRATEGY",0)   ! Disable automatic MIP heuristics
  setparam("XPRS_PRESOLVE",0)       ! Switch off presolve
  TOL:=getparam("XPRS_FEASTOL")     ! Get feasibility tolerance
  setparam("ZEROTOL",TOL)           ! Set comparison tolerance

  maximize(XPRS_LPSTOP,Return)      ! Solve the LP problem
  savebasis(bas)                    ! Save the current basis

 ! Fix all variables 'buy' for which 'frac' is at 0 or at a relatively
 ! large value
  forall(s in SHARES) do
   fsol(s):= getsol(frac(s))        ! Get the solution values of 'frac'
   if (fsol(s) = 0) then
    setub(buy(s), 0)
   elif (fsol(s) >= 0.2) then
    setlb(buy(s), 1)
   end-if
  end-do

! Reset variables to their original bounds
 forall(s in SHARES)
  if ((fsol(s) = 0) or (fsol(s) >= 0.2)) then
   setlb(buy(s), 0)
   setub(buy(s), 1)
  end-if

 loadbasis(bas)                     ! Load the saved basis



  if ifgsol then                      ! Set cutoff to the best known solution
   setparam("XPRS_MIPABSCUTOFF", solval+TOL)
  end-if
 end-procedure

end-model
```

このモデルを理解するには、もっと、詳細な説明が必要でしょう。

8.2.1 サブルーチン

Mosel のサブルーチンは、モデル自体と類似した構造を持っています。procedure は、キーワード procedure で始まり、その後ろに、procedure の名前が続き、end-procedure で終わります。同様に、関数は、キーワード function で始まり、その後ろに、function の名前が続き、

MSI 株式会社 Copy right©2015 All rights reserved

end-function で終わります。この両方のタイプサブルーチンは、アーギュメントのリストからなり、また、function の場合は、さらに、return type も示されなければなりません。例えば、

```
function myfunc(myint: integer, myarray: array(range) of string): real
```

という関数は、インプットのアーギュメントとして、整数 integer とストリングを持つ配列 array(range)ofstring を持ち、そして real を返えします。上で見るように、サブルーチンは

1 つ(または、いくつかの)の declarations ブロックを持てます。サブルーチンで定義されたオブジェクトはローカルに有効であるだけで、サブルーチンの終わりで削除されます。サブルーチンの定義は verloaded が可能です。すなわち、単一のサブルーチンは、異なるアーギュメントの組み合わせを取ることができます。Mosel、および、そのモジュールで定義されたサブルーチンは、それがどのようなものであれ、新しい定義が、少なくとも 1 のアーギュメントが、既存の定義と異なれば、overloaded が可能です。サブルーチン定義の詳細な説明と例については、「Mosel User Guide」を参照して下さい。

8.2.2 Optimizer のパラメータと機能

パラメータ

ヒューリスティックのソリューションは、Xpress-Optimizer のパラメータ設定を使って始まります。IVE では、すべてのパラメータ、および、モジュールから提供される他の機能は、module browser でリストできることを記憶して下さい。それには、Modules >> List available modules を選択するか、または button、 をクリックして下さい。Optimizer のパラメータの詳細な説明は、「Optimizer Reference Manual」を参照して下さい。すべてのパラメータは、Mosel のサブルーチン setparam と getparam によりアクセスできます。(この例では、最初モジュール mmxpr でアウトプットの印刷が行われるようにしています。)その結果、Output/input window では、プリントされるものよりさらに多くの情報が表示されます。Optimizer からのアウトプットは、カラーバーでハイライトされます(LP 部分は青、MIP 部分はオレンジ)。

カットの自動生成(パラメータ XPRS_CUTSTRATGY) およびヒューリスティックス(パラメータ XPRS_HEURSTRATEGY)の切り替えはオプションですが、ここの問題では、presolve 機能(パラメータ XPRS_PRESOLVE でセットして行うマトリックスに加える前処理で、モデルサイズを小さくし、ニューメリカルな特性を改善する)を行わないようにしないといけません。なぜなら、解く過程で Optimizer の中の問題とインタラクトしますが、Optimizer によりマトリックスが変更されていない場合のみ、インタラクションが正しく行えるからです。

パラメータの設定に加え、Xpress-Optimizer が使うフィージビリティ・トレランスも調べる必要がある場合もあります。Xpress-Optimizer は、整数のフィージビリティ、ソリューション・フィージビリティのチェックに、デフォルトで10^{-6}のオーダーのトレランス値を使います。例えば、パーフォーマンスの比

MSI 株式会社 Copy right©2015 All rights reserved

較をして解を評価するときには、トレランスを考慮することが重要です。

最適化ステートメント

ここでは、XPRS_LPSTOPという追加的なアーギュメントを持つ、maximization procedure の新しいバージョンを使いますが、これは、一番、上のノードの LP 緩和のみを解きたいこと(そして、まだ、この段階では、MIP 問題全体のソリューションを求めていないこと)を意味します。アルゴリズムを停止した場所から継続してMIPで解くには、アーギュメントXPRS_CONTを使用します。これは、over loaded subroutine の定義の例です。

セーブ、および、ベイシスのロード

ソリューションプロセスを加速するため、最初の LP 緩和を解いた後に、問題に何の変更も加える前に、シンプレックス・アルゴリズムのベイシスを(メモリー内に)セーブします。このベイシスは、オリジナルな問題をリストアしたら、最後に、再びロードされます。こうして、MIP のソリューション・アルゴリズムは、LP 問題を、「最初から(fromscratch)」から解かなくてもよくなります。こうして、MIP のソリューション・アルゴリズムは、ヒューリスティックにより「中断された状態」から、計算プロセスを続ければよいことになります。

バウンドの変更

すでに、問題が、Optimizer にロードされているとき(例えば、optimization statement の実行や、loadprob を明確に呼び出した後)、setlb、setub を経由して、バウンドの変更は、直接、Optimizer に渡されます。しかし、制約式や変数の追加や削除のなど、モデルを変更する場合は、問題を再ロードする必要があります。

この例で使われている Optimize の諸機能の詳細については、「Mosel Language Reference Manual のモジュール mmxp についての説明を参照して下さい。

MSI 株式会社 Copy right©2015 All rights reserved

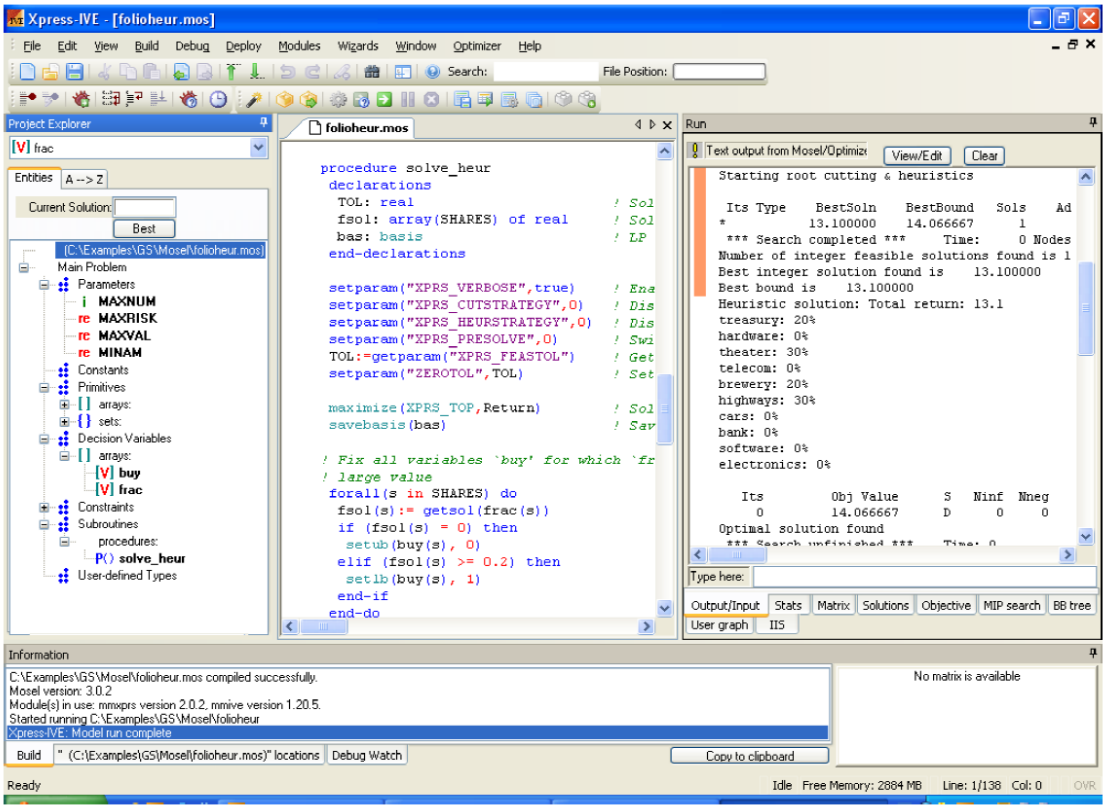


Figure 8.1: Optimizer output display

8.2.3 トレランスを比較する

Optimizer の実行可能解の許容値を得た後、Mosel の Comparison tolerance (ZEROTOL)を TOL 値にセットします。この作業を行うことにより、実際に fsol(s)=0 であることが判別できます。fsol(s)が TOL と TOL 間に位置する場合 fsol(s)>=0.2 が満たされており、fsol(s)の値は最小でも 0.2=TOL であるということを意味します。

常に非常に小さいデェフォルト値を持つ許容値を用いて Mosel で比較することができます。パラメーターを Optimizer の実行可能解の許容値に再セットすることにより Mosel を使い Optimizer と同様、ソリューションの値を評価することができます。

MSI 株式会社 Copy right©2015 All rights reserved

第 9 章 Mosel モデルをアプリケーションに埋め込む

Mosel モデルは、しばしば、使い勝手をよくするために、アプリケーションに埋め込まれて使われます。この章では、下記のことについて説明します。

- deployment テンプレートをどのようにして生成するか
- BIM ファイルの意味と使い方
- パラメータ化されたモデルと BIM ファイル
- どのようにして、Mosel、および、IVE と、マトリックス・ファイルをインポートし、エクスポートするか
- モデルファイルから Optimization Modeler アプリケーションを作成する

Mosel モデルは、しばしば、使い勝手をよくするために、アプリケーションに埋め込まれて使われます。この章では、下記のことについて説明します。

9.1 deployment テンプレートをどのようにして生成するか

Deploy ≫ *Deploy*を選択する、または deploy ボタン をクリックします。開いた選択ウィンドウの、(Figure9.1)Run Mosel モデル下でオプションの Visual Basic を選択します。(C、Java、またはその他のサポートしている言語での開発も手続きは同様です。)

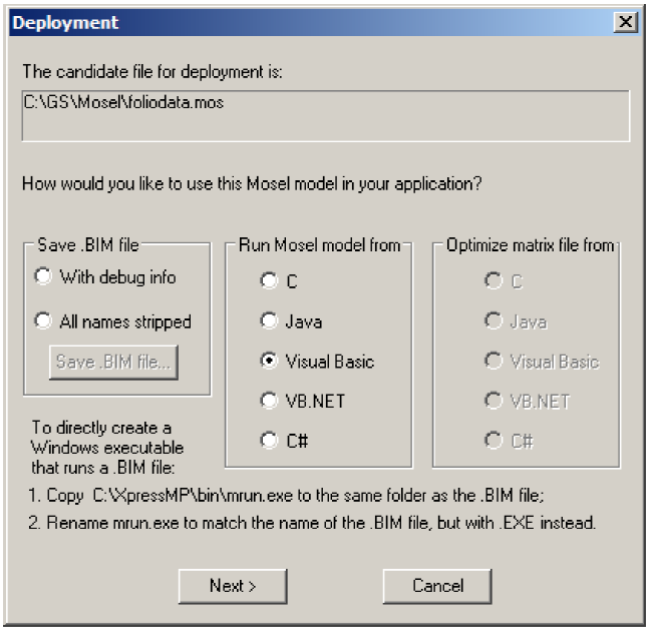


Figure 9.1: Choosing the deployment type

Next ボタンをクリックすると、新しいウィンドウが開き、そこにはコードが表示されています(Figure 9.2)。

MSI 株式会社 Copy right©2015 All rights reserved

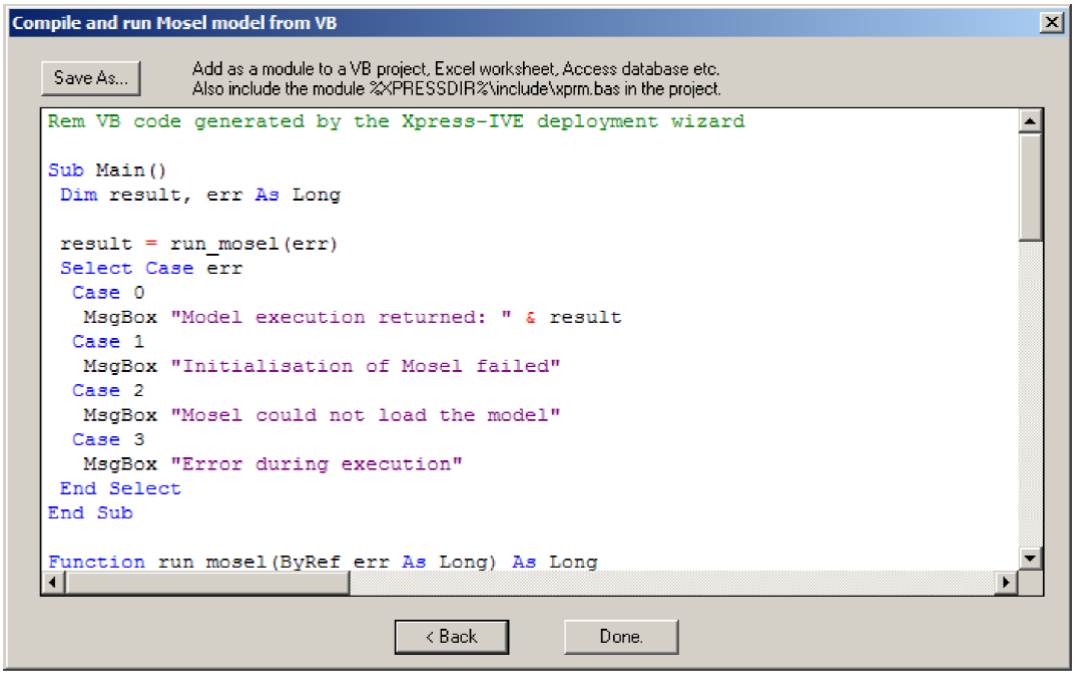


Figure 9.2: Code preview

Saveas ボタンを使い、名前(`foliorun.bas`)と新しいファイルの場所をセットします。このファイルを、Visual Basic Project や MS Excel worksheet などに、モジュールとして追加できます。Visual Basic Project の場合は、モデルをランするためには、モジュール xprm.bas も含めなければなりません。

Mosel が使えるシステムでは、deployment ウィザードによって作成された C、または、Java のどのようなプログラムも実行できます。

9.2 BIM ファイル

ここで、たった今、生成したテンプレートは、Mosel モデルが、BIM ファイルという形式であると仮定しています(BInary Model file)。BIM ファイルは、Mosel が使えるすべてのプラットフォーム全体で、ポータブルな「.mos モデルファイル」のコンパイルされたバージョンです。これには、外部のファイルから読まれたデータは含まれていません。これらのデータは別個のファイルから入れ込まれます。こうすることによって、同じ BIM ファイルを使って、いろいろなデータセットで実行することができるような仕組みになっています。について可能にします(下の 9.3.2 パラメータを参照)。

BIM ファイルを生成するには、Build >>Compile を選択するか、または、button . をクリックして下さい。そうすると、BIM ファイルは、Mosel ファイルと同じディレクトリに、ファイル名に('. mos' ではなく)拡張子'. bim' を持つファイルとして作成されます。また、deployment ウィザードを使い、今回は、別のオプション 'Produce standalone .BIM file(with debug information)'や'Produce standalone .BIM file (all names stripped)' などを使う必要性が出てくる場合もあるでしょう。最初のオプションは IVE デフォルトであり、2 番目のオプションは、特に、モデルに含まれている知的所有権

MSI 株式会社 Copy right©2015 All rights reserved

を保護したいような場合にお勧めできます。これを使った場合、モデルで使われているすべての名前を取り除いてくれます。また、アプリケーションから、直接、Mosel のソースファイル(. mos) を実行することもできます(次のセクションを参照)。この場合、BIM ファイルを生成する必要はありません。

9.3 テンプレートを修正する

9.3.1 Mosel モデルの実行

エラーチェックを外すと、生成されたコードは以下のようになります。

```
Sub Main()
 Dim result As Long
 Dim model

' Initialize Mosel
 XPRMinit

' Load compiled model
 model = XPRMloadmod("foliodata.bim", "")

' Run the model
 XPRMrunmod model, result, ""

' Unload the model
 XPRMunloadmod (model)
End Sub
```

もし、BIM ファイルを別に作成することを望まない場合は、例えば、以下に示すコード部分のように、直接、Mosel モデル`foliodata.mos`をコンパイルし、ロードし、実行できます。

```
Sub Main()
 Dim result As Long
 Dim model

' Initialize Mosel
 XPRMinit

' Execute = compile/load/run a model
 XPRMexecmod "", "foliodata.mos", "", result, model

' Unload the model
 XPRMunloadmod (model)
End Sub
```

9.3.2 パラメータ

第 4 章で、IVE や Mosel のスタンドアロン・バージョンで実行するときに(例えばバッチファイルまたはスクリプトにおいて)、どのようにして、パラメータ設定を修正するかについて学びました。モデルパラメータは、モデル・ソースを変更することなく、多くのいろいろな問題を解くことができるように、Mosel モデルや BIM ファイルがアプリケーションに埋め込まれている場合にも、リセットできるようになってい

MSI 株式会社 Copy right©2015 All rights reserved

ます。ここでは、モデル foliodata.mos の結果ファイルの名前と 2 つの numerical パラメータのセッティングを修正します。他のモデルパラメータは、すべて、モデル内の定義で指定されたそれらのデフォルト値を取ります。

```
Sub Main()
 Dim result As Long
 Dim model

' Initialize Mosel
 XPRMinit

' Execute model with changed parameters
 XPRMexecmod "", "foliodata.mos", "OUTFILE=result2.dat,MAXRISK=0.4,MAXVAL=0.25",
  result, model
```

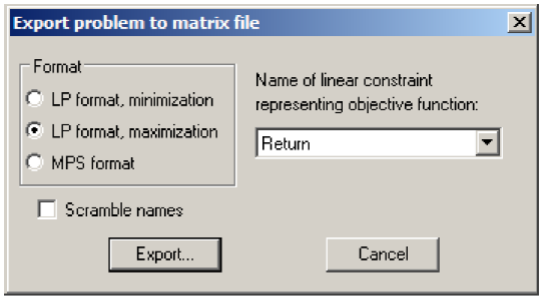


Figure 9.3: Matrix export window

```
' Unload the model
 XPRMunloadmod(model)
End Sub
```

同様に、BIM ファイルのパラメータ設定を変更して実行するには、単に、`run' function call にパラメータを追加するだけです。

```
XPRMrunmod model, result, "OUTFILE=result2.dat,MAXRISK=0.4,MAXVAL=0.25"
```

9.3.3 VB アウトプットの出力先

Visual Basic には、標準の出力チャネルがないので、Mosel の VB インタフェースを使い、Mosel より出力されたすべてのアウトプットをリダイレクトできるようになっています。モデルのすべてのアウトプットを、ファイル folioout.txt にリダイレクトするには、「Mosel モデルの実行(the execution of theMosel model)」を、以下によって囲んでください。

```
' Redirect all output to the file "folioout.txt"
 XPRMsetdefstream vbNull, XPRM_F_OUTPUT, "folioout.txt"
```

ここでの例では、`foliodata.mos` の中ですでにアウトプットがファイル`result.dat` に、モデル自身によりリダイレクトされているので、Mosel により生成されるエラーメッセージをすべて取り出せることが、もっと、重要かもしれません。上の 2 つの行で、`XPRM_F_OUTPUT` を`XPRM_F_ERROR`

MSI 株式会社 Copy right©2015 All rights reserved

に置き換え、ファイルにエラーストリームをリダイレクトして下さい。

9.4 マトリックス・ファイル

9.4.1 マトリックスのエクスポート

Mosel プログラム内で、Xpress-Optimizer で最適化プロセスが始められる場合や、または、ソリューション・プロシージャーが、Mosel モデルが埋め込まれたアプリケーションの一部である場合は、問題マトリックスは、メモリー内でソルバーにロードされ、ランニングタイムが掛かってしまう、ファイルに書くようなことはしません。しかし、場合によっては、マトリックス・ファイルを作成することが必要な場合もあるでしょう。Xpress では、ユーザは、2 つのマトリックス・フォーマットを選択できます。すなわち、extended MPS、および、extended LP format です。この後者は、制約式が数学的な形式で書かれているので、直感的で読み取り易すいフォーマットです。
Mosel と IVE では、マトリックスの生成をいくつかの方法で行えます。

1. IVE のメニューを使う

モデルを実行した後に、Build >> Export matrix file を選択するか、または、button をクリックして下さい。出てくる matrix export window の中で、ファイル・ネームを選択し、目的関数をタイプして選択する

2. モデルファイルの中の matrix generation statement を使う問題の MPS マトリックスを作成するには、下の行を追加する。

```
exportprob(EP_MAX, "folio", Return)
```

LPマトリックス(ある箇所を最大化するために)には、下記の行を入れる

```
exportprob(EP_MAX, "folio", Return)
```

いずれも、optimization statement の直前か、optimization statement の代わりにこれを入れる。

3. model file を実行した後に、アプリケーションから作成する。この方法は、C プログラムにおいてのみ可能

9.4.2 マトリックスのインポート

また、マトリックスをロードし、そして、解くこともできます。Build >>Optimize matrix file を選択するか、または、button をクリックして下さい。そうすると、以下のウィンドウが開きます。

MSI 株式会社 Copy right©2015 All rights reserved

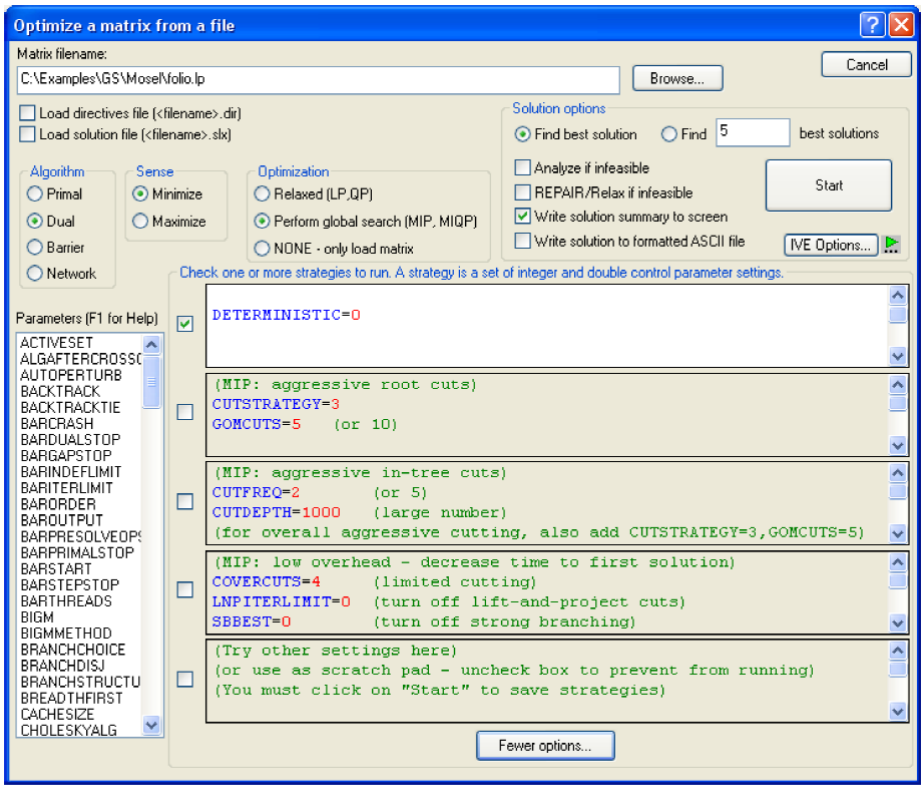


Figure 9.4: Matrix import window

IVE を使ってマトリックスを解くとき、マトリックスに Mosel モデルの完全な情報が含まれていないと、対応する情報がないディスプレイは使えなくなります。しかし、実行ログ、問題統計などを表示する右側のウィンドウはアクセスできる状態で残ります。

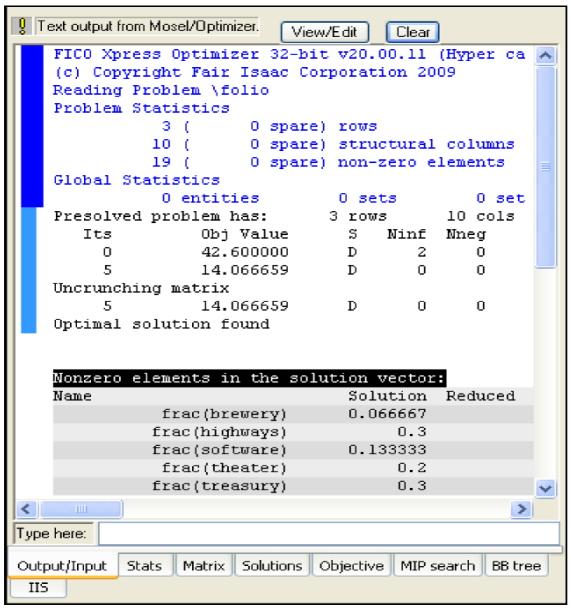


Figure 9.5: Matrix optimization output

MSI 株式会社 Copy right©2015 All rights reserved

9.5 Optimization Modeler の開発

Xpress client-serverアーキテクチャで最適化モデルを開発するためにマルチ・ユーザアプリケーション内にOptimization ModelerでMoselモデルを組み込めます。Optimization Modeler GUIによってユーザは様々なシナリオやモデル構造を評価するために、モデルに直接アクセスしなくてもMoselモデルを確認することができます。下記の解説はOptimization Modelerデスクトップ・インストールのデベロッパーバージョンで開発することを前提としています。

9.5.1 モデルファイルの準備

Optimization Modeler内にMoselモデルを組込むために、Moselモデルを若干修正し、MoselとOptimization Modelerの接続を構築します。最初に、設定に必要な追加的機能を提供しているパッケージmminsightをロードします。Optimization Modelerがデータシナリオを管理するので(またはベースラインの実行と呼ぶ)シナリオをOptimization Modelerにロードする際、オリジナルソースからデータを読み込むだけで準備が完了します。(下記のモデルでinsight_use_original_dataのテストを使用して変更します。)シナリオデータはinsight_end_initializationsでマークした挿入ポイントでOptimization Modelerから直接インプットされ、異なる形式になります。さらに、最適化を開始するソルバーの呼び出しは、insight_minimize / insight_maximizeで置換します。修正したモデルファイルfolioinsight.mos (元のモデルファイルfoliodata.mos)は下記の通りです。

```
model "Portfolio optimization with LP"
 uses "mmxprs"                        ! Use Xpress-Optimizer
 uses "mminsight"                     ! Use Optimization Modeler

 parameters
  DATAFILE= "folio.dat"               ! File with problem data


 MAXRISK = 1/3                       ! Max. investment into high-risk values
 MAXVAL = 0.3                        ! Max. investment per share
 MINAM = 0.5                         ! Min. investment into N.-American valu
end-parameters

declarations
 SHARES: set of string               ! Set of shares
 RISK: set of string                 ! Set of high-risk values among shares
 NA: set of string                   ! Set of shares issued in N.-America
 RET: array(SHARES) of real          ! Estimated return in investment
end-declarations

if insight_use_original_data then    ! Data for baseline
 initializations from DATAFILE
  RISK RET NA
 end-initializations
end-if
```


MSI 株式会社 Copy right©2015 All rights reserved

```
 declarations
  frac: array(SHARES) of mpvar       ! Fraction of capital used per share
  Return, LimitRisk, LimitAM, TotalOne: linctr   ! Constraints
 end-declarations

! Input data injection point for all scenarios other than baseline
 insight_end_initializations

! Objective: total return
 Return:= sum(s in SHARES) RET(s)*frac(s)

! Limit the percentage of high-risk values
 LimitRisk:= sum(s in RISK) frac(s) <= MAXRISK

! Minimum amount of North-American values
 LimitAM:= sum(s in NA) frac(s) >= MINAM

! Spend all the capital
 TotalOne:= sum(s in SHARES) frac(s) = 1

! Upper bounds on the investment per share
 forall(s in SHARES) frac(s) <= MAXVAL

! Solve the problem through Optimization Modeler
 insight_maximize(Return)

end-model
```

注:ここで、このモデルから解のアウトプット全てを削除します。結果を表示させるために Optimization Modelerを使用しています。

9.5.1.1 プロジェクト・アーカイブ

Optimization Modelerは完全な形でモデルを提供します。セクション9.2 BIMファイルでは、モデルソースからBIMファイルを生成する方法を解説しています。Optimization Modelerは分散されているアーキテクチャでMoselモデルを実行します。(モデルファイルが格納されているマシンからのインプットは行えません。)Optimization Modeler project archive内にモデルで使用した他のインプットデータファイルを含めることを推奨します。プロジェクト・アーカイブはBIMファイルとオプションのサブディレクタ `model_resources` (データファイル)、`client_resources` (custom view definitions) および `source` (Moselモデルソースファイル)を持つZIPファイルです。Xpress例題は、ファイル名 `folioinsight.bim` でZIP archive `folioinsight.zip` と、サブディレクトリ `model_resources.` 内にデータファイル `folio.dat` で提供しています。

9.5.2 Optimization Modeler GUI を起動させる

Optimization Modeler Analyst Clientは、Windowsのスタートメニューから、またはコマンド行から起動させることが可能です。Windowsのスタートメニューから clientを起動させるには、Start>>FICO >>Xpress>>Optimization Modeler Analyst Clientを選択してください。コマンド行からclientを起動させるにはコマンドプロンプトを開き、Xpress installationのbin サブディレクトリに入ります。

MSI 株式会社 Copy right©2015 All rights reserved

続いて、下記のコマンドを入力してください。

```
Optimization Modeler Analyst Client
```

プログラムはOptimization Modeler serverに割り当てられるか、またはServerに格納することができない場合は部分的に起動するため、Optimization Modeler Analyst Clientの設定に時間がかかる場合があります。次の図のように、アプリケーションのインストールが成功すると、Optimization Modeler エントリ画面が表示されます。

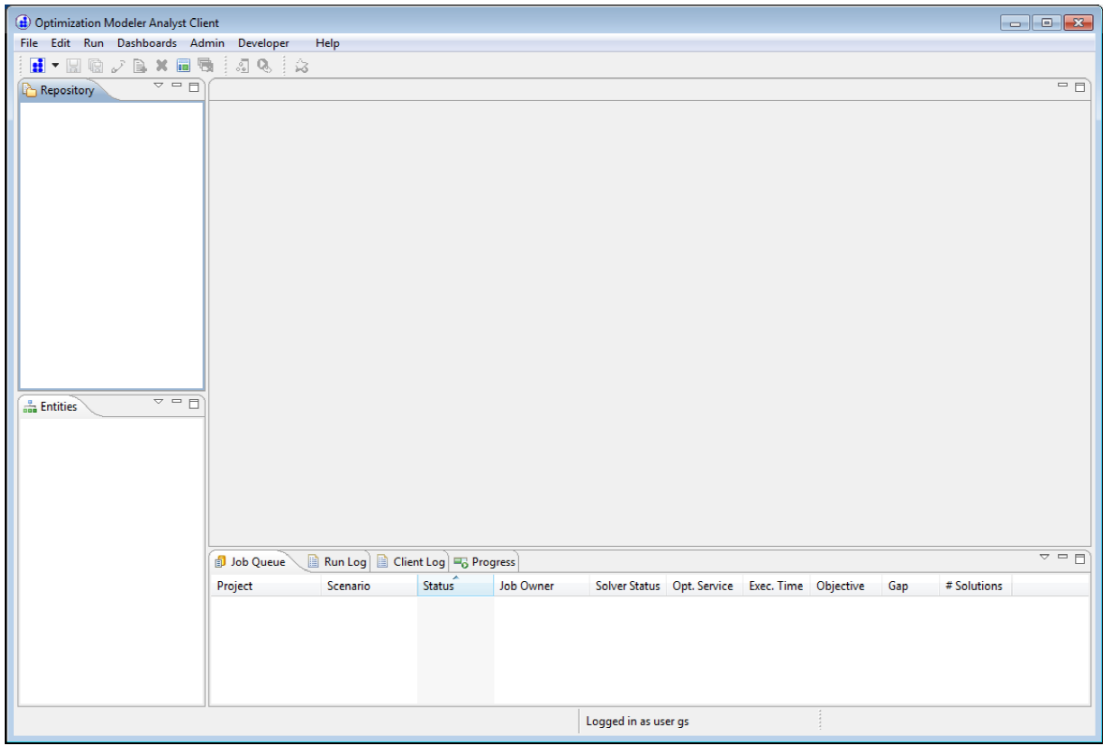


Figure 9.6: Optimization Modeler entry screen

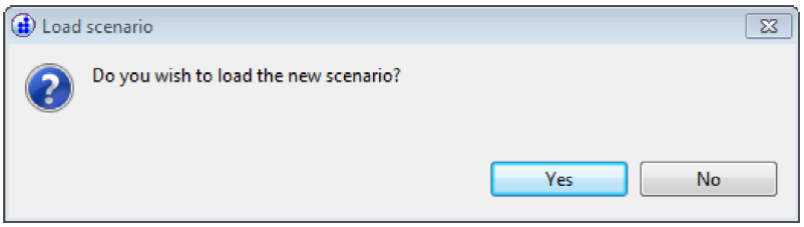
トップにあるOptimization Modeler Menuと Toolバーを確認ください。ウインドウの左側は、*Repository pane*およびEntities paneが表示されています。下部には*Run pane*が表示されています。灰色の部分はデータを確認したり、モデル情報が開かれている場合はここに表示されます。アプリケーションを使用した作業中にこのウインドウのレイアウトを修正した場合は、ボタン を選択すると、再び、元のレイアウトに戻すことができます。

9.5.2.1 新規の Optimization Modeler プロジェクトをスタートする

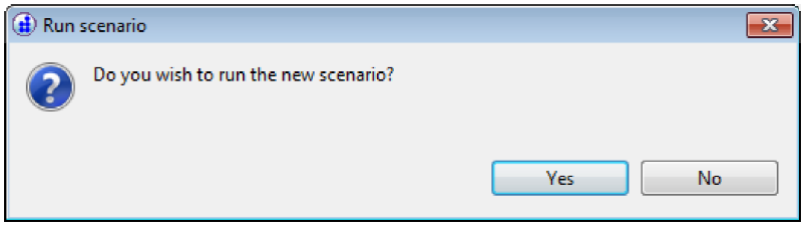
ここで、事前に準備したプロジェクト・アーカイブ `folioinsight.zip`をロードしていきます。

1. *Repository* paneをクリックします
2. ファイル参照ログを開くために、menu *File>>New >>New Project*を選択してください。
3. プロジェクト・アーカイブ `folioinsight.zip` を選択し、Openボタンで確認します。
4. 数回、二者択一する画面が表示されます。毎回、Yesを選択し、作業を続行してください。

MSI 株式会社 Copy right©2015 All rights reserved



シナリオのローディングは外部データにインプットするため、部分的なモデル実行を行います。
(`insight_end_initializations` まで)



シナリオをロードしたときに、インポートしたデータのコピーを使用して、*scenario run*は完全なモデルを実行します。
スクリーン下部にある*Job Queue* paneは、Moselモデルが実行されたこと、およびOptimization Modelerによる実行の準備が完了したことを示すために下記のディスプレイに変更されます。

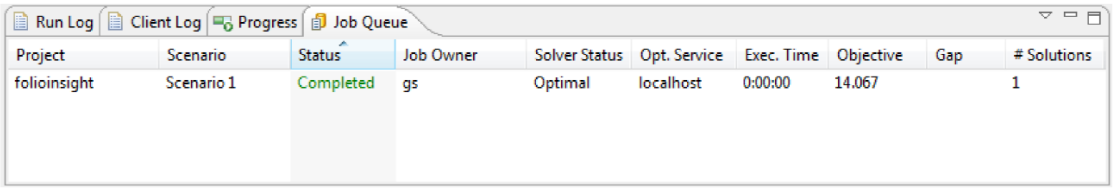


Figure 9.7: Optimization Modeler job queue pane

9.5.2.2 Optimization Modeler プロジェクトを調査する

下記の図に表示されている*Repository* paneは、拡張エントリです(左のマウスキー)

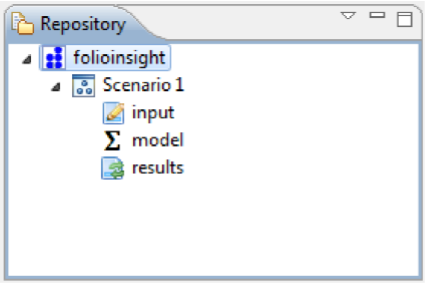


Figure 9.8: Optimization Modeler repository pane

シナリオ1が選択された場合、Entities paneに通常、下記のコンテンツを得ます。

MSI 株式会社 Copy right©2015 All rights reserved

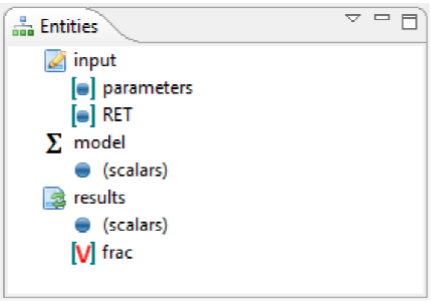


Figure 9.9: Optimization Modeler entities pane

デフォルト設定を使用すると、Optimization Modelerはモデルオブジェクトタイプの`string`、`integer`または`real`を分類し、オブジェクトタイプは、`mpvar`または`linctr`をインプットします。インプット、モデル上または*Repository*や*Entities* panesのエントリの結果をダブルクリックすると、中心の*View* paneにオブジェクトタイプに関するbuilt-in default view(エンティティ:表)が開きます。コンテクストメニューを表示するために、Repository内または*Entities* panes内の別のノードを右クリックします。('Delete','Rename',やXXXを持つOpenリストのように)`folioinsight`モデルを調査するために、下記のコンテクストメニューを使用して*entities* paneから下記のviewを開くことを推奨しています。

1. インプット・パラメータ(input>>Parameters):テーブル(一覧表)で開きます。この数値は、シナリオの実行で使用した実行時間のパラメータ値です。(NB:データファイル名の設定はシナリオのロード中に外部データを読み込む際にのみ、必要です)
2. 結果(results>>scalars):テーブル(一覧表)で開きます。モデルの制約式に関するソリューション情報の詳細(全てスカラー)が表示されます。
3. 結果(results>>frac):テーブルで開きます。ここでは、意思決定変数`frac`の解の値が表示され、追加カラムに配列 RET の値も表示されます。(Optimization Modeler が自動的に選択したエントリと同じインデックスを共有するシナリオデータから全配列を追加します。)
4. 結果(results>>frac):チャートで開きます。チャートの設定は、メニューChart >>Settings を選択し、(またはツールバーからボタン をクリック)Chart type select "Pie"にあるドロップダウン・リストを選択>>OK ボタンで設定します。(別のチャート設定手順:view Table にあるヘッダーfrac を選択し、コンテクストメニューから New Chart を選択します。)

上記に記載した全 view は、センターの表示画面内に展開されます。このウインドウ内の重複した view を準備するには、マウスの左クリックを押しながら view のラベルタグを選択し、view を移動したい場所(ボーダーの線が表示されます。)にドラッグします。例:view のラベルチャート:frac を選択し、view を移動したい場所を示す灰色のボーダーの線が見えるまで、Optimization Modeler の右側のボーダーの方にドラッグします。この操作により、アプリケーション画面は下記のように変更されます。:

MSI 株式会社 Copy right©2015 All rights reserved

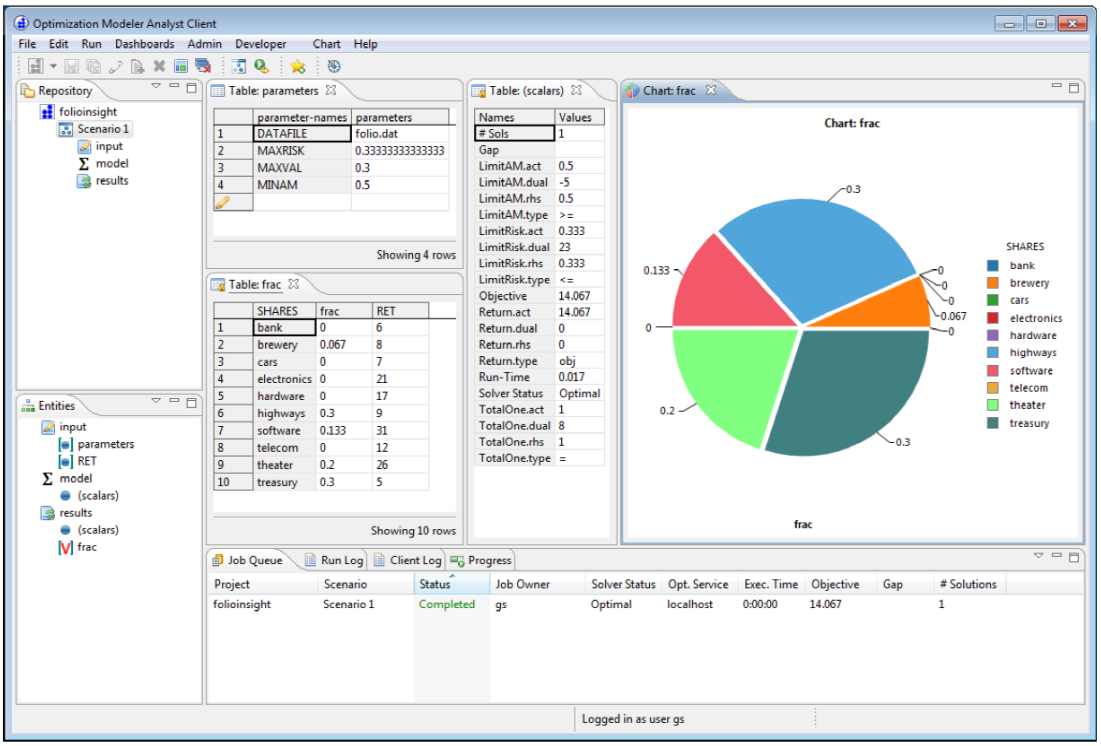


Figure 9.10: Optimization Modeler application window for the 'folioinsight' project

9.5.2.3 モデルデータを修正・編集する

viewを作業しやすいように調整したので、スクリーン上の表示画面に表示されたインプットデータの編集作業を行う準備が完了しました。ユーザが一貫性のないアプリケーション・データを作成することを防ぐために、Optimization Modelerにロックシステムが実装されています。編集したいviewのアイコンが黄色のカギで覆われている場合、viewを編集可能にするために、アイコンをクリックするか、メニュー*Edit>> Edit*を選択します。
変更したプロジェクトをセーブする、しないに関わらず、この作業が促される場合があります。

値の編集：テーブル内のセルをクリックし、ドロップダウン・リストから選択するか、値を入力します。
配列dataに新しいエントリを追加：テーブルの下部にあるブランクの列をクリックします。例：view Table: parameters内のMAXVAL値を修正する等。続いて、メニューRunを選択し、（またはツールバーからrunボタンを選択）表示結果がどのように変更されたかを確認してください。
ここで、カラムSHARESに'newone'にその名前とカラムRETに値50を入力し、view Table：fracのテーブル下部に新しいエントリを追加します。（この結果、カラムfracのセルは空白のままです。）
新しい結果を確認するためモデルに戻ります。
シナリオデータに行った任意の修正・編集はOptimization Modeler内、メニュー*File >>Save*で編集をセーブできます。チャートに関する構成設定の修正は、チャート構成ダイアログで名前を付けたプ

MSI 株式会社 Copy right©2015 All rights reserved

ロファイルとしてセーブできます。このチャートの分割されたメニューオプションとして表示されるようになります。

9.5.2.4 データシナリオを比較する

1つのシナリオを編集、再実行する代わりに、いくつかのシナリオを作成し、複数のシナリオからインプットと結果を表示、比較するためにbuilt-in multi-scenario tableとchart viewsを使用します。
新しいシナリオを作成する、またはプロジェクトから作成します：プロジェクト名を右クリックし、*New Scenario*を選択するか、または他のシナリオから選択する場合：シナリオを右クリックし*Clone*を選択します。

たとえば、オリジナル・シナリオ1のクローン作成を使用し、二つの新しいシナリオを作成してみましょう。最初のコピーで、パラメータMAXVALを0.2に変更し、二回目のコピーで、MAXVALを0.4に設定します。シナリオデータの修正後、各シナリオでRunを選択します。
Repository paneで、プロジェクト名*folioinsight*を選択し、コンテクストメニューから*Open with Multi Scenario Table*を選択します。この結果、各シナリオの目的関数の値を持つテーブルを得ます。この比較にさらにカラムを追加するには、table viewのセルを右クリックし、MS TableにオプションAddを選択します。この比較に意志決定変数fracの全ての解の値の追加を選択し、カラム・ヘッダーでコンテクストメニュー（マウスを右クリック）から*Remove*を選択し、目的関数の値を削除します。ここで、chart viewを生成するために、チャートのアイコン をクリックし、次にChart type *Bar*を選択した*Chart Settings*ダイアログ開くために をクリックし、カスタムタイトルを入力します。（*Use default title*にチェックを入れないでください）その後、下記のスクリーンショットで表示されている図を得るためにボックスのDepth effectをチェックします。

また目的関数の値の他に、パラメータMAXVALの設定（パラメータviewの対応するセルで、MS Tableにコンテクストメニュー Addを選択します。）を表示する二つ目のマルチ・シナリオテーブルを生成し、そして目的関数の値を持つa multi-scenario chartを生成します。（multi-scenario tableのヘッダー’Objective’でコンテクストメニュー*New Multi Scenario Chart*を選択します。）下部で、セクション9.5.2.2で説明したように、個々のシナリオに対するfrac配列のchart viewが開きます。

MSI 株式会社 Copy right©2015 All rights reserved

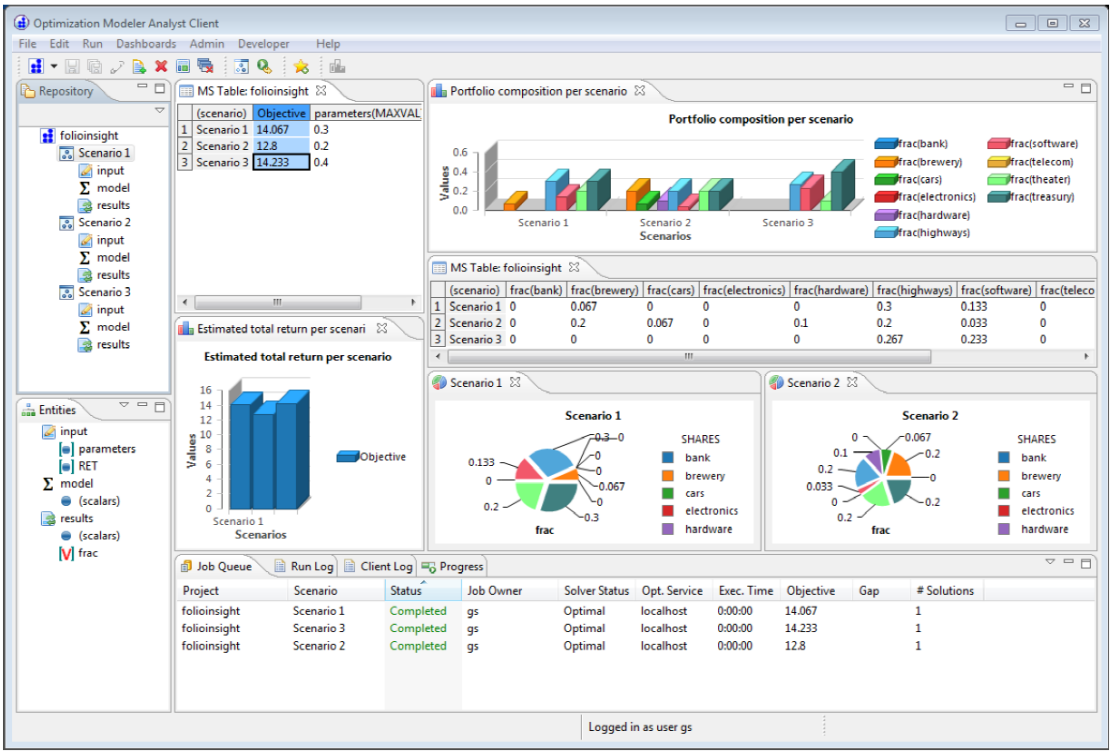


Figure 9.11: Multi-scenario comparison with Optimization Modeler

MSI 株式会社 Copy right©2015 All rights reserved

Ⅱ Getting started with BCL

第 10 章 線形計画問題に入力し、それを解く

この章では、第 2 章で定式化した問題を例として、BCL で、どのようにモデルを実行するかを説明します。具体的には、第 2 章で作ったモデルにいくつかの拡張を加え、それを使って、BCL のインプットとアウトプット機能を説明します。

- BCL で LP モデルを記述する
- インデックス・セットを使い、ファイルからデータをインプットする
- BCL のアウトプット機能
- 問題をマトリックス・ファイルにエクスポートする

第 3 章では、Mosel を使って、同じ例題をどのように定式化し、解くかについて見ました。そして、第 15 章では、Xpress-Optimizer を使って、直接、問題を入力し、解くかを説明します。

10.1 BCL でのインプリメンテーション

この小冊子のすべての BCL の例は、C++により書かれています。C++プログラミング言語では、算術演算子のオーバーローディングができるので、このインタフェースでは、算式の形式に近い形式でモデルを記述でき、非常に便利です。また、同じモデルは、BCL の C、Java、Visual Basic とのインタフェースを使って実行できます。以下の BCL プログラムは、第 2 章で見た LP の問題を実行します。

```
#include <iostream>
#include "xprb_cpp.h"

using namespace std;
using namespace ::dashoptimization;

#define NSHARES 10                    // Number of shares
#define NRISK 5                       // Number of high-risk shares
#define NNA 4                         // Number of North-American shares

double RET[] = {5,17,26,12,8,9,7,6,31,21};  // Estimated return in investment
int RISK[] = {1,2,3,8,9};             // High-risk values among shares
int NA[] = {0,1,2,3};                 // Shares issued in N.-America

int main(int argc, char **argv)
{
 int s;
```


MSI 株式会社 Copy right©2015 All rights reserved

```
 XPRBprob p("FolioLP");              // Initialize a new problem in BCL
 XPRBexpr Risk,Na,Return,Cap;
 XPRBvar frac[NSHARES];              // Fraction of capital used per share

// Create the decision variables
 for(s=0;s<NSHARES;s++) frac[s] = p.newVar("frac");

// Objective: total return
 for(s=0;s<NSHARES;s++) Return += RET[s]*frac[s];
 p.setObj(Return);                   // Set the objective function

// Limit the percentage of high-risk values
 for(s=0;s<NRISK;s++) Risk += frac[RISK[s]];
 p.newCtr("Risk", Risk <= 1.0/3);

// Minimum amount of North-American values
 for(s=0;s<NNA;s++) Na += frac[NA[s]];
 p.newCtr("NA", Na >= 0.5);

// Spend all the capital
 for(s=0;s<NSHARES;s++) Cap += frac[s];
 p.newCtr("Cap", Cap == 1);

// Upper bounds on the investment per share
 for(s=0;s<NSHARES;s++) frac[s].setUB(0.3);


 // Solve the problem
  p.setSense(XPRB_MAXIM);
  p.lpOptimize("");

 // Solution printing
  cout << "Total return: " << p.getObjVal() << endl;
  for(s=0;s<NSHARES;s++)
   cout << s << ": " << frac[s].getSol()*100 << "%" << endl;

  return 0;
 }
```

ここで、上に書かれていることを、詳しく見てみましょう。

10.1.1 初期化

BCL C++のインタフェースを使うためには、ヘッダーファイル xprb_cpp.h が必要です。BCL クラスが属する namespace も定義します。これまでに、ソフトウェアが初期化されていない場合は、下記の行により、最初の問題が生成されるときに、BCL は自動的に初期化されます。

```
XPRBprob p("FolioLP");
```

このステートメントにより、'FolioLP'という名前を持つ新しい問題が生成されます。

10.1.2 構成

モデル自体の定義は、決定変数の生成(メソッド：newVar) から始まり、それに目的関数の定義と制約式の定義が続きます。例に示すように、C++(および、Java) では、制約式は線形式により生成されます。また、制約式は、一項別にも(termwise にも)構成できます。

MSI 株式会社 Copy right©2015 All rights reserved

例えば、ハイリスクの銘柄のパーセンテージを制限している制約式は、以下のようになります。

```
XPRBctr CRisk;
CRisk = p.newCtr("Risk");
for(s=0;s<NRISK;s++) CRisk.addTerm(frac[RISK[s]], 1);
CRisk.setType(XPRB_L);


CRisk.addTerm(1.0/3);
```

この2番目のタイプの制約式の定義は、すべてのBCLインタフェースに共通です。そして、オーバーローディングが使えないC、および、VBでは、この2番目のタイプの制約式による定義が制約式を定義する唯一の方法です。
等式の制約式(ここでの例では、資金をすべて使うことを記述する制約式)を定義する場合には、=(等式記号)を二つ使う(==)必要があることに注意して下さい。メソッドsetUBは、変数fracの上限(upper bound) を設定するために使います。このような別個にファンクションコールする代わりに、変数の生成の際に、直接、バウンズ(bounds)を指定することもできますが、この場合、下に示すように、name、variable type(連続変数にはXPRB_PL) 、boundsの上限、下限を含む、完全な情報が必要です。

```
for(s=0;s<NSHARES;s++) frac[s] = p.newVar("frac", XPRB_PL, 0, 0.3);
```

例題のように、モデリングオブジェクト(決定変数、制約式など)にストリング名(文字列の名前)を付けることはオプションです。もし、ユーザが名前を指定しない場合は、BCL はデフォルト名を生成します。しかし、ユーザ定義の名前を付けると、デバッグや、Optimizer により作成されるアウトプットの解釈が容易になります。

10.1.3 モデルを解く

ソルバーを実行する前に、最適化意志決定変数をsetSenseの呼び出しを使って最大化に設定します。メソッドlpOptimizeは、これまで定義した全制約の対象となるメソッドsetObjで設定した目的関数('Return')を最大化するためにXpress-Optimizerを呼びだします。
lpOptimizeの空文字列の引数はデフォルトのLPアルゴリズムが使われたことを示します。
その他使用できる値はprimalの"p"、dual Simplexの"d"およびNewton-Barrierの"b"です。

10.1.4 アウトプットの印刷

最後の数行のラインにより、最適解と、すべての変数がソリューションで取る値がプリントアウトされます。

10.2 プログラムのコンパイルと実行

標準インストール手順に従って Xpress-Optimizer と BCL をインストールしていれば、Windows環境で、以下のコマンドにより、このファイルをコンパイルできます(注:ここで、フラッグ "/ MD"を使

MSI 株式会社 Copy right©2015 All rights reserved

うことが重要です)。

```
cl /MD /I%XPRESSDIR%\include %XPRESSDIR%\lib\xprb.lib foliolp.cpp
```

Linux や Solaris の場合は、下記を使って下さい。

```
cc -D_REENTRANT -I${XPRESSDIR}/include -L${XPRESSDIR}/lib foliolp.C -o foliolp -lxprb
```

その他のシステムの場合については、corresponding distribution の makefile の例を参照して下さい。こうして得られたプログラムを実行すると、下のようなアウトプットが生成されます。

```
Reading Problem FolioLP
Problem Statistics
           3 (      0 spare) rows
          10 (      0 spare) structural columns

          19 (      0 spare) non-zero elements
Global Statistics
           0 entities        0 sets        0 set members
Maximizing LP FolioLP
Original problem has:
         3 rows           10 cols           19 elements
Presolved problem has:
         3 rows           10 cols           19 elements
    Its         Obj Value      S   Ninf  Nneg        Sum Inf  Time
      0         42.600000      D      2     0        .000000     0
      5         14.066667      D      0     0        .000000     0
Uncrunching matrix
      5         14.066667      D      0     0        .000000     0
Optimal solution found
Problem status: optimal
Total return: 14.0667

0: 30%
1: 0%
2: 20%
3: 0%

4: 6.66667%
5: 30%
6: 0%
7: 0%
8: 13.3333%
9: 0%
```

このディスプレイの上の半分は、Xpress-Optimizer のログです。すなわち、マトリックスのサイズ、3つのロウ(すなわち、制約式)、および、10 のカラム(すなわち、変数)、および、LP のソリューション・アルゴリズム(ここでは、dual Symplex) のログです。下の半分は、プログラムのアウトプットです。すなわち、最大リターンは 14.0667 という値であり、ポートフォリオは銘柄 1、3、5、6、および、9 から成っていることを示しています。このとき、資金全体の 30%は、それぞれ、銘柄 1 と 6 に、20%が銘柄 3 に、そして、13.3333%が銘柄 9 に、6.6667%が銘柄 5 に投下されます。(注意:上のアウトプットでは、行の番号が 0 から始まっている。)すべての制約式が満たされていることは容易に確認できます。すなわち、確かに、北米銘柄へは 50%(銘柄 1 と銘柄 3) の資金が投入され、33.33%がハイリスク銘柄(銘柄 3 と銘柄 9) に投資されるようになっています。最適化の開始の前に、以下のラインを追加して、BCL

MSI 株式会社 Copy right©2015 All rights reserved

と Xpress-Optimizer によるアウトプットの印刷をすべてオフことができます。

```
p.setMsgLevel(1);
```

この設定を行うと、BCLとXpress-Optimizerからのエラーメッセージの印刷は行われないことに注意して下さい。
印刷された0から4のレベル範囲が実行可能値です。第13章では、例えば、メッセージディスプレイの微調整などを行うためのOptimizerコントロールパラメータに、直接、どのようにしてアクセスするかを説明します。

10.3 ファイルからのデータ・インプット

単純に変数に番号を振る代わりに、モデルで、もっと意味のあるインデックスを使うこともできます。例えば、問題データは、ファイル`foliocpplp.dat`などのような文字列インデックスを使うファイルから読み込めます。そして、ファイル`foliocpplp.dat`の内容は、下記のようなものです。

```
! Return
"treasury"  5
"hardware"  17
"theater"  26
"telecom"  12
"brewery"  8
"highways"  9
"cars"  7
"bank"  6
"software"  31

"electronics"  21
```

このデータ・ファイルを使うために、前のモデルを次のように修正します。

MSI 株式会社 Copy right©2015 All rights reserved

```
#include <iostream>
#include "xprb_cpp.h"

using namespace std;
using namespace ::dashoptimization;

#define DATAFILE "foliocpplp.dat"

#define NSHARES 10                    // Number of shares
#define NRISK 5                       // Number of high-risk shares
#define NNA 4                         // Number of North-American shares

double RET[NSHARES];                  // Estimated return in investment
char RISK[][100] = {"hardware", "theater", "telecom", "software",
                    "electronics"};   // High-risk values among shares
char NA[][100] = {"treasury", "hardware", "theater", "telecom"};
                                      // Shares issued in N.-America

XPRBindexSet SHARES;                  // Set of shares

XPRBprob p("FolioLP");                // Initialize a new problem in BCL

void readData(void)
{
 double value;
 int s;
 FILE *datafile;
 char name[100];

 SHARES=p.newIndexSet("Shares",NSHARES);  // Create the 'SHARES' index set

// Read 'RET' data from file
 datafile=fopen(DATAFILE,"r");
 for(s=0;s<NSHARES;s++)
 {
  XPRBreadlinecb(XPRB_FGETS, datafile, 200, "T g", name, &value);
  RET[SHARES+=name]=value;

 }
 fclose(datafile);

 SHARES.print();                      // Print out the set contents
}

int main(int argc, char **argv)
{
 int s;
 XPRBexpr Risk,Na,Return,Cap;
 XPRBvar frac[NSHARES];               // Fraction of capital used per share

// Read data from file
 readData();

// Create the decision variables
 for(s=0;s<NSHARES;s++) frac[s] = p.newVar("frac");

// Objective: total return
 for(s=0;s<NSHARES;s++) Return += RET[s]*frac[s];
 p.setObj(Return);                    // Set the objective function
```


MSI 株式会社 Copy right©2015 All rights reserved

```
// Limit the percentage of high-risk values
 for(s=0;s<NRISK;s++) Risk += frac[SHARES[RISK[s]]];
 p.newCtr("Risk", Risk <= 1.0/3);

// Minimum amount of North-American values

 for(s=0;s<NNA;s++) Na += frac[SHARES[NA[s]]];
 p.newCtr("NA", Na >= 0.5);

// Spend all the capital
 for(s=0;s<NSHARES;s++) Cap += frac[s];
 p.newCtr("Cap", Cap == 1);

// Upper bounds on the investment per share
 for(s=0;s<NSHARES;s++) frac[s].setUB(0.3);

// Solve the problem
 p.setSense(XPRB_MAXIM);
 p.lpOptimize("");

// Solution printing
 cout << "Total return: " << p.getObjVal() << endl;
 for(s=0;s<NSHARES;s++)
  cout << SHARES[s] << ": " << frac[s].getSol()*100 << "%" << endl;

 return 0;
}
```

配列RISKとNAは、文字列のインデックスをストアしており、インデックス集合SHARESという新しいオブジェクトを追加しました。インデックス集合SHARESは、ROI の値を持つRETがデータ・ファイルから読まれる間に定義されます。この例では、インデックス集合を正確なサイズで設定しました。しかし、最初に割り当てられたスペースよりも多い入力が行われると、インデックス集合は、自動的に大きくされますので、最初から正確な大きさで設定する必要はありません。実際のセットサイズはメソッド getSize で得られます。データを読むためには、関数XPRBreadlinecbを使います。これは、データ・ファイルの中にある“！“ を付されたコメント行と何も入っていない空の行を読み飛ばします。フォーマット文字列”T g“ は、実数が後ろに続く（もし、空白が含まれているならば、シングル、または、ダブルクォートにより取り囲んだ）テキスト文字列を読みたいということを意味します。このとき、テキスト文字列と実数は、（タブを含む）スペースにより区切っておかなければなりません。もし、データ・ファイルが‘ , ‘のような別のセパレータ記号を使っているなら、フォーマット文字列は、それに応じて変更して下さい（例えば、”T,g” ）。モデルの中では、線形式RiskとNaの定義は、新しいインデックスに適応したものになります。別の修正部分は、ソリューションの印刷に関するものです。これにより、単なる番号ではなく、すべての銘柄の名前がプリントされ、したがって、ソリューションは、次のように表示されるようになります。

MSI 株式会社 Copy right©2015 All rights reserved

```
Total return: 14.0667
treasury: 30%
hardware: 0%
theater: 20%
telecom: 0%
brewery: 6.66667%
highways: 30%
cars: 0%
bank: 0%
software: 13.3333%
electronics: 0%
```

10.4 アウトプット・ファンクションとエラーの扱い

ほとんどの BCL のモデリングオブジェクト(XPRBprob, XPRBvar, XPRBctr, XPRBsos, XPRBindexSet)は、メソッド print を持っています。変数では、メソッド print がどこで呼び出されるかによりますが、バウンドか解の中での値がプリントされます。例えば、maximization をコールする前に、

```
frac[2].print();
```

という行を入れると、変数名とそのバウンド

```
frac2: [0,0.3]
```

がプリントされます。

それにたいして、問題が解かれた後では、解の中での値

```
frac2: 0.2
```

が表示されます。

BCL がエラーを検出すると、いつでも、エラーメッセージを出して、プログラムの実行を止めます。したがって、通常は、各オペレーションのリターン値をテストする必要はありません。BCL プログラムが、より大きいアプリケーションに埋め込まれているならば、例えば、以下のようにして、explicit initialization を使い、ソフトウェアに正しくアクセスできるかどうかを早くチェックするのがよいでしょう。

```
if(XPRB::init() != 0)
{
  cout << "Initialization failed." << endl;
  return 1;
}
```

メソッドgetLPStatは、LP問題のステータスをテストするのに使います。LP問題が上手く解かれた場合のみ、BCL は、意味のあるソリューションの値を戻して寄越すか、プリントアウトします。

```
char *LPSTATUS[] = {"not loaded", "optimal", "infeasible",
                    "worse than cutoff", "unfinished", "unbounded",
                    "cutoff in dual", "unsolved", "nonconvex"};

cout << "Problem status: " << LPSTATUS[p.getLPStat()] << endl;
```


MSI 株式会社 Copy right©2015 All rights reserved

10.5 マトリックスのエクスポート

Xpress-Optimizer による最適化プロセスが、BCL プログラム（メソッド`lpOptimize`、または`XPRBprob`のメソッド`mipOptimize`）の中で開始されると、問題マトリックスは、メモリーでソルバーにロードされますが、ファイルには書かれません。ファイルに書くことは、時間が掛かり、ランニングタイムが長くなるからです。しかし、場合によってマトリックスを作ることが必要な場合もあるでしょう。Xpress では、2 種類のマトリックス・フォーマットから選択できます。すなわち、拡張 MPS フォーマット、および、拡張 LP フォーマットです。後者は、制約式が数式の形式でプリントされるので、直感的で読み易いものでしょう。MPS フォーマットでマトリックスをエクスポートするには、optimization statement の直前、または optimization statement の代わりに、BCL プログラムへ以下の行を追加します。これにより、作業用ディレクトリに、`Folio.mat`というファイルが作成されます。

```
p.exportProb(XPRB_MPS, "Folio");
```

LP フォーマットのマトリックスの場合は、下記の行を使います。

```
p.setSense(XPRB_MAXIM);
p.exportProb(XPRB_LP, "Folio");
```

LP フォーマットは、最適化が最小化か最大化のどちらかであるかの情報（sense）を含んでいます。デフォルトは、最小化ですので、最大化の場合は、このセンスをリセットする必要があります。結果として生成されるマトリックス・ファイルは、`Folio.lp`という名前を持ちます。

MSI 株式会社 Copy right©2015 All rights reserved

第 11 章 混合整数計画法

この章では、第 10 章のモデルを、混合整数計画法（MIP） 問題に拡張します。ここで、以下のことについて説明します。

・ 離散的な変数のタイプを定義する

・MIP ソリューションのステータスを得て、Xpress-Optimizer で作成される MIP の最適化ログを理解する。

第 6 章では、同じ例を使って、Mosel により、モデルをどのように定式化し、そして解くかを説明しました。また、第 16 章では、Xpress-Optimizer に、直接、問題を入力して、解く方法について説明します。

11.1 拡張問題の説明

投資家は小株の保有を望んでいません。この制約を定式化するために以下の可能性を考慮します。

1. ポートフォリオに含まれる様々な株の数を制限する
2. 株の購入時、少なくとも予算の 10%が株に投資されます。

2 つの別々なモデルの中にあるこの 2 つの制約を扱ってきます。

11.2 MIP モデル 1：様々な株の保有数を制限する

投資している異なる価値（値）の数をカウントするには、第 2 章で作成した LP モデルに変数buy_sの二番目の集合を代入する必要があります。この変数は指標変数かバイナリ変数です。share sがポートフォリオに含まれ、0 以外となる場合、変数buy_sは 1 を取ります。資産の合計を最大$MAXNUM$に制限するために、以下の制約を導入します。変数buy_sの最大$MAXNUM$が、同時に 1 の値を取りうる制約を表現します。

$$\sum_{s \in SHARES} buy_s \leq MAXNUM$$

まだ、新しいバイナリ変数buy_sを変数$frac_s$(ポートフォリオに選択された全ての株の量)と関連付ける必要があります。$frac_s$>0、およびbuy_s=1 の領域である場合、次の関係性を表現します。「株がポートフォリオに選択された場合、値の合計にカウントされます。」または、「$frac_s$>0 ならば、buy_s=1」であることを表現する必要があります。

以下の不等式でこの表現を定式化します。

$$\forall s \in SHARES : frac_s \leq buy_s$$

もし、ある s で、$frac_s$がノンゼロであるならば、buy_sは 0 より大きくなければならず、したがって、buy_sは 1 という値になります。逆に言えば、もし、buy_sが 0 であるなら、そうすると、$frac_s$も 0 であり、銘柄 はポートフォリオに全ったく取り入れられないことを意味します。ここで、これらの制約式がbuy_s、

MSI 株式会社 Copy right©2015 All rights reserved

が 1 という値を取り、かつ、$frac_s$が 0 という値を取る可能性を排除しないことに注意して下さい。しかし、このケースでは、これは問題ありません。なぜならば、解で、buy_sが 1 という値を取り、かつ、$frac_s$が 0 という値をとっても、これらの両方の変数が 0 であるのと同じ効力を持つからです。

11.21 BCL によるインプリメンテーション

ここで、第10章で定式化したLPモデルを、新しい変数と制約式で拡張しましょう。新しい変数がバイナリであるという事実(すなわち、それらの取る値は0と1だけということ)は、変数のタイプを指定するXPRB_BVによって表現します。よく見られる別のタイプの離散的な変数は、整数変数です。整数変数は、所与の上限、下限の間の整数値のみを取る変数です。BCL では、このタイプの変数は、XPRB_UIによって、そのタイプを指定します。次のセクション(MIP2 モデル)では、もう一つのタイプの離散的な変数、すなわち、半連続変数(semi- continuousvariable) の例を見ます。

```
#include <iostream>
#include "xprb_cpp.h"

using namespace std;
using namespace ::dashoptimization;

#define MAXNUM 4                    // Max. number of shares to be selected

#define NSHARES 10                  // Number of shares
#define NRISK 5                     // Number of high-risk shares
#define NNA 4                       // Number of North-American shares

double RET[] = {5,17,26,12,8,9,7,6,31,21};  // Estimated return in investment
int RISK[] = {1,2,3,8,9};           // High-risk values among shares
int NA[] = {0,1,2,3};               // Shares issued in N.-America

int main(int argc, char **argv)
{
 int s;
 XPRBprob p("FolioMIP1");           // Initialize a new problem in BCL
 XPRBexpr Risk,Na,Return,Cap,Num;
 XPRBvar frac[NSHARES];             // Fraction of capital used per share
 XPRBvar buy[NSHARES];              // 1 if asset is in portfolio, 0 otherwise

// Create the decision variables (including upper bounds for 'frac')
 for(s=0;s<NSHARES;s++)
 {
  frac[s] = p.newVar("frac", XPRB_PL, 0, 0.3);
  buy[s] = p.newVar("buy", XPRB_BV);
 }

// Objective: total return
 for(s=0;s<NSHARES;s++) Return += RET[s]*frac[s];
 p.setObj(Return);                  // Set the objective function

// Limit the percentage of high-risk values
 for(s=0;s<NRISK;s++) Risk += frac[RISK[s]];
 p.newCtr(Risk <= 1.0/3);

// Minimum amount of North-American values
 for(s=0;s<NNA;s++) Na += frac[NA[s]];
 p.newCtr(Na >= 0.5);
```


MSI 株式会社 Copy right©2015 All rights reserved

```
// Spend all the capital
 for(s=0;s<NSHARES;s++) Cap += frac[s];
 p.newCtr(Cap == 1);
// Limit the total number of assets

   for(s=0;s<NSHARES;s++) Num += buy[s];
   p.newCtr(Num <= MAXNUM);

  // Linking the variables
   for(s=0;s<NSHARES;s++) p.newCtr(frac[s] <= buy[s]);

  // Solve the problem
   p.setSense(XPRB_MAXIM);
   p.mipOptimize("");

  // Solution printing
   cout << "Total return: " << p.getObjVal() << endl;
   for(s=0;s<NSHARES;s++)
    cout << s << ": " << frac[s].getSol()*100 << "% (" << buy[s].getSol()
         << ")" << endl;

   return 0;
  }
```

追加した変数と制約式に加え、最適化に使うアルゴリズムの選択も、問題のタイプに合わせる必要があります。ここでは、この MIP 問題を分枝限定法によって解きたいと思っています。これは、メソッド mipOptimizeを使います。前章の LP 問題の場合と同じように、MIP ソリューションにアクセスする前に、ソリューション・ステータスをチェックするのはよいことです。MIP の Problem status が、’unfinished (solutionfound)’か‘optimal’の場合のみ、BCL は「意味のあるソリューション」をプリントします。

```
char *MIPSTATUS[] = {"not loaded", "not optimized", "LP optimized",
                     "unfinished (no solution)",
                     "unfinished (solution found)", "infeasible",
                     "optimal", "unbounded"};

cout << "Problem status: " << MIPSTATUS[p.getMIPStat()] << endl;
```

11.2.2 ソリューションを分析する

問題を実行すると、下のようなアウトプットを得ます。

MSI 株式会社 Copy right©2015 All rights reserved

```
Reading Problem FolioMIP1
Problem Statistics
          14 (    514 spare) rows
          20 (      0 spare) structural columns
          49 (   5056 spare) non-zero elements
Global Statistics
          10 entities        0 sets        0 set members
Maximizing MILP FolioMIP1
Original problem has:
        14 rows           20 cols           49 elements        10 globals
Presolved problem has:
        13 rows           19 cols           46 elements         9 globals
LP relaxation tightened


Concurrent solve, 0s
           Dual                        Primal
     objective    suminf        objective    suminf
                           |  p  15.600000  1.4000000
  D  33.933333   .0000000 |  p  15.600000  1.4000000
  D  33.933333   .0000000 |  P  14.066667   .0000000
  ----- interrupted ----- | ------- optimal -------
Deterministic concurrent statistics:
   Dual: 5 simplex iterations, 10802312.490000s
   Primal: 34 simplex iterations, 10821319.870444s
Primal solved problem

   Its         Obj Value      S   Ninf  Nneg        Sum Inf  Time
    34         14.066667      P      0     0        .000000     0

 Optimal solution found

 Starting root cutting & heuristics

  Its Type    BestSoln    BestBound   Sols    Add    Del     Gap     GInf   Time
 +           13.100000    14.066667      1                   7.38% 0
    1  K     13.100000    13.908571      1      2      0    6.17% 2       0
    2  K     13.100000    13.568750      1     11      1    3.58% 3       0
  *** Search completed ***     Time:     0 Nodes:     1
 Number of integer feasible solutions found is 1
 Best integer solution found is    13.100000
 Best bound is    13.100000
 Uncrunching matrix
 Problem status: optimal
 Total return: 13.1
 0: 20% (1)
 1: 0% (0)
 2: 30% (1)
 3: 0% (0)
 4: 20% (1)
 5: 30% (1)
 6: 0% (0)
 7: 0% (0)
 8: 0% (0)
 9: 0% (0)
```

アウトプットの最初の部分には、Xpress-Optimizer の実行ログがあります。問題の変数などの統計が示され、それによると、この問題は、制約式が 14、変数が 20、そのうち、MIP 変数が 10 あることがわかります。ここで、MIP 変数は、’entities’という表現で示されています。さらに、LP アルゴリズ

MSI 株式会社 Copy right©2015 All rights reserved

ムの実行ログ、(タイプ‘k‘［すなわち、knapsack］のカットが合計 13生成されたことを示す)自動カット生成のログ、分枝限定法(Branch-and-Bound) によるサーチのログも示されます。この問題は非常に小さいので、LP 定式化を、LP 緩和の解が整数実行可能解を持つように実行可能領域を厳しくするカットの追加で解けました。ここで、カットとは、MIP ソリューションを狭めないで、LP の実行可能領域のみをカットする制約式です。このカットの追加により、分枝限定法によるサーチ最初のノードで終わりました。このプログラムによりプリントされたアウトプットにより、問題の解が最適解（optimality）、すなわち、MIP サーチは完了したこと、そして、少なくとも、最低 1 個の整数実行可能解が見つかったことがわかります。投資の最大のリターンは、制約式を追加したので、元の LP 問題の最大リターンよりも低くなっています。要件として求められていたように、ポートフォリオは 4 つの異なる銘柄からなっています。

11.3 MIP2：各銘柄の最低投資額の条件を入れる

2 番目の MIP モデルを定式化するために、ここで、再び、第 2 章、および、第 10 章の LP モデルから始めます。定式化したい新しい制約式は、「もし、ある銘柄がポートフォリオの中に組み込まれるならば、少なくとも、その銘柄の保有金額が総投資額に占める割合を投資資本の少なくとも 10%にしたい」ということです。今度は、セット変数 fracs のすべてを、単純に、0 と 0.3 の間の値を取るように制約する代わりに、それらを 0.1 と 0.3 の間の数値を取るか、または、0 という値を取るように制約します。このタイプの変数はセミ連続変数(semi-continuous variable)と呼ばれます。新しいモデルでは、fracs のバウンズを、以下の制約式によって置き換えます。

$$\forall s \in SHARES : frac_s = 0 \text{ or } 0.1 \leq frac_s \leq 0.3$$

11.3.1 BCL によるインプリメンテーション

下のプログラムにより、MIP2 モデルは実行されます。半連続変数は、タイプ`XPRB_SC`により定義されます。BCL は、デフォルトで、連続の限界を 1 に設定します。したがって、ここでは、メソッド`setLim`により、この値を 0.1 に設定しなおすことが必要です。BCL で扱えるもう一つの、類似したタイプの整数変数は、0 という値を取るか、与えられた下限と上限の間の整数値を取る変数です。この変数は、いわゆる、セミ連続整数変数（semi- continuousinteger）と呼ばれています。モデル内で、セミ連続整数変数は、`XPRB_SI`で定義されます。3 番目のタイプは、部分整数変数(partial integer)で、与えられた下限からある限界値までは整数値を取り、この値を越えると連続変数となるものです。これは、`XPRB_PI`で定義されます。

MSI 株式会社 Copy right©2015 All rights reserved

```
#include <iostream>
#include "xprb_cpp.h"

using namespace std;
using namespace ::dashoptimization;

#define NSHARES 10                    // Number of shares
#define NRISK 5                       // Number of high-risk shares
#define NNA 4                         // Number of North-American shares

double RET[] = {5,17,26,12,8,9,7,6,31,21};  // Estimated return in investment
int RISK[] = {1,2,3,8,9};             // High-risk values among shares
int NA[] = {0,1,2,3};                 // Shares issued in N.-America

int main(int argc, char **argv)
{
 int s;
 XPRBprob p("FolioSC");               // Initialize a new problem in BCL
 XPRBexpr Risk,Na,Return,Cap;
 XPRBvar frac[NSHARES];               // Fraction of capital used per share

// Create the decision variables
 for(s=0;s<NSHARES;s++)
 {
   frac[s] = p.newVar("frac", XPRB_SC, 0, 0.3);
   frac[s].setLim(0.1);
  }

 // Objective: total return
  for(s=0;s<NSHARES;s++) Return += RET[s]*frac[s];
  p.setObj(Return);                    // Set the objective functior

 // Limit the percentage of high-risk values
  for(s=0;s<NRISK;s++) Risk += frac[RISK[s]];
  p.newCtr(Risk <= 1.0/3);

 // Minimum amount of North-American values
  for(s=0;s<NNA;s++) Na += frac[NA[s]];
  p.newCtr(Na >= 0.5);

 // Spend all the capital
  for(s=0;s<NSHARES;s++) Cap += frac[s];
  p.newCtr(Cap == 1);

 // Solve the problem
  p.setSense(XPRB_MAXIM);
  p.mipOptimize("");

 // Solution printing
  cout << "Total return: " << p.getObjVal() << endl;
  for(s=0;s<NSHARES;s++)
   cout << s << ": " << frac[s].getSol()*100 << "%" << endl;

  return 0;
 }
```

このプログラムを実行すると、下記のようなソリューションを得ます。

MSI 株式会社 Copy right©2015 All rights reserved

```
Total return: 14.0333
0: 30%
1: 0%
2: 20%
3: 0%
4: 10%
5: 26.6667%
6: 0%
7: 0%
8: 13.3333%
9: 0%
```

この解では、ポートフォリオに 5 つの銘柄が選ばれ、その各々は、総投資の最低で 10%、最大で 30%の間に収まっています。制約式を追加したため、元の LP モデルの解よりも、この MIP 最適解の目的関数の値は、低くなっています。

MSI 株式会社 Copy right©2015 All rights reserved

第 12 章 二次計画法

この章では、第 10 章の LP モデルを二次計画法(Quadratic Programming)のモデルに、そして、第 11 章の最初の MIP モデルを混合整数二次計画法(Mixed Integer QuadraticProgramming)のモデルに拡張します。この章では下記について説明します。

- 二次式の目的関数を定義する
- 問題を少しずつ定義して、問題を解く

第7章では Mosel で同様の例題を使い定式化の方法および解法を説明しています。また第 17 章では OP 問題を直接 Xpress Optimizer を使ってのインプットおよび解法を説明しています。

12.1 問題の説明

この投資家は、もっと、別の角度から、このポートフォリオ選択問題を見ることもできます。すなわち、予想リターンを最大化し、ハイリスクの投資部分を制限するのではなく、あるレベルのリターンを達成しながら、リスクを最小化することです。この投資家は、証券投資の予想リターンの分散/共分散マトリックスの推定値を得る、という Markowitz のアイデアを使おうと思っています。(例えば、ハードウェア銘柄とソフトウェア銘柄の会社の価値は連動する傾向にあるが、人々は、新しいコンピュータやコンピュータゲームで遊ぶことに飽きると、もっと劇場に行くようになるので、演劇プロダクションの成功とは、逆の相関がある、と考えるのが、この考え方の一つの例です。)カナダの財務省証券の利回りは確実なのに対して、劇場プロダクション銘柄のリターンは大幅に変動します。

問題1： 投資家は、ある特定の最小目標利回りを得るために、分散の最小化するにはどの投資戦略を採用すべきでしょうか？

問題2： 投資家が少なくとも、4つの異なる株(ここでも、ある特定の最小利回りを得るために)を選ぶならば、最小の分散投資戦略はどれでしょうか？

最初の問題は、二次計画問題、すなわち二次式の目的関数と一次式の制約を持つ数理計画問題です。二つ目の問題は、株数をカウントするために、離散変数の導入が必要です。したがって混合変数二次計画問題となります。上記 2 つの問題を二つのセクションに分けて、解説していきます。

12.2 QP

第 2 章で作成したモデルを、問題を新しい観点から見ると、下記の変更を行う必要があります。

- 新しい目的関数は、リターンの合計ではなく、平均分散(meanvariance)となる
- リスクに関係する制約式が消える
- 目標利回りという新しい制約式を追加する

MSI 株式会社 Copy right©2015 All rights reserved

新しい目的関数は、下記で表されるポートフォリオの平均分散です。

$$\sum_{s,t \in SHARES} VAR_{st} \cdot frac_s \cdot frac_t$$

ここで、VAR_{st} は、すべての銘柄の分散/共分散マトリックスです。これは、二次の目的関数です。目的関数に、自乗された変数、例えば、$frac_1^2$ や二つの変数が掛け合わされる、例えば、$frac_1 \cdot frac_2$ のような項があると、その目的関数は「二次の目的関数」であると言われます。目標利回りの制約式は、下記のようになります。

$$\sum_{s \in SHARES} RET_s \cdot frac_s \geq TARGET$$

北米銘柄への投資の限度、すべての資金を投資すること、および各銘柄への投資の上限は、このモデルでも適用されます。したがって、このモデルの数学的モデルは下記のようになります。

$$\text{minimize} \sum_{s,t \in SHARES} VAR_{st} \cdot frac_s \cdot frac_t$$

$$\sum_{s \in NA} frac_s \geq 0.5$$

$$\sum_{s \in SHARES} frac_s = 1$$

$$\sum_{s \in SHARES} RET_s \cdot frac_s \geq TARGET$$

$$\forall s \in SHARES : 0 \leq frac_s \leq 0.3$$

12.2.1 BCL で実行する

リターンの推定値、および、分散/共分散マトリックスは、データ・ファイル `foliocppqp.dat` に示されています。

```
! trs  haw  thr  tel  brw  hgw  car  bnk  sof  elc
0.1    0    0    0    0    0    0    0    0    0 ! treasury
  0   19   -2    4    1    1    1  0.5   10    5 ! hardware
  0   -2   28    1    2    1    1    0   -2   -1 ! theater
  0    4    1   22    0    1    2    0    3    4 ! telecom
  0    1    2    0    4 -1.5   -2   -1    1    1 ! brewery
  0    1    1    1 -1.5  3.5    2  0.5    1  1.5 ! highways
  0    1    1    2   -2    2    5  0.5    1  2.5 ! cars
  0  0.5    0    0   -1  0.5  0.5    1  0.5  0.5 ! bank
  0   10   -2    3    1    1    1  0.5   25    8 ! software
  0    5   -1    4    1  1.5  2.5  0.5    8   16 ! electronics
```

このデータファイルを、関数 `XPRBreadarrlinecb` を使って読み込みます。'！' を前に持つコメント行、および、空白行はスキップされます。エントリー 'g ' のフォーマットとセパレータ(スペース、または、タブ表作成のどのような数でも)で、ライン全体を、一気に読みます。目的関数の定義

MSI 株式会社 Copy right©2015 All rights reserved

には、二次式（class XPRBexprで等式を表現)を使います。ここでは、目的関数を最小化したいので、最適化は、メソッドlpOptimizeで始めます（ここでの空の文字列アーギュメントは、デフォルトアルゴリズムを使うことを意味します）。

```
#include <iostream>
#include "xprb_cpp.h"

using namespace std;
using namespace ::dashoptimization;

#define DATAFILE "foliocppqp.dat"

#define TARGET 9                      // Target yield
#define MAXNUM 4                      // Max. number of different assets

#define NSHARES 10                    // Number of shares
#define NNA 4                         // Number of North-American shares

double RET[] = {5,17,26,12,8,9,7,6,31,21};  // Estimated return in investment
int NA[] = {0,1,2,3};                 // Shares issued in N.-America
double VAR[NSHARES][NSHARES];         // Variance/covariance matrix of
                                      // estimated returns

int main(int argc, char **argv)
{
 int s,t;
 XPRBprob p("FolioQP");               // Initialize a new problem in BCL
 XPRBexpr Na,Return,Cap,Num,Variance;
 XPRBvar frac[NSHARES];               // Fraction of capital used per share
 FILE *datafile;

// Read 'VAR' data from file
 datafile=fopen(DATAFILE,"r");
 for(s=0;s<NSHARES;s++)
  XPRBreadarrlinecb(XPRB_FGETS, datafile, 200, "g ", VAR[s], NSHARES);
 fclose(datafile);

// Create the decision variables
 for(s=0;s<NSHARES;s++)
  frac[s] = p.newVar(XPRBnewname("frac(%d)",s+1), XPRB_PL, 0, 0.3);

// Objective: mean variance
 for(s=0;s<NSHARES;s++)
  for(t=0;t<NSHARES;t++) Variance += VAR[s][t]*frac[s]*frac[t];
 p.setObj(Variance);                  // Set the objective function

// Minimum amount of North-American values
 for(s=0;s<NNA;s++) Na += frac[NA[s]];
 p.newCtr(Na >= 0.5);

// Spend all the capital
 for(s=0;s<NSHARES;s++) Cap += frac[s];
 p.newCtr(Cap == 1);

// Target yield
 for(s=0;s<NSHARES;s++) Return += RET[s]*frac[s];
 p.newCtr(Return >= TARGET);
```


MSI 株式会社 Copy right©2015 All rights reserved

```
// Solve the problem
 p.lpOptimize("");


// Solution printing
 cout << "With a target of " << TARGET << " minimum variance is " <<
         p.getObjVal() << endl;
 for(s=0;s<NSHARES;s++)
  cout << s << ": " << frac[s].getSol()*100 << "%" << endl;

 return 0;
}
```

このプログラムにより、以下のソリューションアウトプットが得られます。ここで、QP 問題を解くためのデフォルトアルゴリズムは、これまでの例で使われていた Simplex 法ではなく Newton-barrier を使用します。)

```
  Reading Problem FolioQP
  Problem Statistics
             3 (      0 spare) rows
            10 (      0 spare) structural columns
            24 (      0 spare) non-zero elements
            76 quadratic elements
  Global Statistics
             0 entities        0 sets        0 set members
  Minimizing QP FolioQP
  Original problem has:
           3 rows           10 cols           24 elements
          76 qobjelem
  Presolved problem has:
           3 rows           10 cols           24 elements
          76 qobjelem
  Checking convexity took 0 seconds
  Barrier cache sizes : L1=32K L2=4096K
  Cores per CPU (CORESPERCPU): 2
  Barrier starts, using up to 2 threads, with L2 cache 2048K per thread
  Matrix ordering - Dense cols.:      9   NZ(L):       92   Flops:          584

   Its   P.inf      D.inf      U.inf      Primal obj.     Dual obj.      Compl.
    0   1.90e+01   1.00e+03   3.70e+00   8.7840000e+02  -3.8784000e+03   4.4e+04
    1   2.70e-01   2.50e+00   5.26e-02   8.2966236e+00  -2.6938426e+03   3.2e+03
    2   7.35e-04   2.93e-03   1.43e-04   5.2443717e+00  -9.5768733e+01   1.0e+02
    3   3.31e-05   1.32e-04   6.45e-06   4.1146720e+00  -1.0420241e+01   1.5e+01
    4   6.94e-17   3.33e-15   2.78e-17   1.6111574e+00  -2.5214717e+00   4.1e+00
    5   3.47e-17   1.33e-15   5.55e-17   7.2642096e-01   3.1286182e-01   4.1e-01
    6   9.89e-17   1.62e-15   5.55e-17   5.6822910e-01   5.2583251e-01   4.2e-02
    7   1.46e-16   5.71e-16   1.39e-17   5.5899605e-01   5.5588942e-01   3.1e-03
    8   1.42e-16   9.97e-16   1.59e-17   5.5758472e-01   5.5727257e-01   3.1e-04
    9   3.16e-17   7.81e-16   2.78e-17   5.5740972e-01   5.5738488e-01   2.5e-05
   10   5.38e-17   7.86e-16   2.78e-17   5.5739376e-01   5.5739329e-01   4.7e-07
 Barrier method finished in 0 seconds
 Uncrunching matrix

    Its          Obj Value      S   Ninf  Nneg        Sum Inf  Time
      0            .557394      B      0     0        .000000     0
Optimal solution found
With a target of 9 minimum variance is 0.557394
```


MSI 株式会社 Copy right©2015 All rights reserved

```
0: 30%
1: 7.15401%
2: 7.38237%

3: 5.46362%
4: 12.6561%
5: 5.91283%
6: 0.333491%
7: 29.9979%
8: 1.0997%
9: 6.97039e-06%
```

12.3 MIQP

ここで、前の QP モデルのすべての制約式の下で、最大でも、$MAXNUM$という数の銘柄しかポートフォリオに入れられないという条件をモデルに入れましょう。すでに、第 6 章で、どのようにこれを行えばよいかを見ました。すなわち、バイナリ変数のセットbuy_sを追加して、これにより、下記のように連続変数に論理的にリンクすることでした。

$$\forall s \in SHARES : frac_s \leq buy_s$$

この関係により、もし、$frac_s$>0 ならば、すなわち、$frac_s$がポートフォリオに選ばれるならば、変数buy_sは 1 になり、また、buy_sが 0 であるならば、$frac_s$は 0 でなければならないということになります。

$$\sum_{s \in SHARES} buy_s \leq MAXNUM$$

12.3.1 BCL によるインプリメンテーション

前の QP モデルを修正する、すなわち、以下の行を、前節の QP モデルの終わり(ソリューションのプリンティングの後ろ)に追加します。その後、この問題を、1 回のランで、まず、QP モデルとして解くために下記の行を追加します。

```
 XPRBvar buy[NSHARES];              // 1 if asset is in portfolio, 0 otherwise

// Create the decision variables
 for(s=0;s<NSHARES;s++)
  buy[s] = p.newVar(XPRBnewname("buy(%d)",s+1), XPRB_BV);

// Limit the total number of assets
 for(s=0;s<NSHARES;s++) Num += buy[s];
 p.newCtr(Num <= MAXNUM);

// Linking the variables
 for(s=0;s<NSHARES;s++) p.newCtr(frac[s] <= buy[s]);

// Solve the problem
 p.mipOptimize("");
```


MSI 株式会社 Copy right©2015 All rights reserved

```
// Solution printing
 cout << "With a target of " << TARGET << " and at most " << MAXNUM <<
         " assets, minimum variance is " << p.getObjVal() << endl;

for(s=0;s<NSHARES;s++)
 cout << s << ": " << frac[s].getSol()*100 << "% (" << buy[s].getSol()
      << ")" << endl;
```

MIQP モデルを実行すると、下記のようなソリューション・アウトプットを得ます。

```
  Reading Problem FolioQP
  Problem Statistics
            14 (    514 spare) rows
            20 (      0 spare) structural columns
            54 (   5056 spare) non-zero elements
            76 quadratic elements
  Global Statistics
            10 entities        0 sets        0 set members
  Minimizing MIQP FolioQP
  Original problem has:
          14 rows           20 cols           54 elements        10 globals
          76 qobjelem
  Presolved problem has:
          14 rows           20 cols           54 elements        10 globals
          76 qobjelem
  LP relaxation tightened
  Checking convexity took 0 seconds
  Crash basis containing 0 structural columns created

     Its         Obj Value      S   Ninf  Nneg        Sum Inf  Time
       0           .000000      p      1     0        .100000     0
       8           .000000      p      0     0        .000000     0
       8          4.609000      p      0     0        .000000     0

     Its         Obj Value      S   Nsft  Nneg       Dual Inf  Time
      27           .557393     QP      0     0        .000000     0
  QP solution found
  Optimal solution found

  Starting root cutting & heuristics
```


MSI 株式会社 Copy right©2015 All rights reserved

```
  Its Type    BestSoln    BestBound   Sols    Add    Del     Gap     GInf   Time
 +            4.094716      .557393      1                 86.39% 0
 +            1.839007      .557393      2                 69.69% 0
 +            1.825618      .557393      3                 69.47% 0
 +            1.419003      .557393      4                 60.72% 0
    1  K      1.419003      .557393      4      3      0   60.72% 7      0
    2  K      1.419003      .557393      4      9      2   60.72% 7      0
    3  K      1.419003      .557393      4      7      6   60.72% 7      0
    4  K      1.419003      .557393      4      3      6   60.72% 7      0
    5  K      1.419003      .557393      4      7      2   60.72% 7      0
    6  K      1.419003      .557676      4     18     10   60.70% 7      0
    7  K      1.419003      .561133      4      6     16   60.46% 7      0
    8  K      1.419003      .566040      4      7      4   60.11% 7      0
    9  K      1.419003      .582382      4     19      7   58.96% 8      0
   10  K      1.419003      .587384      4      4     17   58.61% 9      0
   11  K      1.419003      .590887      4      7      2   58.36% 8      0
   12  K      1.419003      .595128      4      6      4   58.06% 9      0
   13  K      1.419003      .598643      4      9      4   57.81% 8      0
   14  K      1.419003      .598924      4      2     12   57.79% 7      0
   15  K      1.419003      .599261      4      5      1   57.77% 7      0
   16  K      1.419003      .600166      4      3      4   57.71% 7      0
   17  K      1.419003      .600166      4      0      2   57.71% 7      0


Heuristic search started
Heuristic search stopped

Cuts in the matrix         : 16
Cut elements in the matrix : 132
Will try to keep branch and bound tree memory usage below 1.6Gb

Starting tree search with up to 2 threads (deterministic mode)

    Node     BestSoln    BestBound   Sols Active  Depth     Gap     GInf   Time
+     24     1.248762     1.045795      5      3      7   16.25% 0
 *** Search completed ***     Time:     0 Nodes:    29
Number of integer feasible solutions found is 5
Best integer solution found is     1.248762
Uncrunching matrix
With a target of 9 and at most 4 assets, minimum variance is 1.24876
0: 30% (1)
1: 20% (1)
2: 0% (0)
3: 0% (0)
4: 23.8095% (1)
5: 26.1905% (1)
6: 0% (0)
7: 0% (0)
8: 0% (0)
9: 0% (0)
```

Branch-andBound サーチのログを見ると、今回は、2 つの整数実行可能解が見つかり、サーチを完了するのに、合計 55 のノードを調べたことがわかります。銘柄数について追加した制約式のため、最小分散の値は、QP 問題のそれと比較して、2 倍以上になっています。

MSI 株式会社 Copy right©2015 All rights reserved

第 13 章 ヒューリスティックス

この章では、Xpress-Optimizer と対話して下記の事柄を行いながら、バイナリ変数を固定して、ヒューリスティックな方法で解を得ることについて説明します。

- パラメータの設定
- ベイシス(基底)をセーブし、ベイシスを戻す、
- 変数のバウンド(bound)を修正する

第 8 章では、Mosel を使って、同じヒューリスティックをどのように実行するかについて説明しました。

13.1 バイナリ変数を固定して行うヒューリスティック

ここで行おうとするヒューリスティックは、下記のステップが必要です。

1. LP 緩和を解き、最適解のベイシスをセーブする。
2. 丸めヒューリスティックス(rounding heuristic):もし、対応する変数 $frac$ の値が、比較的に大きければ、バイナリ変数'buy'を 1 に固定する。
3. こうして得られた MIP 問題を解く。
4. ここで、整数実行可能解が得られたら、その中の最良解をセーブする。
5. すべての変数のバウンドをもともとのバウンドにリセットして、もともとの問題をコンピュータにリストアして、セーブしておいたベイシスをロードする。
6. もともとの MIP 問題を解くが、このとき、ヒューリスティック・ソリューションの最良解の値を「カットオフ値」として使う。

上のステップ 2

割合を示す変数 frac は 0.3 という上限を持つので、「比較的大きな値」として、例えば、0.2 使います。普通、一般に、個々のアプリケーションでは、バイナリ変数にたいしては、$1-\varepsilon$を使います。ここで ε は、10^{-5}のような非常に小さい値です。

ステップ 6

「カットオフ値」を設定する意味は、解の探索を行うときに、この値よりも良い値を持つソリューションを見つけようとすればよく、したがって、あるノードの LP 緩和の解が、この値よりも悪ければ、そのノードは切れます。なぜなら、このノード、および、その先に繋がる子孫のノードの整数実行可能解は、このノードの LP 緩和解よりも悪くなるからです。

13.2 BCL で実行する

ここで、第 11 章の MIP1 モデルを使い、バイナリ変数を固定して、ヒューリスティックな方法で解を得

MSI 株式会社 Copy right©2015 All rights reserved

る方法を実行します。ファンクションという形式で定義されたヒューリスティックにより、モデル自体には、最小限の変更を行います。すなわち、標準のコールでメソッドmipOptimizeを呼び出して問題を解く前に、このヒューリスティック・ソリューションを実行し、ソリューションのプリンティングも、この変化に対応して変更されています。

```
#include <iostream>
#include "xprb_cpp.h"
#include "xprs.h"

using namespace std;
using namespace ::dashoptimization;

#define MAXNUM 4                    // Max. number of shares to be selected

#define NSHARES 10                  // Number of shares
#define NRISK 5                     // Number of high-risk shares
#define NNA 4                       // Number of North-American shares

void solveHeur();

double RET[] = {5,17,26,12,8,9,7,6,31,21};  // Estimated return in investment
int RISK[] = {1,2,3,8,9};           // High-risk values among shares
int NA[] = {0,1,2,3};               // Shares issued in N.-America

XPRBprob p("FolioMIPHeur");         // Initialize a new problem in BCL
XPRBvar frac[NSHARES];              // Fraction of capital used per share
XPRBvar buy[NSHARES];               // 1 if asset is in portfolio, 0 otherwise

int main(int argc, char **argv)
{
 int s;
 XPRBexpr Risk,Na,Return,Cap,Num;

// Create the decision variables (including upper bounds for 'frac')
 for(s=0;s<NSHARES;s++)
 {
  frac[s] = p.newVar("frac", XPRB_PL, 0, 0.3);
  buy[s] = p.newVar("buy", XPRB_BV);
 }

// Objective: total return
 for(s=0;s<NSHARES;s++) Return += RET[s]*frac[s];
 p.setObj(Return);                  // Set the objective function

// Limit the percentage of high-risk values
 for(s=0;s<NRISK;s++) Risk += frac[RISK[s]];
 p.newCtr(Risk <= 1.0/3);

// Minimum amount of North-American values
 for(s=0;s<NNA;s++) Na += frac[NA[s]];
 p.newCtr(Na >= 0.5);

// Spend all the capital
 for(s=0;s<NSHARES;s++) Cap += frac[s];
 p.newCtr(Cap == 1);

// Limit the total number of assets
 for(s=0;s<NSHARES;s++) Num += buy[s];
 p.newCtr(Num <= MAXNUM);
```


MSI 株式会社 Copy right©2015 All rights reserved

```
// Linking the variables
 for(s=0;s<NSHARES;s++) p.newCtr(frac[s] <= buy[s]);

// Solve problem heuristically
 p.setSense(XPRB_MAXIM);


 solveHeur();

// Solve the problem
 p.mipOptimize("");

// Solution printing
 if(p.getMIPStat()==4 || p.getMIPStat()==6)
 {
  cout << "Exact solution: Total return: " << p.getObjVal() << endl;
  for(s=0;s<NSHARES;s++)
   cout << s << ": " << frac[s].getSol()*100 << "%" << endl;
 }
 else
  cout << "Heuristic solution is optimal." << endl;

 return 0;
}

void solveHeur()
{
 XPRBbasis basis;
 int s, ifgsol;
 double solval, bsol[NSHARES],TOL;

 XPRSsetintcontrol(p.getXPRSprob(), XPRS_CUTSTRATEGY, 0);
                                   // Disable automatic cuts
 XPRSsetintcontrol(p.getXPRSprob(), XPRS_HEURSTRATEGY, 0);
                                   // Disable MIP heuristics
 XPRSsetintcontrol(p.getXPRSprob(), XPRS_PRESOLVE, 0);
                                   // Switch presolve off
 XPRSgetdblcontrol(p.getXPRSprob(), XPRS_FEASTOL, &TOL);
                                   // Get feasibility tolerance

 p.mipOptimize("l");               // Solve the LP-relaxation
 basis=p.saveBasis();              // Save the current basis

// Fix all variables 'buy' for which 'frac' is at 0 or at a relatively
// large value
 for(s=0;s<NSHARES;s++)
 {

bsol[s]=buy[s].getSol();         // Get the solution values of 'frac'
if(bsol[s] < TOL)  buy[s].setUB(0);
else if(bsol[s] > 0.2-TOL)  buy[s].setLB(1);
}

p.mipOptimize("c");               // Solve the MIP-problem
ifgsol=0;
if(p.getMIPStat()==4 || p.getMIPStat()==6)
{                                 // If an integer feas. solution was found
```


MSI 株式会社 Copy right©2015 All rights reserved

```
  ifgsol=1;
  solval=p.getObjVal();             // Get the value of the best solution
  cout << "Heuristic solution: Total return: " << p.getObjVal() << endl;
  for(s=0;s<NSHARES;s++)
   cout << s << ": " << frac[s].getSol()*100 << "%" << endl;
 }

// Reset variables to their original bounds
 for(s=0;s<NSHARES;s++)
  if((bsol[s] < TOL) || (bsol[s] > 0.2-TOL))
  {
   buy[s].setLB(0);
   buy[s].setUB(1);
  }

 p.loadBasis(basis);         /* Load the saved basis: bound changes are
                                immediately passed on from BCL to the
                                Optimizer if the problem has not been modified
                                in any other way, so that there is no need to
                                reload the matrix */

   basis.reset();              // No need to store the saved basis any longer
   if(ifgsol==1)
    XPRSsetdblcontrol(p.getXPRSprob(), XPRS_MIPABSCUTOFF, solval+TOL);
                               // Set the cutoff to the best known solution
  }
```

このヒューリスティックのインプリメンテーションについては、確かに、ある程度の説明が必要です。この例では、初めて、Xpress-Optimizer へのダイレクトアクセスを使います。Xpress-Optimizer へのダイレクトアクセスを行うには、Optimizer ヘッダーファイル `xprs.h` を含む必要があります。Optimizer 機能は、Optimizer が保有している（タイプ `XPRSprob` の）問題表現（problem representation）に適用できます。そして、問題表現は、BCL 問題のメソッド `getXPRSprob` によってリトリーブできます。BCL ライブラリと Optimizer ライブラリの結合をどのように使うかについてのもっと詳しい説明は、「BCL リファレンスマニュアル」を参照して下さい。すべての Optimizer 機能とパラメータについての完全なドキュメントは、「Optimizer リファレンスマニュアル」にあります。

パラメータ

ヒューリスティックを解くことは、Xpress-Optimizer のパラメータ設定から始まります。自動的なカットの生成（パラメータ `XPRS_CUTSTRATEGY`）のスイッチ・オフと MIP ヒューリスティックス（パラメータ `XPRS_HEURSTRATEGY` オプションですが、ここでの問題を解く場合には、presolve のメカニズム（パラメータ `XPRS_PRESOLVE`）でセットして行うマトリックスに加える前処理で、モデルサイズを小さくし、ニューメリカルな特性を改善する）を止めておくことが必要です。なぜなら、解く過程で Optimizer の中の問題とインタラクトしますが、Optimizer によりマトリックスが変更されていない場合のみ、インタラクションが正しく行えるからです。

パラメータの設定に加え、Xpress-Optimizer が使うフィージビリティ・トレランスも調べる必要がある場合もあります。Xpress-Optimizer は、整数のフィージビリティ、ソリューション・フィージビリティのチ

MSI 株式会社 Copy right©2015 All rights reserved

ェックに、デフォルトで10^{-6}のオーダーのトレランス値を使います。例えば、パーフォーマンスの比較をして解を評価するときには、トレランスを考慮することが重要です。第 10 章で説明したアウトプットプリンティングの微調整は、パラメータ XPRS_LPLOGとXPRS_MIPLOGを設定することによって行えます。
（両方とも、関数XPRSsetintcontrolで設定する）。

最適化のコール

先端のノードのLP緩和のみを解くことを示している（全体がMIP問題ではない）アーギュメント"l"を付けたメソッド（mipOptimize）を使います。一度、停止させたアルゴリズムの部分からMIPで解くには、アーギュメント"c"を使います。

セーブ、および、ベイシスのロード

ソリューションプロセスを加速するため、最初の LP 緩和を解いた後に、問題に何の変更も加える前に、シンプレックス・アルゴリズムのベイシスを（メモリー内に）セーブします。このベイシスは、オリジナルな問題をリストアしたら、最後に、再びロードされます。こうして、MIP のソリューション・アルゴリズムは、LP 問題を、「最初から」から解かなくてもよくなります。こうして、MIP のソリューション・アルゴリズムは、ヒューリスティックにより「中断された状態」から、計算プロセスを続ければよいことになります。

バウンドの変更

すでに、問題が、Optimizer にロードされているとき（例えば、optimization statement を実行したり、または、loadMatを明確に呼び出した後）、setLB とsetUBを経由して、バウンドの変更は、直接、Optimizer に渡されます。しかし、制約式や変数の追加や削除のようなモデルへの変更の場合は、問題を再ロードする必要があります。このプログラムは、以下のアウトプットを生成します。オリジナル問題を 2 回目に解くときには、Simplex アルゴリズムは、一回もイタレーションを行いません。なぜなら、このとき、前回の計算で得た最適解のベーシス（基底）を使っているからです。

```
Reading Problem FolioMIPHeur
Problem Statistics
          14 (      0 spare) rows
          20 (      0 spare) structural columns
          49 (      0 spare) non-zero elements
Global Statistics
          10 entities        0 sets        0 set members
Maximizing MILP FolioMIPHeur
Original problem has:
        14 rows           20 cols           49 elements        10 globals
```


MSI 株式会社 Copy right©2015 All rights reserved

```
Concurrent solve, 0s
           Dual                          Primal
     objective    suminf         objective    suminf
                           |  p  14.400000   .3666667
                           |  p  14.066667   .0000000
  ----- interrupted ----- | ------- optimal -------
                 -                          -
 Deterministic concurrent statistics:
    Dual: 5 simplex iterations, 10802312.490000s
    Primal: 17 simplex iterations, 10821319.870444s
 Primal solved problem

    Its         Obj Value      S   Ninf  Nneg        Sum Inf  Time
     17         14.066667      P      0     0        .000000     0
 Optimal solution found
  *** Search unfinished ***    Time: 0
 Number of integer feasible solutions found is 0
 Best bound is    14.066667

 Starting root cutting & heuristics

  Its Type    BestSoln    BestBound   Sols    Add    Del     Gap     GInf   Time
 +           13.100000    14.066667      1                  7.38% 0
 Heuristic search started
 Heuristic search stopped
  *** Search completed ***     Time:     0 Nodes:     1
 Number of integer feasible solutions found is 1
 Best integer solution found is    13.100000
 Best bound is    13.100000
 Heuristic solution: Total return: 13.1
 0: 20%
 1: 0%
 2: 30%
 3: 0%
 4: 20%
 5: 30%
 6: 0%
 7: 0%
 8: 0%
 9: 0%
 Maximizing MILP FolioMIPHeur
 Original problem has:
         14 rows           20 cols           49 elements        10 globals
```


MSI 株式会社 Copy right©2015 All rights reserved

```
Concurrent solve, 0s
           Dual                        Primal
     objective    suminf        objective    suminf
  D  14.066667   .0000000
  D  14.066667   .0000000 |  p  14.066667   .0000000
  ------- optimal ------- | ----- interrupted -----
Deterministic concurrent statistics:
   Dual: 0 simplex iterations, 10802312.490000s
   Primal: 0 simplex iterations, 10821319.870444s
Dual solved problem

   Its         Obj Value      S   Ninf  Nneg        Sum Inf  Time
     0         14.066667      P      0     0        .000000     0
Optimal solution found

Starting root cutting & heuristics

 Its Type    BestSoln    BestBound   Sols    Add    Del     Gap     GInf   Time
Heuristic search started
Heuristic search stopped
Will try to keep branch and bound tree memory usage below 1.6Gb

Starting tree search with up to 2 threads (deterministic mode)

    Node     BestSoln    BestBound   Sols Active  Depth     Gap     GInf   Time
 *** Search completed ***     Time:     0 Nodes:     5

Problem is integer infeasible
Number of integer feasible solutions found is 0
Best bound is    13.100001
Heuristic solution is optimal.
```

このヒューリスティックは 13.1 という解を見つけたこと、ヒューリスティックなしの MIP optimizer が、（実行不可能というメッセージを出し）よりよい解を見つけることができなかったことに注意して下さい。したがって、このヒューリスティック解は最適（optimal） です。

MSI 株式会社 Copy right©2015 All rights reserved

Ⅲ Getting started with the Optimizer

第 14 章 マトリックスのインプット

この章では、以下のことについて説明します。

- Xpress-Optimizer の初期化
- MPS フォーマット、または、LP フォーマットで、マトリックスを Xpress-Optimizer にロードする
- 問題を解く
- ソリューションをファイルに書く

14.1 マトリックス・ファイル

Xpress では、ユーザは、2 つのマトリックス形式、すなわち、拡張 MPS フォーマット、および、拡張 LP フォーマットのどちらかを選択できます。これら二つのフォーマットのうち、後者は、制約式が数式でプリントされるので、通常、人間が容易に読み取れるフォーマットです。マトリックスは、Xpress-Optimizer でも書けますが、それらは、通常は、他のツールで生成されたものでしょう。かりに、Mosel、または、BCL プログラムの中で、Xpress-Optimizer による最適化プロセスが始められているならば、問題のマトリックスは、外のファイルに書かれることなく、メモリーで、ソルバーにロードされます。(外部ファイルに書くことは、ランニング・タイムのロスにつながるため)しかし、Mosel、および、BCL の両方を使って、マトリックスを生成することもできます(Mosel によるマトリックス・ジェネレーションについては第 9 章を、BCL によるマトリックス・ジェネレーションについては第 10 章を参照下さい)。

14.2 インプリメンテーション

Xpress-Optimizer にマトリックスをロードするには、次のステップを取ります。

1. Xpress-Optimizer を初期化する
2. 新しい問題をクリエートする
3. マトリックス・ファイルを読み込む

(同様なインタフェースは Java と Visual Basic にもありますが)以下の C プログラム `folioinput.c`は、どのようにしてマトリックス・ファイルをロードし、問題を解き、解を書き出すかを示したものです。判りやすくするために、ここではこのプログラムでは、すべてのエラーチェックを省略してあります。一般に、初期化機能の「戻り値」をテストし、また 問題が作成され、正しく読み込まれたかどうかをチェックすることをお勧めします。Xpress-Optimizer を使うには、ヘッダーファイル `xprs.h`.を含めることが必要です。

MSI 株式会社 Copy right©2015 All rights reserved

```
#include <stdio.h>
#include "xprs.h"

int main(int argc, char **argv)
{
 XPRSprob prob;

 XPRSinit(NULL);                        /* Initialize Xpress-Optimizer */
 XPRScreateprob(&prob);                 /* Create a new problem */

 XPRSreadprob(prob, "Folio", "");       /* Read the problem matrix */

 XPRSchgobjsense(prob, XPRS_OBJ_MAXIMIZE);   /* Set sense to maximization */
 XPRSlpoptimize(prob, "");              /* Solve the problem */

 XPRSwriteprtsol(prob, "Folio.prt", "");  /* Write results to 'Folio.prt' */

 XPRSdestroyprob(prob);                 /* Delete the problem */
 XPRSfree();                            /* Terminate Xpress */

 return 0;
}
```

14.3 プログラムのコンパイルと実行

Xpress-Optimizer の標準のインストール手順に従うと、Windows の下で、以下のコマンドによってこのファイルをコンパイルします。

```
cl /MD /I%XPRESSDIR%\include %XPRESSDIR%\lib\xprs.lib foliomat.c
```

Linux や Solaris の場合は、以下のコマンドを使います。

```
cc -D_REENTRANT -I${XPRESSDIR}/include -L${XPRESSDIR}/lib foliomat.c -o foliomat -lxprs
```

他のシステムについては、対応するシステムの makefile を参照してください。BCL により作成された第 2 章の例題のマトリックス`Folio.mat`で、このプログラムを実行すると、以下の内容を持つアウトプットファイル`Folio.prt`を得ます。

```
Problem Statistics
Matrix FolioLP
Objective *OBJ*

RHS *RHS*
Problem has      4 rows and     10 structural columns

Solution Statistics
Maximization performed
Optimal solution found after      5 iterations
Objective function value is    14.066659


Rows Section
   Number     Row      At      Value      Slack Value   Dual Value       RHS
 N       1  *OBJ*      BS    14.066659    -14.066659        .000000      .000000
 E       2  Cap        EQ     1.000000       .000000       8.000000    1.000000
 G       3  NA         LL      .500000       .000000      -5.000000      .500000
 L       4  Risk       UL      .333333       .000000      23.000000      .333333
```


MSI 株式会社 Copy right©2015 All rights reserved

```
 Columns Section
    Number    Column    At      Value      Input Cost   Reduced Cost
  C       5   frac      UL       .300000      5.000000      2.000000
  C       6   frac_1    LL       .000000     17.000000     -9.000000

C       7   frac_2    BS       .200000     26.000000       .000000
C       8   frac_3    LL       .000000     12.000000    -14.000000
C       9   frac_4    BS       .066667      8.000000       .000000
C      10   frac_5    UL       .300000      9.000000      1.000000
C      11   frac_6    LL       .000000      7.000000     -1.000000
C      12   frac_7    LL       .000000      6.000000     -2.000000
C      13   frac_8    BS       .133333     31.000000       .000000
C      14   frac_9    LL       .000000     21.000000    -10.000000
```

上の部分は、問題のサイズとソリューション・アルゴリズム関連の統計を示しています。見つかったオプティマム LP ソリューションは、14.066659 の値を持っています。Rows Section では、問題の制約式の詳細なソリューション情報が示され、Columns Section では、Value というカラムのところに、変数の値が示されています。

MSI 株式会社 Copy right©2015 All rights reserved

第 15 章 線形計画法（LP）の問題をインプットし、解く

この章では、第 2 章で定式化した問題を例にとり、Xpress-Optimizer を使って、それをどのように入力し、解くかについて説明します。以下のトピックについて、詳細に解説します。

- LP モデルをマトリックス・フォーマットに変換する
- Xpress-Optimizer での LP 問題インプット
- 問題を解くき、ソリューションをアウトプットする

第 3 章では、Mosel を使い、どのようにこの例題を定式化し、また、第 10 章では、BCL を使った場合が説明されています。

15.1 マトリックス表示

数式的に表現されている問題を、Xpress-Optimizer の LP 問題のインプット機能により求められる形式に変換する最初のステップとして、各列(column)が一つの意思決定変数を表し、各行(row)が制約式を表すようなテーブルという形式で問題を書きます。このテーブルに全てのノンゼロ係数を書き込みます。そうすると問題マトリックスが得られます。それに、各式が右辺とどのような関係にあるかを示す=、$\leq$、$\geq$記号と、右辺の定数を加えてこの表を完成させます（右辺は、通常、RHS(right handside)の値と呼ばれます）。Xpress-Optimizer に示されるマトリックスについての情報は、行数と列数を掛け合わしたマトリックス全体ではなく、ノンゼロ係数のリスト、および、それらの係数のマトリックス内での位置についての情報です。このテーブルの中の上付きの添え字は、マトリックス・エントリーがリストに入力される順序を示しています。係数の値は、配列 `matval`に、また、対応する行番号は、配列`rowidx`, にストアされます。そして、これらの配列の最初のいくつかのエントリは、それらを強調するためにイタリック体でプリントされます（さらに詳しい説明は、「Optimizer Reference Manual」を参照ください）。そして、配列 `colbeg` には、カラムごとの最初のエントリのインデックスが、また、配列 `nelem` には、カラムあたりのエントリの数がストアされ、マトリックスについての情報が完成します。

Table 15.1: LP matrix

		$frac_1$	$frac_2$	$frac_3$	$frac_4$	$frac_5$	$frac_6$	$frac_7$	$frac_8$	$frac_9$	$frac_{10}$		
		0	1	2	3	4	5	6	7	8	9	Oper.	RHS
Risk	0		1^{2}	1^{5}	1^{8}					1^{15}	1^{17}	$\leq$	1/3
MinNA	*1*	1^{0}	1^{3}	1^{6}	1^{9}							$\geq$	0.5
Allfrac	*2*	1^{1}	1^{4}	1^{7}	1^{10}	1^{11}	1^{12}	1^{13}	1^{14}	1^{16}	1^{18}	=	1
	↑ *rowidx*	↑ *matval*											
colbeg		0	2	5	8	11	12	13	14	15	17	19	
nelem		2	3	3	3	1	1	1	1	2	2		

MSI 株式会社 Copy right©2015 All rights reserved

15.2 Xpress-Optimizer によるインプリメンテーション

以下の C プログラム`foliolp.c`は、Xpress-Optimizer を使って、LP 問題をどのように入力し、解くかを示しています。ここでは、ソリューションの印刷も追加してあります。また、ソリューションにアクセスする前に、LP 問題のステータスをチェックするようになっています。
（さらに詳しい説明は「OptimizerReferenceManual」を参照ください）。Xpress-Optimizer を使うには、ヘッダーファイル `xprs.h`が必要です。

1. Xpress-Optimizer を初期化する
2. 新しい問題をクリエートする
3. マトリックス・ファイルを読み込む

```
#include <stdio.h>
#include <stdlib.h>
#include "xprs.h"

int main(int argc, char **argv)
{
 XPRSprob prob;
 int s, status;
 double objval, *sol;

 /* Problem parameters */
 int ncol = 10;
 int nrow = 3;

 /* Row data */
 char rowtype[] = {  'L','G','E'};
 double rhs[]   = {1.0/3,0.5, 1};

 /* Column data */
 double obj[] = {  5, 17, 26, 12,  8,  9,  7,  6, 31, 21};
 double lb[]  = {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0};
 double ub[]  = {0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3};

 /* Matrix coefficient data */
 int colbeg[]    = {0,  2,    5,    8,    11,12,13,14,15,  17,  19};
 int rowidx[]    = {1,2,0,1,2,0,1,2,0,1,2, 2,  2,  2,  2,  0,2,  0,2};
 double matval[] = {1,1,1,1,1,1,1,1,1,1,1, 1,  1,  1,  1,  1,1,  1,1};

 XPRSinit(NULL);                          /* Initialize Xpress-Optmizer */
 XPRScreateprob(&prob);                   /* Create a new problem */

                                          /* Load the problem matrix */
 XPRSloadlp(prob, "FolioLP", ncol, nrow, rowtype, rhs, NULL,
            obj, colbeg, NULL, rowidx, matval, lb, ub);

 XPRSchgobjsense(prob, XPRS_OBJ_MAXIMIZE);  /* Set sense to maximization */
 XPRSlpoptimize(prob, "");                /* Solve the problem */

 XPRSgetintattrib(prob, XPRS_LPSTATUS, &status);  /* Get LP sol. status */
```


MSI 株式会社 Copy right©2015 All rights reserved

```
if(status == XPRS_LP_OPTIMAL)
{
 XPRSgetdblattrib(prob, XPRS_LPOBJVAL, &objval); /* Get objective value */
 printf("Total return: %g\n", objval);


  sol = (double *)malloc(ncol*sizeof(double));
  XPRSgetlpsol(prob, sol, NULL, NULL, NULL);      /* Get primal solution */
  for(s=0;s<ncol;s++) printf("%d: %g%%\n", s+1, sol[s]*100);
 }

 XPRSdestroyprob(prob);                   /* Delete the problem */
 XPRSfree();                              /* Terminate Xpress */

 return 0;
}
```

最後のカラムの後ろに(すなわち、last+1 に)、エントリーを追加して、`colbeg`を定義する代わりに、配列`nelem`で、カラムあたりの係数の数を与えます。

```
    /* Matrix coefficient data */
    int colbeg[]    = {0,  2,    5,    8,    11,12,13,14,15,  17};
    int nelem[]     = {2,  3,    3,    3,     1, 1, 1, 1, 2,   2};
    int rowidx[]    = {1,2,0,1,2,0,1,2,0,1,2, 2, 2, 2, 2, 0,2, 0,2};
    double matval[] = {1,1,1,1,1,1,1,1,1,1,1, 1, 1, 1, 1, 1,1, 1,1};

    ...
                                          /* Load the problem matrix */
    XPRSloadlp(prob, "FolioLP", ncol, nrow, rowtype, rhs, NULL,
               obj, colbeg, nelem, rowidx, matval, lb, ub);
```

最適化関数`XPRSloadlp`の 7 番目のアーギュメントは、NULL になっていますが、これは、この問題では、制約式についての range information の情報のためにとっておかれているからです。
最適化関数`XPRSlpoptimize`の 2 番目のアーギュメントは、使うアルゴリズムを示し、空の文字列はデフォルト LP アルゴリズムを使うことを示します。問題を解いた後に、LP がうまく解かれたかどうかをチェックし、もしそうなら、目的関数の値と意思決定変数が集問題(primal problem)の解で取る値を取り出します。

15.3 コンパイルと問題の実行

Xpress-Optimizer の標準的なインストール手順を行う場合、Windows 上では下記のコマンドを使用してファイルをコンパイルします。

```
cl /MD /I%XPRESSDIR%\include %XPRESSDIR%\lib\xprs.lib foliolp.c
```

Linux または Solaris の場合は、下記のコマンドを使用します。

```
cc -D_REENTRANT -I${XPRESSDIR}/include -L${XPRESSDIR}/lib foliolp.c -o foliolp -lxprs
```


MSI 株式会社 Copy right©2015 All rights reserved

その他のシステム上については、対応する問題で提供されている例題を参照ください。
この問題を実行すると、下記のアウトプットを得ます。

```
Total return: 14.0667
1: 30%
2: 0%
3: 20%
4: 0%
5: 6.66667%
6: 30%
7: 0%
8: 0%
9: 13.3333%
10: 0%
```

UNIX の場合は、Xpress-Optimizer のログが前に付きます。

```
Reading Problem FolioLP
Problem Statistics
            3 (      0 spare) rows
           10 (      0 spare) structural columns
           19 (      0 spare) non-zero elements
Global Statistics
            0 entities        0 sets        0 set members
Maximizing LP FolioLP
Original problem has:
          3 rows           10 cols            19 elements
Presolved problem has:
          3 rows           10 cols            19 elements

    Its         Obj Value      S   Ninf  Nneg        Sum Inf  Time
      0         42.600000      D      2     0        .000000     0
      5         14.066667      D      0     0        .000000     0
Uncrunching matrix
      5         14.066667      D      0     0        .000000     0
Optimal solution found
```

Windows ユーザは、Optimizer のログをファイルに転送してリトリーブします。その場合、プログラムの中で、問題の作成直後に、以下のラインを追加してください。

```
XPRSsetlogfile(prob, "logfile.txt");
```

Optimizer のログは、マトリックスのサイズ、3 つの row(すなわち、制約式)、および、10 の column(意思決定変数)それに使った LP アルゴリズム(ここでは、dual Simplex を意味する'D')のログが示されます。このプログラムにより出力されたアウトプットから、最大のリターンの値が 14.0667 であること、ポートフォリオは銘柄 1、3、5、6、および、9 から成っていることが読めます。また、この解では、資金合計の 30%が、それぞれ、銘柄 1、および、銘柄 6 に、20%が銘柄 3 に、13.3333%が銘柄 9 に、そして、6.6667%が銘柄 5 に使われることが読めます。また、すべての制約式が満たされていることが容易に確認できます。すなわち、50%が北米銘柄(銘柄 1 と銘柄 3) であること、そして、33.33%がハイリスク銘柄(銘柄 3 と銘柄 9) になっています。

MSI 株式会社 Copy right©2015 All rights reserved

第 16 章 混合整数計画法

この章では、第 2 章の LP 問題を混合整数計画法（MIP） 問題に拡張します。ここで、以下のことについて説明します。

- MIP モデルをマトリックス形式に変換する
- Xpress-Optimizer に、いろいろな離散的な変数を持つ MIP 問題を入力する
- MIP 問題を解き、ソリューションをアウトプットする

16.1 問題の拡張についての説明

投資家は、ある銘柄を持つとしたら、ある程度の資金をその銘柄に投資したいと思っています。彼は、この制約式を定式化するのに、以下の 2 つのことを考えています。

1. ポートフォリオに組み込む銘柄数を 4 個に制限する。
2. もし、ある銘柄が買われるならば、少なくとも、予算の 10%をその銘柄に投資したい。

以下では、これらの 2 点を、2 つの別個のモデルで扱います。

16.2 MIP1： ポートフォリオに組み込む銘柄数を制限する

投資する銘柄数をカウントできるように、第 2 章で開発した LP モデルに、2 つ目のセット変数 buy_s を導入します。これらの変数は、インディケータ変数、すなわち、バイナリ変数です。変数 buy_s は、銘柄 s がポートフォリオに組み込まれると 1 という値を取り、そうでない場合は、0 という値を取ります。
ポートフォリオに組み込む銘柄数を MAXNUM というに最大値に制限するために、以下の制約式を導入します。この制約式は、同時には、最大でも、4 個の変数 buy_s しか 1 という値を取ることができないということを表現する制約式です。

$$\sum_{s \in SHARES} buy_s \leq 4$$

ここで、ポートフォリオに選ばれたすべての銘柄の量を示す変数 $frac_s$ を、新しいバイナリ変数 buy_s に結びつける必要があります。表現したい関係は、「もし、ある銘柄がポートフォリオに選ばれるならば、それは、銘柄数の一つとして、全体の中でカウントされる」、換言すると、$frac_s > 0$ であるならば、$buy_s = 1$ ということです。以下の不等式により、このことを定式化します。

$$\forall s \in SHARES : frac_s \leq buy_s$$

もし、ある s で、$frac_s$ がノンゼロであるならば、buy_s は 0 より大きくなければならず、したがって、buy_s は 1 という値になります。逆に言えば、もし、buy_s が 0 であるなら、そうすると、$frac_s$ も 0 であり、銘柄 はポートフォリオに全ったく取り入れられないことを意味します。ここで、これらの制約式が、

MSI 株式会社 Copy right©2015 All rights reserved

buy_sが 1 という値を取り、かつ、$frac_s$が 0 という値を取る可能性を排除しないことに注意して下さい。しかし、このケースでは、これは問題ありません。なぜならば、解で、buy_sが 1 という値を取り、かつ、$frac_s$が 0 という値をとっても、これらの両方の変数が 0 であるのと同じ効力を持つからです。

16.2.1 マトリックスによる表現

数式で表現されたモデルは、次のようなテーブルに変換できます。前章の LP モデルのマトリックスに較べ、このマトリックスは、変数buy_sのための新しいカラムが 10 個追加され、二つのタイプの変数を結びつけるための制約式が 10 本追加されています。ここで、これらの制約式の変数を含むすべての項が等号、不等号の左辺に来るように変形されていることに気をつけて下さい。

マトリックス係数の上付き文字は、ここでも、配列 rowidxと配列 matval への入力の順序を表し、最初の 3 つのエントリは、強調のため、イタリック体でプリントされています。

Table 16.1: MIP matrix

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	Op.	RHS
Risk	**0**		1^{3}	1^{7}	1^{11}					1^{23}	1^{26}										≤	1/3
MinNA	***1***	1^{0}	1^{4}	1^{8}	1^{12}																≥	0.5
Allfrac	***2***	1^{1}	1^{5}	1^{9}	1^{13}	1^{15}	1^{17}	1^{19}	1^{21}	1^{24}	1^{27}										=	1
Maxnum	**3**										1^{29}	1^{31}	1^{33}	1^{35}	1^{37}	1^{39}	1^{41}	1^{43}	1^{45}	1^{47}	≤	4
Linking	***4***	1^{2}									-1^{30}										≤	0
	5		1^{6}									-1^{32}									≤	0
	6			1^{10}									-1^{34}								≤	0
	7				1^{14}									-1^{36}							≤	0
	8					1^{16}									-1^{38}						≤	0
	9						1^{18}									-1^{40}					≤	0
	10							1^{20}									-1^{42}				≤	0
	11								1^{22}									-1^{44}			≤	0
	12									1^{25}									-1^{46}		≤	0
	13										1^{28}									-1^{48}	≤	0
	↑	↑																				
rowidx	*matval*																					
colbeg		0	3	7	11	15	17	19	21	23	26	29	31	33	35	37	39	41	43	45	47	49	

16.2.2 Xpress-Optimizer で実行する

LP 問題と共通しているマトリックス係数と関連する構造だけではなく、MIP 問題固有情報、すなわち、MIP 変数のタイプ（ここでは、すべて、バイナリ変数であることを示す'B ‘ がマークされています）を配列miptypeで、また、配列 mipcolの対応したカラムインデックスで指定する必要があります。普通にみられる別のタイプの離散的な変数は、整数変数、すなわち、与えられた下限と上限の間の整数値のみを取ることのできる変数です。これらのタイプの変数は、'I'によって定義されます。次のセクションで説明する MIP2 モデルでは、もう一つの離散的な変数、すなわち、半連続変数（semi- continuousvariable）の例を見ます。

MSI 株式会社 Copy right©2015 All rights reserved

```
#include <stdio.h>
#include <stdlib.h>
#include "xprs.h"

int main(int argc, char **argv)
{
 XPRSprob prob;
 int s, status;
 double objval, *sol;

 /* Problem parameters */
 int ncol = 20;
 int nrow = 14;
 int nmip = 10;

 /* Row data */
 char rowtype[]= {  'L','G','E','L','L','L','L','L','L','L','L','L','L','L'};
 double rhs[]  = {1.0/3,0.5,  1,  4,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0};

 /* Column data */
 double obj[]= {  5, 17, 26, 12,  8,  9,  7,  6, 31, 21,0,0,0,0,0,0,0,0,0,0};
 double lb[] = {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,0,0,0,0,0,0,0,0,0,0};
 double ub[] = {0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,1,1,1,1,1,1,1,1,1,1};

 /* Matrix coefficient data */
 int colbeg[] = {0,3,7,11,15,17,19,21,23,26,29,31,33,35,37,39,41,43,45,47,49};
 int rowidx[] = {1,2,4,0,1,2,5,0,1,2,6,0,1,2,7,2,8,2,9,2,10,2,11,0,2,12,0,2,
                 13,3,4,3,5,3,6,3,7,3,8,3,9,3,10,3,11,3,12,3,13};
 double matval[] = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
                    1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1,1,-1};

 /* MIP problem data */
 char miptype[] = {'B','B','B','B','B','B','B','B','B','B'};
 int mipcol[]   = { 10, 11, 12, 13, 14, 15, 16, 17, 18, 19};


 XPRSinit(NULL);                          /* Initialize Xpress-Optimizer */
 XPRScreateprob(&prob);                   /* Create a new problem */

                                           /* Load the problem matrix */
XPRSloadglobal(prob, "FolioMIP1", ncol, nrow, rowtype, rhs, NULL,
               obj, colbeg, NULL, rowidx, matval, lb, ub,
               nmip, 0, miptype, mipcol, NULL, NULL, NULL, NULL, NULL);

XPRSchgobjsense(prob, XPRS_OBJ_MAXIMIZE);  /* Set sense to maximization */
XPRSmipoptimize(prob, "");                 /* Solve the problem */

XPRSgetintattrib(prob, XPRS_MIPSTATUS, &status);  /* Get MIP sol. status */

if((status == XPRS_MIP_OPTIMAL) || (status == XPRS_MIP_SOLUTION))
{
 XPRSgetdblattrib(prob, XPRS_MIPOBJVAL, &objval); /* Get objective value */
 printf("Total return: %g\n", objval);

  sol = (double *)malloc(ncol*sizeof(double));
  XPRSgetmipsol(prob, sol, NULL);        /* Get primal solution */
  for(s=0;s<ncol/2;s++)
   printf("%d: %g%% (%g)\n", s, sol[s]*100, sol[ncol/2+s]);
 }
```


MSI 株式会社 Copy right©2015 All rights reserved

```
 XPRSdestroyprob(prob);                  /* Delete the problem */
 XPRSfree();                             /* Terminate Xpress */

 return 0;
}
```

Xpress-Optimizer に問題をロードするため、ここで関数 XPRSloadglobalを使います。このファンクションの最初の 14 のアーギュメントは、XPRSloadlpについてのものと同じです。19 番目のアーギュメントの使い方については、次のセクションで議論します。残りの 4 つのアーギュメントは、SOS (Special Ordered Set)の定義に関したものです。16 番目のアーギュメントの値 0 は、ここでの問題には、SOS がないことを示しています。このプログラムには、問題をロードするためのファンクションだけでなく、問題をどのように解き、また、ソリューションへのアクセスについてのファンクションも、問題タイプに適応した形になっています。ここでは、この MIP 問題を、分枝限定探索により解きます（最適化関数XPRSmipoptimizeの 2 番目のアーギュメント"" は、デフォルトで MIP アルゴリズムを使用することを意味します）。次いで、MIP ソリューションのステータスを取り出し、もし、整数実行可能解が見つかっていたならば、見つけられた最もよい整数解の目的関数の値と、対応する変数の値をプリントアウトします。このプログラムを実行することにより、以下のソリューションアウトプットが生成されます。最大のリターンは、追加された制約式のため、もともとの LP 問題の値よりも低くなっています。求められたように、ポートフォリオは、4 つの異なる銘柄で形成されています。

```
Total return: 13.1
0: 20% (1)
1: 0% (0)
2: 30% (1)
3: 0% (0)
4: 20% (1)
5: 30% (1)
6: 0% (0)
7: 0% (0)
8: 0% (0)
9: 0% (0)
```

16.3 MIP2：各銘柄の最低投資額の条件を入れる

2 番目の MIP モデルを定式化するために、ここで、再び、第 2 章、および、第 15章の LP モデルから始めます。定式化したい新しい制約式は、「もし、ある銘柄がポートフォリオの中に組み込まれるならば、少なくとも、その銘柄の保有金額が総投資額に占める割合を投資資本の少なくとも 10%にしたい」ということです。今度は、セット変数 $frac_s$ のすべてを、単純に、0 と 0.3 の間の値を取るように制約する代わりに、それらを 0.1 と 0.3 の間の数値を取るか、または、0 という値を取るように制約します。このタイプの変数は半連続変数(semi-continuous variable)と呼ばれます。新しいモデルでは、$frac_s$ のバウンズを、以下の制約式によって置き換えます。

MSI 株式会社 Copy right©2015 All rights reserved

$$\forall s \in SHARES : frac_s = 0 \text{ or } 0.1 \leq frac_s \leq 0.3$$

16.3.1 マトリックスによる表現

この問題は、前章の LP 問題と同じマトリックスを持っているので、ここでは、繰り返しません。唯一の違いは、MIP に関係するカラムのデータです。

16.3.2 Xpress-Optimizer で実行する

下のプログラム`foliomip2.c`により、MIP2 モデルは Optimizer にロードされます。前章の LP 問題と同じマトリックスデータですが、今度は、変数は半連続変数ですからタイプマーカー`'S'`をつけて定義されています。Xpress-Optimizer は、デフォルトで、連続的な限界を 1 と仮定しますので、配列`sclim`で、値 0.1 を指定します。これに関連して、配列`sclim`の中のセミ連続変数、半連続整数変数は、もし、後者が 0 でないならば、配列`lb`の中の値で上書きされることに注意してください。このような合成変数は他にもありますが、その一は、上の説明で触れたセミ連続整数変数(semi-continuous integer variable)で、このタイプの変数は、0 という値を取るか、与えられた限界と上限の間の整数値のみを取る変数です(`'R'` でマークされる)。また、部分整数(partial integer variable)は、下限から、ある与えられた上限までは整数値を取り、この値より上では、連続になる変数です(`'P'`でマークされる)。

```
#include <stdio.h>

#include <stdlib.h>
#include "xprs.h"

int main(int argc, char **argv)
{
 XPRSprob prob;
 int s, status;
 double objval, *sol;

 /* Problem parameters */
 int ncol = 10;
 int nrow = 3;
 int nmip = 10;

 /* Row data */
 char rowtype[] = {  'L','G','E'};
 double rhs[]   = {1.0/3,0.5, 1};

 /* Column data */
 double obj[] = {  5, 17, 26, 12,  8,  9,  7,  6, 31, 21};
 double lb[]  = {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0};
 double ub[]  = {0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3};

 /* Matrix coefficient data */
 int colbeg[]    = {0,  2,    5,    8,    11,12,13,14,15,  17,  19};
 int rowidx[]    = {1,2,0,1,2,0,1,2,0,1,2, 2, 2, 2, 2, 0,2, 0,2};
 double matval[] = {1,1,1,1,1,1,1,1,1,1,1, 1, 1, 1, 1, 1,1, 1,1};
```


MSI 株式会社 Copy right©2015 All rights reserved

```
 /* MIP problem data */
 char miptype[] = {'S','S','S','S','S','S','S','S','S','S'};
 int mipcol[]   = {  0,  1,  2,  3,  4,  5,  6,  7,  8,  9};
 double sclim[] = {0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1};

 XPRSinit(NULL);                          /* Initialize Xpress-Optimizer */
 XPRScreateprob(&prob);                   /* Create a new problem */

                                          /* Load the problem matrix */
 XPRSloadglobal(prob, "FolioSC", ncol, nrow, rowtype, rhs, NULL,
                obj, colbeg, NULL, rowidx, matval, lb, ub,
                nmip, 0, miptype, mipcol, sclim, NULL, NULL, NULL, NULL);

 XPRSchgobjsense(prob, XPRS_OBJ_MAXIMIZE);  /* Set sense to maximization */
 XPRSmipoptimize(prob, "");               /* Solve the problem */

 XPRSgetintattrib(prob, XPRS_MIPSTATUS, &status);  /* Get MIP sol. status */

 if((status == XPRS_MIP_OPTIMAL) || (status == XPRS_MIP_SOLUTION))
 {
  XPRSgetdblattrib(prob, XPRS_MIPOBJVAL, &objval); /* Get objective value */
  printf("Total return: %g\n", objval);

  sol = (double *)malloc(ncol*sizeof(double));
  XPRSgetmipsol(prob, sol, NULL);         /* Get primal solution */
  for(s=0;s<ncol;s++) printf("%d: %g%%\n", s, sol[s]*100);
 }

 XPRSdestroyprob(prob);                   /* Delete the problem */
 XPRSfree();                              /* Terminate Xpress */

 return 0;
}
```

このプログラムを実行すると、下記のようなアウトプットを得ます。

```
Total return: 14.0333
0: 30%
1: 0%
2: 20%

3: 0%
4: 10%
5: 26.6667%
6: 0%
7: 0%
8: 13.3333%
9: 0%
```

今回は、ポートフォリオに 5 つの銘柄が選ばれ、各銘柄とも、合計投資額の少なくとも 10%、最大でも 30%を占めています。制約式を追加したので、最適 MIP ソリューションの値は、この場合も、元の LP 問題の最適解の値よりも低くなっています。

MSI 株式会社 Copy right©2015 All rights reserved

第 17 章 二次計画法

この章では、第 15 章の LP モデルを二次計画法(Quadratic Programming)のモデルにし、下記の点について説明します。

- QP モデルをマトリックス形式に変換する
- Xpress-Optimizer に QP 問題を入力し、問題を解く

17.1 問題の説明

この投資家は、もっと、別の角度から、このポートフォリオ選択問題を見ることもできます。すなわち、予想リターンを最大化し、ハイリスクの投資部分を制限するのではなく、あるレベルのリターンを達成しながら、リスクを最小化することです。この投資家は、証券投資の予想リターンの分散/共分散マトリックスの推定値を得る、というMarkowitzのアイデアを使おうと思っています。このとき、9という最低目標利回りを得ることを前提にしたら、分散を最小化するには、投資家は、どのような投資戦略を採用するべきでしょうか。

17.2 QP

第 2 章で作成したモデルを、上述の新しい観点から見ると、下記の変更を行う必要があります。

- 新しい目的関数は、リターンの合計ではなく、平均分散(meanvariance)となる
- リスクに関係する制約式が消える
- 目標利回りという新しい制約式を追加する

新しい目的関数は、下記で表されるポートフォリオの平均分散です。

$$\sum_{s,t \in SHARES} VAR_{st} \cdot frac_s \cdot frac_t$$

ここで、VARst は、すべての銘柄の分散/共分散マトリックスです。これは、二次の目的関数です。目的関数に、自乗された変数、例えば、$frac_1^2$ や二つの変数が掛け合わされる、例えば、$frac_1 \cdot frac_2$のような項があると、その目的関数は「二次の目的関数」であると言われます。
目標利回りの制約式は、下記のようになります。

$$\sum_{s \in SHARES} RET_s \cdot frac_s \geq 9$$

北米銘柄への投資の限度、すべての資金を投資すること、および、各銘柄への投資の上限は、このモデルでも適用されます。したがって、このモデルの数学的モデルは以下のようになります。

MSI 株式会社 Copy right©2015 All rights reserved

$$\text{minimize} \sum_{s,t \in SHARES} VAR_{st} \cdot frac_s \cdot frac_t$$

$$\sum_{s \in NA} frac_s \geq 0.5$$

$$\sum_{s \in SHARES} frac_s = 1$$

$$\sum_{s \in SHARES} RET_s \cdot frac_s \geq 9$$

$$\forall s \in SHARES : 0 \leq frac_s \leq 0.3$$

17.3 マトリックス表示

Xpress-Optimizer へ問題を入力するために、数式的に表現されているモデルを以下の制約式行列に変換します。(Table 17.1)

Table 17.1: QP matrix

		$frac_1$	$frac_2$	$frac_3$	$frac_4$	$frac_5$	$frac_6$	$frac_7$	$frac_8$	$frac_9$	$frac_{10}$		
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Oper.**	**RHS**
MinNA	***0***	1^0	1^3	1^6	1^9							≥	0.5
Allfrac	***1***	1^1	1^4	1^7	1^{10}	1^{12}	1^{14}	1^{16}	1^{18}	1^{20}	1^{22}	=	1
Yield	***2***	5^2	17^5	26^8	12^{11}	8^{13}	9^{15}	7^{17}	6^{19}	31^{21}	21^{23}	≥	9
	↑ *rowidx*	↑ *matval*											
colbeg		0	3	6	9	12	14	16	18	20	22	24	

前章と同様、マトリックス係数の上付き文字は、配列 `rowidx` と配列 `matval` へのエントリの順番を示し、最初の 3 つのエントリは、強調するためにイタリック体でプリントされています。二次形式の目的関数の係数は、以下の分散/共分散行列で与えられています(Table 17.2) 。

17.4 Xpress-Optimizer で実行する

以下のプログラム `foliocp.c` により、QP 問題は Xpress-Optimizer にロードされ、そして、解かれます。目的関数の二次項の部分は三角形の形式、すなわち、オリジナルの分散/共分散マトリックス(第 7 章の Table 7.1 分散/共分散マトリックスを参照)の上部、または、下部の三角形のどちらかで指定されなければならないことに注意して下さい。ここでの定式化では、分散/共分散マトリックスの上部の三角形を選択していますが、これは、$4 \cdot frac_2 \cdot frac_4 + 4 \cdot frac_4 \cdot frac_2$ の代わりに、これらの項の合計、すなわち、$8 \cdot frac_2 \cdot frac_4$ を指定することを意味しています。Optimizer のインプットについての約束事により、主対角線上の値は 2 倍にする必要があります。マトリックス係数と同様、Optimizer にはノンゼロ係数を持つ二次項のみが指定されます(それにより、配列定義では、下のようなスペースとなります)QP 問題は、関数 `XPRSloadqp` により、Optimizer にロードされます。このファンクションは、目的関数の二次部分のための 4 つの追加的なアーギュメントを後ろに持っ

MSI 株式会社 Copy right©2015 All rights reserved

ていますが、これ以外は、関数XPRSloadlpと同じアーギュメントを持っています。それらは、二次項の数を示すnqt、すべての二次項(qcol1とqcol2)の変数のカラムの数、および、それらの係数qvalです。混合整数二次計画法(MIQP)問題をロードする場合には、関数XPRSloadqglobalを使う必要があります。この関数は、関数XPRSloadqpと同じアーギュメントを取りますが、それらのアーギュメントに加え、前章で関数XPRSloadglobalを使用して導入したMIP問題の向けの9つのアーギュメントも使う必要があります。前の例とは反対に、今度は、目的関数を最小化します。ここで、問題を解き、ソリューションにアクセスするのには、LP問題の場合と同じファンクションを使っていることに注意して下さい。また、MIQP問題を解くときには、第16章で示したMIPのソリューション・ファンクションを使う必要があります。

Table 17.2: Variance/covariance matrix

		$frac_1$	$frac_2$	$frac_3$	$frac_4$	$frac_5$	$frac_6$	$frac_7$	$frac_8$	$frac_9$	$frac_{10}$
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$frac_1$	**0**	0.1									
$frac_2$	**1**		19	-2	4	1	1	1	0.5	10	5
$frac_3$	**2**		-2	28	1	2	1	1		-2	-1
$frac_4$	**3**		4	1	22		1	2		3	4
$frac_5$	**4**		1	2		4	-1.5	-2	-1	1	1
$frac_6$	**5**		1	1	1	-1.5	3.5	2	0.5	1	1.5
$frac_7$	**6**		1	1	2	-2	2	5	0.5	1	2.5
$frac_8$	**7**		0.5			-1	0.5	0.5	1	0.5	0.5
$frac_9$	**8**		10	-2	3	1	1	1	0.5	25	8
$frac_{10}$	**9**		5	-1	4	1	1.5	2.5	0.5	8	16

```
#include <stdio.h>
#include <stdlib.h>
#include "xprs.h"

int main(int argc, char **argv)
{
 XPRSprob prob;
 int s, status;
 double objval, *sol;

 /* Problem parameters */
 int ncol = 10;
 int nrow = 3;
 int nqt  = 43;

 /* Row data */
 char rowtype[] = {'G','E','G'};
 double rhs[]   = {0.5, 1,  9};

 /* Column data */
 double obj[] = {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0};
 double lb[]  = {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0};
 double ub[]  = {0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3};
```


MSI 株式会社 Copy right©2015 All rights reserved

```
/* Matrix coefficient data */
int colbeg[]    = {0,    3,     6,     9,    12, 14, 16, 18, 20,  22,  24};
int rowidx[]    = {0,1,2,0,1, 2,0,1, 2,0,1, 2,1,2,1,2,1,2,1,2,1, 2,1,2};
double matval[] = {1,1,5,1,1,17,1,1,26,1,1,12,1,8,1,9,1,7,1,6,1,31,1,21};

 /* QP problem data */
 int qcol1[]   = {0,
                     1,1,1,1,1,1,1,1,1,
                       2,2,2,2,2,  2,2,
                         3,  3,3,  3,3,
                           4,4,4,4,4,4,
                             5,5,5,5,5,
                               6,6,6,6,
                                 7,7,7,
                                   8,8,
                                     9};
 int qcol2[]   = {0,
                     1,2,3,4,5,6,7,8,9,
                       2,3,4,5,6,  8,9,
                         3,  5,6,  8,9,
                           4,5,6,7,8,9,
                             5,6,7,8,9,
                               6,7,8,9,
                                 7,8,9,
                                   8,9,
                                     9};


  double qval[] = {0.1,
                        19,-2, 4,1,   1, 1,0.5, 10,  5,
                           28, 1,2,   1, 1,     -2, -1,
                              22,     1, 2,      3,  4,
                                 4,-1.5,-2, -1,  1,  1,
                                    3.5, 2,0.5,  1,1.5,
                                         5,0.5,  1,2.5,
                                             1,0.5,0.5,
                                                25,  8,
                                                    16};
  for(s=0;s<nqt;s++) qval[s]*=2;

  XPRSinit(NULL);                          /* Initialize Xpress-Optimizer */
  XPRScreateprob(&prob);                   /* Create a new problem */
                                           /* Load the problem matrix */
  XPRSloadqp(prob, "FolioQP", ncol, nrow, rowtype, rhs, NULL,
             obj, colbeg, NULL, rowidx, matval, lb, ub,
             nqt, qcol1, qcol2, qval);

 XPRSchgobjsense(prob, XPRS_OBJ_MINIMIZE);  /* Set sense to maximization */
 XPRSlpoptimize(prob, "");                 /* Solve the problem */

 XPRSgetintattrib(prob, XPRS_LPSTATUS, &status);   /* Get solution status */

if(status == XPRS_LP_OPTIMAL)
{
 XPRSgetdblattrib(prob, XPRS_LPOBJVAL, &objval);  /* Get objective value */
 printf("Minimum variance: %g\n", objval);
```


MSI 株式会社 Copy right©2015 All rights reserved

```
 sol = (double *)malloc(ncol*sizeof(double));
 XPRSgetlpsol(prob, sol, NULL, NULL, NULL);       /* Get primal solution */
 for(s=0;s<ncol;s++) printf("%d: %g%%\n", s, sol[s]*100);
}

 XPRSdestroyprob(prob);                  /* Delete the problem */
 XPRSfree();                             /* Terminate Xpress */

 return 0;
}
```

```
Minimum variance: 0.557394
0: 30%
1: 7.15401%
2: 7.38237%
3: 5.46362%
4: 12.6561%
5: 5.91283%
6: 0.333491%
7: 29.9979%
8: 1.0997%
9: 6.97039e-06%
```


MSI 株式会社 Copy right©2015 All rights reserved

Appendix A　さらに理解を深めるには

A.1 Xpress のインストレーション、ライセンシング、トラブルシューティング

「Xpress のインストール方法」に関する詳細情報は、ソフトウェアのご提供時、必ず、付属しています。また、Xpress　website のクライアントエリアから、「Licensing User Guide」がダウンロード頂けます。ライセンスキーの取得をご希望のお客さまは Xpress 日本配給元 MSI（株）Xpress 事業部までお問合せください：xpress@msi-jp.cm

万一、ソフトウェアのインストールやライセンスの設定で問題が発生した場合は、下記にご連絡下さい: xpress@msi-jp.com

A.2 ユーザー・ガイド、参照マニュアル、その他の資料

<<Xpress 日本配給元 MSI の WEB サイトでは Xpress に関連するホワイトペーパーや各種参考資料、マニュアルを数多く掲載しております。是非、お役立てください>>

・Xpress　日本配給元 MSI WEB サイト：http://www.msi-jp.com/xpress/

・Xpress を初めてご使用の方へ：http://www.msi-jp.com/xpress/biginners.html

（Xpress ダウンロード、マニュアル、例題等を掲載したページです。）

・学習広場：http://www.msi-jp.com/xpress/learning/square/

（Xpress 学習に関する様々な資料を網羅したページです。）

A.2.1 モデル作成

・Xpress を使用した様々なアプリケーション問題（60 例題集）に関する定式化方法が掲載されています。是非、ご活用ください：https://www.msi-jp.com/xpress/download/xpress60.html

A.2.2 Mosel

・Mosel の使用方法に関する詳細情報は、「Mosel User Guide」をご確認ください。

：http://www.msi-jp.com/xpress/learning/square/mosel_jp.pdf

・Mosel 言語に関する詳細情報は「Mosel libraries Reference Manual」をご確認ください。

：http://www.msi-jp.com/xpress/learning/square/Xpress-Mosel-Libraries.pdf

上記のマニュアルには Mosel Distribution（mmxps、mmodbc、mmive など）のモジュールにより定義されたフィーチャーも含まれています。Mosel Compiler と Mosel Run-time libraries は、”Mosel Libraries Reference Manual”に文書化されています。

A.2.3 BCL

・BCL の使用方法や、C ライブラリ・ファンクション、C++クラスに関する詳細情報は「BCL Reference Manual」をご確認ください。

MSI 株式会社 Copy right©2015 All rights reserved

：https://www.msi-jp.com/xpress/learning/square/bcl_jp.pdf

A.2.4 Optimizer

・Xpress Optimizer に関する詳細情報は「Optimizer Reference Manual」をご確認ください。Optimizer の全機能を掲載しています。また Xpress Optimizer で提供されている問題や、コントロールパラメータを網羅したリストも掲載しています。
：http://www.msi-jp.com/xpress/learning/square/optimizer_jp.pdf

A.2.5 その他のソルバーおよび解法

当ガイド「Xpress Getting started」で紹介していないソルバーや解法に関する詳細情報は、Xpress 日本配給元 MSI(株)Xpress 事業部の WEB サイトをご確認ください。
：http://www.msi-jp.com/xpress/

・Successive Linear programming(SLP)は非線形計画問題を最適に解く方法です。Xpress-SLP ソルバーは Mosel モジュール、mmxslp またはライブラリ形式として提供されています。SLP ソルバーの機能に関する詳細情報は「Xpress-SLP Reference Manual」をご確認ください
：http://www.msi-jp.com/xpress/learning/square/slpref.pdf

・Stochastic Programming(SP)は多くの場合不安定なサブジェクトのアプリケーションデータを扱います。Xpress-SP ソフトウエアは Mosel、mmsp 形式で利用でき SP 問題を表現するためシナリオ・ツリーとして扱うことが可能です。SP ソフトウエアのご使用方法は「Xpress-SP Reference Manual」で解説しています。（Xpress-SP Reference manual は Xpress のウエブサイトにある Mosel open surce からダウンロードしご利用ください。）

・Constraint Programming(CP)は問題を解くアプローチの 1 つでとくにスケジュール作成問題、計画問題でしばし発生する離散的変数に関する非線形制約式を扱うのに適しています。Xpress-Kail ソフトウエアは Artelys Kails Constraint ソルバー向けのインターフェースで特定のスケジュール作成問題、計画問題の総合的なモデル作成サブジェクトとして定義されており、Mosel モジュール、Kails 形式で提供されています。詳細は Xpress-Kails Reference Maunal および Xpress-Kails User Guide をご確認ください。

MSI 株式会社 Copy right©2015 All rights reserved

Appendix B グロッサリー

•Basis（基底）：シンプレックス・アルゴリズムで LP 問題を解くとき、基底は、所与のソリューション内で、どの変数と制約式がアクティブかについての完全な情報を与えます。したがって、問題を解いているある時点で、その時点でのソリューション・アルゴリズムの状態を基底を使ってセーブし、逆に、その基底を使い、ソリューション・アルゴリズム内に基底をセーブした時点での状態を迅速に復元するために使用できます。

•Binary model file（BIM ファイル）：Mosel が利用可能なすべてのプラットフォームで横断的に使えるポータブルな .mos モデルファイルのコンパイル・バージョン。これには、外部のファイルから読まれたデータを含みません。これらのデータは、別個のファイルから読み込む必要があります。こうすることにより、同じ BIM ファイルを、異なるデータセットで実行することができるようになり、便利です。

•Bound（ バウンド）：単一の変数についての等式、または、不等式制約式。Mosel または BCL を通じて Xpress-Optimizer を使うとき、バウンドを使うと、問題を再ロードすることなく、バウンドを変更できます。

•Branch-and-Bound（ 分枝限定法）：MIP 問題の解を得る方法で、LP で実行可能領域を確認しつつ、離散的な変数の実行可能領域内の値を調べ上げることからなっている。一般に、すべてのノードが LP 問題の解を表す Branch-and-Bound ツリーという形で示されます。あるノードと次のノードは、親のノードのバウンドを変更したり制約式を追加したりして結びついています。このような列挙型の方法では、問題が相対的に小さい場合でも、コンピュータでのデータ処理量を爆発的に大きくするので、常に、optimal な MIP 解を得ようとすることは現実的ではありません。

•Branch-and-Cut（ ブランチ・アンド・カット）：Branch- and-Bound に似た MIP 問題の解を得るアルゴリズム。このアルゴリズムでは、サーチ・ツリーの中で、いくつかのノード、または、すべてのノードで、LP 緩和の条件を厳しくして、そこから先のサーチを行わないようにします。

•Builder Component Library（BCL）：プログラミング言語の中で、モデルを、直接、開発するためのモデルビルダーライブラリ。BCL により、ユーザは、専門のモデリング言語のそれと同様な（変数、制約式、インデックス・セットのような）オブジェクトを使って、モデルを定式化することができます。

•Constraint（制約式）：変数の間の関係。制約式のタイプには、等式（演算子は Mosel では=、BCL、C++では==）、不等式（演算子は Mosel、BCL、C++では、> =とく =）、および、
Integrality conditions があります。バウンドは、等式、または、不等式の制約式の特別なケースで

MSI 株式会社 Copy right©2015 All rights reserved

す。

•Cut(カット)：別名 validinequality。あらゆる実行可能解では満足されなければならないが、整数解のセットの特性を示すのには必要ではない MIP 問題に追加される制約式。Cut は、LP 緩和を厳しくして、LP 解の実行可能領域を MIP 解スペースの convex hull に近付けます。

•Variable(変数)：ソリューション・アルゴリズムにより値を割り当てられる未知数。基本的な変数タイプは、連続変数（所与の下限、および、上限の間の連続的なドメインの値を取る）、可変の、取っている値）。離散的な変数タイプには、バイナリ変数（indicator variable とも呼ばれる。この変数は、値 0、または、値 1 のみを取る)、整数変数（所与の範囲の間の整数値のみを取る）、セミ連続変数（値 0 か、所与の下限と上限の範囲の連続的な値を取る）、セミ連続整数変数（値 0 か、所与の下限と上限の範囲の整数値を取る）、部分整数変数（所与の下限と上限の範囲では整数値を取り、上限を越えると連続変数となる）があります。

•Declaration(デクラレーション)：オブジェクトのデクラレーションにより、そのフォームとタイプを宣言します。通常、その内容の定義に先行します。Mosel では、基本タイプとリニアの制約式のデクラレーションはオプションですが、変数は、いつも宣言されなければなりません。サブルーチンは、モデル内で使われるならば、それらの定義の前で宣言されていなければなりません。

•Dynamicset, dynamicarray(動的セット、動的配列)：Mosel では、サイズがわからない場合は、セットと配列は、すべて、ダイナミック・オブジェクトとして生成されます。これらのオブジェクトは、それらに割り当てられた内容によって動的に大きくなり、また、セットは小さくもなります。動的なセットは、静的にすることもできます（finalize） 。こうすることにより、これらのセットによりインデックスを付され、finalize した後に宣言された配列の処理が効率的になります。さらに重要なことは、これにより、もしセットが動的ならば、検出できない‘out of range’エラーを Mosel がチェックできるようになります。スパース（密度の低い）データテーブルは動的な配列に蓄えられるべきですが、密度の高いテーブルは、静的な配列に蓄えたほうが、より効率的です。

•Heuristic(ヒューリスティック)：問題の実現可能解を見つけるためのアルゴリズム。ヒューリスティックによっては、解のよしあしの限界を保証できるけれど、普通、optimality については、なんの保障もありません。

•Index set （インデックス・セット)：セットは、配列にインデックスを付けるために使われます。文字列インデックスを使うことにより、Mosel や BCL により生成されるアウトプットの理解が容易になります。

•Interactive Visual Environment(IVE)：Mosel のための開発環境で、他にもいろいろツールを持

MSI 株式会社 Copy right©2015 All rights reserved

っていますが、これにより、特に、ソリューション情報のグラフィカルなディスプレイが便利です。

•Linear Programming (LP) problem(線形計画法問題)：すべての制約式、および、目的関数が、変数の一次式で表現される数理計画法の問題。ここで、変数は、すべて、連続変数、すなわち、(通常)、非負の真数を取ります。広く知られている問題で、そこでは、効率的なアルゴリズム(Simplex 法、内点法)の存在が、よく知られています。

•Loop(ループ)：ループは、何度も繰り返えす必要のある動作をグループ化します。繰り返えしは、インデックスのすべての値、または、カウンター(Mosel の forall) 、または、ある特定の条件が満たされているかどうかによって(Mosel の while、repeat until) 行われます。

•LP relaxation (LP 緩和)：LP 緩和は、MIP 問題で、変数の整数条件を緩めることによって得られます。

•Mathematical Programming problem(数理計画問題)：一組の変数、これらの変数を組み合わせた制約式、および、最大化、または、最小化される目的関数からなる問題。

•Matrix(マトリックス)：一次制約式からなる数理計画問題のマトリックス表示で、このテーブルでは、column が変数、row が制約式を表すように作成されます。このテーブルのエントリーは、制約式の変数の係数であり、通常、スパース(まばら)なフォーマットで蓄えられており、ノンゼロの係数だけが入力されます。

•Mixed Integer Programming (MIP) problem(混合整数計画法(MIP) 問題)：ちょうど LP と同じように、制約式と目的関数が一次式ですが、変数は、離散的な、もしくは、連続的なドメインを持つことができる数理計画法問題。このタイプの問題を解くには、LP 技法と Branch-and-Bound(分枝限定法)のテクニックを結び付けたもの使います。

•Modeling language(モデリング言語)：(Mosel 言語などのような)高級言語で、ユーザーは、数理計画問題を数式的な表現に近い形式で記述することができます。そして、自動的にソルバーが必要とする表現に変換します。

•Model(モデル)：問題を数学的に表現したもの。また、Mosel や BCL などのモデリングツールでのインプリメンテーションを意味するのにも使われます。

•Module(モジュール)： Dynamic shared object (DSO) とも呼ばれます。Mosel Native

MSI 株式会社 Copy right©2015 All rights reserved

Interface により規定されている約束事を守る、Cプログラミング言語によって書かれた dynamic library。モジュールによって、ユーザーは、（例えば、問題固有のデータ処理を行ったり、外部のソルバーやソリューション・アルゴリズムとの接続などを行ったりして）新しいフィーチャーを加え、Mosel 言語を拡張できます。FICO 社によって提供されているモジュールには、Xpress-Optimizer（LP 、MIP、QP） や Xpress-SLP へのアクセス、（例えば、ODBC 経由の）データ・ハンドリング機能、システム・ファンクションへのアクセスなどがあります。

・Mosel（ モーゼル）：モデリングとプログラミング言語である Mosel 言語、Mosel モデルをアプリケーションに埋め込むための Mosel ライブラリ、および、モジュールという形で外部に Mosel 言語を開く Mosel ネイティブインタフェースからなっているモデル作成とソルビング環境。

・Mosel Native Interface（NI） モーゼル・ネイティブ・インタフェース：実行の間に、Mosel モデルへのアクセスを行えるようにするサブルーチンライブラリ。また、Mosel モジュールにより守られるべき約束事も定義します。NI により、ユーザーは、新しい機能によって Mosel 言語を拡張することができます。

・Newton-Barrier algorithm（ ニュートン・バリア・アルゴリズム）：内点法アルゴリズムとも呼ばれます。LP 問題と QP 問題のためのソリューション・アルゴリズムで、実行可能領域内のある点から、実行可能領域セットの境界に触れることなく、最適解に進む方式を取るアルゴリズム。

・Non-linear Programming （NLP) problem（ 非線形計画法（NLP） 問題）：非線形な制約式、または、目的関数を持つ数理計画問題。しばしば、よい（ローカル・オプティマム）解を得るために、ヒューリスティック、ないしは、近似法が使用されます。Xpress-MP により提供される、このタイプの問題を解く方法は、Successive LinearProgramming（SLP） です。

・Objective function（ 目的関数）：変数の組合せで表現される最小化、または、最大化される数式（このマニュアルでは、一次式、および、二次式の数式だけについて、説明されています）。

・Optimization（ 最適化）：与えられた目的関数の値を最小化するか、または、最大化する問題の実行可能解を見つけること。

・Optimizer： LP 、MIP、および、QP のための Xpress-MP ソルバー。ライブラリ、または、スタンドアロンのプログラムという形で利用可能です。

・Overloading（ オーバーローディング）：いろいろなタイプ、または、いろいろなアーギュメントのため、いくつかのバージョンで定義されているサブルーチン。いろいろなオペランドタイプ、または、オ

MSI 株式会社 Copy right©2015 All rights reserved

ペランドタイプの組合せ向けに定義されるオペレータ。

・Parameter（ パラメータ）：この用語は、文脈によって、少し違う意味をいくつか持っています。（Mosel の）モデルパラメータの設定は、例えば、ランを行うときに、種々の入力データセットを定義するために変更できます。（Optimizer の中で、通常、問題アトリビュートと呼ばれている）問題パラメータにより、問題についての情報（例えば、ソリューションステータス）へのアクセスが行え、普通、リードオンリーです。また、アルゴリズムコントロールパラメータは、（アルゴリズムの選択、トレランスの設定などの）アルゴリズムの設定をコントロールするために用いられます。

・Problem instance（ 問題例）：特定のデータセットを備えている数理計画問題。

・Quadratic Programming （QP) problem（ 二次計画法（QP） 問題）：目的関数に二次項を持つという点で、 LP 問題と異なります。ただし、制約式は一次式のままです。変数は、連続でも離散的でもかまいません。後者の場合、これは、混合整数二次計画（MIQP） モデルのことを話していることを意味します。

・Range set （レンジ・セット）：（Mosel において）連続的な整数の集合。

・Right hand side （RHS) （右手側（RHS））：（ 一次）制約式の右辺定数。（例えば、マトリックス表示で使われる）標準のフォーマットでは、等号、不等号の左側に、変数を含むすべての項を、そして、等号、不等号の右側に定数を書きます。

・Selection Statement （選択ステートメント）：プログラムの中で、取り得る種々のアクションの中からの選択を表現するステートメント。Mosel では、これらの選択ステートメントは、
if/then/elif/then/else/end-if/case です。

・Simplex algorithm（ シンプレックス・アルゴリズム）：LP 問題のための解法アルゴリズム。シンプレックス・アルゴリズムのアイデアは、目的関数の値を改善するために、LP 問題の実行可能解の頂点から別の頂点に動いて行きます。

・Solution（ ソリューション、解）：この用語は 2 つの異なる意味で使われます。その一は、制約式の条件、すべてを満たしているすべての変数に値を割り当てることを意味しますが、この場合、すべての実行可能解が含まれます。また、その二は、最もよい実行可能解が捜される最適化問題で、すなわち、与えられた目的関数の値を最小化するか、または、最大化する解です。この場合、これは、通常、最適解（optimum solution） と同じです。

MSI 株式会社 Copy right©2015 All rights reserved

•Solver(ソルバー):問題を解く(通常、最適化を行う)のに使われるソフトウェア。Xpress-MP では、Xpress-Optimizer がこれに当たります。

•Status information(ステータス情報): Mosel 、BCL、Xpress-Optimizer は、LP や MIP のステータス情報を提供するのに、異なるパラメータを定義します。ステータス情報を見ることで、ユーザーは、問題が正しく解かれたか、そして、ソリューションが利用可能かなどのしることができます。

•Subroutine (サブルーチン):プログラムを小さいサブタスクに分割し、より理解しやすく、したがって、使いやすくするための基礎構造。Mosel では、サブルーチンは、手順(procedure) 、ファンクション(function) という形で使用できます。Procedure は、プログラム・ステートメントとしてコールされ、値を返して寄越しませんが(no returnvalue)、function は、返して寄越す値を使用する expression としてコールされなければなりません。

•Successive Linear Programming(連続線形計画法(SLP)) : LP 問題を繰り返し解くことにより、NLP 問題を解く方法。

MSI 株式会社 Copy right©2015 All rights reserved