S+NUOPT V2 マニュアル

目次

１．はじめに 5
１‐１　本書の使い方 5
１‐２　S+NUOPT の構造 5
１‐３　S+NUOPT に付属しているモデル一覧 7
２．数理計画モデルの記述（基本編） 8
２‐１　定数（Parameter），集合（Set）の宣言と定義 9
２－１－１．集合（Set）の宣言と初期化 10
２－１－２．定数（Parameter）の宣言と初期化 11
２－２　集合の要素（Element）の宣言と利用 12
２－３　変数（Variable）の宣言と利用 12
２－３－１．変数（Variable）に初期値を与える 13
２－４　式オブジェクト（Expression）の宣言と代入演算子による定義 13
２－５　目的関数オブジェクト（Objective）の宣言と代入演算子による定義 14
２－６　添え字の範囲の和を取る演算子 Sum 14
２－７　定式の展開 System と最適化の実行 solve 15
２－７－１．数理計画モデルのデータを引数として，System から与える 17
２－８　制約式 Constraint の追加 18
２－８－１．制約式の名前付け，除去，復活（delete.con,restore.con） 19
２－９　整数変数 IntegerVariable 20
３．数理計画問題の記述（応用編） 23
３‐１　条件式と資金調達問題 23
３‐２　SymmetricMatrix と最小固有値問題 26
３‐３　wcsp と集合分割問題 27
３‐３‐１．wcsp による１指標の２分割問題 27
３‐３‐２．wcsp による３指標の２分割問題 28
３‐３‐３．wcsp による多分割問題（selection とハード，セミハード，ソフト制約） 32
３‐３‐４．乱数の種の設定による解の違い 34
３‐３‐５．アルゴリズム wcsp を使うときの注意点 35
４．S-PLUS とのデータ連携 36
４‐１　vector 型で定数オブジェクト（Parameter）を初期化する 36
４‐１‐１．添え字を持たない Parameter を初期化する 36
４‐１‐２．添え字を持つ Parameter を初期化する 36
４‐１‐３．整数列以外を添え字として持つ Parameter を初期化する 39

４‐２ matrix 型で Parmeter を初期化する 39
４－３ list 型で定数オブジェクト Parameter を初期化する 41
４‐４ 最適化の結果（Variable/Expression）を取り出す 42
４‐４‐１．変数や式（Variable/Expression）の値を array 型のデータとして取り出す 43
４‐４‐２．変数や式（Variable/Expression）の値を list 型のデータとして取り出す . 45
５．ポートフォリオ最適化 47
５‐１ 基本的なマルコビッツモデル 47
５‐２ さまざまなリスク尺度によるポートフォリオ最適化 51
５－２－１． 分散をリスクとした場合 51
５－２－２． 絶対偏差をリスクとした場合 53
５－２－３． １次の下方部分積率（LPM）をリスクとした場合 55
５－２－４．CVaR をリスクとした場合 56
５‐３ コンパクト分解（大規模ポートフォリオ最適化） 57
５‐４ 端株処理 59
５‐５ 銘柄グルーピング 62
５‐６ Maximum Drawdown 67
５－７ Sharpe Ratio 71
６．半正定値計画法の利用 73
６‐１ 半正定値行列の取得 73
６‐２ ロバスト最適化 74
７．非線形フィッティング 78
７‐１ イールドカーブのフィッティング 78
７‐２ 格付け推移行列の推定 80
７‐３ ロジスティック回帰 82
８．最適化ソルバーNUOPT 84
８‐１ nuopt.options()を使って NUOPT をカスタマイズする 84
８‐１‐１．特殊な大規模二次計画問題を高速に解く 85
８‐１‐２．線形計画問題・二次計画問題をより高精度で解く 85
８‐１‐３．計算機資源の利用を制限する 85
８‐１‐４．分枝限定法のチューニング 86
８‐１‐５．非線形計画法，半正定値計画法の安定化のためのチューニング 86
８‐１‐６．ヒープメモリをクリアする 88
８‐２ エラーメッセージ 89
８‐２‐１．モデリング言語解釈部からのエラー 89
８‐２‐２．NUOPT が出力するエラー 91

８-３ solveQP.. 95

１．はじめに

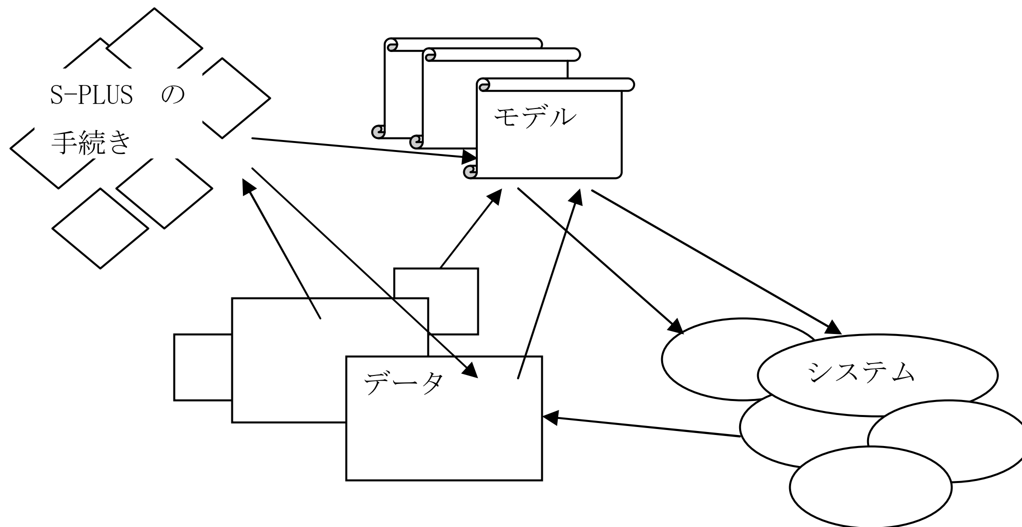
S+NUOPT は汎用の数理計画法のパッケージです．汎用というのは様々な分野に応用できることを示している一方で，具体的な結果を出すのにはいずれも「ひと手間」かけねばならないということをも同時に示しています．このマニュアルは統計や金融など，具体的な応用を抱えているユーザーの方々を想定して，その「ひと手間」をできるだけ軽減することを目的として書かれました．S+NUOPT と S-PLUS の最大の違いは,「数理計画モデル」と呼ぶ問題の定式がかならず必要となる点です．たとえば S-PLUS には重回帰を行う lsfit という手続きがあらかじめ用意されていて，ユーザーはデータを決まったフォーマットで用意してこの手続きをコールすれば結果が得られるのですが，S+NUOPT では重回帰一つ行うのにも，何が変数で，何を最小化したいのかという数理計画モデルを記述する必要があり，一般に導入コストがかかります．ただし，モデル記述の幅は非常に柔軟でありかつ広く，通常の重回帰モジュールとちがってパラメータの範囲を設定することや，パラメータ同士の関係式（制約と言います）を定義すること，またパラメータを自動的に絞り込むといったことも可能です．そこで，このマニュアルには導入部分に関してはできるだけ豊富なモデルとデータのサンプルを与え，その応用の筋道を示すことによって，S+NUOPT の可能性を実感していただくように設計しました．

１−１　本書の使い方

次項でこのマニュアルに解説されている（パッケージに含まれている）サンプルの一覧とその書いてある場所を示します．まずはご自身の興味のあるところを拾い読みして感じをつかんでいただければと思います．最適化パッケージの利用上「数理計画モデル」作成は避けて通ることができませんが，サンプルで要領をつかんでしまえば，ご自分の応用に役立てるように書き換えるのはかなり少ない手間で可能である場合もあります．特に NUOPT の大きな応用分野である金融ポートフォリオについては主要なものをほぼ含んでいるため，金融ポートフォリオと最適化の関係については本パッケージとマニュアルを用いればほぼ体感できるのではと考えております．
２章は簡単な重回帰モデルを題材として，数理計画モデルを記述する基本的な道具立てについて網羅的に解説しています．３章は比較的発展的な使い方を想定して，特定の場合に有効なモデル記述法・解法について解説しています．４章は S-PLUS でのデータ解析を実際に行われている方を想定し，お持ちのデータをどのようにしたら S+NUOPT に渡して，最適化のためのデータとすることができるのかを解説します．５章以降は例題と解説です．

１−２　S+NUOPT の構造

S+NUOPT は汎用統計解析パッケージである S-PLUS の環境の中で対話的に最適化が行えるというところに最大の特徴があります．S+NUOPT が扱うオブジェクトは大きく分けて３つあります．一つは数理計画モデルの入出力データ（変数の上下限や解など），もう一つは数理計画の定式のパターンを記述したモデルそのもの．最後の一つは数理計画モデルとデータを結び付けて解析の対象となる具体的な問題です．S+NUOPT はこれらをそれぞれデータ，モデル，システムと呼び，いずれも S-PLUS 環境の中のオブジェクトとして扱います．その構造は次のように記述できます．問題（システム）を解いた結果もまたデータとして S-PLUS 環境に蓄えられることになりますので，最適化を連鎖的に行ったりすることも S+NUOPT の中で可能です．
S+NUOPT の特徴はモデルの記述が S-PLUS に埋め込まれた別の言語に従って行われるということです．２，３章でその記述の具体的な意味と方法について解説いたします．



S-PLUS には統計演算をはじめとするデータ処理機能が充実していますので，その機能を使えば最適化モデルで利用するデータを素早く準備することができます．S-PLUS が備えている実際的なデータ例（例えば”freeny” や “state” はこのマニュアルでも利用しています）も最適化モデルのテスト的な入力として適しています．さらに，最適化の結果も S-PLUS のデータとして扱われますので，S-PLUS が持つ豊富な図示機能を結果の理解に役立てることができます.

１−３　S+NUOPT に付属しているモデル一覧

S+NUOPTV2 には次のような例があらかじめ組み込まれています．興味のある例題を中心に試して感触をつかむのも良い方法です．

例題	内容	モデル名	適合するデータの例	章番号
重回帰	データに特化	Nlsfit	freeny.x/y	2.1
	汎用	Nlsfit.gen	freeny.x/y,stack.x/loss	2.7.1
	パラメータの線形制約付き	Nlsfit.eq	freeny.x/y,stack.x/loss	2.8
	名前つき制約	Nlsfit.eq.named	freeny.x/y,stack.x/loss	2.8.1
	パラメータの選択付き	Nlsfit.int	freeny.x/y,stack.x/loss	2.9
キャッシュフローマッチング	資金調達の問題(条件式の応用)	Cashflow	Cashflow.flow/bf	3.1
半正定値計画導入	最小固有値の取得	MinLambda	var(air)	3.2
集合の分割	1指標2分割	Half	state.x77	3.3.1
	多指標2分割	Half2	state.x77	3.3.2
	1指標多分割	Partition	state.x77	3.3.3
ポートフォリオ最適化	マルコビッツモデル基本	Marko	R.60x4	5.1
	分散	MinVar	R.8000x5	5.2.1
	絶対偏差	MinMad	R.8000x5	5.2.2
	1次の下方部分積率	MinLPM1	R.8000x5	5.2.3
	CVaR	MinCVaR	R.8000x5	5.2.4
	コンパクト分解	MinVar	R.60x1000	5.3
	Maximum Drawdown	MinMaxDD	R.521x95	5.6
	Sharpe Ratio最大化	Sharpe	R.60x200	5.7
	Sharpe Ratio最大化(QP)	Sharpe.qp	R.60x200	5.7
離散最適化とポートフォリオ	端株処理	RoundLot	RoundLot.unit/x	5.4
	銘柄のグルーピング	Basket	Basket.flow/fhigh/fbar/W	5.5
半正定値計画の応用	ロバスト最適化	Robust	Robust.sigU/sigL/mu	6.2
	相関行列の成形	Cormat	Cormat.A	6.1
非線形回帰	イールドカーブの推定	Yield	Yield.telem/term/price	7.1
	格付け推移行列の推定	Rating	Rating.Q0	7.2
	ロジスティック回帰	LogReg	LogReg.X/t/test.X/test.t	7.3

２．数理計画モデルの記述（基本編）

ここでは S-PLUS の組み込み関数 lsfit が行っている重回帰モデルを例にとって，S+NUOPT におけるモデル記述について解説します．この章において，モデル記述の道具はほぼすべて現れます．より高度な機能や各手法に特化した記述については３章で触れています．
そもそも重回帰という操作は，被説明変数Yを説明変数Xの線形な関数で記述したときの残差の二乗和を最小化していることになるので，数理計画問題を解いているとみなすことができます．定式化は次のようになります．

変数	v_j	（$j \in P$ ：線形関数の一次の係数の集合）
	v_0	（定数項）
最小化	$\sum_{i \in S} e_i^2$	（残差の二乗和）
定義	$e_i \equiv (v_0 + \sum_{j \in P} X_{i,j} v_j) - Y_i$	（$i \in S$ ：観測点）
		$X_{i,j}$ ：観測点$i \in S$における説明変数$j \in P$の値
		Y_i ：観測点$i \in S$における被説明変数の値

S+NUOPT で数理計画問題を解くには，上記のような問題を記述する数式の情報を独自の記述方法で表現し，S-PLUS の手続きとしておく必要があります．以降ではその手続きのことを「(数理計画) モデル記述」と呼びます．次が上記の数式に対応するモデル記述です．

```
NIsfit <- function()
{
# freeny.y を Y として, freeny.x を X として用いる準備
   ydata <- as.vector(freeny.y)
   S <- Set()
   Y <- Parameter(index=S,as.array(ydata))
   P <- Set()
   X <- Parameter(index=dprod(S,P),freeny.x)
#  ここからモデル記述本体
   i <- Element(set=S)
   j <- Element(set=P)
   v <- Variable(index=j)
   v0 <- Variable()
   e <- Expression(index=i)
   e[i] ~ Sum(X[i,j]*v[j],j) + v0 - Y[i]
   esum <- Objective(type=minimize)
   esum ~ Sum(e[i]*e[i],i)
}
```

２－１　定数（Parameter），集合（Set）の宣言と定義

最初の部分は，重回帰に freeny.x および freeny.y という S-PLUS ビルトインのデータ（説明は help(freeny)にございます）を用いるための準備です．

```
ydata <- as.vector(freeny.y)
S <- Set()
Y <- Parameter(index=S,as.array(ydata))
P <- Set()
X <- Parameter(index=dprod(S,P),freeny.x)
```

freeny.y を vector 型に変換

観測点の添え字集合の宣言

観測点の値の宣言と初期化

説明変数の集合の宣言

観測点における説明変数の値の定義

定数については宣言と同時に，S-PLUS オブジェクトを渡して初期化を行っています．初期化した結果，その内容が意図通りとなっているかは，コマンドプロンプトで確認できます．具体的にはコマンドプロンプトから

```
ydata <- as.vector(freeny.y)
S <- Set()
Y <- Parameter(index=S,as.array(ydata))
P <- Set()
X <- Parameter(index=dprod(S,P),freeny.x)
```

と入力して，Y と X を表示させます．

```
> Y
      1       2       3       4       5       6       7       8
8.79236 8.79137 8.81486 8.81301 8.90751 8.93673 8.96161 8.96044

      9      10      11      12      13      14      15      16
9.00868 9.03049 9.06906 9.05871 9.10698 9.12685 9.17096 9.18665

     17      18      19      20      21      22      23     24
9.23823 9.26487 9.28436 9.31378 9.35025 9.35835 9.39767 9.4215

     25      26      27     28      29      30      31     32      33
9.44223 9.48721 9.52374 9.5398 9.58123 9.60048 9.64496 9.6439 9.69405

     34      35      36      37      38      39
9.69958 9.68683 9.71774 9.74924 9.77536 9.79424
attr(, "indexes"):
[1] "*"
> X
   income level lag quarterly revenue market potential price index
```

```
 1      5.82110                 8.79636         12.9699    4.70997
 2      5.82558                 8.79236         12.9733    4.70217
 3      5.83112                 8.79137         12.9774    4.68944
 4      5.84046                 8.81486         12.9806    4.68558
 5      5.85036                 8.81301         12.9831    4.64019
 6      5.86464                 8.90751         12.9854    4.62553
 7      5.87769                 8.93673         12.9900    4.61991
 8      5.89763                 8.96161         12.9943    4.61654
 9      5.92574                 8.96044         12.9992    4.61407
10      5.94232                 9.00868         13.0033    4.60766
11      5.95365                 9.03049         13.0099    4.60227
12      5.96120                 9.06906         13.0159    4.58960
13      5.97805                 9.05871         13.0212    4.57592
14      6.00377                 9.10698         13.0265    4.58661
15      6.02829                 9.12685         13.0351    4.57997
16      6.03475                 9.17096         13.0429    4.57176
17      6.03906                 9.18665         13.0497    4.56104
18      6.05046                 9.23823         13.0551    4.54906
19      6.05563                 9.26487         13.0634    4.53957
20      6.06093                 9.28436         13.0693    4.51018
21      6.07103                 9.31378         13.0737    4.50352
22      6.08018                 9.35025         13.0770    4.49360
23      6.08858                 9.35835         13.0849    4.46505
24      6.10199                 9.39767         13.0918    4.44924
25      6.11207                 9.42150         13.0950    4.43966
26      6.11596                 9.44223         13.0984    4.42025
27      6.12129                 9.48721         13.1089    4.41060
28      6.12200                 9.52374         13.1169    4.41151
29      6.13119                 9.53980         13.1222    4.39810
30      6.14705                 9.58123         13.1266    4.38513
   income level lag quarterly revenue market potential price index
31      6.15336                 9.60048         13.1356    4.37320
32      6.15627                 9.64496         13.1415    4.32770
33      6.16274                 9.64390         13.1444    4.32023
34      6.17369                 9.69405         13.1459    4.30909
35      6.16135                 9.69958         13.1520    4.30909
36      6.18231                 9.68683         13.1593    4.30552
37      6.18768                 9.71774         13.1579    4.29627
38      6.19377                 9.74924         13.1625    4.27839
39      6.20030                 9.77536         13.1664    4.27789
attr(, "indexes"):
[1] "*" "*"
```

どんなS-PLUSオブジェクトを定数の初期化に用いることができるかは４章「S-PLUSとのデータ連携」で具体的に解説しています.

２−１−１.集合（Set）の宣言と初期化

集合の宣言と初期化の一般的な形は

```
名前 <- Set()
名前 <- Set(集合の内容)
```

という形で行います．「集合の内容」はS-PLUSのベクトル型を与えることができます．

```
S <- Set(1:5)
S1 <- Set(c("apple","orange","banana"))
S2 <- Set(seq(1,10,3))
```

確認してみましょう．

```
> S
{ 1 2 3 4 5 }
> S1
{ apple orange banana }
> S2
{ 1 4 7 10 }
```

S+NUOPTでは宣言のときに集合の内容を与えなくても，定数の入力の際に同時に定義される機能（自動代入）があるので不要な場合もあります．

２－１－２．定数（Parameter）の宣言と初期化

定数の宣言と初期化の一般的な形は以下です．

```
名前 <- Parameter() # 添え字，初期化なし
名前 <- Parameter(定数の内容) # 添え字なし
名前 <- Parameter(index=添え字集合/要素) # 初期化なし
名前 <- Parameter(index=添え字集合/要素,定数の内容) #  初期化，添え字あり
```

S+NUOPTでは，定数（最適化において変化しない量）はすべてParameterと呼び，添え字をいくつ持つかによって名前は変化しません．たとえばスカラは添え字を持たないParameter，定数ベクトルは一つの添え字を持つParameter，定数行列は二つの添え字を持つParameterです．
スカラ以外の定数の場合，index=の後には，添え字を示す集合や要素の組を並べます．「定数の内容」はarray型もしくはlist型のS-PLUSオブジェクトです．次がその例です．

```
p <- Parameter(2.85)
q <- Parameter()  # 0 だと解釈される
Y <- Parameter(index=S,as.array(ydata))
i <- Element(set=S)                    # 集合の要素の宣言
a <- Parameter(index=i,as.array())  # 要素を添え字として並べてもよい
X <- Parameter(index=dprod(S,P),freeny.x)
X <- Parameter(index=dprod(i,P),freeny.x)  # 要素と集合を混ぜてもよい
```

Xの記述のように，宣言において二つ以上の集合（もしくは要素）をならべるときにはdprod()を使わねばならないことにご注意ください．定数値がうまく代入されているかどう

かはコマンドプロンプトから確認することができます．詳しくは４章の「S-PLUS とのデータ連携」をご覧ください．

２-２　集合の要素（Element）の宣言と利用

モデル記述 Nlsfit では，Set,Parameter の宣言のあとに，数理計画問題本体の記述があります．


```
i <- Element(set=S)
j <- Element(set=P)
v <- Variable(index=j)
v0 <- Variable()
e <- Expression(index=i)
e[i] ~ Sum(X[i,j]*v[j],j) + v0 - Y[i]
esum <- Objective(type=minimize)
esum ~ Sum(e[i]*e[i],i)
```

式の実体の定義に対応するのは e[i]の定義と目的関数 esum の定義のみで，その他は宣言です．
まず式を表現するときの添え字として用いる，集合の要素，Element オブジェクトの宣言を行っています．S+NUOPT は，集合の要素を添え字として宣言し，添え字として用います．
集合の要素の宣言の形は

```
名前 <- Element(set=集合)
```

という形式です．Element は宣言されたときの集合を常に覚えており，式の中で使われるとその集合の範囲内を回ります．たとえば，この場合，i という要素オブジェクト（Element）は，

```
e[i] ~ Sum(X[i,j]*v[j],j) + v0 - Y[i]  # e[i]の定義
esum ~ Sum(e[i]*e[i],i)                # e[i]の和を esum とする
```

のように使われていますが，e[i]の定義は，i が定義されている集合 S の要素すべてについて記述したのと同じことになりますし，Sum は二つ目の引数に与えられた要素すべてについて和を取るものと解釈されます．
また，要素オブジェクトは他のオブジェクトの添え字を定義するときに集合の代わりに使うことができます．

```
v <- Variable(index=j)     # Variable(index=P)と同じ意味
e <- Expression(index=i)   # Expression(index=S)と同じ意味
```

がその例です．

２-３　変数（Variable）の宣言と利用

変数は数理計画問題における未知の値です．この問題においては v と v0 という変数が定義されています．前者は j という添え字付き，後者はスカラーであると宣言されています．
宣言の形式は

```
名前 <- Variable() # 添え字なし
名前 <- Variable(index=添え字集合/要素) # 添え字あり
```

２－３－１.変数（Variable）に初期値を与える

変数に対する代入演算 ~ を用いて変数に初期値を与えることができます.
次がその例です.

```
v0 ~ 2.5    # v0 の初期値は 2.5 とする
v[j]  ~ 1   # v[j]すべてに 1 という初期値を設定する
```

デフォルトの初期値は 0 です.
S+NUOPT はユーザの指定がなくとも，適切な初期値を自分で設定する機能をもっていますが，初期値を与える必要があるのは，変数が初期値のままでは，値または微係数が評価できない式が存在するケースです．たとえば x を変数として，

```
1.0/x          # 割り算（原点において値が評価できない）
sqrt(x)        # 平方根（原点において微係数が評価できない）
```

といった関数がその例です．このような場合には 1 などの無難な初期値を与えておく工夫が計算の安定性に寄与します.

２－４　式オブジェクト（Expression）の宣言と代入演算子による定義

Nlsfit において e[i]は

$$e_i \equiv (v_0 + \sum_{j \in P} X_{i,j} v_j) - Y_i$$

の e_i に相当します．このようにまとまった式の断片に名前を付けるときには，式オブジェクト（Expression）を使います．宣言の形は Parameter と全く同じです.

```
名前 <- Expression() # 添え字なし
名前 <- Expression(index=添え字集合/要素) # 添え字なし
```

Expression は代入しなければ常に値０と解釈されます（初期化されていない Parameter も同じです）.
Expression や Parameter に値を代入するには代入演算子 ~ を用います（=ではないことに注意してください）.
代入の左辺には単体の Expression/Parameter オブジェクトが来なければければなりません.

```
x[i] + y[i] ~ z[i]   # エラー
```

という記述はエラーとなります．意味上，右辺の添え字は残らず左辺の添え字に表れる必要があります．次はエラーとなります.

```
z ~ x[i]         # エラー
```

次のような記述も，添え字ｊのどの要素に関する記述なのかわからないのでエラーになります．

```
y[i] ~ u[i,j]    # エラー
```

左辺の添え字が右辺にかならずしも表れる必要はありません．

```
w[i,j] ~ x[i]    # 正しい記述
```

上記は，w[i,j]に添え字ｊにはよらず同じ式 x[i]を代入する，という正しい記述です．

２－５　目的関数オブジェクト（Objective）の宣言と代入演算子による定義

目的関数オブジェクト（Objective）は特殊な Expression と見なすことができ，使用方法は Expression と全く同様に代入して使います．目的関数とは，最小化，あるいは最大化したい式の内容です．宣言の形式は次のように，最小化すべきなのか，最大化すべきなのかを与えます．目的関数は一つでなければならないという性質上，Objective は添え字を持ちません．

```
名前 <- Objective(type=miminize) # 最小化
名前 <- Objective(type=maximize) # 最大化
```

次は目的関数の宣言および定義の例です．

```
esum <- Objective(type=minimize)
esum ~ Sum(e[i]*e[i],i)  # e[i]の和
```

２－６　添え字の範囲の和を取る演算子 Sum

数理計画問題の記述には

$$\sum_{i \in S} e_i^2$$

のように添え字のわたる範囲で和を取る，という操作がよく現れます．これは，S+NUOPT において Sum 演算を用いて次のように記述します．

```
Sum(e[i]*e[i],i)  # e[i]の和
```

Sum 演算は一般に次のような形式で記述します．

```
Sum(式,添え字)
          #  例  Sum(a[i],i)
Sum(式,添え字 1,添え字 2,...,添え字 n)
          #  例 Sum(M[i,j],i,j)
Sum(式,添え字 1,添え字 2,...,添え字 n,条件式 1,条件式 2,...,条件式 m)
          #  例 Sum(G[i,j],i,j,i>j,i!=0)
```

「添え字」のわたる範囲で「式」の和を取ります．条件式は，和を取る範囲を制限するものです．S-PLUS 組み込みのものとは違って，先頭が大文字（Sum）であることにご注意ください．S+NUOPT のオブジェクトに S-PLUS の sum 演算を適用することはできません．

2－7　定式の展開 System と最適化の実行 solve

定義したモデルを実際に解くには次の２ステップを行います．

```
sys <- System(Nlsfit)      # 数理計画モデルの展開
sol <- solve(sys)          # 展開したシステムを解く
```

System は数理計画モデルを定義した手続きを与え，データと連結して具体的な数理計画問題を作成し，戻り値として問題そのものを示す System オブジェクトを返します．System は一般に次のように記述します．

```
名前 <- System(モデルを記述した手続き)     # モデル定義に引数がない場合
名前 <- System(モデルを記述した手続き, 引数 1,引数 2,...,引数 3)
                                     # モデル定義に引数がある場合
```

上記で sys を表示させてみた結果は

```
> sys

esum<objective>: (8.79636*(v[lag quarterly revenue])+4.70997*(v[price in
dex])+5.8211*(v[income level])+12.9699*(v[market potential])+v0+(-8.7923
6))*(8.79636*(v[lag quarterly revenue])+4.70997*(v[price index])+5.8211*
(v[income level])+12.9699*(v[market potential])+v0+(-8.79236))+(8.79236*
(v[lag quarterly revenue])+
                              ...(中略)...
13.1664*(v[market potential])+v0+(-9.79424))*(9.77536*(v[la
g quarterly revenue])+4.27789*(v[price index])+6.2003*(v[income level])+
13.1664*(v[market potential])+v0+(-9.79424)) (minimize)
```

となり，式（この場合には目的関数）にデータが代入され，展開されていることがわかります．こうして生成した System オブジェクト solve に与えると，

```
> sol <- solve(sys)
NUOPT 11.1.9a (NLP/LP/IP/SDP module)
       <with META-HEURISTICS engine "wcsp"/"rcpsp">
      , Copyright (C) 1991-2009 Mathematical Systems Inc.
NUMBER_OF_VARIABLES                                        5
NUMBER_OF_FUNCTIONS                                        1
PROBLEM_TYPE                                    MINIMIZATION
METHOD                                      TRUST_REGION_IPM
<preprocess begin>..........<preprocess end>
<iteration begin>
    res=1.6e-001 ... 4.4e-013
<iteration end>
STATUS                                               OPTIMAL
```

```
VALUE_OF_OBJECTIVE                                        0.007374997682
ITERATION_COUNT                                                        4
FUNC_EVAL_COUNT                                                        8
FACTORIZATION_COUNT                                                    7
RESIDUAL                                                4.425550204e-013
ELAPSED_TIME(sec.)                                                  0.02
```

のように最適化の経過が出力されます．

```
名前 <- solve(System オブジェクト)            #  デフォルト
名前 <- solve(System オブジェクト,trace=F)    #  出力抑制
```

戻り値は最適化の結果を，S-PLUS のリスト型のオブジェクトとして示したものです．たとえばこの場合，戻り値のリスト型を表示すると次のようになります．

```
> sol
$variables:         # 変数値
$v:    # 変数 v
 income level lag quarterly revenue market potential price index
    0.7674609            0.1238646         1.330558  -0.7542401
attr(, "indexes"):
[1] "j"

$v0:  # 変数 v0
[1] -10.47261

$objective:  # 最適解における目的関数値
[1] 0.007374998

$counter:
 iters fevals vars
     3      6    5

$termination:
 tolerance      residual
  1.5e-006 1.138916e-009

$elapsed.time:
[1] 0.015

$errorCode:
[1] 0
```

特定の名前の変数値のみを取り出したいときには次のようにします．

```
sol$variables$v$current        # v の現在値（current）を表示
```

変数に関するサマリ情報を表示させるには次のようにします．

```
summary(sol)
```

```
> summary(sol)
$variables:
$v:
                            current lower upper    initial dual
         [income level]  0.7674609  -Inf   Inf  0.7674609    0
[lag quarterly revenue]  0.1238646  -Inf   Inf  0.1238646    0
     [market potential]  1.3305577  -Inf   Inf  1.3305577    0
          [price index] -0.7542401  -Inf   Inf -0.7542401    0

$v0:
   current lower upper   initial dual
 -10.47261  -Inf   Inf -10.47261    0
                    ... (以下は戻り値の表示と同様)...
```

次のようにすれば,S-PLUS 組み込みの lsfit の結果と S+NUOPT の結果が一致していることが確認できます.

```
> regfreeny <- lsfit(freeny.x, freeny.y)
> ls.print(regfreeny)
Residual Standard Error = 0.0147,  Multiple R-Square = 0.9981
N = 39,  F-statistic = 4354.254 on 4 and 34 df, p-value = 0

                          coef std.err  t.stat p.value
            Intercept -10.4726  6.0217 -1.7391  0.0911
lag quarterly revenue   0.1239  0.1424  0.8699  0.3904
          price index  -0.7542  0.1607 -4.6927  0.0000
         income level   0.7675  0.1339  5.7305  0.0000
     market potential   1.3306  0.5093  2.6126  0.0133
```

２－７－１. 数理計画モデルのデータを引数として，System から与える

手続き Nlsfit は，データ freeny.x および freeny.y をデータとして利用するように書かれていましたが，次のように，データの部分を引数として書くと，同一の形のデータならどのようなものでもデータとして利用することができるようになります.

```
# データを問わない一般の重回帰モデル
Nlsfit.gen <- function(x,y)  # x が説明変数 y が被説明変数
{
   S <- Set()
   Y <- Parameter(index=S,as.array(y))
   P <- Set()
   X <- Parameter(index=dprod(S,P),x)
   i <- Element(set=S)
   j <- Element(set=P)
   v <- Variable(index=j)
   v0 <- Variable()
   e <- Expression(index=i)
```

```
    e[i] ~ Sum(X[i,j]*v[j],j) + v0 - Y[i]
    esum <- Objective(type=minimize)
    esum ~ Sum(err[i]*err[i],i)
}
```

データの名前はSystemコマンドから引数として与えます．S-PLUSにビルトインされているstack.x,stack.loss（help(stack)でデータの説明が得られます）は，freeny.x,freeny.yと同様の説明変数と被説明変数の組です．同じNlsfit.genから二つのデータに対してシステムを生成して，解いてみましょう．

```
sys1 <- System(Nlsfit.gen,freeny.x,freeny.y) # freeny.x/y を与える
sol1 <- solve(sys1)
sys2 <- System(Nlsfit.gen,stack.x,stack.loss) # stack.x/loss を与える
sol2 <- solve(sys2)
```

同じモデルから二つの数理計画問題が生成され，解かれます．
次のようにするとそれぞれの結果を見ることができます．

```
> current(sys1,v) # freeny.x,freeny.y から生成した問題の結果
 income level lag quarterly revenue market potential price index
    0.7674609              0.1238646            1.330558  -0.7542401

> current(sys2,v) # stack.x,stack.loss から生成した問題の結果
 Acid Conc.   Air Flow Water Temp
 -0.1521225 0.7156402    1.295286
```

2-8　制約式 Constraint の追加

制約式を追加することで，パラメータの範囲を設定することができます．たとえば，簡単な制約としてすべてのフィッティングの係数の和が0であることを要求してみましょう．

制約　$\sum_{j \in P} v_j = 0$

を追加します．それには手続きNlsfit.genの末尾に

```
Sum(v[j],j) == 0
```

と加えて，Nlsfit.gen.eq0を生成し，解きます．一般に制約式は

```
式1  @ 式2              # 二項
式 1 @ 式 2 @ 式 3      # 三項

@ は { ==, =>, <= } のいずれか
```

のいずれかの形を取ります．< , > , != は制約式の定義に使うことはできません．

```
sys3 <- System(Nlsfit.eq,freeny.x,freeny.y)
```

```
sol3 <- solve(sys3,trace=F)   # 出力を抑制して解く
current(sys3,v)
```

とすると

```
 income level lag quarterly revenue market potential price index
    0.802618             0.3010901        -0.1054927  -0.9982154
```

という出力がなされます．制約が充足されているかどうか検証してみましょう．

```
sum(as.array(current(sys3,v)))  # as.array で array 型に変換して和を取る
```

とすると

```
 [1] 4e-006
```

が出力されて制約が充足されていることがわかります．このように重回帰問題も，数理計画問題として表現すれば，パラメータの値に任意の制約を加えることができます．

２−８−１. 制約式の名前付け，除去，復活（delete.con,restore.con）

S+NUOPT において，制約式は Constraint というオブジェクトに代入することで，名前をつけることができます．前項のモデル Nlsfit.eq と内容は同じですが，今度は Nlsfit.gen の末尾に

```
c1 <- Constraint()
c1 ~ Sum(v[j],j) == 0    # c1 という名前で制約式を代入
```

としたモデル Nlsfit.eq.named ならば，追加された制約式を"c1"という名前で参照することができます．具体的には

```
sys3 <- System(Nlsfit.eq.named,freeny.x,freeny.y)
sol3 <- solve(sys3,trace=F)   # suppress the output
current(sys3,c1)
```

と問い合わせると，最適解における制約式の値が

```
 [1] 4e-006
```

と返されます．
次が Constraint オブジェクトの宣言の形です

```
名前 <- Constraint()            #  添え字なし
名前 <- Constraint(index=添え字集合/要素)    #  添え字あり
```

宣言しただけの制約式（Constraint）オブジェクトは空の状態で，~（代入演算子）によって代入されてはじめて意味を持ちます．
名前を付けておくと，モデルを定義しなおすことなく，

```
delete.con(sys3,c1)   # 制約式 c1 を消去
restore.con(sys3,c1)  # 制約式 c1 を復活
```

とすると，名前の付いた制約式 c1 を消去したり復活させたりすることができます.
ここで

```
delete.con(sys3,c1)
sol4 <- solve(sys3,trace=F)
sum(as.array(current(sys3,v)))
```

とすると，

```
[1] 1.467643
```

が，再び

```
restore.con(sys3,c1)
sol4 <- solve(sys3,trace=F)
sum(as.array(current(sys3,v)))
```

とすると，

```
[1] 4.000003e-010
```

となることから効果を確かめることができます.
delete.con，restore.con の呼び出しの形は次のようになります.

```
delete.con(システムオブジェクト, 制約式の名前)  # 制約式の除去
restore.con(システムオブジェクト, 制約式の名前)  # 制約式の復活
```

2－9　整数変数 IntegerVariable

整数値しか取りえない変数（整数変数）は数理計画問題のモデル化において重要なコンポーネントです．ここでは，重回帰の問題を変形して，最もよくあてはまる説明変数を K 個選ぶようにしてみます.
定式化は次の通りです.

変数　v_j　（$j \in P$ ：線形関数の一次の係数の集合）

　　　v_0　（定数項）

　　　$\delta_j \in \{0,1\}$　（$j \in P$ ：説明変数 j を採用するかどうか）

最小化　$\sum_{i \in S} e_i^2$　（残差の二乗和）

定義　$e_i \equiv (v_0 + \sum_{j \in P} X_{i,j} v_j) - Y_i$　（$i \in S$ ：観測点）

　　　$X_{i,j}$ ：観測点 $i \in S$ における説明変数 $j \in P$ の値

　　　Y_i ：観測点 $i \in S$ における被説明変数の値

制約

$$-M \cdot \delta_j \le v_j \le M \cdot \delta_j \quad (j \in P)$$

：$\delta_j = 0$ならば$v_j = 0$になる．Mは十分大きな数と設定し，$\delta_j = 1$ならば実質制約なしにする．

$$\sum_{j \in P} \delta_j \le K$$

：変数はK個選ぶ

この定式は元のモデル記述に，０－１整数変数δ_jと制約を二つ追加すればよいことになります．記述は次のようになります．

```
# ０－１整数変数を使った説明変数選択モデル
NIsfit.int <- function(x,y,K)
{
   S <- Set()
   Y <- Parameter(index=S,as.array(y))
   P <- Set()
   X <- Parameter(index=dprod(S,P),x)
   i <- Element(set=S)
   j <- Element(set=P)
   v <- Variable(index=j)
   v0 <- Variable()
   e <- Expression(index=i)
   e[i] ~ Sum(X[i,j]*v[j],j) + v0 - Y[i]
   esum <- Objective(type=minimize)
   esum ~ Sum(e[i]*e[i],i)
   delta <- IntegerVariable(index=j,type=binary)
   -10*delta[j] <= v[j] <= 10*delta[j];
   Sum(delta[j],j) <= K
}
```

選択数を引数で与える

制約の追加部分 $M = 10$としている

整数変数（IntegerVariable）の宣言の形は次です．

```
名前 <- IntegerVariable() # 添え字なし
名前 <- IntegerVariable(index=添え字集合/要素) # 添え字付き
名前 <- IntegerVariable(type=binary) # 添え字なし,０－１整数変数
名前 <- IntegerVariable(index=添え字集合/要素,type=binary)
                                        # 添え字付き，０－１整数変数
```

$K = 2$として解いてみましょう．

```
sys <- System(NIsfit.int,freeny.x,freeny.y,2)
sol <- solve(sys)
current(sys,v)
```

以下のような結果が得られます．

```
 income level lag quarterly revenue market potential price index
   0.03609152              0.9792798                0            0
attr(, "indexes"):
[1] "j"
```

選択された説明変数

３．数理計画問題の記述（応用編）

ここでは，数理計画問題を記述する上でのより応用的な技法について述べます．

３−１　条件式と資金調達問題

次のような資金調達問題を考えます．現時点で現金で 1000 万円を持っているとします．市場には次のようなキャッシュフローをもたらす３種類の債券 A,B,C があるとし，いつでも購入できるとします．

```
> Cashflow.bf
     A    B    C
0 -100 -100 -100
1    3    5    2
2    3    5    2
3  110    5    2
4    0  105    2
5    0    0    2
6    0    0    2
7    0    0  130
```

いずれも額面 100 円ですが，満期とクーポンが違い，A は 3 年，B は 5 年，C は 7 年です．将来 10 年にわたって

```
> cash.flow
  1 2   3    4 5 6   7     8 9  10
  0 0 400 -200 0 0 400 -1000 0 600
```

のようなキャッシュフローが予測されるとき，手持ちの現金を常に 200 万円にしておき，10 年目の手持ちの現金を最大化する債券の購入計画が知りたいと思います．
この問題は次のような線形な数理計画問題として定式化できます．

変数　　$x_{b,t}$　（$b \in B, t \in T$ ：債券 b の t 時点における購入量）

　　　　y_t　（t 時点における現金）

最大化　y_{TL}　（最終時点における現金）

　　　　TL ：最終時点

制約　　$y_0 = P$　　← 初期時点の現金

$$y_t = y_{t-1} + \sum_{b \in B} \sum_{k, t-k \geq 1, k \leq L_b} BF_{k,b} x_{b,t-k} + C_t \ , \quad t \geq 1$$

（現金の時間発展）（予測されるキャッシュフロー：C_t）（購入した債券が t 時点にもたらすキャッシュフロー：$\sum BF_{k,b} x_{b,t-k}$）

$x_{b,t} \geq 0$，$b \in B, t \in T$ （投資額は非負）

$x_0 = 0$ （０時点では投資できない）

$x_{b,t} = 0, t + L_b + 1 > TL$ （満期が最終時点以降になるような債券は買わない）

$y_t \geq C_l$　　← 手持ちの現金の最小

このタイプの時系列を扱う問題ではよくあることですが，「現金の時間発展」「満期が最終

次元以降になるような債券は買わない」という制約式では式が定義される範囲を条件づけしています．また，「購入した債券が t 時点にもたらすキャッシュフロー」の表現では和を取る範囲を条件づけしています．このような場合のモデル記述では添え字に関する条件式を定義します．以下がこのモデルの記述です．

```
Cashflow <- function(flow,bf,P,CI)
{
    Time <- Set(0:length(flow))
    K <- Set()
    B <- Set()
    C <- Parameter(index=Time,as.array(flow))
    BF <- Parameter(index=dprod(K,B),bf)
    k <- Element(set=K);
    b <- Element(set=B);
    t <- Element(set=Time)
    len <- Parameter(index=b)
    len[b] ~ Sum(Parameter(1),k,BF[k,b]!=0) - 1
    TL <- length(flow)
    x <- Variable(index=dprod(b,t))
    y <- Variable(index=t)
    y[0] == P
    y[t,t>=1] == y[t-1] + Sum(x[b,t-k]*BF[k,b],b,k,t-k>=1,k<=len[b]) + C[t]
    x[b,0] == 0
    x[b,t,t+len[b]+1>TL] == 0
    x[b,t] >= 0
    y[t] >= CI
    f <- Objective(type=maximize)
    f ~ y[TL]
}
```

各債券の満期の定義

制約の添え字を記述する場所に要素に関する不等式を入れると，制約を定義する範囲が制限されます．条件式は式の中のいずれの部分の添え字に含めてもかまいません．

```
y[t,t>=1] == y[t-1] +...
```

という記述では t>=1 という部分が条件式で，この式全体が定義する範囲が t>=1 である部分であることを示しています．

```
x[b,t,t+len[b]+1>TL] == 0  # 満期が TL よりやってくる債券は購入しない
```

という記述では t+len[b]+1>TL という部分が条件式です．条件式はこのように添え字を含む定数の表現である場合にも機能します．
また，制約式の定義には >=, <=, == のみしか利用できませんでしたが，条件式の定義には

```
!=      （等しくない）
<       （より大きい）
>       （より小さい）
```

を利用することができます.
条件式が Sum 関数に現れると和を取る範囲を制限する働きをします. 例としては

```
len[b] ~ Sum(Parameter(1),k,BF[k,b]!=0) - 1  # 各債券の満期の計算
```

と

```
Sum(x[b,t-k]*BF[k,b],b,k,t-k>=1,k<=len[b])
                    # 過去に購入した各債券が t 時点にもたらすキャッシュフロー
```

に表れています. 条件式が複雑なので意図通りに式が定義されているかを知るには, 表示の量は多くなりますが,

```
sys <- System(Cashflow,Cashflow.flow,Cashflow.bf,1000,200)
```

と展開したのち, sys を表示してみることをお勧めします.

```
> sys
1-1 :  y[0] == 1000

2-1 :  -y[1]+y[0]-100*x[A,1]-100*x[B,1]-100*x[C,1] == 0
2-2 :  -y[2]+y[1]-100*x[A,2]-100*x[B,2]-100*x[C,2]+3*x[A,1]+5*x[B,1]+2*x
[C,1] == 0
2-3 :  -y[3]+y[2]-100*x[A,3]-100*x[B,3]-100*x[C,3]+3*x[A,2]+5*x[B,2]+2*x
[C,2]+3*x[A,1]+5*x[B,1]+2*x[C,1]+400 == 0
                    ..(中略)..
6-7 :  y[6] >= 200
6-8 :  y[7] >= 200
6-9 :  y[8] >= 200
6-10 :  y[9] >= 200
6-11 :  y[10] >= 200

f<objective>: y[10] (maximize)
```

次のようにすると投資計画と現金の推移をみることができます.

```
sol <- solve(sys)
round(as.array(current(sys,x),digits=2)
round(as.array(current(sys,y),digits=2)
```

次のように結果が出力されます.

```
> round(as.array(current(sys,x),digits=2) # 投資計画
  0    1    2    3 4    5   6 7 8 9 10
A 0 1.52 0.00 4.25 0 0.00 4.9 0 0 0  0
B 0 2.37 0.00 0.00 0 2.71 0.0 0 0 0  0
C 0 4.11 0.25 0.00 0 0.00 0.0 0 0 0  0
```

```
attr(, "indexes"):
[1] "b" "t"

> round(as.array(current(sys,y)),digits=2) # 現金の推移
    0   1   2   3   4   5   6      7   8       9      10
 1000 200 200 200 200 200 200 636.95 200 1055.28 1655.28
attr(, "indexes"):
```

３－２ SymmetricMatrix と最小固有値問題

S+NUOPT は要素が一般の式からなる行列の半正定値性を制約として与えることができます．簡単な例として次のような問題を考えましょう．

変数	λ
最大化	λ
制約	$A-\lambda I$ が半正定値
	A ：任意の対称行列

この問題を定義するには，SymmetricMatrix というオブジェクトを使って次のように記述します．

```
MinLambda <- function(A)
{
    S <- Set()
    i <- Element(set=S)
    j <- Element(set=S)
    A <- Parameter(index=dprod(i,j),A)
    lambda <- Variable()
    M <- SymmetricMatrix(dprod(i,j))
    M[i,j,i>j] ~ A[i,j]
    M[i,i] ~ A[i,i] - lambda
    M >= 0
    f <- Objective(type=maximize)
    f ~ lambda
}
```


SymmetricMatrix は対称行列に対応するオブジェクトで，実体は行列の形に並んだ Expression の集合体です．SymmetricMatrix の宣言の形式は次です．

```
名前 <- SymmetricMatrix(dprod(集合, 集合))  # 同じ集合を与える
名前 <- SymmetricMatrix(dprod(要素 1, 要素 2))
                                  # 要素 1 と要素 2 は同じ集合の要素
```

この例では M というオブジェクトが SymmetricMatrix として定義されています．宣言では同一の集合 S の要素 i と j が与えられています．index=右辺には dprod(S,S)と書くこともできます．続いて，~ 演算子を用いて要素の設定を行っていますが方法は Expression と全く同様です．対称行列であることは定義から保証されているので，下三角あるいは上三角部分の要素いずれか一方のみを与えています．

解いてみます．サンプル行列として S-PLUS 付属のデータセット air の分散共分散行列を使います.

```
mat <- var(air)
sys <- System(MinLambda,mat)
sol <- solve(sys,trace=F)
current(sys,lambda)
```

次のように出力されます.

```
[1] 0.2510515
```

この値は S-PLUS を使って求めた固有値の最小値と正確に一致しています.

```
> eigen(mat)$values
[1] 8317.0294527   86.5362664    9.2065573    0.2510515
```

３－３　wcsp と集合分割問題

３-３-１.wcsp による１指標の２分割問題

S-PLUS には napsack という関数があり，ベクトルのコンポーネントの部分集合で，和が与えられた値に近いものを求めることができます．次は napsack のマニュアルに載っているサンプルです．S-PLUS にビルトインされているアメリカの州の面積をちょうど半分にするように部分集合を選びます.

```
state.area <- state.x77[,"Area"]  # 面積のベクトルを取る
state.half <- napsack(state.area, sum(state.area)/2) # napsack を起動
```

napsack の戻り値には解(部分集合の選択を示す真偽値のベクトル)が複数返されますので，次のようにして解の一つを取得します.

```
x1 <- state.half[,1]  # 解の一つめを取得する
sum(state.area[x1])   # 選ばれたもので面積の和
[1] 1768397
sum(state.area[!x1])  # 選ばれなかったもので面積の和
[1] 1768397
```

選ばれたものと選ばれなかったものの和が正確に一致していますので最適な答えが返されています．この問題は次のような離散的な数理計画問題として表現できます.

変数　　$\delta_s \in \{0,1\}$　　（$s \in S$ ：部分集合に入るか否か）

制約　　$\sum_{s \in S} a_s \cdot \delta_s = \sum_{s \in S} a_s \cdot (1-\delta_s)$

a_s ：州 $s \in S$ の面積

この問題には目的関数がありませんので，厳密には制約充足問題ということになります．この問題を S+NUOPT で解くために，次のモデル Half を用意します.

```
Half <- function()
```

```
    {
        S <- Set()
        Cat <- Set()
        Data <- Parameter(index=dprod(S,Cat),state.x77)
        delta <- IntegerVariable(index=S,type=binary)
        s <- Element(set=S)
        Sum(Data[s,"Area"]*delta[s],s) == Sum(Data[s,"Area"]*(1-delta[s]),s)
    }
```

この問題は次のようにして解くことができます.

```
sys <- System(Half)   # 展開
sol <- solve(sys)     # 解く
```

ほぼ瞬時に応えが返されます. 内容を検証してみます.

```
delta <- as.array(current(sys,delta))
sum(state.x77[,"Area"][delta==1])
sum(state.x77[,"Area"][delta==0])
```

S+NUOPT も解を求めることができたことがわかります.

```
> sum(state.x77[,"Area"][delta==1])
[1] 1768397
> sum(state.x77[,"Area"][delta==0])
[1] 1768397
```

3-3-2.wcsp による３指標の２分割問題

面積のみならず，別の指標でもおおむね半分となるように二分割する方法を探してみることにします. 元にしたデータは state.x77)（下図参照）とし，ここに収められている３つの統計値（Area,Population,Illiteracy）を指標としてすべて半分になるように分割してみます.

state.x77

		1 Population	2 Income	3 Illiteracy	4 Life.Exp	5 Murder	6 HS.Grad	7 Frost	8 Area
1	Alabama	3615.00	3624.00	2.10	69.05	15.10	41.30	20.00	50708.0
2	Alaska	365.00	6315.00	1.50	69.31	11.30	66.70	152.00	566432.0
3	Arizona	2212.00	4530.00	1.80	70.55	7.80	58.10	15.00	113417.0
4	Arkansas	2110.00	3378.00	1.90	70.66	10.10	39.90	65.00	51945.0
5	California	21198.00	5114.00	1.10	71.71	10.30	62.60	20.00	156361.0
6	Colorado	2541.00	4884.00	0.70	72.06	6.80	63.90	166.00	103766.0
7	Connecticut	3100.00	5348.00	1.10	72.48	3.10	56.00	139.00	4862.0
8	Delaware	579.00	4809.00	0.90	70.06	6.20	54.60	103.00	1982.0
9	Florida	8277.00	4815.00	1.30	70.66	10.70	52.60	11.00	54090.0
10	Georgia	4931.00	4091.00	2.00	68.54	13.90	40.60	60.00	58073.0
11	Hawaii	868.00	4963.00	1.90	73.60	6.20	61.90	0.00	6425.0
12	Idaho	813.00	4119.00	0.60	71.87	5.30	59.50	126.00	82677.0
13	Illinois	11197.00	5107.00	0.90	70.14	10.30	52.60	127.00	55748.0
14	Indiana	5313.00	4458.00	0.70	70.88	7.10	52.90	122.00	36097.0
15	Iowa	2861.00	4628.00	0.50	72.56	2.30	59.00	140.00	55941.0

. . .

前項の最適化モデルを若干拡張して次のようにします.

変数　$\delta_s \in \{0,1\}$　（$s \in S$ ：集合に入るか否か）

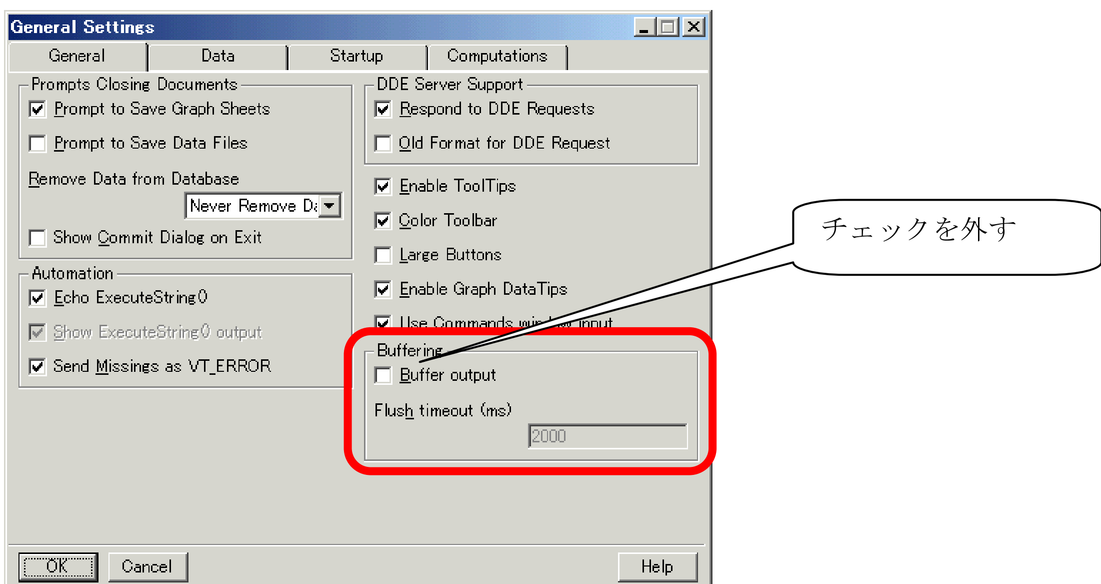
制約　$\sum_{s \in S} f_s^i \cdot \delta_s = \sum_{s \in S} f_s^i \cdot (1-\delta_s),\ i \in F$

f_s^i ：指標値（統計値）　$i \in F$ of state $s \in S$

次のモデルを定義します.

```
Half2.infs <- function()
{
    S <- Set()
    Cat <- Set()
    Data <- Parameter(index=dprod(S,Cat),state.x77)
    delta <- IntegerVariable(index=S,type=binary)
    s <- Element(set=S)
    f <- Objective(type=minimize)
    Sum(Data[s,"Area"]*delta[s],s) == Sum(Data[s,"Area"]*(1-delta[s]),s)
    Sum(Data[s,"Population"]*delta[s],s) ==
                Sum(Data[s,"Population"]*(1-delta[s]),s)
    Sum(Data[s,"Illiteracy"]*delta[s],s)                          ==
Sum(Data[s,"Illiteracy"]*(1-delta[s]),s)
}
```

計算を始める前に，S-PLUS のオプションウインドウから出力のバッファを行わない設定としておいてください．これは計算の経過を見るためです.



制限時間５秒として実行してみましょう.

```
sys <- System(Half2.infs)
nuopt.options(maxtim=5)
sol <- solve(sys)
```

解は得られない，と出ます．

```
<<NUOPT 22>> B & B itr. timeout (no feas.sol.).
```

これは，３指標を同時にちょうど半分にするような解が存在しないためです．NUOPT がその判定自体にかなり時間を所要しているため，このような出力が出ます．
しかし，この結果は我々の望んだものではありません．厳密解が得られなければ，近似解を求めてほしいのが普通です．そんな場合に，解法 wcsp が役立ちます．wcsp は離散最適化問題を近似的に解きます．
次が wcsp を適用するための，３指標２分割問題の記述です．

softConstraint(1)は wcsp に，この制約はソフト制約として扱う（ハード制約と呼ばれる厳密な制約ではなく，最小化する対象として扱う）ように指示する記述です．引数の値１はソフト制約の違反量そのものに掛ける重みであり，複数のソフト制約があった場合には，重みが大きく設定されたソフト制約が優先されることになります．

最後の制約式の両辺を 1000 倍していますが，これは **wcsp が制約の違反量の計算を常に整数値で行う**ためです．他の指標とオーダーをそろえて整数値にするために，1000 倍しています．

```
Half2 <- function()
{
    S <- Set()
    Cat <- Set()
    Data <- Parameter(index=dprod(S,Cat),state.x77)
    delta <- IntegerVariable(index=S,type=binary)
    s <- Element(set=S)
    softConstraint(1)
    Sum(Data[s,"Area"]*delta[s],s) == Sum(Data[s,"Area"]*(1-delta[s]),s)
    Sum(Data[s,"Population"]*delta[s],s)                             ==
Sum(Data[s,"Population"]*(1-delta[s]),s)
    1000*Sum(Data[s,"Illiteracy"]*delta[s],s)                        ==
1000*Sum(Data[s,"Illiteracy"]*(1-delta[s]),s)
}

sys2 <- System(Half2)
nuopt.options(method="wcsp")
sol2 <- solve(sys2)
nuopt.options(nuopt.options(NA))     # パラメータをデフォルトに戻す
```

次のような実行経過が現れます．

```
NUOPT 11.1.9a (NLP/LP/IP/SDP module)
       <with META-HEURISTICS engine "wcsp"/"rcpsp">
```

```
      , Copyright (C) 1991-2009 Mathematical Systems Inc.
wcspMain: Invoke wcsp with NO TIME LIMIT. Set default maxtim = 5 .
wcspMain: CONSIDER SETTING nuopt.options(maxtim=<appropriate time>)
NUMBER_OF_VARIABLES                                        50
(#INTEGER/DISCRETE)                                        50
NUMBER_OF_FUNCTIONS                                         3
PROBLEM_TYPE                                     MINIMIZATION
METHOD                                                   WCSP
<preprocess begin>...<preprocess end>
preprocessing time= 0(s)
<iteration begin>
# random seed = 1
(hard/soft) penalty= 0/986245, time= 0(s)
<greedyupdate begin>.........<greedyupdate end>
greedyupdate time= 0(s)
--- End Phase-I iteration ---
(hard/soft) penalty= 0/61443, time= 0.016(s), iteration= 1
(hard/soft) penalty= 0/60119, time= 0.016(s), iteration= 2
(hard/soft) penalty= 0/39847, time= 0.031(s), iteration= 15
(hard/soft) penalty= 0/34401, time= 0.031(s), iteration= 16
(hard/soft) penalty= 0/31565, time= 0.031(s), iteration= 17
(hard/soft) penalty= 0/16107, time= 0.031(s), iteration= 231
(hard/soft) penalty= 0/13653, time= 0.047(s), iteration= 233
(hard/soft) penalty= 0/6051, time= 0.047(s), iteration= 235
(hard/soft) penalty= 0/4421, time= 0.047(s), iteration= 238
(hard/soft) penalty= 0/4203, time= 0.063(s), iteration= 626
(hard/soft) penalty= 0/3833, time= 0.063(s), iteration= 627
(hard/soft) penalty= 0/2727, time= 0.063(s), iteration= 628
(hard/soft) penalty= 0/1473, time= 0.063(s), iteration= 690
(hard/soft) penalty= 0/1073, time= 0.078(s), iteration= 899
(hard/soft) penalty= 0/991, time= 0.141(s), iteration= 3850
# (hard/soft) penalty= 0/991
# cpu time = 0.141/5.016(s)
# iteration = 3850/292808
<iteration end>
STATUS                                                 NORMAL
TERMINATE_REASON                          End_by_CPU_time_limit
ITERATION_COUNT                                        292808
PENALTY                                                   991
RANDOM_SEED                                                 1
ELAPSED_TIME(sec.)                                       5.03
> nuopt.options(nuopt.options(NA))
```

penalty=…という出力が探索で良い解が見つかるたびに更新されてゆきます．デフォルトでは５秒探索を行って止まります．このケースでは 991 というペナルティ値を持つ解が得られています．結果を検証してみると，Area 以外にも，Population,Illiteracy でほぼ半分になっていることがわかります．

```
delta2 <- as.array(current(sys2,delta))
sum(state.x77[,"Area"][delta2==1])
sum(state.x77[,"Area"][delta2==0])
sum(state.x77[,"Population"][delta2==1])
sum(state.x77[,"Population"][delta2==0])
sum(state.x77[,"Illiteracy"][delta2==1])
sum(state.x77[,"Illiteracy"][delta2==0])
```

出力は次のようになります．
このアルゴリズムは解の良さを判定して停止するというロジックを持ちません．そのため，５秒という探索時間の上限が設定されています．実行経過の出力によると，このケースでは最良の解が１秒以下で求められています．

```
(hard/soft) penalty= 0/991, time= 0.125(s), iteration= 3850
```

この結果を見て制限時間を調整することができます．通常ペナルティは最初急速に減り，減少が緩やかになり，止まるという経過をたどります．もし制限時間が来ても更新が行われているようであれば，次のように長めの実行時間を設定してください．

```
options.method(method=" wcsp",maxtim=60)  # 60 秒制限
```

アルゴリズム wcsp は厳密な解を求めることができませんが，多くの離散計画問題に対し，かなりの短時間（１分以下）で比較的有効な実効可能解を与えます．制限時間を延長しても，結果に満足できない場合には乱数の種の変更（方法は 3-3-4 節で述べます）を検討してください．
ただ，wcsp は離散計画問題のみしか扱えない，汎用性の点では劣る解法なので，通常の簡単な数理計画問題を解く場合にエラーを発生させて混乱しないよう，wcsp を指定した後にはアルゴリズムの指定は

```
nuopt.options(nuopt.options(NA))    # デフォルトに戻す
```

としてデフォルト（自動選択）に戻すことをお勧めします．

３-３-３.wcsp による多分割問題（selection とハード，セミハード，ソフト制約）

次は２分割ではなく，面積を指標としたＮ分割を行う問題の記述です．

変数　$u_{s,k} \in \{0,1\}$　（$s \in S$，$k \in K$ ：s をいずれのグループ k に振り分けるか）

ハード制約

$$\sum_{k \in K} u_{s,k} = 1,\quad s \in S$$

（振り分けられるのはいずれか一つ）

ソフト制約

$$\sum_{s \in S} a_s \cdot u_{s,k} = \frac{1}{|K|}\sum_{s \in S} a_s$$

a_s ：州 $s \in S$ の面積

モデル記述は以下です．

```
Partition <- function(N)
{
    S <- Set()
    Cat <- Set()
    Data <- Parameter(index=dprod(S,Cat),state.x77)
    K <- Set(1:N)
    u <- IntegerVariable(index=dprod(S,K),type=binary)
    k <- Element(set=K)
    s <- Element(set=S)
    hardConstraint()
    selection(u[s,k],k)
    softConstraint(1)
    Sum(Data[s,"Area"]*u[s,k],s) == Sum(Data[s,"Area"],s)/N
    dst <- Expression(index=s)
    knum <- Parameter(index=k)
    knum[k] ~ k
    dst[s] ~ Sum(knum[k]*u[s,k],k)
}
```

次の制約がハード制約であることの定義

Sum(u[s,k],k) == 1
と等価だがこの方が wcsp はより高速に動作する

各 s が何番目のグループに分類されたかという指標

hardConstraint()は以降に定義される制約式が最も優先される，ハード制約であることを示します．ハード制約とソフト制約の中間にセミ・ハード制約が定義されています．すべて以降に引き続く制約がその種別であることを示します．形式は次の通りです．

```
softConstraint(1 より大きな整数値)   # ソフト制約の定義のはじまり
semiHardConstraint()  # セミハード制約の定義のはじまり
hardConstraint()    # ハード制約の定義のはじまり
```

これらの構文は何度でもコールすることができ，常に直前のコールが有効です．
selection()構文は，０－１変数の和が１であるという定式化の場合に使うことができ，wcsp アルゴリズムの動作を高速化する働きがあります．
では，wcsp で５分割を解いてみます．

```
sys <- System(Partition,5)
nuopt.options(method="wcsp",maxtim=1,wcspTryCount=1)
sol <- solve(sys)
```

式（Expression）dst を取り出して，和を計算してみます．

```
dst <- as.array(current(sys,dst))
for(i in 1:5) print(sum(state.area[dst==i]))
```

```
> for(i in 1:5) print(sum(state.area[dst==i]))
[1] 709978
[1] 705870
[1] 706540
[1] 707348
[1] 707058
```

比較的均等にわかれていることがわかります．各グループを表示させてみます．

```
for(i in 1:5) print(state.area[dst==i])
```

```
> for(i in 1:5) print(state.area[dst==i])
 Alaska Colorado Virginia
 566432   103766    39780
 Alabama California Florida Hawaii Indiana Kentucky Mississippi
   50708      156361   54090   6425   36097    39650       47296

 New Jersey New Mexico North Carolina Oregon Tennessee
       7521      121412           48798  96184      41328
 Delaware Idaho Illinois  Iowa Kansas Maine Michigan  Texas
     1982 82677    55748 55941  81787 30920    56817 262134

 West Virginia Wisconsin
         24070     54464
 Arizona Georgia Massachusetts Nebraska New York  Ohio Oklahoma
  113417   58073          7826    76483    47831 40975    68782

 Pennsylvania South Dakota Vermont Washington Wyoming
        44966        75955    9267      66570   97203
 Arkansas Connecticut Louisiana Maryland Minnesota Missouri Montana
    51945        4862     44930     9891     79289    68995  145587

 Nevada New Hampshire North Dakota Rhode Island South Carolina  Utah
 109889          9027        69273         1049          30225 82096
```

３-３-４.乱数の種の設定による解の違い

wcsp は乱数を利用して解を発見しますので，乱数のシード値によって結果が異なります．前項の Partition モデルにおいて，乱数のシード値（デフォルトで１）を 4 に変更してみます．

```
nuopt.options(method="wcsp",maxtim=1,wcspRandomSeed=4)
sol <- solve(sys)
dst <- as.array(current(sys,dst))
for(i in 1:5) print(sum(state.area[dst==i]))
```

別の答えが得られます．

```
> for(i in 1:5) print(sum(state.area[dst==i]))
[1] 707621
[1] 709041
[1] 706960
[1] 706429
[1] 706743
```

S+NUOPT にはシード値を変更して自動的に最も良いものを求める機能があります．

wcspTryCount というパラメータを設定すると，wcspRandomSeed で設定した値から，この回数だけシード値を１刻みで設定しなおして最も良いものを出力します.

```
nuopt.options(method="wcsp",maxtim=1,wcspRandomSeed=1,wcspTryCount=5)
```

３-３-５.アルゴリズム wcsp を使うときの注意点

アルゴリズム wcsp は S+NUOPT に実装されているほかのアルゴリズムとはかなり違う特徴を持っています. 利用上の注意点を以下にまとめておきます.

- 0-1 整数変数しか扱うことができません
- 起動したら，nuopt.options(nuopt.options(NA))で元にもどしましょう
- 真の最適解が得られる保証はありません
- nuopt.options(maxtim=..)で停止時間（デフォルトは５秒）をチューニングできます
- 制約のクラスが（ソフト/セミハード/ハード）の３つあります
- 与える乱数のシード値 nuopt.options(wcspRandomSeed=..)によって違う解を返します
- nuopt.options(wcspTryCount=...)で回数を与えればその回数だけ乱数のシード値を振って試行し，最良の結果を返します.
- 制約式の違反量は整数値に丸めて判定されるので制約式の定数倍が必要な場合があります.

４．S-PLUS とのデータ連携

S-PLUS のデータ（vector, matrix, array, list）は S+NUOPT のモデルを記述するオブジェクト（主に Set や Parameter）の初期化に用いることができます．また S+NUOPT による最適化結果（主に Variable や Expression）を S-PLUS のデータとして取り出すこともできます．この章ではその方法について，例を通じて具体的に述べます．
データが意図通り変換されているかどうかは，コマンドプロンプトから S+NUOPT のオブジェクトを実際に作って，確認しながら行うのが最も確実で簡便です．以降の説明では S+NUOPT がロードされているとして実行を行います．S+NUOPT をロードするには

```
> module(nuopt)
```

のようにします．

４－１　vector 型で定数オブジェクト（Parameter）を初期化する

S-PLUS の最も基本的なデータ型は vector 型です．S-PLUS はスカラーとベクトルを明確に区別せず，両方を vector 型で表現することができます．そのため，添え字を持つ Parameter を vector で初期化しようとする場合には，array 型に変換する必要があります．

４-１-１.添え字を持たない Parameter を初期化する

次はスカラー値を与えて S+NUOPT の定数オブジェクト（Parameter）クラスのオブジェクト p,q を宣言し初期化する例です.作ってみてから内容を表示させてみて確認しています．

```
p <- Parameter(3)        # 値を与えて初期化
p                         # 確認
b <- 3^5                  # b は S-PLUS のオブジェクト
q <- Parameter(b)        # 初期化
q                        # 値の確認
```

p,q が

```
> p
[1] 3
> q
[1] 243
```

と表示されることを確認してください．

４-１-２.添え字を持つ Parameter を初期化する

S-PLUS ではスカラーとベクトルが厳密に区別されません（スカラー値は長さ（length 属性）が１の特殊な vector 型のオブジェクトです）．一方 S+NUOPT ではスカラーとベクトルを添え字の数によって厳密に区別します．そのため，vector 型をベクトル値として S+NUOPT に渡す場合には as.array 関数を使って，array 型に変更する必要があります．
次はベクトルデータを S-PLUS に渡す例です．
S-PLUS にビルトインされているサンプルデータ freeny.y を vector 型にしたものを ydata としましょう（freeny.y は重回帰を行う S-PLUS 関数 lsfit の説明の例に用いられます）．
次のような時系列の添え字を持つデータ（一年が 1Q～4Q で４分割されており，1962 年 2Q から 1971 年の 4Q までの 39 個のデータです．

```
> freeny.y
           1Q      2Q      3Q      4Q
1962:         8.79236 8.79137 8.81486
1963: 8.81301 8.90751 8.93673 8.96161
1964: 8.96044 9.00868 9.03049 9.06906
1965: 9.05871 9.10698 9.12685 9.17096
1966: 9.18665 9.23823 9.26487 9.28436
1967: 9.31378 9.35025 9.35835 9.39767
1968: 9.42150 9.44223 9.48721 9.52374
1969: 9.53980 9.58123 9.60048 9.64496
1970: 9.64390 9.69405 9.69958 9.68683
1971: 9.71774 9.74924 9.77536 9.79424
```

まずこれを S+NUOPT で扱うため，通常のベクトル型に変換します．

```
ydata <- as.vector(freeny.y)
ydata
length(ydata)
```

次のように変換されたことを確認します．

```
> ydata
 [1] 8.79236 8.79137 8.81486 8.81301 8.90751 8.93673 8.96161 8.96044 9.00868
9.03049 9.06906
[12] 9.05871 9.10698 9.12685 9.17096 9.18665 9.23823 9.26487 9.28436 9.31378
9.35025 9.35835
[23] 9.39767 9.42150 9.44223 9.48721 9.52374 9.53980 9.58123 9.60048 9.64496
9.64390 9.69405
[34] 9.69958 9.68683 9.71774 9.74924 9.77536 9.79424
> length(ydata)
[1] 39
```

次のようにして，このデータに等しい S+NUOPT の定数オブジェクト（Parameter）Y を作ります．

```
S <- Set()   # 添え字を入れる S+NUOPT の集合オブジェクトを準備
Y <- Parameter(index=S,as.array(ydata))  # S を添え字とする Y という定数
オブジェクトを, as.array(ydata)で初期化する
Y
S
```

まず，Y に先だって，Y の添え字の入れ場所である集合オブジェクト S を定義しています．S+NUOPT ではかならず添え字が何なのか，ということをオブジェクトの宣言のときに意識する必要があります．いざ Y を作るときには，ydata をそのまま渡すのではなく，array 型に変換して，as.array(ydata)を渡しているのに注意してください．
Y の内容が次のように表示されることを確認します．

```
> Y
      1       2       3       4       5       6       7       8       9
10      11      12
 8.79236 8.79137 8.81486 8.81301 8.90751 8.93673 8.96161 8.96044 9.00868 9.03049
9.06906 9.05871

      13      14      15      16      17      18      19      20      21
22      23      24
 9.10698 9.12685 9.17096 9.18665 9.23823 9.26487 9.28436 9.31378 9.35025 9.35835
9.39767 9.4215

      25      26      27      28      29      30      31      32      33
34      35      36
 9.44223 9.48721 9.52374 9.5398 9.58123 9.60048 9.64496 9.6439 9.69405 9.69958
9.68683 9.71774

      37      38      39
 9.74924 9.77536 9.79424
attr(, "indexes"):
[1] "*"
```

添え字（1,2,3,...）と内容（8.79236,8.79137,8.81486,...）が一緒に表示されますのでちょっと長いですが，Y に ydata と同じ内容が入っているかが確認できます．Y を定義する際に，添え字の入れ場所として（index=S）与えた S も同時に定義されます．S の内容は次のように表示されます．

```
> S
{ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 }
```

このデータが 1,2,3,...,39 という添え字を持つことが確認できます．
as.array で array に変換すればどのようなベクトルデータでも受け渡すことが可能です．
次は seq(1,10,3)で定義されたベクトルデータ s で初期化された定数オブジェクト Parameter を作っています．

```
s <- seq(1,10,3)          # s は S-PLUS オブジェクト
D <- Set()                # D は r の添え字の入れ場所
r <- Parameter(index=D,as.array(s))     # r を作る
r
D
```

次のように r と D が出力されます．

```
> r
 1 2 3  4
 1 4 7 10
attr(, "indexes"):
[1] "*"
```

```
> D
{ 1 2 3 4 }
```

4-1-3.整数列以外を添え字として持つ Parameter を初期化する

これまでの例ではベクトルの添え字は 1,2,3,.. という整数の集合でしたが，S+NUOPT の定数オブジェクト（Parameter）を作るときに，元になる vector 型データの names 属性として，任意の添え字の列を与えることができます．次では s の要素に a,b,c,d という names 属性を与えています．

```
s <- seq(1,10,3)                        # s は S-PLUS オブジェクト
names(s) <- c("a","b","c","d") # names 属性として a,b,c,d を定義
D <- Set()
r <- Parameter(index=D,as.array(s))     # r を作る
s
r
D
```

結果として定数オブジェクト r の添え字は a,b,c,d であるとみなされています．

```
> s
 a b c  d
 1 4 7 10
> r
 a b c  d
 1 4 7 10
attr(, "indexes"):
[1] "*"
> D
{ a b c d }
```

添え字には任意の文字列を与えることができます．

4－2　matrix 型で Parmeter を初期化する

S-PLUS の matrix 型は array 型の一種であるため，S+NUOPT のオブジェクトに直接渡すことができます．やはり S-PLUS にビルトインされている説明変数のデータ freeny.x は次のような matrix 型のオブジェクトです．

```
> freeny.x
numeric matrix: 39 rows, 4 columns.
      lag quarterly revenue price index income level market potential
  [1,]                8.79636     4.70997      5.82110          12.9699
  [2,]                8.79236     4.70217      5.82558          12.9733
  [3,]                8.79137     4.68944      5.83112          12.9774
  [4,]                8.81486     4.68558      5.84046          12.9806
  [5,]                8.81301     4.64019      5.85036          12.9831
  [6,]                8.90751     4.62553      5.86464          12.9854
  [7,]                8.93673     4.61991      5.87769          12.9900
```

```
                    ...（中略）...
[35,]              9.69958    4.30909    6.16135          13.1520
[36,]              9.68683    4.30552    6.18231          13.1593
[37,]              9.71774    4.29627    6.18768          13.1579
[38,]              9.74924    4.27839    6.19377          13.1625
[39,]              9.77536    4.27789    6.20030          13.1664
>
attr(, "indexes"):
[1] "*"
```

このデータは matrix 型であるため，そのまま S+NUOPT の定数データの初期化に使うことができます．次では A という S+NUOPT の定数データを初期化しています．添え字の入れ場所として S,P を定義しています．

```
S <- Set()      # 行方向の添え字の入れ場所
P <- Set()      # 列方向の添え字の入れ場所
A <- Parameter(index=dprod(S,P),freeny.x)
S
P
A
```

S,P の内容は次のように出力されます．

```
> S
{ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 2
7 28 29 30 31 32 33 34 35 36 37 38 39 }
> P
{ lag quarterly revenue price index income level market potential }
```

これは元になる freeny.x の行，列の添え字に対応しています．行，列の添え字を出力させてみましょう．

```
rownames(freeny.x)              # 行の添え字
colnames(freeny.x)              # 列の添え字
```

次のようになります．

```
> rownames(freeny.x)         # 行の添え字
NULL
> colnames(freeny.x)         # 列の添え字
[1] "lag quarterly revenue" "price index"
[3] "income level"          "market potential"
```

行の添え字は定義されていないので，NUOPT は 1,2,3,...,39 であるとみなしています．
A の内容を見てみましょう．

```
> A
   income level lag quarterly revenue market potential price index
```

```
 1       5.82110            8.79636          12.9699     4.70997
 2       5.82558            8.79236          12.9733     4.70217
 3       5.83112            8.79137          12.9774     4.68944
 4       5.84046            8.81486          12.9806     4.68558
 5       5.85036            8.81301          12.9831     4.64019
                          (中略)
37       6.18768            9.71774          13.1579     4.29627
38       6.19377            9.74924          13.1625     4.27839
39       6.20030            9.77536          13.1664     4.27789
attr(, "indexes"):
[1] "*" "*"
```

４－３　list 型で定数オブジェクト Parameter を初期化する

これまで array 型から Parameter 型への変換を見てきましたが，list 型も次のような規則で Parameter 型に変換することができます．list の最初から length(list)-1 個の要素が添え字のベクトル，最後の要素が値のベクトルとみなします．たとえば次は一次元のオブジェクトを定義します．

```
list1 <- list(c("a","b","c"),1:3)
S <- Set()
a <- Parameter(index=S,list1)
a
```

a の表示は次のようになります．

```
> a
 a b c
 1 2 3
attr(, "indexes"):
[1] "*"
```

二つの添え字を持つオブジェクトは

```
list2 <- list(c("a","a","b","b"),c(1,2,1,2),7:10)
S1 <- Set()
S2 <- Set()
b <- Parameter(index=dprod(S1,S2),list2)
b
```

b の表示は次のようになります．

```
> b
  1  2
a 7  8
b 9 10
attr(, "indexes"):
[1] "*" "*"
```

次は３次元の例です.

```
list3        <-        list(c(1,1,1,2,2,2),c("a","a","b","b","c","c"),
c("A","A","A","B","B","B"),1:6)
S1 <- Set()
S2 <- Set()
S3 <- Set()
d <- Parameter(index=dprod(S1,S2,S3),list3)
d
```

d の表示は次のようになります.

```
> d
, , A
  a b c
1 2 3 0
2 0 0 0

, , B
  a b c
1 0 0 0
2 0 4 6
attr(, "indexes"):
[1] "*" "*" "*"
```

４－４　最適化の結果（Variable/Expression）を取り出す

前項までのようにして定義した Parameter を使って，数理計画モデルを例えば次のように定義することができます．次の Nlsfit はデータの初期化を含んでいます．

```
Nlsfit <- function()
{
#  データの宣言および初期化
   ydata <- as.vector(freeny.y)   #  ydata を vector である ydata に
   S <- Set()                      #  Y の添え字の入れ場所として S を定義
   Y <- Parameter(index=S,as.array(ydata))  #  被説明変数 Y を作る
   P <- Set()              #  X の二つ目の添え字の入れ場所として P を定義
   X <- Parameter(index=dprod(S,P),freeny.x) # 説明変数 X を作る
 # 以降数理計画問題の記述
   i <- Element(set=S)
   j <- Element(set=P)
   v <- Variable(index=j)
   v0 <- Variable()
   e <- Expression(index=i)
   e[i] ~ v0 + Sum(X[i,j]*v[j],j) - Y[i]
   esum <- Objective(type=minimize)
   esum ~ Sum(e[i]*e[i],i)
```

```
}
```

このモデルを解くには，式を展開して System オブジェクトを作り，最適化を実行します．

```
sys <- System(Nlsfit)   # 式の展開
sol <- solve(sys)       # 最適化実行
```

次の Nlsfit.gen は数理計画のためのデータを引数で渡すようにしたより汎用的なモデルです．

```
Nlsfit.gen <- function(x,y) # x は matrix 型， y は vector 型
{
   S <- Set()
   Y <- Parameter(index=S,as.array(y))
   P <- Set()
   X <- Parameter(index=dprod(S,P),x)
   i <- Element(set=S)
   j <- Element(set=P)
   v <- Variable(index=j)
   v0 <- Variable()
   e <- Expression(index=i)
   e[i] ~ Sum(X[i,j]*v[j],j) + v0 - Y[i]
   esum <- Objective(type=minimize)
   esum ~ Sum(e[i]*e[i],i)
}
```

System オブジェクトを作成するときに，データを渡して，データとモデルを結びつける必要があります．

```
sys <- System(Nlsfit.gen,freeny.x,freeny.y)  # bind freeny.x,freeny.y
sol <- solve(sys) # solve
```

４-４-１.変数や式（Variable/Expression）の値を array 型のデータとして取り出す

solve()を起動すれば，最適化の結果は System()で作成されたシステムオブジェクト sys に問い合わせれば得られます．sys には，最適化モデルの情報がすべて集約しています．例えば v0 および v が幾つになっているか調べるには次のように current という関数を用いて，数理計画問題の中の S+NUOPT のオブジェクトを取り出して表示します．

```
sys <- System(Nlsfit.gen,freeny.x,freeny.y)
sol <- solve(sys,trace=F)
current(sys,v0)   # sys の中の"v0"を取り出して表示
current(sys,v)    # sys の中の"v"を取り出して表示
```

v0 と v は次のように表示されます．

```
> current(sys,v0)
[1] -10.47261
> current(sys,v)
```

```
  income level lag quarterly revenue market potential price index
     0.7674609             0.1238646         1.330558  -0.7542401
 attr(, "indexes"):
 [1] "j"
```

しかし，current の戻り値は S+NUOPT のオブジェクトの参照（この場合には変数オブジェクト Variable)なので,この後 S-PLUS 側で様々な操作を行うためにはそれぞれ as.array()を使って次のように S-PLUS のオブジェクトに変換する必要があります.

```
v0 <- as.array(current(sys,v0))
v <- as.array(current(sys,v))
```

current を使って取り出した S+NUOPT のオブジェクトに対して，直接 as.vector()は使えません（意味のない結果 0 が帰ります）. Vector がほしい場合でも，一旦は as.array を用いて array に変換する必要があります.
次の S-PLUS コードはこうして得られた変数をまとめて一つのベクトル型の変数 vvec として表現します．array 型の変数である v に対して as.vector とすると，添え字の情報が落ちてしまうので，添え字を rownames で取り出し，names 属性としています.

```
v0 <- as.array(current(sys,v0))
v <- as.array(current(sys,v))
vvec <- as.vector(v)      # vector 型に変換
names(vvec) <- rownames(v) # 落ちてしまった添え字の情報を付与
vvec <- c(Intercept=v0,vvec) # v0 をつなげる
vvec
```

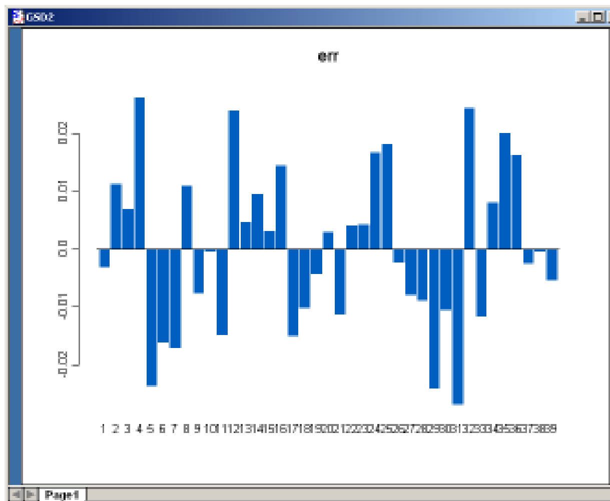
vvec は次のように表示されます.

```
 > vvec
  Intercept income level lag quarterly revenue market potential price index
  -10.47261    0.7674609             0.1238646         1.330558  -0.7542401
```

各点の誤差を示す式オブジェクト（Expression）である e の内容も全く同様にして，sys に対して current を適用して取り出します.

```
err <- as.array(current(sys,e))
barplot(err,names=rownames(err))
```

次の棒グラフが描かれます.



4-4-2.変数や式（Variable/Expression）の値を list 型のデータとして取り出す

S+NUOPT のオブジェクトは，すべて as.list を用いて list 型に変換して処理することができます．変換の規則は入力の場合と同じで，list の最初から length(list)-1 個の要素が添え字のベクトル，最後の要素が値のベクトルとみなします．
具体的に表示してみます．

```
sys <- System(NIsfit.gen,freeny.x,freeny.y)
sol <- solve(sys)
err <- as.list(current(sys,e))
err
```

結果は次のようになります．

```
> err
$indexes:
$i:
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
[23] 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

$values:
 [1] -0.0031897999  0.0111499356  0.0068458061  0.0259426225
 [5] -0.0236273123 -0.0160653218 -0.0169512378  0.0108668721
 [9] -0.0075620169 -0.0003823201 -0.0147084654  0.0237528911
[13]  0.0045025693  0.0093397389  0.0029600214  0.0142621487
[17] -0.0149334126 -0.0102146132 -0.0042357193  0.0028433516
[21] -0.0113535030  0.0039590091  0.0041339447  0.0165713365
[25]  0.0180124412 -0.0022507261 -0.0078694559 -0.0089016807
[29] -0.0241231336 -0.0104325444 -0.0267123682  0.0242586554
[33] -0.0115643780  0.0079192007  0.0200001064  0.0160025223
[37] -0.0024336174 -0.0003716668 -0.0054385080
```

二次元のオブジェクトの場合には添え字の部分が増えます．

```
S <- Set()
P <- Set()
X <- Parameter(index=dprod(S,P),freeny.x)
Xlist <- as.list(X)
Xlist
```

Xlist の内容は次のように表示されます.

```
> Xlist
$indexes:
$"*":
  [1] "1"  "1"  "1"  "1"  "2"  "2"  "2"  "2"  "3"  "3"  "3"  "3"  "4"
 [14] "4"  "4"  "4"  "5"  "5"  "5"  "5"  "6"  "6"  "6"  "6"  "7"  "7"

              ..(中略)..
$"*":
 [1] "income level"          "lag quarterly revenue"
 [3] "market potential"      "price index"
 [5] "income level"          "lag quarterly revenue"
 [7] "market potential"      "price index"
 [9] "income level"          "lag quarterly revenue"
[11] "market potential"      "price index"
[13] "income level"          "lag quarterly revenue"
[15] "market potential"      "price index"
[17] "income level"          "lag quarterly revenue"
[19] "market potential"      "price index"
              ..(中略)..
$values:
  [1]  5.82110  8.79636 12.96990  4.70997  5.82558  8.79236 12.97330
  [8]  4.70217  5.83112  8.79137 12.97740  4.68944  5.84046  8.81486
 [15] 12.98060  4.68558  5.85036  8.81301 12.98310  4.64019  5.86464
              ..(中略)..
```

５． ポートフォリオ最適化

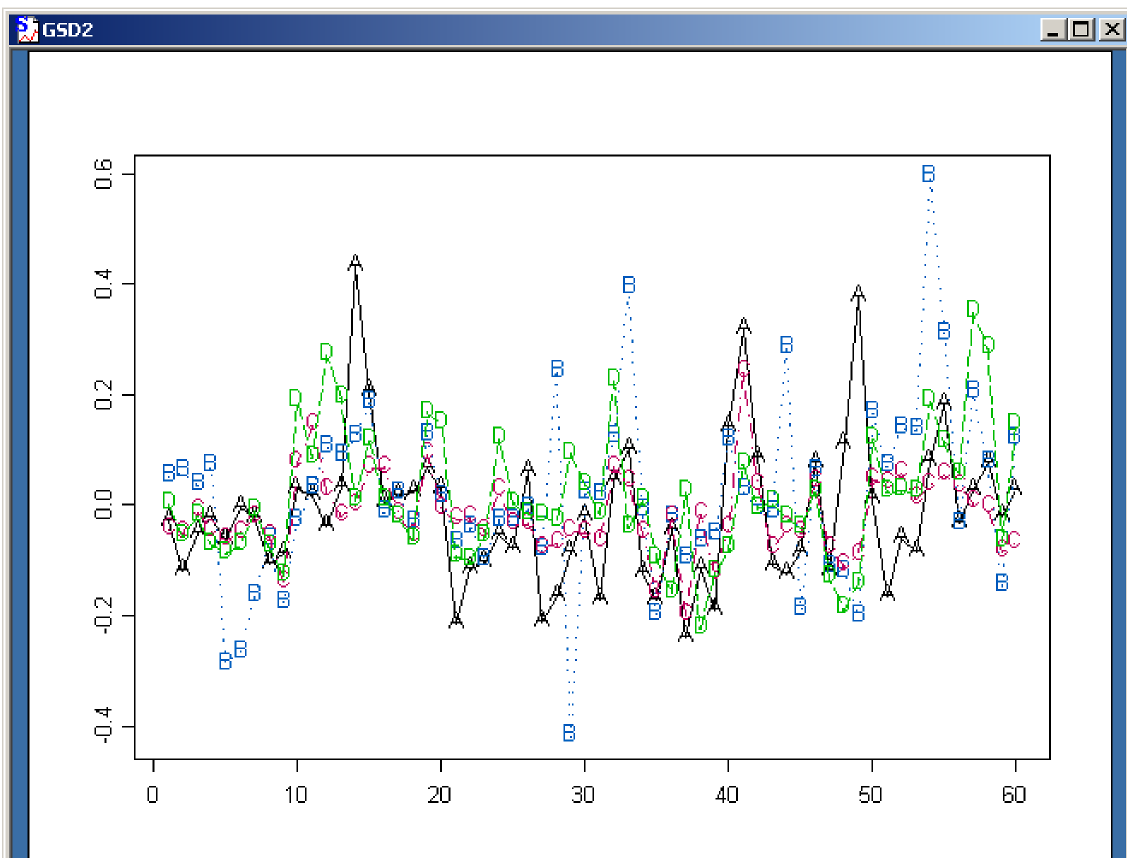
５－１　基本的なマルコビッツモデル

4 銘柄の時系列データを用いて，マルコビッツモデル：

最小化	$\sum_{i,j \in Asset} Q_{ij} x_i x_j$	（収益率の分散）
制約	$\sum_{j \in Asset} x_j = 1$	
	$\sum_{j \in Asset} \bar{r}_j x_j \geq r_{\min}$	（最低期待収益率の確保）
	$x_j \geq 0, \quad j \in Asset$	（空売りの禁止）

による最適ポートフォリオを求めます．ここでは $r_{\min}$ を 1% とします．
元にするのは４銘柄の６０期間分の収益率データ　R.60x4 です(S+NUOPT にサンプルとして含まれています)．
S-PLUS でグラフ化してみましょう．

```
# 60x4 の時系列の収益率推移 R.60x4 をグラフ化
matplot(rownames(R.60x4),R.60x4,pch=colnames(R.60x4),type="o")
```



次にS-PLUSの機能を使って最適化の材料となる分散Qを求めます.

```
> Q <- var(R.60x4)
> Q
            A           B           C           D
A 0.017788838 0.005730923 0.004552905 0.003779794
B 0.005730923 0.026978759 0.005153147 0.008770707
C 0.004552905 0.005153147 0.005200091 0.004284923
D 0.003779794 0.008770707 0.004284923 0.014216287
```

平均$\bar{r}$も求めましょう.

```
> rbar <- apply(R.60x4,2,mean)
> rbar
          A         B           C          D
 -0.01483683 0.0147342 -0.01805572 0.01391076
```

NUOPTが認識するためには, S-PLUSのvector型オブジェクトはarray型として定義しておく必要があります.

```
        rbar <- as.array(rbar)
```

最適化モデルはS-PLUSの手続きとして定義しておきます.

```
        Marko <- function(Q,rbar,rmin)
        {
            Asset <- Set()
            Q <- Parameter(index=dprod(Asset,Asset),Q)
            rbar <- Parameter(index=Asset,rbar)
            i <- Element(set=Asset)
            j <- Element(set=Asset)
            x <- Variable(index=Asset)
            risk <- Objective(type="minimize")
            risk ~ Sum(x[j]*Q[i,j]*x[i],i,j)
            Sum(rbar[j]*x[j],j) >= rmin
            Sum(x[j],j) == 1
            x[j] >= 0
        }
```

NUOPTをロードします.

```
> module(nuopt)
NUOPT for S-PLUS Version 2.11.1.9 Release 1 for Microsoft Windows
```

モデルにデータを代入するには次のように Systemコマンドを使います. この命令で最適化問題が定義されます.

```
> s <- System(model=Marko,Q,rbar,0.01)
```

```
Evaluating Marko(Q,rbar,0.01) ... ok!
Expanding objective  (1/4 name="risk")
Expanding constraint (2/4)
Expanding constraint (3/4)
Expanding constraint (4/4)
```

展開した結果は s というオブジェクトになりますので，次のように内容を表示することができます.

```
> s
1-1 :  -0.0148368*x[A]+0.0147342*x[B]-0.0180557*x[C]+0.0139108*x[D] >= 0
.01

2-1 :  x[A]+x[B]+x[C]+x[D] == 1

3-1 :  x[A] >= 0
3-2 :  x[B] >= 0
3-3 :  x[C] >= 0
3-4 :  x[D] >= 0

risk<objective>: 0.0177888*x[A]*x[A]+0.00573092*x[A]*x[B]+0.00455291*x[A
]*x[C]+0.00377979*x[A]*x[D]+0.00573092*x[A]*x[B]+0.0269788*x[B]*x[B]+0.0
0515315*x[B]*x[C]+0.00877071*x[B]*x[D]+0.00455291*x[A]*x[C]+0.00515315*x
[B]*x[C]+0.00520009*x[C]*x[C]+0.00428492*x[C]*x[D]+0.00377979*x[A]*x[D]+
0.00877071*x[B]*x[D]+0.00428492*x[C]*x[D]+0.0142163*x[D]*x[D] (minimize)
```

では解きましょう.

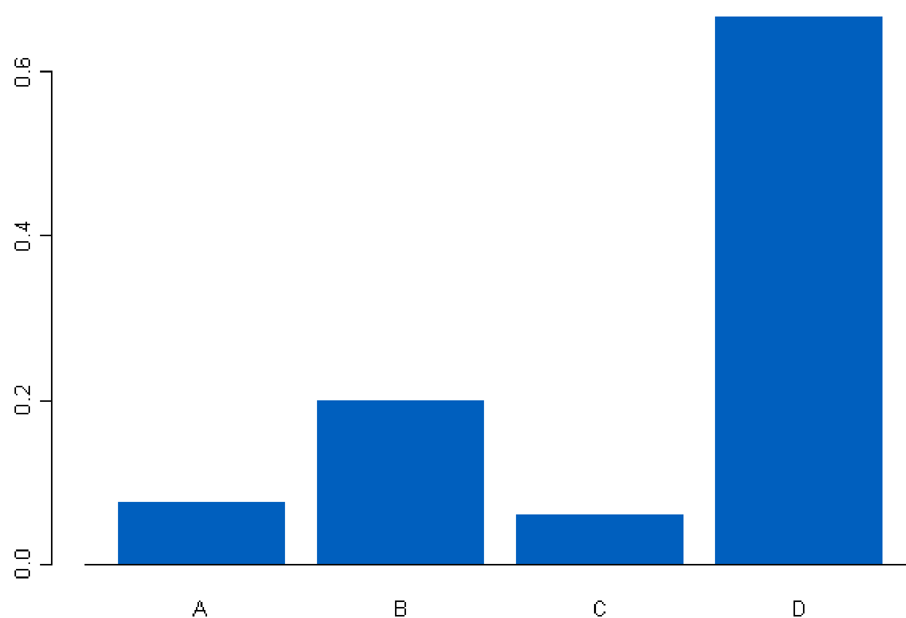
```
> sol <- solve(s)
NUOPT 11.1.9 (NLP/LP/IP/SDP module)
       <with META-HEURISTICS engine "wcsp"/"rcpsp">
      , Copyright (C) 1991-2008 Mathematical Systems Inc.
NUMBER_OF_VARIABLES                                          4
NUMBER_OF_FUNCTIONS                                          3
PROBLEM_TYPE                                      MINIMIZATION
METHOD                                        TRUST_REGION_IPM
<preprocess begin>..........<preprocess end>
<iteration begin>
    res=1.0e+001 .... 8.3e-005  5.9e-007
<iteration end>
STATUS                                                 OPTIMAL
VALUE_OF_OBJECTIVE                               0.01087261487
ITERATION_COUNT                                              6
FUNC_EVAL_COUNT                                              9
FACTORIZATION_COUNT                                          8
RESIDUAL                                      5.879111059e-007
ELAPSED_TIME(sec.)                                        0.00
```

解ベクトルx（組み入れ比率）を取り出します．オブジェクト s に対して，current（現在の値の取得）という手続きを使います．

```
> xopt <- as.array(current(s,x))
> xopt
          A          B          C          D
 0.07466462 0.1986515 0.06031033 0.6663735
attr(, "indexes"):
[1] "*"
```

棒グラフにしてみましょう．

```
> barplot(names=rownames(xopt),xopt)
```



ポートフォリオの組み入れ比率に従った場合の収益率の時系列(pf)を出してみます．

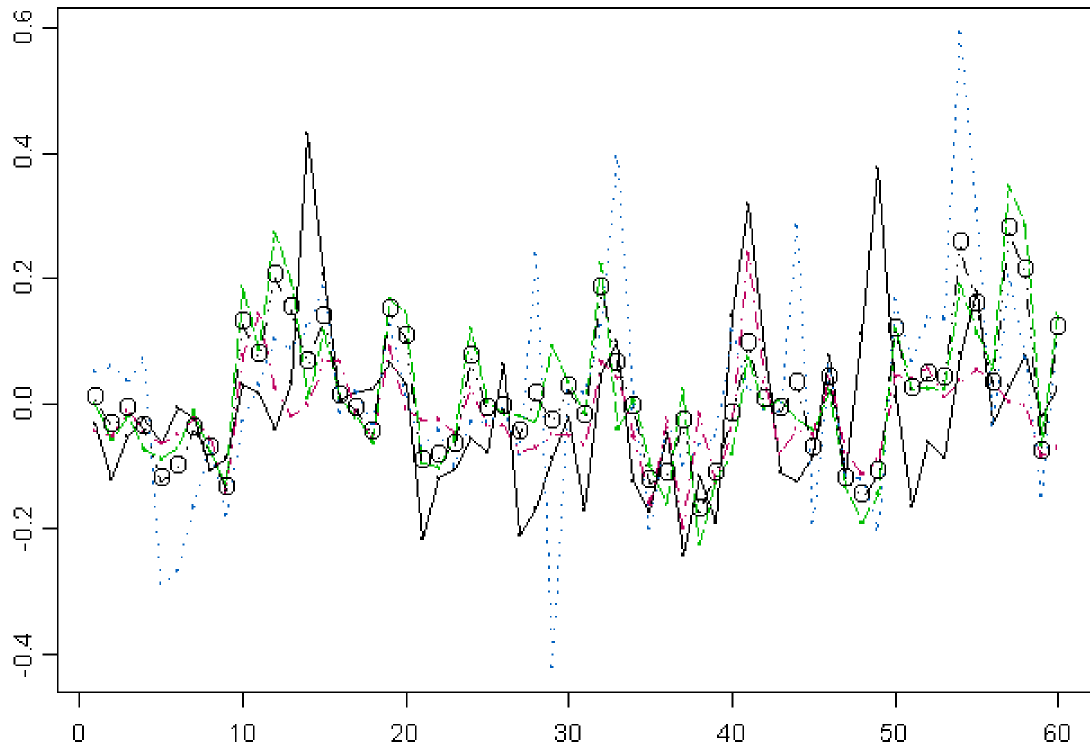
```
> pf <- R.60x4 %*% as.vector(xopt)
```

分散は

```
>var(pf)
           [,1]
[1,] 0.01087261
```

とあって，上で表示されている目的関数値(VALUE_OF_OBJECTIVE)とほぼ一致しています．
元の収益率グラフにこのポートフォリオの収益率の変動を重ねてみてみます．

```
> Raug <- cbind(R.60x4,pf)
> matplot(rownames(Raug),Raug,pch=c(".",".",".",".","0"),type="o")
```



"0"のプロットがあるのがポートフォリオで，変動が抑えられていることがわかります.

5-2 さまざまなリスク尺度によるポートフォリオ最適化

ポートフォリオ最適化におけるリスク尺度（変動の測り方）には分散以外にも様々なものがあります. ここでは, 分散以外のリスク尺度として, 絶対偏差, 1 次の下方部分積率 (LPM), Conditional Value at Risk (CVaR) をリスク尺度としたポートフォリオ最適化問題を扱います. これらのリスク尺度を用いたモデルはすべて線形計画問題として記述できます.
ここで元にするデータは 8000 ケースの 5 銘柄の収益率を含む R.8000x5 です.

5-2-1.分散をリスクとした場合

まずは比較のため, 分散最小化モデル（コンパクト分解表現）を解いてみましょう. S+NUOPT ではモデルを次のように定義します.

```
# 分散最小化モデル
MinVar <- function(r.d)
{
    Asset <- Set()
    j <- Element(set=Asset)
    Sample <- Set()
    t <- Element(set=Sample)
    r <- Parameter(index=dprod(t,j),r.d)
    rb <- Parameter(index=j)
    rb[j] ~ Sum(r[t,j],t)/nrow(r.d)
```

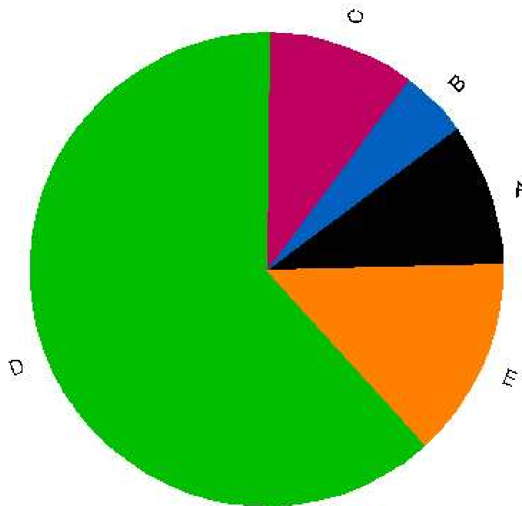
```
    x <- Variable(index=j)
    s <- Variable(index=t)
    f <- Objective(type=minimize)
    f ~ Sum(s[t]*s[t],t)/nrow(r.d)
    Sum(x[j],j) == 1
    x[j] >= 0
    Sum((r[t,j]-rb[j])*x[j],j) == s[t]
}
```

次のようにして最適化を行います.

```
# 展開（データの代入）
sys <- System(MinVar, R.8000x5)
# 最適化実行
sol <- solve(sys)
# 解の取得
x <- as.array(current(sys,x))
```

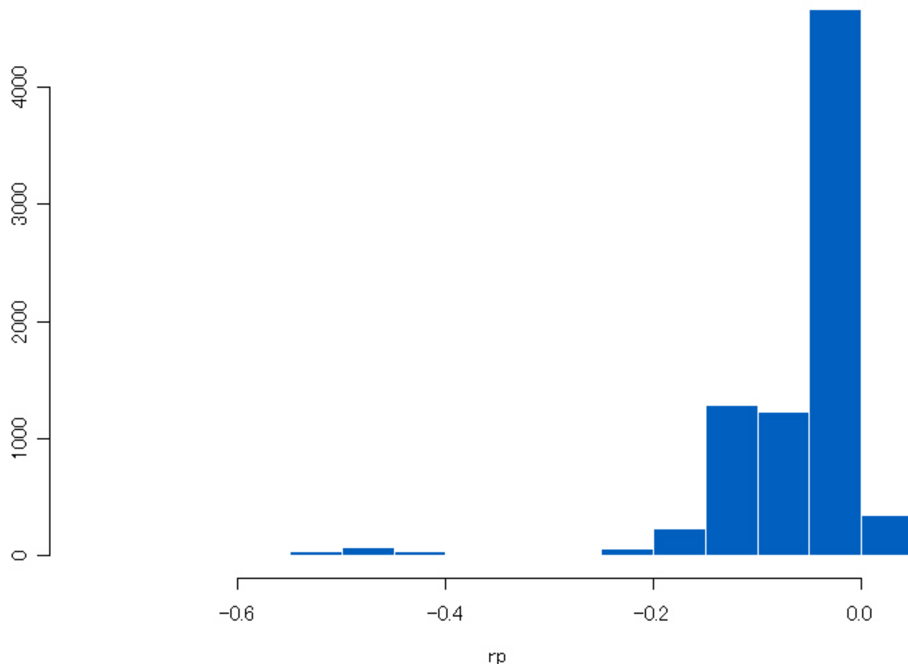
さて，結果の解析に入りましょう．まず最適ポートフォリオ x を見てみます.

```
# 図示
pie(1000*x,names=dimnames(x)[[1]],inner=1.2)
```



次に，ポートフォリオに従った場合の収益率の分布のヒストグラムを書いてみます.

```
rp <- as.matrix(R.8000x5) %*% as.vector(x)
graphsheet()
hist(rp)
```



8000 ケースのうち，わずかですが，収益率が非常に悪いケース（収益率 -0.6 ～ -0.4）が存在します．対象としているアセットが社債でデフォルトするリスクが含まれている場合にこのようなことが起きます．このようなアセットに対して分散最小化モデルを適用するのは危険であるといえるでしょう．

5-2-2.絶対偏差をリスクとした場合

次に，リスク尺度として絶対偏差を用いた絶対偏差最小化モデルを解いてみましょう．このモデルは絶対値関数 abs() を用いずに，線形計画問題として定式化するテクニックを利用しています．

```
#### 絶対偏差最小化モデル ####
MinMad <- function(r.d)
{
    Asset <- Set()
    j <- Element(set=Asset)
    Sample <- Set()
    t <- Element(set=Sample)
    r <- Parameter(index=dprod(t,j),r.d)
    rb <- Parameter(index=j)
    rb[j] ~ Sum(r[t,j],t)/nrow(r.d)
    x <- Variable(index=j)
    s <- Variable(index=t)
```

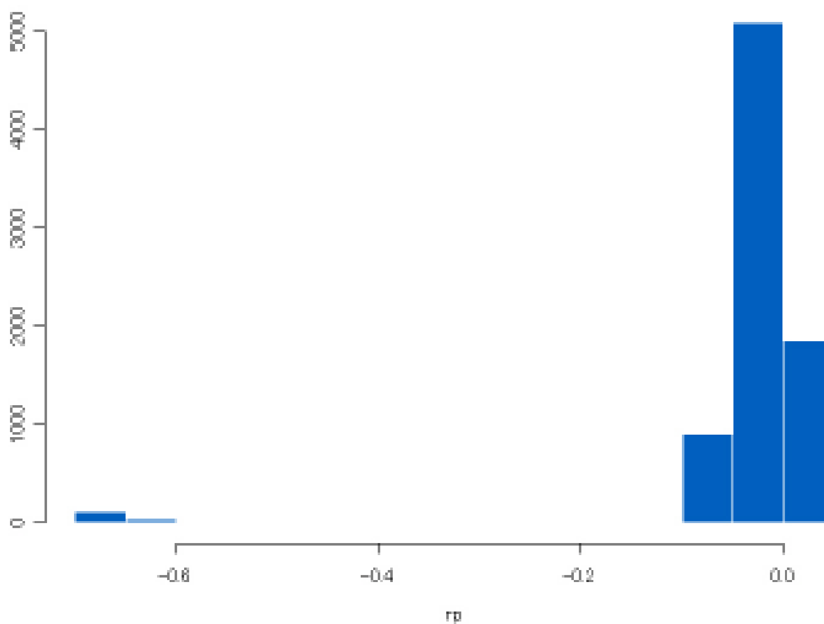
```
    u <- Variable(index=t)
    v <- Variable(index=t)
    f <- Objective(type=minimize)
    f ~ Sum(u[t]+v[t],t)/nrow(r.d)
    Sum(x[j],j) == 1
    x[j] >= 0
    Sum((r[t,j]-rb[j])*x[j],j) == s[t]
    u[t] >= 0
    v[t] >= 0
    s[t] == u[t] - v[t]
}
```

次は最適化の実行手順です.

```
# 展開
sys <- System(MinMad, R.8000x5)
# 実行
sol <- solve(sys)
# 解の取得
x <- as.array(current(sys,x))
```

ヒストグラムを見てみましょう.

```
rp <- as.matrix(R.8000x5) %*% as.vector(x)
hist(rp)
```



分散最小化モデルと比べて，線形計画問題として定式化できるという利点がありますが，

大崩に対する感度が小さく，収益率 -0.6 以下のケースが見受けられます．

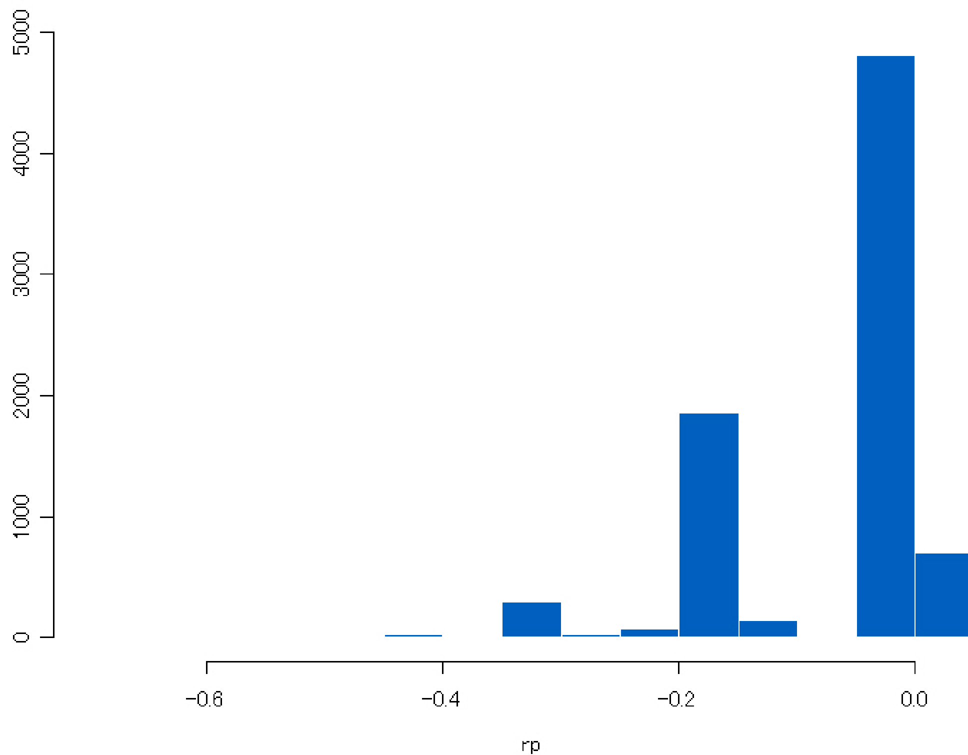
５－２－３． １次の下方部分積率（LPM）をリスクとした場合

次に，下方リスクモデルの一つである 1 次の下方部分積率（LPM）最小化モデルの結果を見てみましょう．このモデルは収益率 rG 以上のサンプル点については関知しないかわりに，目標収益率 rG を下回る部分の平均を最小化しようとします．

```
#### 1 次の下方部分積率（LPM）最小化モデル ####
MinLPM1 <- function(r.d, rG)
{
    Asset <- Set()
    j <- Element(set=Asset)
    Sample <- Set()
    t <- Element(set=Sample)
    r <- Parameter(index=dprod(t,j),r.d)
    rb <- Parameter(index=j)
    rb[j] ~ Sum(r[t,j],t)/nrow(r.d)
    x <- Variable(index=j)
    s <- Variable(index=t)
    f <- Objective(type=minimize)
    f ~ Sum(s[t],t)/nrow(r.d)
    Sum(x[j],j) == 1
    x[j] >= 0
    s[t] >= 0
    rG <= Sum(r[t,j]*x[j],j) + s[t]
}
```

rG は下回ってほしくない（リスクとみなせる）収益率を与えます．ここでは-0.4 としましょう．

```
# 展開
sys <- System(MinLPM1, R.8000x5, -0.4)
# 実行
sol <- solve(sys)
# 解の取得
x <- as.array(current(sys,x))
rp <- as.matrix(R.8000x5) %*% as.vector(x)
graphsheet()
hist(rp)
```



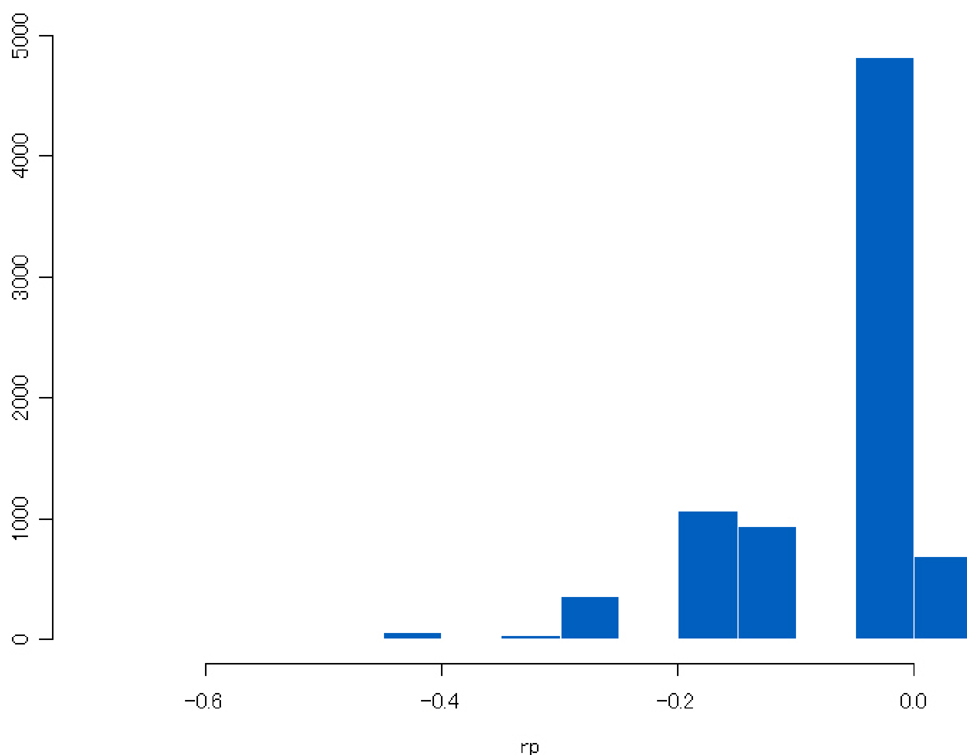
目標収益率 -0.4 を下回るケースが少なく，結果的に分散最小化モデルや絶対偏差最小化モデルと比べて大崩れするケースが少なくなっていることがわかります．

5-2-4． CVaR をリスクとした場合

最後に Conditional Value at Risk（CVaR）最小化モデルです．CVaR は収益率の損失がある確率水準（パーセント点）β を上回るときの平均損失のことです．つまり，収益率の下位 $(1-\beta)\times|Asset|$ （$|Asset|$ は銘柄数）の平均損失を最小化します．この例では $\beta=0.97$，$|Asset|=8000$ ですので，収益率の下位 240 個のサンプル点の平均損失を最小化します．

```
#### Conditional Value at Risk (CVaR) 最小化モデル ####
MinCVaR <- function(r.d, beta)
{
    Asset <- Set()
    j <- Element(set=Asset)
    Sample <- Set()
    t <- Element(set=Sample)
    r <- Parameter(index=dprod(t,j),r.d)
    rb <- Parameter(index=j)
    rb[j] ~ Sum(r[t,j],t)/nrow(r.d)
    x <- Variable(index=j)
    s <- Variable(index=t)
    VaR <- Variable()
    f <- Objective(type=minimize)
```

```
    f ~ Sum(s[t],t)/(nrow(r.d)*(1-beta)) + VaR
    Sum(x[j],j) == 1
    x[j] >= 0
    s[t] >= 0
    Sum(r[t,j]*x[j],j) + VaR + s[t] >= 0
}

# 展開
sys <- System(MinCVaR, R.8000x5, 0.97)
# 実行
sol <- solve(sys)
# 解の取得
x <- as.array(current(sys,x))
# 図示
graphsheet()
rp <- as.matrix(R.8000x5) %*% as.vector(x)
hist(rp)
```



1 次の下方部分積率（LPM）最小化モデル同様，分散最小化モデルや絶対偏差最小化モデルと比べて大崩れするケースが少なくなっていることがわかります．

５-３　コンパクト分解（大規模ポートフォリオ最適化）

基本的なマルコビッツモデルの弱点として

$$\sum_{i,j \in Asset} Q_{ij} x_i x_j, \qquad \text{（収益率の分散）}$$

という式に表れる分散・共分散行列を取得しなければならないという点があります．昨今実務家が解かねばならない 1000 銘柄～3000 銘柄のポートフォリオではこのような分散・共分散行列を陽に生成すると計算負荷が増大してしまいます．
そのような場合には

R ：各銘柄の収益率のヒストリカルデータ
（各列ｊは，期間 $Period$ にわたる銘柄ｊの収益率の観測値を並べたもの）

を直接データとして入力，

$$Q_{ij} = \frac{1}{|Period|-1} \sum_{t \in Period} (R_{ti} - \bar{r}_i)(R_{tj} - \bar{r}_j)$$

と表されることを利用して，補助変数 s を定義して次のような制約を導入し，

$$s_t = \sum_j (R_{tj} - \bar{r}_j) x_j,$$

リスクを次のように書き換えます．

$$\frac{1}{|Period|-1} \sum_{t \in Period} {s_t}^2$$

こうすれば，分散・共分散行列を直接生成しなくともよいので，効率的に計算を行うことができます． 1000 銘柄のポートフォリオを求めてみます．データとして月次５年分（６０期間）のデータ R.60x1000 が得られているとします．
次のモデルでは分散共分散行列を作らず，R.60x1000 を直接入力データとして最適化を行うことができます．

```
MinVar <- function(r.d)
{
    Asset <- Set()
    j <- Element(set=Asset)
    Sample <- Set()
    t <- Element(set=Sample)
    r <- Parameter(index=dprod(t,j),r.d)
    rb <- Parameter(index=j)
    rb[j] ~ Sum(r[t,j],t)/nrow(r.d)
    x <- Variable(index=j)
    s <- Variable(index=t)
    f <- Objective(type=minimize)
    f ~ Sum(s[t]*s[t],t)/nrow(r.d)
    Sum(x[j],j) == 1
    x[j] >= 0
```

```
    Sum((r[t,j]-rb[j])*x[j],j) == s[t]
}
# 展開
sys <- System(MinVar, R.60x1000)
# 実行
sol <- solve(sys)
# 解の取得
x <- as.array(current(sys,x))
```

同様のテクニックは収益率が

$$r_j = \alpha_j + \sum_k \beta_{jk} f_k + \varepsilon_j \qquad k \in Factor,\ j \in Asset,\ \varepsilon_j \in N(0,\sigma_j^2)$$

のようにファクターリターン f_k によって表現されているファクターモデルの場合にも利用することができます．このとき収益率の分散は次のように表現されますので

$$\sum_{i,j \in Asset} \sum_{k,l \in Factor} Q_{k,l}^f \beta_{j,k} \beta_{j,l} x_i x_j + \sum_{j \in Asset} \sigma_j^2 x_j^2$$

中間変数

$$s_k = \sum_j \beta_{j,l} x_j$$

なる線形制約式を導入し分散は

$$\sum_{k,l} Q_{k,l}^f s_k s_l + \sum_j \sigma_j^2 x_j^2$$

と表現することができます．
コンパクト分解は効率的であるのみならず，下方リスクモデルなど様々なリスク尺度によるポートフォリオモデルを記述する場合の基本となります．

５－４　端株処理

一般にポートフォリオ最適化問題においては，取引額を連続量として求めることが多いのですが，実際には各銘柄ごとの最小取引額の整数倍を取引しなければなりません．連続量として求められた取引額を何らかの形で最小取引額の整数倍に補正する作業のことを端株処理と呼びます．
紹介するモデルでは，各銘柄において端株を切り上げるなら 1 切り下げるなら 0 を取るような 0-1 整数 d を用いて，取引の総額が与えられた値の±1% の範囲に収まるという制約の下で，各サンプル点における収益額の絶対偏差を最小化しています．
ここでは，前項のコンパクト分解モデルによって得られた解 RoundLot.x を端株処理してみましょう．取引の総額を 50 億円 とします．また，以下のような最小取引額データ RoundLot.unit が与えられているとします．

```
> RoundLot.unit
    A0001    A0002   A0003    A0004    A0005    A0006    A0007     A0008    A0009
 10880000 5400000 873000 1875000 7040000 4980000 3360000 15100000 5390000

   A0010    A0011    A0012   A0013    A0014     A0015    A0016    A0017     A0018
 2360000 4330000 6160000 994000 1385000 15150000 2200000 4390000 18000000

   A0019    A0020    A0021    A0022    A0023    A0024    A0025    A0026    A0027
 2760000 3790000 2395000 2730000 4430000 4160000 2960000 2265000 5410000
…
```

次が切り上げ，切り下げを決定する最適化モデルです．比較的規模の大きな０－１計画問題ですのでメタヒューリスティクス解法 wcsp を用いて解きます．

```
#### 端株処理モデル ####
RoundLot <- function(r.d, unit.d, x.d, total)
{
    Asset <- Set()
    j <- Element(set=Asset)
    Sample <- Set()
    t <- Element(set=Sample)
    r <- Parameter(index=dprod(t,j),r.d)
    rb <- Parameter(index=j)
    rb[j] ~ Sum(r[t,j],t)/nrow(r.d)
    unit <- Parameter(index=j,unit.d)
    base <- Parameter(index=j,(total*x.d)%/%unit.d*unit.d)
    d <- IntegerVariable(type="binary",index=j)
    fund <- Expression(index=j)
    fund[j] ~ base[j] + unit[j]*d[j]
    0.99*total <= Sum(fund[j],j) <= 1.01*total
    s <- Expression(index=t)
    s[t] ~ 0.01*Sum((r[t,j]-rb[j])*fund[j],j)
    softConstraint(1,0,1)
    s[t] == 0
}
# 総額（50億円）
total <- 5e9
# 展開
sys <- System(RoundLot, R.60x1000, RoundLot.unit, RoundLot.x, total)
```

次で解を求めます．wcsp を用いる場合には時間制限が必要です．

```
# 実行（wcsp を指定して解く）
nuopt.options(method="wcsp")
nuopt.options(maxtim=20)
sol <- solve(sys)
```

解が得られたら，次のようにして,解 d とそれを採用した場合の取引額を取得しましょう．

```
# 解の取得
d <- as.array(current(sys,d))
fund <- as.array(current(sys,fund))
```

次のようにして，連続量から計算した理想的な投資量と具体的な投資量を定義します．

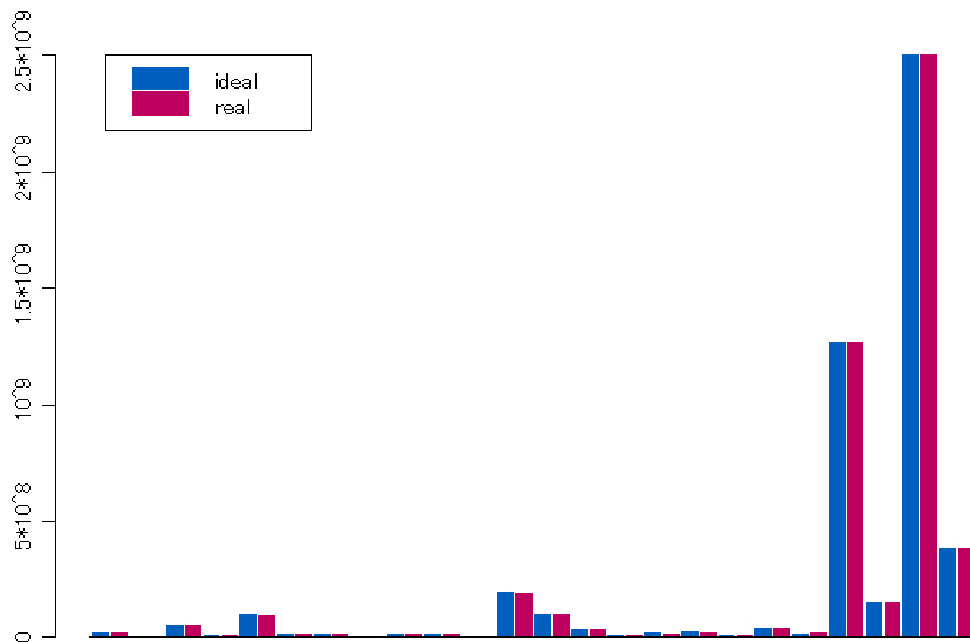
```
n <- sum(fund > 0)
ideal <- (total* RoundLot.x)[fund > 0] # 理想的な投資量
up <- d[fund > 0]
real <- fund[fund > 0]          # 具体的な投資量
```

理想的な取引額(ideal)，切り上げるか否か(up)，実際の取引額(real)を表示します．

```
> cbind(ideal, up, real)
              ideal up       real
A0072 2.268026e+007  0   22275000
A0113 1.116033e-006  1    3740000
A0150 4.873316e+007  1   50140000
A0461 1.049896e+007  0   10296000
A0508 9.945964e+007  0   97500000
A0647 1.501497e+007  1   15030000
A0664 1.513937e+007  1   16450000
A0712 1.759865e-006  1    3880000
A0713 1.312477e+007  0   12705000
A0717 1.757595e+007  0   17492400
A0725 2.784842e-007  1     609000
A0746 1.929582e+008  0  188160000
A0751 1.009987e+008  0   99960000
A0870 3.162644e+007  1   32890000
A0875 7.202727e+006  1    9060000
A0907 2.333477e+007  0   17970000
A0919 2.447410e+007  0   21700000
A0935 7.131422e+006  0    6769000
A0945 4.145754e+007  0   38940000
A0962 1.749186e+007  1   18480000
A0975 1.265938e+009  0 1265850000
A0977 1.487018e+008  1  150960000
A0987 2.505445e+009  0 2505170000
          ideal up      real
A0997 385938255  0 385640000
```

グラフに描きます．

```
barplot(rbind(ideal,real),beside=T,col=rbind(rep(2,n),rep(3,n)))
legend(1,2.5e9,c("ideal","real"),fill=2:3)
```



５－５　銘柄グルーピング

ポートフォリオ最適化後に適当な端株処理を行い，各銘柄に対する投資額が決まれば，あとは実際に買い／売り注文を出すだけです．しかし，一度に巨額の注文を出すのは，マーケットインパクト等を考慮すれば，あまり好ましいことではありません．そこで，銘柄を複数のグループに分けてグループ毎に注文を出すことがあります．このとき，各グループの性質がそれほど大きく異ならないようにしたい，というのは自然に発生する要求です．ここでは，各銘柄にはいくつかのファクター値（例えば，投資額）が与えられており，グループごとのファクター値が均等になるように銘柄をグルーピングすることを目的とします．

ここでは，50 個の銘柄を 5 個のグループに分けることを考えます．各銘柄は次のように，F0 から F9 までの 10 個のファクター値で特徴づけられるとします．

```
> Basket.fcoef
      F0    F1    F2    F3    F4    F5    F6    F7     F8    F9
1   0.81  0.29  0.42  0.00 -0.14  0.03  0.04  1.83 -14.89 -0.20
2   1.29  0.39  0.41  0.04  0.10  0.18 -0.19  4.43 -15.35  0.23
3   1.00 -0.16  0.32  0.02 -0.02  0.25  0.14  2.12 -26.86  0.19
4  -2.75 -0.24 -0.40  0.01  0.29 -0.15 -0.40 -2.09  28.49 -0.18
5  -0.64 -0.12 -0.19 -0.14  0.00  0.22 -0.05 -1.84  24.50  0.72
6  -5.94  0.05 -0.20  0.03 -0.08 -0.24 -0.06 -1.57  49.74 -0.17
7   1.85  0.03  0.59  0.04  0.31  0.28  0.12  2.12 -42.61 -0.88
8   5.20  0.07  0.22  0.03 -0.07  0.01  0.19 -2.98  -2.71 -0.28
9  12.82  0.12  0.24  0.03 -0.10  0.13  0.47  3.40 -42.22  0.21
```

10	4.15	0.05	0.12	0.01	-0.05	0.09	0.10	-0.35	23.63	0.22
11	-32.92	0.12	-0.11	0.03	-0.12	-0.12	0.17	-1.81	16.64	-0.17
12	4.72	-0.20	-0.08	-0.03	-0.17	-0.04	0.20	3.84	-6.97	0.34
13	4.75	0.31	0.67	0.00	-0.01	0.48	-0.45	7.06	-31.22	0.18
14	-12.12	-0.07	-0.04	-0.07	0.36	-0.10	-0.21	-4.90	4.29	0.06
15	-7.92	-0.22	-0.26	-0.01	0.06	-0.01	-0.21	0.84	8.11	0.83
16	6.30	0.03	0.23	0.02	-0.07	0.19	0.00	2.46	-16.53	0.14
17	-30.70	-0.52	-0.04	-0.02	0.33	0.17	-0.22	-2.99	81.08	-0.18
18	-5.14	-0.09	-0.46	0.01	-0.17	-0.29	0.04	-1.05	64.62	-0.35
19	-5.92	0.23	-0.02	-0.01	0.08	-0.20	-0.23	-0.04	-11.09	-0.04
20	-12.54	-0.16	-0.06	0.00	0.18	0.06	-0.07	1.78	-2.60	-0.23
21	2.46	0.29	0.56	0.06	0.06	0.41	0.14	0.99	24.71	0.17
22	2.91	-0.26	0.16	0.01	-0.02	0.00	0.03	-1.26	12.49	0.16
23	3.78	0.11	0.16	0.00	-0.06	0.18	0.10	-0.66	10.73	0.14
	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
24	1.83	0.16	0.35	0.01	0.03	-0.05	0.11	-2.04	15.40	0.17
25	3.02	0.15	0.51	0.00	-0.10	0.07	0.34	3.21	-17.92	0.35
26	-17.63	0.00	0.12	0.03	-0.05	0.03	-0.03	1.55	-39.72	-0.19
27	-1.54	-0.60	-0.54	0.01	0.13	-0.16	-0.27	-2.41	20.25	-0.18
28	1.56	-0.15	-0.22	0.03	-0.29	-0.20	0.19	-3.81	33.04	-0.01
29	1.56	-0.15	-0.22	0.03	-0.29	-0.20	0.19	-3.81	33.04	-0.01
30	5.65	-0.18	-0.28	0.02	-0.15	-0.19	0.12	-4.15	33.42	0.15
31	1.36	0.40	0.57	0.06	-0.24	0.11	0.18	2.04	0.87	0.17
32	-1.69	-1.01	-0.48	-0.01	0.01	0.01	-0.37	-1.61	-6.85	-0.20
33	-1.32	0.00	-0.27	0.01	0.07	0.02	-0.44	3.43	-20.62	-0.22
34	-1.60	-0.14	-0.44	0.02	0.09	-0.33	0.16	-0.34	15.63	-0.25
35	2.46	0.00	-0.06	0.10	0.10	-0.11	0.10	3.93	-16.98	-0.88
36	-3.28	-0.03	-0.27	-0.02	0.03	-0.25	-0.03	-1.54	-19.38	-0.30
37	-1.48	-0.17	-0.28	0.03	0.21	0.03	-0.22	-2.21	1.19	-0.12
38	-2.38	-0.24	-0.02	-0.01	0.17	0.09	-0.01	-2.26	-14.43	0.25
39	-0.50	-0.25	-0.24	0.02	0.04	0.19	0.10	2.10	-1.32	-0.21
40	-0.35	-0.44	-0.27	0.01	0.11	0.28	-0.26	-2.88	-3.57	0.66
41	0.78	0.32	0.22	0.02	-0.13	0.00	-0.71	0.42	-26.84	-0.88
42	-14.75	0.00	-0.11	0.04	0.17	-0.23	-0.10	0.30	-5.03	0.00
43	0.46	0.86	0.45	0.07	0.12	-0.09	0.37	5.16	-71.47	-0.12
44	13.48	-0.47	-0.17	-0.01	0.06	0.06	0.09	1.86	-1.98	-0.31
45	-1.83	-0.54	-0.44	0.01	-0.49	-0.21	-0.25	2.32	0.31	0.40
46	-2.56	0.25	0.31	0.00	-0.06	0.19	0.02	3.26	-6.25	-0.28
	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
47	3.83	-0.48	-0.06	-0.02	-0.09	0.10	0.20	2.18	5.77	-0.40
48	-3.67	0.41	-0.03	0.03	0.14	-0.18	-0.04	-4.86	0.09	0.56
49	6.17	-0.26	0.09	-0.01	-0.05	0.13	0.16	1.81	44.54	0.00
50	4.11	0.45	0.35	0.00	0.24	-0.03	0.52	-0.47	-15.80	0.19

それぞれのグループに割り当てられた銘柄での F0 ～ F9 の合計が，与えられた範囲に収まること，また，特に重要な F2,F3,F4,F5,F6,F9 の属性については与えられた値にできるだけ近いことを要請します．

そのための数理計画モデルは以下のようになります.

```
#### 銘柄グルーピングモデル ####
Basket <- function(ng.d, fcoef.d, flow.d, fbar.d, fhigh.d, W.d)
{
    M <- Set()
    m <- Element(set=M)
    J <- Set()
    j <- Element(set=J)
    K <- Set(1:ng.d)
    k <- Element(set=K)
    f <- Parameter(index=dprod(j,m), fcoef.d)
    flow <- Parameter(index=m, flow.d)
    fbar <- Parameter(index=m, fbar.d)
    fhigh <- Parameter(index=m, fhigh.d)
    W <- Parameter(index=m, W.d)
    u <- IntegerVariable(index=dprod(j,k),type="binary")
    .F <- Expression(index=dprod(m,k))
    .F[m,k] ~ Sum(f[j,m]*u[j,k], j)
    selection(u[j,k],k)
    scf <- 1000
    scf*fhigh[m] >= scf*.F[m,k] >= scf*flow[m]
    softConstraint(scf,1)
    W[m]*(.F[m,k] - fbar[m]) == 0
    g <- Expression(index=j)
    g[j] ~ Sum(u[j,k]*k, k)
}
```

各ファクターのグループごとの下限,

```
> Basket.flow
  F0  F1  F2  F3  F4  F5  F6  F7  F8  F9
 -15 -15 -15 -15 -15 -15 -15 -15 -15 -15
```

上限,

```
> Basket.fhigh
 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9
 15 15 15 15 15 15 15 15 15 15
```

平均,

```
> Basket.fbar
 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9
  0  0  0  0  0  0  0  0  0  0
```

特に大事なものを定義する重みベクトル

```
> Basket.W
 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9
  0  0  1  1  1  1  1  0  0  1
```

が与えられているものとします．モデルを展開して求解します．グループの数は "5" としています．

```
# 展開
sys <- System(Basket, 5, Basket.fcoef, Basket.flow, Basket.fbar,
Basket.fhigh, Basket.W)
# 実行
nuopt.options(method = "wcsp",maxtim = 10)
sol <- solve(sys)
# 解の取得
gnum <- as.array(current(sys,g))
```

得られた解について，グループごとのファクター値がどのようになっているか，適当にグルーピングした場合と比較します．fopt は上記のモデルを使ってグルーピングした場合，frand は適当にグルーピングした場合の結果です．

```
# 検証
fopt <- rbind(
    apply(Basket.fcoef[gnum == 1,  ], 2, sum),
    apply(Basket.fcoef[gnum == 2,  ], 2, sum),
    apply(Basket.fcoef[gnum == 3,  ], 2, sum),
    apply(Basket.fcoef[gnum == 4,  ], 2, sum),
    apply(Basket.fcoef[gnum ==5,  ], 2, sum)
)
frand <- rbind(
    apply(Basket.fcoef[1:10,  ], 2, sum),
    apply(Basket.fcoef[11:20,  ], 2, sum),
    apply(Basket.fcoef[21:30,  ], 2, sum),
    apply(Basket.fcoef[31:40,  ], 2, sum),
    apply(Basket.fcoef[41:50,  ], 2,sum)
)
```

```
> fopt
         F0    F1    F2    F3    F4    F5    F6    F7    F8    F9
[1,] -14.65  0.76  0.36  0.16  0.09 -0.23 -0.23 -4.52 13.94  0.59
[2,] -14.13 -0.13 -0.41  0.24 -0.56 -0.32  1.01 -0.57 14.82 -0.77
[3,] -14.72 -0.24  0.06  0.16  1.02  0.35 -0.07 -2.66 14.22 -0.38
[4,] -14.60 -0.30  0.69  0.03  0.41  0.62 -0.37  8.34 14.89  0.42
[5,] -14.76 -2.15  0.11 -0.06 -0.49  0.19 -0.57  9.92 13.50 -0.61
> frand
         F0    F1    F2    F3    F4    F5    F6     F7     F8    F9
[1,]  17.79  0.48  1.53  0.07  0.24  0.80  0.36   5.07  -18.28 -0.14
[2,] -91.49 -0.57 -0.17 -0.08  0.47  0.14 -0.98   5.19  106.33  0.58
[3,]   3.60 -0.63  0.60  0.20 -0.74 -0.11  0.92 -12.39  125.44  0.75
```

```
[4,]  -8.78 -1.88 -1.76  0.21  0.59  0.04 -0.79   0.66  -65.46 -1.10
[5,]   6.02  0.54  0.61  0.13 -0.09 -0.26  0.26  11.98  -76.66 -0.84
```

適当にグルーピングした場合と比べて，均質なグループが得られていることが分かります.

５－６　Maximum Drawdown

Maximum Drawdown はポートフォリオ最適化問題におけるリスク尺度の一つであり，ある特定の期間において，資産額の最も大きな減少量のことを指します．特徴として，分散や下方部分積率と異なり，入力データをサンプル点ではなく時系列として扱う，という点が挙げられます．ここでは所与の期間において Maximum Drawdown が最も小さくなるような各投資対象に対する投資量を決定します．なお，投資量は初期時点でのみ決定され，その後リバランスは行われないものとします．
まずは，収益率データを初期価格を 1 とした価格データに変換します．

```
tmp <- rbind(rep(0,ncol(R.521x95)),R.521x95)+1
P.521x95 <- apply(tmp,2,cumprod)
```

以下は Maximum Drawdown 最小化モデルです．変数 x は初期時点における各投資対象に対する投資量です．変数 W は時点 t における資産額，U は時点 t における W の累積最大値，V は時点 t における W の時間を逆方向に見た累積最小値です．Maximum Drawdown は U と V の差の最大値として表現することができます．

```
#### Maximum Drawdown 最小化モデル ####
MinMaxDD <- function(P.d)
{
    Asset <- Set()
    j <- Element(set=Asset)
    Period <- Set()
    t <- Element(set=Period)
    P <- Parameter(index=dprod(t,j),P.d)
    x <- Variable(index=j)
    U <- Variable(index=t)
    V <- Variable(index=t)
    W <- Variable(index=t)
    maxdd <- Variable()
    risk <- Objective()
    risk ~ maxdd
    W[t] == Sum(P[t,j]*x[j],j)
    U[t,t>1] >= U[t-1,t>1]
    U[t] >= W[t]
    V[t-1,t>1] <= V[t,t>1]
    V[t] <= W[t]
    U[t] - V[t] <= maxdd
    Sum(x[j],j) == 1
    x[j] >= 0
}
```

システムを作成して解きます．

```
sys <- System(MinMaxDD, P.521x95)
sol <- solve(sys)
```

解を取得します.

```
x <- as.array(current(sys,x))
```

ここで，確認のために価格データと組入比率から Maximum Drawdown の値を求める関数を定義します. 各期における資産額 W，累積最大値 U，時間を逆方向に見た累積最小値 V も求めます.

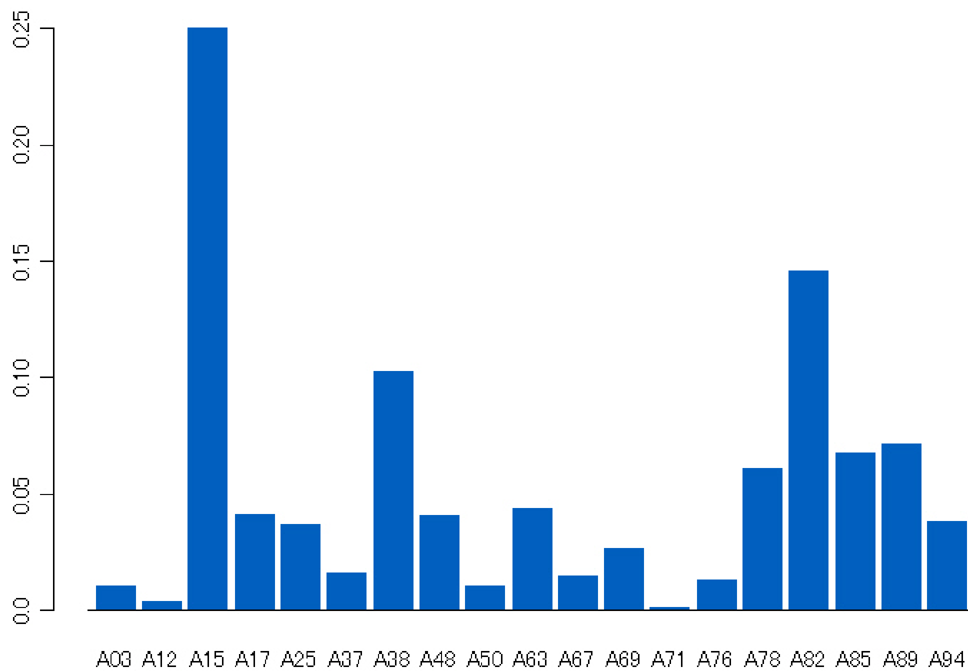
```
calc.MaxDD <- function(P,x)
{
    W <- as.vector(P %*% as.vector(x))
    U <- cummax(W)
    V <- rev(cummin(rev(W)))
    return(list(maxdd=max(U-V),W=W,U=U,V=V))
}
```

Maximum Drawdown を計算します.

```
res <- calc.MaxDD(P.521x95,x)
res$maxdd
```

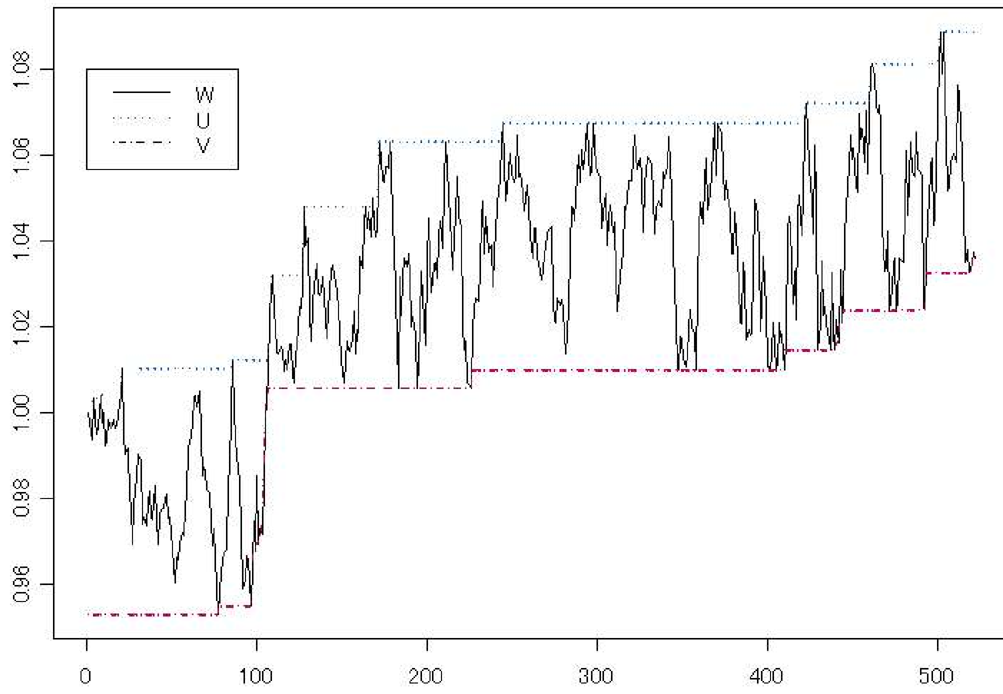
得られた解を棒グラフとして表示します.

```
eps <- 1e-4
barplot(x[x>=eps],names=names(x[x>=eps]))
```



先ほど計算した W, U, V を図示します.

```
m <- cbind(res$W,res$U,res$V)
matplot(1:nrow(m),m,name=colnames(m),type="l")
legend(0,1.08,c("W","U","V"),lty=1:3)
```



W
U
V
1.08
1.06
1.04
1.02
1.00
0.98
0.96
0
100
200
300
400
500

５－７　Sharpe Ratio

ポートフォリオ最適化問題の目的関数として Sharpe Ratio と呼ばれる指標を用いることがあります．Sharpe Ratio は

$$\frac{\sum_{j \in \mathrm{A}} \overline{r}_j x_j - r_f}{\sqrt{\sum_{j \in \mathrm{A}} \sum_{k \in \mathrm{A}} \sigma_{jk} x_j x_k}}$$

と表すことができます．ここで，A は投資対象の集合，x_j は組入比率，$\overline{r}_j$ は平均収益率，σ_{jk} は収益率の分散共分散行列，r_f は安全資産の収益率を表します．Sharpe Ratio 最大化問題は非線形計画問題（目的関数が非線形）となりますが S+NUOPT でそのまま解くことができます．

```
#### Sharpe Ratio 最大化モデル ####
Sharpe <- function(Q.d,rb.d)
{
    Asset <- Set()
    j <- Element(set=Asset)
    k <- Element(set=Asset)
    Q <- Parameter(index=dprod(j,k),Q.d)
    rb <- Parameter(index=j,rb.d)
    x <- Variable(index=j)
    x[j] ~ 0.1
    f <- Objective(type="maximize")
    f ~ (Sum(rb[j]*x[j],j)-0.002)/Sum(Q[j,k]*x[j]*x[k],j,k)^0.5
    Sum(x[j],j) == 1
    x[j] >= 0
}
```

早速解いてみましょう．まずは収益率データから収益率の分散共分散行列と平均収益率を求めます．

```
Q <- var(R.60x200)
rb <- apply(R.60x200,2,mean)
rb <- as.array(rb)
```

システムを作成，求解を行い，解を得ます．

```
sys <- System(Sharpe, Q, rb)
sol <- solve(sys)
x <- as.array(current(sys,x))
```

実は Sharpe Ration 最大化問題は目的関数の分子を $\sum_{j \in \mathrm{A}} \overline{r}_j x_j - r_f = \lambda (>0)$ とおいて，変数を $w_j = x_j / \lambda$ と変換することによって，等価な二次計画問題（目的関数が二次式）に置き換

えることができます．目的関数を一般の非線形ではなく，二次式として表現できるということは，数理計画問題を解く上で大きなメリットとなります．

```
#### Sharpe Ratio 最大化モデル（QP 版） ####
Sharpe.qp <- function(Q.d,rb.d)
{
    Asset <- Set()
    j <- Element(set=Asset)
    k <- Element(set=Asset)
    Q <- Parameter(index=dprod(j,k),Q.d)
    rb <- Parameter(index=j,rb.d)
    w <- Variable(index=j)
    f <- Objective(type="minimize")
    f ~ Sum(Q[j,k]*w[j]*w[k],j,k)
    Sum((rb[j]-0.002)*w[j],j) == 1
    w[j] >= 0
}
```

解いてみましょう．

```
sys <- System(Sharpe.qp, Q, rb)
nuopt.options(eps=1e-10)
sol <- solve(sys)
w <- as.array(current(sys,w))
```

得られた解 w は元の問題の解 x に変換することができます．

```
x.qp <- w/sum(w)
```

得られた解を比較すると同等の解が得られていることがわかります．

```
> eps <- 1e-4
> x[x>=eps]
     A071       A080      A081        A087        A103        A113         A139
0.1317304 0.06223414 0.1040298 0.03068688 0.02559427 0.03757759 0.003409419

     A189      A190        A195     A197         A200
0.235764 0.1985228 0.005258932 0.164983 0.0002087379
> x.qp[x.qp>=eps]
     A071       A080      A081        A087        A103        A113         A139
0.1317538 0.06221308 0.1040643 0.03067362 0.02548632 0.03759601 0.003395785

     A189      A190        A195      A197          A200
0.2357851 0.1985498 0.005192057 0.1649768 0.0003129249
```

６．半正定値計画法の利用

６－１　半正定値行列の取得

相関を持ついくつかのアセットの値動きをモンテカルロ法でシミュレーションする場合などには，アセット間の相関行列を与える必要があります．その際，相関行列は原理的にコレスキー分解が可能であるように正定値である必要があります．しかし，正定値であるという性質は直観的に明らかではありません．S+NUOPT に備わった半正定値計画アルゴリズムを用いれば，与えられた行列（正定値であるとは限らない）に最も近い正定値な行列を算出させることができます．

例えば次のすべて正の要素から成る 10×10 の対称行列は正定値ではありませんのでコレスキー分解することができません．

```
> Cormat.A
     X01  X02 X03 X04 X05 X06 X07 X08 X09 X10
X01 1.00 0.17 0.5 0.2 0.1 0.0 0.2 0.0 0.8 0.7
X02 0.17 1.00 0.1 0.9 0.6 0.4 0.0 0.0 0.0 0.0
X03 0.50 0.10 1.0 0.2 0.0 0.0 0.0 0.2 0.0 0.2
X04 0.20 0.90 0.2 1.0 0.2 0.2 0.2 0.2 0.0 0.0
X05 0.10 0.60 0.0 0.2 1.0 0.4 0.0 0.0 0.0 0.0
X06 0.00 0.40 0.0 0.2 0.4 1.0 0.0 0.0 0.0 0.0
X07 0.20 0.00 0.0 0.2 0.0 0.0 1.0 0.0 0.0 0.0
X08 0.00 0.00 0.2 0.2 0.0 0.0 0.0 1.0 0.0 0.0
X09 0.80 0.00 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
X10 0.70 0.00 0.2 0.0 0.0 0.0 0.0 0.0 0.0 1.0
```

これは，S-PLUS を使って求めた固有値：

```
> eigen(Cormat.A)$values
[1]  2.6192649  2.0903778  1.2534419  1.0818527  1.0000000  0.836797
[7]  0.7030020  0.6042132 -0.0285787 -0.1603709
```

となり，最小固有値（-0.1603709）が負になることからわかります．次は与えられた行列とフロベニウスノルムの意味で近い，正定値行列を求めるための数理計画モデルです．

```
Cormat <- function(A, minEig = 0.001)
{
    N <- Set()
    i <- Element(set = N)
    j <- Element(set = N)
    A <- Parameter(A, index = dprod(i, j))
    minEig <- Parameter(minEig)
    X <- Variable(index = dprod(i, j))
    M <- SymmetricMatrix(dprod(i, j))
    M[i, j, i >= j] ~ X[i, j]
    diffnrm <- Objective(type = minimize)
    diffnrm ~ Sum((X[i, j] - A[i, j]) * (X[i, j] - A[i, j]), i, j, i >= j)
    # 最小固有値が minEig 以上
    M >= minEig
```

```
        # 初期値
        X[i, i] == 1
        # 対称行列であることを要請
        X[i,j,i<j] == X[j,i]
    }
    s <- System(model=Cormat,Cormat.A)
    sol <- solve(s)
    Aopt <- as.array(current(s,X))
```

このようにして算出された Aopt は次のような行列です．各要素は元の行列とそれなりに近い値になっていることにご注意ください．

```
> round(Aopt,2)
     X01  X02  X03  X04  X05  X06  X07   X08   X09  X10
X01 1.00 0.16 0.46 0.19 0.09 0.01 0.18  0.01  0.71 0.63
X02 0.16 1.00 0.10 0.88 0.59 0.40 0.01  0.00  0.01 0.00
X03 0.46 0.10 1.00 0.20 0.00 0.00 0.01  0.20  0.03 0.22
X04 0.19 0.88 0.20 1.00 0.21 0.20 0.20  0.20  0.00 0.00
X05 0.09 0.59 0.00 0.21 1.00 0.40 0.00  0.00  0.01 0.00
X06 0.01 0.40 0.00 0.20 0.40 1.00 0.00  0.00  0.00 0.00
X07 0.18 0.01 0.01 0.20 0.00 0.00 1.00  0.00  0.01 0.01
X08 0.01 0.00 0.20 0.20 0.00 0.00 0.00  1.00 -0.01 0.00
X09 0.71 0.01 0.03 0.00 0.01 0.00 0.01 -0.01  1.00 0.05
X10 0.63 0.00 0.22 0.00 0.00 0.00 0.01  0.00  0.05 1.00
```

これは，Cormat.A にフロベニウスノルムの意味で最も近い，最小固有値が 0.001 以上の行列です．

```
> eigen(Aopt)$values
 [1] 2.580796251 2.028567070 1.236807350 1.067534416 0.968329999
 [6] 0.830140682 0.685764222 0.600045416 0.001012264 0.001002330
```

となることから，Aopt の最小固有値が 0.001 以上であることが確認できます．

６−２　ロバスト最適化

ポートフォリオのリスクを投資対象の収益率の分散共分散行列を用いて計測するマルコビッツモデルにおいて，与えられた分散共分散行列は，しばしば不確実性を伴います．この不確実性に対してロバストな解を得る以下の問題を考えます．
以下の典型的な平均・分散モデルを考えます．ここでは期待収益率とリスクの線形結合（効用関数）を目的関数としています．

$$\max_x \ \mu^T x - \lambda x^T \Sigma x$$
$$s.t. \quad x^T e = 1$$

ここで，μ を期待リターン，Σ をリターンの分散共分散行列，x をポートフォリオの重み，λ をリスク回避係数とします．

分散共分散行列Σに不確実性が伴うとして，以下のロバストポートフォリオ最適化問題を考えます．なおU_Σは，不確実性が伴うことによって取りうるΣに関する行列の集合とします．

$$\max_{x} \left\{ \mu^T x - \lambda \max_{\Sigma \in \mathrm{U}_\Sigma} \left\{ x^T \Sigma x \right\} \right\}$$
$$s.t. \quad x^T e = 1$$

U_Σとして，分散共分散行列Σの各要素が$\underline{\Sigma} \le \Sigma \le \overline{\Sigma}$のような区間を持つとします．このとき(2)は，新たに行列U, Lを用いて以下のような問題に置き換えることができます[3]．なお行列 A，B に対し，$A \bullet B$は，A と B の内積（$\sum_i \sum_j A_{ij} \times B_{ij}$）を表すものとします．

$$\max_{x} \mu^T x - \lambda \left(\overline{\Sigma} \bullet U - \underline{\Sigma} \bullet L \right)$$
$$s.t. \quad x^T e = 1$$
$$\begin{pmatrix} U - L & x \\ x^T & 1 \end{pmatrix} \succeq 0$$
$$U \ge 0, L \ge 0$$

次の S+NUOPT のスクリプトは，分散共分散行列Σの各要素の下限を表す行列 sigL，上限を表す行列 sigU，収益率の期待値 mu，および lambda が与えられているとき，上記 (3) を解くものです．

```
Robust <- function(sigL, sigU, mu, lambda)
{
  Asset <- Set()
  i <- Element(set = Asset)
  j <- Element(set = Asset)
  sigL <- Parameter(sigL, index = dprod(i, j))
  sigU <- Parameter(sigU, index = dprod(i, j))
  lambda <- Parameter(lambda)
  mu <- Parameter(mu, index = i)
  U <- Variable(index = dprod(i, j))
  L <- Variable(index = dprod(i, j))
  x <- Variable(index = i)
  V <- Set(c(as.list(Asset),"dummy")) # Asset に "dummy" を加えた集合 V を
作る
  v <- Element(set = V)
  w <- Element(set = V)
  M <- SymmetricMatrix(dprod(v, w))
  M[i, j] ~ U[i, j] - L[i, j]
  M[j, "dummy"] ~ x[j]
  M["dummy", "dummy"] ~ 1
```

```
    f <- Objective(type = maximize)
    f ~ Sum(mu[i] * x[i], i) - lambda * Sum(sigU[i, j] * U[i, j] - sigL[i, j]
  * L[i, j], i, j)
    M >= 0
    Sum(x[i], i) == 1
    U[i, j, i > j] == U[j, i]
    L[i, j, i > j] == L[j, i]
    U[i, j] >= 0
    L[i, j] >= 0
  }
```

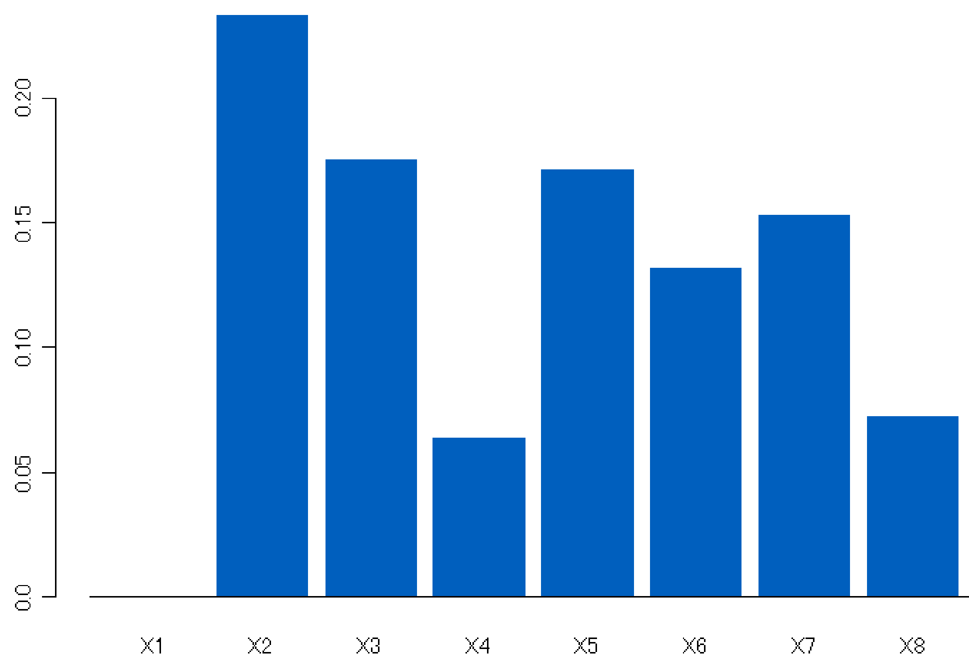
データは以下のように与えられているとします.

```
> Robust.sigU    # 分散共分散の要素の値の上限
     X1   X2   X3   X4    X5    X6    X7    X8
X1  3.0  1.0  2.5 -0.9  1.00  -0.4  2.50   2.0
X2  1.0  4.5 -1.4  1.2  0.50  -1.1  0.50   2.6
X3  2.5 -1.4  6.0 -0.5  1.20   1.0  0.40  -0.1
X4 -1.0  1.2 -0.5  6.5  1.60   0.5  1.30  -0.8
X5  1.0  0.5  1.2  1.6  5.50  -1.3  0.16  -1.9
X6 -0.4 -1.1  1.0  0.5 -1.30  11.0 -0.90   0.6
X7  2.5  0.5  0.4  1.3  0.16  -0.9  6.50  -1.2
X8  2.0  2.6 -0.1 -0.8 -1.90   0.6 -1.20  13.5

> Robust.sigL    # 分散共分散の要素の値の下限
     X1   X2     X3     X4    X5   X6    X7     X8
X1  1.9 -1.4 -1.000 -1.000  1.00 -0.5 -1.00   0.10
X2 -1.4  3.5 -1.500  0.500 -1.20 -1.2  0.40   0.60
X3 -1.0 -1.5  5.000 -1.125  1.10  0.9  0.30  -0.20
X4 -1.0  0.5 -1.125  6.000  1.50  0.4  1.20  -0.90
X5  1.0 -1.2  1.100  1.500  4.50 -1.4  0.15  -2.00
X6 -0.5 -1.2  0.900  0.400 -1.40  9.0 -1.00   0.50
X7 -1.0  0.4  0.300  1.200  0.15 -1.0  5.50  -1.25
X8  0.1  0.6 -0.200 -0.900 -2.00  0.5 -1.25  11.50

> Robust.mu      # 収益率の期待値
  X1  X2  X3  X4  X5  X6  X7  X8
 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
```

次のようにして解きます.

```
s <- System(model=Robust,Robust.sigL,Robust.sigU,Robust.mu,1.0)
# 求解
sol <- solve(s)
# 図示
xopt <- as.array(current(s,x))
barplot(names=rownames(xopt),xopt)
```

次のようなポートフォリオが得られました.



７．非線形フィッティング

非線形なモデル関数をデータに合わせこむ問題は次の形の非線形な数理計画問題として一般的に記述することができます.

変数：$a_1, \cdots, a_n$　（モデルパラメータ）

最小化：$\sum_i e_i{}^2$　（誤差の二乗和）

制約：$e_i = f(x_i; a_1, \cdots, a_n) - y_i$, $i \in$ ｛観測点｝（誤差の定義）

$g(a_1, \cdots, a_n) \le 0$　（パラメータについての制約）

ここで，$f(x; a_1, \cdots, a_n)$ はパラメータに対して線形あるいは非線形な式として定義されているモデル式，（$a_1, \cdots, a_n$ ：パラメータ，x ：変数）で，m 個の観測点 x_i （$i \in$ ｛観測点｝）において，観測値 y_i が与えられているとします．$g(a_1, \cdots, a_n) \le 0$ はパラメータ値に関する制限を記述した制約式です.

重回帰はこの最も基本的な形で，$f(x; a_1, \cdots, a_n)$ が線形関数である場合に対応します.

$f(x; a_1, \cdots, a_n)$ が線形関数でもパラメータの値に制限（制約）がある場合にもこの数理計画問題を解く必要があります.

７−１　イールドカーブのフィッティング

金融工学に表れるフィッティングで最も多く現れるのは，スポットレート（割引債の利回り）を，観測された利付債の現在価格から推定するイールドカーブフィッティングです.

償還期間 t における額面価格 100，クーポンレート 1% の利付債の理論現在価格 $S(t)$ は，スポットーレート r_t を用いて以下のように表すことができます.

$$S(t) = \frac{100}{(1 + 0.01 \cdot r_t)^t} + \sum_{k=1}^{t} \frac{1}{(1 + 0.01 \cdot r_k)^k}$$

償還期間 T_i の利付債の現在価格 S_i が以下のように観測されたとします.

```
> Yield.term.data
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
 1 1 1 1 1 1 2 2 2  2  2  2  3  3  3  3  3  3  4  4  4  4  4  4  5  5  5  5

29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
 5  5  6  6  6  6  6  6  7  7  7  7  7  7  8  8  8  8  8  8  9  9  9  9  9

54 55 56 57 58 59 60
 9 10 10 10 10 10 10
> Yield.price.data
     1      2     3      4     5      6      7     8     9    10     11
100.39 102.15 99.24 101.32 99.72 101.51 100.62 99.34 98.27 98.31 100.97
```

```
    12     13     14    15    16    17    18    19    20    21    22    23    24
101.73 99.78 100.47 98.19 99.55 99.79 98.4 96.6 96.93 96.8 94.91 96.28 95.33

   25    26    27    28    29    30    31    32    33    34    35    36    37
95.13 93.5 93.42 91.56 92.67 96.28 89.66 89.46 91.21 91.42 93.32 89.76 90.18

   38    39    40    41    42    43    44    45    46    47    48    49    50
88.58 87.41 90.33 87.5 87.66 86.06 85.55 84.74 88.78 86.79 88.76 84.95 84.31

   51    52    53    54    55    56    57    58    59    60
87.24 84.73 83.76 84.02 81.75 84.46 82.51 85.42 81.45 85.36
```

次の数理計画モデルは各観測点iにおける理論現在価格$S\left(T_i\right)$と観測された現在価格S_iの差の二乗和：

$$\sum_i \left\{ S\left(T_i\right) - S_i \right\}^2$$

を最小にするスポットーレートr_tを求めます.

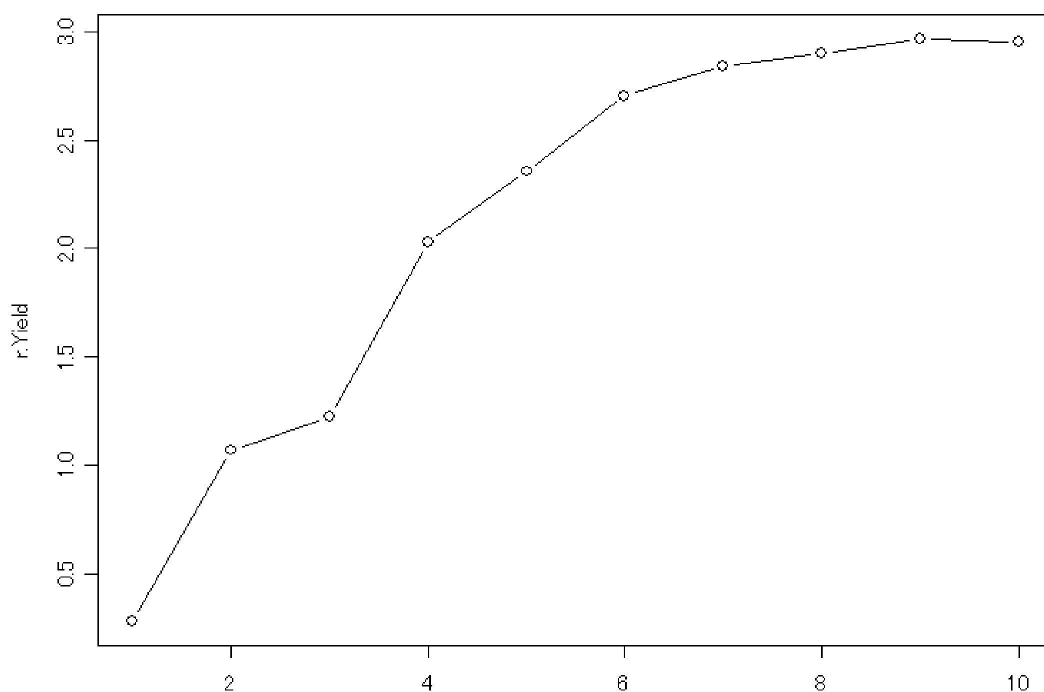
```
#### イールドカーブ推定モデル ####
Yield <- function(telem.d, tvalue.d, Svalue.d)
{
    Term <- Set()
    Point <- Set()
    t <- Element(set=Term)
    i <- Element(set=Point)
    r <- Variable(index=t)
    telem <- Parameter(index=t,telem.d)
    tvalue <- Parameter(index=i,tvalue.d)
    Svalue <- Parameter(index=i,Svalue.d)
    d <- Expression(index=t)
    d[t] ~ 1 / (1+0.01*r[t])^telem[t]
    S <- Expression(index=i)
    S[i] ~ Sum(d[t],t,t<=tvalue[i]) +
Sum(100/(1+0.01*r[t])^telem[t],t,t==tvalue[i])
    diff <- Expression(index=i)
    diff[i] ~ S[i] - Svalue[i]
    err <- Objective()
    err ~ Sum(diff[i]*diff[i], i)
    0 <= r[t]
}
```

次はデータを与えて解きます．データは各債券の償還期間（tvalue.d）および市場で観測された価格（Svalue）となります．telem.d は$S\left(t\right)$の各項のべきの値のベクトルです.

解いてみます.

```
# 展開
sys <- System(Yield, Yield.telem, Yield.term, Yield.price)
# 実行
sol <- solve(sys)
# 解の取得
r <- as.array(current(sys,r))
# 図示
plot(r,type="b")
```

以下のイールドカーブが得られます.



7-2　格付け推移行列の推定

格付け（Rating）とは，企業の発行する社債の元本，利息の支払い能力をランク形式で表示したものです．格付け会社は独自の調査結果のもと，A，B，C などの記号を用いて対象社債のリスク度合いを示します．格付け推移行列とは，ある格付け評価を受けている企業が，一定期間後にどのような格付けとなるかについての確率を表す行列のことをいいます．ここでは，1 年分の格付け推移行列から 1 ヵ月分の格付け推移行列を推定する問題を扱います．

```
#### 格付け推移行列推定モデル ####
Rating <- function(q0.d)
```

```
{
    Rating <- Set()
    i <- Element(set=Rating)
    j <- Element(set=Rating)
    k <- Element(set=Rating)
    q0 <- Parameter(index=dprod(i,j),q0.d)
    q <- Variable(index=dprod(i,j))
    q2 <- Expression(index=dprod(i,j))
    q4 <- Expression(index=dprod(i,j))
    q8 <- Expression(index=dprod(i,j))
    q12 <- Expression(index=dprod(i,j))
    q2[i,j] ~ Sum(q[i,k]*q[k,j], k)
    q4[i,j] ~ Sum(q2[i,k]*q2[k,j], k)
    q8[i,j] ~ Sum(q4[i,k]*q4[k,j], k)
    q12[i,j] ~ Sum(q8[i,k]*q4[k,j], k)
    diff <- Expression(index=dprod(i,j))
    diff[i,j] ~ q0[i,j] - q12[i,j]
    diffnrm <- Objective()
    diffnrm ~ Sum(diff[i,j]^2, i, j)
    Sum(q[i,j], j) == 1
    0 <= q[i,i] <= 1
    0 <= q[i,j, i!=j] <= 0.05
    q[i,i] ~ 0.9
}
# 展開
sys <- System(Rating, Rating.Q0)
# 実行
sol <- solve(sys)
# 解の取得
q <- as.array(current(sys,q))
# 確認
q12 <- diag(1,nrow(q))
dimnames(q12) <- dimnames(q)
for(i in 1:12) q12 <- q12 %*% q
# 表示
r.order <- order(rownames(Rating.Q0)) # 格付けの順番を定義
```

得られた 1 か月分の格付け推移行列を表示します.

```
> round(q[r.order,r.order],digits=4)
       AAA     AA      A    BBB     BB      B    CCC     CC      C
AAA 0.9970 0.0030 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 AA 0.0031 0.9946 0.0023 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
  A 0.0001 0.0038 0.9945 0.0016 0.0000 0.0000 0.0000 0.0000 0.0000
BBB 0.0000 0.0000 0.0031 0.9952 0.0014 0.0003 0.0000 0.0000 0.0000
 BB 0.0000 0.0000 0.0000 0.0098 0.9885 0.0004 0.0013 0.0000 0.0000
  B 0.0000 0.0000 0.0000 0.0000 0.0069 0.9889 0.0036 0.0006 0.0000
CCC 0.0000 0.0000 0.0000 0.0000 0.0000 0.0025 0.9970 0.0005 0.0000
```

```
  CC 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0007 0.9972 0.0021
   C 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0023 0.9977
```

1 か月分の格付け推移行列を 12 乗，つまり 1 年分に変換した値を示します.

```
> round(q12[r.order,r.order],digits=4)
       AAA     AA      A    BBB     BB      B    CCC     CC      C
AAA 0.9649 0.0346 0.0004 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 AA 0.0355 0.9381 0.0261 0.0002 0.0000 0.0000 0.0000 0.0000 0.0000
  A 0.0014 0.0432 0.9364 0.0186 0.0003 0.0001 0.0000 0.0000 0.0000
BBB 0.0000 0.0007 0.0348 0.9454 0.0152 0.0036 0.0002 0.0000 0.0000
 BB 0.0000 0.0000 0.0018 0.1071 0.8716 0.0045 0.0149 0.0001 0.0000
  B 0.0000 0.0000 0.0000 0.0041 0.0734 0.8752 0.0400 0.0071 0.0001
CCC 0.0000 0.0000 0.0000 0.0000 0.0010 0.0274 0.9650 0.0064 0.0001
 CC 0.0000 0.0000 0.0000 0.0000 0.0000 0.0001 0.0082 0.9674 0.0241
  C 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0001 0.0265 0.9733
```

与えられた 1 年分の格付け推移行列を表示します.

```
> round(Rating.Q0,digits=4)
       AAA     AA      A    BBB     BB      B    CCC     CC      C
AAA 0.9651 0.0349 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 AA 0.0356 0.9382 0.0262 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
  A 0.0014 0.0433 0.9364 0.0186 0.0003 0.0000 0.0000 0.0000 0.0000
BBB 0.0000 0.0000 0.0352 0.9456 0.0154 0.0038 0.0000 0.0000 0.0000
 BB 0.0000 0.0000 0.0000 0.1078 0.8720 0.0049 0.0153 0.0000 0.0000
  B 0.0000 0.0000 0.0000 0.0000 0.0747 0.8762 0.0410 0.0081 0.0000
CCC 0.0000 0.0000 0.0000 0.0000 0.0000 0.0278 0.9654 0.0068 0.0000
 CC 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0083 0.9675 0.0242
  C 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0266 0.9734
```

与えられた1 年分の格付け推移行列と推定した1 か月分の格付け推移行列を 1 年分に変換した行列を見比べ，1 ヵ月分の格付け推移行列が適切に推定できていることを確かめることができます.

7-3 ロジスティック回帰

S+NUOPT を用いてロジスティック回帰を行うモデルを作成することもできます.

```
#### ロジスティック回帰モデル ####
LogReg <- function(X.d,t.d)
{
    M <- Set()
    i <- Element(set=M)
    L <- Set()
    l <- Element(set = L)
    X <- Parameter(index=dprod(l,i),X.d)
    t <- Parameter(index=l,t.d)
```

```
    a <- Variable(index=i)
    a0 <- Variable()
    y <- Expression(index=l)
    y[l] ~ exp(Sum(a[i]*X[l,i],i)+a0) / (1+exp(Sum(a[i]*X[l,i],i)+a0))
    mle <- Objective(type="maximize")
    mle ~ Sum(t[l]*log(y[l])+(1-t[l])*log(1-y[l]),l)
}
```

ここでは，例として Iris virsinica と Iris versicolor の判別分析を扱います．

```
# 展開
sys <- System(LogReg,LogReg.X,LogReg.t)
# 実行
sol <- solve(sys)
# 解の取得
a <- as.array(current(sys,a))
a0 <- as.array(current(sys,a0))
```

得られた解について，学習データとは別のテストデータで予測力を評価します．

```
# 予測
eta <- a0 + LogReg.test.X %*% as.vector(a)
p <- exp(eta)/(1+exp(eta))
res <- cbind(round(LogReg.test.t,0),round(p,0),eta)
dimnames(res)[[2]] <- c("answer","predict","eta")
res
```

S+NUOPT には，数理計画モデルを自在に設計できるというメリットがあります．例えば，推定したいパラメータに何らかの制約を課して回帰を行ったり，ロジット関数を 2 次式で表して予測力向上のために 2 次の係数行列に半正定値制約を課したりすることができます．

８．最適化ソルバーNUOPT

S+NUOPT は汎用の最適化エンジン NUOPT と接続しています．System オブジェクトを引数として solve()コマンドをコールすると NUOPT が起動します．System オブジェクトには問題のタイプ（制約式や目的関数の次数や変数の種類）の情報が含まれているため，NUOPT はその情報に基づいて適切な最適化アルゴリズムの選択やそのパラメータ設定を自動的に行うことができます．自動設定で通常は問題ありませんが，実際規模の問題や複雑な問題においては，手動でアルゴリズムの選択やパラメータの設定を行う方が有利な場合はございます．セクション 8-1 はそうした場合に nuopt.options()という関数を使って NUOPT の動作をカスタマイズする方法について述べています．続いて 8-2 では S+NUOPT のご利用上によく現れるエラーの意味や回避方法について述べています．

S+NUOPT は最適化ソルバーNUOPT にモデリング言語を介さず直接データを与えて最適化を行うための関数 solveQP を備えています．セクション 8-3 は solveQP の使い方について解説しています．この方法は最適化問題を記述する生データである制約式を記述する係数行列やヘッセ行列を S-PLUS 上のデータとして既にお持ちの場合に有効です．

８－１　nuopt.options()を使って NUOPT をカスタマイズする

NUOPT には動作をカスタマイズするための様々なチューニングパラメータがあります．以下の表がその一覧を示しています．

名前	デフォルト値	設定可能な値	設定例
method	"auto"	“auto”,”simplex”,”asqp”, “wcsp”,”higher”,”lipm” ,”tipm”,”lepm”,”tepm”,”lsqp”,”tsqp”,”line”,”trust”	"simplex"
scaling	"on"	“on”,”off”	"off"
mipfeasout	"on"	“on”,”off”	"off"
addToCutoff	-10	double	0.99
cutoff	1.00E+50	double	4.5
eps	-10	double	1.00E-4
epsint	1.00E-04	double	1.00E-8
maxitn	150	int	300
maxnod	-1	int	500000
maxtim	-1(except wcsp),5(wcsp)	int	60
told	1.00E-06	double	1.00E-8
tolx	1.00E-08	double	1.00E-10
maxintsol	-1	int	1
maxmem	-10	int	-100
wcspTryCount	1	int	5
wcspRandomSeed	1	int	23

これらのパラメータは nuopt.options コマンドから次のようにして設定することができます．

```
nuopt.options(method=”simplex”) # 最適化アルゴリズムを“simplex”に
nuopt.options(eps=1.0e-10) #  停止条件をより小さめに（目安は 1.0e-8)
```

```
nuopt.options(scaling=”off”,maxitn=300) # スケーリングをし，反復回数の上限を設定する
```

nuopt.options はカンマ（,）で区切られた “名前=値”の列を引数としてとります．現状のパラメータ設定の内容は次のように入力することによって表示されます．

```
nuopt.options()  # seeing current settings of parameters
```

現状のパラメータ設定を S-PLUS データに保存するには次のようにします

```
cops <- nuopt.options()  # 現状のパラメータ設定をオブジェクト cops に保存
```

保存したパラメータを読み込むには次のようにします．

```
nuopt.options(cops)  # cops に保存した内容を設定する
```

パラメータ設定は S+NUOPT セッションの間中保存されることにご注意ください．デフォルトの設定は nuopt.options()に引数 NA を与えることによって得られます．したがって，次のようにするとデフォルトの設定に戻ります．

```
nuopt.options(nuopt.options(NA))  # デフォルトのパラメータに戻す
```

デフォルトの設定は通常問題が起きないように選ばれていますが，計算のパフォーマンスよりも安定性を主眼として設定されております．そのため，ユーザーが問題に関する情報を元に手動で調整することによってより良い結果を得ることが可能です．次からのセクションではその典型的なケースについて述べます．

8-1-1.特殊な大規模二次計画問題を高速に解く

ポートフォリオ最適化問題は大規模二次計画問題（制約は線形で，目的関数は二次関数）となります．これらの二次計画問題は変数に比べ，制約式の数が少ないという特徴をもつものが多く見られます．そのような問題はアルゴリズム”asqp”（有効制約法）が有利です．次のように設定してみてください．より速く最適化が終了する可能性があります．

```
nuopt.options(method=”asqp”)   # アルゴリズム asqp を用いる
```

8-1-2.線形計画問題・二次計画問題をより高精度で解く

特に設定を行わなければ，NUOPT は線形計画問題（線形な目的関数と線形な制約式からなる問題）と二次計画問題（制約は線形で，目的関数は二次関数）を，内点法によって解きます．大規模問題に対しても高速な求解ができるという意味でこの設定は一般的に有効ですが，線形計画問題は”simplex”（単体法），二次計画問題は”asqp”（有効制約法）を用いる方が，問題に関する有益な情報が得られる場合があります．特に問題が実行不可能に近い（制約の充足が不可能か非常に難しい）場合，また問題のスケールが悪い（変数や制約式の大きさに非常にばらつきがある）場合には内点法の動作は不安定となり，反復回数オーバー（NUOPT エラーコード 10）が起きることがあります．そのような場合には，

```
nuopt.options(method=”simplex”)   # 線形計画問題の場合
nuopt.options(method=”asqp”)      # 二次計画問題の場合
```

のように設定してみてください．変数や制約の数が一万個を超えるなどの場合には若干時間を所要しますが，これらの手法は内点法にくらべて頑健で，より正確な解を与えます．

8-1-3.計算機資源の利用を制限する

整数変数を含む問題を与えた場合，NUOPT は分枝限定法を用いて解きます．一般に分枝限

定法は計算時間およびメモリを所要するアルゴリズムです．大規模な問題など，難しい問題では次のようにして計算時間およびメモリーの利用を制限することが時に有効です．

```
nuopt.options(maxtim=600)    # 計算時間を 10 分（ 6　秒）に制限
nuopt.options(maxmem=10000) # メモリの利用量を　1G byte に制限
nuopt.options(maxmem=-100)  # システムの残りメモリが 100Mbte をしたまわ
るまで計算を行う
```

これらの上限の一つでも満たすことができなかった場合には，NUOPT は時間制限オーバー，あるいはメモリーオーバーと出力して計算を停止，そのときまでに見つかった最良の解を返します．

アルゴリズム”wcsp”を起動している場合には，時間制限が必須です．これは”wcsp”　アルゴリズムが最適性を判定することができないためです．アルゴリズム”wcsp”の利用の方法とパラメータ wcspRandomSeed, wcspTryCount の設定についてはセクション 3-3 をご覧ください．．

８-１-４.分枝限定法のチューニング

整数変数を含む数理計画問題を解く際に起動される分枝限定法を次のパラメータで制御することができます．

```
nuopt.options(maxnod=100000)    # ノード数を 100000 以下に制限する
nuopt.options(maxintsol=1)  # 実行可能解を一つ見つけたら止まる
nuopt.options(addToCutoff=0.99) # 目的関数についての最適性の判定条件を
0.99 だけ緩める
```

maxintsol が与えられている場合，その個数の実行可能解が発見された段階で停止します．addToCutoff が与えられている場合，分枝限定法の反復の中で，addToCutoff よりも改善度の少ない解は無視します．もし目的関数が整数であることがわかっているのであれば，addtoCutoff を 0.99（１以下）に設定してもアルゴリズムは正確性を失いません．

８-１-５.非線形計画法，半正定値計画法の安定化のためのチューニング

これまで述べた例にあるように，S+NUOPT は非線形計画問題，半正定値計画問題を特にパラメータの設定なく解くことができますが，難しい問題になるとソルバーの計算が失敗し，解の精度が落ちたり，エラーを起こしたりすることがあります．計算の失敗の理由としては次のようなものがあり，対策としてこれらのいずれかを除去することが有力です．

1. 問題が実行不可能に近い
   制約条件が厳しすぎる場合に，内点法のふるまいは不安定になります．
2. スケールのばらつきが大きい
   制約，および目的関数はスケーリングされているのが好ましいといえます．すなわち絶対値において１程度の値に揃っているのが望まれる状況です．典型的な悪条件としては，変数や制約式の一部が他の変数や制約式と比べて突出して（1000 あるいは 1/1000 程度の）大きな・あるいは小さな絶対値を持っていることです．
3. 問題が凸ではない
   実行可能領域が凸であってかつ目的関数が凸関数である場合，その最適化問題は凸な問題であると呼ばれます．凸な問題は解きやすく比較的解析が容易なクラスに属します．しかし，その保障がない場合，問題は凸なケースに比べて非常に難しくなります．凸なケースでは局所最適解が大域的最適化である保障がありましたが，凸でないケースではその保障がなく，複数個の局所的最適解が存在することが解析を難しくしています．NUOPT は凸でない問題を扱うことができますが，

そのアルゴリズムはいずれか一つの局所的な最適解を求めるためのもので，大域的な最適解を常に出力できるものではありません．

4. 問題の非線形性が高い
   もし，問題が exp(x)や 1/x，高次の多項式を含んでいたら，その分問題は線形な問題から遠ざかり，扱いが難しくなります．最適化ソルバーは内部で非線形モデルを線形に近似するということを行いますので，その近似が正確であるほど性質は良好になります．

上記を解消するため，以下の手段が一般的に有効です．

A) 問題の実行可能性を調べる
   制約を緩めたり，スラック変数を導入して振舞が変化するかを見ることによって，制約がどの程度厳しいか，制約の厳しさが振舞いにどの程度影響しているかを見ることができます．
B) スケールを変えてみる
   手動で変数あるいは制約式をスケールしてみます．また，NUOPT が自動的に行っているスケーリングを，次のようにしてやらせないで試してみます．
   ```
   nuopt.options(scaling=”off”) # 自動スケーリングを行わない
   ```
C) 変数に初期値を与える
   変数に 2-3-1 において説明した方法で初期値を与えます．特に 1/x のように，関数が極をもち，その周辺では発散するような場合には初期値の工夫が有効です．
D) アルゴリズムを変更して試してみる
   NUOPT にはデフォルトでは起動されない，多数のアルゴリズムが実装されています．手動でそれらの一つを起動するように設定される方が良い結果をもたらすことはあり得ます．

	凸な問題用	*非凸な問題用*
非線形	"lipm","line","lepm",”lsqp”	"tipm","trust","tepm","lbfgs",”tsqp”
半正定値	"lsdp"(linear SDP only) ,"csqp"	"trsdp","qnsdp"

   特にパラメータを指定しない場合には，NUOPT は問題が線形であるか非線形であるか，半正定値性の制約を含むかという情報を元にアルゴリズムの選択を行います．問題が凸であるかどうかという情報はアルゴリズムの選択上重要ですが，問題が凸であるかどうかはわからないので，NUOPT は非線形計画問題であれば常に非凸な状況に対応できる”tipm”を用いて問題を解きます．半正定値性の制約を含み，問題がすべて線形であるならば”lsdp”，そうでなければ”trsdp”を選択します．例えば，ユーザーが扱っている問題が凸であることがわかっているのであれば，アルゴリズムとして”lipm”を用いるのは良い選択です．
   ```
   nuopt.options(method=”lipm”) # lipm を方法として採用する
   ```

   なぜなら，凸性を仮定しない分，”lipm”は”tipm”よりも高速であるためです．

８-１-６.ヒープメモリをクリアする

S+NUOPT は最適化のためにヒープメモリを使用します．セッションの間に最適化を多数回繰り返して，ヒープメモリが積みあがってくると，NUOPT はメモリーエラー（エラーコード 1 あるいは 8）で終わってしまうようになることがあります．そのような場合には，次のコマンドでヒープメモリをクリアしてください．

```
module(nuopt,unload=”T”)      # アンロード（解放が行われる）
module(nuopt)                 # ロード（もう一度開始）
```

８-２　エラーメッセージ

この章では S+NUOPT を使うにあたって出会うエラーの意味について解説します．解決の方法も一部示しています．

８-２-１.モデリング言語解釈部からのエラー

次はモデリング言語の解釈部から出される典型的なエラーの表です．

メッセージ	意味
Problem: Syntax error: illegal "," after ..	index の右辺に ‘dprod’ が抜けています
Error: Illegal use of summary function on Simple objects	‘Sum’を使うべきところで’sum’ が用いられています
Error: SIMPLE object not created in this SPLUS session	前回以前のセッションで作成された，S+NUOPT 固有のオブジェクトなので内容がわからないオブジェクトです．
<<SIMPLE 1>> Infeasible bound for variable x （defining infeasible bound …）	変数の上下限が矛盾しています．
<<SIMPLE 67>> Index error in reference of <objname> with no index but should be with index of dimension 1	添え字があると定義しているオブジェクトに添え字を与えていません．
<<SIMPLE 67>> Index error in reference of <objname> scalar but with index of dimension 1.	添え字がなしで定義しているオブジェクトに添え字を与えています．
<<SIMPLE 82>> Subscript <elem> of <objname> out of range.	インデックスの範囲が誤っています．
<<SIMPLE 168>> Objective can only be assigned once.	目的関数に複数回代入しています．
<<SIMPLE 214>> constraint#1 reduce to … (never satisfied) [/ (always satisfied)] <<SIMPLE 216>> Trivial and Infeasible constraint appeared.	常に満たされないか，常に満たされる（意味のない）制約が定義されています．

Problem: Syntax error: illegal "," after ..

このメッセージは複数の添え字を持つオブジェクトの宣言で，”index=”の右辺に dprod を忘れると出力されます．例えば

```
S <- Set()
```

```
U <- Set()
x <- Variable(index=(S,U)) # error
```
あるいは
```
S <- Set()
U <- Set()
i <- Element(index=S)
j <- Element(index=U)
x <- Variable(index=(i,j))  # error
```
のようにすると，このエラーが出ます．複数の添え字を持つオブジェクトの宣言にはかならず”dprod”が必要になります．
```
x <- Variable(index=dprod(i,j))
```

Error: Illegal use of summary function on Simple objects
モデリング言語の記述で，変数や式の和を定義するときには”sum”ではなく，”Sum”を用いなければなりません．このメッセージは”sum”が誤って用いられた以下のような場合に出現します．
```
S <- Set()
i <- Element(set=S)
x <- Variable(index=i)
sum(x[i],i) >= 1   # error,  should be “Sum(x[i],i) >= 1”
```

Error: SIMPLE object created in previous SPLUS session
System, Variable, Parameter, Set, ElementなどのオブジェクトのS-PLUSのセッションをまたがっては保存されません．このメッセージは以前のセッションで作成した，既に内容が失われているオブジェクトを参照した場合に出現します．重要な最適化の結果などは，セクション 4-4 で述べたように as.array や as.list を用いて S-PLUS オブジェクトの形で蓄えておくことが必要です．

<<SIMPLE 1>> Infeasible bound for variable
変数の上下限はすべての設定の共通部分をとります．このメッセージはその際に矛盾が起きており，共通部分が存在しないことを示します．
```
x <- Variable()
x >= 7
x <= 5          # error (conflict)
```

<<SIMPLE 67>> Index error in reference of <objname>
<<SIMPLE 82>> Subscript <elem> of <objname> out of range
このメッセージは添え字の使い方の矛盾を指摘するものです．
```
x <- Variable()
x[3] >= 1         #  エラー（x は添え字をつけてはならない）
S <- Set(1:3)
y <- Variable(index=S)
y <= 2          # エラー（y は添え字をつけねばならない）
y[4] >= 5       # エラー（添え字の範囲が違う）
```

<<SIMPLE 168>> Objective can only be assigned once.
目的関数は二度以上の代入が行われた場合に出力されます．
```
f <- Objective(type=minimize)
x <- Variable()
```

```
y <- Variable()
f ~ x+y
f ~ x-y    # エラー（二度以上の代入を行おうとしています）
```

<<SIMPLE 214>> constraint#1 reduce to ... (never satisfied) [/ (always satisfied)]
<<SIMPLE 216>> Trivial and Infeasible constraint appeared.
定数の初期化忘れのミスなどで，常に満たすことができない制約式や意味のない制約式を定義してしまうことはあります．このメッセージはこのような制約式が現れたことを示しています．

```
S <- Set(1:3)
i <- Element(set=S)
x <- Variable(index=i)
a <- Parameter(index=i)  # a の中身は与えられていない（零だと解釈する）
Sum(a[i]*x[i],i) == 1    # 常に満たすことができない制約（左辺が零）
Sum(a[i]*x[i],i) <= 0    # 意味のない制約（左辺が零）
```

８-２-２.NUOPT が出力するエラー

問題の解釈が完了した後に最適化の過程でエラーを発見すると，次のようなメッセージが現れます．

```
  <<SIMPLE 193>> Error in solve():
      "<<NUOPT 10>> IPM iteration limit exceeded."    # ソルバーエラーの内容
```

エラーコードは，solve()の戻り値のリストの中の"errorCode"というアイテムに格納されていますので取得することができます．次の表にはメッセージの意味と解説と前項での関連項目について記述しています．

コード	メッセージ	意味
1	memory error in preprocessing.	前処理のフェーズでメモリーエラーが起きました（8-1-6 を参照）.
2	infeasible(linear constraints and variable bounds)	線形制約と変数の上下限が矛盾しています.
3	no variables in this model.	変数が定義されていません.
4	no functions in this model.	制約も目的関数も定義されていません.
5	infeasible variable bounds.	変数の上下限が矛盾しています.
7	internal error.	内部エラー. (nuopt-support@msi.co.jp に連絡をお願いします)
8	memory error in	最適化アルゴリズム適用のフェーズでメモリーエラ

	optimization phase.	ーが起きました（8-1-6 を参照).
9	step reduction limit exceeded.	直線探索を行うアルゴリズムで，ステップサイズの設定に失敗し，降下方向が発見できませんでした．数値的なエラーか，問題が凸でない可能性があります（8-1-5 を参照).
10	IPM iteration limit exceeded.	内点法の反復回数が上限を超えました．（8-1-2 および 8-1-5 を参照）
11	infeasible	問題が実行不可能です（制約を満たす変数が存在しません）
13	unbounded.	有界な問題ではありません (目的関数を任意に小さくあるいは大きくできる実行可能解の列があります)
14	integrality is violated.	出力された解においては整数変数が整数値以外の値をとっています（整数変数を含む問題に対して，強制的に tipm や line などの内点法を適用した場合に現れます．これらのアルゴリズムは整数性を意識しないのでこのような結果になります).
15	simplex misapplied to QP.	二次計画問題に対してアルゴリズム”simplex”を指定しました．
15	simplex/asqp misapplied to NLP.	一般の非線形計画問題に対して，アルゴリズム”simplex”，”asqp”を指定しました．
16	Infeasible MIP.	制約式と変数に対する整数条件を満たす解が存在しないことがわかりました（整数条件を緩めれば解は存在します)
17	B & B node limit reached (with feas.sol.).	実行可能解が見つかりましたが，分枝限定法のノード数（子問題数）がパラメータ maxnod で指定された個数を超えたので停止しました．見つかった解が最適であるかどうかはわかりません(8-1-4 参照).
18	MIP iteration failed (with feas.sol.).	実行可能解が見つかりましたが，分枝限定法のプロセスの途中で数値的な問題が発生して停止しました．見つかった解が最適であるかどうかはわかりません．
19	B & B node limit reached (no feas.sol.).	分枝限定法のノード数（子問題数）がパラメータ maxnod で指定された個数を超えました．実行可能解は見つけられませんでした(8-1-4 参照).

20	MIP iter. failed (no feas.sol.).	分枝限定法のプロセスが数値的問題によって失敗し，実行可能解も見つけられませんでした．
21	B & B itr. timeout (with feas.sol.).	分枝限定法が，パラメータ maxtim で与えた時間上限を超過したため終了しました．実行可能解は見つかっていますが，最適であるかどうかはわかりません(8-1-3 参照).
22	B & B itr. timeout (no feas.sol.).	分枝限定法が，パラメータ maxtim で与えた時間上限を超過したため終了しました．実行可能解も見つけられませんでした（8-1-3 参照）
27	SIMPLEX iteration limit exceeded.	単体法の反復回数が上限値を超えました．
28	higher-order method is only for LP.	アルゴリズム"higher"を非線形計画問題に指定しようとしました．
29	iteration diverged.	計算が発散しました（目的関数を任意に小さくあるいは大きくできる実行可能解の列があると思われます）
33	Bound violated.	内部的に施していた自動スケーリングを戻したら変数の上下限を満たしていないことがわかりました．変数や制約式のスケールのばらつきがかなり大きいことが想像されます．scaling="off"として自動スケーリングを行わないか，入力前に問題をスケーリングすることをお勧めします（8-1-5 参照）．
34	Bound and Constraint violated.	33 と同じですが変数の上下限とあわせて制約式の上下限も満たしていませんでした（8-1-5 参照).
35	Constraint violated.	33 と同じですが，制約式の上下限を満たせていないことがわかりました（8-1-5 参照）．
36	Equality constraint violated.	33 と同じですが，制約式の上下限を満たせていないことがわかりました（8-1-5 参照）．
37	B&B terminated with given # of feas.sol.	パラメータ maxintsol で指定されただけの個数の整数解が求められましたので計算を終了しました(8-1-4 参照).
38	dual infeasible.	分枝限定法の途中で解く部分問題が実行不可能と判定されました．変数や制約式のスケールのばらつきがかなり大きく，数値的条件が悪いことが考えられます．
39	IPM iteration timeout.	内点法の反復がパラメータ maxitn（デフォルト 150

		回）で設定した上限を超えました(8-1-5 参照)．数値的条件が悪いことが考えられます．
40	SQP iteration limit exceeded.	逐次二次計画法の反復回数が上限を超えました(8-1-5 参照)．数値的条件が悪いことが考えられます．
41	SQP internal error.	逐次二次計画法の内部エラーです (nuopt-support@msi.co.jp までご連絡ください)．
43	B&B memory error (with feas.sol.).	実行可能解を求めることができましたが，分枝限定法のプロセスでメモリーを使い尽くしたために停止しました（8-1-3 参照)．求められた解が最適解であるかどうかはわかりません．
44	B&B memory error (no feas.sol.).	実行可能解を求めることができましたが，分枝限定法のプロセスでメモリーを使い尽くしたために停止しました（8-1-3 参照)．実行可能解を求めることができませんでした．
46	trust region too small	信頼領域法が失敗しました．数値的条件が悪いと思われます（8-1-5 参照)．
47	Continuous Variable <varname> cannot be included in model for wcsp.	wcsp は連続変数を扱うことができるのは０－１整数変数のみですが，０－１整数変数以外が現れました．
48	Variable <varname> appear in two selection()	変数<varname>が二つ以上の selection 文に現れています．変数は高々一つの selection 文にしか現れることはできません．
49	Variable <vaname> is fixed to infeasible value.	変数<varname>が上下限の外に固定されているので制約式を満たすことができません．
50	Both of two variables <varname1> and <varname2> cannot be 1.	変数<varname1> と変数 <varname2> がひとつの selection 文に現れていますが，両者ともに１に固定されています．selection 文の意図と矛盾しています．
51	wcsp is not available without SIMPLE.	アルゴリズム wcsp は solveQP()を用いた時に使うことはできません．
52	<number>th selection() statement has no movable binary variables.	ある selection 文に入っているすべての変数は，固定されており，動くことができません．

54	Constraint <name>'s weight is <value> should be -1 or non-negative value.	ソフト制約の重みとしては-1 あるいは生の整数値のみが許されています.
55	exterior solution obtained.	外点法（アルゴリズム”tepm”/”lepm”） found が実行可能領域の外の解を見つけて停止しました（8-1-5 参照）
110	CF failed at getcky	半正定値計画法を解く過程で数値的な問題が起きました(8-1-5 参照).
111	CF failed at logdet	110 と同じです.
112	InvMat cannot obtained at calxx1	110 と同じです.
113	GSEP failed at minevl	110 と同じです.
114	trianglization failed at minevl	110 と同じです.
115	Minimum eigenValue cannot obtained	110 と同じです.
120	PDgap is too large[PDfeasible]	双対ギャップが大きい状態で終了していますが，実行可能解は求められています.
121	PDgap is too large[Pfeasible]	双対ギャップが大きい状態で終了していますが，実行可能解は求められています.
122	minus stepsize detected	負のステップサイズが現れました．問題の非線形性が強いことが想像されます(8-1-5 参照).
123	The SDP constraint cannot be treated by specified algorithm.	指定された方法では半正定値条件を考慮することはできません.
124	No SDP constraint is detected.	半正定値条件が存在しませんが，半正定値条件を考慮するアルゴリズムが起動されました.

８－３　solveQP

solveQP という関数を通じて S-PLUS オブジェクトを，モデリング言語を介さず直接最適化アルゴリズム NUOPT に渡すことが可能です．solveQP は次のような混合整数線形・二次計画問題を解くためのものです.

最小化 $\frac{1}{2}x^T Qx + q^T x$

制約

$$
\begin{aligned}
c_{LO} \;\le\; Ax \;\le\; c_{UP} \\
b_{LO} \;\le\; x \;\le\; b_{UP} \\
x_i : Integer\,,\; i \in I \\
x_i : Continuous\,,\; i \notin I
\end{aligned}
$$

引数並びは以下の通りです.

```
> args(solveQP)
function(objQ, objL, A, cLO, cUP, bLO, bUP, x0, isint, type = minimize, trace = T)
```

次が引数の解説です．objQ を除いて，省略可能です.

ObjQ: (matrix or list)

へっせ行列 Q

ObjL: (vector) <optional>

目的関数の線形部分の係数 q，省略すると 0 であると解釈されます

A: (maxtrix or list) <optional>

制約式の左辺行列 A,省略すると 0 であると解釈されます

cLO,cUP: (vector) <optional>

制約の上下限．省略するとないものと思われます.

bLO,bUP: (vector) <optional>

変数の上下限．省略するとないものと思われます.

x0:(vector)<optional>

変数の初期値．省略すると NUOPT が適当に設定します.

isint:(vector)<optional>

変数が整数変数であるかの論理値ベクトル(T が整数変数であることを示します).

省略すると，すべて連続変数であると解釈されます.

objQ と A は，S-PLUS の行列あるいはリストオブジェクトのいずれとして定義することもできます．リストオブジェクトとして与える場合には，非零要素の行番号，列番号，値という三つのアイテムを持つリストとります.

A は行列として表現すると次のようになります.

```
> A             # matrix
      [,1] [,2] [,3] [,4]
 [1,]  1.1  0.0 -1.3  0.0
 [2,]  0.0 -2.2  0.0  2.4
```

以下のようにリストで表現しても等価です.

```
X <- list(c(1,1,2,2), c(1,3,2,4), c(1.1,-1.3,-2.2,2.4))
```

リストによる表現は行列の非零要素が少ない（ほとんどの要素が零）である場合には有効です.