<White paper FICO Xpress Optimization Suite>

Robust Optimization with Xpress

ユーザーガイド・例題集

＜日本語版＞

「誤訳、不適切な訳があるかもしれませんので、

英文と併読なさるようにしてください。」

<更新日：2015 年 5 月>

MSI株式会社
Xpress事業部
Tel: 043-2978841
Fax: 043-2978836
Email: xpress@msi-jp.com
Web: http://www.msi-jp.com

目次

Xpress を使用したロバスト最適化

通常、目的関数の期待値が不確実なデータの確率分布で最適な解を導く場合、不確実な値の実現を問わず、ロバスト最適化は実行可能な解を導くことに焦点を置きます：すなわちロバストとは、不確実性がモデルのオブジェクトに関連する場合に、ロバスト最適化は不確実性の数を仮定し、その中における最悪なケースの最適化という形で最適な解を導きます。

1.1　不確実性とロバスト制約

値に不確実が必要なモデルの値や係数は不確実性と呼ばれています。直感的に不確実性は、コントロール不能な未知数と見なされていますが、実はコントロールすることが可能です。私たちがある意思決定をした後、すなわちソルバーがモデルの変数に対して最適値を発見した後に、不確実性の値に対して意思決定を生成します。従って、モデルの変数の値は最悪のケースを仮定する必要があります。

不確実な係数や右辺の数式におけるある不確実を含むモデル制約はロバスト制約と呼ばれています。ロバスト最適化の概念を視覚化する最善な方法は、2 つのフェーズとして考えることです。議論のために、ある不確実なモデル値を含む制約より小さいまたは等しいと仮定します：

$$a_1x_1+\ldots+a_lx_l+(a_k+u_k)x_k+\ldots+(a_n+u_n)x_n \le b.$$

ここでは、全ての係数 ai は既知です。しかし、変数 xk から変数 xn の係数の値は、uk から un によって示され、不確実性のあるレベルが分っているだけです。典型的な変数に関して、不確実性が取りうる値を定義する必要があります。

不確実性の可能領域は不確実集合と呼ばれ、不確実性の上にある制約によってモデル化されます。上記の例題では、不確実性の値は小さい可能性があり、それらのノルムは上から制約されるとわかっている場合、与えられた信頼性を満たすロバストな解を導きたいと思います。これは、 $u_k^2+\ldots+u_n^2 \le r$ の式における制約を暗示しています。

不確実性の実行可能値を表現することで、ロバスト最適化は常に実行可能解の発見を目指します。すなわちロバスト制約は制約式

$$a_1x_1+\ldots+a_lx_l+max_{u_k^2+\ldots+u_n^2\le r}(a_k+u_k)x_k+\ldots+(a_n+u_n)x_n \le b.$$

と等しいことを意味します。

ロバストなモデルは複数のロバスト制約や複数の不確実性集合が含まれている場合があります。ロバスト最適化問題の可解性はモデルの中にあるロバスト制約は利用可能な数理的プログラミングソルバーで解ける形式に変換できるか否かにかかっています。結果としてその変換モデル、すなわち解かれたモデルを The robust counterpart と呼びます。

1.2　ロバスト制約の種類

ロバスト制約は変数または右辺で増加した不確実性を含めることができます。ロバスト制約の種類はそれに影響を与える不確実性集合をどう表現するかによって定義します。ナップザック問題の小さい例題を使用し、不確実性集合の代わりに明確な表現を考察します。

```
model Knapsack
 uses "mmrobust"                                  ! Load the robust library

 parameters
  NUM=5                                           ! Number of items
  MAXVAL=100                                      ! Maximum value
  MAXWEIGHT=80                                    ! Maximum weight
  WIMAX=102                                       ! Max weight allowed for haul
 end-parameters

 declarations
  Items=1..NUM                                    ! Index range for items
  VALUE: array(Items) of real                     ! Value of items
  WEIGHT: array(Items) of real                    ! Weight of items
  x: array(Items) of mpvar                        ! Decision variables
 end-declarations

 setrandseed(5);
 forall(i in Items) do
  VALUE(i):=50+random*MAXVAL
  WEIGHT(i):=1+random*MAXWEIGHT
 end-do
 forall(i in Items) x(i) is_binary                ! All x are 0/1

 MaxVal:= sum(i in Items) VALUE(i)*x(i)           ! Objective: maximize total value
 WtMax:= sum(i in Items) WEIGHT(i)*x(i) <= WIMAX  ! Weight restriction

 maximize(MaxVal)
```

不確実性を含まないこの基本的なモデルは 256.601 の解を得ます。

```
 Objective: 256.601
Item  Weight  Value
1: 0   74.37  118.04
2: 1   62.34  141.75
3: 1   27.74  114.85
4: 0   53.17  100.60
5: 0   74.02   65.82
```

1.2.1 不確実性係数における簡単な制限

不確実性の値に簡単なボックス制約を与えることができます。これは単に、係数が係数単位で取りうる値の範囲を定義する際ときに、有効なアプローチとなります。不確実性をモデルに導入すると、下記のようになります。

```
parameters
 NUM=5                                            ! Number of items
 MAXVAL=100                                       ! Maximum value
 MAXWEIGHT=80                                     ! Maximum weight
 WIMAX=102                                        ! Max weight allowed for haul
 WIPERCENT=0.3                                    ! Uncertainty as a percentage
end-parameters

declarations
 Items=1..NUM                                     ! Index range for items
 VALUE: array(Items) of real                      ! Value of items
```

```
 WEIGHT: array(Items) of real                ! Weight of items
 x: array(Items) of mpvar
 WeightUncertanty: array(Items) of uncertain ! Uncertains representing
                                             ! deviation from weight
end-declarations

forall(i in Items) x(i) is_binary            ! All x are 0/1
setrandseed(5);
forall(i in Items) do
 VALUE(i):=50+random*MAXVAL
 WEIGHT(i):=1+random*MAXWEIGHT
end-do

MaxVal:= sum(i in Items) VALUE(i)*x(i)       ! Objective: maximize total value

forall(i in Items) do
 WeightUncertanty(i) <=  WTPERCENT*WEIGHT(i) ! Uncertainty is a percentage of
                                             ! the expected weight
 WeightUncertanty(i) >=  0                   ! and only expected to go up this time
end-do
WtMax:= sum(i in Items) (WEIGHT(i) + WeightUncertanty(i))*x(i) <= WTMAX
                                             ! Weight restriction

maximize(MaxVal)
```

新しいモデルは望ましい重量から 30%の最大偏差を予測しています。
新しいモデルは、141.751 の解を得ます。

```
 Objective: 141.751
Item  Weight  Value
1: 0   74.37  118.04
2: 1   62.34  141.75
3: 0   27.74  114.85
4: 0   53.17  100.60
5: 0   74.02   65.82
```

最も重い品物を取ったことで解が変わりました。許容範囲の 30%の偏差によって取られた品物の重量が増加し、解が 81.042 になりました。最適解はロバストです。すなわち最大の目的関数を持つ、いかなる 2 次式の解に対してもロバスト制約に違反する可能性がある不確実性ベクトルが存在しています。
目的関数は不確実性のため、大幅に減少しました。当初の問題とロバスト問題の目的関数の差異は、ロバストネス価格（BS04）と言います。不確実性集合の部分集合が、例えば WTPERCENT より小さい値を持つ場合、不確実性の影響は減少し、より大きい目的関数を持つ解を得ます。むしろ、増加している WTPERCENT は増加する不確実性と等しくなり、ロバスト最適解の目的関数値を減少させます。
不確実性係数上の明示的な非負の制限を意味します。典型的な意思決定変数と対比すると、不確実性には与えられたデフォルトの下限が全くありません。これは一般的に、不確実性が正値に偏っていないことが考慮されているためです。ロバストなモデリング機能が解をそのような簡単なモデルにしたとしても、各個別の不確実性が独立し、簡単な範囲で制限した場合、同じモデルが多くの制約を制限する方法で全ての係数を修正することにより導かれることがわかります。ナップザック問題では、全ての変数が非負でなければならず、全ての重量が負ではないときに、不確実性の上限を従来の係数に追加することを意味しています。

```
declarations
 Items=1..NUM                                  ! Index range for items
 VALUE: array(Items) of real                   ! Value of items
 WEIGHT: array(Items) of real                  ! Weight of items
 x: array(Items) of mpvar                      ! Decision variables
end-declarations

forall(i in Items) x(i) is_binary              ! All x are 0/1

setrandseed(5);
forall(i in Items) do
 VALUE(i):=50+random*MAXVAL
 WEIGHT(i):=1+random*MAXWEIGHT

end-do

MaxVal:= sum(i in Items) VALUE(i)*x(i)         ! Objective: maximize total value
WtMax:= sum(i in Items) WEIGHT(i)*(1+WIPERCENT)*x(i) <= WIMAX
                                               ! Weight restriction

maximize(MaxVal)
```

実際、解は 141.751 のままです。

```
 Objective: 141.751
Item  Weight  Value
1: 0   74.37  118.04
2: 1   62.34  141.75
3: 0   27.74  114.85
4: 0   53.17  100.60
5: 0   74.02   65.82
```

1.2.2　不確実性集合上の線形制約

有限制約式はモデル化が可能な不確実性集合の 1 例にすぎません。複数、または全ての不確実性を集約することができ、その不確実性によって不確実性制約を記述することができます。

下記は、簡単な有限の問題から例題を改良し、代わりに不確実性の合計が全重量の約 10%を要求する式です。

```
parameters
 NUM=5                                         ! Number of items
 MAXVAL=100                                    ! Maximum value
 MAXWEIGHT=80                                  ! Maximum weight
 WIMAX=102                                     ! Max weight allowed for haul
 WIPERCENT=0.3                                 ! Uncertainty as a percentage
end-parameters

declarations
 Items=1..NUM                                  ! Index range for items
 VALUE: array(Items) of real                   ! Value of items
 WEIGHT: array(Items) of real                  ! Weight of items
 x: array(Items) of mpvar                      ! Decision variables
 WeightUncertanty: array(Items) of uncertain ! Uncertains representing
                                               ! deviation from weight
end-declarations

forall(i in Items) x(i) is_binary            ! All x are 0/1
```

```
setrandseed(5);
forall(i in Items) do
 VALUE(i):=50+random*MAXVAL
 WEIGHT(i):=1+random*MAXWEIGHT
end-do

MaxVal:= sum(i in Items) VALUE(i)*x(i)       ! Objective: maximize total value

forall(i in Items) do
 WeightUncertanty(i) >=  0
end-do

sum(i in Items) WeightUncertanty(i) <= WIPERCENT * sum(i in Items) WEIGHT(i)
WtMax:= sum(i in Items)  (WEIGHT(i) + WeightUncertanty(i))*x(i) <= WIMAX
                                              ! Weight restriction

maximize(MaxVal)
```

これが妥当と思われる一方で、大規模な右辺（全重量の 10%）のため、修正したモデルは非常に保守的な不確実性集合となり解が全て 0 になります。

```
  Objective: 0
Item  Weight  Value
1: 0   74.37  118.04
2: 0   62.34  141.75
3: 0   27.74  114.85
4: 0   53.17  100.60
5: 0   74.02   65.82
```

この動作はロバスト最適化が実行される方法を強調しています。実は、不確実性上の狭い上界を従来の不確実性集合から新しい不確実性集合に移行するとき、不確実性集合を緩和し、ロバストな実行可能領域を拡張して全ての不確実性を 1 つの品物に置くことを可能にします。各品物が個別に存在していたとしても、重量が重すぎるため全ての零解が唯一の実行可能なものになります。これはロバスト最適化の動作上で非常に重要な点です。不確実性集合を緩和する際、ロバストの対象物はより制限的になります。またその逆も同様に、ロバスト制約を緩和するには（保守性を低減させるために）対応する不確実性集合を厳しくする必要があります。例題のモデルの解はその後、不確実性値の実行可能領域を制限します。すなわち新たな不確実性制約の追加で設定した不確実性をより厳密にします。事実、この制約は各係数で不確実性の数を制限する必要があり、簡単な有限のケースと類似しています。線形制約集合で定義した不確実性集合の多くは多面的な不確実性集合と呼ばれています。多面的な不確実性集合は大変、有用性がありますが、例題で見てきた通り、多面的な不確実性集合を適用する際は注意が必要です。

1.2.3　楕円不確実性集合

前述の例題で見てきた通り、The robust counterpart またはオポーネントは意図した物よりも、より保守的なモデルを生成することで、楕円不確実性集合の頂点の解を最大限活用することができます。これは制約式などを制限するかまたは集合を改良することで、（例えば、新たな線形制約を追加することで）対応することができます。これは潜在的な問題のみの

近似かつより大きい次元で実用的なことがわかります。不確実性集合が二次制約式で記述されることがわかっている場合、例えば不確実性のベクトルのノームは上から制限し、その後、楕円不確実性集合を使用することができます。楕円不確実性の別の利用方法は、不確実性集合を信頼楕円体で記述する場合、すなわち平均ベクトル a、共分散マトリックス Q、および信頼性レベル α で定義することができます。この場合、不確実性集合は、
$(u-a)^T Q(u-a) \leq \alpha$ となります。

```
parameters
 NUM=5                                          ! Number of items
 MAXVAL=100                                     ! Maximum value
 MAXWEIGHT=80                                   ! Maximum weight
 WIMAX=102                                      ! Max weight allowed for haul
 WIPERCENT=0.3                                  ! Uncertainty as a percentage
end-parameters

declarations
 Items=1..NUM                                   ! Index range for items
 VALUE: array(Items) of real                    ! Value of items
 WEIGHT: array(Items) of real                   ! Weight of items
 x: array(Items) of mpvar                       ! Decision variables
 WeightUncertanty: array(Items) of uncertain ! Uncertains representing
                                                ! deviation from weight
end-declarations


 forall(i in Items) x(i) is_binary              ! All x are 0/1

 setrandseed(5);
 forall(i in Items) do
  VALUE(i):=50+random*MAXVAL
  WEIGHT(i):=1+random*MAXWEIGHT
 end-do

 MaxVal:= sum(i in Items) VALUE(i)*x(i)         ! Objective: maximize total value

 sum(i in Items) WeightUncertanty(i)^2 <= WIPERCENT * sum(i in Items) WEIGHT(i)
 WtMax:= sum(i in Items) (WEIGHT(i) + WeightUncertanty(i))*x(i) <= WIMAX
                                                ! Weight restriction

 maximize(MaxVal)
```

下記の通り、このモデルは要求した解を得ます。

```
  Objective: 215.445
 Item  Weight  Value
 1: 0   74.37  118.04
 2: 0   62.34  141.75
 3: 1   27.74  114.85
 4: 1   53.17  100.60
 5: 0   74.02   65.82
```

多面的不確実性が robust counterpart を生成する、つまり従来の問題と同じクラスとなり、線形制約の集合を追加することを意味しています。楕円不確実性の場合とは異なり、楕円不確実性に左右される各ロバスト制約に対し、二次錐の制約を含むロバスト counterpart を生成するため、問題の本質を変更する場合があります。従来の問題が線形計画問題（LP）の場合、robust counterpart は二次錐計画問題(SOCP)です。従来の問題が混合整数線形計画問題(MILP)の場合、robust counterpart は MISOCP です。MISOCP ソルバーは強力ですが、

MISOCP 問題は MILP よりも求解が困難です。

1.2.4 等式制約かつ不確実性制約を持たない不確実性の値

ここでは、注意が必要なロバスト最適化の落とし穴をいくつか言及していきます。はじめに、等式のロバスト制約はロバスト最適化において非常に制限的です。

```
declarations
 x, y, z : mpvar
 e : uncertain
end-declarations

e <= 1
e >= 0

x + y + e*z = 10

maximize(z)
```

この問題の解では、z は常に 0 となります。理由は簡単です：変数 x と変数 y の値が固定されているとき、z は常にノンゼロ値に等しく、不確実性 e の値のいかなる変更も、等式制約は実行不可能となります。一般的に、等式ロバスト制約における不確実性の影響は最も小さい次元に、従来の問題の実行可能領域を想定します。次に、不確実性集合が下限と上限の双方で制限されていること確認することが大切です。下記の例題は等式制約の場合と同じ影響を示しています。

```
declarations
 x, y, z : mpvar
 e : uncertain
end-declarations

x + y + e*z <= 10

maximize(z)
```

ここで、z は再び 0 を得ます。Z の任意のノンゼロ値は、十分に大きい e が実行不可能な制約となります。多くの場合、このような不明な制限は実行不可能なロバスト最適化問題をもたらします。

12.5 混合不確実性の設定と入力

ロバスト最適化理論によって、不確実性集合とロバスト制約における不確実性の使用に制限があるため、この点を留意する必要があります。最も顕著な制約事項としては、異なるロバスト制約間で同じ不確実性値の使用がデフォルトで禁止されています。これは、直接適用する場合のみならず、任意の不確実性制約式を使用した異なるロバスト制約と結ぶ（つながる）すべての不確実性に適用される禁止事項です。この禁止事項は、対応する robust counterpart が作成された方法によって強制されオポーネントにその戦略の選択を許可します。例えば、各制約基底における不確実の値などです。従って、2 つのロバスト制約の値の中に同一の不確実性がある場合、各ロバスト制約に異なる値を取れるようにします。しかし、複数のロバスト制約に対して同一の不確実集合の記述を使用することは、とても便利

です。下記の例題を参照ください。

```
declarations
 x, y: mpvar
 e, f : uncertain
end-declarations

x+e <= 1
y+f <= 1

e >= 0
f >= 0
e+f <= 1

maximize(x+y)
```

この例題は不確実性制約 $e+f \leq 1$ が 2 つのロバスト制約を同一の不確実性集合につながれ、重複しているため、デフォルト設定では解けません。その場合、
ROBUST_UNCERTAIN_OVERLAP 設定することで、この問題を求解できます。

```
declarations
 x, y: mpvar
 e, f : uncertain
end-declarations

x+e <= 1
y+f <= 1

e >= 0


f >= 0
e+f <= 1

setparam("XPRS_PROBNAME","h")

setparam("ROBUST_UNCERTAIN_OVERLAP", true);
```

しかし返された解は x=1=y です。 $e+f \leq 1$ が充足している場合、これが最適解ではないとすぐにわかります。重複を許可することで、2 つのロバスト制約に対し、個々にその意思決定の最適化を可能にする opponent が結果的に生じているのが現状です。このモードが、デフォルト設定で許可されていない理由は次の通りです：この方法は、モデル構築を行う人にとってまさに最適な手段であり、ROBUST_UNCERTAIN_OVERLAP を真に設定することは、不確実性制約を繰り返すことなく、複数回同一の不確実性集合の定義を再利用することができますが、問題を求解するとき、不確実性の値が定義した値を取らない場合、注意が必要です。異なるロバスト制約のために、値が異なることがあります。

1.2.6 不確実性に対する濃度制限

濃度制限は、0 とは異なる不確実性値の数を制限します。すなわち、濃度制限はノンゼロである不確実性の数や、さらに一般的に述べると名目値ではない数を制限します。ロバストに対応するものが生成される方法であるため、濃度制限における不確実性は常に上限と下限を持たせる必要があります。下記の例題で複数のロバスト制約（すなわち、重複する不

確実性において）どのように、不確実性を利用するか示します。
3 つの品物のうち、1 つのみが重量にゼロではない数を取りうる小さいナップザック問題です。

```
declarations
  x,y,z: mpvar
  ex, ey, ez: uncertain
  S={ex,ey,ez}
end-declarations

x is_binary ; y is_binary ; z is_binary

20*x + 10*y + 5*z >= 20

ex>=0 ; ex<=1
ey>=0 ; ey<=1
ez>=0 ; ez<=1

cardinality({ex,ey,ez},1)   ! Allow only one of the uncertains to be nonzero

setparam("ROBUSI_UNCERTAIN_OVERLAP",true)   ! allow overlaps

10*(x-x*ex) + 10*(y-y*ey) + 10*(z-z*ez) >= 10
10*(x-x*ex) + 10*(y-y*ey) + 10*(z-z*ez) <= 20

maximize(x+y+z)
```

返された解は x=y=1 と z=0 です。ここで注意すべきことは、他の制約式に関して、各ロバスト制約基底における濃度制約を考慮することおよび ROBUST_UNCERTAIN_OVERLAP が真に設定されない限り、モデルの求解ができないことを考慮することが重要です。

1.2.7 過去に使用したデータを活用する–シナリオ

シナリオとは不確実性の過去データを使用することで、ロバスト最適化問題を構築する効率的な方法の 1 つです。不確実性ベクトル u がある場合、モデルに適合する不確実性集合は未知ですが、不確実性ベクトルに対するこの 20 年間の月毎の各不確実性の値を含めた記録のデータベースがあります。従って、1 つ以上の制約がデータベースの不確実性ベクトルの各記録に充足しているモデルを構築したいと思います。不確実性集合のよい近似はないけれど、過去データに対するロバスト性を満たすことができる場合があります。シナリオを宣言することは、問題に全てのデータに対応する制約を手動で追加することと同じです。この機能は、シナリオデータがソルバーによって効率的に処理されたことの確認と過度に大規模な問題の生成を避けるために利用することができます。シナリオは過去データを処理することができ、過去の測定を取る前に分析したナップザック問題の次の拡張を示しめします。

```
parameters
 NUM=5                                  ! Number of items
 MAXVAL=100                             ! Maximum value
 MAXWEIGHT=80                           ! Maximum weight
 WTMAX=102                              ! Max weight allowed for haul
 UNCERTAINTY_LEVEL=0.3                  ! How much we are uncertain about the weight
 HISTORIC_PERIODS=100                   ! Number of scenarios
end-parameters

declarations
 Items=1..NUM                           ! Index range for items
 VALUE: array(Items) of real            ! Value of items
 WEIGHT: array(Items) of real           ! Weight of items
 UncertainWeight:array(Items) of uncertain
 x: array(Items) of mpvar               ! 1 if we take item i; 0 otherwise
 historical_weights: array (range, set of uncertain) of real
end-declarations

forall(i in Items) x(i) is_binary       ! All x are 0/1

setrandseed(5);
forall(i in Items) do
 VALUE(i):=50+random*MAXVAL
 WEIGHT(i):=1+random*MAXWEIGHT
end-do

MaxVal:= sum(i in Items) VALUE(i)*x(i)  ! Objective: maximize total value
WtMax:= sum(i in Items)  (WEIGHT(i)+UncertainWeight(i))*x(i) <= WTMAX

! Generate historical data, this would be data collected from actual realizations
forall(period in 1..HISTORIC_PERIODS, i in Items)
  historical_weights(period, UncertainWeight(i)) :=
    WEIGHT(i)*UNCERTAINTY_LEVEL*random
! Generate a solution that would be feasible for ALL historic realizations
scenario(historical_weights)

maximize(MaxVal)
```

ロバストモデルに基づいたシナリオが決定理論として等価であることを確認することは有益です。モデルに基づく簡単なシナリオは下記の通りです。

```
declarations
 x, y : mpvar
 e, f : uncertain
 historical_data: array (range, set of uncertain) of real
end-declarations

! load historical data for e and f
historical_data(1, e) := 1
historical_data(1, f) := 2


 historical_data(2, e) := 3
 historical_data(2, f) := 1
 historical_data(3, e) := 3
 historical_data(3, f) := 2

 e*x + f * y <= 1

 ! Generate a solution that would be feasible for ALL historic realizations
 scenario(historical_data)

 maximize(x+y)
```

この決定論的等価は：

```
declarations
 x, y : mpvar
 e, f : uncertain
 historical_data: array (range, set of uncertain) of real
end-declarations


scenario1 := 1 * x + 2 * y <= 1
scenario2 := 3 * x + 1 * y <= 1
scenario3 := 3 * x + 2 * y <= 1

! Generate a solution that would be feasible for ALL historic realizations
scenario(historical_data)

maximize(x+y)
```

1.2.8 目的の不確実性

不確実性係数が目的に導入される場合、ロバストに対応する解は最も保守的な目的を導きます。詳しく言うと、目的関数が$f(x;u)$かつ、値のベクトルは x であり不確実性のベクトルは u である最小化問題における最適な解は関数$g(x) = \max_u f(x;u)$となります。新しい補助変数 z が目的を表現するために使用される場合、これは目的をロバスト制約$z \geq f(x,u)$に変換することに等しくなります。

1.3　名目値：中心と非中心の不確実性

不確実性の名目値はそのデフォルト値、不確実性を仮定、または非ロバスト性の実数値を表しています。これまで見てきた例題において、不確実性の各名目値は 0 であったように、すべての不確実性が処理され、実際、不確実性は誤差項として処われます。

不確実性のデフォルトまたは中心（center）は 0 とは異なる名目値を指定することで変換することが可能です。名目値を使用する優位点は、同じモデルに不確実性を固定することにより得られたロバスト最適化モデルと決定なモデル双方でモデルを実行することができます。実数の不確実性係数として不確実性を使用し、実行させることが可能です。

使用する前に、不確実性の名目値を使用して実行する時の使用法とルールを下記の例題で示します。

1. Zero centered 不確実性を使用することで、そこでは Zero centered error　として不確実性はモデル化されます。

名目値を持つ不確実性：

$(5 + uncertain)x \leq 1$

式は下記の通りです：

```
declarations
  x : mpvar
  u : uncertain
end-declarations

(5+u)*x <= 1
```

2.係数としての不確実性、そこでは係数として uncertain がモデル化されます。名目値として 5 を持つ uncertain 係数となります：

$(5 + uncertain)x \leq 1$

また下記のように表現することもできます。

```
declarations
  x : mpvar
  u : uncertain
end-declarations

u:= 5 ! setting the nominal value
u*x <= 1
```

上記の例題で見てきた通り、名目値は演算子':='.を使用して設定します。等価演算子':='.と上記の演算子を区別することが重要です。(それが、与えられた値に不確実性の値を代わりに固定します。)

1.3.1 ロバスト性の値

zero-centered 不確実性の使用、または名目値を設定するとき、不確実性を取り除いて使用し、モデルを解くことができます。すなわちそれらの名目値に固定されているすべての不確実性係数を使用します。これは不確実性を追加せずに、モデルが実行可能かどうかを確認するのみならず、モデルの中で不確実性を持つ値を計算することもできます。
下記の例題は名目値に固定されたすべての不確実性を使用し、どのように解くかを示します。

```
declarations
 x : mpvar
 u : uncertain
end-declarations

0 <= u
     u <= 3

(1+u)*x <= 1

maximize(XPRS_NOMINAL, x)
nominal_objective := getobjval
writeln("Objective at nominal values:", nominal_objective );

maximize(x)
robust_objective := getobjval
writeln("Robust objective:", robust_objective );

writeln("Price of robustness:", nominal_objective - robust_objective );
```

モデルは下記の出力を得ます：

```
Objective at nominal values: 1
Robust objective: 0.25
Price of robustness: 0.75
```

不確実性に対する名目値を全く定義していないとき、デフォルト設定の 0 が使用されます。

1.3.2　名目値を使用して実行させる

名目値の実行は 2 つの簡単な規則で定義します。

規則 1：名目値を設定するために、不確実性ドメインを変更します。u:=a に不確実性として名目値を設定することで、u の後に生じるすべてが u+a に置換されます。下記の簡単なモデルは x=0.5 の解を得ます。

```
declarations
 x : mpvar
 u : uncertain
end-declarations

0 <= u
     u <= 1

(1+u)*x <= 1

maximize(x)
```

最悪なケースは、u が上限に 1 を取る場合です。2 を示すために u の名目値を変更することで、u は u+2 により置換され、代わりにモデルの解は 0.25 となります。

```
declarations
 x : mpvar
 u : uncertain
end-declarations

0 <= u
     u <= 1

u := 2  ! shifts the center of the uncertain

(1+u)*x <= 1

maximize(x)
```

不確実性 u は未だに 1 の上限を取りますが、制約式は、1 の上限を取る u を持つ $(1+u+2)x \leq 1$ を読み込みます。

規則 2：変更された名目値は割当てられた後に、定義されるロバストかつ不確実性集合の制約に反映されるのみです。

この作用は、実数値のパラメータの実行方法と類似すると考えると、分かりやすいと思います。規則 1 の例題を見てください。この作用がすでに使用されています。名目値の変更はロバスト制約にのみ影響を与えますが、このポイント前に宣言されたバウンドにも影響を与えます。

```
declarations
 x : mpvar
 r : real
```

```
end-declarations

r := 3

c1 := (1+r)*x <= 1

r := 5

maximize(x)
```

このモデルは x = 0.25 を得ます。制約が使用されている(3)を宣言するとき、それは、r の実数値であり、後でそれを再定義しても、制約に影響を与えませんでした。この作用は、不確実性の名目値で実行するときに、再利用されます。

```
declarations
 y : mpvar
 u : uncertain
end-declarations

u := 3

(1+u)*y <= 1

u := 5

maximize(XPRS_NOMINAL, y)
```

実数値の係数が y=0.25 で使用されるときと同様に、不確実性係数が使用されるモデルの名目バージョンが解かれます。（注：名目の引数を最大化する）これらの規則は、不確実性に対する名目値を扱う方法における優れた柔軟性を提供しますが、注意が必要で、規則 2 の影響を常に配慮しなくてはなりません。

1.3.3 不確実性集合のシフトに名目の値を使用する

名目値の付近にもその影響を同等にする一方で、名目値は不確実性係数をシフトするために、使用することができます。
下記の例題を見てください。

```
declarations
 x,y : mpvar
 e,f : uncertain
end-declarations

e^2 + f^2 <= 2  ! An ellipsoidal uncertainty constraint

e*x + f*y <= 1

maximize(x + y)
```

モデルは x=0.5、y=0.5 の解を得ます。不確実性 e の値と f はそれらの値を半径$\sqrt{2}$から取りえます。名目値が追加された時、モデルは下記の通りです。

```
declarations
 x,y : mpvar
 e,f : uncertain
end-declarations

e^2 + f^2 <= 2

e := 1
f := 1


e*x + f*y <= 1

maximize(x + y)
```

解は x=0.25、y=0.25 になります。この変化の要因を考えるために、問題に直接、名目値の影響を置換していきます。

```
declarations
 x,y : mpvar
 e,f : uncertain
end-declarations

e^2 + f^2 <= 2

(e+1)*x + (f+1)*y <= 1

maximize(x + y)
```

不確実性のレンジは変わりませんが、ロバスト制約において、それらが集中する値は変化に応じて、名目値にシフトします。

1.3.4　係数に名目値の不確実性を使用する

はじめて不確実性を使用する前に、名目値を割り当てることができますが、想定外の方法で動く場合があります。同じモデルの 3 つのバージョンを下記に示します。

Case 1 No nominal values are set	Case 2 Nominal values are set to 1	Case 3 Nominal values are set to different values
	declarations	declarations
declarations	x,y : mpvar	x,y : mpvar
x,y : mpvar	e,f : uncertain	e,f : uncertain
e,f : uncertain	end-declarations	end-declarations
end-declarations	e := 1	e := 2
e + f ≤ 2	f := 1	f := 5
e*x + f*y ≤ 1	e + f ≤ 2	e + f ≤ 2
maximize(x + y)	e*x + f*y ≤ 1	e*x + f*y ≤ 1
	maximize(x + y)	maximize(x + y)

この 3 つのバージョンはどれも x=y=0.5 の解を得ます。理由は、見ただけでは分かりにくいですが、例えば 3 番目のケースでは、規則 1 の影響を考えたときその理由が理解できます。

Case 3	becomes	which is
declarations x,y : mpvar e,f : uncertain end-declarations e := 2 f := 5 e + f ≤ 2 e*x + f*y ≤ 1 maximize(x + y)	declarations x,y : mpvar e,f : uncertain end-declarations e+2 + f+5 ≤ 2 (e+2)*x + (f+5)*y ≤ 1 maximize(x + y)	declarations x,y : mpvar e,f : uncertain end-declarations e + f ≤ -5 (e+2)*x + (f+5)*y ≤ 1 maximize(x + y)

不確実性制約にシフトが適用されるとき、（範囲を含め）例題では、それを相互的に相殺します：線形のロバスト制約と対応するエラー制約は、ともにシフトしています。しかし、万が一、モデルがその名目値によって解かれる場合は明らかに異なります。この意味を明確に示した下記の表を見てください。

The nominal equivalent of case 1:	**The nominal equivalent of case 2:**	**The nominal equivalent of case 3:**
declarations x,y : mpvar e,f : uncertain end-declarations 0*x + 0*y ≤ 1 maximize(x + y)	declarations x,y : mpvar e,f : uncertain end-declarations 1*x + 1*y ≤ 1 maximize(x + y)	declarations x,y : mpvar e,f : uncertain end-declarations 2*x + 5*y ≤ 1 maximize(x + y)
is unbounded.	solves to x=1 and y=0.	solves to x=0.5 and y=0.

1.4　ロバストモデルの例題

このホワイトペーパーでは実世界の問題に適用されたロバスト最適化の完全な例題を複数掲載しています。この例題に対応したモデルやデータファイルは、Xpress7.7 のパッケージに含まれています。（サブディレクトリ examples/robust/Mosel）

2 ロバスト最短経路

2.1　問題記述

毎日、自宅から仕事場まで車で通勤しています。複数のルートがあります。このルートを接続ノード（ルートの交差点）の集合の線として考えます。各ラインは（出発点）について、ある終点からもう一方への到着までにかかる所要時間は既知です。道路工事やその他要因による渋滞や遅延は未知です。

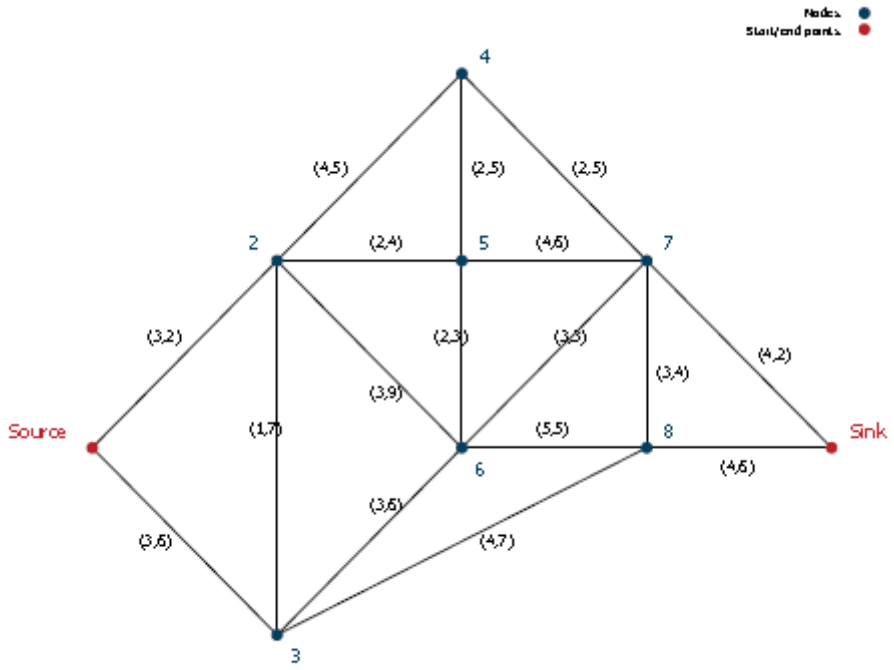

Figure 1: Road network

図 1 は、起点と終点を、Source と Sink として示した交通網の例を表しています。
接点でのﾗﾍﾞﾙﾀﾌﾟﾙ(L,D)は、接点と接点上で道路工事が原因となりうる最大遅延Dの長さ（移動時間）を示しています。このデータ例では、接点に従って走行する双方向の時間が同じとなり、双方向の最大遅延は同じですが、多くの場合、値が実際には異なることがあります。

2.2　数理的定式化

2.2.1　最短経路問題

ネットワークを通じて、最小費用経路を見出す問題は最短経路問題として知られています。アーク a が 0 以外を選択した場合、各アーク a は $use_a = 1$ と関連したバイナリ変数 use_a を使ってこの問題を説明できます。
Source と sink のノードを除いた全ネットワークの中にあるノードに対し、フローバランス制約を生成します：入ってくる交通量の合計は、出ていく交通量と同じです。Source ノードは 1 つのみ出ていく交通量で使用したアークを持ち、sink ノードは 1 つのみ入ってくる交通量に使用したアークを持ちます。下記は、原点 $ARC_{a,1}$ 経由し目的地ノード $ARC_{a,2}$ を定義します。

$$\min \quad \sum_{a \in Arcs} LEN_a \cdot use_a$$

$$\text{s.t.} \quad \sum_{a \in Arcs | ARC_{a1} = Source} use_a = 1, \quad \sum_{a \in Arcs | ARC_{a2} = Sink} use_a = 1$$

$$\sum_{a \in Arcs | ARC_{a2} = Source} use_a = 0, \quad \sum_{a \in Arcs | ARC_{a1} = Sink} use_a = 0$$

$$\forall n \in Nodes - \{Source, Sink\} : \sum_{a \in Arcs | ARC_{a2} = n} use_a = \sum_{a \in Arcs | ARC_{a1} = n} use_a$$

$$\forall a \in Arcs : use_a \in \{0, 1\}$$

上記のモデル定式化において、最短経路の計算で考慮すべき道路工事による遅延を追加的係数として目的関数に追加することができます。

$$\min \quad \sum_{a \in Arcs} (LEN_a + MAXDELAY_a) \cdot use_a$$

しかし、上記の追加は下記を仮定しています。
(a)全遅延は最大値を取りうる　かつ
(b)全道路で遅延が発生する
上記の仮定は極端に保守的な予測を算出します。
このケースでは移動時間全体の推測における上限を提供しますが、確実に起こりうる可能性はかなり低いと考えられます。

2.2.2　ロバスト最適化問題

本来、記述したかった問題は下記の通りです。
(a)遅延は指定した最大値までの値を取ります。
(b)かつ N 道路が原因となる遅延の値をとります。
これは、各アークの遅延の値は固定されず、不確実性を含む問題であることを意味しています。$arc\, a \in Arcs$ と関連した遅延の不確実な期間を delay a と示すならば、濃度制限された不確実性集合は下記のようになります。

$$U = \{delay : |\{delay_a : delay_a > 0\}| \leq N, 0 \leq delay_a \leq MAXDELAY_a \forall a \in Arcs\}$$

上記の結果、生成されるロバスト最適化問題は下記の通りです。

$$\min \quad \sum_{a \in Arcs} (LEN_a + delay_a) \cdot use_a$$

$$\text{s.t.} \quad \sum_{a \in Arcs | ARC_{a,1}=Source} use_a = 1, \quad \sum_{a \in Arcs | ARC_{a,2}=Sink} use_a = 1$$

$$\sum_{a \in Arcs | ARC_{a,2}=Source} use_a = 0, \quad \sum_{a \in Arcs | ARC_{a,1}=Sink} use_a = 0$$

$$\forall n \in Nodes - \{Source, Sink\} : \sum_{a \in Arcs | ARC_{a,2}=n} use_a = \sum_{a \in Arcs | ARC_{a,1}=n} use_a$$

$$\forall a \in Arcs : use_a \in \{0, 1\}$$

$$delay \in U = \{delay : |\{delay_a : delay_a > 0\}| \leq N, 0 \leq delay_a \leq MAXDELAY_a \forall a \in Arcs\}$$

2.3 実装

下記の Mosel モデルは 2.2 で生成したロバスト最適化モデルを実装します。注:最大 NWORK、N=2 に対するパラメータに道路工事を要因とする遅延の全数値を制限する不確実な遅延に関する密度の使用法に留意ください。このような密度制約に使用する不確実性は、明示的に下限と上限の集合を設定してください。

```
model "road network"
 uses "mmxprs", "mmrobust"

 parameters
   DATAFILE="roads_9.dat"
   NWORK = 2                                        ! Number of roadworks
 end-parameters

 declarations
   Nodes: range                                     ! Set of nodes
   ARC: array(Arcs:range,1..2) of integer           ! Arc origins/destinations
   LEN,MAXDELAY: array(Arcs) of real                ! Length and max delay per arc
   Source,Sink: integer                             ! Source and sink node numbers

   use: array(Arcs) of mpvar                        ! 1 iff arc is used
   TotalLength: robctr                              ! Objective function
   delay: array(Arcs) of uncertain                  ! Uncertain delay
 end-declarations

 !**** Input datafile ****
 initializations from DATAFILE
   Nodes  Arcs  Source  Sink  ARC
   [LEN,MAXDELAY] as "ArcData"
 end-initializations

!**** Robust problem formulation ****
 forall(a in Arcs) use(a) is_binary

! Sink and source of flow
 sum(a in Arcs | ARC(a,1)=Source) use(a)=1
 sum(a in Arcs | ARC(a,2)=Sink) use(a)=1
 sum(a in Arcs | ARC(a,2)=Source) use(a)=0
 sum(a in Arcs | ARC(a,1)=Sink) use(a)=0

! Flow balance in intermediate nodes
 forall(n in Nodes-{Source,Sink})
   sum(a in Arcs | ARC(a,2)=n) use(a) = sum(a in Arcs | ARC(a,1)=n) use(a)

! Random construction work on NWORK arcs
 forall(a in Arcs) delay(a) <= MAXDELAY(a)
```

2.4　結果

図 2 はロバスト解で定義したネットワークを通る経路です。総距離は 21 です。経路は、source →2 →5 → 6 → 7 → Sink この経路は 14 の名目の距離を持ちますが、経路 2-5 と経路 6-7 で道路工事が行われていた場合、7 の全遅延を追加する必要があります。この 2 つの経路は最大遅延の（経路 2-5 と経路 5-6）道路工事が行われている他のいかなる組合せでも全所要時間は 21 分以下となる結果が得られます。

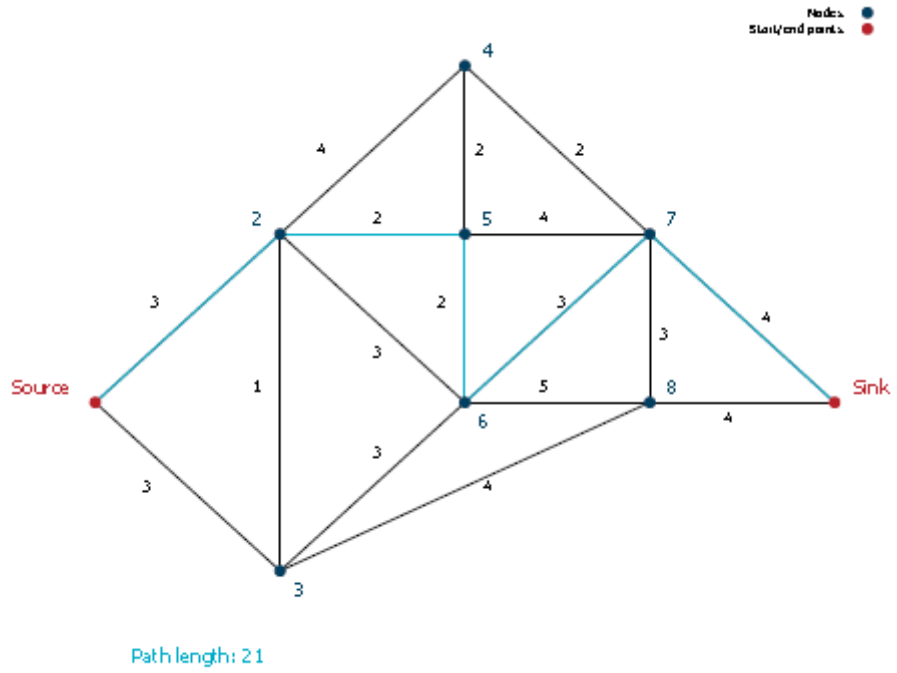

Figure 2: Robust solution

図 3 の解を考察していきます。道路工事による遅延を考慮せずに得た解です。この場合の最適な経路は全所要時間 11 となる Source → 3 → 8 →Sink です。しかし、最悪な場合、経路 Sink-3 と経路 3-8 は、所要時間が道路工事によって 13 分から 24 分に増大する可能性があり、この経路は明らかに考慮不足な解です。

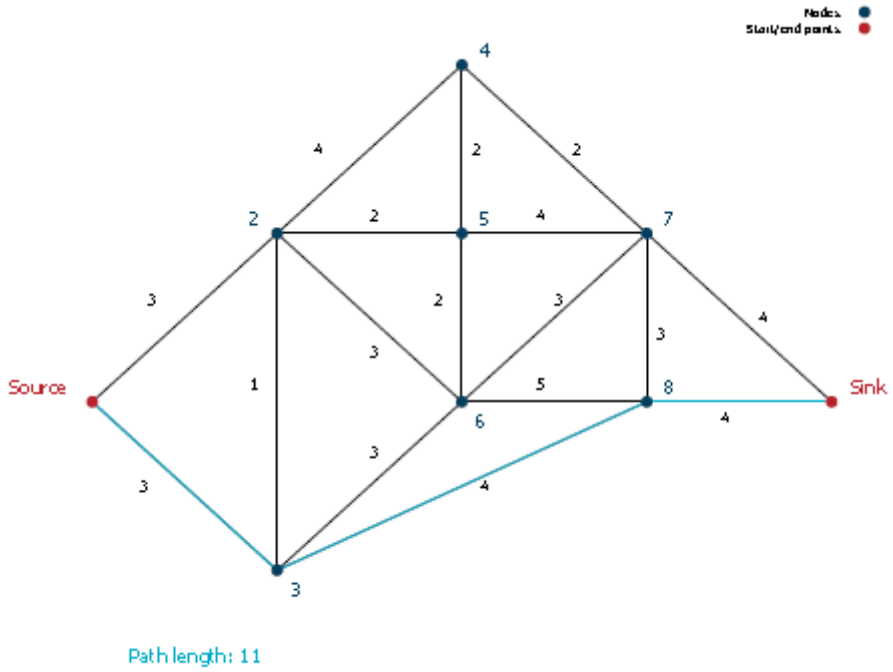

Figure 3: Zero-delay solution

最後に、全てのアークコストに最大値を設定して得た解（全起点には最大値の遅延があると仮定したグラフにおいて計算された最短経路）を考察していきます。図 4 で示されている通り、この解の経路は所要時間 27 分の Source →2 →4 →7 →Sink となっています。
多くても 2 つの経路で道路工事が行われていると仮定のもと、この所要時間は意味を持ちません。実際、名目の所要時間は 13 分であり、最悪な場合の所要時間は経路 2-4 と経路 4-7 で道路工事行っていると仮定して得た解に 10 分追加した解となり、従って 23 の全所要時間を導きます。上記で示した 21 のロバスト最短経路よりも悪い解となります。
上記の結果を要約した表 1 は、3 つのケースにおける 3 つのアプローチで得た所要時間を示しています。（ロバスト、名目値を持つ最短経路、全経路に最大遅延を持たせた最短経路）
道路工事がない場合、グラフ N＝2 の起点で道路工事が行われている場合、いたるところで道路工事が行われている場合も、N は無限です。

Table 1: Path costs

N	0	2	+inf
Robust	14	21	29
Nominal	11	24	30
All delays	13	23	27

注：このセクションのはじめで作成した仮定に基づき、最小の最悪なケースの所要時間で問題を解いた唯一の解は、モデルに不確実性を使用したこと、ネットワーク上に道路工事を持たせること（制限はありますが）は既に述べました。

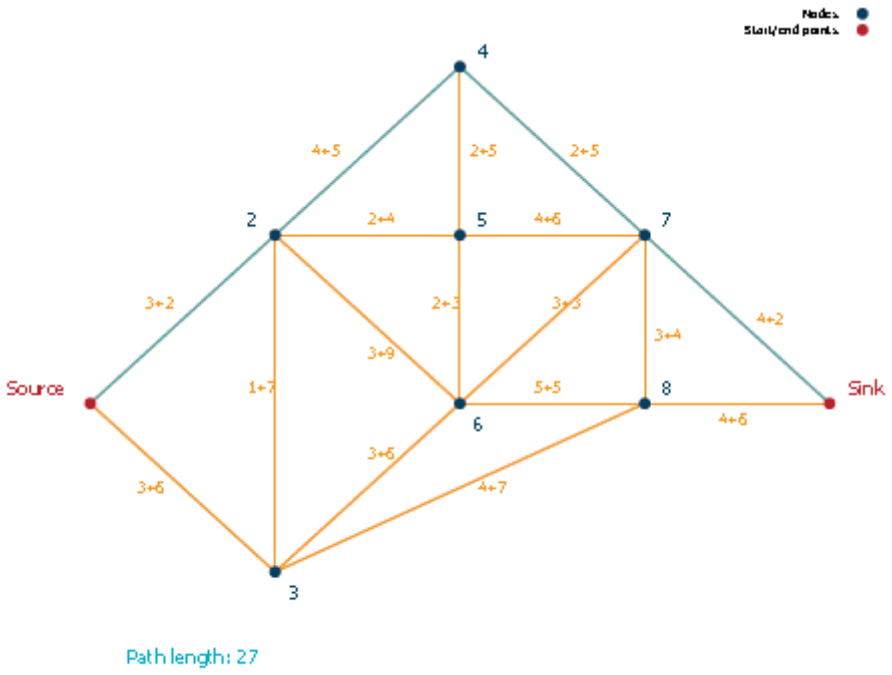

Figure 4: Maximum-delay solution

3　需要不確実性に基づいた製造計画

3.1　問題記述

グラスを製造している工場のマネージャは、翌 12 週間における製造計画の作成を考えています。工場では 1000 個のグラスのバッチに 6 種類の異なるタイプ(V1 から V6)を製造しています：不揃いな（1000 個未満）バッチもあります。全種類のガラスに対する 12 週間の需要予測があります。最初の在庫と要求される最終的な在庫の基準（数千個）と同様に既知です。全種類のガラス製品におけるバッチあたりの労働者と機械に要求される稼働時間と要求される在庫スペース（トレイ数で測定）および€で格納コストを与えます。

3.2　数理的定式化

3.2.1　多期間、多品目製造計画問題

$PRODS$を製品（グラスの種類）の集合および、期間の集合を$WEEKS = \{1, \ldots, NT\}$とします。期間 t 内の製品 p の需要に関する$DEM_{pt}$を記述します。また製品$CPROD_p$および製品$CSTOCK_p$と、グラスの種類 p に対する在庫費用があります。この費用は全ての期間において同一ですが、期間に対してインデックスを追加することにより、簡単に各期間に様々な費用をモデル化することができます。

$TIMEW_p$と$TIMEM_p$は製品 p の各ユニットでそれぞれに要求される機械の稼働時間と労働者の作業時間を示し、それに対応する在庫スペースは、$SPACE_p$です。各製品（表 2 のデータ

を参照ください。）あたりに要求される最終的な在庫基準$FSTOCK_p$と同一基準になるように最初の在庫$ISTOCK_p$を与えます。労働者と機械の許容量に対し、それぞれ$CAPW$と$CAPM$を記述し、在庫場所の許容量に対して$CAPS$を記述します。
この問題を解くために、期間 t のグラスタイプ p の製品を表現する変数$produce_{pt}$が必要です。期間 t の終了日の各製品 p の在庫基準に対応する変数は$store_{pt}$と呼ばれます。慣例により、最初の在庫基準$ISTOCK_p$は期間 0 の終了日の在庫基準と考えうる場合、ストックバランス制約の定式化を簡素化するために$store_{p0}$の表記法を使用します：

$$\forall p \in PRODS, t \in WEEKS : store_{pt} = store_{p,t-1} + produce_{pt} - DEM_{pt}$$

これらのストックバランス制約は製品の量$store_{pt}$を示し、すなわち期間 t の終了日の在庫水準は前期の終了日における$store_{p,t-1}$+期間 t の製品$produce_{pt}$p−期間終了日の需要DEM_{pt}に等しいことを示しています。
計画期間終了日に在庫が 0 にならないように、計画期間終了日における製品の在庫量を一定に保つ必要があります：

$$\forall p \in PRODS : store_{p,NT} \geq FSTOCK_p$$

すべての期間に対して様々な容量制約を定式化していきます。下記の制約式は労働者の許容制約、機械の運行時間の制約および、在庫スペースにおける制約を生成します：

$$\forall t \in WEEKS : \sum_{p \in PRODS} TIMEW_p \cdot produce_{pt} \leq CAPW$$

$$\forall t \in WEEKS : \sum_{p \in PRODS} TIMEM_p \cdot produce_{pt} \leq CAPM$$

$$\forall t \in WEEKS : \sum_{p \in PRODS} SPACE_p \cdot produce_{pt} \leq CAPS$$

費用関数とは、すなわち製品と全ての製品に対する在庫費用および期間の合計を最小化します。

$$\min \sum_{p \in PRODS} \sum_{t \in WEEKS} (CPROD_p \cdot produce_{pt} + CSTORE_p \cdot store_{pt})$$

製品の変数に対する非負制約式と上記で記述した定式の在庫量によって完全な数理モデルを得ます。

$$\min \quad \sum_{p \in PRODS} \sum_{t \in WEEKS} \left(CPROD_p \cdot produce_{pt} + CSTORE_p \cdot store_{pt} \right)$$

$$\text{s.t.} \quad \forall p \in PRODS, t \in WEEKS : store_{pt} = store_{p,t-1} + produce_{pt} - DEM_{pt}$$

$$\forall p \in PRODS : store_{p,NT} \geq FSTOCK_p$$

$$\forall t \in WEEKS : \sum_{p \in PRODS} TIMEW_p \cdot produce_{pt} \leq CAPW$$

$$\forall t \in WEEKS : \sum_{p \in PRODS} TIMEM_p \cdot produce_{pt} \leq CAPM$$

$$\forall t \in WEEKS : \sum_{p \in PRODS} SPACE_p \cdot produce_{pt} \leq CAPS$$

$$\forall p \in PRODS, t \in WEEKS : produce_{p,t} \geq 0$$

$$\forall p \in PRODS, t \in WEEKS : store_{p,t} \geq 0$$

3.2.2　ロバスト最適化問題

上記で述べた数理モデルの背景にある様々な仮定は疑問がある場合：

1. 一定の資源許容量：人材の労働枠は、経時的に一定にならないと考えうるであろうし、（休日や研修、病欠などで左右される）計画的な機械のメンテナンスや不慮の機械故障による供給停止などのリスクもあります。
2. 正確な需要量は既知：一般的に、需要の予測は推定値であり、過去のデータを分析した結果から算出されます。

従業員の欠勤

機械の供給停止のケース（不測の事態 k として定式化）は、このマニュアルのセクション 7 と 6 に掲載しています。従ってここでは、人材の労働枠（可用性）における不確実性をとらえる方法を見ていきます。

各期間 t （実際の欠勤の多くがこの値を取りうる）の欠勤時間における上限を$ABSENCE_t$としますし、経験値により長期間にわたる（全労働時間のパーセンテージ A として示します。）平均的な欠勤の事由は既知です。各期間あたり、実際の欠勤時間を示すために、不確実性$absent_t$を導入します。オリジナル・モデルで生じた従業員許容制約式は下記の制約式に置き換えます。

$$\forall t \in WEEKS : \sum_{p \in PRODS} TIMEW_p \cdot produce_{pt} \leq CAPW - absent_t$$

$$absent \in U_{absent}$$

多面的不確実性集合 Uabsent の表現は、下記の通りです。

$$U_{absent} = \{absent : \sum_{t \in WEEKS} absent_t \leq A \cdot CAPW \cdot |WEEKS|,$$

$$0 \leq absent_t \leq ABSENCE_t \forall t \in WEEKS\}$$

需要シナリオ

需要のロバスト定式化に関して、同時期の過去データや、様々な予測方法により算出された実現可能な様々なシナリオがあると仮定し、需要の実行可能領域を記述します。
固定された需要量 DEM_{pt} では、シナリオ $s \in SCEN$ の所与の集合に対する需要シナリオデータ $SCENDEM_{spt}$ によって決定される不確な量 $demand_{pt}$ を使用します。

$$\forall p \in PRODS, t \in WEEKS : store_{pt} = store_{p,t-1} + produce_{pt} - DEM_{pt}$$

単純な’robustification’　は不確実性 $demand_{pt}$ によって固定需要量に簡単に置き換えられる場合があります。しかしこのアプローチは期待される結果には結びつかないでしょう。等式制約に 1 つの不確実な量を導入することで、不確実性の様々な実現に対するいかなる領域も残すことができません。従ってここでは、在庫バランス制約に 2 つの不等式集合を使用します。

$$\forall p \in PRODS, t \in WEEKS : store_{pt} \leq store_{p,t-1} + produce_{pt} - demand_{pt}$$
$$\forall p \in PRODS, t \in WEEKS : store_{pt} \geq store_{p,t-1} + produce_{pt} - \max_{s \in SCEN} SCENDEM_{pt}$$

一番目の不等式は需要が期間内の製造と開始時と期間終了期間の異なる在庫基準によって充足すべきであることを表現しています。不等式の変換したモデルは下記の通りです。

$$demand_{pt} \leq store_{p,t-1} + produce_{pt} - store_{pt}$$

3.3　実装

標準的な決定論的モデルの実装は下記の通りです。

```
model "C-2 Glass production"
 uses "mmxprs"

 declarations
  NT = 12                              ! Number of weeks in planning period
  WEEKS = 1..NT
  PRODS = 1.. 6                        ! Set of products

  CAPW,CAPM: integer                   ! Capacity of workers and machines
  CAPS: integer                        ! Storage capacity
  DEM: array(PRODS,WEEKS) of integer   ! Demand per product and per week
  CPROD: array(PRODS) of integer       ! Production cost per product
  CSTOCK: array(PRODS) of integer      ! Storage cost per product
  ISTOCK: array(PRODS) of integer      ! Initial stock levels
  FSTOCK: array(PRODS) of integer      ! Min. final stock levels
  TIMEW,TIMEM: array(PRODS) of integer ! Worker and machine time per unit


  SPACE: array(PRODS) of integer       ! Storage space required by products

  produce: array(PRODS,WEEKS) of mpvar ! Production of products per week
  store: array(PRODS,WEEKS) of mpvar   ! Amount stored at end of week
 end-declarations

 initializations from 'c2glass.dat'
  CAPW CAPM CAPS DEM CSTOCK CPROD ISTOCK FSTOCK TIMEW TIMEM SPACE
 end-initializations
```

```
! Objective: sum of production and storage costs
 Cost:=
  sum(p in PRODS, t in WEEKS) (CPROD(p)*produce(p,t) + CSTOCK(p)*store(p,t))

! Stock balances
 forall(p in PRODS, t in WEEKS)
   Bal(p,t):=
     store(p,t) = if(t>1, store(p,t-1), ISTOCK(p)) + produce(p,t) - DEM(p,t)

! Final stock levels
 forall(p in PRODS) store(p,NT) >= FSTOCK(p)

! Capacity constraints
 forall(t in WEEKS) do
  LimitW(t):= sum(p in PRODS) TIMEW(p)*produce(p,t) <= CAPW     ! Workers
  LimitM(t):= sum(p in PRODS) TIMEM(p)*produce(p,t) <= CAPM     ! Machines
  LimitS(t):= sum(p in PRODS) SPACE(p)*store(p,t)   <= CAPS     ! Storage
 end-do

! Solve the problem
 minimize(Cost)

! Solution printing
 writeln("Total cost: ",getobjval)
end-model
```

従業員の欠勤

従業員の欠勤に関するより詳細な処理の実装を行うために、各期間(ABSENCE)あたりの欠勤時間の推定時間の最大数を使用して不確実性 absent を導入し、計画期間中の全欠勤時間における制限は 5%と仮定します。作業能力の制限において、欠勤は、理論上利用可能な勤務時間から推測しなければなりません。（ここでのこの値は、基本モデルのデフォルト制限値より 20%高いと仮定します。）

```
declarations
  ABSENCE: array(WEEKS) of real          ! Max. absence (hours)
  absent: array(WEEKS) of uncertain      ! Absence of personnel
 end-declarations

! Limit on total absence (uncertainty set)
 sum(t in WEEKS) absent(t) <= 0.05*CAPW*WEEKS.size
 forall(t in WEEKS) absent(t) <= ABSENCE(t)
 forall(t in WEEKS) absent(t) >= 0

! Uncertains occur in several constraints
 setparam("ROBUSI_UNCERTAIN_OVERLAP", true)

! Worker capacity constraints
 forall(t in WEEKS)
   LimitW(t):= sum(p in PRODS) TIMEW(p)*produce(p,t) <= CAPW*1.2 -absent(t)
```

需要シナリオ

需要シナリオを定式化するために、需要のデータ（すなわち在庫バランス制約）に関する制約の定義を修正しなければなりません。シナリオ制約を解説する前に、シナリオの構成、（すなわちシナリオ・カウンターや全期間に関係する不確定需要でインデックス付けした配列 SCENDATA）を使用して推測される形にシナリオ・データをコピーする必要があります。

```
declarations
 SCEN: range                                          ! Demand scenarios
 SCENDEM: array(SCEN,PRODS,WEEKS) of integer          ! Demand per product & week
 demand: array(PRODS,WEEKS) of uncertain              ! Uncertain demand
 SCENDATA: array(SCEN,set of uncertain) of real       ! Aux. data structure
end-declarations

! Demand scenarios
forall(s in SCEN, p in PRODS, t in WEEKS | SCENDEM(s,p,t)>0)
  SCENDATA(s, demand(p,t)):= SCENDEM(s,p,t)
scenario(SCENDATA)

! Stock balances
forall(p in PRODS, t in WEEKS) do
  Bal(p,t):=
    store(p,t) <= if(t>1, store(p,t-1), ISTOCK(p)) + produce(p,t) - demand(p,t)
  Bal2(p,t):=
    store(p,t) >= if(t>1, store(p,t-1), ISTOCK(p)) + produce(p,t) -
    max(s in SCEN) SCENDEM(s,p,t)
end-do

! Uncertains occur in several constraints
setparam("ROBUSI_UNCERTAIN_OVERLAP", true)
```

不確実性要素の 2 つの集合を同時に適用することができます。しかし不確実性要素の効果を分析するために、一度に 1 つの集合のみを適用するほうが望ましいでしょう。

3.4　結果

元の問題記述とインスタンスデータは[GHPS02]から取られています。（セクション 8.2　ガラスコップ製品）図 2 は 6 種類のグラスに関する費用と資源使用法のデータを示しています。

Table 2: Data for six glass types

	Production cost	Storage cost	Initial stock	Final stock	$Time_{worker}$	$Time_{machine}$	Storage space
V1	100	25	50	10	3	2	4
V2	80	28	20	10	3	1	5
V3	110	25	0	10	3	4	5
V4	90	27	15	10	2	8	6
V5	200	10	0	10	4	11	4
V6	140	20	10	10	4	9	9

基本的な需要シナリオを持つオリジナル問題の解は、総費用が€185,899 となります。結果として得られる製造計画の詳細は図 3 に示します。利用可能な人材資源はほぼ完全に消費され（最初の週で、351 時間が消費され、他の全週で、390 時間の制限に達しています。）機械はある期間（週 1-3 と 5）において、全ての許容量を消費していますが、利用可能な在庫スペースは現実のニーズを超過しています。

ロバスト性をこのような厳しい制約を持つ問題に導入する場合、選択肢を組み込む余地が全くないため、結果はおそらく「実行不可能な問題」となります。従って、この元の値が最悪な推測であると見なし、実際の利用可能な勤務時間を 20%と仮定した従業員の作業時間の制限を緩和させます。

Table 3: Optimal production plan for basic demand scenario

	Week	1	2	3	4	5	6	7	8	9	10	11	12
V1	Prod.	*8.8*	5.5	0.6	30.2	27.4	*8.6*	23	20	29	30	28	42
	Store	*38.8*	*22.2*	*4.8*	-	*10.4*	–	–	–	–	–	–	*10*
V2	Prod.	0	16	23	20	11	10	12	34	21	23	30	22
	Store	*3*	–	–	–	–	–	–	–	–	–	–	*10*
V3	Prod.	18	35	17	10	9	21	23	15	10	0	13	27
	Store	–	–	–	–	–	–	–	–	–	–	–	*10*
V4	Prod.	16	45	24	38	41	20	19	37	28	12	30	47
	Store	–	–	–	–	–	–	–	–	–	–	–	*10*
V5	Prod.	47.7	14.6	35.1	14.3	23.5	*22.8*	*43.8*	0	26.5	*2.8*	0	0
	Store	*24.7*	*19.3*	*31.4*	*30.8*	*44.2*	*45*	*70.8*	*40.75*	*39.25*	*35*	*20*	*10*
V6	Prod.	12	18	20	19	18	35	0.8	27.2	12	49	29.2	5.8
	Store	–	–	–	–	–	–	*0.75*	–	–	*19*	*27.25*	*10*
Workers		351	390	390	390	390	390	390	390	390	390	390	390
Machines		850	850	850	753.2	850	*836.8*	790	675.2	742.5	650.2	641.2	*641.8*
Space		*268.8*	166.2	*144.8*	123	*218.4*	180	*289.8*	163	157	311	325.2	330

図 4 に欠勤の推測結果の詳細をまとめました。€181、210 の総費用が前述の固定された最悪な欠勤ケースよりわずかに減少しています。

Table 4: Summary results with robust formulation of absence

Week	1	2	3	4	5	6	7	8	9	10	11	12
Workers	243	*378*	370	*407*	325	*398*	383	414	412	*445*	429	*437*
CAPW-ABSENCE	*437*	*437*	*429*	*429*	*414*	*398*	*398*	*414*	*445*	*445*	*429*	*437*
Machines	609	*850*	736	770	736	794	*772.2*	*739.8*	803	*839.5*	734	747.5
Space	152.5	32	0	0	20	0	99	0	16	130	219.5	330

シナリオベースのアプローチは€203,545 の高額な総費用が結果として生じています。図 5 で纏めた結果において機械の稼働時間許容量と従業員の労働時間許容量は、双方ともにほぼ許容量制限まで消費されていることがわかります。（各 850 and 429=1.1 ・390）
さらに、大量の製品が在庫として残っています。

Table 5: Summary results with demand scenarios

Week	1	2	3	4	5	6	7	8	9	10	11	12
Workers	363	425	425	425	425	425	425	425	425	425	425	425
Machines	*850*	*850*	*850*	*850*	*850*	*850*	*850*	*850*	*850*	*786.7*	711	700.3
Space	230.7	145.9	150.4	126.3	233.2	211.9	*306.8*	181.7	169.7	256.7	305.7	330

もっと簡単に比較するために、図 5 のロバスト定式化で使用したものと同じ許容制限を使ったオリジナル・モデルに対する要約結果を図 6 で示しています。このケースにおける総費用は€ 181,432 となります。

Table 6: Summary results for deterministic model with 1.1 · CAPW

Week	1	2	3	4	5	6	7	8	9	10	11	12
Workers	243	*378*	370	*407*	305	*418*	393	425	425	425	425	425
Machines	609	*850*	736	770	681	*849*	799	771.2	*840*	783.2	721	721.5
Space	152.5	32	0	0	0	0	108.7	21.2	50.6	151.5	245.5	330

4　ロバスト・ネットワークデザイン

4.1　問題記述

電気通信ネットワークを作成していきます。ルータの集合、リンクの集合やトラフィック需要量（M/Bで計測）の集合をルータのペア間に与えます。便宜上、各々は、都市にあると仮定します。全てのトラフィック需要量を満たすために、各リンクに許容量を設定していきます。科学文献におけるこの問題の様々なバリエーションや、プロビジョニング・仮想私設ネットワーク問題（略：VPN）を扱う重要なサブクラスもあります。一般的にVPNトラフィック需要量は短期間でさえ、予測が難しいためトラフィック需要量は不確実です。この点に注意しながら、VPNを現実的に満たすことを考慮していきます。

2つのパラメータs_i^+とs_i^-は、それぞれノードiで、最大の送信と受信の合計を予測しているものと仮定します。従って、この問題はネットワークの各ノードに対して、トラフィックの送受信において上限と下限により、モデル化した需要における不確実性に関するノードhからノードkに任意のトラフィック需要量d_{hk}を適用するためにネットワーク(所与の許容量Cのユニットの倍数)の各アークの許容量の値を探索します。

4.2　数理的定式化

nとmアークを持つ有向グラフGを考察します。G = (V,A)として定義し、$V = \{1, \ldots, n\}$はnノードの集合を示し、$A = \{(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)\}$はアークの集合です。各アーク$(i, j) \in A$に設定した許容量（MB/s）の各ユニットの費用は$a_{ij}$で示され、各ペアのノード(h,K)に対するhからkまでの（unknown）トラフィック需要量は、d_{hk}により与えられます。

バイナリフロー変数f_{ij}^{hk}を導入します。：この変数は需要（h,k）がアーク(i,j)を通り、0またはその他と取るルート、かつ各資源/距離間のデータのフローをモデル化することに役立ちます。

ネットワークに設定される許容量Cのチャンネル数を表す整数変数x_{ij}も導入する必要があります。この問題はトラフィック需要おける不確実性の重要な付加を持つマルチ・コモディティネットワークフロー問題です。はじめに、不確実性集合をモデル化します：全ての不確実な需要d_{hk}は非負かつノードiの送受信合計需要は所与のパラメータによる上限から固定されています。

$$\forall k \in V, \sum_{h \in V : h \neq k} d_{hk} \leq s_k^-$$

$$\forall h \in V, \sum_{k \in V : k \neq h} d_{hk} \leq s_h^+$$

この問題に対する 2 つの主要な制約式のクラスを考察します：変数 f と許容量制約式に対す

るフローの予測は、下記の通りです：各ノードiと各需要(h,k)に関して、iがこの需要に対する中間ノードである場合、すなわちhでもkでもない場合、送受信のフローは等しくなければなりません。すなわち

$$\forall i,h,k \in V : h \neq i \neq k, \sum_{j:(i,j)\in A} f_{ij}^{hk} = \sum_{j:(j,i)\in A} f_{ji}^{hk}.$$

i = hのとき、フローのトータルバランスは下記の通りです。

$$\forall i,h,k \in V : i = h, \sum_{j:(i,j)\in A} f_{ij}^{hk} - \sum_{j:(j,i)\in A} f_{ji}^{hk} = 1.$$

i = kのときは冗長となりこれに対応する制約は省略することができます。ここで、許容量の制約を作成します：アーク(i, j)における全トラフィックは、(unknown)トラフィック需要量で重み付けされ、アークに設定された全許容量に等しいか、または小さくする必要があります：

$$\forall (i,j) \in A, \sum_{h\in V}\sum_{k\in V:k\neq h} d_{hk} f_{ij}^{hk} \leq C x_{ij}.$$

最後に、目的関数を最小化します。つまり

$$\sum_{(i,j)\in A} a_{ij} x_{ij}.$$

4.3　実装

下記はMoselで実装したモデルです：ROBUST_OVERLAP_UNCERTAINはオプションであり、各アークで許容量制約式を不確実性需要の部分集合として使用するために用います。

```
model "Robust Network"
  uses "mmrobust", "mmxprs"

  parameters
    vpn_data="vpn_data.dat"
  end-parameters

  declarations
    NODES: range                          ! Set of nodes
    ARCCOST: dynamic array(NODES, NODES) of real ! Per-unit cost of arc (i,j)

    DEM_IN:   array(NODES) of real        ! Total INCOMING demand at each node
    DEM_OUT:  array(NODES) of real        ! Total OUTGOING demand at each node

    UNITCAP: integer                      ! Per-unit capacity (independent of arc)

    NETCOST: linctr                       ! Objective function


! Decision variables
flow: dynamic array(NODES, NODES, NODES, NODES) of mpvar
                        ! flow(i,j,h,k) is 1 iff demand h->k uses arc (i,j)
capacity: dynamic array(NODES, NODES) of mpvar  ! Capacity to install on arc (i,j)

  ! Uncertain parameters
  demand: array(NODES, NODES) of uncertain  ! Uncertain traffic demand
end-declarations
```

```
! The following option is necessary to allow uncertain demands to be
! used across multiple capacity constraints

setparam("robust_uncertain_overlap", true)

! Set verbosity level

setparam("xprs_verbose", true)

!**** Data input ****

initializations from vpn_data
  NODES ARCCOST
  DEM_IN DEM_OUT
  UNITCAP
end-initializations

!**** Model formulation ****

forall(i in NODES, j in NODES, h in NODES, k in NODES |
       exists(ARCCOST(i,j)) and ARCCOST(i,j) > 0 and h <> k) do
  create(flow(i,j,h,k))
  flow(i,j,h,k) is_binary
end-do

forall(i in NODES, j in NODES | exists(ARCCOST(i,j)) and ARCCOST(i,j) > 0) do
  create(capacity(i,j))
  capacity(i,j) is_integer
end-do

! Flow balance at intermediate nodes for each demand(h,k)
forall(i in NODES, h in NODES, k in NODES | i <> h and i <> k and k <> h)
  sum(j in NODES | exists(flow(i,j,h,k))) flow(i,j,h,k) -
  sum(j in NODES | exists(flow(j,i,h,k))) flow(j,i,h,k) = 0

! Flow balance at source nodes (unnecessary for sink nodes)
forall(i in NODES, h=i, k in NODES | k <> h)
  sum(j in NODES | exists(flow(i,j,h,k)) ) flow(i,j,h,k) -
  sum(j in NODES | exists(flow(j,i,h,k)) ) flow(j,i,h,k) = 1

! Capacity (robust) constraint

forall(i in NODES, j in NODES | exists(ARCCOST(i,j)) and ARCCOST(i,j) > 0)
  sum(h in NODES, k in NODES | h <> k) demand(h,k) * flow(i,j,h,k) <=
    UNITCAP * capacity(i,j)

! Uncertainty set: all demands are nonnegative and constrained by
! total outgoing and incoming demand

forall(h in NODES, k in NODES | h <> k) demand(h,k) >= 0


forall(h in NODES) sum(k in NODES | h <> k) demand(h,k) <= DEM_OUT(h)
forall(h in NODES) sum(k in NODES | h <> k) demand(k,h) <= DEM_IN(h)

! Shortest path length
NetCost := sum(i in NODES, j in NODES | exists(ARCCOST(i,j)) and ARCCOST(i,j) > 0)
  ARCCOST(i,j) * capacity(i,j)


!**** Solving ****

minimize(NetCost)

declarations
  curNode: integer ! Local variable used in following the path of each demand
end-declarations
```

```
! Printing capacities
writeln("Robust solution has total cost ", getobjval)
forall(i in NODES, j in NODES | exists(capacity(i,j)) and capacity(i,j).sol > 0)
  writeln("arc ", i, " -- ", j, ": ", capacity(i,j).sol, " units")

writeln("Paths:")

forall(h in NODES, k in NODES | h <> k) do

    write("Demand ", h, " --> ", k, ": ")
    curNode := h
    write(h)
    while(curNode <> k) do
      forall(j in NODES | flow(curNode, j, h, k).sol > 0.5) do
        write(" -- ", j)
        curNode := j
      end-do
    end-do
    writeln("")
  end-do

end-model
```

4.4　データをインプットする

図5に示したネットワークを考察していきます。各リンクは、逆方向の2つのアークを表しています。下記の表7は不確実性集合（各ネットワークのノードに対する送受信の最大トラフィック需要）を説明しているデータファイルvpn_data.datで指定されたパラメータのレポートです。

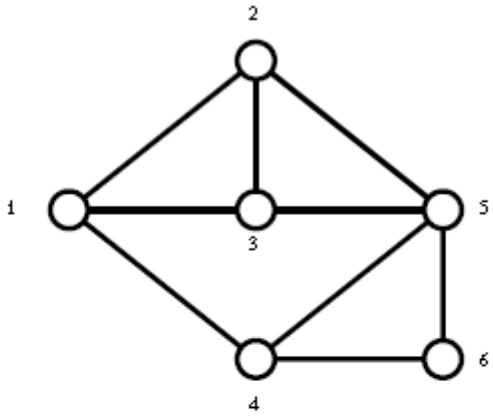

Figure 5: Network topology

Table 7: Maximum incoming/outgoing demand

Node	Incoming	Outgoing
1	120	105
2	34	95
3	35	82
4	101	102
5	78	76
6	75	140

表8は、ネットワークで使用された全arc (i, j)に対し、設定されたアークコストを説明しています。(“–”は交差がないことを示しています。)

4.5 結果

vpn_data.dat の例題は、6 つのノードと 18 のアークを持つネットワークを説明し、総費用 566 のデザインを得ます。表 9 は各アークに設定される許容量のユニット数をレポートし、表 10 では起点（Source）から目的地への各需要がたどった経路を示しています。

Table 8: Arc costs

i/j	1	2	3	4	5	6
1	–	10	9	12	–	–
2	8	–	10	–	12	–
3	10	7	–	–	9	–
4	10	–	–	–	12	5
5	–	9	8	11	–	12
6	–	–	–	10	9	–

Table 9: Optimal arc capacity

i/j	1	2	3	4	5	6
1	–	2	2	13	–	–
2	5	–	0	–	0	–
3	5	0	–	–	0	–
4	10	–	–	–	4	4
5	–	0	0	4	–	0
6	–	–	–	7	0	–

Table 10: Optimal routes

Source	Destination	Path	Source	Destination	Path
1	2	1→2	4	1	4→1
1	3	1→3	4	2	4→1→2
1	4	1→4	4	3	4→1→3
1	5	1→4→5	4	5	4→5
1	6	1→4→6	4	6	4→6
2	1	2→1	5	1	5→4→1
2	3	2→1→3	5	2	5→4→1→2
2	4	2→1→4	5	3	5→4→1→3
2	5	2→1→4→5	5	4	5→4
2	6	2→1→4→6	5	6	5→4→6
3	1	3→1	6	1	6→4→1
3	2	3→1→2	6	2	6→4→1→2
3	4	3→1→4	6	3	6→4→1→3
3	5	3→1→4→5	6	4	6→4
3	6	3→1→4→6	6	5	6→4→5

5. ロバスト・ポートフォリオ最適化

このセクションでは、単一期間ポートフォリオ選択問題のロバスト最適化における定式化を説明しています。ロバスト最適化から得られる結果を比較するために、Monte-Carlo simulation を使用して解のロバスト性を数値化する方法も説明しています。

5.1　問題の説明

単一期間におけるポートフォリオ選択問題は、最大予想収益値を持つポートフォリオを作成するために所与のリストから資産の選択を行う問題です。資産は市場価格で購入するため、価格が変動します。投資予算額は固定され選択された各資産は一定の割合の利用可能な予算を使用します。各資産の市場価格はあるユニットを購入するために投資家が支払う額に応じて変動するため購入時に確定します。将来的な資産の価格は未知ですが、確率変数の分布特徴は既知と仮定します。

投資家はリスクを避けるために、ポートフォリオの最悪なケースにおける価格に関してある保証を設定したいと考えています。最悪な状況下での資産価値の下方変動は資産変動の 1.5 倍に達すると見込んでいます。言い換えると、1.5 倍を超える価格変動による資産価値の減少が最悪なケースだと考えています。例：資産#20 は市場価格 100$であり、資産#20 の変動性は+/- 10$と仮定します。そして資産価値を考慮した最悪なケースは 85$です。

この問題における投資家の目的はポートフォリオの予想収益価格を最大にしつつ、過度なリスクを避けることが目的です。保守的な投資家は、最悪な結果を避けようとするあまり、最悪なケースの価格を持つ資産に間違いなく予算の全てを注ぎ込むでしょう。残念なことに、この戦略はポートフォリオの予想収益値を劇的に減少させると考えられます。

別の極端な投資家は、最悪な状況下をあまり考慮せずに、最も高い予想収益値を持つ資産に予算のすべてを注ぎ込む楽観的な方針を持っています。2 つのアプローチは極端であり、同程度まで全ての資産価格は上下するという現実をまったく考慮に入れていません。通常、現実の場では、最小収益値が保証されている投資先よりも最大収益値の保証がある投資先をポートフォリオに選びます。

この問題は、機会制約付き最適化問題として知られています。セクション 5.4 の「結果」に、どのようにして、ロバスト最適化が最悪な状況下を避ける制約を守りつつ、最大収益を上げる解を導くかを示しています。

5.2　数理的定式化

集合 S は利用可能な資産を含み、インデックス $s \in S$ は集合 S から生じる資産を表現しています。資産 S の市場価格は $PRICE_s$ と表記され、この変動範囲は VAR_s を使って与えます。各資産 $s \in S$ に対し、変数 X_s は、資産を購入するために費やす予算の一部を表現しています。変数 W はポートフォリオの最悪なケースの価格を表現しています。変数 dev_s は、資産 S

の値の変動可能性を表現しています。値 N は投資家が考慮しうる最悪なケースを表現し、その標準偏差の倍数として表現された資産価値の最大減少に一致しています。

5.2.1　過度な対策

保守的な投資家は、ポートフォリオの最悪なケースを最大化すると予期できます。最悪な状況を避けるために、ポートフォリオの最悪なケースの価値を最大化しようとするでしょう。例題ではこの最悪なケース、全ての資産の価値がいつ N 時までにその標準偏差を下回るかを説明しています。

$$
\begin{aligned}
\max \quad & w \\
\text{s.t.} \quad & w \le \sum_{s \in S} (PRICE_s + N \cdot dev_s) \cdot x_s \ (\text{with } dev_s = -VAR_s) \\
& \sum_{s \in S} x_s = 1 \\
& 0 \le x_s \le 1
\end{aligned}
$$

5.2.2　予算の減少を回避する対策

セクション 5.4 の結果の説明を理解するために、最悪なケースの発生を確実に避けることは、平均的なケースにおいて、価値の重大な損失を招きます。従って平均的なケースが最も発生しうるので、このような戦略は賢い選択ではないでしょう。投資家は保護レベルを制限するために制御パラメータを追加してモデルの改善を行います。保護レベルを表現している割合を k、全体の変動予算を G で表現した下記の式を使い定義します。

$$G = k \sum_{s \in S} VAR_s^2$$

楕円体不確実性集合

利用可能な資産価格の減少は不確実性集合 U(G)を使って制御します。下記のように定義します。

$$U(G) = \{e : \sum_{s \in S} VAR_s \cdot e_s^2 \le G\}$$

ロバスト制約式

結果として得られるロバスト最適化問題は、下記の通りです。

$$
\begin{aligned}
\max \quad & w \\
\text{s.t.} \quad & w \le \sum_{s \in S} (PRICE_s + VAR_s \cdot e_s) \cdot x_s \quad (\forall e \in U(G)) \\
& \sum_{s \in S} x_s = 1 \\
& 0 \le x_s \le 1
\end{aligned}
$$

5.3　実装

前セクションの数理モデル下記の Mosel モデルで実装します。下記のモデルは、ロバスト最適化問題を生成するために楕円体不確実性集合をオリジナル問題に追加する方法を説明しています。また Monte-Carlo simulation method を使用し、解の品質を数値化する方法も示しています：NMC = 5000 のイタレーションでは、この価格（平均値）に集中した正規分布を適用し、その既知の変動を使って、すべての資産予想収益値に対するランダムな値を表します。異なる解の品質可用性を決定するため、目的関数の値（より正確に言うと、結果として得る解の値はその値の範囲に属している）の生成を考慮します。確立変数の分布関数の正確な成形を決定することが難しい場合、解の品質に関して洞察するのに、大変有益な方法です。

```
model "Robust portfolio optimization"
  uses "mmrobust", "random"

  parameters
    ZEROTOL=1e-6
    SEED=12345
  end-parameters

  declarations
    Problems: set of mpproblem
    ProtectLevel: set of real

    mp_problemA: mpproblem              ! Worst case optimization
    mp_problemB: array(ProtectLevel) of mpproblem  ! Robust optimization
    NSHARES = 25                        ! Max number of shares
    Shares = 1..NSHARES                 ! Set of shares

    PRICE: array(Shares) of real        ! Estimated return in investment
    VAR: array(Shares) of real          ! Uncertainty measure (deviation) per SHARE

    expReturn: mpvar                    ! Expected portfolio value
    wstReturn: mpvar                    ! Worst case portfolio value
    frac: array(Shares) of mpvar        ! Fraction of capital used per share
    frac_sol: array(Shares,Problems) of real
    obj: mpvar

    e: array(Shares) of uncertain       ! Deviation of share values

    N=1.5                               ! Worst case metric
  end-declarations

!***************************Subroutines*****************************
 !**** Create the nominal model ****
  procedure create_nominalmodel
    ! Nominal model
    expReturn = (sum(s in Shares) PRICE(s)*frac(s))
    wstReturn = sum(s in Shares) ( PRICE(s) - N*VAR(s) )*frac(s)

    ! Spend all the budget
    sum(s in Shares) frac(s) = 1
  end-procedure

!**** Optimize for the worst case realization ****
 procedure solve_det
   obj = wstReturn
   maximize(obj)
 end-procedure
```

```
!**** Optimize for a given protection level ****
 procedure solve_rob(k: real)
   ! The value variation domain
   G := k*sum(s in Shares) VAR(s)^2 + ZEROTOL
   sum(s in Shares) (VAR(s)*e(s))^2 <= G


    ! The robust constraint
    obj <= sum(s in Shares) (PRICE(s) + N*VAR(s)*e(s))*frac(s)

    maximize(obj)
  end-procedure

!***************************Main model****************************
  !**** Input data generation ****
  setrandseed(12345)
  forall(s in Shares | s<>NSHARES) do
    PRICE(s) := round(30+s*4+random*(2+s))
    VAR(s) := round((PRICE(s)/4)*random)
  end-do
  PRICE(NSHARES) := round(1.01*(max(s in Shares | s<>NSHARES) PRICE(s)))
  VAR(NSHARES) := round(PRICE(NSHARES)*0.99/N)

  writeln("Shares | Mean | S.Dev. | Worst value")
  forall(s in Shares)
    writeln(strfmt(s,6), " |", strfmt(PRICE(s),5), " |", strfmt(VAR(s),7),
      " |", strfmt(PRICE(s)-N*VAR(s),6,0))
  write("\n\n")

  !**** Optimize the worst case ****
  with mp_problemA do
    create_nominalmodel
    solve_det
    forall(s in Shares) frac_sol(s,mp_problemA) := frac(s).sol*100
  end-do

 !**** Print results ****
 write("\n\nShares | Wst price | Price |   A    |")
 forall(k in Ks) write(" k=", strfmt(k*100,2,1), "% |" ) ; writeln
 forall(s in Shares) do
   write(strfmt(s,6), " |", strfmt(PRICE(s)-N*VAR(s),10,0), " |",
     strfmt(PRICE(s),6), " | ")
   forall(mp in Problems) do
     if (frac_sol(s,mp)>1) then
       write(strfmt(frac_sol(s,mp),5,0), "% | ")
     else
       write("       | ")
     end-if
   end-do
   writeln
 end-do


!**** Simulate results (Monte-Carlo simulation) ****
forall(mp in Problems) do
  NMC:=5000
  expRev(mp) := 0.0 ; wstRev(mp) := 0.0
  cntBelow110(mp) := 0.0 ; cntAbove130(mp) := 0.0
  c := 0.0
  forall(i in 1..NMC, c as counter) do
    totalVal := 0.0


 forall(s in Shares | frac_sol(s,mp)>0) do
   v := maxlist(normal(PRICE(s),VAR(s)),0)
   totalVal += v*frac_sol(s,mp)/100
   expRev(mp) += v*frac_sol(s,mp)/100
```

```
        while(v >PRICE(s)-N*VAR(s)) v := maxlist(normal(PRICE(s),VAR(s)),0)
        wstRev(mp) += v*frac_sol(s,mp)/100
      end-do
      if (totalVal<110) then cntBelow110(mp) += 1
      elif (totalVal>130) then cntAbove130(mp) += 1
      end-if
    end-do
    expRev(mp) := expRev(mp) / c
    wstRev(mp) := wstRev(mp) / c
    cntBelow110(mp) := cntBelow110(mp) / c
    cntAbove130(mp) := cntAbove130(mp) / c
  end-do

  !**** Print simulation results ****
  write("\n                              |   A     |")
  forall(k in Ks) write(" k=", strfmt(k*100,2,1), "% |" )
  write("\n        Expected value      |")
  forall(mp in Problems) write(strfmt(expRev(mp),7,1), " |")
  write("\n        Worst case value    |")
  forall(mp in Problems) write(strfmt(wstRev(mp),7,1), " |")
  write("\n\n        P (value<110)       |")
  forall(mp in Problems) write(strfmt(cntBelow110(mp),7,2) , " |")
  write("\n        P (110<=value<130)  |")
  forall(mp in Problems) write(strfmt(1-cntBelow110(mp)-cntAbove130(mp),7,2),      " |")
  write("\n        P (value>130)       |")
  forall(mp in Problems) write(strfmt(cntAbove130(mp),7,2), " |")
  writeln

end-model
```

5.4　結果

不確実性集合がどのように解に影響を与えるかを理解するために、様々な保護レベル k に対するポートフォリオ割当問題を解き、保守的なアプローチに対する提案と比較していきます。投資家は 110 から 130 の範囲でポートフォリオの価値を予想できると知っています。従って、0-110、110-130、130-無限の 3 つの範囲に各々に対し、ポートフォリオの値がこの範囲に属する可能性の計算を望んでいます。

5.4.1　入力データ

図 11 は、資産平均値を示し、つまり市場価格（平均値）、値（S. Dev）の標準偏差、および N=1.5(最も悪い値)を持つ考慮された最悪なケースの値である資産の平均値を示しています。これは最悪なケースの値を持つけれど、予想収益値に対して最もよいケースを持っているので、資産#25 は、楽観的な投資家が好む資産であり、資産#22 は、最悪な値を持つため保守的な投資家にとって最善な選択であることが、下記のデータからわかります。

Table 11: Asset value distribution

Shares	Mean	S.Dev.	Worst value
1	35	2	32
2	42	8	30
3	47	11	31
4	49	3	45
5	51	12	33
6	60	1	59
7	61	8	49
8	64	16	40
9	72	10	57
10	81	12	63
11	85	6	76
12	83	5	76
13	83	15	61
14	91	2	88
15	102	8	90
16	97	1	96
17	109	18	82
18	107	12	89
19	126	30	81
20	132	19	104
21	128	11	112
22	124	4	118
23	136	22	103
24	130	12	112
25	137	90	2

5.4.2　分析する

名目の問題と比較し、ロバスト最適化がどのように動作するかを理解するために、名目の問題(A)と保護レベル k を使ってパラメータ化された 4 つのロバスト最適化問題を解いていきます。そして、保護レベルが k=0.0 の場合、偏差予算も 0 となり、従って解は非常に楽観的な値を得ます。

二番目のステップは、5 つの解を使って選ばれた資産の実質価値をシュミレーションし、ポートフォリオの値は、以前に投資家が決定した 3 つの値の範囲の 1 つが持つ可能性を計算するために、Monte-Carlo method を使います。

表 12 は上記の問題に対するポートフォリオ選択の提案です。分かりやすくするために、1 つの解に選ばれた資産のみをリストしています。保守的な解(A)は最悪なケースの値を使って予算のすべてを資産に割当てます。楽観的な解(k=0.0)は最も高い予想収益値を使って、予算のすべてを資産に割当てます。

改善した保護レベルによって、提案した解は最大の予想収益値と最悪なケースを使って、資産間のバランスを改善します。

表 13 は、各ポートフォリオ選択の提案に対するポートフォリオの値の Monte-Carlo simulation の結果を示しています。保守的な解(A)は最も悪いケースの値を持ち、予想されている通り値が 110 未満に下回るこのポートフォリオの値の予想収益は 0 ですが、投資家は、

ポートフォリオの価値が 130 を上回ることは極めて少ないと予想しているため、この解だと判断するかもしれません。

Table 12: Results of the parameterized robust optimization

Shares	Wst price	Price	A	k=0.0%	k=0.1%	k=1.0%	k=5.0%
19	81	126					9%
20	104	132				18%	16%
21	112	128				6%	12%
22	118	124	100%				7%
23	103	136			43%	31%	20%
24	112	130				12%	14%
25	2	137		100%	57%	34%	21%

楽観的な解は最大予想収益値ですが、投資家の目標である 110 のすぐ下までポートフォリオの値が達するリスクも極めて高くなっています。(38%)ポートフォリオの価値は、高い変動性をもっていることに注意が必要です。他の解(k>0)の動きは超過予算が増大した場合、予想収益値が減少する一方で、ポートフォリオの最悪なケースの値も同様に増大することを示しています。3 つの範囲を背景とした値の分布の不安定さは、減少傾向にあります。

Table 13: Results of the Monte-Carlo simulation

	A	k=0.0%	k=0.1%	k=1.0%	k=5.0%
Expected value	124.0	139.5	138.3	135.5	133.1
Worst case value	116.3	0.1	40.3	64.6	76.3
P (value ≤ 110)	0.00	0.38	0.30	0.22	0.13
P (110 ≤ value ≤ 130)	0.93	0.08	0.14	0.21	0.33
P (value ≤ 130)	0.07	0.53	0.56	0.57	0.54

k=5%の解は予想収益値と最悪なケースの値間のよいバランスを示しています。さらにこの解には 110 を上回るポートフォリオの値を導く大きなチャンスが(87%)あります。

5.5 参考資料

ここに掲載した問題は、[BTEGN09]に記載されている単一期間ポートフォリオ選択問題からヒントを得て作成した問題です。

6 ロバストなユニットコミットメント

6.1 問題説明

ユニットコミットメント問題は、電力需要量を満たすために、発電所の発電レベルをスケジュールする問題です。パワーシステムを安全に操業するために、常に発電量を需要量と一致させる必要があり、（パワーバランス制約と呼ばれる）この要求が満たせない場合、深刻な問題が発生し、停電する可能性があります。

発電機の発電レベルは、このコミットメントに影響されます。発電機のスイッチを切れば、

発電が停止し、発電機のスイッチがオンの状態の時にのみ発電しています。技術的な制約のため、発電レベルは、運転業務規定も考慮する必要があります。どの発電機を稼働させるか、どの発電機を停止したらよいかを決定するとき、最も重要なパラメータは、発電レベルの最小化です。
どの発電機を稼働するかを決定する時、上記以外の重要なパラメータは起動コストです。起動コストが低い発電機は計画期間中、オン/オフを複数回行うことができる一方、起動コストが高い発電機は一度起動させた後、継続的に稼働させることが効率的です。シンプルなモデルを作成するために、ランプ速度制約、シャットダウンコストまたは様々な予測タイプなどは考慮せず、モデルを作成しています。
例題：一日ごとのユニットコミットメント・スケジューリングを考慮します。業務計画の課題は、安定的な電力需要を満たすために、1 日の発電機の起動フェーズ/シャットダウン・フェーズを決定することです。しかし、スケジューリングを行うとき、正確な需要量は未知のためこれは最適化問題における不確実なパラメータと考えられます。この不確実性を扱うためによく用いられる方法は、十分な発電設備を所有し、厳密に要求されるより多くの発電機を稼働させることです。効率的にこの不確実性を考慮したロバスト最適化テクニックを適用する方法を下記で説明します。
リアルタイムで変更する必要がないスケジュールを持つ 2 つのユニットコミットメント問題を実装します。厳密に言うと、提案したユニットコミットメントスケジュールは十分な発電量を最終的に起動やシャットダウンをせずに、需要の供給が可能なことを確認します。最初のモデルはシナリオ不確性集合に基づき、実際の電力需要量が予測と異なっていても、ユニットコメットメントのステータスはいかなる部分修正も必要がないこと保証します。この問題では電力需要の実現性は、予測値と過去の電力需要曲線と類似すると仮定します。2 つめのモデルは、重要性不確実性集合に基づき、k まで同時に発電機の供給停止を制限しても、ユニットコミットメントステータスはいかなる部分修正も必要がないことを保証しています。

6.1.1　電力需要のバリエーションに対するロバスト性

電力需要のバリエーションに対する典型的なユニットコメットメントのロバスト性は、総電力量と稼働している発電機で得られる発電量との差をバウンドする追加制約を使って実現させます。この値は電力量レベルの上限として認識され、追加的な発電機の起動を行わずに安定した電力供給が可能な最大需要電力の増加を定義します。この経験値は過去の電力需要量から恣意的に決定します。
ここでは、過去データの電力需要量から形成されるロバスト制約の集合と予測需要量を置き換えることで制約と同様に、シームレスな実装が可能になるロバスト最適化のフレームワークを見ていきます。過去データと予測電力需要量は実際の電力需要量を決定するために使います。実際の電力需要はシナリオの不確実性集合を使って作成します。図 6 は予測電

力需要と並行する過去 6 年間の翌日の電力需要の過去データを示しています。

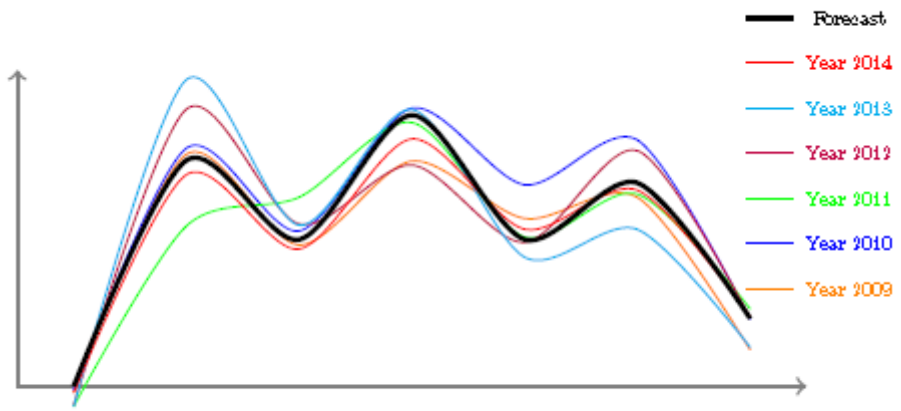

Figure 6: Historical power demand and forecast

6.1.2　不確実性 N － k に対するロバスト性

稼働中の発電機 k の損失結果である不確実性の影響を分析し、発電機の強制的な停止に対する典型的なユニットコミットメントのロバスト性を実現させます。総損失発電量が既存の発電機の予測発電量の上限より大きい場合、システムは危険な状態にあるため需要を満たすことができません。このようなシュミレーションを実践することは大変重要な作業であり、パワーシステムのセキュリティに関する考察を得ることができます。不運にも k に耐えうるスケジュール能力の欠如が証明された場合、モデルは安全な制約を充足させるスケジュールの改善を提案することができないでしょう。ロバスト最適化フレームワークは稼働中の発電機 k の損失をシュミレーションし、安定的な電力供給を行うために新しい発電機の起動/シャットダウンを提案することができます。重要性不確実性集合は、発電機の供給停止状態をモデル化するために使用します。

6.2　数理的定式化

連続的な期間の集合 Horizon は、調査の計画期間を説明します。期間 $t \in$ は様々な長さを持ちます。各期間 $t \in$ の領域は電力需要DEM_tと関連しています。

集合 Units は利用可能な発電機の集合を示しています。発電機の最小発電レベルは $PMIN_u$ であり、その最大能力は $PMAX_u$ です。単一発電機の稼働コストは$CSTR_u$となり、最小発電レベルでの稼働コストは$CMIN_u$です。

各時間に対する $PMIN_u$ の発電生産コストは$CADD_u$です。

各発電機 u は 3 つの意思決定変数に関連しています。バイナリ変数$start_{ut}$は期間 t のはじまりで発電機 u が稼働する場合、1 に等しくなります。バイナリ変数 $work_{ut}$ は発電機 u が稼働し、期間 t の間に稼働し続けるならば、1 に等しくなります。もう一つのバイナリ変数

$paddd_{ut}$は期間 t 中、発電機 u の発電レベルに設定します。

6.2.1　オリジナル　ユニットコミットメント問題

最小化する目的関数は予測された発電機の運転コストです。運転コストは起動コストと発電コストで構成されています。式は下記の通りです。

$$\sum_{u \in Units, t \in Horizon} CSTR_u \cdot start_{ut} + LEN_t \cdot (cmin_u work_{ut} + cadd_u \cdot padd_{ut})$$

起動の制約式は、発電機が稼働している間の期間を表現します、式は下記の通りです。

$$start_{ut} \geq work_{ut} - work_{ut-1}, \quad \forall u \in Units, if \quad t > 1$$
$$start_{u1} \geq work_{u1} - work_{uT}, \quad \forall u \in Units, if \quad t = 0$$

最大発電レベルの制約式は発電機の発電レベルで制限します。式は下記の通りです。

$$\leq (PMAX_u - PMIN_u) \cdot work_{ut} \quad \forall u \in Units, t \in Horizon$$

パワーバランス制約式は常に総電力生産が電力需要と等しくなることを確約します。下記の式を使って定式化します。

$$\sum_{u \in Units} PMIN_u \cdot work_{ut} + padd_{ut} = DEM_u, \quad \forall t \in Horizon$$

6.2.2　ロバストユニットコミットメント問題をロードする

ロードしたロバストユニットコミットメント問題は様々な電力需要の基づき安定的に供給するためのユニットコミットメントを制約化し、オリジナルのユニットコミット問題を拡張します。昨年分の電力需要の過去データは既知です。

不確実性集合のシナリオ

$Udem$ を将来的な電力需要の可能性とします。考察するために $Years$ を過去データの集合とし、$HDEM_{yt}$ は y 年の期間 t に対する需要とします。不確実性集合の式は下記の通りです。

$$Udem = \{e : \forall t \in Horizon, \exists y \in Years : e_t = HDEM_{yt}\}$$

ロバスト制約式

$$\sum_{u \in Units} PMAX_u \cdot work_{ut} \geq dem_t \quad \forall dem \in Udem, t \in Horizon$$

改善したエンジンは、結果的に生じる大量な過去データの集合（潜在的な大規模制約式）

をも効率的に処理します。

6.2.3　N-k 不測事態-制約付きユニットコミットメント問題

このN-k不測事態-制約付きユニットコミットメント問題はk発電機が同時に供給停止となった場合でも、委任した発電機による電力供給が可能なことを確認するために、オリジナルのユニットコミットメント問題を拡張します。不確実e_{out}は供給停止の状態を表現し、不確実性集合は強制的停止状態にある発電機の集合を表現しています。

重要性不確実性集合

Uoutageを強制停止状態におけるk発電機の組合せグループを説明する集合とします。

$$Uoutage = \{e : \sum_{u \in Units} e_u \leq k\}$$

ロバスト制約

$$\sum_{u \in Units} PMAX_u + work_{ut} \cdot (1 - e_u) \geq DEM_t \quad \forall e \in Uout, t \in Horizon$$

6.3　実装

6.3.1　オリジナルユニットコミットメントの実装

Moselコードは下記を実装し、ユニットコミット問題のオリジナル形式を解きます。

```
declarations
 NT = 7                              ! Number of time periods
 TIME = 1..NT                        ! Time periods
 PLANTS = 1..4                       ! Power generation plant
 UNITS: set of string                ! Power generation units

 LEN, DEM: array(TIME) of integer    ! Length and demand of time periods
 PMIN,PMAX: array(PLANTS) of integer ! Min. and max output of a generator type
 CSTART: array(PLANTS) of integer    ! Start-up cost of a generator
 CMIN: array(PLANTS) of integer      ! Hourly cost of gen. at min. output
 CADD: array(PLANTS) of real         ! Cost/hour/MW of prod. above min. level
 PLANT: array(UNITS) of integer      ! Unit generation plant

 YEARS: range                        ! Historical years
 HDEM: array(YEARS,TIME) of integer  ! Historical demand profiles

 start: array(UNITS,TIME) of mpvar   ! Is generation unit starting ?
 work: array(UNITS,TIME) of mpvar    ! Is generation unit up ?
 padd: array(UNITS,TIME) of mpvar    ! Production above min. output level
end-declarations

initializations from 'a6electr_ro_simple.dat'
 LEN DEM PMIN PMAX CSTART CMIN CADD PLANT HDEM
end-initializations

! Create decision variables
 forall(u in UNITS, t in TIME) do
  start(u,t) is_binary
  work(u,t) is_binary
 end-do
```

```
! Objective function: total daily cost
 Cost:= sum(u in UNITS, t in TIME) (CSTART(PLANT(u))*start(u,t) +
  LEN(t)*(CMIN(PLANT(u))*work(u,t) + CADD(PLANT(u))*padd(u,t)))

! Number of generators started per period and per type
 forall(u in UNITS, t in TIME)
  start(u,t) >= work(u,t) - if(t>1, work(u,t-1), work(u,NT))

! Limit on power production range
 forall(u in UNITS, t in TIME)
  padd(u,t) <= (PMAX(PLANT(u))-PMIN(PLANT(u)))*work(u,t)



 ! Power balance
  forall(t in TIME)
   sum(u in UNITS) (PMIN(PLANT(u))*work(u,t) + padd(u,t)) = DEM(t)

 ! Symmetry breaker
  forall(t in TIME, p in PLANTS)
   forall(u1, u2 in UNITS | PLANT(u1) = PLANT(u2) and u1<u2)
    work(u1,t) >= work(u2,t)

 !**** Solving and solution reporting ****
 ! Solve the original problem
  minimize(Cost)

  writeln("\n=== Nominal case ===\n")
  print_solution

 ! Add robust constraints
  robust_demand
 ! Solve robust the problem
  minimize(Cost)

  writeln("\n=== Robust against demand variation ===\n")
  print_solution
```

下記のアウトプットを印刷するルーチンは上記に記したモデルから呼び出されます。

```
!**** Print highest loss of load in case of worst case scenario realization
 procedure print_risk
   write("\n", strfmt("Loss of load",20))
   forall(t in TIME) do
     s:= sum(u in UNITS) (work(u,t).sol*PMAX(PLANT(u))) ! Total capacity
     v:= maxlist(DEM(t), max(y in YEARS) HDEM(y,t))     ! Hist. peak demand
     if(s < v) then
       write(strfmt(v-s,8))
     else
       write(strfmt("",8))
     end-if
   end-do
 end-procedure


!**** Solution printing ****
 procedure print_solution
   writeln("Total cost: ", getobjval)

   write(strfmt("Time period ",-20))
   ct:=0
   forall(t in TIME) do
     write(strfmt(ct,5), "-", strfmt(ct+LEN(t),2), "")
     ct+=LEN(t)
   end-do
```

```
  write("\n",strfmt("Up Units",-20))
  forall(t in TIME) write(strfmt(sum(u in UNITS) work(u,t).sol,8))
  write("\n", strfmt("Gen. Pwr.",20))
  forall(t in TIME)
    write(strfmt(sum(u in UNITS) (work(u,t).sol*PMIN(PLANT(u))+padd(u,t).sol),8))
  write("\n", strfmt("Gen. Cap.",20))
  forall(t in TIME)
    write(strfmt(sum(u in UNITS) (work(u,t).sol*PMAX(PLANT(u))),8))
  write("\n", strfmt("Res. Dn",20))
  forall(t in TIME) write(strfmt(sum(u in UNITS) (padd(u,t).sol),8))
  write("\n", strfmt("Res. Up",20))
  forall(t in TIME)
    write(strfmt(sum(u in UNITS) work(u,t).sol*PMAX(PLANT(u)) - DEM(t),8))

  print_risk


  writeln
end-procedure
```

6.3.2　ロード　ロバストユニットコミットメントの実装

ロバスト最適化をベースとするシナリオの拡張は下記のサブルーチンで実装します。

```
!**** Helper routine to create scenario uncertain set ****
  public procedure makescenario(a:array(r1:range,c1:range) of integer,
  u:array(c2:range) of uncertain)
    declarations
      aa:dynamic array(r0:range,U:set of uncertain) of real
    end-declarations
    forall(i in r1, j in c1|exists(a(i,j)) and exists(u(j)),n as counter) do
      aa(n,u(j)):=a(i,j)
    end-do
    scenario(aa)
  end-procedure


!**** Robust against past scenario ****
 public procedure robust_demand
   declarations
     demand: array(TIME) of uncertain    ! Uncertain power demand
   end-declarations

  ! Add uncertainty set (time correlation)
   makescenario(HDEM,demand)

  ! Security reserve upward
   forall(t in TIME) sum(u in UNITS) PMAX(PLANT(u))*work(u,t) >= demand(t)

 end-procedure
```

6.3.3　N-k 不測事態-制約付きユニットコミットメントの実装

手続き robust_demand の呼び出しを下記に記した手続き robust_contingency の呼び出しと置き換えることで、N-k 不測事態の制約付きユニットコミットメントモデルの問題を定式化することができます。

```
!**** Ensure enough power production capacity allows to
!**** satisfy load even if 'k' units are forced in outage.
 procedure robust_contingency(k: integer)
   declarations
```

```
  outage: array(UNITS,TIME) of uncertain ! uncertain event
end-declarations

forall(t in TIME, u in UNITS) outage(u,t) >= 0
! In forced outage, loosing 100% of
! the unit generation capacity
forall(t in TIME, u in UNITS) outage(u,t) <= 1

! forall(t in TIME) sum(u in UNITS) outage(u,t) <= k
forall(t in TIME) cardinality(union(u in UNITS) {outage(u,t)}, k)

forall(t in TIME) sum(u in UNITS) PMAX(PLANT(u))*work(u,t) -
  sum(u in UNITS) PMAX(PLANT(u))*work(u,t)*outage(u,t) >= DEM(t)
!forall(t in TIME)
! sum(u in UNITS) PMAX(PLANT(u))*work(u,t)*(1 - outage(u,t)) >= DEM(t)

end-procedure
```

6.4　結果

ここに、3 つのモデルから導かれた結果を示します。オリジナルユニットコミットメント問題を基本的なシナリオとして使用し、2 つのロバスト・バージョンのモデルをオリジナルモデルと比較しています。

3 つのモデル定式化は、技術的な制約を満たし、ロードを供給する実現可能なユニットコミットメントを見つけます。テスト例（タイプ、最小発電レベルと最大発電レベル、最小レベルで稼働時のコスト、最小発電レベルと最大発電レベル間の可変コスト、起動時のコストを固定）で使用された発電機の特徴は表 14 に示しています。

Table 14: Description of power generators

Unit type	Number	Pmin	Pmax	Fix cost	Add. MW cost	Start-up cost
1	10	750	1750	2250	2.7	5000
2	4	1000	1500	1800	2.2	1600
3	8	1200	2000	3750	1.8	2400
4	3	1800	3500	4800	3.8	1200

表 15 は名目の最適化問題の結果を示します。各期間で、起動させた発電機の数は（Up Units）発電力(Gen. Pwr)とともに記載しています。発電機の容量（Gen. Cap）は起動した発電機で発電可能な最大発電力の合計です。カラムの予約の下降(Res. Dn)と予約の上昇はそれぞれ、最大ロードの減少または、いかなる発電機の起動や停止を行わずに、増加ロードを安全にサポートできるかを示しています。例えば、最初の期間(0-6)の間、ロードは 12000 kWh（電力需要の予測値）から 13250 kWh まで（電力需要の予測値+予約の上昇分）の増加に追加的な発電機の起動を行わずに対応することが可能です。

最後の 2 行は最悪なシナリオが実現した場合における最大ロード損失を示しています。需要のバリエーションに関する最悪なシナリオは最大の発電力増加を要求するロード・プロファイルの実現です。

不測事態の対策に関して、k 'critical'ユニットが供給停止を強制されたときに最悪なシナリオが生じます。発電機の臨界はその発電レベルとすべての条件に依存します。全ロードの

大部分を供給する発電機のように、上昇する需要に対する準備がない発電機は危機的状況にあります。

Table 15: Nominal case (Total cost: 1.40679 Million)

	0-6	6-9	9-12	12-14	14-18	18-22	22-24
Up units	8	18	15	20	15	17	11
Gen. Pwr.	12000	32000	25000	36000	25000	30000	18000
Gen. Cap.	13250	37750	27250	44750	27250	34250	19250
Res. Dn	3850	10050	8450	10450	8450	9850	6250
Res. Up	1250	5750	2250	8750	2250	4250	1250
Loss of load							
Demand variation	0	1395	1491	0	2592	0	0
2 Contingencies	2750	1250	1750	0	1750	2750	2750

Table 16: Robust against demand variation (Total cost: 1.409 Million)

	0-6	6-9	9-12	12-14	14-18	18-22	22-24
Up Units	8	18	15	20	15	17	11
Gen. Pwr.	12000	32000	25000	36000	25000	30000	18000
Gen. Cap.	13250	39250	28750	44750	30250	37250	19250
Res. Dn	3850	9450	7850	10450	7250	8650	6250
Res. Up	1250	7250	3750	8750	5250	7250	1250
Loss of load	0	0	0	0	0	0	0

名目のケースが適用され、需要バリエーションのケースが適用されるならば、朝の電力需要のピーク時に 1395 kWh の需要が供給されないリスクがあります。通常、ロード損失リスクがある全ての時間に、リスクを補うために追加の発電機を起動させます。しかし、表 16 に示している需要バリエーション問題に対するロバスト定式の結果は、単にコミットメントした発電機を変更するだけで、リスクを回避できることを示しています。

Table 17: Robust against 2 contingencies (Total cost: 1.420 Million)

	0-6	6-9	9-12	12-14	14-18	18-22	22-24
Up Units	9	18	16	20	15	17	11
Gen. Pwr.	12000	32000	25000	36000	25000	30000	18000
Gen. Cap.	19750	39250	32250	44750	33250	38750	25250
Res. Dn	850	9450	6050	10450	6050	8050	3850
Res. Up	850	9450	6050	10450	6050	8050	3850
Loss of load	0	0	0	0	0	0	0

2 つの不測事態のケースにおけるロード損失は最初の期間で 2750 kWh の供給が不可能になるリスクが潜在的にあることを強調しています。これは総需要の 20%が補えないことを意味しています。実際、需要が上昇した場合に利用できる余分の電力は 1250 kWh のみであり、これは全能力で稼働している発電機の 1 つが停止した場合（1500 kWh)電力需要を供給することができないことを意味しています。先のケースのように、通常の結果は追加的に発電機を起動させることです。しかし、表 17 で示している 2 つの不測事態の問題に対するロバスト定式による結果は、ロード損失のリスクを冒さずにかつ、過剰な発電機の追加を行わ

ずに（そして高い起動コストを導かずに）ユニットコミットメントのスケジュールを見つけることが可能なことを示しています。

6.5　参考資料

ここに掲載した名目のユニットコミットメントモデルは Applications of optimization with Xpress-MP' [GHPS02]に掲載されている問題に基づきます。

7.電力供給不確実性に基づく生産計画

7.1　問題の説明

ある会社は液体窒素と液体酸素（以下、LIN と LOX と称す）を製造しています。各 8 時間のシフトおよび、特定されている翌 N 期間における生産計画を立てる必要があります。資源調達は困難ではありませんが、(2 つの要素が対流圏の大部分を構成) 2 つの液体を得るためのプロセスは莫大な電力を必要とします。(蓄蔵された液体ガスを冷蔵するための動力と工場に動力を供給するための電力)

エネルギーコストが生産コストの大部分を占めると仮定すると、電力供給者は、複数の電力供給の契約を提供します：Interruptible Load Contracts（略：ILC）がその 1 つで、電力供給者は急な通知（数分前）により、制限付き期間内で工場など大規模な顧客への電力供給を中断することができます。：おそらく電力供給者は真夏日など、電力需要がピークに達するときに、ILC を利用します。しかし、ILC は計画期間中に k 中断があることのみを義務付ける契約です。

ここでは電力供給における不確実性に基づいた生産計画問題に取り組んでいきます。インプットに生産、在庫費用、最大の生産、在庫容量、初期の在庫、各期間に対する LIN と LOX の需要および中断の最大期間 k を与えます。

需要を契約条項の範囲内で起こりうる電力供給の中断とは無関係に LIN と LOX 生産量を毎日満たすことから問題を構成しています。この最適化問題はいくつかある理由の 1 つによりロバスト最適化アプリケーションが必要です：この会社は顧客に病院があります。このため LOX の需要を満たす必要があります。

7.2　数理的定式化

ガス $G = \{lin, lox\}$ の集合と期間 $T = \{1, 2, \ldots, N\}$ の集合、各期間に対する最大生産レベル P_{lin} と P_{lox} および各ガスに対する在庫容量 S_{lin} と S_{lox} を与えます。$t \in T, g \in G$ に対する重要はパラメータ dem_{tg} を使用して与えます。不確実性集合は中断 $(\xi)_{i=1..n}$ の各一次元配列を使用して定義します。つまり $\xi \in [0, 1]$ かつ、この要素 K 以下での実現を定義します。

$$\sum_{i=1}^{n} \xi_i \leq K.$$

生産レベルに変数$prod_{tg}$、在庫レベルに変数inv_{tg}を定義し、ガス G に対し期間 t を定義します。インプットデータに不確実性がなければ、モデルは単純な制約式となります：変数と初期の在庫レベルの固定を制約します。

$$0 \leq prod_{tg} \leq P_g \quad \forall g, \forall t \in \{1, 2, \ldots, N\}$$

$$0 \leq inv_{tg} \leq S_g \quad \forall g, \forall t \in \{1, 2, \ldots, N\}$$

$$0 \leq inv_{0g} = I_g \quad \forall g;$$

そして生産計画の制約は各ガス g と期間 t に対する大規模な保全制約から構成されます。

$$inv_{t-1,g} + prod_{tg} = inv_{tg} + dem_{tg} \quad \forall t \in T, g \in G.$$

モデル化はこのケースに、深く関係します。最初の試みは$\xi_t = 1$ならば、期間 t の実際の生産はゼロとなるので、$(1 - \xi_t)$で生産の変数の乗算でロバスト制約を表現します。

$$inv_{t-1,g} + (1 - \xi_t) prod_{tg} = inv_{tg} + dem_{tg} \quad \forall t \in T, g \in G.$$

しかしこの式は実行可能解を導きません：期間 t の生産と関連した各ロバスト制約は、不確実性 t に設定されるため、生産を導きません。生産と需要間のバランスが、毎回必ず非負の在庫を導くような方法で生産制約を集める必要があります。上記の制約式を下記の制約式に置き換えます。

$$inv_{0,g} + sum_{\tau \in T; \tau \leq t}((1 - \xi_\tau) prod_{\tau g} - dem_{\tau g}) \geq 0 \quad \forall t \in T, g \in G.$$

このロバスト制約は中断における不確実性を考慮し、K 中断を持つ全ての需要を満たす生産計画も考慮しています。輪郭線の条件と初期在庫を考慮するために、不確実性集合を若干修正し、期間 1 で中断が発生しないことを要求します。

$$\xi_1 = 0.$$

最後に、在庫のロバスト値は、生産を計画した後に、いかなる中断も実際に起こらない状況を考慮しなければなりません。：累積した需要を使用して割引いた全ての累積生産の合計として各期間で在庫を制約する単純な制限です。

$$inv_{tg} \geq inv_{0,g} + sum_{\tau \in T; \tau \leq t}(prod_{\tau g} - dem_{\tau g}) \quad \forall t \in T, g \in G.$$

7.3　実装

Mosel を使った実装は下記の通りです。注：非負の在庫制約が同一の不確実性集合を複数回使用するためオプション ROBUST_OVERLAP_UNCERTAIN を再び使用します。

```
model robust_prodplan

  uses "mmrobust"

  parameters
    plan_data = "prodplan_robust.dat"
  end-parameters

  declarations
    NDAYS:    integer                      ! Planning horizon
    PERDAY:   integer                      ! Number of periods per day
    NPERIODS: integer                      ! Total number of periods

    PERIODS,PERIODS0: range                ! Time periods
    GASES: set of string                   ! Set of products


     PROD_CAP: array(GASES) of real         ! Production capacity every day
     INV_CAP:  array(GASES) of real         ! Inventory capacity
     INV_0:    array(GASES) of real         ! Initial inventory

     PROD_COST: real                        ! Production cost
     INV_COST:  real                        ! Inventory cost

     DEMAND: array (PERIODS, GASES) of real ! Demand of each gas every day

     MAX_NINTERR: integer                   ! Maximum number of interruption
                                            ! (as per contract)
  end-declarations

!**** Initialize data ****
  initializations from plan_data
    NDAYS PERDAY
    PROD_CAP  INV_CAP
    PROD_COST INV_COST INV_0
    DEMAND
    MAX_NINTERR
  end-initializations

  NPERIODS := NDAYS * PERDAY
  PERIODS0 := 0..NPERIODS

!**** Problem formulation ****
  declarations
    produce:   array     (PERIODS, GASES) of mpvar ! Production every day
    inventory: array (PERIODS0, GASES) of mpvar    ! Inventory level every day,
                                                   ! including initial level

    interruption: array (PERIODS) of uncertain     ! Is power cut at this time?
  end-declarations


!**** Constraints ****

  ! Start inventory
  forall(g in GASES)
    inventory (0,g) = INV_0 (g)

  ! Inventory balance
  forall(t in PERIODS, g in GASES) do
```

```
    inventory(0,g) + sum(tp in PERIODS | tp <= t)
      ((1 - interruption(tp)) * produce(tp,g) - DEMAND(tp,g)) >= 0

    inventory (0,g) + sum (tp in PERIODS | tp <= t)
      (produce(tp,g) - DEMAND(tp,g)) <= inventory(t,g)

    inventory(t,g) <= INV_CAP(g)
    produce(t,g)   <= PROD_CAP(g)

  end-do

  ! Interruptions of production
  forall (t in PERIODS) do
    interruption (t) <= 1
    interruption (t) >= 0
  end-do

  sum(t in PERIODS) interruption (t) <= MAX_NINTERR
  interruption(1) = 0

!**** Solving ****
  setparam("robust_uncertain_overlap", true)

  ! Set verbosity level


      setparam("xprs_verbose", true)

      ! Objective function: total cost of production and storage
      minimize(sum (t in PERIODS, g in GASES)
                  (PROD_COST * produce (t,g) + INV_COST * inventory(t,g)))

    !**** Soution reporting ****
      writeln("\nNumber of interruptions: ", MAX_NINTERR)
      writeln("\nOptimal solution has cost ", getobjval)

      COLWIDTH := 6

      forall(g in GASES) do

        writeln("\nProduction of ", g)

        write(strfmt ("Time",-COLWIDTH))
        forall(t in PERIODS0) write (strfmt(t,COLWIDTH))

        write("\n", strfmt("Dem",-2*COLWIDTH))
        forall(t in PERIODS) write(strfmt(DEMAND(t,g),COLWIDTH,1))

        write("\n", strfmt("Prod",-2*COLWIDTH))
        forall(t in PERIODS) write(strfmt (produce(t,g).sol,COLWIDTH,1))

        write("\n", strfmt("Inv*",-COLWIDTH))
        forall(t in PERIODS0)
          write (strfmt(inventory(0,g).sol +
                 sum (tp in PERIODS | tp <= t)
                   (produce(tp,g).sol - DEMAND(tp,g)),COLWIDTH,1))
        writeln("")

      end-do

    end-model
```

7.4　インプットデータ

この例題では、5 日間、日ごとに 3 つの時間枠を持つ計画の簡単な例題を考慮します。生産コストが 4、在庫コストが 3 となり、ILC 契約は 4 回の中断を考慮しています。下記の表 18 は生産能力と在庫容量および 2 つのガスに対する初期在庫レベルを示しています。

Table 18: Production and inventory capacity

Gas	Prod. capacity	Inv. capacity	Initial inv.
LIN	29	60	20
LOX	5.5	27	20

表 19 は、全 15 期間の各ガスに対する需要を示しています。

Table 19: Customer demand

Gas/Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LIN	6	14	10	8	11	15	10	9	10	9	10	12	11	15	9
LOX	2	5	3	4	8	4	8	7	5	4	3	3	5	9	7

7.5 結果

この問題の最適解はコスト 4727.17 です。表 20 と表 21 は中断の発生に関わらず、需要を満たすために要求される最適な生産レベルを示しています。生産レベルは最初が最も高く、液体酸素関しては、需要と等しいことに留意ください。これは初期期間で生産が計画期間内において早期に K=4 の連続的な中断のような最悪なケースを仮定しなければならないという事実に依存しています。在庫レベルが需要と生産のバランスと等しく、中断が発生しない状況を考慮する必要があります。

Table 20: Production of LIN

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Dem	6	14	10	8	11	15	10	9	10	9	10	12	11	15	9
Prod	29	15	15	15	15	15	10	9	10	9	10	12	11	15	9
Inv*	43	44	49	56	60	60	60	60	60	60	60	60	60	60	60

Table 21: Production of LOX

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Dem	2	5	3	4	8	4	8	7	5	4	3	3	5	9	7
Prod	5.5	5.2	5.2	5.2	5.2	5.2	5.2	5.2	5	5.2	5.2	5.2	5.2	5.2	5.2
Inv*	23.5	23.7	25.8	27	24.2	25.3	22.5	20.7	20.7	21.8	24	26.2	26.3	22.5	20.7

最後に、期間の最初で解かれた最適化問題の結果であることに留意ください。rolling horizon approach,などを使用することで、既に発生した中断を考慮しつつ、すべての期間でモデルを再び解くことで、計画した生産レベルを変更する必要が起こりうるが、高額にならない生産計画を得ることができます。

7.6 参考資料

この例題は[LBS13]のドキュメントに掲載されている現実世界のアプリケーションの簡易版です。

Bibliography

[BS04] D. Bertsimas and M. Sim. The price of robustness. Operations research,

52(1):35–53, 2004.
[BTEGN09] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. Robust Optimization.
Princeton Series in Applied Mathematics,
2009.
[GHPS02] C. Guéret, S. Heipcke, C. Prins, and M. Sevaux. Applications of Optimization with Xpress-MP. Dash
Optimization, Blisworth, UK, 2002.
[LBS13] C. Latifoglu, P. Belotti, and L.V. Snyder. Models for production planning under power interruptions. Naval
Research Logistics (NRL), 60(5):413–431, 2013.