シラバス参照[2016年度／最適化・シミュレーション演習／今泉　淳／大久保　寛基／椎名　孝之]

編 集　印 刷

授業情報

開講年度	2016年度	開講箇所	大学院創造理工学研究科		
科目名	最適化・シミュレーション演習				
担当教員	今泉　淳／大久保　寛基／椎名　孝之				
学期曜日時限	秋学期　01:金1時限				
科目区分	講義	配当年次	1年以上	単位数	2
使用教室		キャンパス	西早稲田(旧大久保)		
科目キー	5201031084	科目クラスコード	01		
授業で使用する言語	日本語				
コース・コード	MGTX66ZS				
大分野名称	経営工学				
中分野名称	経営工学				
小分野名称	数理工学(含OR, 確率・統計)				
レベル	修士レベル	授業形態	演習／ゼミ		

シラバス情報

授業概要	学部2年必修科目「基礎オペレーションズリサーチ／オペレーションズリサーチ演習」の知識を前提として，数理計画による最適化と（離散事象型）シミュレーションに関する演習を行う．使用するソフトウェアは，AMPL（＋GurobiまたはCPLEX)，および，Simul8を想定している．最適化の演習では，AMPLを用いて線形計画問題や各種の組み合わせ最適化問題を解けるようにするとともに，列生成法，ラグランジュ緩和法，ベンダースの分解原理などのより高度な求解テクニックを学ぶ．一方，離散型シミュレーションの演習では，C，および，Simul8を用いて様々な待ち行列系システム，離散事象システムのモデル化，実験の進め方，簡単な分散減少法と結果の分析について学ぶ．さらに，シミュレーションを用いた最適化の演習を行う．演習では，数理計画による最適化やシミュレーションの実践的能力を身につけることを目指す。履修者は，Cを使用できる環境を有するPCを持参すること。受講者は，実験室にて，演習で使用する C，AMPL，Simul8をダウンロードできる。
授業の到達目標	実際に数理計画やシミュレーションのソフトウェアを使用して，具体的問題を解き，結果を適切に読みとることができるようにすると同時に，数理計画やシミュレーションの基礎概念の理解を深める．数理計画に関しては，(1)AMPLを使って基本的な数理計画問題が解けること，(2)列生成法，ラグランジュ緩和法やベンダース分解原理などのより高度な使い方を理解すること，(3)ナップザック問題，施設配置問題，ロットサイズ決定問題，最短路問題など代表的な定型的最適化問題の理解を深め，実際に解けること，を本授業の到達目標とする．また，シミュレーションに関しては，(1)汎用の離散型離散型シミュレーションソフトウェアで簡単なモデルを作れ，適正な形でシミュレーション実験が行えるようにすること，(2)Cのような汎用言語で簡単なシミュレーションを行うことによって，コンピュータがどのようにしてシミュレーションを実行しているかのメカニズムを理解すること，(3)ある程度複雑な離散事象型の問題に対して，適切なモデル構築とシミュレーションが実行できるようにすること，を本授業の到達目標とする．

授業計画	第1回　09/30 本演習の目的と概要，AMPLイントロダクション（椎名） 本演習の目的，概要 集合，配列，ソーティング，乱数，その他 第2回　10/07 AMPLによる基本的な数理計画モデルの作成と求解（椎名） 線形計画問題，ナップザック問題，施設配置問題など 第3回　10/14 分枝限定法（椎名） 第4回　10/21 動的計画法（椎名） ナップザック問題，動的ロットサイズ決定モデル（Wagner-Whitinモデル） 第5回　10/28 列生成法（椎名） カッティングストック問題 第6回　11/11 ラグランジュ緩和法（椎名） 時間制約付き最短路問題，施設配置問題など 第7回　11/18 ベンダース分解原理（椎名） 施設配置問題 第8回　11/25 シミュレーションの種類とモンテカルロシミュレーション（今泉・大久保） シミュレーションの種類，モンテカルロシミュレーション，一様乱数，C による円周率πの推定 注意：　第9回以降の予定は，当初予定から変更されています． 第9回　12/02　Simul8によるシミュレーション（今泉・大久保） Simul8とは，単一サーバー待ち行列，M/M/1，並列キャッシュマシンの並び方，有限バッファ直列システム，レンタルビデオの本数 第10回　12/09　ラベル，リソース，シミュレーション実験の進め方と結果の表示・分析（今泉・大久保） ラベル，リソース，試行，シミュレーション実験の進め方，結果の表示と分析 第11回　12/16　より複雑なシミュレーション，分散減少法（今泉・大久保） 宅配ピザ屋のオーブン容量とバイク台数，分散減少法（円周率πの推定，M/M/1） 第12回　2017/01/06 離散事象型シミュレーションの考え方（今泉・大久保） 事象，事象カレンダ，事象ルーチン，事象制御ロジック，単一サーバ待ち行列モデル実装の考え方 第13回　2017/01/13 CによるM/M/1モデルの実装の講評（今泉・大久保） 指数乱数など，実装，結果の分析，解析結果との比較 第14回　2017/01/20　シミュレーションを用いた最適化（今泉・大久保） サーバー数やバッファ配分の最適化など 第15回　未定　自作の最適化モデルとシミュレーションモデル，および，それらを用いた分析（椎名）
教科書	テキストを配布
参考文献	久保幹雄，田村明久，松井知己，「応用数理計画ハンドブック」，朝倉書店，2002. L. Wolsey，Integer Programming，Wiley，1998. 森戸晋，逆瀬川浩孝，「システムシミュレーション」，朝倉書店，2000. J. Banks, J.S. Carson, II, B.L. Nelson, and D.M. Nicol, Discrete-Event System Simulation, Third Edition, Prentice Hall, 2001. 伏見正則，逆瀬川浩孝（監訳），「モンテカルロ法ハンドブック」，朝倉書店，2014.
成績評価方法	割合 / 評価基準 試験: 0%　通常の試験は実施しない． レポート: 40%　原則として毎週レポート課題（宿題）を出す．課題に対して，多くの場合，モデルを自分で構築し，ソフトウェアにかけて問題を解き，適切な考察を加えて，レポートにまとめる．レポートは，作成したモデルやプログラムの内容，動作，納期を評価する． 平常点評価: 60%　できる限り，授業（演習）時間内に内容の理解につとめる．授業時間内に出す演習課題を解くことによって，最適化やシミュレーションを実際に解けるかをみる．受講生の演習時間内の取り組みや具体的問題に対する実装結果などを評価する．また，受講生からの積極的質問や演習時間内の積極的取り組み，議論への参加を評価の対象とする． その他: 0%　なし．
備考・関連URL	4回以上の欠席，もしくは遅刻は不合格とする． 「基礎オペレーションズリサーチ／オペレーションズリサーチ演習」の知識を前提とする．「オペレーションズリサーチA」も履修済であることが望ましい． 試験を行わないため，成績評価はレポートと平常点評価をもとに行う． 講義内での演習時間には限りがあるため，手法の理解とプログラミングは講義外で時間をかけることが必要である．

編　集　印　刷

最適化・シミュレーション演習

第1回 AMPLの基礎

- 授業サポートページ
  http://www.shiina.mgmt.waseda.ac.jp/optsim/
- 数理計画による最適化と(離散事象型)シミュレーションに関する演習を行う. 使用するソフトウェアは, AMPL(+GurobiまたはCPLEX), および, Simul8を想定している.
- 演習では, 数理計画による最適化やシミュレーションの実践的能力を身につけることを目指す。履修者は, Cを使用できる環境を有するPCを持参すること。受講者は, 実験室にて, 演習で使用する C, AMPL, Simul8をダウンロードできる。

主な内容(最適化)

- 最適化の演習では, AMPLを用いて線形計画問題や各種の組み合わせ最適化問題を解けるようにするとともに, 列生成法, ラグランジュ緩和法, ベンダースの分解原理などのより高度な求解テクニックを学ぶ.
- **第1回 本演習の目的と概要, AMPLの基礎**
  本演習の目的, 概要, 集合, 配列, ソーティング, 乱数, その他
- **第2回 AMPLによる基本的な数理計画モデルの作成と求解**
  線形計画問題, ナップザック問題, 施設配置問題など
- **第3回 動的計画法**
  ナップザック問題, 動的ロットサイズ決定モデル(Wagner-Whitinモデル)
- **第4回 分枝限定法**
- **第5回 列生成法** カッティングストック問題
- **第6回 最短路問題** 時間制約付き最短路問題, 施設配置問題
- **第7回 ベンダース分解原理** 施設配置問題
- 第3回以降は独立したテーマで、取り扱う内容は前後する可能性があります。

主な内容(シミュレーション)

- 離散型シミュレーションの演習では, C, および, Simul8を用いて様々な待ち行列系システム, 離散事象システムのモデル化, 実験の進め方, 簡単な分線減少法と結果の分析について学ぶ. さらに, シミュレーションを用いた最適化の演習を行う.
- **第8回 シミュレーションの種類とモンテカルロシミュレーション**
  シミュレーションの種類, モンテカルロシミュレーション, 一様乱数, C による円周率πの推定
- **第9回 離散事象型シミュレーションの考え方** 事象, 事象カレンダ, 事象ルーチン, 事象制御ロジック, 単一サーバ待ち行列モデル実装の考え方
- **第10回 CによるM/M/1モデルの実装**
  指数乱数など, 実装, 結果の分析, 解析結果との比較
- **第11回 Simul8によるシミュレーション**
  M/M/1, 並列キャッシュマシンの並び方, レンタルビデオの本数
- **第12回 シミュレーション結果の分析と分散減少法**
  定常状態シミュレーションの留意点, 実験の進め方, 分散減少法(円周率πの推定, M/M/1)
- **第13回 やや複雑なSimul8のモデル**
- **第14回 シミュレーションを用いた最適化** サーバー数の最適化
- **第15回 未定 自作の最適化モデルとシミュレーションモデル**, および, それらを用いた分析

AMPL (A Modeling Language for Mathematical Programming)

- 数理計画問題用モデリング言語(Bell Laboratory)
- 各種数理計画問題をモデル化し、記述
- AMPLモデルは、数理計画ソルバー(CPLEX, lp_solve, MINOS, Xpress-MP, Gurobi など)を用いて求解
- 同時に、ソルバーの反復使用機能により、様々の解法アルゴリズムを実装できることが大きな特徴
- AMPLのホームページ http://www.ampl.com/
- 解説書「AMPL: A Modeling Language for Mathematical Programming」(by Robert Fourer (数理計画法), David M. Gay, and Brian W. Kernighan『プログラミング言語C』, Duxbury Press / Brooks/Cole Publishing Company, 2002. ISBN 0-534-38809-4)上記HPから読める
- C言語と関連

AMPL コマンドスクリプト

- AMPLのコマンドの列は *.runファイル（バッチファイル）に記述（複数のコマンドを逐次実行）
- 実行はコマンドプロンプトから「ampl *.run」と入力（*はファイル名）
- コマンドスクリプトの終わりには「exit;」を記述し、AMPLを終了
- コメントを記入したい場合は、行頭に「#」をk記入#以降が無視される
- コマンドを記述したら、行の終わりに「;」（セミコロン）を記入

5

表示について

- display ＊; で＊の内容が画面に表示
- display ＊ > ファイル名; とすると画面ではなくファイルに書き込まれる
- display文は改行が複数行われるため、表示を制御したい場合はprintf文を使用
- 例：「printf : "i= %d ¥n", j; 」の意味は、「j」の内容を「i= %d ¥n」という形式で表示させる。
- 「%d」は整数型、「%f」は実数型
- 記述形式はC言語に準拠
- 「¥n」は改行コード
- まとめると、「printf : "i= %d ¥n", j; 」は「i=」をテキストとして表示した後に、「j」を形式「%d」で表示して改行するというコマンド

6

パラメタ、変数の定義、代入文

- 一般的にはコマンドスクリプトの中で使われるパラメタ（定数、あるいは式の値など）は最初に定義しなければならない。
- パラメタの定義は次のように行う。
- param a;「aというパラメタを定義」
- var x;「xという変数を定義」
- AMPLでは変数とは、数理計画問題の定式化で用いられる変数（ソルバで最適化されるもの）を表す
- 変数や反復回数などに応じて変わる値はすべてパラメタ
- パラメタや変数に値を代入する場合、単純に「a=a+1;」と記述してはならない。
- 正しくは、let文を「let a:=a+1; 」（コロン＋等号）と記述

7

反復と条件

- 反復 for 文
- 「a..b」（ただし、a<b は整数値）はaからbまでの整数値の集合
- #カウンタiの値が1から10まで文1を実行
- for {i in1..10}
  {
  文1;
  }
- 注意事項：for文で使われたfor {i in 1..5}などに含まれる反復回数を示すカウンタ「i」は、for文の中でのみ使われていれば、パラメータとして定義する必要はない。

- repeat while 文
- repeat while{条件}
  {
  文1：
  }
- #条件が成立する限り文1を実行
- If-then-else 文
  if(条件 (and/or 条件)) then
  {
  文1;
  }
  else #else以下は省略可能
  {
  文2;
  }
- break;で反復より脱出

8

乱数と関連する関数

- 区間(m,n)における一様分布に従う乱数はUniform(m,n)
- 正規分布N(μ,σ^2)に基づく乱数はNormal(μ,σ^2)
- 区間[0,2^{24})における整数値をとる一様乱数はIrand224( )
- 例：(0,1)における一様乱数を10個生成

```
option randseed 0; #乱数の初期値を更新
#0以外の正整数の場合は同じ系列の乱数
for{i in 1..10}
{
display Uniform(0,1); #Uniform(0,1)は0以上1以下の乱数
}
```

- ガウス記号[x]（x以下の最大整数） floor(x)
- x以上の最小整数を表す関数 ceil(x)

9

受講に際して注意する点

- 手法の理解⇒講義時間は限られている
- 演習時間が足りない⇒演習時間も限られている
- 大学設置基準⇒1コマの授業につき，1.5コマ分の予習，1.5コマ分の復習を「必要とする」
- オペレーションズ・リサーチの研究室⇒OR-Aなどの講義資料を見直すこと（他にも、所属研究室の研究に関連する講義を受講すること）

10

最適化・シミュレーション演習

第2回 AMPLの定式化

- 授業サポートページ
  http://www.shiina.mgmt.waseda.ac.jp/optsim/
- 数理計画による最適化と(離散事象型)シミュレーションに関する演習を行う．使用するソフトウェアは，AMPL(＋GurobiまたはCPLEX)，および，Simul8を想定している．
- 演習では，数理計画による最適化やシミュレーションの実践的能力を身につけることを目指す。履修者は，Cを使用できる環境を有するPCを持参すること。受講者は，実験室にて，演習で使用する C，AMPL，Simul8をダウンロードできる。

- **典型的な数理計画問題の定式化と演習**
  - **輸送問題**
  - **施設配置問題**
  - **集合被覆／分割問題**
- **輸送問題**：工場m=2か所から倉庫n= 3か所への輸送問題を考える。工場iから倉庫jへの単位輸送量あたりの輸送コストc_{ij} (i=1,...,2 , j=1,...,3), 工場iからの供給量S_i (i=1,...,2) , 倉庫jの需要量D_j (j=1,...,3)は表のように与えられている。
- すべての需要を満たし、総費用(輸送費)を最小にする輸送量を求めたい。この問題を数理計画問題として定式化せよ。

倉庫 / 工場	倉庫1	倉庫2	倉庫3	供給量
工場1	6	5	4	35
工場2	8	9	5	15
需要量	25	15	10	

輸送問題の定式化

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\text{subject to} \sum_{i=1}^{m} x_{ij} = D_j,\ j = 1,...,n$$

$$\sum_{j=1}^{n} x_{ij} = S_i,\ i = 1,...,m$$

$$x_{ij} \geq 0,\ i = 1,...,m,\ j = 1,...,n$$

- 変数 x_{ij} : 工場 iから倉庫 jへの輸送量
- 目的関数:総費用
- 各倉庫での需要制約
- 各工場での供給制約
- 変数の非負制約
- これをAMPLを用いて定式化する。

AMPLによる輸送問題の定式化

```
#tp.mod

param NPLANT ;
param NDEMAND ;
param c {i in 1..NPLANT, j in 1..NDEMAND}>=0;
param S {i in 1..NPLANT}>=0;
param D {j in 1..NDEMAND}>=0;

var x {i in 1..NPLANT, j in 1..NDEMAND} >=0;

minimize totalcost:
 sum {i in 1..NPLANT, j in 1..NDEMAND} c[i,j]*x[i,j];

subject to supply {i in 1..NPLANT}:
 sum {j in 1..NDEMAND}x[i,j] = S[i];

subject to demand {j in 1..NDEMAND}:
 sum {i in 1..NPLANT}x[i,j]  = D[j];
```

- パラメータ定義
- パラメータ名{添字 in 集合}
  - 工場数
  - 倉庫数
  - 輸送費用
  - 供給量
  - 需要量
- 決定変数定義
- 決定変数名{添字 in 集合}
  - 決定変数 x
- 目的関数定義
- 目的関数名:数式
  - 最小化 totalcost: 目的関数記述
- 制約条件定義
- 制約条件名:数式
  - supply-const: 制約条件記述
  - demand-const: 制約条件記述

輸送問題：データ、コマンドスクリプト

```
#tp.dat
param NPLANT := 2;
param NDEMAND :=3;
param varcost :=
[1,*] 1 6 2 5 3 4
[2,*] 1 8 2 9 3 5 ;
param S :=
1 35 2 15 ;
param D :=
1 25 2 15 3 10;
```

- データの記述はかなり融通効く

```
#tp.run;
model tp.mod ;
data tp.dat ;
option solver cplexamp;
solve;
option display_round 6;
expand > to.sol;
display totalcost > tp.sol;
display supply.dual > tp.sol;
display demand.dual > tp.sol;
display x > tp.sol;
exit;
```

- モデルファイル指定
- データファイル指定
- ソルバ指定
- 求解
- 展開して表示
- 総費用表示
- 双対変数表示
- 双対変数表示

施設配置問題

- 関東地方を中心に営業を行ってきた輸入品販売業のある会社は、関西地方に活動を拡大するため、京阪神地方に倉庫の賃借を行う計画を立てている。
- 賃借の候補となる倉庫はmカ所にあって、第i地点の倉庫$(i=1,\cdots,m)$の月間処理能力はa_i(トン/月)で、その経費(賃借料や維持費など毎月の固定費用)はd_i(千円/月)である。
- また、関西一円に広がる消費地 j $(j=1,\cdots,n)$での輸入品の需要量b_j(トン/月)と、倉庫iから需要地jへのトン当たり輸送費c_{ij}(千円)が与えられている。
- すべての需要を満たし、<u>毎月の総費用(倉庫経費＋輸送費)</u>を最小にする倉庫配置と輸送計画を求めたい。この問題を数理計画問題として定式化せよ。

施設配置問題の定式化

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} d_i y_i$$

$$\text{subject to} \sum_{i=1}^{m} x_{ij} \geq b_j,\ j=1,\ldots,n$$

$$\sum_{j=1}^{n} x_{ij} \leq a_i y_i,\ i=1,\ldots,m$$

$$x_{ij} \geq 0,\ i=1,\ldots,m,\ j=1,\ldots,n$$

$$y_i \in \{0,1\},\ i=1,\ldots,m$$

強い定式化

$$x_{ij} \leq b_j y_i,\ i=1,\ldots,m,\ j=1,\ldots,n$$

- 変数 x_{ij}: 倉庫iから消費地 jへの輸送量
- 変数 y_i は倉庫iを賃借するかしないか
- $y_i=1$ならば、倉庫iを借り供給能力はa_iとなる。
- $y_i=0$ならば、倉庫iを借りずに供給能力は0となる。
- 実際に問題を扱う場合、このような条件のみで十分である。例えば、倉庫iを借りながらも、$\sum_{j=1}^{n} x_{ij}=0$ となる解が得られることはない。なぜなら、各倉庫の固定費$d_i>0$ であるから、供給が行われない倉庫を借りるという選択よりも、その倉庫を借りないという解の方が目的関数値が小さく、解として選択されるためである。

AMPLによる施設配置問題

```
#fl.mod;
param NPLANT ;
param NDEMAND ;

var y {i in 1..NPLANT, j in 1..NDEMAND} >=0;
var x {i in 1..NPLANT}  binary;

param varcost {i in 1..NPLANT, j in 1..NDEMAND}>=0;
param fixcost {i in 1..NPLANT}>=0;
param caplimit {i in 1..NPLANT}>=0;
param demand {j in 1..NDEMAND}>=0;

minimize cost:
sum {i in 1..NPLANT}fixcost[i]*x[i]
+sum {i in 1..NPLANT, j in 1..NDEMAND} varcost[i,j]*y[i,j];

subject to capacity {i in 1..NPLANT}:
  sum {j in 1..NDEMAND}y[i,j] <= caplimit[i]*x[i];

subject to demconst {j in 1..NDEMAND}:
 sum {i in 1..NPLANT}y[i,j]  = demand[j];
```

```
#fl.dat
param NPLANT := 2;
param NDEMAND :=3;
param varcost :=
[1,*] 1 72 2 9 3 34
[2,*] 1 85 2 78 3 79
;
param fixcost :=
1 1500 2 1400 ;
param caplimit :=
1 96 2 74 ;
param demand :=
1 10 2 28 3 21 ;
```

AMPLによる輸送問題の結果

```
#fl.run;
model fl.mod ;
data fl.dat ;
option solver cplexamp;
option display_round 6;
solve;

display cost > fl.sol;
display sum {i in 1..NPLANT}fixcost[i]*x[i] >
fl.sol;
display sum {i in 1..NPLANT, j in 1..NDEMAND}
varcost[i,j]*y[i,j] > fl.sol;

display x > fl.sol;
display y > fl.sol;
quit;];
```

```
#fl.sol
cost = 3186.000000
sum{i in 1 .. NPLANT}
fixcost[i]*x[i] = 1500.000000

sum{i in 1 .. NPLANT, j in 1 ..
NDEMAND} varcost[i,j]*y[i,j] =
1686.000000

x [*] :=
1  1.000000
2  0.000000;
y :=
1 1   10.000000
1 2   28.000000
1 3   21.000000
2 1    0.000000
2 2    0.000000
2 3    0.000000
;
```


集合被覆問題の応用

- あるコンサートをアリーナで開催することになった。そのコンサートは熱狂的なファンが殺到することで知られており、アリーナのコンサートも事故防止のために厳重な警備が必要と考えられる。
- アリーナでは、どこに警備員を配置したらよいかを検討することにした。会場担当者は、会場全体を小さなブロック（たとえば、ブロック1,2,…, m）に分け、仮に警備員をある候補地点（たとえば、候補地点1,2,…,n）に配置したときに警備員がどのブロックを同時にみることができるかを調べた。（例：m=5, n=5の場合、地点1よりブロック1,2を監視可能）
- この情報をもとに、会場全体を最小人数の警備員で警備するための配置とそのときの警備要員の必要人数を求めたい。

候補地点	1	2	3	4	5
ブロック1	○		○		
ブロック2	○	○		○	○
ブロック3		○			○
ブロック4			○	○	
ブロック5					○

集合被覆問題の定式化

- 問題データ：　配置候補地点（1,2,…,n)、警備すべきブロック(1,2,…,m)；ブロックi が候補地点 j から監視可能ならa_{ij}=1（さもなくば0）とする制約係数行列A=[a_{ij}]
- 変数（＝列）の定義：　x_j＝警備員を地点 j に配置するとき1、さもなくば0
- 目的関数（警備員を配置する地点数最小）：　最小化 $z = \sum_{j=1}^{n} c_j x_j$
- 制約条件（各ブロックは監視可能）：　制約 $\sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i = 1, \cdots, m$, $x_j \in \{0,1\}$

$$\min\ x_1 + x_2 + x_3 + x_4 + x_5$$

$$\begin{aligned}
\text{subject to}\ \ & x_1 + x_3 \geq 1 \\
& x_1 + x_2 + x_4 + x_5 \geq 1 \\
& x_2 + x_5 \geq 1 \\
& x_3 + x_4 \geq 1 \\
& x_5 \geq 1 \\
& x_1, x_2, x_3, x_4, x_5 \in \{0,1\}
\end{aligned}$$

集合被覆問題

AMPLによる集合被覆問題

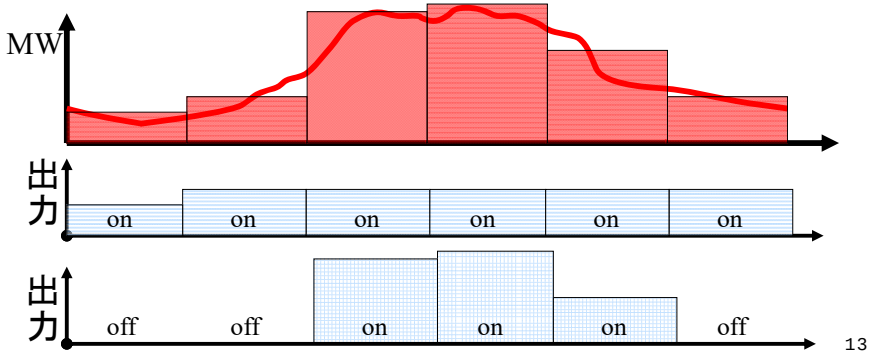
```
#sc.mod
param NBLOCK ;
param NPOINT ;
param a {i in 1..NBLOCK, j in 1..NPOINT} binary;
var x {j in 1..NPOINT} binary;

minimize totalcost:  sum {j in 1..NPOINT} x[j];

subject to watch {i in 1..NBLOCK}:
sum {j in 1..NPOINT}a[i,j]*x[j] >= 1;
```

```
#sc.dat
param NPOINT := 5;
param NBLOCK := 5;
param a :=
1 1 1  1 2 0  1 3 1  1 4 0  1 5 0
2 1 1  2 2 1  2 3 0  2 4 1  2 5 1
3 1 0  3 2 1  3 3 0  3 4 0  3 5 1
4 1 0  4 2 0  4 3 1  4 4 1  4 5 0
5 1 0  5 2 0  5 3 0  5 4 0  5 5 1;
```

```
#sc.run
model sc.mod ;
data sc.dat ;
option solver cplexamp;
option display_round 6;
solve;

expand > sc.sol;
display totalcost > sc.sol;

display x > sc.sol;
display _ampl_time >sc.sol;
display _total_solve_time
>sc.sol;
exit;
```

発電機起動停止問題

- unit commitment：電力システムにおけるスケジューリング問題
- 各時間帯に与えられた電力需要を満足するように発電機のオン/オフおよび発電量を決定



13

起動停止問題モデル

- 問題(UC): I 台 の発電機による T 時間にわたる電力供給
- 変数 u_{it} は発電機 i の時間 t における状態を表す0-1変数
- 変数 x_{it}は発電機 i の時間 t の出力
- 関数 $f_i(x_{it})$ は発電機 の燃料費を表す の凸2次関数
- 関数 $g_i(u_{i,t-1}, u_{it})$は発電機 の起動費用を表し $g_i(0,1) >0$となり、それ以外の場合には0となる関数

起動停止問題－目的関数

- 総費用の最小化
- 燃料費 $f_i(x_{it})$:凸2次関数
- 起動費用$g_i(u_{i,t-1},u_{it})$: $g_i(0,1)>0$

$$\min \sum_{i=1}^{I}\sum_{t=1}^{T} g_i(u_{i,t-1},u_{it}) + \sum_{i=1}^{I}\sum_{t=1}^{T} f_i(x_{it})\,u_{it}$$

0-1変数u_{it} ：発電機の状態（稼動／停止）

非負変数x_{it} :発電機の出力

I,T:発電機数、計画期間数

15

起動停止問題―制約

- 需要制約

$$\sum_{i=1} x_{it} \geq d_t, \forall t$$

- 運転出力の上下限制約

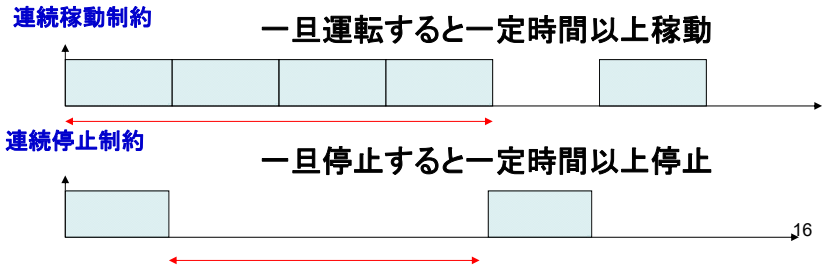
$$q_i u_{it} \leq x_{it} \leq Q_i u_{it}, u_{it} \in \{0,1\},\ \forall i, \forall t$$

- 連続稼動制約:L_i期間

$$u_{it} - u_{i,t-1} \leq u_{i\tau},\ \tau = t+1, \ldots, \min\{t+L_i-1, T\},\ \forall i, t=2,\ldots,T$$

- 連続停止制約:l_i 期間

$$u_{i,t-1} - u_{it} \leq 1 - u_{i\tau},\ \tau = t+1, \ldots, \min\{t+l_i-1, T\}\ \forall i, t=2,\ldots,T$$



16

最適化・シミュレーション演習

第3回動的計画法

- 授業サポートページ
  http://www.shiina.mgmt.waseda.ac.jp/ora/
- 数理計画による最適化と（離散事象型）シミュレーションに関する演習を行う．使用するソフトウェアは，AMPL（＋GurobiまたはCPLEX），および，Simul8を想定している．
- 演習では，数理計画による最適化やシミュレーションの実践的能力を身につけることを目指す。履修者は，Cを使用できる環境を有するPCを持参すること。受講者は，実験室にて，演習で使用する C，AMPL，Simul8をダウンロードできる。

ナップサック問題に対する動的計画法

（KP）最大化　$z=\Sigma_{j=1}^{n} c_j x_j$
制約　$\Sigma_{j=1}^{n} a_j x_j \leqq b$
$x_j \in \{0,1\}, \ \forall j$

$f_k(\lambda)=\max \{\Sigma_{j=1}^{k} c_j x_j \mid \Sigma_{j=1}^{k} a_j x_j \leqq \lambda,$
$x_j \in \{0,1\}, \ j=1,\ldots,k\}$

$f_k(\lambda)$: n個の品物のうち、$j=1,\ldots,k$ 個までの 品物を容量λのナップサックに入れる問題の最適目的関数値

最終的に $f_n(b)$を求めればよい

漸化式　$f_k(\lambda)=\max\{ f_{k-1}(\lambda), c_k+f_{k-1}(\lambda-a_k)\}$
$=\max\{$品物kを入れない利益, 品物kを入れる利益$\}$

- $f_k(\lambda)$を定義するナップサック問題の解においては、品物kが入るかまたは入らないということを考える
- 品物kが入らない場合：容量λのナップサックに$j=1,\ldots,k-1$個までの品物を入れるときの利益
- 品物kが入る場合：品物kを入れて、残り容量$\lambda-a_k$に$j=1,\ldots,k-1$個までの品物を入れるときの利益

ナップサック問題：例題

(KP)　最大化　$z=7x_1+8x_2+3x_3$
制約　$3x_1+4x_2+2x_3 \leqq 6$
$x_j \in \{0,1\}, \ \forall j$

最適解を求めよ。ただし、記号を以下のように定義する。

$f_1(\lambda)=\max\{ 7x_1 \mid 3x_1 \leqq \lambda,$
$x_j \in \{0,1\}, \ j=1\}$

$f_2(\lambda)=\max\{ 7x_1+8x_2 \mid 3x_1+4x_2 \leqq \lambda,$
$x_j \in \{0,1\}, \ j=1,2\}$

$f_3(\lambda)=\max\{ 7x_1+8x_2+3x_3 \mid 3x_1+4x_2+2x_3 \leqq \lambda,$
$x_j \in \{0,1\}, \ j=1,2,3\}$

$f_1(\lambda)$ の結果から$f_2(\lambda)$を求め、$f_2(\lambda)$の結果から$f_3(\lambda)$

$f_k(\lambda)$を計算する場合$f_{k-1}(\lambda)$は既に計算済

初期設定：$f_k(0)=0$ ($\lambda=0$のとき), $k=0,1,2,3$, $f_0(\lambda)=0$ ($\lambda>0$のとき),
f_k(負の数)$=-\infty$ ($\lambda<0$のとき) , $k=0,1,2,3$
実行不可能（容量オーバー）な場合を除外するため

ナップサック問題：解答例

(KP)　最大化　$z=7x_1+8x_2+3x_3$
制約　$3x_1+4x_2+2x_3 \leqq 6, \ x_j \in \{0,1\}, \ \forall j$
$f_k(\lambda)=\max\{ f_{k-1}(\lambda), c_k+f_{k-1}(\lambda-a_k)\}$

$k=1$のとき

$f_1(1)=\max\{$, $\}=\max\{$, $\}=$,
$f_1(2)=\max\{$, $\}=\max\{$, $\}=$,
$f_1(3)=\max\{$, $\}=\max\{$, $\}=$,
$f_1(4)=\max\{$, $\}=\max\{$, $\}=$,
$f_1(5)=\max\{$, $\}=\max\{$, $\}=$,
$f_1(6)=\max\{$, $\}=\max\{$, $\}=$,

最終的に、$f_n(b)$を求めたい
次の順に計算
$f_1(1) \Rightarrow f_1(2) \Rightarrow \cdots \Rightarrow f_1(b)$
$f_2(1) \Rightarrow f_2(2) \Rightarrow \cdots \Rightarrow f_2(b)$
・・・
$f_n(1) \Rightarrow f_n(2) \Rightarrow \cdots \Rightarrow f_n(b)$

$k=2$のとき

$f_2(1)=\max\{$, $\}=$,
$f_2(2)=\max\{$, $\}=$,
$f_2(3)=\max\{$, $\}=$,
$f_2(4)=\max\{$, $\}=$,
$f_2(5)=\max\{$, $\}=$,
$f_2(6)=\max\{$, $\}=$,

$k=3$のとき

$f_3(1)=\max\{$, $\}=$,
$f_3(2)=\max\{$, $\}=$,
$f_3(3)=\max\{$, $\}=$,
$f_3(4)=\max\{$, $\}=$,
$f_3(5)=\max\{$, $\}=$,
$f_3(6)=\max\{$, $\}=$,

ナップサック問題：解答例

(KP) 最大化 $z = 7x_1 + 8x_2 + 3x_3$

制約 $3x_1 + 4x_2 + 2x_3 \leqq 6,\ x_j \in \{0,1\},\ \forall j$

$f_k(\lambda) = \max\{ f_{k-1}(\lambda),\ c_k + f_{k-1}(\lambda - a_k) \}$

k=1のとき

$f_1(1) = \max\{\mathbf{f_0(1)},\ 7 + f_0(1\text{-}3)\} = \max\{0, -\infty\} = 0,$
$f_1(2) = \max\{\mathbf{f_0(2)},\ 7 + f_0(2\text{-}3)\} = \max\{0, -\infty\} = 0,$
$f_1(3) = \max\{f_0(3),\ \mathbf{7 + f_0(3\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(4) = \max\{f_0(4),\ \mathbf{7 + f_0(4\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(5) = \max\{f_0(5),\ \mathbf{7 + f_0(5\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(6) = \max\{f_0(6),\ \mathbf{7 + f_0(6\text{-}3)}\} = \max\{0, 7\} = 7,$

最終的に、$f_n(b)$を求めたい
次の順に計算
$f_1(1) \Rightarrow f_1(2) \Rightarrow \cdots \Rightarrow f_1(b)$
$f_2(1) \Rightarrow f_2(2) \Rightarrow \cdots \Rightarrow f_2(b)$
・・・
$f_n(1) \Rightarrow f_n(2) \Rightarrow \cdots \Rightarrow f_n(b)$

k=2のとき

$f_2(1) = \max\{\quad,\quad\} = \quad,$
$f_2(2) = \max\{\quad,\quad\} = \quad,$
$f_2(3) = \max\{\quad,\quad\} = \quad,$
$f_2(4) = \max\{\quad,\quad\} = \quad,$
$f_2(5) = \max\{\quad,\quad\} = \quad,$
$f_2(6) = \max\{\quad,\quad\} = \quad,$

k=3のとき

$f_3(1) = \max\{\quad,\quad\} = \quad,$
$f_3(2) = \max\{\quad,\quad\} = \quad,$
$f_3(3) = \max\{\quad,\quad\} = \quad,$
$f_3(4) = \max\{\quad,\quad\} = \quad,$
$f_3(5) = \max\{\quad,\quad\} = \quad,$
$f_3(6) = \max\{\quad,\quad\} = \quad,$

5

ナップサック問題：解答例

(KP) 最大化 $z = 7x_1 + 8x_2 + 3x_3$

制約 $3x_1 + 4x_2 + 2x_3 \leqq 6,\ x_j \in \{0,1\},\ \forall j$

$f_k(\lambda) = \max\{ f_{k-1}(\lambda),\ c_k + f_{k-1}(\lambda - a_k) \}$

k=1のとき

$f_1(1) = \max\{\mathbf{f_0(1)},\ 7 + f_0(1\text{-}3)\} = \max\{0, -\infty\} = 0,$
$f_1(2) = \max\{\mathbf{f_0(2)},\ 7 + f_0(2\text{-}3)\} = \max\{0, -\infty\} = 0,$
$f_1(3) = \max\{f_0(3),\ \mathbf{7 + f_0(3\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(4) = \max\{f_0(4),\ \mathbf{7 + f_0(4\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(5) = \max\{f_0(5),\ \mathbf{7 + f_0(5\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(6) = \max\{f_0(6),\ \mathbf{7 + f_0(6\text{-}3)}\} = \max\{0, 7\} = 7,$

最終的に、$f_n(b)$を求めたい
次の順に計算
$f_1(1) \Rightarrow f_1(2) \Rightarrow \cdots \Rightarrow f_1(b)$
$f_2(1) \Rightarrow f_2(2) \Rightarrow \cdots \Rightarrow f_2(b)$
・・・
$f_n(1) \Rightarrow f_n(2) \Rightarrow \cdots \Rightarrow f_n(b)$

k=2のとき

$f_2(1) = \max\{\mathbf{f_1(1)},\ 8 + f_1(1\text{-}4)\} = 0,$
$f_2(2) = \max\{\mathbf{f_1(2)},\ 8 + f_1(2\text{-}4)\} = 0,$
$f_2(3) = \max\{\mathbf{f_1(3)},\ 8 + f_1(3\text{-}4)\} = 7,$
$f_2(4) = \max\{f_1(4),\ \mathbf{8 + f_1(4\text{-}4)}\} = 8,$
$f_2(5) = \max\{f_1(5),\ \mathbf{8 + f_1(5\text{-}4)}\} = 8,$
$f_2(6) = \max\{f_1(6),\ \mathbf{8 + f_1(6\text{-}4)}\} = 8,$

k=3のとき

$f_3(1) = \max\{\quad,\quad\} = \quad,$
$f_3(2) = \max\{\quad,\quad\} = \quad,$
$f_3(3) = \max\{\quad,\quad\} = \quad,$
$f_3(4) = \max\{\quad,\quad\} = \quad,$
$f_3(5) = \max\{\quad,\quad\} = \quad,$
$f_3(6) = \max\{\quad,\quad\} = \quad,$

6

ナップサック問題：解答例

(KP) 最大化 $z = 7x_1 + 8x_2 + 3x_3$

制約 $3x_1 + 4x_2 + 2x_3 \leqq 6,\ x_j \in \{0,1\},\ \forall j$

$f_k(\lambda) = \max\{ f_{k-1}(\lambda),\ c_k + f_{k-1}(\lambda - a_k) \}$

k=1のとき

$f_1(1) = \max\{\mathbf{f_0(1)},\ 7 + f_0(1\text{-}3)\} = \max\{0, -\infty\} = 0,$
$f_1(2) = \max\{\mathbf{f_0(2)},\ 7 + f_0(2\text{-}3)\} = \max\{0, -\infty\} = 0,$
$f_1(3) = \max\{f_0(3),\ \mathbf{7 + f_0(3\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(4) = \max\{f_0(4),\ \mathbf{7 + f_0(4\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(5) = \max\{f_0(5),\ \mathbf{7 + f_0(5\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(6) = \max\{f_0(6),\ \mathbf{7 + f_0(6\text{-}3)}\} = \max\{0, 7\} = 7,$

最終的に、$f_n(b)$を求めたい
次の順に計算
$f_1(1) \Rightarrow f_1(2) \Rightarrow \cdots \Rightarrow f_1(b)$
$f_2(1) \Rightarrow f_2(2) \Rightarrow \cdots \Rightarrow f_2(b)$
・・・
$f_n(1) \Rightarrow f_n(2) \Rightarrow \cdots \Rightarrow f_n(b)$

k=2のとき

$f_2(1) = \max\{\mathbf{f_1(1)},\ 8 + f_1(1\text{-}4)\} = 0,$
$f_2(2) = \max\{\mathbf{f_1(2)},\ 8 + f_1(2\text{-}4)\} = 0,$
$f_2(3) = \max\{\mathbf{f_1(3)},\ 8 + f_1(3\text{-}4)\} = 7,$
$f_2(4) = \max\{f_1(4),\ \mathbf{8 + f_1(4\text{-}4)}\} = 8,$
$f_2(5) = \max\{f_1(5),\ \mathbf{8 + f_1(5\text{-}4)}\} = 8,$
$f_2(6) = \max\{f_1(6),\ \mathbf{8 + f_1(6\text{-}4)}\} = 8,$

k=3のとき

$f_3(1) = \max\{\mathbf{f_2(1)},\ 3 + f_2(1\text{-}2)\} = 0,$
$f_3(2) = \max\{f_2(2),\ \mathbf{3 + f_2(2\text{-}2)}\} = 3,$
$f_3(3) = \max\{\mathbf{f_2(3)},\ 3 + f_2(3\text{-}2)\} = 7,$
$f_3(4) = \max\{\mathbf{f_2(4)},\ 3 + f_2(4\text{-}2)\} = 8,$
$f_3(5) = \max\{f_2(5),\ \mathbf{3 + f_2(5\text{-}2)}\} = 10,$
$f_3(6) = \max\{f_2(6),\ \mathbf{3 + f_2(6\text{-}2)}\} = 11,$

7

ナップサック問題：解答例

(KP) 最大化 $z = 7x_1 + 8x_2 + 3x_3$

制約 $3x_1 + 4x_2 + 2x_3 \leqq 6,\ x_j \in \{0,1\},\ \forall j$

$f_k(\lambda) = \max\{ f_{k-1}(\lambda),\ c_k + f_{k-1}(\lambda - a_k) \}$

k=1のとき

$f_1(1) = \max\{\mathbf{f_0(1)},\ 7 + f_0(1\text{-}3)\} = \max\{0, -\infty\} = 0,$
$f_1(2) = \max\{\mathbf{f_0(2)},\ 7 + f_0(2\text{-}3)\} = \max\{0, -\infty\} = 0,$
$f_1(3) = \max\{f_0(3),\ \mathbf{7 + f_0(3\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(4) = \max\{f_0(4),\ \mathbf{7 + f_0(4\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(5) = \max\{f_0(5),\ \mathbf{7 + f_0(5\text{-}3)}\} = \max\{0, 7\} = 7,$
$f_1(6) = \max\{f_0(6),\ \mathbf{7 + f_0(6\text{-}3)}\} = \max\{0, 7\} = 7,$

最終的に、$f_n(b)$を求めたい
次の順に計算
$f_1(1) \Rightarrow f_1(2) \Rightarrow \cdots \Rightarrow f_1(b)$
$f_2(1) \Rightarrow f_2(2) \Rightarrow \cdots \Rightarrow f_2(b)$
・・・
$f_n(1) \Rightarrow f_n(2) \Rightarrow \cdots \Rightarrow f_n(b)$

k=2のとき

$f_2(1) = \max\{\mathbf{f_1(1)},\ 8 + f_1(1\text{-}4)\} = 0,$
$f_2(2) = \max\{\mathbf{f_1(2)},\ 8 + f_1(2\text{-}4)\} = 0,$
$f_2(3) = \max\{\mathbf{f_1(3)},\ 8 + f_1(3\text{-}4)\} = 7,$
$f_2(4) = \max\{f_1(4),\ \mathbf{8 + f_1(4\text{-}4)}\} = 8,$
$f_2(5) = \max\{f_1(5),\ \mathbf{8 + f_1(5\text{-}4)}\} = 8,$
$f_2(6) = \max\{f_1(6),\ \mathbf{8 + f_1(6\text{-}4)}\} = 8,$

k=3のとき

$f_3(1) = \max\{\mathbf{f_2(1)},\ 3 + f_2(1\text{-}2)\} = 0,$
$f_3(2) = \max\{f_2(2),\ \mathbf{3 + f_2(2\text{-}2)}\} = 3,$
$f_3(3) = \max\{\mathbf{f_2(3)},\ 3 + f_2(3\text{-}2)\} = 7,$
$f_3(4) = \max\{\mathbf{f_2(4)},\ 3 + f_2(4\text{-}2)\} = 8,$
$f_3(5) = \max\{f_2(5),\ \mathbf{3 + f_2(5\text{-}2)}\} = 10,$
$f_3(6) = \max\{f_2(6),\ \mathbf{3 + f_2(6\text{-}2)}\} = 11,$

8

k λ	1	2	3	4	5	6
$f_1(\lambda)$						
x_1						
$f_2(\lambda)$						
x_2						
$f_3(\lambda)$						
x_3						

k λ	1	2	3	4	5	6
$f_1(\lambda)$	0	0	7	7	7	7
x_1	0	0	1	1	1	1
$f_2(\lambda)$	0	0	7	8	8	8
x_2	0	0	0	1	1	1
$f_3(\lambda)$	0	3	7	8	10	11
x_3	0	1	0	0	1	1

- $f_3(6)=\max\{f_2(6), \mathbf{3+}\ \boldsymbol{f_2(6\text{-}2)}\}=11$, より逆算 $x_3=1$
- $f_2(4)=\max\{f_1(4), \mathbf{8+}\ \boldsymbol{f_1(4\text{-}4)}\}=8$, $x_2=1$
- $f_1(0)=0$, $x_1=0$

AMPLによるナップサック問題

```
#knapsack.mod

param Nitem;
param weight{i in 1..Nitem};
param value{i in 1..Nitem};
param l{i in 1..Nitem};
param u{i in 1..Nitem};
param capacity;

var x{i in 1..Nitem} binary;

maximize Profit: sum{i in 1..Nitem}value[i]*x[i];

subject to Capacity_Constraint:
sum{i in 1..Nitem}weight[i]*x[i] <= capacity;
subject to Upper_Lower_Bound {i in 1..Nitem}:
l[i] <= x[i] <=u[i];
```

```
#knapsack.dat
param Nitem:=3;
param:value weight:=
1 7 3
2 8 4
3 3 2;
param capacity:=6;
param l :=
1 0  2 0  3 0;
param u :=
1 1  2 1  3 1;

#knapsack.run
model knapsack.mod ;
data knapsack.dat ;
option display_round 6;
option solver cplexamp;
solve;
display Profit > knapsack.sol;
display x > knapsack.sol;
```


AMPLによるナップサック問題:DP

```
#knapsack-dp.run
param Nitem;
param value{i in 1..Nitem};
param weight{i in 1..Nitem};
param capacity;
param l{i in 1..Nitem};
param u{i in 1..Nitem};
param f{k in 0..Nitem, la in -capacity..capacity};

data knapsack.dat;
option display_transpose -20; #行列表示
for{k in 0..Nitem, la in -capacity..-1} #laはλを表す
{
        let f[k,la]:=-1000;   #初期設定
}
for{la in 0..capacity}
{
        let f[0,la]:=0;       #初期設定
}
for{k in 1..Nitem}
{
        let f[k,0]:=0;        #初期設定
}
```

```
#左から続く
for{k in 1..Nitem}
{
    for{la in 1..capacity}
    {
        if(f[k-1,la] >=
         value[k]+f[k-1,la-weight[k]])then
        {
            let f[k,la]:=f[k-1,la];
        }
        if(f[k-1,la] <
          value[k]+f[k-1,la-weight[k]])then
          {
    let f[k,la]:=value[k]+f[k-1,la-weight[k]];
          }
    }
}
display f > knapsack-dp.sol;
```

動的ロットサイズ決定問題
Wagner-Whitin（WW)モデル

期(月)	1	2	3
需要 d_t	10	10	10
固定費 a_t	100	120	100
製造変動費 v_t	20	30	10
在庫費 h_t	8	2	(3)

- 計画期間T=3，各期の需要d_t
- t 期に生産する場合の製造固定費は生産量に関係なく a_t (万円)
- t期に生産する場合の製造変動費は常に製品あたり v_t (万円)
- t期末の在庫に対する、製品1個当たりの在庫保管費h_t（万円）
- 生産は瞬時； 品切れは不許可
- 総費用を最小にする生産計画を決めたい

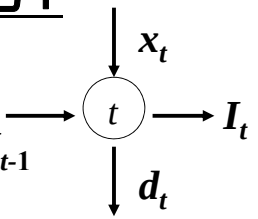
WWモデルの定式化1

最小化 $z=\Sigma_{t=1,\ldots,T}(p_t(x_t)+h_tI_t)$

制約 $I_{t-1}+x_t-d_t=I_t$，$\forall t$（流量保存） I_{t-1}

$I_t \geqq 0$（品切れ不許可），$x_t \geqq 0$，$\forall t$

$I_0=0$，$I_T=0$ 変数は x, I

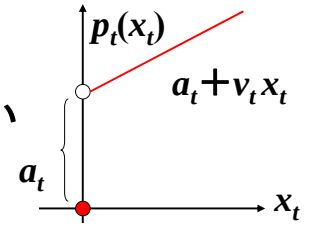


製造費を表す関数$p_t(x_t)$は以下のようになる

$p_t(x_t)=a_t+v_tx_t$ $x_t>0$のとき

$=0$ $x_t=0$のとき

このような関数を取り扱うのは難しい



WWモデルの定式化2

最小化 $z=\Sigma_{t=1,\ldots,T}(a_ty_t+v_tx_t+h_tI_t)$

制約 $I_{t-1}+x_t-d_t=I_t$， $\forall t$ （流量保存）

（$x_t>0 \Rightarrow y_t=1$, y_t: t 期に製造を行うとき1,

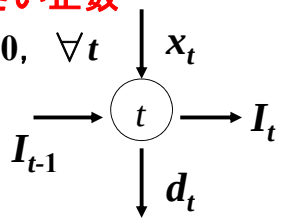
製造を行わないとき0

ここで$y_t=0 \Rightarrow x_t=0$を表すようにするには)

$x_t \leqq My_t$ ただしMは十分大きい正数

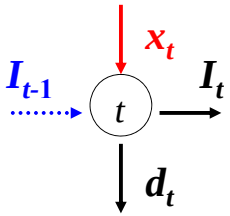
$I_t \geqq 0$（品切れ不許可），$x_t \geqq 0$，$\forall t$

$I_0=0$，$I_T=0$

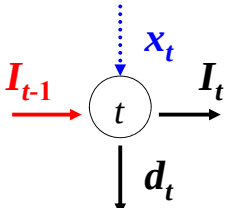


WWモデルの最適解の性質(1)

- WWモデルの最適解は、$I_{t-1}x_t=0$ という性質を満足する．T－1期末在庫I_{t-1}、または、t期生産量 x_tのいずれか一方が0
- $I_{t-1}x_t=0$: 2通りの場合がある
- 第t-1期末に $I_{t-1}=0$かつ$x_t>0$ (在庫がない場合製造する)
- 第t-1期末に $I_{t-1}>0$かつ$x_t=0$ (在庫がある場合製造しない)
- 在庫があり同時に生産するとコスト的に無駄である



$I_{t-1}=0$かつ$x_t>0$



$I_{t-1}>0$かつ$x_t=0$

WWモデルの最適解の性質(2)
Zero－Inventory Property

- 性質１　t期に生産する（$x_t > 0$）ということは、$t-1$期末在庫が0（$I_{t-1}=0$）である.
- 性質２　t期に生産する場合、t期の生産量x_tはt期に始まり、向こう何期分かの需要に見合う量である.
  - t期の生産量x_tはd_t, d_t+d_{t+1}, $d_t+d_{t+1}+d_{t+2}$, . . . , $d_t+d_{t+1}+d_{t+2}+\cdots+d_T$のいずれか（ただし、$T$は計画期間の長さ）

動的計画法（DP）による解法

- t期の生産量x_tはd_t, d_t+d_{t+1}, $d_t+d_{t+1}+d_{t+2}$, . . . , $d_t+d_{t+1}+d_{t+2}+\cdots+d_T$のいずれか
- $c_{t,k-1}$＝t期からk-1期の需要に見合う量をt期の生産量$x_t = d_t+d_{t+1}+d_{t+2}+\cdots+d_{k-1}$としたときの$t$期から$k$-1期の総費用（$c_{tk}$は計算可能）
- f_t＝（$t-1$期末在庫I_{t-1}が0のときに）t期以降最終T期末までを最適に決定したときのt期以降の最小費用（f_1を求めたい）
- $f_t = \min_{k=t+1,\ldots,T+1}\{c_{t,k-1} + f_k\}$　（DPの漸化式 $f_{T+1}=0$）

$$= \min_{k=t+1,\ldots,T+1}\{(c_{t,t}+f_{t+1}),\ (c_{t,t+1}+f_{t+2}),\ (c_{t,t+2}+f_{t+3}),\ \cdots\ (c_{t,T-1}+f_T),\ (c_{t,T}+f_{T+1}),\}$$

- f_tを求めるとき、$f_{t+1}, f_{t+2}, \cdots, f_T, f_{T+1}$は既に計算済
- $f_{T+1}=0$として$f_T, \cdots, f_{t+2}, f_{t+1}$の順に計算する

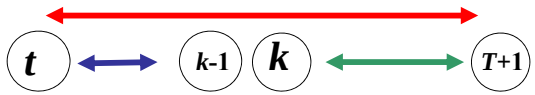
動的計画法（DP）の漸化式

- $c_{t,k-1}$＝t期からk-1期の需要に見合う量をt期の生産量x_tとしたときのt期からk-1期の総費用

$$c_{t,k-1} = a_t + v_t(d_t + d_{t+1} + \cdots + d_{k-1}) + h_t(d_{t+1} + \cdots + d_{k-1}) + h_{t+1}(d_{t+2} + \cdots + d_{k-1}) + \cdots + h_{k-2}(d_{k-1})$$

$$= a_t + v_t \sum_{j=t}^{k-1} d_t + \sum_{i=t}^{k-2} h_i \sum_{j=i+1}^{k-1} d_j$$

- $f_t = \min_{k=t+1,\ldots,T+1}\{c_{t,k-1} + f_k\}$　（DPの漸化式）
  ただし、$f_{T+1}=0$（境界条件）

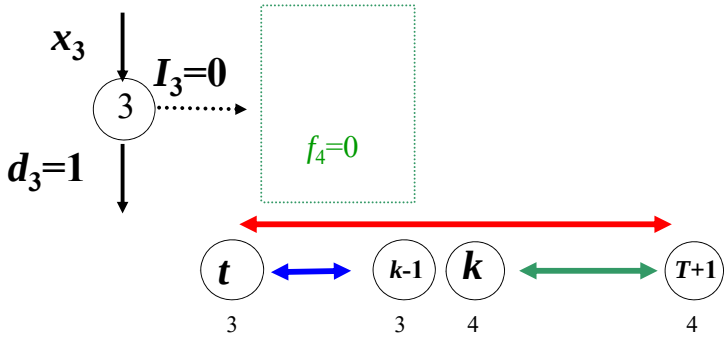


解答：$f_t = \min_{k=t+1,\ldots,T+1}\{c_{t,k-1} + f_k\}$
ただし、$f_{T+1}=0$（境界条件）

- $f_{T+1}=0$ より、 $f_4=0$と定める。
- $f_3 = 100+10\times10+f_4 = 200^*$ (3月に3月分生産)
- $f_2 = \min\{$2月分のみ生産, 2－3月分同時生産 $\}$
  $= \min\{120+30\times10+f_3,\ 120+30\times20+2\times10+f_4\}$
  $= \min\{620^*, 740\} = 620$　（2月分のみ生産）
- $f_1 = \min\{$1月分のみ生産, 1-2月分同時生産, 1-3月分同時生産$\}$
  $= \min\{100+20\times10+f_2,\ 100+20\times20+8\times10+f_3,\ 100+20\times30+8\times20+2\times10+f_4\}$
  $= \min\{920, 780^*, 880\} = 780$　（1月に1－2月分生産）
- これより、総費用は780, 最適解は$x_1=20$, $I_1=10$, $x_3=10$

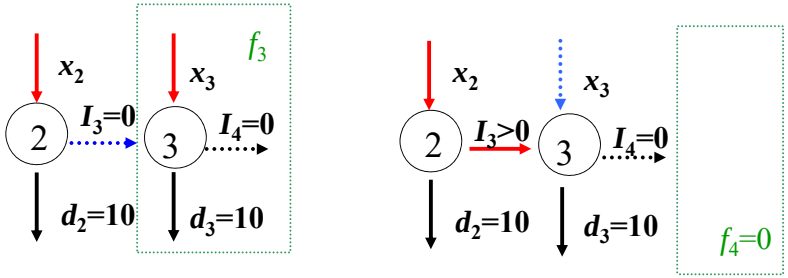
解答1：$f_t = \min_{k=t+1,\ldots,T+1}\{c_{t,k-1} + f_k\}$
ただし、$f_{T+1} = 0$（境界条件）

- T=3, f_{T+1}=0 より、f_4=0
- t=3 の場合（3期から後を考える）k=4のみを計算
  $c_{3,3}$=100+10×10=200, f_4=0
- f_4=min{200+0}=200* 3期に第3期分生産



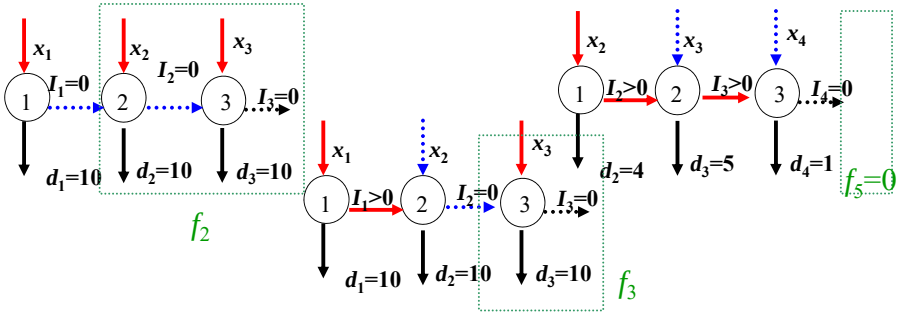
解答2：$f_t = \min_{k=t+1,\ldots,T+1}\{c_{t,k-1} + f_k\}$

- t=2の場合（2期から後を考える）k=3, 4 について計算
- $c_{2,2}$=120+30×10=420, f_3=200, 2期に第2期分のみ生産
- $c_{2,3}$=120+30×20+2×10=740, f_4=0, 2期に第2-3期分を生産
- f_2=min {620*, 740}＝620 * 2期に第2期分生産



解答3：$f_t = \min_{k=t+1,\ldots,T+1}\{c_{t,k-1} + f_k\}$

- t=1の場合（1期から後を考える）k=2, 3, 4 について計算
- $c_{1,1}$=100+20×10=300, f_2=620, 1期に第1期分のみ生産
- $c_{1,2}$=100+20×20+8×10=580, f_3=200, 1期に第1-2期分を生産
- $c_{1,3}$=100+20×30 +8×20 +2×10=880, f_4=0, 1期に第1-3期分を生産
- f_1=min{920, 780*, 880}＝780* 2期に第2-4期分生産



最適化・シミュレーション演習

第4回 分枝限定法

- 授業サポートページ
  http://www.shiina.mgmt.waseda.ac.jp/optsim/
- 数理計画による最適化と(離散事象型)シミュレーションに関する演習を行う．使用するソフトウェアは，AMPL(＋GurobiまたはCPLEX)，および，Simul8を想定している．
- 演習では，数理計画による最適化やシミュレーションの実践的能力を身につけることを目指す。履修者は，Cを使用できる環境を有するPCを持参すること。受講者は，実験室にて，演習で使用する C，AMPL，Simul8をダウンロードできる。

数理計画問題を解く解法(アルゴリズム)

- 「定式化」された問題は、適当な解法で解き、解を求める
- ソフトウェアを利用して解くか、自分でプログラムを作成
- 解法の種類(数理計画法)
  - 厳密解法(最適性を保証する解を求める解法)
  - 近似解法(最適性は保証しないが、良い解を早く求める)
- 列挙法:いかに「すべての」解を『列挙』するか(直接／間接)
  - 陽に(explicit)列挙／陰に(implicit)列挙
- バックトラック法(backtrack algorithm)
  - プログラムは比較的容易(列挙順序は固定)
- 分枝限定法(branch and bound method)
  - 解の列挙順序は解法の詳細に大きく依存

ナップサック問題の緩和問題
連続緩和:整数条件を緩め、$0 \leqq x_j \leqq 1$ とする

(KP-LP) $\max \ \Sigma_{j=1}^{n} c_j x_j$
s.t. $\Sigma_{j=1}^{n} a_j x_j \leqq b, 0 \leqq x_j \leqq 1, \ \forall j$

$c_j/a_j \geqq c_{j+1}/a_{j+1}, j=1,\ldots,n\text{-}1$が成り立つと仮定してよい(効率順)

例: (KP-LP) 最大化 $z = 7x_1 + 8x_2 + 3x_3$
制約 $3x_1 + 4x_2 + 2x_3 \leqq 6$
$0 \leqq x_j \leqq 1, \ \forall j$

- 2.33(万円/kg)≧2(万円/kg)≧1.5(万円/kg)
- 緩和問題の最適解は効率順に品物を詰め込み、入りきらない品物を一部分だけ入れる $x_1 =$, $x_2 =$, $x_3 =$
- 緩和問題の最適解は0-1条件を満たしていない。
- よって、$x_1 =$, $x_2 =$, $x_3 =$ へと小数値を切り捨てると(KP)の実行可能解が得られるが、最適解ではない。
- 緩和問題の最適解を、四捨五入、切上げ、切下げなどにより整数値に丸めても、対応する整数計画問題の最適解が得られるとは限らない
- 緩和問題の最適目的関数値 ≧ 元問題の最適目的関数値 .

ナップサック問題の連続緩和問題(貪欲解)
効率の良い品物から(小数解を許し)詰め込む

(KP1) 最大化 $z = 10x_1 + 6x_2 + 2x_3$
制約 $4x_1 + 3x_2 + 2x_3 \leqq 5, \quad 0 \leqq x_j \leqq 1, \ \forall j$
緩和問題の最適解: $x_1 =$, $x_2 =$, $x_3 =$
ナップサック問題の最適解: $x_1 =$, $x_2 =$, $x_3 =$
緩和問題の最適目的関数値 ≧ 元問題の最適目的関数値 .

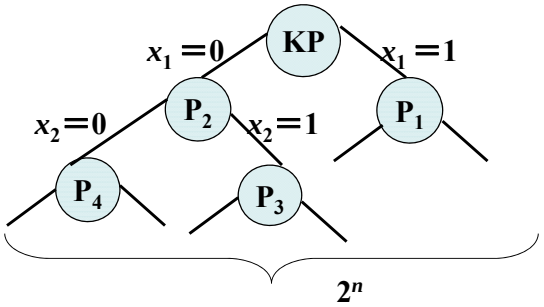
(KP2) 最大化 $z = 18x_1 + 15x_2 + 6x_3 + 2x_4$
制約 $5x_1 + 4x_2 + 2x_3 + x_4 \leqq 6, \quad 0 \leqq x_j \leqq 1, \ \forall j$
効率 ≧ ≧ ≧ なので、 の順に入れる
緩和問題の最適解: $x_1 =$, $x_2 =$, $x_3 =$, $x_4 =$
ナップサック問題の最適解: $x_1 =$, $x_2 =$, $x_3 =$, $x_4 =$
緩和問題の最適目的関数値 ≧ 元問題の最適目的関数値 .

緩和問題は、条件を緩めているので良い値が出る

分枝限定法 Branch and Bound Method

- 分枝(branching) 変数値を枝分かれしながら固定
  ===> 子問題、部分問題へ分枝
- 限定(bounding) 条件を考慮して解の範囲を限定
  ===>全列挙を行う必要がない場合もある
- 基本的用語(z^*:元問題の最適目的関数値)
- 下界値≦z^*≦上界値
- 最大化問題では、下界値は実行可能解から、上界値は緩和問題から得られる
  - 暫定(目的関数)値: これまでに見つかっている最良の解の目的関数値
  - 暫定解: これまでに見つかっている最良解
  - 分枝変数: 分枝の対象となる変数
  - 分枝頂点: 分枝の対象となる頂点(子問題)
  - 未分枝頂点: まだ探索が終わっていない頂点(子問題)

分枝操作(branching operation)



- ある問題(「親問題」)の実行可能領域をいくつかに分割して, 複数の「子問題(部分問題)」を生成する操作.
- 分枝変数: 分割に用いた変数
- 列挙木
- 子問題＝列挙木の頂点

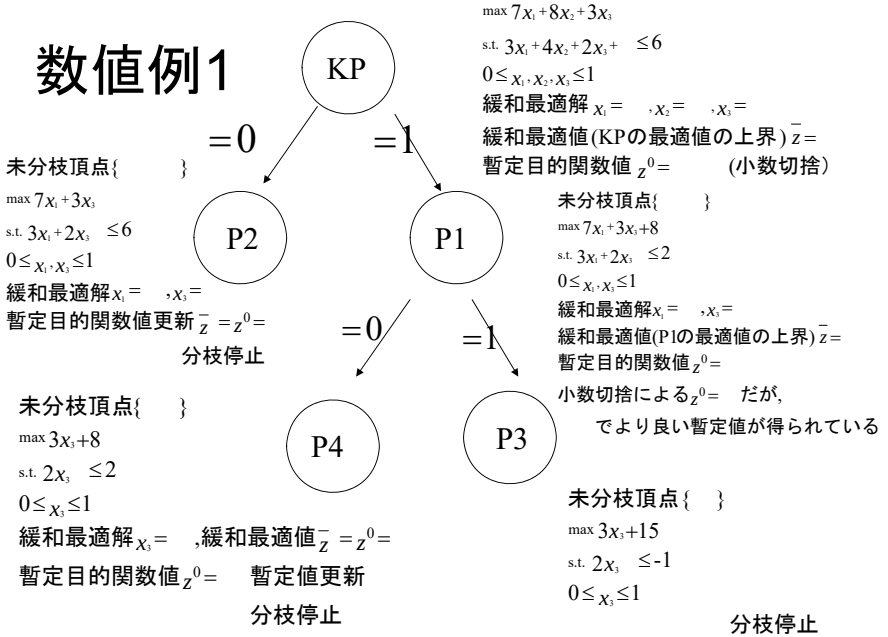
限定操作(bounding operation)　<最大化の場合>

- ある子問題に分枝操作を施す必要があるか否かを判定
- 具体的には, 緩和問題から得られる上界値が, 手元にある暫定値より小さい場合、分枝を停止
- 緩和してもこれまでに得られた解より悪い値しか得られない

0-1ナップサック問題に対する分枝限定法の設計例

1)上界値の算出方法
　線形緩和(LP)
2)下界値(実行可能解)の算出方法(KPは容易)
　LP最適解の切り下げ
3)分枝変数の選択(どの変数で枝分かれするか？)
　LP最適解で小数値をとる変数=0 問題番号を大きく(左子問題)
　LP最適解で小数値をとる変数=1 問題番号を小さく(右子問題)
4)分枝頂点(分枝子問題)の選択(どの頂点から分枝するか？)
　深さ優先規則(未分枝頂点リストから最も番号の大きいもの)
5)分枝が停止する場合
　0-1条件を満たす
　実行不可能な場合
　限定操作

数値例1



分枝限定法の実装

- 分枝限定法(BB)は、混合整数計画問題に対しては、一般的なソルバーでも使用可能
- 分枝限定法のカスタマイズ
  - 分枝切除法(分枝限定法＋切除平面法)
  - 分枝価格法(分枝限定法＋列生成法)
- 他にも、特定の細かい分枝ルールを制御したい、あるいはLPベースでない場合など
- これまではLPに基づくBBですら、実装は困難
  - 各子問題のデータ、未分枝頂点の管理が難しい
- AMPL(second edition以降)によると、反復計算(データ置換(定数、係数)、制約式、変数追加など)容易に

ナップサック問題(緩和問題)のモデルファイル

- 次のような 3 つのファイルを作業フォルダAMPLWORK (C:¥amplcml¥amplwork)に作成する。
- knapsack-relax.mod ファイル:ナップサック問題の連続緩和問題を記述したファイル
- knapsack-relax.dat ファイル:ナップサック問題のデータを記述したファイル
- knapsack-relax.run ファイル: AMPL のコマンドを記述したファイル
- ファイル knapsack-relax.mod の内容 (＃以降はコメント)

```
param Nitem;                    #品物数をNitem と定義
param weight{i in 1..Nitem};    #各品物に重さ(weight)を定義(各定数、変数の次元はNitem でよい)
param value{i in 1..Nitem};     #各品物に価値(value)を定義
param l{i in 1..Nitem};         #0-1 制約の緩和制約における下限
param u{i in 1..Nitem};         #0-1 制約の緩和制約における上限
param capacity;                 #ナップサックの容量
var x{i in 1..Nitem} >=0 ;      #各品物(入れる/入れない)に対応する変数、binary にすればMIP
maximize Profit: sum{i in 1..Nitem}value[i]*x[i];                                  #目的関数Profit
subject to Capacity_Constraint: sum{i in 1..Nitem}weight[i]*x[i] <= capacity;  #ナップサックの容量制約
subject to Upper_Lower_Bound {i in 1..Nitem}: l[i] <= x[i] <=u[i];              #0-1 制約を緩和した上下限
```

ナップサック問題(緩和問題)のコマンドスクリプト

- knapsack-relax.run ファイル: AMPL のコマンドを記述したファイル

```
model knapsack-relax.mod ;      #モデルファイル指定
data knapsack-relax.dat ;       #データファイル指定
option display_round 6;         #小数点以下6 桁表示
option solver cplex;            #ソルバにCPLEX を指定、これがないとMINOS が動く
solve;                          #問題の求解(貪欲解だが、ソルバで解く)
expand > knapsack-relax.sol;          #ファイル(knapsack-relax.sol)に定式化表示
display Profit > knapsack-relax.sol;  #ファイル(knapsack-relax.sol)に目的関数値
display x > knapsack-relax.sol;       #ファイル(knapsack-relax.sol)に最適解
display Capacity_Constraint.dual > knapsack-relax.sol;      #ナップサック制約の双対変数
display Upper_Lower_Bound.dual > knapsack-relax.sol;      #0-1 緩和制約の双対変数
                        #制約名.dual で双対変数
                        # >ファイル名 を省略すると、表示は画面に
```

- コマンドプロンプトにおいて、「ampl knapsack-relax.run 」と入力
- 最適解情報ファイル(knapsack-relax.sol)が作成される
- 演習問題の解を確認しよう

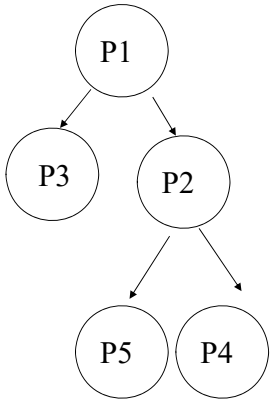
結果ファイル(knapsack-relax.sol)

```
Profit = 13.000000      #最適目的関数値

x [*] :=                #最適な x の値;
1  1.000000
2  0.750000
3  0.000000
;

#双対変数 y の値
Capacity_Constraint.dual = 2.000000

#双対変数 z の値
Upper_Lower_Bound.dual [*] :=
1  1.000000
2  0.000000
3  0.000000
;
```

未分枝頂点リスト ACTIVE_NODELIST

- 未分枝頂点を配列ACTIVE_NODELISTに保存
- 子問題を解く場合、その頂点番号が削除
- 分枝する場合、新たな頂点番号が追加
- ACTIVE_NODELIST={1}
- ACTIVE_NODELIST={2,3}
- ACTIVE_NODELIST={2}
- ACTIVE_NODELIST={5,4}
- ACTIVE_NODELIST={4}
- ACTIVE_NODELIST=φ
- 未分枝頂点リストが空集合なら終了

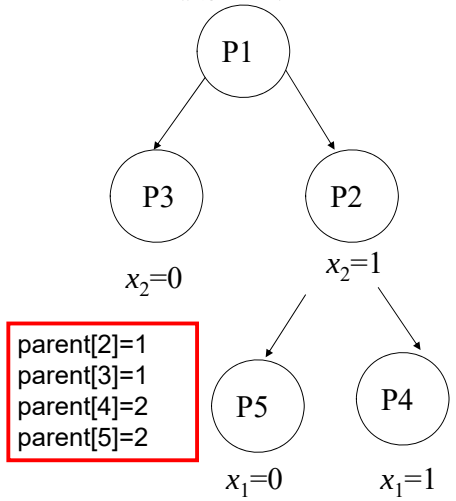
- 配列のサイズ size_activenode
- 子問題を解くと、サイズは1減少
- 分枝すると、サイズは2つ増加
- size_activenode=1
- size_activenode= 2
- size_activenode= 1
- size_activenode= 2
- size_activenode= 1
- size_activenode= 0
- 配列のサイズが0ならば終了



各頂点でのデータ保存

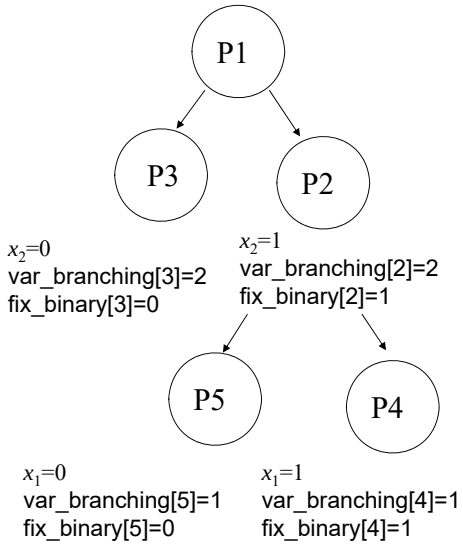
根ノードを問題1と定義
演習とは異なる

- 各子問題のデータ(定式化、knapsack-relax.datに対応するデータ)をすべての子問題について保存すると膨大な記憶領域必要
- 各子問題に加えられる0-1限定制約のみを保存
- 各頂点の親頂点を保存



子問題のデータ作成:データ構造

- 同時に解ける問題は1つ
- 子問題のデータをその都度作成
- 分枝制約を等式制約として加えるのでなく、
  l[i] <= x[i] <=u[i]
  のl[i],u[i]を書き換える
- var_branching[i]：子問題 i で変数 $x_{\text{var\_branching[i]}}$ が分枝変数
- fix_binary[i]：子問題 i で分枝変数 $x_{\text{var\_branching[i]}}$ が固定される値



子問題のデータ作成:子問題ー根への道

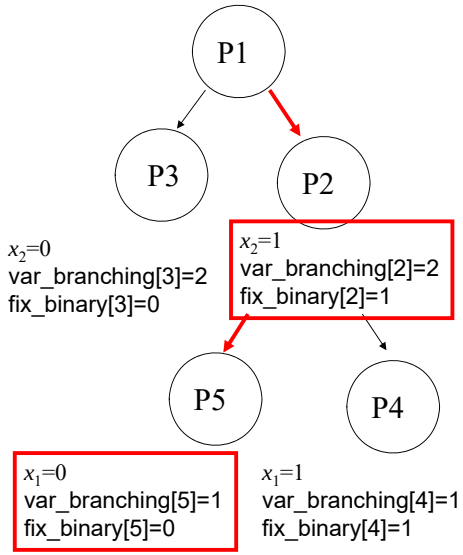
- 子問題選択は、未分枝頂点リストの末尾(奥行優先)
  ACTIVE_NODELIST[size_activenode]
- 当該子問題より、親ノードを根ノードまで探索
- 子問題5で、0=l[1]<=x_1<=u[1]=0

l[var_branching[5]]:= fix_binary[5];
u[var_branching[5]]:= fix_binary[5];

- 子問題2=parent[5]を探索
- 子問題2で、1=l[2]<=x_2<=u[2]=1

l[var_branching[2]]:= fix_binary[2];
u[var_branching[2]]:= fix_binary[2];

- 子問題2の親問題は根(頂点1)なので終了



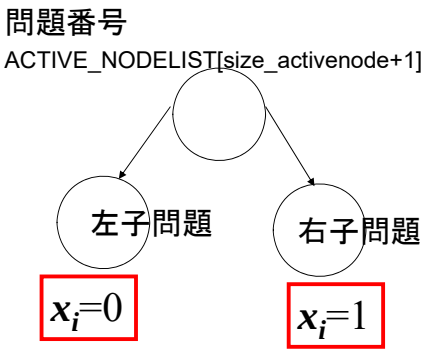
分枝するための条件

- 次の3つが同時に成り立つときのみ分枝（and条件）
- 整数解が得られない
  exists {i in 1..Nitem}
  x[i]*(1-x[i]) > epsilon
  $x_i(1-x_i)$, $i=1,...,n$ のうち、判定係数εより大きいものが存在
- 子問題が実行可能
  (solve_result="solved")
- 子問題の目的関数値が暫定値よりよい(Profit > tmpprofit)

- 左の条件より次の3つのいずれかが成り立つとき分枝停止（or条件）
- 整数解が得られた
  exists {i in 1..Nitem}
  x[i]*(1-x[i]) <= epsilon
- 子問題が実行不可能
  (solve_result="infeasible")
- 子問題の目的関数値が暫定値以下になる(これ以上分枝操作を続けても、暫定値よりよい解は得られない：限定操作) Profit <= tmpprofit

子問題への分枝

- 解く問題は常に未分枝頂点リストの末尾
- 現在までに生成された問題総番号total_node_number
- 子問題解いた時点で
  size_activenode＝
  size_activenode－1



- 左子問題
  total_node_number+2
- 分枝変数
  var_branching[total_node_number+2]=i
  fix_binary[total_node_number+2]=0

- 右子問題
  total_node_number+1
- 分枝変数
  var_branching[total_node_number+1]=i
  fix_binary[total_node_number+2]=1

演習問題

- 分枝停止となる場合、分枝停止理由をファイルに表示させる
  - 整数解が得られた
  - 実行不可能
  - 暫定値よりよい解がない（限定操作）
- 施設配置問題への適用
- （発展的課題：奥行優先以外の子問題選択規則（幅優先）は難しい…）

結果ファイル（.sol）：出力はカスタマイズできる

```
反復1
**********
Total number of subproblems 1
Current Problem 1
ACTIVE_NODELIST[1]=1
size_activenode = 1.000000

solve_result = solved

Profit = 13.000000

:    l        u        x        :=
1   0.000000   1.000000   1.000000
2   0.000000   1.000000   0.750000
3   0.000000   1.000000   0.000000
;

INTEGER CONSTRAINT VIOLATED
Subproblem 2 and 3 generated
TEMPORALLY objective value       -
      10000000000.000000
```

```
反復2
**********
Total number of subproblems 3
Current Problem 3
ACTIVE_NODELIST[1]=2
ACTIVE_NODELIST[2]=3
size_activenode = 2.000000

solve_result = solved

Profit = 10.000000

:    l        u        x        :=
1   0.000000   1.000000   1.000000
2   0.000000   0.000000   0.000000
3   0.000000   1.000000   1.000000
;

BRANCHING stopped: integer solution obtained
TEMPORALLY objective value       -10000000000.000000
TEMPORALLY objective value refined 10.000000
```

分枝停止理由をファイルに表示させる

- 実行不可能

```
if (solve_result="infeasible") then
 {
    printf: "BRANCHING stopped: infeasibility ¥n" >bb-knapsack.sol;
 }
```

- 整数解が得られた

```
if ((forall {i in 1..Nitem} x[i]*(1-x[i]) < epsilon)
   and (solve_result="solved")) then
{
  printf: "BRANCHING stopped: integer solution obtained ¥n" >bb-knapsack.sol;
 }
```

- 暫定値よりよい解がない（限定操作）

```
 if ( (Profit <= tmpprofit)
    and (solve_result="solved")) then
{
    printf: "BRANCHING stopped: bounding procedure ¥n" >bb-knapsack.sol;
}
```

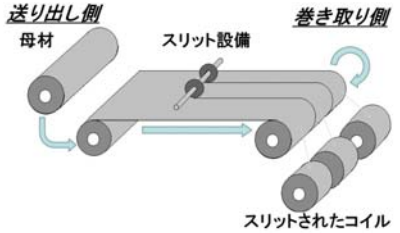
最適化・シミュレーション演習

第5回 列生成法

- 授業サポートページ
  http://www.shiina.mgmt.waseda.ac.jp/
- 数理計画による最適化と(離散事象型)シミュレーションに関する演習を行う．使用するソフトウェアは，AMPL(＋GurobiまたはCPLEX)，および，Simul8を想定している．
- 演習では，数理計画による最適化やシミュレーションの実践的能力を身につけることを目指す。履修者は，Cを使用できる環境を有するPCを持参すること。受講者は，実験室にて，演習で使用する C，AMPL，Simul8をダウンロードできる。

板取問題とは

- 製造業者は原材料である素材を、取引先が要求する大きさに合わせて多種多様にカットする必要がある。しかし、カット方法を工夫しなければ、材料に無駄が生じる。

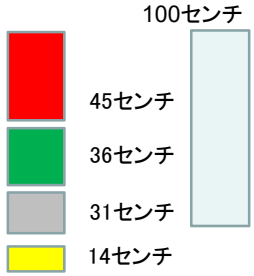


- ある長さの板から、求められる長さを取りだすときに使用する板の枚数が最小になるよう、どのような切り出しパターンで各何枚必要になるかを求めるような問題。
- Cutting stock problemとも呼ばれ、現有の素材を切り分けて所望の製品を得るための経済的な方法を求める

例題1

- 素材の幅が100センチであり、以下の需要を満たす

幅45センチの製品97個
幅36センチの製品610個
幅31センチの製品395個
幅14センチの製品211個



- 必要枚数を満たす制約
- 第j パターンに含まれる製品の個数
- 製品 i をa_{ij}個，製品iの必要枚数をb_i

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad (i = 1,2,3,4)$$

※この場合、b_1=97, b_2=610, b_3=395, b_4=211である
定式化を行うには各切り分けパターンを全て列挙する必要がある。

表1-1

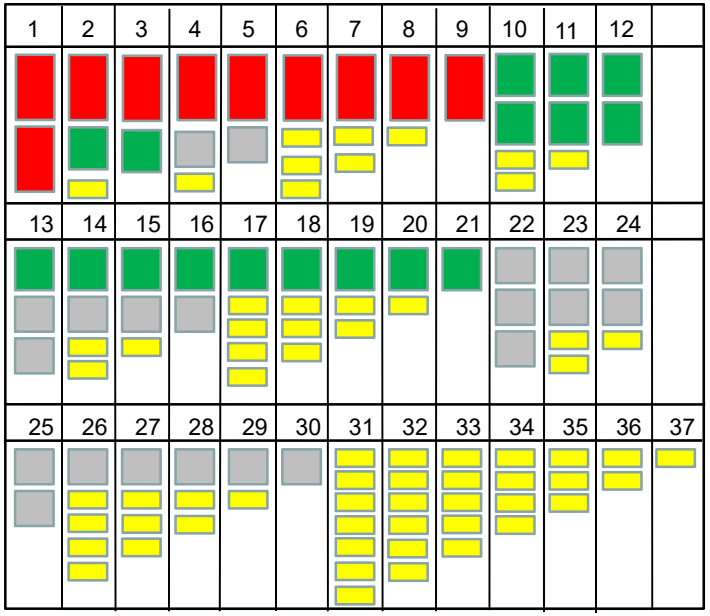
j	1	2	3	4	5	6	7	8	9	10	11	12
$a1j$	2	1	1	1	1	1	1	1	1	0	0	0
$a2j$	0	1	1	0	0	0	0	0	0	2	2	2
$a3j$	0	0	0	1	1	0	0	0	0	0	0	0
$a4j$	0	1	0	1	0	3	2	1	0	2	1	0

j	13	14	15	16	17	18	19	20	21	22	23	24
$a1j$	0	0	0	0	0	0	0	0	0	0	0	0
$a2j$	1	1	1	1	1	1	1	1	1	0	0	0
$a3j$	2	1	1	1	0	0	0	0	0	3	2	2
$a4j$	0	2	1	0	4	3	2	1	0	0	2	1

j	25	26	27	28	29	30	31	32	33	34	35	36	37
$a1j$	0	0	0	0	0	0	0	0	0	0	0	0	0
$a2j$	0	0	0	0	0	0	0	0	0	0	0	0	0
$a3j$	2	1	1	1	1	1	0	0	0	0	0	0	0
$a4j$	0	4	3	2	1	0	7	6	5	4	3	2	1

a_{1j}＝45**センチ**　a_{2j}＝36**センチ**　a_{3j}＝31**センチ**　a_{4j}＝14**センチ**

表1-2



問題の定式化

最小化: $\sum_{j=1}^{37} x_j$ （使用する素材の枚数を最小化する）

制約: $\sum_{j=1}^{37} a_{ij} x_j = b_i \quad (i = 1,2,3,4)$

$x_j \geq 0 \ (j = 1,2,\ldots,37)$ 整数

目的関数を含めて全体を定式化すると、上記のような整数計画問題になり、使用枚数を最小化する解を求める。

計算結果

- LP最適解

x_1=48.5, x_{10}=105.5, x_{12}=100.75, x_{13}=197.5

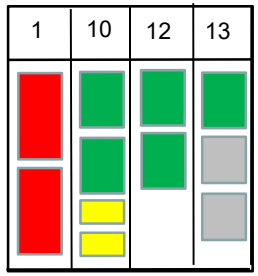
- LP 最適値 452.25
- 整数解への丸め
- 実現可能な最適解は整数でなければならないので、計算により得られた最適解の最も近い整数に切り捨てる。

48+105+100+197＝450

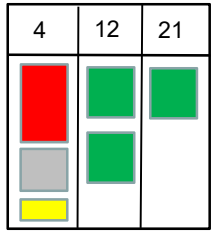
- 満たされない需要

幅45センチの製品1個
幅36センチの製品3個
幅31センチの製品1個
幅14センチの製品1個

- 追加素材3枚により、453枚の素材で注文全体を満たす事が出来る

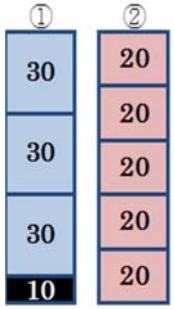


追加素材3枚



例題2

100センチの板から、30センチの製品を60枚と20センチの製品を80枚切り出したい。そのため、初めは以下のような切り出しパターンを用いて行った。



30センチを3枚切り出すパターンを1、20センチを5枚を切り出すパターンを2と定義する。

要求枚数を満たす式 $3x_1 = 60$, $2x_2 = 80$ ⇒ $x_1 = 20$, $x_2 = 16$

2つのパターンから問題の要求を満たすと、パターン1を20枚、パターン2を16枚使い、100センチの材料を36枚使うことになる。

例題2の続き

•2つの切り出しパターンを用いて、合計36枚の板を使うことにより要求を満たすことができた。
•新たに切り出しパターンを増やすことにより、板の使用枚数が最も少ない組み合わせを求める。新しいパターン3を使用することを考えたい。
パターン3では製品1をa_1枚、製品2をa_2枚を切り出す。

$$\min x_1 + x_2 + x_3$$
$$\text{s.t. } 3x_1 + a_1 x_3 = 60$$
$$5x_2 + a_2 x_3 = 80$$

$$z = x_1 + x_2 + x_3$$
$$x_1 = -\frac{a_1}{3} x_3 + 20$$
$$x_2 = -\frac{a_2}{5} x_3 + 16$$

例題2：単体法による解法

基底変数	定数項	z	x_1	x_2	x_3
z	36	1	0	0	$(a_1/3+a_2/5-1)$
x_1	20	0	1	0	$a_1/3$
x_2	16	0	0	1	$a_2/5$

- 新しいパターン3が選択され、目的関数値が改善されるためには非基底変数x_3に対する係数$(a_1/3+a_2/5-1)$が正の値にならなければならない。
- パターン3で切り出す製品1・製品2の枚数 a_1, a_2 を改めて変数と考える。
- パターン3を選択したとき、目的関数の改善の度合が最も大きくなるには次の問題を解いた最適目的関数値が正になればよい。

列生成問題

$\frac{a_1}{3} + \frac{a_2}{5} - 1 > 0$でなければならない

a_1とa_2が切り出しパターンなので

$30a_1 + 20a_2 \le 100$

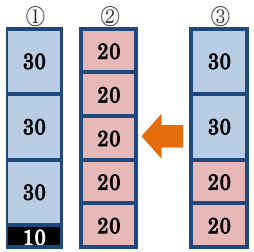
(列生成問題) $\max \frac{1}{3}a_1 + \frac{1}{5}a_2 - 1$
s.t. $30\,a_1 + 20\,a_2 \le 100$
a_1 , a_2 は 0 以上の整数

この列生成問題を解くと、

$a_1 = 2 \quad a_2 = 2$(新たなカッティングパターン)

例題2：新たな列の追加

100センチの板から30センチを2枚、20センチを2枚切り出すパターンを加え、3つのパターンから、100センチの板の使用枚数が最も少ない組み合わせを求める。



$$\min x_1 + x_2 + x_3$$
$$\text{s.t. } 3x_1 + 2x_3 = 60$$
$$5x_2 + 2x_3 = 80$$

計算結果 $x_2 = 4 \quad x_3 = 30$

最初の解では100センチの材料を36枚用いた
新たなパターンを加えると、材料が34枚の使用で済む

改訂単体法と列生成
線形計画問題の一般形

$$(\mathrm{LP})\quad \begin{array}{rl} \min & z = c^\top x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

制約行列の係数ベクトル
$A=(B,N)$=(基底行列, 非基底行列)
へと分解

$A=(B,N)$
$x^{\mathrm{T}}=(x_B{}^{\mathrm{T}}, x_N{}^{\mathrm{T}})$
$c^{\mathrm{T}}=(c_B{}^{\mathrm{T}}, c_N{}^{\mathrm{T}})$

$$(\mathrm{LPm})\quad \begin{array}{rl} \min & z = c_B^\top x_B + c_N^\top x_N \\ \text{subject to} & Bx_B + Nx_N = b \\ & x_B, x_N \geq 0 \end{array}$$

基底形式への変形

- 制約行列$Bx_B+Nx_N=b$を基底変数x_Bについて解く

$x_B+B^{-1}Nx_N=B^{-1}b \Rightarrow x_B=B^{-1}b-B^{-1}Nx_N$

- 目的関数からx_Bを消去

$$\begin{aligned} z &= c_B{}^{\mathrm{T}}x_B+c_N{}^{\mathrm{T}}x_N = c_B{}^{\mathrm{T}}(B^{-1}b-B^{-1}Nx_N)+c_N{}^{\mathrm{T}}x_N \\ &= c_B{}^{\mathrm{T}}B^{-1}b - c_B{}^{\mathrm{T}}B^{-1}Nx_N + c_N{}^{\mathrm{T}}x_N \\ &= c_B{}^{\mathrm{T}}B^{-1}b + (c_N{}^{\mathrm{T}} - c_B{}^{\mathrm{T}}B^{-1}N)\,x_N \end{aligned}$$

	z	x_B	x_N
$c_B{}^{\mathrm{T}}B^{-1}b$	1	0	$c_B{}^{\mathrm{T}}B^{-1}N - c_N{}^{\mathrm{T}}$
$B^{-1}b$	0	I	$B^{-1}N$

基底形式
単体表

- 基底解 $x_B=B^{-1}b$, $x_N=0$, 目的関数値 $z=c_B{}^{\mathrm{T}}B^{-1}b$

単体法：新たな列の生成

- 入力データA, b, cを保存して、必要となる場合、その都度元データから使用（列B,Nの情報も保持）

	z	x_B	x_N	
$c_B{}^{\mathrm{T}}B^{-1}b$	1	0	$c_B{}^{\mathrm{T}}B^{-1}N - c_N{}^{\mathrm{T}}$	
$B^{-1}b$	0	I	$B^{-1}N$	$B^{-1}a$

緑色の新たなカッティングパターンを表すベクトルを変数 $(a_1,a_2,\ldots,a_m)$とおく
このとき赤の部分が正であればこのカッティングパターンを選択すると目的関数値が改善される
次の列（カッティングパターン）生成問題を解く

$$\max \quad c_B^T B^{-1} \quad a - c_N^T$$

$$\text{subject to} \sum_{i=1}^{m} w_i\, a_i \leq r$$

弱双対定理

$$(\mathrm{LP})\quad \begin{array}{rl} \min & c^\top x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array} \qquad (\text{LP-dual}) \qquad \begin{array}{rl} \max & y^\top b \\ \text{subject to} & y^\top A \leq c^\top \end{array}$$

- **目的関数値 $c^{\mathrm{T}}x \geqq y^{\mathrm{T}}b$**

双対定理

- 主問題(LP)が最適解を持つ場合，双対問題(LP-dual)も最適解を持ち，$c^{\mathrm{T}}x$の最小値と$y^{\mathrm{T}}b$の最大値は一致する。

双対定理の証明(方針)

$$\text{(LP)}\ \begin{array}{|rl} \min & c^\top x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array} \qquad \text{(LP-dual)}\ \begin{array}{|rl} \max & y^\top b \\ \text{subject to} & y^\top A \leq c^\top \end{array}$$

- 主問題(LP)の最適基底をB,基底解を$x_B^* = B^{-1} b$, $x_N^* = 0$とする。
- $y^{*\mathrm{T}} = c_B^{\mathrm{T}} B^{-1}$とする。
- 次の2つを示せばよい。
- 1.目的関数値は一致 $c^{\mathrm{T}} x^* = y^{*\mathrm{T}} b$
- 2. y^*が双対問題(LP-dual)の実行可能解となる

双対定理の証明(概要)

$$\text{(LP)}\ \begin{array}{|rl} \min & c^\top x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array} \qquad \text{(LP-dual)}\ \begin{array}{|rl} \max & y^\top b \\ \text{subject to} & y^\top A \leq c^\top \end{array}$$

- 1.目的関数値は一致($x_B^* = B^{-1} b$, $x_N^* = 0$, $y^{*\mathrm{T}} = c_B^{\mathrm{T}} B^{-1}$)
  主問題: $c^{\mathrm{T}} x^* = c_B^{\mathrm{T}} x_B^* + c_N^{\mathrm{T}} x_N^* = c_B^{\mathrm{T}} B^{-1} b$
  双対問題: $y^{*\mathrm{T}} b = c_B^{\mathrm{T}} B^{-1} b$
- 2. y^*が双対問題(LP-dual)の実行可能解となる
  $y^{*\mathrm{T}} A \leqq c^{\mathrm{T}}$ を示す。$A=(B,N)$とすると,
  $y^{*\mathrm{T}} (B,N) \leqq (c_B^{\mathrm{T}}, c_N^{\mathrm{T}})$ より,
  第1成分$c_B^{\mathrm{T}} B^{-1} B \leqq c_B^{\mathrm{T}}$は明らかに成立
  第2成分$c_B^{\mathrm{T}} B^{-1} N \leqq c_N^{\mathrm{T}}$は$B$の最適性より成立

$c_B^{\mathrm{T}} B^{-1} N - c_N^{\mathrm{T}} \leqq 0$より

列生成問題

	z	x_B	x_N	
$c_B^{\mathrm{T}} B^{-1} b$	1	0	$c_B^{\mathrm{T}} B^{-1} N - c_N^{\mathrm{T}}$	
$B^{-1} b$	0	I	$B^{-1} N$	$B^{-1} a$

- 緑色の新たなカッティングパターンを変数($a_1, a_2, \ldots, a_m$)
- 負の被約費用(赤の部分)が正になればよい
- 列(カッティングパターン)生成問題を解く

$$\max \quad c_B^T B^{-1} \quad a - c_N^T \qquad \Rightarrow \qquad \max \quad \sum_{i=1}^{m} \pi_i a_i - 1$$

$$\text{subject to} \sum_{i=1}^{m} w_i a_i \leq r \qquad \text{subject to} \sum_{i=1}^{m} w_i a_i \leq r$$

目的関数の係数$c_B^{\mathrm{T}} B^{-1}$は$Ax=b$の最適双対解π

演習問題

- 先の例1において、初期列パターンを1種類の製品のみを切り出す4列とする。
- 新たに生成する列は、生成される製品の長さの和が、原材料のある一定の割合αを越えなければならないとする。
- 現在の最適解の値(原材料の枚数453枚)が変わるようなαの範囲を求めよ。
- 実際に列生成法を用いて考察すること。

最適化シミュレーション演習

第6回 最短路問題
ラグランジュ緩和問題

- 授業サポートページ
  http://www.shiina.mgmt.waseda.ac.jp/
- ネットワーク問題の基礎
- 最短路問題Dijkstra法
- ラグランジュ緩和問題

ネットワークに関する定義

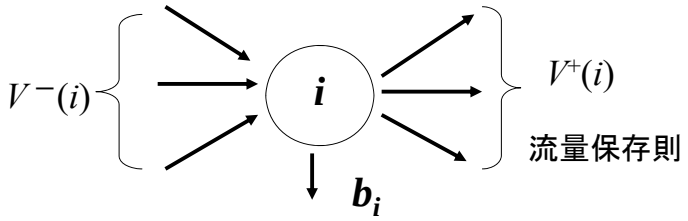
- 有向グラフ $G=(V, A)$を考える（V: 点集合、A: 辺集合）
- 各辺$(i, j) \in A$に対して辺容量h_{ij} が与えられている
- 各点$i \in V$に対してb_i が与えられている（ $b_i>0$ ならば需要量、$b_i<0$ならば供給量）
- x_{ij} は辺(i, j)上のフロー（流量）を表す
- $V^+(i)=\{k: (i,k)\in A\}$ 点i から出る辺の終点集合
- $V^-(i)=\{k: (k,i) \in A\}$ 点i に接続する辺の始点集合
- 参考文献: Ahuja, Magnanti, Orlin, NETWORK FLOWS, Prentice Hall, 1993.

最小費用流問題

- 定式化 $\min \sum_{(i,j)\in A} c_{ij} x_{ij}$

$$\text{subject to } \sum_{k\in V^-(i)} x_{ki} - \sum_{k\in V^+(i)} x_{ik} = b_i, \ \forall i \in V$$

$$0 \le x_{ij} \le h_{ij}, \forall (i,j) \in A$$



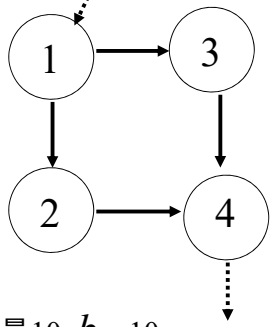
ネットワークの例

- 流量保存則

$-x_{12}-x_{13} = -10$, 点1への供給量10
$x_{12} - x_{24} = 0$, 点2への流入量＝流出量
$x_{13} - x_{34} = 0$, 点3への流入量＝流出量
$x_{24}+x_{34} = 10$, 点4での需要量10

$$\begin{pmatrix} -1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_{12} \\ x_{13} \\ x_{24} \\ x_{34} \end{pmatrix} = \begin{pmatrix} -10 \\ 0 \\ 0 \\ 10 \end{pmatrix}$$

接続行列



接続行列の性質

- 線形計画問題として定式化
- b,hが整数ベクトルならば最適解は必ず整数解
- 接続行列：完全単模性を持つ（任意の小行列式が0,-1,1のどれかに等しい (totally unimodular)
- クラメルの公式：

  基底解 $x_B=B^{-1}b=(1/\det B)(\mathrm{adj}\ B)b$ の分母が－1または1⇒整数性

最短路問題（Shortest Path Problem)

- ネットワーク上の点s（始点；ソース）から点t（終点；シンク）までの最短路
- 辺(i,j)の距離：c_{ij}

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

subject to

$$\sum_{k\in V^-(i)} x_{ki} - \sum_{k\in V^+(i)} x_{ik} = -1,\ i = s$$

$$\sum_{k\in V^-(i)} x_{ki} - \sum_{k\in V^+(i)} x_{ik} = 0,\ \forall i \in V \setminus \{s,t\}$$

$$\sum_{k\in V^-(i)} x_{ki} - \sum_{k\in V^+(i)} x_{ik} = 1,\ i = t$$

$$0 \le x_{ij},\ \forall (i,j) \in A$$

最適解は$x_{ij}\in\{0,1\}$

最短路問題の双対問題

$$\max\ \pi_t - \pi_s$$
$$\text{subject to}\ \ \pi_j - \pi_i \le c_{ij},\ \forall (i,j) \in A$$

- 任意のaについてπ_iをπ_i+a と置き換えても実行可能⇒ 始点で$\pi_s=0$と定めてよい
- π_iとは？⇒始点sから点iへの最短距離（の下界）
- π_iをsからiへの最短距離と定めると双対問題の実行可能解なので、双対問題を解いてもsからtへの最短距離が求められることがわかる

Dijkstra（ダイクストラ）法

- ステップ1：　$\pi_1=0,\ \pi_i=\infty\ \ (i\in V-\{1\})$,
  $M=\{2,...,n\}$, $i=0$
- ステップ2：　$M\neq\phi$（空集合）でない場合(i)(ii)(iii)を繰り返す

  (i) $j\in M$に対し$\pi_j > \pi_i+c_{ij}$ならば$\pi_j=\pi_i+c_{ij}$, $p_j=i$

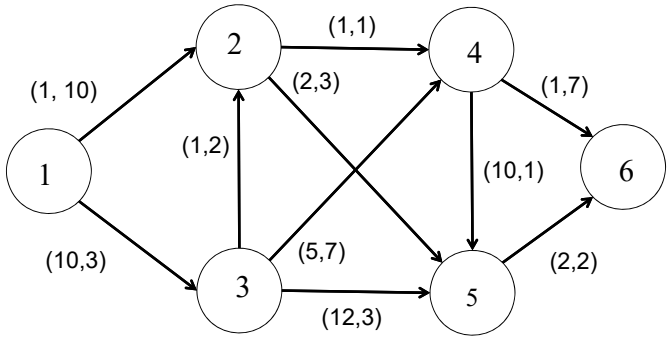
  (ii) $\min_{j\in M} \pi_j = \pi_k$となる$k$を求める

  (iii) kをMから除く、$i=k$ とする

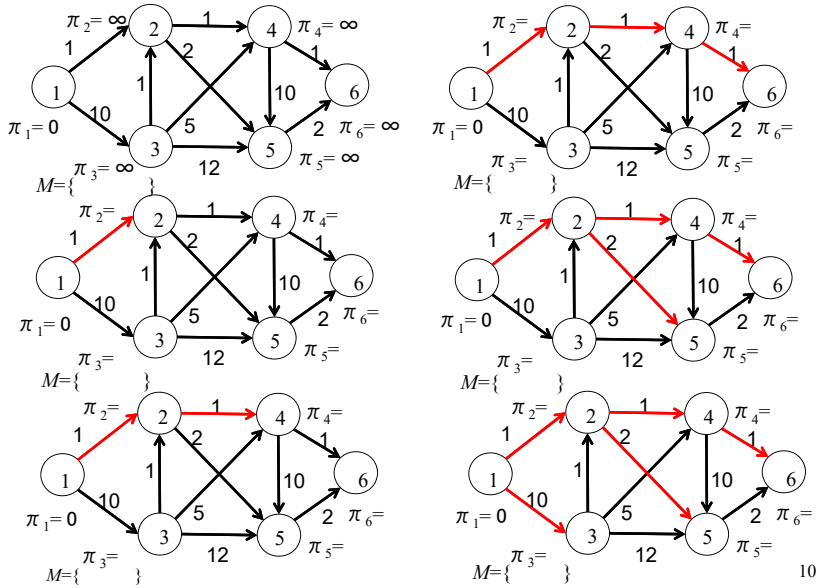
- M：最短路未確定な点の集合
- 集合Mから削除されると最短路確定
- p_j：点jへの最短路の直前点（終了時）

最短路問題：演習

- 辺上の数字は(c_{ij}, t_{ij})=(辺の長さ, 時間)



費用$c_{ij}+\mu t_{ij}(\mu=0)$



最短路問題:時間制約付

- 辺(i,j)の距離：c_{ij}, 辺(i,j)を移動するために要する時間：t_{ij}
- 制約（総移動時間 $\leqq T$）を考慮⇒難しい問題
- 制約（総移動時間 $\leqq T$）に乗数λを掛けて目的関数に入れる
- ラグランジュ緩和問題
- 以下の目的関数値は、時間制約付問題の最適目的関数値の下界

$$L(\mu) = \min \sum_{(i,j)\in A} c_{ij} x_{ij} + \mu\left(\sum_{(i,j)\in A} t_{ij} x_{ij} - T\right)$$

subject to

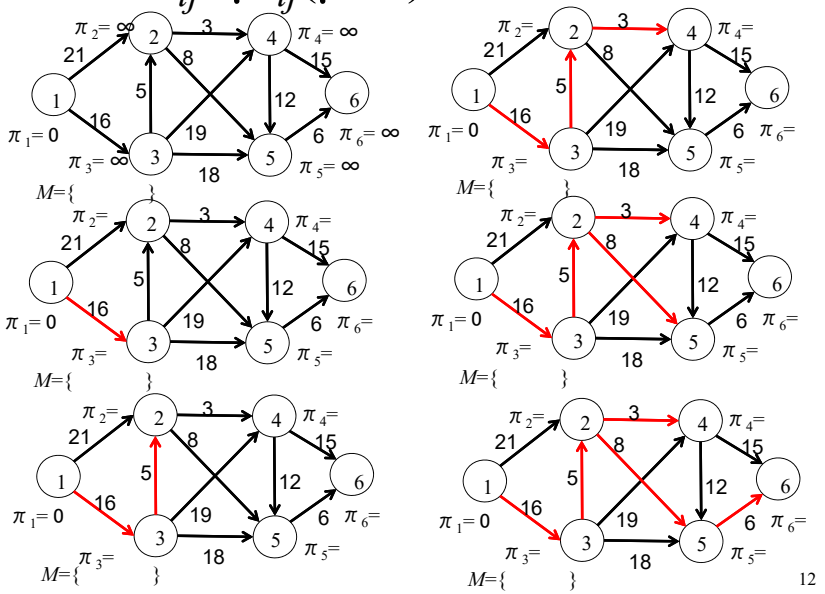
$$\sum_{k\in V^-(i)} x_{ki} - \sum_{k\in V^+(i)} x_{ik} = -1, \quad i = s$$

$$\sum_{k\in V^-(i)} x_{ki} - \sum_{k\in V^+(i)} x_{ik} = 0, \quad \forall i \in V \setminus \{s,t\}$$

$$\sum_{k\in V^-(i)} x_{ki} - \sum_{k\in V^+(i)} x_{ik} = 1, \quad i = t$$

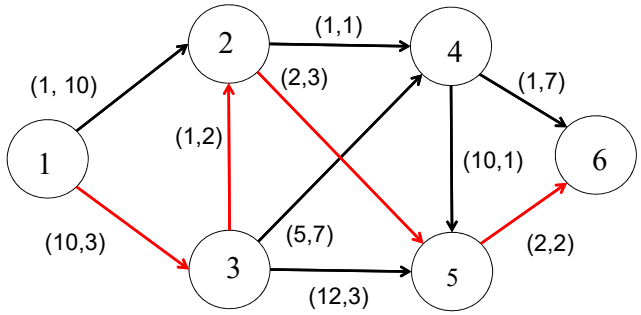
$$x_{ij} \in \{0,1\}, \ \forall (i,j) \in A$$

費用$c_{ij}+\mu t_{ij}(\mu=2)$課題：Dijkstraプログラム修正



緩和問題(μ=2の場合)の最適解

- T=10の場合、$L(2)$=15+2(10−T)=15(最適目的関数値の下界)
- 13256とたどるパスの時間は10時間(時間制約を満たす実行可能解)
- 13256とたどるパスの距離は15(下界と等しい)
- よって、13256は時間制約付最短路問題の最適解



ラグランジュ緩和法

- 元問題(最適目的関数値を z^*)　min cx
  subject to $Ax=b$, $x \in X$
- 難しい制約$Ax=b$を緩和(ラグランジュ乗数μ)

- ラグランジュ緩和問題　$L(\mu)=\min_x cx+\mu(Ax-b)$
  (最小値を$L(\mu)$、明らかに $L(\mu) \leqq z^*$)　subject to $x \in X$
- ラグランジュ関数 $L(\mu)$のμに関する最大値 $L^*=\max_{\mu \geqq 0} L(\mu)$
- $L(\mu) \leqq L^* \leqq z^* \leqq cx$

- 等号制約の場合の最適性テスト:(a)元問題の実行可能解xが、ラグランジュ乗数μに対し、$L(\mu)=cx$を満たすとき、xは元問題の最適解であり、$L^*=L(\mu)$.
- (b)ラグランジュ乗数μに対して $L(\mu)$を与えるx^*が元問題の実行可能解ならば、x^*は元問題の最適解であり、$L^*=L(\mu)$.

関数$L(\mu)$の性質

- 不等号制約の場合の最適性テスト(不等号制約$Ax \leqq b$を緩和):あるμ≧0に対して $L(\mu)$を与えるx^*が元問題の実行可能解であり、かつ$\mu(Ax^*-b)=0$のとき、x^*は元問題の最適解であり、$L^*=L(\mu)$.(これは十分条件にすぎないことに注意。)

- ラグランジュ緩和問題　$L(\mu)=\min cx+\mu(Ax-b)$
  (最小値を$L(\mu)$)　subject to $x \in X$
- $X=\{x^1,x^2,\ldots,x^K\}$の場合、$L(\mu)=\min_{k=1,\ldots,K} cx^k+\mu(Ax^k-b)$
- 最大の下界を求める
  $L^*=\max_{\mu \geqq 0} L(\mu)=\max\{w \mid w \leqq cx^k+\mu(Ax^k-b),\ k=1,\ldots,K,\ \mu \geqq 0\}$

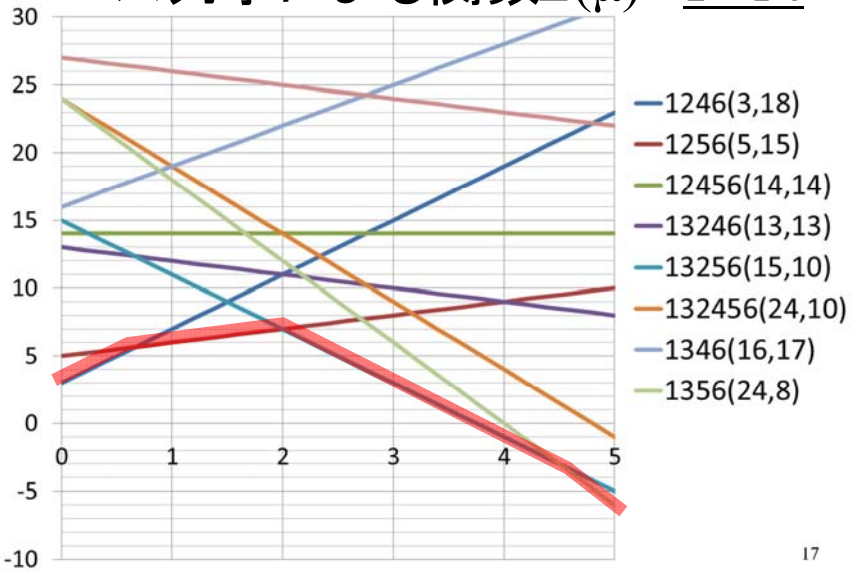
- 時間制約付最短路問題の場合、Xは始点と終点を結ぶパスの集合(理論的には列挙できる)
- 最大の下界L^*を求める問題は線形計画問題

全ての1-6パスを列挙:T=14の場合

パス	距離 c_p	時間 t_p	$c_p+\mu(t_p-T)$
1246	3	18	3+4μ
1256	5	15	5+μ
12456	14	14	14
13246	13	13	13-μ
13256	15	10	15-4μ
132456	24	9	24-5μ
1346	16	17	16+3μ
13456	27	13	27-μ
1356	24	8	24-6μ

- T=14の場合、最適距離は13、そのときの時間は13(パス13246)
- $L(2)$=5+2×(15-14)=7(最適目的関数値13の下界)
- ラグランジュ緩和問題の解1256の時間は15時間(実行不可能)
- ラグランジュ緩和問題の解1256の距離は5(実行不可能)

パス列挙による関数$L(\mu)$：T=14



17

下界の最大値L^*を求める

- 関数$L(\mu)$の最大化：関数$f(x)$への勾配法
- Taylor展開$f(x+\theta d)=f(x)+\theta\nabla f(x)d+o(\|\theta d\|)$

$\lim_{\theta\to 0}\frac{f(x+\theta d)-f(x)}{\theta}=\nabla f(x)d$ 解xからd方向にステップサイズθ

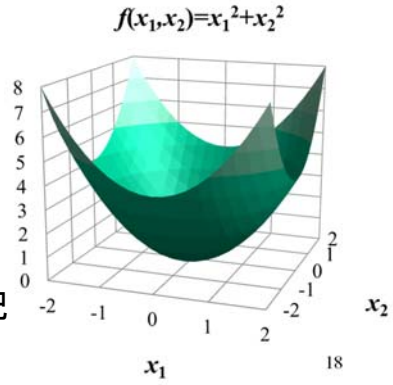
$d=\nabla f(x)$最急上昇方向

$d=-\nabla f(x)$最急降下方向

右の例

$$\nabla f(x)=\left(\frac{\partial f}{\partial x_1},\frac{\partial x}{\partial x_2}\right)=\left(2x_1,2x_2\right)$$

$f(x)$が微分不可能ならば劣勾配



18

列勾配法

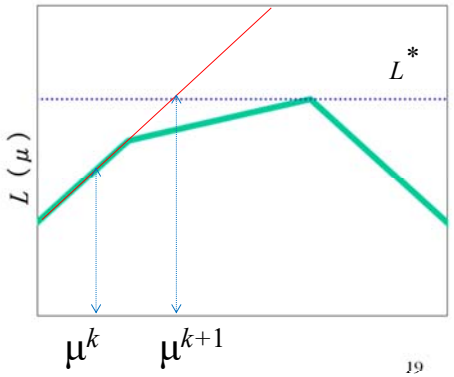
- 関数$L(\mu)=cx+\mu(Ax-b)$の場合、$d=Ax-b$
- $\mu^{k+1}=\mu^k+\theta_k(Ax-b)$と更新
- L^*がわかっていれば、$cx^k+\mu^{k+1}(Ax^k-b)=L^*$となるようにする

$$\theta_k=\frac{L^*-L(\mu^k)}{\left\|Ax^k-b\right\|^2},$$

実際には$\theta_k=\dfrac{\mathrm{UB}-L(\mu^k)}{\left\|Ax^k-b\right\|^2}$

または$\theta_k=\dfrac{\lambda_k(\mathrm{UB}-L(\mu^k))}{\left\|Ax^k-b\right\|^2}$

ただし$0<\lambda_k<2$



19

列勾配法の例

- 例：$\mu^0=0$のとき、パス1246が最適解
- $Ax^0-b=18-14=4$

$$\theta_0=\frac{L^*-L(\mu^0)}{\left\|Ax^0-b\right\|^2}=\frac{7-3}{4^2},$$

$$\mu^1=\mu^0+\theta_0\left(Ax^0-b\right)$$

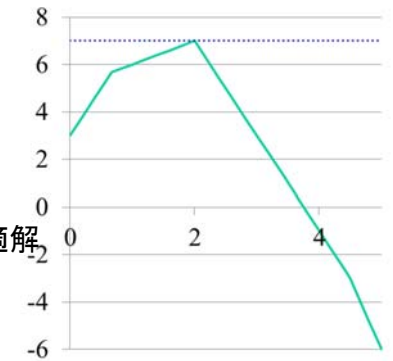
$$=0+1/4\times 4=1$$

- 例：$\mu^1=1$のとき、パス1256が最適解
- $Ax^1-b=15-14=1$

$$\theta_1=\frac{L^*-L(\mu^1)}{\left\|Ax^1-b\right\|^2}=\frac{7-(5+1)}{1^2},$$

$$\mu^2=\mu^1+\theta_1\left(Ax^1-b\right)$$

$$=1+1\times 1=2$$



- 課題：T=12のときラグランジュ緩和法により問題を解け。
- 得られた解は最適解であるかどうか検討せよ。

20

最適化・シミュレーション演習

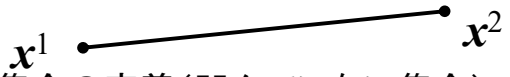
第7回 Benders分解

- 授業サポートページ
  http://www.shiina.mgmt.waseda.ac.jp/
- 数理計画による最適化と(離散事象型)シミュレーションに関する演習を行う. 使用するソフトウェアは, AMPL(+GurobiまたはCPLEX), および, Simul8を想定している.
- 凸解析の基礎、凸集合、凸関数
- Benders分解による施設配置問題
- Benders分解の一般論

凸結合、凸集合

- 凸結合の定義(2点を結ぶ線分上の点)

$x^1, x^2 \in \Re^n, 0 \le \lambda \le 1$ であるとき,
$\lambda x^1 + (1-\lambda)x^2$ を x^1 と x^2 の凸結合という.



- 凸集合の定義(凹んでいない集合)

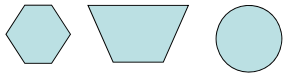
$S \subset \Re^n$ は, S の任意の 2 点 x^1, x^2 の凸結合を含むとき,
すなわち

$$x^1, x^2 \in \Re^n, 0 \le \lambda \le 1 \Rightarrow \lambda x^1 + (1-\lambda)x^2 \in S$$

を満たすとき, 凸集合であるという.

実行可能集合
LP:凸集合
MIP, IP: 非凸集合

端点(頂点)



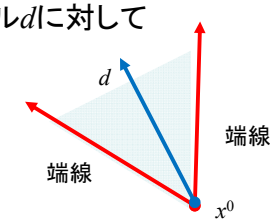
- 端点(頂点)の定義(端の点は異なる2点の間にない)

凸集合S上の点xはSに含まれる相異なる2点$x^1, x^2 \in S$を結ぶ線分上の中間点とならないとき、Sの端点であるという。

$x=\lambda x^1 + (1-\lambda)x^2,\ 0<\lambda<1 \Rightarrow x=x^1=x^2$

- 無限端線と端線

凸集合S上の点x^0と0でないベクトルdに対して
$x^0+\lambda d \in S,\ \forall \lambda \ge 0$となるとき
$\{x \mid x^0+\lambda d,\ \forall \lambda \ge 0\}$を起点とする
方向dの無限射線という



- 端線

並行でない異なる2つの無限射線ベクトルの和として表せない無限射線を端線という

凸関数(下に凸となる関数)

定義　凸関数(convex function)

凸集合$S \subseteq R^n$に対して定義される関数$f: S \to R^1$を考える。関数$f(x)$はS上の2点$x^1, x^2 \in S$に対して

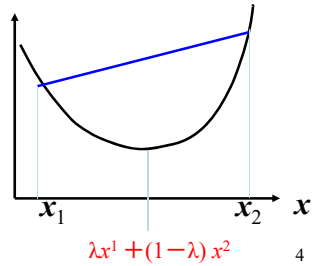
$$f(\lambda x^1 + (1-\lambda)x^2) \le \lambda f(x^1) + (1-\lambda) f(x^2)$$

$\lambda \in R^1,\ 0 \le \lambda \le 1$

のときS上の凸関数という

(−fが凸関数のとき、fを凹関数)

凸集合上で凸関数を最小化:凸計画
凸計画は容易に解ける場合が多い



今回扱う数理計画問題

$$\begin{aligned}\min\ & cx+qy\\ \text{subject to}\ & Ax=b,\ x\ge 0,\ x\in X\\ & Tx+Wy=h,\ y\ge 0\end{aligned}$$

異なる性質の変数を含む

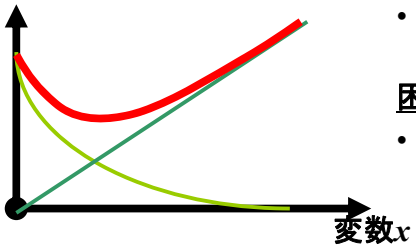
x：0－1変数，施設配置など

y：連続変数，輸送量など

多くの変数（特にyの次元が高い場合）を含む⇒変数yを消去

$$\begin{aligned}\min\ & cx+Q(x)\\ \text{subject to}\ & Ax=b,\ x\ge 0,\ x\in X\end{aligned}$$

- 変数x, yを含む大規模問題
- 変数yのみを含む小規模問題へ分解
- 目的関数：線形⇒凸計画
  yに関わるコストを$Q(x)$



困難なポイント

- 変形の方法、凸計画（非線形）の取り扱い

施設配置問題の変形

$$\begin{aligned}\min\ & \sum_{i\in I} f_i x_i+\sum_{i\in I}\sum_{j\in J} q_{ij}y_{ij}\\ \text{subject to}\ & \sum_{i\in I} y_{ij}=d_j,\ j\in J\\ & \sum_{j\in J} y_{ij}\le C_i x_i,\ i\in I\\ & x_i\in\{0,1\},\ i\in I,\ y_{ij}\ge 0,\ i\in I,\ j\in J\end{aligned}$$

通常の定式化（固定費＋輸送費）最小化

$$\begin{aligned}\min\ & \sum_{i\in I} f_i x_i+Q(x)\\ \text{subject to}\ & x_i\in\{0,1\},\ i\in I\\ \text{where}\ & Q(x)=\min\Big\{\sum_{i\in I}\sum_{j\in J} q_{ij}y_{ij}\ \Big|\ \sum_{i\in I} y_{ij}=d_j,\ j\in J,\ \sum_{j\in J} y_{ij}\le C_i x_i,\ i\in I,\ y_{ij}\ge 0,\ i\in I,\ j\in J\Big\}\end{aligned}$$

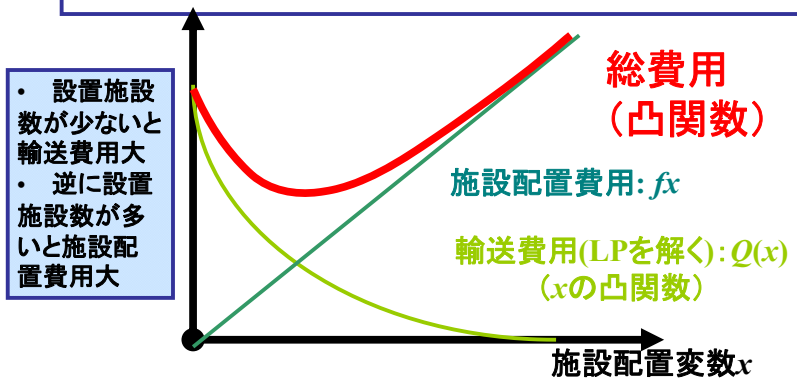
Benders分解による定式化
輸送費を施設配置変数xの関数$Q(x)$と変形

2段階最適化：（$Q(x)$）が最小化問題を解いて定義される（困難なポイント）
最小化の中に最小化が入っている

問題の取り扱いにおける難しさ

$$Q(x)=\min\Big\{\sum_{i\in I}\sum_{j\in J} q_{ij}y_{ij}\ \Big|\ \sum_{i\in I} y_{ij}=d_j,\ j\in J,\ \sum_{j\in J} y_{ij}\le C_i x_i,\ i\in I,\ y_{ij}\ge 0,\ i\in I,\ j\in J\Big\}$$

- 輸送費用関数$Q(x)$は陽に与えられていない（非線形）
- 施設配置変数xが与えられた後に輸送費最小化問題を解くことによって定義される（困難なポイント）



- 設置施設数が少ないと輸送費用大
- 逆に設置施設数が多いと施設配置費用大

例題：施設配置問題

- 1施設，設置費用100，容量20
- 1需要地，需要4，輸送費用2
- 需要制約、容量制約、総需要≦総容量

$$\begin{aligned}\min\ & 100x+2y\\ \text{subject to}\ & y=4,\ y\le 20x, 4\le 20x,\\ & y\ge 0, x\in\{0,1\}\end{aligned}$$

- 輸送費用を$Q(x)$と表す
- $Q(x)$は施設配置xの値が与えられた場合、LPを解いて計算可能
- 最小化して得られる目的関数を最小化できるか？

$$\begin{aligned}\min\ & 100x+Q(x)\\ \text{subject to}\ & 4\le 20x, x\in\{0,1\}\\ & Q(x)=\min\{2y\mid y=4,\ y\le 20x, y\ge 0\}\end{aligned}$$

- 問題MASTER
- $Q(x)$を消去
  （困難なポイントの解消）
- **$\theta\ge Q(x)$（θは$Q(x)$の上界）となるθを導入**

$$\begin{aligned}(\text{MASTER})\min\ & 100x+\theta\\ \text{subject to}\ & 4\le 20x,\ x\in\{0,1\},\theta\ge 0\end{aligned}$$

現時点では$\theta\ge Q(x)$を満たしていないが後から満たされるように制約を追加する（困難なポイントの解消）

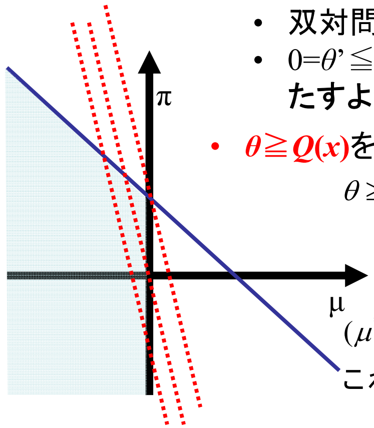
例題:最適性カット(困難なポイントの解消)

(MASTER) min $100x+\theta$

subject to $4 \le 20x,\ x \in \{0,1\}, \theta \ge 0$

- 主問題の最適解は$x'=1, \theta'=0$

$$Q(x') = \min\{2y \mid y=4, y \le 20x', y \ge 0\}$$
$$= \max\{20x'\mu + 4\pi \mid \mu + \pi \le 2, \mu \le 0\}$$



- 双対問題の最適解$(\mu^*,\pi^*)=(0,2)$
- $0=\theta' \leqq Q(x')=8$となる($\theta \geqq Q(x)$を満たすようにしたい)
- $\theta \geqq Q(x)$を満たすために最適性カットを追加

$$\theta \ge Q(x) = \max\{20x\mu + 4\pi \mid \mu + \pi \le 2, \mu \le 0\}$$
$$\ge 20x\mu^* + 4\pi^* = 8$$

(μ^*,π^*)はx'に対する双対最適解だから

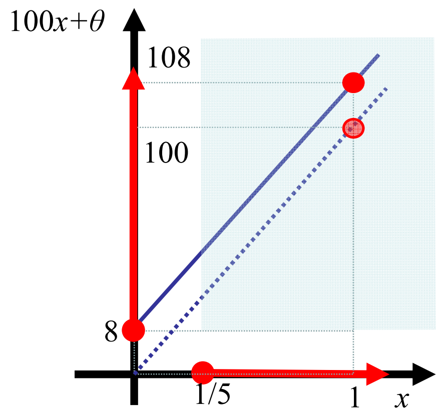
これは$x'=1, \theta'=0$を排除

例題:問題MASTER(最適性カット追加後)

(MASTER) min $100x+\theta$

subject to $x \in \{0,1\}, \theta \ge 0, 20x \ge 4$

$\theta \ge 8$



- マスター問題
- 最適性カット$\theta \geqq 8$を追加
- 最適性カットは、需要を満たすためには輸送費が8以上となることを示す
- 最適解$x=1$, $\theta=8$
- 最適目的関数値108

施設配置問題への応用:定式化

$\min \sum_{i \in I} f_i x_i + Q(x)$ （固定費＋輸送費）最小化

subject to $x_i \in \{0,1\},\ i \in I$ 施設配置変数∈{0,1}

$\sum_{i \in I} C_i x_i \ge \sum_{j \in J} d_j$ 総容量≧総需要

where $Q(x) = \min\{\sum_{i \in I}\sum_{j \in J} q_{ij} y_{ij} \mid$ $Q(x)$の定義:輸送費

$\sum_{i \in I} y_{ij} = d_j, j \in J$ 制約:需要充足

$\sum_{j \in J} y_{ij} \le C_i x_i,\ i \in I$ 制約:設備容量

$y_{ij} \ge 0, i \in I, j \in J\}$

施設配置変数 x
輸送量変数 y
輸送費用$Q(x)$

施設配置xに応じて輸送費$Q(x)$を決定
$LB \leqq z^*$(本問題の最適目的関数値)$\leqq UB$

Benders分解による施設配置問題

(MASTER): $\min \sum_{i \in I} f_i x_i + \theta$

subject to $x_i \in \{0,1\}, i \in I, \theta \ge 0,$

$\sum_{i \in I} C_i x_i \ge \sum_{j \in J} d_j$

$Q(x') = \min\ \{\sum_{i \in I}\sum_{j \in J} q_{ij} y_{ij} \mid$

$\sum_{j \in J} y_{ij} \le C_i x'_i,\ i \in I$

$\sum_{i \in I} y_{ij} = d_j, j \in J, y_{ij} \ge 0, i \in I, j \in J\}$

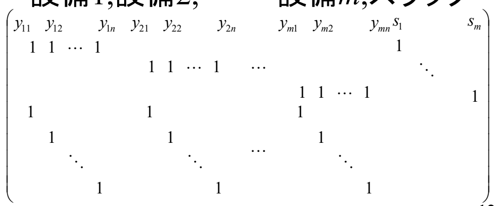
$= \max\{\sum_{i \in I} C_i x'_i \mu_i + \sum_{j \in J} d_j \pi_i \mid$

$\mu_i + \pi_j \le q_{ij},\ i \in I, j \in J$

$\mu_i \le 0,\ i \in I\}$

- MASTER問題を解く($Q(x)$を含まないことに注意)
- 最適目的関数値$=LB$
- MASTER問題から得られたx'を用いて$Q(x')$を求める
- $UB=LB-\theta+Q(x')$
- 双対問題の定式化を確認(制約行列の形より)

設備1,設備2,　設備m,スラック

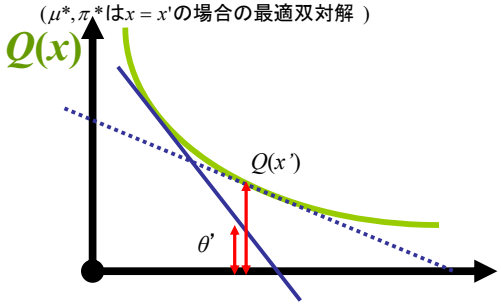


Benders分解:最適性カット

$$Q(x) \geq \max\{\sum_{i \in I} C_i x_i \mu_i + \sum_{j \in J} d_j \pi_i \mid \mu_i + \pi_j \leq q_{ij},\ i \in I, j \in J,\ \mu_i \leq 0,\ i \in I\}$$

$$= \sum_{i \in I} C_i x_i \mu_i^* + \sum_{j \in J} d_j \pi_i^*$$

(μ^*, π^*は$x = x'$の場合の最適双対解)



- マスター問題を解いて得られた解について $\theta' < Q(x')$ のとき、$\theta \geqq Q(x)$となるように近似
- $Q(x)$は凸関数であるため、$Q(x)$の線形近似は常に$Q(x)$の下界となる
- **最適性カット**
- **x' ではなく、xは変数として不等式を主問題に追加**

$$\theta \geq \sum_{i \in I} C_i x_i \mu_i^* + \sum_{j \in J} d_j \pi_i^*$$

例題2:施設配置問題

- 2施設, 3需要地
- x=(0,1), θ=0
- LB=150,UB=2485

$$\begin{aligned}&\min\ 150x_1 + 150x_2 + \theta\\ &\text{subject to } 96x_1 + 66x_2 \geq 40, x \in \{0,1\}, \theta \geq 0\\ &Q(x) = \min\{49y_{11} + 79y_{12} + 44y_{13} + 51y_{21} + 79y_{22} + 37y_{23} \mid\\ &y_{11} + y_{12} + y_{13} \leq 96x_1, y_{21} + y_{22} + y_{23} \leq 66x_2\\ &y_{11} + y_{21} = 10, y_{12} + y_{22} = 15, y_{13} + y_{23} = 15, y \geq 0\}\end{aligned}$$

- 反復1
- x=(0,1), θ=0
- LB=150,UB=2485
- 反復2
- x=(0,1), θ=1873
- LB=2023,UB=2400
- 反復3
- x=(1,1), θ=2058
- LB=2358,UB=2400
- 反復4
- x=(0,1), θ=2250
- LB=2400,UB=2400

```
subject to optcut[1]:
        462*x[2] + theta >= 2335;

subject to optcut[2]:
        192*x[1] + theta >= 2250;

subject to optcut[3]:
        theta >= 2230;

subject to optcut[4]:
        192*x[1] + theta >= 2250;
```


異なる種類の変数を含む数理計画問題

$$\begin{aligned}\min\ & cx + qy\\ \text{subject to}\ & Ax = b,\ x \geq 0,\ x \in X\\ & Tx + Wy = h,\ y \geq 0\end{aligned}$$

異なる性質の変数を含む

x:第1段階,0−1変数など

y:第2段階変数,連続変数など

次のように問題を変形: Benders decomposition Method

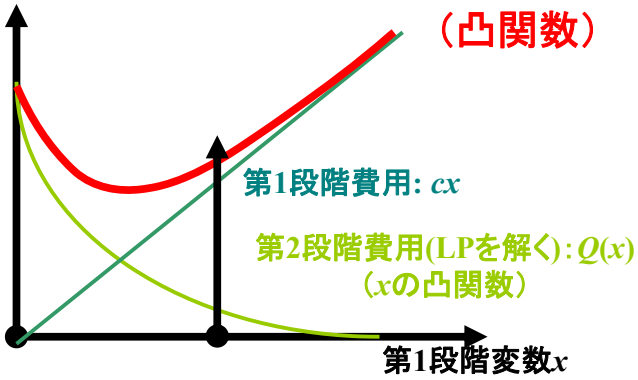
$$\begin{aligned}&\min\ cx + Q(x)\\ &\text{subject to}\ Ax = b,\ x \geq 0, x \in X\\ &Q(x) = \min\{qy \mid Tx + Wy = h, y \geq 0\}\end{aligned}$$

- 決定xに対して、制約 $Tx+Wy=h$ が満たされるように変数 $y \geqq 0$ に関連する費用$Q(x)$を目的関数に含める

解法のアウトライン

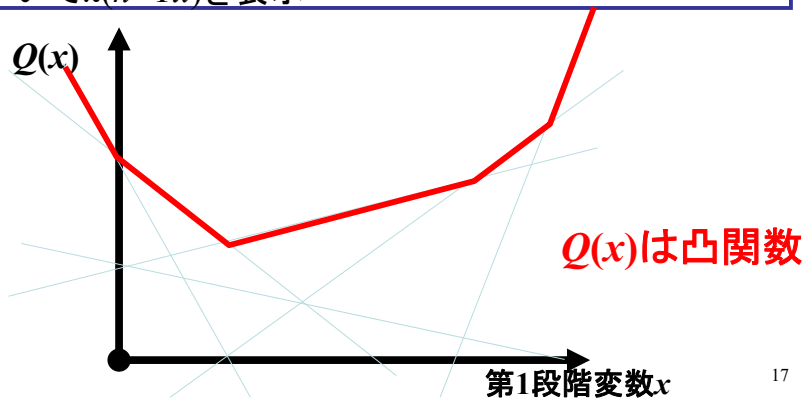
- 関数$Q(x)$は陽に与えられていない
- 第1段階変数xが与えられた後に第2段階のLPを解くことによって定義される

- $Q(x) = \min\{qy \mid Wy = h - Tx,\ y \geqq 0\}$



関数$Q(x)$の性質（詳しい証明は補足資料）

- $Q(x)=\min\{qy|Wy=h-Tx,\ y\geqq 0\}$
  $=\max\{\pi(h-Tx)|\pi W\leqq q\}$
- 双対問題の制約のすべての端点 $\pi^1,\pi^2,\ldots,\pi^K$について$\pi(h-Tx)$を表示



$Q(x)$は凸関数

マスター問題(Master Problem)

θは関数$Q(x)$の上界となる変数

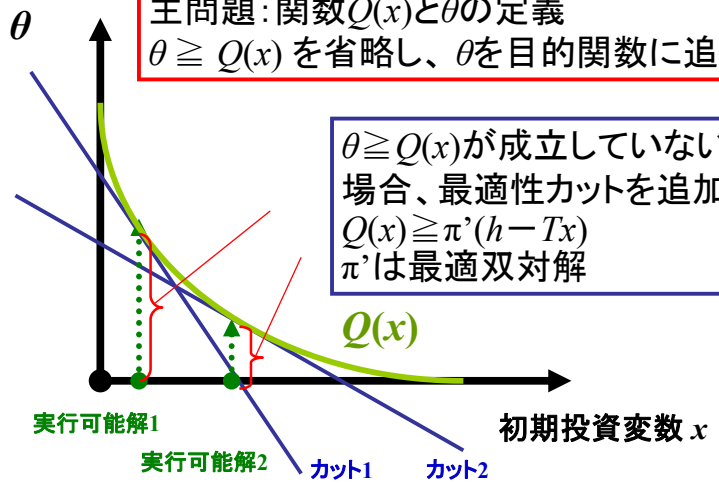
$$\min \quad cx+\theta$$
$$\text{subject to} \quad Ax=b,\ x\geq 0,\ x\in X$$
$$\theta\geq 0\text{(下界値)}\ \left(\theta\geq Q(x)\text{を省略}\right)$$

主問題：リコース関数とθの定義$\theta\geqq Q(x)$を省略
リコース関数$Q(x)$を近似する最適性カットを追加

- 関数$Q(x)$を定義する主問題が実行不可能である場合、
  $Q(x)=\min\{qy|Wy=h-Tx,\ y\geqq 0\}$
  $=\max\{\pi(h-Tx)|\pi W\leqq q\}=+\infty$
- $\pi'(h-Tx)>0$かつ$\pi' W\leqq 0$となる方向ベクトルπ'が存在
  これを除外するため実行可能性カット$\pi'(h-Tx)\leqq 0$を追加
  $(\pi_0+\theta\pi')(h-Tx)\to\infty$, $(\pi_0+\theta\pi')W\leqq q$は明らか

解法のアルゴリズム：区分線形近似

θ:関数$Q(x)$の上界となる変数
主問題：関数$Q(x)$とθの定義
$\theta\geqq Q(x)$ を省略し、θを目的関数に追加



$\theta\geqq Q(x)$が成立していない場合、最適性カットを追加
$Q(x)\geqq\pi'(h-Tx)$
π'は最適双対解

Benders分解法のアルゴリズム

ステップ0. 初期設定：暫定目的関数値$UB=+\infty$,
最適値に対する下界値$\mathrm{LB}=0$.

ステップ1. $(MASTER)$問題を解き、解 $(\hat{x},\hat{\theta},)$ を得る.

ステップ2. $Q(x)$を定義する第2段階問題が実行不可能ならば、
実行可能性カットを追加し、ステップ1へ。

ステップ3. 暫定目的関数値と下界値の更新：
$c\hat{x}+\hat{\theta}>LB$ならば$LB=c\hat{x}+\hat{\theta}$,
$c\hat{x}+Q(\hat{x},\hat{y})<UB$ならば$UB=c\hat{x}+Q(\hat{x},\hat{y})$.

ステップ4. 収束判定
If $LB\geq(1-\varepsilon)UB$ならば終了.(ε：許容誤差)

ステップ5. 最適性カットの追加
$\hat{\theta}<Q(\hat{x})$となれば,最適性カットを主問題に追加し、ステップ1へ.

補足資料：関数$Q(x)$の凸性

- $Q(x)=\min\{qy|Wy=h-Tx,\ y\geqq 0\}$
  $=\max\{\pi(h-Tx)|\pi W\leqq q\}$
- 元問題の2つの実行可能解x^1,x^2を考え、$Q(x^1)=qy^1$, $Q(x^2)=qy^2$であると仮定
- $Wy^1=h-Tx^1$, $Wy^2=h-Tx^2$より
  $W(\lambda y^1+(1-\lambda)y^2)=h-T(\lambda x^1+(1-\lambda)x^2)$
- $Q(\lambda x^1+(1-\lambda)x^2)$
  $=\min\{qy|Wy=h-T(\lambda x^1+(1-\lambda)x^2),\ y\geqq 0\}$
  $\leqq \lambda qy^1+(1-\lambda)qy^2$
  $=\lambda Q(x^1)+(1-\lambda)Q(x^2)$

補足Benders分解:実行可能性カット

$$Q(x')=\max\{\sum_{i\in I} C_i x_i' \mu_i+\sum_{j\in J} d_j\pi_i \mid \mu_i+\pi_j\leq q_{ij},\ i\in I, j\in J,\ \mu_i\leq 0,\ i\in I\}$$

- 2段階費用$Q(x')=+\infty$ならば、双対問題において、**無限方向（制約の端線）**が存在する

$$\max\{\sum_{i\in I} C_i x_i' \tilde{\mu}_i+\sum_{j\in J} d_j\tilde{\pi}_i \mid \tilde{\mu}_i+\tilde{\pi}_j\leq 0,\ i\in I, j\in J,\ -1\leq\tilde{\mu}_i\leq 0,\ i\in I,\ -1\leq\tilde{\pi}_j\leq 1,\ j\in J\}$$

- **端線生成問題**
- **方向だけが問題なので、各μ, πの成分に制限（長さ1以下）**

実行可能性カット

$$\sum_{i\in I} C_i x_i \tilde{\mu}_i+\sum_{j\in J} d_j\tilde{\pi}_i\leq 0$$

- **実行可能性カット**
- **x' ではなく、xは変数として主問題に追加**

最適化・シミュレーション演習

補足：切除平面法

- 授業サポートページ
  http://www.shiina.mgmt.waseda.ac.jp/ora/
- 数理計画による最適化と（離散事象型）シミュレーションに関する演習を行う．使用するソフトウェアは，AMPL（＋GurobiまたはCPLEX），および，Simul8を想定している．
- 演習では，数理計画による最適化やシミュレーションの実践的能力を身につけることを目指す。履修者は，Cを使用できる環境を有するPCを持参すること。受講者は，実験室にて，演習で使用する C，AMPL，Simul8をダウンロードできる。

整数計画法と妥当不等式

$$(\mathrm{IP}): \max \sum_{i=1}^{n} c_i x_i$$

subject to $x \in X$

$$X = \{x \mid Ax \le b, x \in Z_+^n\}$$

整数計画問題（IP)の実行可能集合をXとする。
すべての $x \in X$に対して $\pi x \leqq \pi_0$が成立するとき、
$\pi x \leqq \pi_0$をXに対する妥当不等式という。

問題(IP)の連続緩和問題 $\max\{cx \mid Ax \leqq b, x \in R^n\}$の最適解$x'$が $x' \in X$とならない場合、x'を排除したい。
すべての $x \in X$に対して $\pi x \leqq \pi_0$ を満たし、
$x' \notin X$に対して $\pi x \geqq \pi_0$ となる妥当不等式を切除平面という。

(IP)と(LP)の実行可能領域

$$(\mathrm{IP}): \max 2x_1 + x_2$$

subject to $x \in X$

$$X = \{x \mid 2x_1 + 2x_2 \le 3, x \in \{0,1\}^2\}$$

最適解は$x = (1,0)$

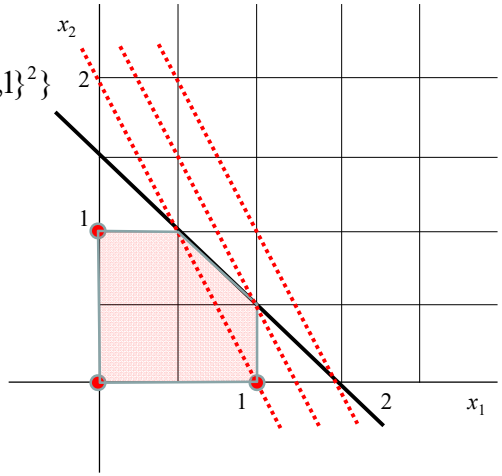
$$(\mathrm{LP}): \max 2x_1 + x_2$$

subject to $2x_1 + 2x_2 \le 3,$

$$0 \le x_1, x_2 \le 1\}$$

最適解は

$x^* = (1, \frac{1}{2})$



ナップサック問題に対する妥当不等式

（KP）最大化 $z = \Sigma_{j=1}^{n} c_j x_j$
制約 $\Sigma_{j=1}^{n} a_j x_j \leqq b$
$x_j \in \{0,1\}, \ \forall j$

被覆：集合$C \subseteq N = \{1,2,\ldots,n\}$に対して
$\Sigma_{j \in C} a_j > b$が成立するとき、Cをbに対する被覆という。
（容量bを上回る品物の組合せ）

妥当不等式：$\Sigma_{j \in C} x_j \leqq |C| - 1$
被覆Cに含まれる品物を全てナップサックに入れることはできない。
（容量bを上回る品物の組合せに対し、ナップサックに入れられる品物の個数は、多くてもCに属する品物の数－1）

実行可能領域と妥当不等式

$(IP): \max 2x_1 + x_2$

subject to $x \in X$

$X = \{x \mid 2x_1 + 2x_2 \le 3, x \in \{0,1\}^2\}$

$(LP): \max 2x_1 + x_2$

subject to $2x_1 + 2x_2 \le 3,$

$0 \le x_1, x_2 \le 1\}$

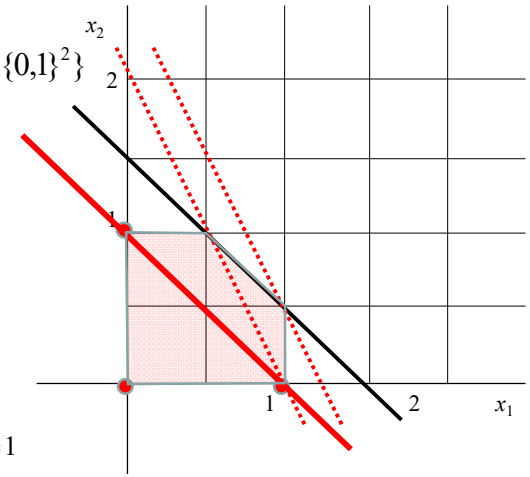
$\sum_{j \in C} x_j \le |C| - 1, \quad \sum_{j \in C} a_j > b$

$C = \{1,2\}$とすると

$\sum_{j \in C} a_j = 2 + 2 > b = 3$

妥当不等式は$x_1 + x_2 \le 2 - 1 = 1$

$x_1^* + x_2^* = 1 + 1/2 = 3/2 > 1$



妥当不等式と分離問題

(KP) 最大化 $z = \Sigma_{j=1}^{n} c_j x_j$

制約 $\Sigma_{j=1}^{n} a_j x_j \leqq b$

$x_j \in \{0,1\}, \ \forall j$

問題(KP)の連続緩和問題(KLP)の最適解 x^*に対して

$\Sigma_{j \in C} x_j^* > |C| - 1$

を満たすような被覆Cを求めたい。変形すると、以下を満たすCを求めればよい。

$\Sigma_{j \in C} (x_j^* - 1) > -1$

各品物jが被覆Cに属するか否かを示す0-1変数z_jを用いると、緩和LP最適解x_j^*を分離する妥当不等式を求める問題（分離問題）は以下のようになる。

分離問題: max violation$= \Sigma_{j \in N} (x_j^* - 1) z_j$

subject to $\Sigma_{j \in N} a_j z_j > b$

$z_j \in \{0,1\}, \ \forall j$

6

分離問題と貪欲解法

分離問題: max violation$= \Sigma_{j \in N} (x_j^* - 1) z_j$

subject to $\Sigma_{j \in N} a_j z_j \geqq b + \varepsilon$

$z_j \in \{0,1\}, \ \forall j$

先のデータを用いると、

$\max (x_1^* - 1) z_1 + (x_2^* - 1) z_2 = -(1/2) \ z_2$

subject to $2 z_1 + 2 z_2 \geqq 3 + \varepsilon$

$z_j \in \{0,1\}, \ \forall j$

この0-1整数計画問題の最適解は$z=(1,1)$

貪欲解法: $(x_j^* - 1) / a_j$ の大きい順にCに加えていく

この例では、a_1, a_2 の順に加える。

7

プログラミング言語としてのAMPLの基本

基礎まとめ

1. AMPL コマンドスクリプト

・AMPLのコマンドの列は *.run ファイル（バッチファイル）に記述される。
・実行はコマンドプロンプトから「ampl *.run」と入力する（*はファイル名）。
・コマンドスクリプトの終わりには「exit;」を記述し、AMPLを終了させる。
・コメントを記入したい場合は、行頭に「#」を入れる。#以降が無視される。
・コマンドを記述したら、行の終わりに「;」（セミコロン）を入れることを忘れてはならない。

2. 表示について

・display *; で*の内容が画面に表示される。
・display * > ファイル名; とすると画面ではなくファイルに書き込まれる。
・display 文は改行が複数行われるため、表示を制御したい場合はprintf 文を使う。
・例：「printf : "i= %d ¥n", j; 」の意味は、「j」の内容を「i= %d ¥n」という形式で表示させる。
・「%d」は整数型であり、「%f」は実数型を表す。
・記述形式はC 言語に準拠する。
・「¥n」は改行コードを示す。
・まとめると、「printf : "i= %d ¥n", j; 」は「i=」をテキストとして表示した後に、「j」を形式「%d」で表示して改行するというコマンドである。

3. パラメタ、変数の定義

・一般的にはコマンドスクリプトの中で使われるパラメタ（定数、あるいは式の値など）は最初に定義しなければならない。
・パラメタの定義は次のように行う。

```
param a; 「a というパラメタを定義」
var x; 「x という変数を定義」
```

・AMPLでは変数とは、数理計画問題の定式化で用いられる変数（ソルバで最適化されるもの）を表す。
・変数や反復回数などに応じて変わる値はすべてパラメタであることに注意が必要。

5. 代入文

・パラメタや変数に値を代入する場合、単純に「a=a+1;」と記述してはならない。
・正しくは、let 文を「let a:=a+1; 」（コロン＋等号）と記述する。

反復と条件文

1. 反復 for 文

・「a..b」（ただし、a<b は整数値）はａからｂまでの整数値の集合を表す。
・例：「1..10」は集合{1,2,3,…,10}を表す。for {i in 1..10} でｉが1から10までの反復を表す。
・例：1から10までの整数値を表示

```
for {i in 1..10}
{
        display i;
}
```

・例：2重ループ、以下のように2重ループも記述できる。

```
for {i in 1..5}
{
    for {j in 1..5}
    {

    }
}
```

・注意事項： for 文で使われた for {i in 1..5}などに含まれる反復回数を示すカウンタ「i」は、for 文の中でのみ使われていれば、パラメータとして定義する必要はない。

2. repeat while 文

・例：

```
repeat while(条件１ )
{
      条件1が成立する限りここに示すコマンドを実行する。
}
```

3. if-then-else 文

・ある条件が成立するとき、特定の操作を行うようにする。
・条件に対する算術式としての表現は以下のようなものがある。
=, <>, <=, >, >=, 等がある。等しいかどうかはC言語と異なり、単に「=」と記述。
・例：if(条件1)then

```
{
      条件1が成立するときここに示すコマンドを実行する。
}
else
{
      if-then での条件1が成立しないときにここに示すコマンドを実行する。
}
```

・else 以降は省略してもよい

4. 反復からの脱出

・break; コマンドは反復からの脱出を行う。

乱数の使用

1. 乱数の基礎

・区間(m,n)における一様分布に従う乱数はUniform(m,n)
・正規分布N(μ, σ2)に基づく乱数はNormal(μ, σ2)
・区間[0,2^{24})における整数値をとる一様乱数はIrand224( )で与えられる。

例：(0,1)における一様乱数を10個生成

```
option randseed 0; #乱数の初期値を更新 0以外の正整数の場合は同じ系列の乱数
for{i in 1..10}
{
   display Uniform(0,1); #Uniform(0,1)は0以上1以下の乱数
}
```

2. サイコロを振るシミュレーション

・ガウス記号[x] (x以下の最大整数) を表すには関数 floor(x)を用いる。
・x以上の最小整数を表すには関数 ceil(x)を用いる。

例：関数値ceil(Uniform(0,6))で1から6までの整数値をとる一様乱数を作成する。

```
option randseed 1;
for{i in 1..10}
{
   display ceil(Uniform(0,6)); #ceilの場合(0,6) floorの場合(1,7)
}
```

例：サイコロを1000回振って各目が出る回数をカウントせよ。

```
param ransu;
param count{i in 1..6};
for{i in 1..6}
{
        let count[i]:=0; #カウント値のリセット
}
option randseed 1;
for{i in 1..100}
{
        let ransu:=ceil(Uniform(0,6)); #乱数値をransuに与える
        for{j in 1..6}
        {
             if(ransu=j)  then     #乱数値がjとなれば
             {
                  let count[j]:=count[j]+1; #jが出る回数に1加える
              }
        }
}
display count;
```

例題1

1. 割算の基礎

割算 20÷6=3 あまり 2 の計算を行う。単に 20/6 を表示すると 3.3333 が答となる。商の 3 を表示するには、20÷6 の整数部分を表すことが必要である。ガウス記号[x] (x 以下の最大整数) を表すには関数 floor(x)を用いる。

```
param a;
param b;
let a:=20;
let b:=6;
display a/b;
display floor(a/b);
display a-b*floor(a/b); #floor は切り捨て関数 ceil は切り上げ
printf:"%d 割る % d は %d あまり %d ¥n",a,b,floor(a/b),a-b*floor(a/b);
```

2. 約数を求める

パラメータ c=20 の約数を求める。20 を 1 から 20 までの整数 i で割って、余りが 0 ならば i は c=20 の約数であることに注意して、以下の空白を埋めよ。

```
param c;
let c:=20;
for{空白 }
{
        if( 空白                        ) then #c を i で割ってあまりが 0 ならば
        {
        printf:"%d は %d の約数である ¥n",i,c; #i は c の約数→約数 i を表示
        }
}
```

3. 素因数分解

パラメータ a=88 を素因数分解すると、88=2*2*2*11 となる。88 を 2 から 88 までの数で割っていき、割り切れる場合は、88=2*44 なので、次は a=44 を 2 から 44 までの数で割っていく。

88=2*44, 44=2*22, 22=2*11, 11=1*11

終了するのは 11 を 11 で割った商が 1 となる場合である。終了判定条件を含めて次の空白を埋めよ。

```
param a; #a というパラメータを定義
let a:=88;
display a;
repeat while( 空白 )
{
        for{i in 空白 } #a を 2 から a までの数で割っている
        {
                if(空白 ) then #a を i で割って余りが 0 ならば
                {
                 let a:=  空白;  #a を更新
                 display i;
                 空白;  #ここで脱出すると repeat に戻る break;
                }
        }
}
```

例題2：ソーティング

1. 並べ替えの基本

次の5個のデータを小さい順に並べ替える。

```
param a{i in 1..5};
let a[1]:=8; let a[2]:=3; let a[3]:=6; let a[4]:=5; let a[5]:=4;
a[1], a[2], a[3], a[4], a[5]
```

8,	3,	6,	5,	4	初期値
3,	8,	6,	5,	4	1と2を比較して入れ替え
3,	6,	8,	5,	4	2と3を比較して入れ替え
3,	6,	5,	8,	4	3と4を比較して入れ替え
3,	6,	5,	4,	8	4と5を比較して入れ替え、最大の8がa[5]に入る
3,	6,	5,	4,	8	1と2を比較
3,	5,	6,	4,	8	2と3を比較して入れ替え
3,	5,	4,	6,	8	3と4を比較して入れ替え、8を除いて最大の6がa[4]に入る
3,	5,	4,	6,	8	1と2を比較
3,	4,	5,	6,	8	2と3を比較して入れ替え、6,8を除いて最大の5がa[3]に入る
3,	4,	5,	6,	8	1と2を比較、5,6,8を除いて最大の2がa[2]に入り、最小の3がa[1]

2. プログラミング

```
param a{i in 1..5};
let a[1]:=8;
let a[2]:=3;
let a[3]:=6;
let a[4]:=5;
let a[5]:=4;
param aa; #入れ替えのために用いるパラメータ
display a;
for{k in 0..3}
{
        for{j in 空白  }
        {
                if(条件  a[j] と a[j+1] の比較)then
                {
                 a[j] と a[j+1]の入れ替えを行う;
                }
        }
}
display a;
```

課題：上のデータを大きい順に並べよ。

課題：10個のデータを乱数により作成し、小さい順に並べよ。

例題3：ニュートン法

1. ニュートン法

非線形方程式 $f(x)=0$ を解く Newton 法について示す。関数 $y=f(x)$ をある x^0 において、直線（接線）で近似する。接線の方程式は以下のようになる。

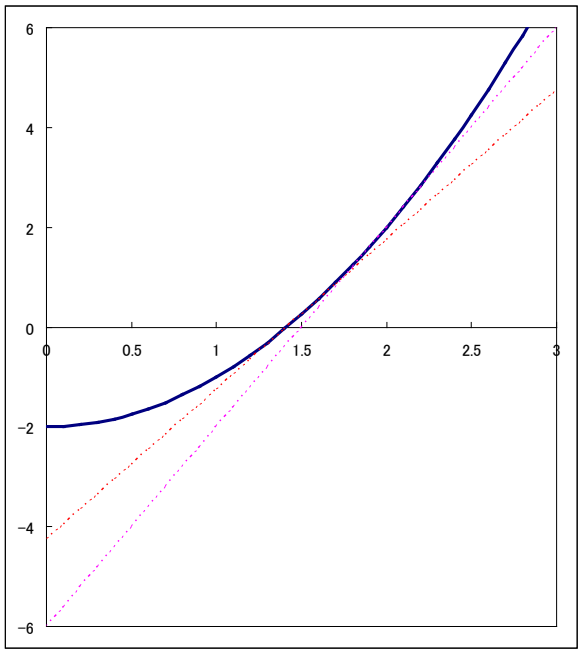
$y-f(x^0)=f'(x^0)(x-x^0)$

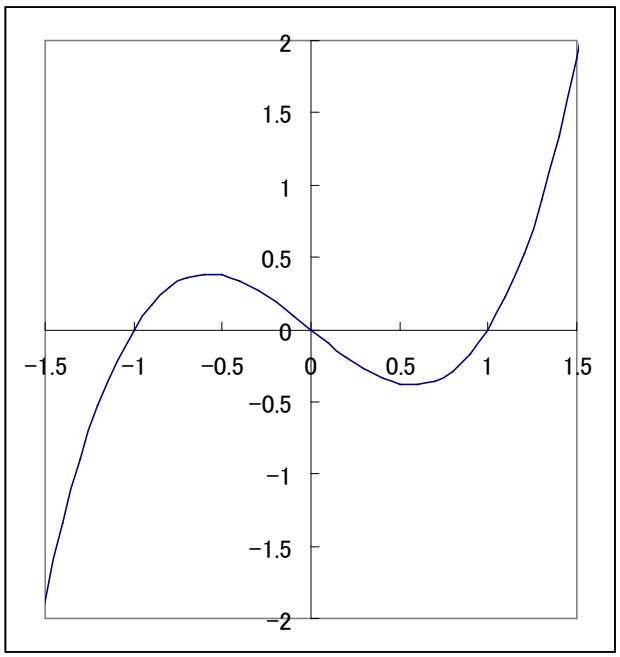
この接線が x 軸と交わる点を x^1 とすると、$f(x^0)$ よりも $f(x^1)$ の方が 0 に近いと考えられる。よって、接線が x 軸と交わる x^1 を求める。

$x^1=x^0-f(x^0)/f'(x^0)$

これより、以下の更新公式が導かれる。

$x^{n+1}=x^n-f(x^n)/f'(x^n)$





課題: $f(x)=x^2-2=0$ を満たす解を求める。

```
param x0;
param x1;
param i;
param n;
let i:=1; #反復回数
let n:=10; #反復回数の上限
let x0:=10; #初期値
display i, x0;
repeat while(i<n)
{
   let x1:=      空白;
   let x0:=x1; #得られた x1 を新たな初期値に
   let i:=i+1; #反復回数の更新
   display i, x0;
}
exit;
```

課題: f(x)= x(x−1)(x+1)=0 の解を 初期値-1.2, −0.3, 0.4, 0.8 の値について求めよ。

最適化・シミュレーション演習 第 2 回 数理計画問題の定式化

1. 輸送問題（イントロダクション）

工場 2 か所から倉庫 3 か所への輸送問題を考える。工場 i から倉庫 j への単位輸送量あたりの輸送コスト c_{ij} (i=1,...,2 , j=1,...,3), 工場 i からの供給量 S_i (i=1,...,2) , 倉庫 j の需要量 D_j (j=1,...,3)は表のように与えられている。最小費用となる各工場から各倉庫への輸送量を求める。

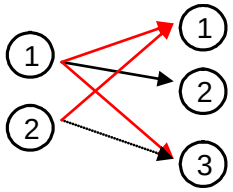
(1)ハウザッカー法により実行可能解を求め、表に輸送量を記入し、総輸送費用を求めよ。

倉庫 / 工場	倉庫 1	倉庫 2	倉庫 3	供給量
工場 1	6 10	5 15	4 10	35
工場 2	8 15	9 0	5 0	15
需要量	25	15	10	

総輸送費用

295

(2) 輸送量 0 の辺で最も輸送コストが安い辺を 1 つ選ぶと、輸送量が正の辺と閉路（ループ）をなす。この閉路を以下のグラフに記入せよ。またこのとき選んだ輸送量が 0 の辺に、輸送量を θ (≧0) 追加する場合、総輸送費用の変動量を θ の関数として表せ。



総輸送費用の変動量(辺 23 を選択)

=(θ 追加前の総輸送費用 − θ 追加後の総輸送費用)

$= (c23 - c13 + c11 - c21)\theta$ =(5−4+6−8)× θ =− θ

(3) 問題(2)のループの各辺における輸送量が非負であるという条件を用いて、最も輸送費用が少なくなる θ の値を求めよ。このように求められた θ を用いて変更された輸送量を以下の表に記入し、更新された総輸送費用を求めよ。

倉庫 / 工場	倉庫 1	倉庫 2	倉庫 3	供給量
工場 1	6 20	5 15	4 0	35
工場 2	8 5	9 0	5 10	15
需要量	25	15	10	

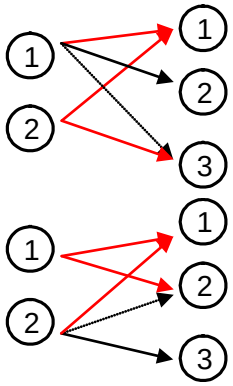
θ の値

10

総輸送費用

285

(4) 問題 1(3)で更新された解において、新たに輸送量が 0 となる辺を選ぶと問題 1(3)と同様に閉路が得られる。この閉路を以下の図に示せ。また、どの辺を選んで輸送量を θ (≧0) 追加させても、これ以上総輸送費用は改善できないこと、すなわち総輸送費用の変動量が正であることを示せ。



総輸送費用の変動量(辺 13 を選択)

=(θ 追加前の総輸送費用 − θ 追加後の総輸送費用)

$= (c13 - c23 + c21 - c11)\theta$ =(4−5+8−6)× θ = θ

総輸送費用の変動量(辺 22 を選択)

=(θ 追加前の総輸送費用 − θ 追加後の総輸送費用)

$= (c22 - c12 + c11 - c21)\theta$ =(9−5+6−8)× θ =2 θ

また、輸送問題を次のように線形計画問題として定式化する。

目的関数 $\min z = 6x_{11} + 5x_{12} + 4x_{13} + 8x_{21} + 9x_{22} + 5x_{23}$

制約条件(1−1) $x_{11} + x_{12} + x_{13} = 35$

制約条件(1−2) $x_{21} + x_{22} + x_{23} = 15$

制約条件(2−1) $x_{11} + x_{21} = 25$

制約条件(2−2) $x_{12} + x_{22} = 15$

制約条件(2−3) $x_{13} + x_{23} = 10$

非負条件 $x_{ij} \geq 0, i = 1,2,\ j = 1,...,3$

この問題を基底形式に変形するため、目的関数から $x_{11}, x_{12}, x_{13}, x_{21}$ を消去し、制約条件(1-1)を取り除く。また、制約(2-1)から制約(1-2)を引く。この操作により次の基底形式の線形計画問題が得られる。

目的関数 $\min z = 6(10 + x_{22} + x_{23}) + 5(15 - x_{22}) + 4(10 - x_{23}) + 8(15 - x_{22} - x_{23}) + 9x_{22} + 5x_{23}$

$= 2x_{22} - x_{23} + 295$

制約条件(1−2) $x_{21} + x_{22} + x_{23} = 15$

制約条件(2−1) $x_{11} - x_{22} - x_{23} = 10$

制約条件(2−2) $x_{12} + x_{22} = 15$

制約条件(2−3) $x_{13} + x_{23} = 10$

非負条件 $x_{ij} \geq 0, i = 1,2,\ j = 1,...,3$

この問題を単体法により解け。

基底変数	値	z	$x11$	$x12$	$x13$	$x21$	$x22$	$x23$	θ
z	295	1	0	0	0	0	-2	1	
$x21$	15	0	0	0	0	1	1	1	15
$x11$	10	0	1	0	0	0	-1	-1	
$x12$	15	0	0	1	0	0	1	0	
$x13$	10	0	0	0	1	0	0	1	10
基底変数	値	z	$x11$	$x12$	$x13$	$x21$	$x22$	$x23$	θ
z									

最適目的関数値 $z=$.

最適解 $x11=$, $x12=$, $x13=$, $x21=$, $x22=$, $x23=$.

#houthakker.run

```
param NPLANT ;       #工場数
param NDEMAND ;  #倉庫数
param c {i in 1..NPLANT, j in 1..NDEMAND}>=0; #輸送コスト
param x {i in 1..NPLANT, j in 1..NDEMAND}>=0; #輸送量
param S {i in 1..NPLANT}>=0; #供給量
param D {j in 1..NDEMAND}>=0; 需要量
param newc {k in 1..NPLANT*NDEMAND}>=0; #安い順に並べた費用
param n_pl {k in 1..NPLANT*NDEMAND};       #並べ替えた費用(k 番目)に対応する工場
param n_dem {k in 1..NPLANT*NDEMAND};    #並べ替えた費用(k 番目)に対応する倉庫
param tot_num; #並べ替えた費用の順番
param tmp; #変数内容入れ替えのためのパラメタ
data tp.dat ; #データファイルの読み込み
let tot_num:=1;
for{i in 1..NPLANT}  #c[i,j] を new_c[tot_num]に入れる、tot_num=1,…, NPLANT*NDEMAND
{
    for{j in 1..NDEMAND}
    {
        let newc[tot_num]:=c[i,j];
        let n_pl[tot_num]:=i;
        let n_dem[tot_num]:=j;
        let tot_num:=tot_num+1;
        if(j = NDEMAND) then #c[1,1],c[1,2],c[1,3]まで new_c[1]から new_c[3]に入れ、次は c[2,1]
        {
            break;
}}}
for{i in 0..NPLANT*NDEMAND-2} # new_c[tot_num] を昇順に並べ替え
{
     for{j in 1..NPLANT*NDEMAND-1-i}
     {
          if(newc[j]>newc[j+1])then
          {
              let tmp:=newc[j]; let newc[j]:=newc[j+1]; let newc[j+1]:=tmp;
              let tmp:=n_pl[j]; let n_pl[j]:=n_pl[j+1]; let n_pl[j+1]:=tmp;
              let tmp:=n_dem[j]; let n_dem[j]:=n_dem[j+1]; let n_dem[j+1]:=tmp;
}}}
let tot_num:=1;
repeat while((sum{i in 1..NPLANT}S[i]>0) and (sum{j in 1..NDEMAND}D[j]>0)
                  and (tot_num <=NPLANT*NDEMAND) ) #供給または需要の未達成量がある限り
{
     if(min(S[n_pl[tot_num]], D[n_dem[tot_num]]) >0)then 供給または需要の未達成量の小さい値
     {#min(a,b)は a と b の小さい方を与える
       let (空白 輸送量 x の更新 );
       let (空白 供給未達成量 S の更新 );
       let (空白 需要未達成量 D の更新 );
     }
    let tot_num:= tot_num+1;
}
```

```
display x;
exit;
```

2. 輸送問題の AMPL による定式化：モデルファイルとデータファイルの分離
次のような 3 つのファイルを作業フォルダ AMPLWORK (例えば C:¥amplcml¥amplwork)に作成する。
tp.mod ファイル：輸送問題を記述したファイル
tp.dat ファイル：輸送問題のデータを記述したファイル
tp.run ファイル： AMPL のコマンドスクリプトを記述したファイル
以下に内容を記す。ただし「#」以降はコメントである。
ファイル tp.mod の内容

```
param NPLANT ;          #工場数定義
param NDEMAND ;     #倉庫数定義
param c {i in 1..NPLANT, j in 1..NDEMAND}>=0; #輸送費用
param S {i in 1..NPLANT}>=0;                    #供給量
param D {j in 1..NDEMAND}>=0;                   #需要量
var x {i in 1..NPLANT, j in 1..NDEMAND} >=0;   #変数：輸送量
minimize totalcost: sum {i in 1..NPLANT, j in 1..NDEMAND} c[i,j]*x[i,j];     #目的関数：輸送費用
subject to supply {i in 1..NPLANT}: sum {j in 1..NDEMAND}x[i,j] = S[i];     #供給量制約
subject to demand {j in 1..NDEMAND}: sum {i in 1..NPLANT}x[i,j]   = D[j];   #需要制約
```

ファイル tp.dat の内容

```
param NPLANT := 2;          #工場数設定
param NDEMAND :=3;          #倉庫数設定
param c :=                  #添字を有するパラメタの記述は自由度が高い。テキストを参照。
[1,*] 1 6 2 5 3 4
[2,*] 1 8 2 9 3 5 ;
param S := 1 35 2 15 ;
param D := 1 25 2 15 3 10 ;
```

ファイル tp.run の内容

```
model tp.mod ;
data tp.dat ;
option solver cplexamp; #ソルバに CPLEX を指定、これがないと MINOS が動く
option display_round 6; #小数点以下 6 桁表示
solve; #問題の求解
expand > tp.sol; #ファイルに定式化を展開して表示
display totalcost > tp.sol;
display supply.dual > tp.sol; #供給制約の双対変数
display demand.dual > tp.sol; #需要制約の双対変数
display x > tp.sol;
display _ampl_time >tp.sol;
display _total_solve_time >tp.sol;
exit;
```

3. 施設配置問題の AMPL による定式化

関東地方を中心に営業を行ってきた輸入品販売業のある会社は、関西地方に活動を拡大するため、京阪神地方に倉庫の賃借を行う計画を立てている。賃借の候補となる倉庫は m カ所にあって、第 i 地点の倉庫 (i=1,･･･,m)の月間処理能力は a_i（トン/月）で、その経費（賃借料や維持費など毎月の固定費用）は d_i（千円/月）である。また、関西一円に広がる消費地 j (j=1,･･･,n)での輸入品の需要量 b_j（トン/月）と、倉庫 i から需要地 j へのトン当たり輸送費 c_{ij}(千円)が与えられている。すべての需要を満たし、毎月の総費用（倉庫経費＋輸送費）を最小にする倉庫配置と輸送計画を求めたい。この問題を数理計画問題として定式化せよ。

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{m} d_i y_i$$

$$\text{subject to} \sum_{i=1}^{m} x_{ij} \geq b_j,\ j = 1,\ldots,n$$

$$\sum_{j=1}^{n} x_{ij} \leq a_i y_i,\ i = 1,\ldots,m$$

$$x_{ij} \geq 0,\ i = 1,\ldots,m,\ j = 1,\ldots,n$$

$$y_i \in \{0,1\},\ i = 1,\ldots,m$$

```
#fl.mod;
param NPLANT ;
param NDEMAND ;
var y {i in 1..NPLANT, j in 1..NDEMAND} >=0;
var x {i in 1..NPLANT} binary;
param varcost {i in 1..NPLANT, j in 1..NDEMAND}>=0;
param fixcost {i in 1..NPLANT}>=0;
param caplimit {i in 1..NPLANT}>=0;
param demand {j in 1..NDEMAND}>=0;
minimize cost:
sum {i in 1..NPLANT}fixcost[i]*x[i]
+sum {i in 1..NPLANT, j in 1..NDEMAND}
varcost[i,j]*y[i,j];
subject to capacity {i in 1..NPLANT}:
  sum {j in 1..NDEMAND}y[i,j] <= caplimit[i]*x[i];
subject to demconst {j in 1..NDEMAND}:
 sum {i in 1..NPLANT}y[i,j]  = demand[j];
```

```
#fl.dat
param NPLANT := 2;
param NDEMAND :=3;
param varcost :=
[1,*] 1 72 2 9 3 34
[2,*] 1 85 2 78 3 79 ;
param fixcost :=
1 1500 2 1400 ;
param caplimit :=
1 96 2 74 ;
param demand :=
1 10 2 28 3 21 ;
```

```
#fl.run;
model fl.mod ;
data fl.dat ;
option solver cplexamp;
option display_round 6;
solve;
display cost > fl.sol;
display sum {i in 1..NPLANT}fixcost[i]*x[i] > fl.sol;
display sum {i in 1..NPLANT, j in 1..NDEMAND} varcost[i,j]*y[i,j] >
fl.sol;
display x > fl.sol;
display y > fl.sol;
quit;
```

4. 集合被覆問題の応用

あるコンサートをアリーナで開催することになった。そのコンサートは熱狂的なファンが殺到することで知られており、アリーナのコンサートも事故防止のために厳重な警備が必要と考えられる。アリーナでは、どこに警備員を配置したらよいかを検討することにした。会場担当者は、会場全体を小さなブロック（たとえば、ブロック 1,2,…, m）に分け、仮に警備員をある候補地点（たとえば、候補地点 1,2,…,n）に配置したときに警備員がどのブロックを同時にみることができるかを調べた。（例：m=5, n=5 の場合、地点 1 よりブロック 1,2 を監視可能）この情報をもとに、会場全体を最小人数の警備員で警備するための配置とそのときの警備要員の必要人数を求めたい。

候補地点	1	2	3	4	5
ブロック1	○		○		
ブロック2	○	○		○	○
ブロック3		○			○
ブロック4			○	○	
ブロック5					○

問題データ：　配置候補地点（1,2,…,n)、警備すべきブロック(1,2,…,m)；ブロック i が候補地点 j から監視可能なら a_{ij}=1（さもなくば 0）とする制約係数行列 A=［a_{ij}］

変数（＝列）の定義：　x_j＝警備員を地点 j に配置するとき 1、さもなくば 0

目的関数：（警備員を配置する地点数最小）

制約条件：（各ブロックは監視可能）

```
#sc.mod
param NBLOCK ;
param NPOINT ;
param a {i in 1..NBLOCK, j in 1..NPOINT} binary;
var x {j in 1..NPOINT} binary;
minimize totalcost:  sum {j in 1..NPOINT} x[j];
subject to watch {i in 1..NBLOCK}:
sum {j in 1..NPOINT}a[i,j]*x[j] >= 1;
```

```
#sc.dat
param NPOINT := 5;
param NBLOCK := 5;
param a :=
1 1 1  1 2 0  1 3 1  1 4 0  1 5 0
2 1 1  2 2 1  2 3 0  2 4 1  2 5 1
3 1 0  3 2 1  3 3 0  3 4 0  3 5 1
4 1 0  4 2 0  4 3 1  4 4 1  4 5 0
5 1 0  5 2 0  5 3 0  5 4 0  5 5 1;
```

```
#sc.run
model sc.mod ;
data sc.dat ;
option solver cplexamp;
option display_round 6;
solve;
expand > sc.sol;
display totalcost > sc.sol;
display x > sc.sol;
display _ampl_time >sc.sol;
display _total_solve_time >sc.sol;
exit;
```

5. 発電機起動停止問題

確率計画法に基づく起動停止問題モデルを以下の問題（UC）に示す。台数Iの発電機による電力供給を考える。変数u_{it}は発電機iの時間tにおける状態を表す 0-1 変数である。変数x_{it}は発電機iの時間tの出力である。関数$f(x_{it})$は発電機iの燃料費を表すx_{it}の凸 2 次関数である。関数$g_i\,(u_{i\,t-1},u_{it})$は発電機iの起動費用を表し、$(u_{i\,t-1},u_{it})=(0,1)$の時に正の起動費用となり、それ以外の場合には 0 となる関数である。

（UC）：$\min \quad \sum_{t=1}^{T} g_i(u_{i,t-1},u_{it}) + \sum_{i=1}^{I}\sum_{t=1}^{T} f_i(x_{it})u_{it}$,

subject to $\sum_{i=1}^{I} x_{it} \geq d_t, \forall$ t

$u_{it} - u_{i,t-1}, \leq$u,

τ =t+1,…,min{t+$L_i - 1$,T}, $\forall i,t$ =2,…,T

$u_{i,t-1} - u_{it} \leq 1 - u_{it}$,

τ =t+1,…,min{t+$l_i - 1$,T}, $\forall i,t$ =2,…,T

$q_i\,u_{it} \leq x_{it} \leq, Q_i\,u_{it}, \quad u_{it} \in \{0,1\}, \forall i, \forall t,$

目的関数は、供給コストの最小化である。供給コストは燃料費の全てのシナリオに対する期待値と起動費用の総和となる。第 1 制約は、出力の総和が電力需要を満たすための条件である。第 2 制約は、発電機iは一旦起動したらL_i時間連続で運転しなければならないことを表す。同様に第 3 制約は、発電機iは一旦停止したらl_i時間連続で停止しなければならないことを表す。第 4 制約は発電機の出力の上下限を与える。Q_i, q_iはそれぞれ発電機iの出力の上限値、下限値である。

```
#uc.mod
param N; #設備数
param T; #時間数
param a{i in 1..N};
param b{i in 1..N};
param c{i in 1..N};
param d{t in 1..T};
param g{i in 1..N};
param xmin{i in 1..N};
param xmax{i in 1..N};
param uptime{i in 1..N};
param downtime{i in 1..N};

var x{i in 1..N, t in 1..T} >=0;
var u{i in 1..N, t in 0..T} binary; #時間 t における状態を表す
var v{i in 1..N, t in 1..T} >=0;    #時間 t に起動するかどうかを表す
```

```
minimize cost: sum{i in 1..N, t in 1..T}(a[i]*x[i,t]*x[i,t]+b[i]*x[i,t])
+sum{i in 1..N, t in 1..T}(c[i]*u[i,t])
+sum{i in 1..N, t in 1..T}g[i]*v[i,t];

subject to demand{t in 1..T}: sum{i in 1..N}x[i,t] >= d[t];
subject to minoutput{i in 1..N, t in 1..T}: xmin[i]*u[i,t] <= x[i,t];
subject to maxoutput{i in 1..N, t in 1..T}: x[i,t] <= xmax[i]*u[i,t];

subject to minup{i in 1..N, t in 2..T, s in t+1..min(t+uptime[i]-1,T)}:
                                u[i,t]-u[i,t-1] <= u[i,s];
subject to mindown{i in 1..N, t in 2..T, s in t+1..min(t+downtime[i]-1,T)}:
                                u[i,t-1]-u[i,t] <=1- u[i,s];
subject to uv{i in 1..N, t in 1..T}: u[i,t]-u[i,t-1] <= v[i,t];
subject to init_u {i in 1..N}: u[i,0]=0;
```

最適化・シミュレーション演習 第 3 回 動的計画法

1. Knapsack Problem

次のナップサック問題（KP）を考える。品物 1,2,3 の値段はそれぞれ 7,8,3(万円)、重さはそれぞれ 3,4,2(kg)である。重量制限 6 kgのナップサックに入る品物のうち、最も値段の和が大きくなるような組合せを求めよ。

決定変数は以下のように定義される。

x_j =0, 品物 j をナップサックに入れる

x_j =1, 品物 j をナップサックに入れない

(KP) 最大化 $z = 7x_1 + 8x_2 + 3x_3$

制約 $3x_1 + 4x_2 + 2x_3 \leqq 6, \quad x_j \in \{0,1\}, \quad \forall\ j$

最適解は $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$ 最適目的関数値 $z = 11$

次のような 3 つのファイルを作業フォルダに作成する。

knapsack.mod ファイル：ナップサック問題の連続緩和問題を記述したファイル
knapsack.dat ファイル：ナップサック問題のデータを記述したファイル
knapsack.run ファイル： AMPL のコマンドを記述したファイル

以下に内容を記す。ただし「#」以降はコメントである。

ファイル knapsack.mod の内容

```
param Nitem;                #品物数を Nitem と定義
param weight{i in 1..Nitem};  #各品物に重さ（weight）を定義
param value{i in 1..Nitem};   #各品物に価値（value）を定義
param l{i in 1..Nitem};       #0-1 制約の緩和制約における下限
param u{i in 1..Nitem};       #0-1 制約の緩和制約における上限
param capacity;              #ナップサックの容量
var x{i in 1..Nitem} binary;           #各品物（入れる/入れない）に対応する変数
maximize Profit: sum{i in 1..Nitem}value[i]*x[i]; #目的関数 Profit
subject to Capacity_Constraint:  sum{i in 1..Nitem}weight[i]*x[i] <= capacity;
                                  #ナップサックの容量制約
subject to Upper_Lower_Bound {i in 1..Nitem}:  l[i] <= x[i] <=u[i];
                                    #0-1 制約を緩和した上下限
```

ファイル knapsack-relax.dat の内容

```
param Nitem:= 3;
param value :=
1 10
2 6
3 2 ;
param weight :=
1 4
2 3
3 2 ;
param capacity:=
5;
param l :=
1 0
2 0
```

```
3 0;
param u :=
1 1
2 1
3 1
;
```

ファイル knapsack.run の内容

```
model knapsack.mod ;
data knapsack.dat ;
option display_round 6; #小数点以下6桁表示
option solver cplex;     #ソルバにCPLEXを指定、これがないとMINOSが動く
solve;                   #問題の求解
expand > knapsack.sol;          #ファイル(knapsack.sol)に定式化表示
display Profit > knapsack.sol;   #ファイル(knapsack.sol)に目的関数値
display x > knapsack.sol;        #ファイル(knapsack.sol)に最適解
```

この問題を動的計画法により解く。

最適性の原理を表す再帰方程式： $f_k(\lambda) = \max \{ f_{k-1}(\lambda),\ c_k + f_{k-1}(\lambda - a_k) \}$

上の式を漸化式と考えて、最適解を求める。
ただし、$f_k(\lambda)$ を $f_k(\lambda)=\max \{\Sigma_{j=1}^{k} c_j x_j \mid \Sigma_{j=1}^{k} a_j x_j \leqq \lambda, x_j \in \{0,1\}, j=1,...,k\}$ と定義する。
また初期設定として次の値を用いる。: $f_0(\lambda)=0$ ($\lambda \geqq 0$ のとき), $f_0(\lambda)=-\infty$ ($\lambda<0$ のとき)
$f_k(\lambda)=0$ ($\lambda=0$ のとき), $f_k(\lambda)=-\infty$ ($\lambda<0$ のとき)

以下に動的計画法のプログラムを示す。

```
#knapsack-dp.run
param Nitem;
param value{i in 1..Nitem};
param weight{i in 1..Nitem};
param capacity;
param l{i in 1..Nitem};
param u{i in 1..Nitem};
param f{k in 0..Nitem, la in -capacity..capacity}; #関数 f_k(λ)を定義、λをインデックスla
data knapsack.dat;
option display_transpose -20; #行列表示パラメータ
for{k in 0..Nitem, la in -capacity..-1}
{
        let f[k,la]:=-1000;     #初期設定、実行不可能な場合を除外するため十分に小さく
}
for{la in 0..capacity}
{
        let f[0,la]:=0;         #初期設定、変数kの個数が0の場合
}
for{k in 1..Nitem}
{
        let f[k,0]:=0;          #初期設定、容量λの値が0の場合
}
```

```
#以下はDPの本体部分
for{k in 1..Nitem} #変数個数k=1,…,Nitemの値まで反復
{
        for{la in 1..capacity} #λ=1,…,capacityの値まで反復
        {
                if(                                  )then # f_{k-1}(λ)≧ c_k + f_{k-1}(λ−a_k)
                 {
                        let                                 # f_k(λ)=f_{k-1}(λ)
                 }
                if(                                 )then  # f_{k-1}(λ)< c_k + f_{k-1}(λ−a_k)
                 {
                        let                                 # f_k(λ)= c_k + f_{k-1}(λ−a_k)
                 }
        }
}
display f > knapsack-dp.sol;
```

結果は以下の通りである。

```
#knapsack-dp.sol
f [*,*]
:    -6      -5      -4      -3      -2      -1      0   1   2   3   4   5   6 :=
0   -1000   -1000   -1000   -1000   -1000   -1000   0   0   0   0   0   0   0
1   -1000   -1000   -1000   -1000   -1000   -1000   0   0   0   7   7   7   7
2   -1000   -1000   -1000   -1000   -1000   -1000   0   0   0   7   8   8   8
3   -1000   -1000   -1000   -1000   -1000   -1000   0   0   3   7   8   10  11
;
```

#下線部以外は初期設定の値による。

2. ロットサイズ決定問題

T 期間のロットサイズ決定問題を考える。決定変数として t 期における生産量と t 期末の在庫量をそれぞれ、x_t, I_t と定義する。また、t 期における需要、製造固定費、単位製品あたりの製造変動費、期末在庫維持費をそれぞれ d_t, a_t, v_t, h_t とする。次の表で与えられる数値を用いて、3 期間のロットサイズ決定問題を考える。

期 t	1 月	2 月	3 月
需要　d_t,	10	10	10
固定費　a_t	100	120	100
変動費　v_t,	20	30	10
期末在庫費　h_t	8	2	(3)　3 月期末在庫は 0 なので考えない

以下に定式化を示す。

最小化　$z=\Sigma_{t=1,\ldots,T}(a_t y_t+v_t x_t+h_t I_t)$

制約　$I_{t-1}+x_t-d_t=I_t$,　$t=1,\ldots,T$　（流量保存）

$x_t \leqq M y_t$　ただし M は十分大きい正数、$M=\Sigma d$

$I_t \geqq 0$（品切れ不許可），　$x_t \geqq 0$,　$\forall t$

$I_0=0$,　$I_T=0$

AMPL のモデルファイルは以下の通りである。

```
#ww.mod
param term; #計画期間数 T はインデックスで用いることの多い t と混同しないようにした。
param a{i in 1..term};
param v{i in 1..term};
param h{i in 1..term};
param d{i in 1..term};
var y{i in 1..term} binary;
var x{i in 1..term} >=0;
var inv{i in 0..term} >=0; #在庫量 I はインデックスの i と混同しないよう inv とした。
minimize cost: sum {i in 1..term} (a[i]*y[i]+v[i]*x[i]+h[i]*inv[i]);
s.t. c1 {i in 1..term} :inv[i-1]+x[i]-d[i]=inv[i];
s.t. c2 :inv[0]=0;
s.t. c3 {i in 1..term} :x[i]<=30*y[i]; #大きい正数　M=Σd
```

#ww.dat

param term := 3;

param a:=

1 100　2 120　3 100;

param d:=

1 10　2 10　3 10;

param v:=

1 20　2 30　310;

param h:=

1 8　2 2　3 3;

この問題を解くための動的計画法の再帰方程式は次のように表される。

再帰方程式：$f_t = \min_{k=t+1,\ldots,T+1} \{ c_{t,k-1} + f_k \}$

ただし、f_t と $c_{t,k-1}$ を次のように定義する。

f_t＝t－1 期の期末在庫量 I_{t-1}＝0 のときに、t 期以降の最小費用

$c_{t,k-1}$ =t 期の生産量を= $d_t + d_{t+1} + \cdots + d_{k-1}$ としたときの t 期から k－1 期までの総費用

<u>解答：</u>

f_4＝0 と定める。

f_3＝100＋10×10＋f_4＝200* (3 月に 3 月分生産)

f_2＝min{2 月分のみ生産,　2－3 月分同時生産 }

＝min{120＋30×10＋f_3, 120＋30×20＋2×10＋f_4 }

＝min{620*, 740}＝620　（2 月分のみ生産）

f_1＝min{1 月分のみ生産, 1-2 月分同時生産, 1-3 月分同時生産}

＝min{100＋20×10＋f_2, 100＋20×20＋8×10＋f_3, 100＋20×30＋8×20＋2×10＋f_4 }

＝min{920, 780*, 880}＝780　（1 月に 1－2 月分生産）

これより、最適解は次のようになる。総費用は 780 である。

x_1,＝20, I_1＝10, x_3,＝10

```
option solver cplex;
model ww.mod;
data ww.dat;
solve;
display cost;
display x, y, inv;

param f{i in 1..term+1};
param t;
param c{i in 1..term, j in 1..term};

let t:=term;
let f[term +1]:=0;
for{i in 1..term}
{
     let f[i]:=10000;
}
repeat while (t>0) #t 期から後の最適費用が f[t]なので計算は t>0 で行う
{
    display t >ww.sol;
    for{k in t+1..term+1}
    {
       let c[t,k-1]:=0;
       let c[t,k-1]:=c[t,k-1]+a[t]+v[t]*sum{j in t..k-1}d[j]; #固定費と変動費
       for{l in t..k-2}
       {
           let c[t,k-1]:=c[t,k-1]+h[l]*sum{j in l+1..k-1}d[j]; #在庫費用
       }

       print 'k=',k, 'c[t,k-1]=' ,c[t,k-1], 'f[k]=',f[k]  >ww.sol;
       if (                                    )then
       {
            let                                  ;                  #f[t]の更新
       }
    }
    printf:"********************** ¥n"   >ww.sol;
    let t:=t-1;
}
display f >ww.sol;
display f;

quit;
```

```
model knapsack-relax.mod ;
data knapsack-relax.dat ;
option display_round 6;              #小数点以下 6 桁表示
option solver cplex;                 #ソルバを CPLEX に指定


param maxsubproblems;                #子問題の最大生成数
param size_activenode;               #未探索頂点(子問題)の個数

param parent {k in 1..maxsubproblems};                    #子問題 k の親問題番号
param var_branching {k in 1..maxsubproblems} ;            #子問題 k において選択された分枝変数
param fix_binary {k in 1..maxsubproblems} binary;         #子問題 k における分枝変数の値(0-1)
param ACTIVE_NODELIST {k in 1..size_activenode+2};        #未探索頂点(子問題)のリスト
param total_node_number;                                  #生成された頂点(子問題)の数

param epsilon;                                            #整数変数が 0-1 条件を満たしているか判定係数
param ancestor;                                           #各子問題に対応する頂点の先祖

param tmpprofit;                                          #暫定目的関数値
param tmpx {i in 1..Nitem} >=0;                           #暫定解の値
```

```
###### 初期設定 #####
        let maxsubproblems :=1000;
        let total_node_number :=1;
        let size_activenode :=total_node_number;
        let ACTIVE_NODELIST[size_activenode] :=total_node_number;

        let epsilon :=1e-5;
        let tmpprofit := -1e10;
```

repeat while (size_activenode >0 and total_node_number < maxsubproblems)　#分枝限定法の主要部分
{

```
##### 未分枝頂点、生成された問題情報などの表示 #####
printf: "********************************* ¥n" >bb-knapsack.sol;
printf: "Total number of subproblems %d ¥n", total_node_number >bb-knapsack.sol;
printf: "Current Problem %d ¥n", ACTIVE_NODELIST[size_activenode]   >bb-knapsack.sol;
for{i in 1..size_activenode}
{
        printf: "ACTIVE_NODELIST[%d]=%d ¥n", i, ACTIVE_NODELIST[i] > bb-knapsack.sol;
}
display size_activenode > bb-knapsack.sol;
```

```
###### 当該問題のデータ作成 #####
let ancestor :=ACTIVE_NODELIST[size_activenode] ; #未分枝頂点リストの最終要素を選択
for{i in 1..Nitem}                                    #0-1 変数の制限を初期化
{
        let l[i]:=0;
        let u[i]:=1;
}
repeat while (ancestor >1)          #選択した子問題の親が根でない限り、0-1 変数の固定を行う
{
        let   l[var_branching[ancestor]]:= fix_binary[ancestor];
        let   u[var_branching[ancestor]]:= fix_binary[ancestor];

        let ancestor :=parent[ancestor];   #選択した子問題の先祖ノードを探索
}
```

```
##### 子問題を解き、未分枝頂点リストから除く #####
solve;                                          #問題をソルバで解く
display solve_result >bb-knapsack.sol;
if (solve_result="solved") then
{
    display Profit >bb-knapsack.sol;
    display l, u, x> bb-knapsack.sol;
}
let size_activenode :=size_activenode-1 ;       #未分枝頂点リストの長さを 1 つ減少
```

```
##### 実行可能で非整数解が得られ、限定操作が適用されない場合 子問題へ分枝 #####
if ( (exists {i in 1..Nitem} x[i]*(1-x[i]) >= epsilon) and
    (solve_result="solved") and (Profit > tmpprofit)   ) then
{
        for{ii in 1..Nitem}
        {
            if ( x[ii]*(1-x[ii]) > epsilon ) then
            {
                printf: "INTEGER CONSTRAINT VIOLATED ¥n" >bb-knapsack.sol;
                printf: "Subproblem %d and %d generated ¥n",
                        total_node_number+1, total_node_number+2 >bb-knapsack.sol;

                let parent[total_node_number+1]:=ACTIVE_NODELIST[size_activenode+1];
                let parent[total_node_number+2]:=ACTIVE_NODELIST[size_activenode+1];

                let var_branching[total_node_number+1]:=ii;
                let fix_binary[total_node_number+1]:=1;
                let ACTIVE_NODELIST[size_activenode+1] :=total_node_number+1;

                let var_branching[total_node_number+2]:=ii;
                let fix_binary[total_node_number+2]:=0;
                let ACTIVE_NODELIST[size_activenode+2] :=total_node_number+2;
                let total_node_number:=total_node_number+2;
                let size_activenode :=size_activenode+2 ;
                break;                                  #for 文から脱出
             }
        }
}
```

```
##### 暫定解表示 #####
printf: "TEMPORALLY objective value             %lf ¥n", tmpprofit >bb-knapsack.sol;
if ((forall {i in 1..Nitem} x[i]*(1-x[i]) < epsilon) and
   (solve_result="solved") and (Profit > tmpprofit)   ) then
{
      let tmpprofit:=Profit;
      let {i in 1..Nitem} tmpx[i]:=x[i];
      printf: "TEMPORALLY objective value refined %lf ¥n", tmpprofit >bb-knapsack.sol;
}
```

} #分枝限定法の主要部分(終わり)

```
##### 最終結果表示 #####
printf: "########################################## ¥n" >bb-knapsack.sol;
printf: "Branch and Bound FINISHED ¥n"   >bb-knapsack.sol;
display tmpprofit > bb-knapsack.sol;
display tmpx > bb-knapsack.sol;
```

```
quit;
```

最適化・シミュレーション演習 第5回 列生成法

1. 計算実験の方法

フォルダ CG 全体をコピーして別の名前をつける。たとえばCG-4(製品数)など。CG-40 で以下の実験を行う。

1. データ作成：data-const.run を実行してデータを作成する。全列挙の計算時間は allpat.dat に示される。
2. 全列挙問題： allpat.run を実行し、列を全列挙してできた問題を直接混合整数計画として解く。計算時間は allpat.sol に示される。
3. 列生成問題：cut.run パターンを加えながら解く。計算時間は cut.sol に示される。

注意： data-const.run において param nproduct := 20; によって製品数が変わる。
data-const.run において
param small_ratio :=15;　短い製品は材料全体の 1/15
param big_ratio :=2;　　長い製品は材料全体の 1/2
なので、これも変えてみることが望ましい。

2. データ作成方法の詳細

2.1 data-const.run の前半部：基本的データ、素材長さ、製品種類、製品長さなどの作成

```
#data-const.run;
param roll_width := 100;          # width of raw rolls
param nproduct:= 4;               # 製品種類数
param small_ratio :=10;           # 最小製品長さ:全体の 1/small_ratio
param big_ratio :=3;              # 最大製品長さ:全体の 1/big_ratio
param llength :=floor(roll_width/small_ratio);
param ulength :=floor(roll_width/big_ratio);;
param ldemand :=80;  #需要量範囲
param udemand :=100; #需要量範囲
param product_length {i in 1..nproduct};
param product_demand {i in 1..nproduct};
param one_pattern_number {i in 1..nproduct};
param ii;
param jj;
param aa;
param bb;
param number_use {i in 1..nproduct};
param number_feasiblepattern ;
param dupl{i in llength..ulength}; #製品長さの重複判定
for{i in llength..ulength}
{
   let dupl[i]:=0;
}
printf: "param nproduct := %d;¥n", nproduct >allpat.dat;
printf: "param roll_width := %d;¥n", roll_width >allpat.dat;
printf: "param: WIDTHS orders :=¥n" >allpat.dat;

printf: "param nproduct := %d;¥n", nproduct >cut.dat;
printf: "param roll_width := %d;¥n", roll_width >cut.dat;
printf: "param: WIDTHS orders :=¥n" >cut.dat;

let ii :=1;
repeat while (ii <=nproduct)
{
   let product_length[ii]:=floor(Uniform(llength,ulength));
   let product_demand[ii]:=floor(Uniform(ldemand,udemand));
   let dupl[product_length[ii]]:=dupl[product_length[ii]]+1;
   if(dupl[product_length[ii]]=1)then
   {
      let ii:=ii+1;
   }
}
```

```
#以下２行は例題用のデータ
let product_length[1]:=45; let product_length[2]:=36; let product_length[3]:=31; let
product_length[4]:=14;
let product_demand[1]:=97;let product_demand[2]:=610; let product_demand[3]:=395; let
product_demand[4]:=211;
#例題用のデータ：ここまで

#製品を長さの昇順にソート
for{i in 1..nproduct-1}
{
   for{j in 1..nproduct-i}
   {
       if(product_length[j]>product_length[j+1])then
        {
             let aa:=product_length[j];
             let product_length[j] :=product_length[j+1];
             let product_length[j+1]:=aa;
             let bb:=product_demand[j];
             let product_demand[j] :=product_demand[j+1];
             let product_demand[j+1]:=bb;
        }
    }
}
for{i in 1..nproduct}
{
      printf: "%d %d %d¥n", i, product_length[i], product_demand[i] >allpat.dat;
      printf: "%d %d %d¥n", i, product_length[i], product_demand[i] >cut.dat;
}
printf: ";¥n" >allpat.dat;
printf: ";¥n" >cut.dat;
printf: "param nbr:=¥n" >allpat.dat;
# ここまで基本データ作成
```

これより、cut.dat; が作成される。

```
#cut.dat はパターンデータを含んでいない。
param nproduct := 4;
param roll_width := 100;
param: WIDTHS orders :=
1 14 211
2 31 395
3 36 610
4 45 97
;
```

2.2 data-const.run の中盤部：単一製品のみのパターンだけで解く場合との比較

```
#単一製品のみのパターンだけで解くとどうなるか (無駄が多い)。
    for{i in 1..nproduct}
    {
      let one_pattern_number[i]:= ceil(product_demand[i]/floor(roll_width/product_length[i]));
      printf: "製品%d を%d=floor(%lf)枚含む素材を %d 枚使用 ¥n",
      i, floor(roll_width/product_length[i]), roll_width/product_length[i], one_pattern_number[i]
            >one_pattern_opt.dat;
    }
    display sum{i in 1..nproduct}one_pattern_number[i] >one_pattern_opt.dat;
```

これより、ファイル one_pattern_opt.dat;が作成される。

2.3 data-const.run の後半部：すべてのパターンの列挙

#全ての切り出しパターンを列挙して解くとどうなるか（時間がかかる)。

```
    for{i in 1..nproduct}
    {
       let number_use[i]:=0;
    }
       let ii :=1;
       let number_feasiblepattern:=0;
       let number_use[ii]:=number_use[ii]+1;
    repeat while (number_use[nproduct]<= floor(roll_width/product_length[nproduct]) )#最大長さ製品≦最大数
    {
       if(sum{i in 1..nproduct} product_length[i]*number_use[i] <=roll_width)then  #実行可能製品が
                                                                                    #得られている場合
       {
           let number_feasiblepattern:=number_feasiblepattern+1;
           printf:"[%d,*]  ", number_feasiblepattern >allpat.dat;
           for {i in 1..nproduct}
           {
               printf:"%d %d ",i, number_use[i] >allpat.dat;                    #各製品の枚数を記述
           }
           printf:"¥n" >allpat.dat;
           let ii :=1;                                                          #製品 ii よりスタート
           let number_use[ii]:=number_use[ii]+1;                                #製品 ii を 1 枚増やす
           repeat while( number_use[ii] >floor(roll_width/product_length[ii]) ) #製品 ii の切り取り
                                                                                #不可能な場合
           {
              let number_use[ii]:=0;                                            #製品 ii の枚数を 0 として
              let number_use[ii+1]:=number_use[ii+1]+1;                         #製品 ii+1 を 1 枚増やす
              let ii:=ii+1;
           }
       }
       if(sum{i in 1..nproduct} product_length[i]*number_use[i] > roll_width)then #実行可能製品でない場合
       {
            let number_use[ii+1]:=number_use[ii+1]+1; #製品 ii+1 を 1 つ増やし、
                                                      #それ以下の長さの製品数をリセット
            for{j in 1..ii}
            {
                let number_use[j]:=0;
            }
            let ii:=ii+1;
        }
    }
           printf:";¥n" >allpat.dat;

           printf:"param nPAT=%d ; ¥n  ", number_feasiblepattern >allpat.dat;
           printf:"#TIME=%lf ; ¥n  ", _ampl_time >allpat.dat;
```

これより、cut.dat; が作成される。

```
#allpat.dat すべてのパターンを列挙
param nproduct := 4;
param roll_width := 100;
param: WIDTHS orders :=
1 14 211
途中省略
4 45 97
;
param nbr:=
[1,*]  1 1 2 0 3 0 4 0
途中省略
[37,*]  1 0 2 0 3 0 4 2
;
param nPAT=37 ;
```

3. 全列挙法のモデル

```
#allpat.mod;
# ---------------------------------------
# CUTTING STOCK USING PATTERNS
# ---------------------------------------

param roll_width > 0;         # width of raw rolls
param nproduct;
param WIDTHS {i in 1..nproduct};                  # set of widths to be cut
param orders {i in 1..nproduct} > 0;    # number of each width to be cut

param nPAT integer >= 0;      # number of patterns
param nbr {i in 1..nPAT, j in 1..nproduct} integer >= 0;

var Cut {i in 1..nPAT} integer >= 0;   # rolls cut using each pattern

minimize Number:                  # minimize total raw rolls cut
   sum {i in 1..nPAT} Cut[i];

subject to Fill {j in 1..nproduct}:
   sum {i in 1..nPAT} nbr[i,j] * Cut[i] >= orders[j];
```

以下は実行コマンドスクリプトの内容を示す。

```
#allpat.run
model allpat.mod;
data allpat.dat;
#option solver cplex, solution_round 6;
option cplex_options $cplex_options 'mipcuts=-1';

solve;
       display Cut >allpat.sol;
       printf:"#TIME=%lf ; ¥n  ", _ampl_time >allpat.sol;
       printf:"#TIME=%lf ; ¥n  ", _total_solve_time >allpat.sol;
```

4. 列生成法のモデル

以下のファイル cut.mod では、前半部にこれまでに生成された列パターンのみを用いた LP 緩和問題のモデルファイルを含み、後半部は列生成問題に相当する。このように一つのモデルファイルに 2 つの問題の定式化を含むことに注意されたい。

```
#cut.mod
# ----------------------------------------
# CUTTING STOCK USING PATTERNS
# ----------------------------------------

param roll_width > 0;          # width of raw rolls

param nproduct;
param WIDTHS{i in 1..nproduct};                    # set of widths to be cut
param orders {i in 1..nproduct} > 0;    # number of each width to be cut

param nPAT integer >= 0;       # number of patterns
#set PATTERNS = 1..nPAT;       # set of patterns

param nbr {i in 1..nproduct, j in 1..nPAT} integer >= 0;

   check {j in 1..nPAT}:
      sum {i in 1..nproduct} WIDTHS[i] * nbr[i,j] <= roll_width;

                           # defn of patterns: nbr[i,j] = number
                           # of rolls of width i in pattern j

var Cut {j in 1..nPAT} integer >= 0;   # rolls cut using each pattern

minimize Number:                   # minimize total raw rolls cut
   sum {j in 1..nPAT} Cut[j];

subject to Fill {i in 1..nproduct}:
   sum {j in 1..nPAT} nbr[i,j] * Cut[j] >= orders[i];

                                   # for each width, total
                                   # rolls cut meets total orders

# ----------------------------------------
# KNAPSACK SUBPROBLEM FOR CUTTING STOCK
# ----------------------------------------

param price {j in 1..nproduct} default 0.0;

var Use {j in 1..nproduct} integer >= 0;

minimize Reduced_Cost:
   1 - sum {i in 1..nproduct} price[i] * Use[i];

subject to Width_Limit:
   sum {i in 1..nproduct} WIDTHS[i] * Use[i] <= roll_width;
```

5. 列生成法の解法コマンドスクリプト

```
#cut.run
model cut.mod;   #モデルファイル指定
data cut.dat;    #データファイル指定
option solver cplexamp, solution_round 6;
option display_1col 0, display_transpose -10;
option display_width 30;

problem Cutting_Opt: Cut, Number, Fill;  #既存カットのみで解く問題Cutting_Opt を定義
                                         #モデルファイルにおける変数、目的関数、制約の Cut, Number, Fill を
使用
option relax_integrality 1;              #問題 Cutting_Opt ではLP 緩和問題を解く

problem Pattern_Gen: Use, Reduced_Cost, Width_Limit; #列生成問題 Pattern_Gen を定義
                        #モデルファイルにおける変数、目的関数、制約の Use, Reduced_Cost, Width_Limit を使用
option relax_integrality 0;              #列生成では整数計画を解く

let nPAT := 0;
for {i in 1..nproduct} {       #初期パターン生成
   let nPAT := nPAT + 1;
   let nbr[i,nPAT] := floor (roll_width/WIDTHS[i]);    #一つの製品 i のみを含む
   let {i2 in 1..nproduct: i2 <> i} nbr[i2,nPAT] := 0;
}

repeat {
   solve Cutting_Opt;
   let {i in 1..nproduct} price[i] := Fill[i].dual; #列生成問題の目的関数係数 c_B^T B^-1=π

   solve Pattern_Gen;
   if Reduced_Cost < -0.00001 then {
      let nPAT := nPAT + 1;
      let {i in 1..nproduct} nbr[i,nPAT] := Use[i];
   }
   else break; #被約費用が負にならなければ列生成終了
}
display nbr, Cut;

option Cutting_Opt.relax_integrality 0; #列を加え終わった問題では整数計画として解く
solve Cutting_Opt;
display Cut;

       display Cut >cut.sol;
       printf:"#TIME=%lf ; ¥n  ", _ampl_time >cut.sol;
       printf:"#TIME=%lf ; ¥n  ", _total_solve_time >cut.sol;
```

最適化シミュレーション演習 第 6 回

1. Dijkstra 法

以下の Dijkstra 法のアルゴリズムに従って、最短路を求めよ。なお、π_i（現在の反復までに求められた点 i への最短距離）は空欄に記入し、 p_i（現在の反復までに求められた点 i への最短路の直前点）はネットワークの辺に記入せよ。

- ステップ 1 ： $\pi_0=0, \pi_i=\infty$ $(i \in V-\{0\}), M=\{1,2,...,n\}$, $i=0$
- ステップ 2 ： $M\neq\varphi$（空集合）でない場合(i)(ii)(iii)を繰り返す
  (i) $j\in M$ に到達する直前点を可能なら仮に i とする。 $j\in M$ に対し $\pi_j > \pi_i+c_{ij}$ ならば $\pi_j=\pi_i+c_{ij}$, $p_j=i$
  (ii) $j\in M$ への暫定的最短距離から最小の $\min_{j\in M}\pi_j = \pi_k$ となる k を求める。
  (iii) k を M から除く、$i=k$ とする(k からの探索を行う)。

```
#dijnstra-kadai.run
param n;   #ノード数
param k;   #探索開始点
param jj; #先行点を表す

param c{i in 1..n, j in 1..n};       #ij 間距離
param tim{i in 1..n, j in 1..n};    #ij 間時間
param in_number{i in 1..n};            #i への入次数
param out_number{i in 1..n};           #i から出次数
param in_node{i in 1..n, j in 1..n};   #i への入ノード
param out_node{i in 1..n, j in 1..n} ;#i から出ノード
param b{i in 1..n};                          #minimum cost flow としての右辺定数

param pi{i in 1..n};          #点 i までの暫定的最短距離
param prev{i in 1..n};       #点 i への暫定的最短路における直前店
param fix{i in 1..n};        #点 i への最短路確定を示す
param pi_min;                      #暫定的最短距離
param mu;
data dijkstra.dat;


for {i in 1..n}
{
      let pi[i]:=1e10;     #ステップ 1 初期設定 π
      let fix[i]:=0;        #初期設定：全点最短路未確定
}
let pi[1]:=0;
let prev[1]:=0;
let fix[1]:=1;
let k:=1;                    #探索開始点
```

```
repeat while(sum{i in 1..n}fix[i] < n) #すべての点が最短路決定されるまで
                                       #Dijkstra 法全体の反復
{
        #ステップ 2(i)
        for{j in 1..out_number[k]}   #探索開始点 k から接続する点 j すべてに対し
        {
                if(pi[out_node[k,j]]>pi[k]+c[k,out_node[k,j]] and out_node[k,j] != k and fix[out_node[k,j]]=0)then
                {     #点 k から out_node[k,j]への距離が近い場合更新
                      let pi[out_node[k,j]]:=pi[k]+c[k,out_node[k,j]];
                      let prev[out_node[k,j]]:=k;
                }
        }
        #ステップ 2(ii)
        let pi_min:=10000;
        for{j in 1..n}#最短路決定されていない点(fix[j]=0)でπの値が最小のものを選ぶ
        {
                if(pi[j]<pi_min and j != k and fix[j]=0)then
                {
                      let pi_min:=pi[j];
                      let k:=j;
                 }
        }
        #ステップ 2(iii)
        let fix[k]:=1;
} #Dijkstra 法全体の反復
display pi;

for{j in 1..n} #すべての点への最短経路表示
{
    let jj:=j;
    printf:"node %d ",jj;
    repeat while( prev[jj]>0)
    {
          printf:"node %d ",prev[jj];
          let jj :=prev[jj];
     }
     printf:"¥n";
}
```

単純に点 i から n までの最短路を求めたい場合は、fix[n]=1 となった時点で終了すればよい。

課題：費用を $c_{ij}+\mu t_{ij}(\mu=2)$ とした場合の最短路を Djikstra 法で求めるように、run ファイルを修正せよ。

2. Lagrange 緩和法

```
#lag-lp.run
model lag-lp.mod;
data lag-lp.dat;

param theta;  #ステップサイズθ
param Lopt;   #暫定目的関数値
param lambda; #係数λ
param renzoku;#下界値の上昇が起こらない連続回数
param oldL;   #前反復までの下界値
param tmpx{i in 1..n, j in 1..n}; #暫定解

let T:=14;
let Lopt:=24;  #初期暫定値として、24 を与えた。
let mu:=0.0;
let lambda:=0.8;
let renzoku:=0;
let oldL:=-1e10;
printf:" k,  mu,       time-T,   L(mu),  lambda,  theta ¥n" > lag-lp.sol;
for{iter in 0..200} #ラグランジュ緩和法の反復回数
{
    solve;

    if((sum{i in 1..n, j in 1..out_number[i]}tim[i,out_node[i,j]]*x[i,out_node[i,j]] <=T)
    and (sum{i in 1..n, j in 1..out_number[i]}c[i,out_node[i,j]]*x[i,out_node[i,j]] < Lopt))then
    {    #実行可能（時間制約）かつ目的関数値が暫定値より良い場合、暫定値更新
        let Lopt :=  (sum{i in 1..n, j in 1..out_number[i]}
                      c[i,out_node[i,j]]*x[i,out_node[i,j]]);
        for{i in 1..n}
        {
            for{j in 1..n}
            {
                let tmpx[i,j]:=x[i,j];    #暫定解更新
            }
        }
    }


    if(distance <= oldL)then     #前反復までの下界値が更新されない場合
    {
        let renzoku:=renzoku+1; #下界値が更新されない回数を増加
        if(renzoku>=4)then      #下界値が更新されない回数が 4 回連続なら
        {
             let renzoku:=0;
             let lambda:=lambda*1/2; #λを 1/2 倍
        }
    }
```

```
    else   #前反復までの下界値が更新される場合
    {
         let oldL:=distance;
         let renzoku:=0;
    }


    if(sum{i in 1..n, j in 1..out_number[i]}tim[i,out_node[i,j]]*x[i,out_node[i,j]] <> T)then
    #最適性チェック成立しない場合
    {
         let theta:= lambda*(Lopt-distance)
        /(sum{i in 1..n, j in 1..out_number[i]}tim[i,out_node[i,j]]*x[i,out_node[i,j]]-T)**2;
         #ステップサイズθを決定
    }
    else
    {
         break;
    }

    printf:"%3d, %6.6f, %3d, %10.6f, %8.6f, %8.6f¥n",
     iter, mu, (sum{i in 1..n, j in 1..out_number[i]}tim[i,out_node[i,j]]*x[i,out_node[i,j]]-T),
     distance,  lambda, theta > lag-lp.sol;

    let mu:=mu+theta*(sum{i in 1..n, j in 1..out_number[i]}
                      tim[i,out_node[i,j]]*x[i,out_node[i,j]]-T);
    #ラグランジュ乗数μの更新

}

display tmpx > lag-lp.sol;
display sum{i in 1..n, j in 1..out_number[i]}c[i,out_node[i,j]]*tmpx[i,out_node[i,j]] >
lag-lp.sol;
```

課題:*T*=12 のときラグランジュ緩和法により問題を解け。得られた解は最適解であるかどうか検討せよ。

最適化シミュレーション演習 第 7 回

1. Benders 分解法で使用するモデルファイル

使用するモデルファイル fl-master.mod では 2 つの問題を含んでいることに注意せよ。

```
#problem MASTER; #マスター問題
param NPLANT ;
param NDEMAND ;
param alpha {k in 1..NCUT, i in 1..NPLANT};     #最適性カットの x 係数
param beta {k in 1..NCUT};                      #最適性カットの定数部
param alpha1 {k in 1..NfeaCUT, i in 1..NPLANT}; #省略
param beta1 {k in 1..NfeaCUT};                  #省略
param xx {i in 1..NPLANT};                      #マスター問題で得られた x の値
param fixcost {i in 1..NPLANT}>=0;
param demand {j in 1..NDEMAND}>=0;
param caplimit {i in 1..NPLANT}>=0;
param l {i in 1..NPLANT};
param u {i in 1..NPLANT};
var x {i in 1..NPLANT} >=0, integer;            #変数 x および θ
var theta >=0;

minimize costmaster:     #MASTER 問題では Σfx+θ を最小化
sum {i in 1..NPLANT}fixcost[i]*x[i] +theta;

subject to luz {i in 1..NPLANT}:        #変数 x の上下限制約
 l[i] <= x[i] <= u[i];

subject to feascut {k in 1..NfeaCUT}:   #省略：実行可能性カット
0>=sum{i in 1..NPLANT}alpha1[k,i]*x[i] +beta1[k];

subject to optcut {k in 1..NCUT}:       #最適性カット、最初は NCUT=0 なので制約の本数=0
theta >= sum{i in 1..NPLANT}alpha[k,i]*x[i] +beta[k];

subject to feasibility: #Q(x)を定義する LP が必ず実行可能になるように追加した制約
 sum {i in 1..NPLANT}caplimit[i]*x[i]  >= sum {j in 1..NDEMAND}demand[j];

#problem SECOND;          #Q(x)を求める問題
var y {i in 1..NPLANT, j in 1..NDEMAND}>=0;    #輸送量 y が変数
param varcost {i in 1..NPLANT, j in 1..NDEMAND}>=0;

minimize secondcost:          #輸送費用の最小化
sum {i in 1..NPLANT, j in 1..NDEMAND} varcost[i,j]*y[i,j];

subject to capacity {i in 1..NPLANT}:    #輸送量≦容量となる制約 (MASTER 問題の解は xx)
 -sum {j in 1..NDEMAND}y[i,j] >= -caplimit[i]*xx[i];

subject to dem1 {j in 1..NDEMAND}:       #需要を満足させる制約
 sum {i in 1..NPLANT}y[i,j]  = demand[j];
```

2. Benders 分解法のアルゴリズム

```
     option presolve 0; #前処理を行わない

param NCUT;      #最適性カット本数=0
param NfeaCUT;   #省略：実行可能性カット本数=0
let NCUT:=0;
let NfeaCUT:=0;
model fl-master.mod ;
data fl.dat ;
option solver cplexamp;
problem MASTER: x, theta, costmaster, luz, feascut, optcut, feasibility; #前頁 MASTER
problem SECOND: y, secondcost, capacity, dem1;                           #前頁 Q(x)
problem DIRECTION: mu, pi, directgen, direct_mupi, direct_mu, direct_pi; #省略

param UB; #暫定目的関数値
param LB; #下界値
let UB:=1e12;
let LB:=0;

display UB, LB;


repeat
{
     printf: "¥n######## NCUT=%d ¥n", NCUT;
     printf: "¥n######## NCUT=%d ¥n", NCUT > fl-master.sol;
     solve MASTER;           #MASTER 問題の求解
     for {i in 1..NPLANT}
     {
          let xx[i]:=x[i];   #MASTER 問題の解を xx
     }

     if (costmaster>LB)then    #下界値の更新
     { let LB:=costmaster;}

     solve SECOND;

     if ( solve_result="infeasible") then    #以下本ページ内は省略：実行可能性カット
     {
          solve DIRECTION;

          let NfeaCUT:=NfeaCUT+1;
          for {i in 1..NPLANT}
          {
               let alpha1[NfeaCUT,i]:=caplimit[i]*mu[i];
          }
          let beta1[NfeaCUT]:=sum{j in 1..NDEMAND}demand[j]*pi[j];
     }
```

```
     else if ( solve_result="solved" and theta < secondcost) then  #最適性カット生成
     {
          let NCUT:=NCUT+1;                    #カット本数追加
          for {i in 1..NPLANT}
          {
               let alpha[NCUT,i]:=-caplimit[i]*capacity[i].dual;        #係数μ定義
          }
          let beta[NCUT]:=sum{j in 1..NDEMAND}(demand[j]*dem1[j].dual); #係数π定義

          if (sum {i in 1..NPLANT}fixcost[i]*x[i]+secondcost <UB)then   #上界の更新
          { let UB:=sum {i in 1..NPLANT}fixcost[i]*x[i]+secondcost;}
     }
     display x, theta > fl-master.sol;
     display UB, LB;
     if(LB>(1-1e-2)*UB)then   #終了判定条件
     {
          break;
     }
}
     printf: "¥n####### FINISHED ¥n";
     printf: "¥n####### FINIFHED ¥n" > fl-master.sol;

solve MASTER >fl-master.sol;
expand > fl-master.sol;
display costmaster > fl-master.sol;
display x > fl-master.sol;
display y > fl-master.sol;

display _ampl_time >fl-master.sol;
display _total_solve_time >fl-master.sol;
quit;
```

課題：施設配置候補地数や需要地数の異なる問題を解き、(Benders 分解を用いない) 通常の方法と比較せよ。特に、データの与え方（施設の容量と固定費を大きく）を変化すると、どのような結果が得られるか考察せよ。

最終課題：
本演習において取り扱った適当な問題をとりあげるか、または異なる問題をきちんと提示せよ。その上、数理計画モデルとして定式化し、適切なデータを作成（収集）せよ（データの与え方は、これまでに配布した dataconst.run などのプログラムを参考にすること)。本演習で扱った分枝限定法などの解法を用いて、結果を得て、それを評価せよ。

最適化・シミュレーション演習 補足 切除平面法

1. Knapsack Problem

次のナップサック問題（KP）を考える。品物 1,2,3 の値段はそれぞれ 7,8,3(万円)、重さはそれぞれ 3,4,2(kg)である。重量制限 6 kgのナップサックに入る品物のうち、最も値段の和が大きくなるような組合せを求めよ。

決定変数は以下のように定義される。

x_j =0, 品物 j をナップサックに入れる

x_j =1, 品物 j をナップサックに入れない

(KP) 最大化 $z = 7x_1 + 8x_2 + 3x_3$

制約 $3x_1 + 4x_2 + 2x_3 \leqq 6, \quad x_j \in \{0,1\}, \quad \forall\ j$

最適解は $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$ 最適目的関数値 $z = 11$

以下のモデルファイル knapsack-cut.mod では 2 つの問題#Master Problem と#Cut Generation が記述されていることに注意せよ。

```
#knapsack-cut.mod
#Master Problem
param Nitem;
param Ncut;   #妥当不等式の貸す
param weight{i in 1..Nitem};
param value{i in 1..Nitem};
param capacity;
param acut {i in 1..Ncut, j in 1..Nitem}; #妥当不等式係数
param bcut {i in 1..Ncut};                #妥当不等式定数

var x{i in 1..Nitem} binary;
maximize Profit: sum{i in 1..Nitem}value[i]*x[i];
subject to Capacity_Constraint:  sum{i in 1..Nitem}weight[i]*x[i] <= capacity;
subject to Valid {i in 1..Ncut}: sum{j in 1..Nitem} acut[i,j]*x[j] <=bcut[i];

#Cut Generation
param xx{i in 1..Nitem};
var y{i in 1..Nitem} binary;

maximize Violation: sum{i in 1..Nitem}(xx[i]-1)*y[i];
subject to Cover: sum{i in 1..Nitem}weight[i]*y[i] >=capacity+0.001;
```

データは以下の通りである。

```
param Nitem:=3;
param:value weight:=
1 7 3
2 8 4
3 3 2;
param capacity:=6;
```

2. 妥当不等式と分離問題

```
model knapsack-cut.mod ;
data knapsack.dat ;
option display_round 6;
option solver cplexamp;

problem knapsack_master: x, Profit, Capacity_Constraint, Upper_Lower_Bound, Valid;
   option relax_integrality 1; #Master 問題は緩和 LP を解く
problem cut_generaton: y, Violation, Cover;
   option relax_integrality 0; #妥当不等式生成問題は、厳密解を求める。

let Ncut:=0;
repeat
{
   solve knapsack_master;
   expand > knapsack.sol;
   display Profit > knapsack.sol;
   display x > knapsack.sol;
   for{i in 1..Nitem}
   {
        let xx[i]:=x[i];  #緩和 Master 問題の最適解を妥当不等式生成問題の係数とする。
   }
   solve cut_generaton;   #妥当不等式生成問題を解く。
   expand > knapsack.sol;
   display Violation > knapsack.sol;
   display y > knapsack.sol;

   if (Violation > -1) then  #切除平面が生成されるならば
   {
       let Ncut:=Ncut+1;     #妥当不等式の本数を増加。
       for{j in 1..Nitem}    #妥当不等式の係数を定める。
       {
          let  acut[Ncut, j]:=y[j];
       }
          let bcut[Ncut]:=sum{j in 1..Nitem}y[j]-1;
   }
   else
   {
        break;  #切除平面が生成されなければ、repeat 反復より脱出。
   }
}

  option knapsack_master.relax_integrality 0;  #Master 問題の厳密解を求める。
  solve knapsack_master;
```

3. 実行結果

```
C:¥Document¥MyDocument¥最適化シミュレーション演習 2014¥補足 切除平面¥KP-cut>ampl
knapsack-cut.run
AMPL Version 20080423 (x64_win64)
CPLEX 12.4.0.0: optimal solution; objective 13   #Master 緩和問題 (1 回目)
1 dual simplex iterations (0 in phase I)
CPLEX 12.4.0.0: optimal integer solution; objective -0.25  #妥当不等式生成問題 (1 回目)
0 MIP simplex iterations
0 branch-and-bound nodes
CPLEX 12.4.0.0: optimal solution; objective 11  #Master 緩和問題 (2 回目)
1 dual simplex iterations (0 in phase I)
Solution determined by presolve;                #妥当不等式生成問題 (2 回目)
objective Violation = -1.                       シャープカット生成できなかった。
CPLEX 12.4.0.0: optimal integer solution; objective 11 #Master 問題 (厳密解)
0 MIP simplex iterations
0 branch-and-bound nodes
```

以下 sol ファイルの内容を示す。

```
#Master 緩和問題 (1 回目)
maximize Profit:
        7*x[1] + 8*x[2] + 3*x[3];

subject to Capacity_Constraint:
        3*x[1] + 4*x[2] + 2*x[3] <= 6;

subject to Upper_Lower_Bound[1]:
        0 <= x[1] <= 1;

subject to Upper_Lower_Bound[2]:
        0 <= x[2] <= 1;

subject to Upper_Lower_Bound[3]:
        0 <= x[3] <= 1;

Profit = 13.000000

x [*] :=
1  1.000000
2  0.750000
3  0.000000
;
#妥当不等式生成問題 (1 回目)
maximize Violation:
        -0.25*y[2] - y[3];

subject to Cover:
        3*y[1] + 4*y[2] + 2*y[3] >= 6.001;
```

```
Violation = -0.250000

y [*] :=
1  1.000000
2  1.000000
3  0.000000
;
#Master 緩和問題 (2 回目)
maximize Profit:
        7*x[1] + 8*x[2] + 3*x[3];

subject to Capacity_Constraint:
        3*x[1] + 4*x[2] + 2*x[3] <= 6;

subject to Upper_Lower_Bound[1]:
        0 <= x[1] <= 1;

subject to Upper_Lower_Bound[2]:
        0 <= x[2] <= 1;

subject to Upper_Lower_Bound[3]:
        0 <= x[3] <= 1;

subject to Valid[1]:
        x[1] + x[2] <= 1;

Profit = 11.000000

x [*] :=
1  0.000000
2  1.000000
3  1.000000
;
#妥当不等式生成問題 (2 回目)
maximize Violation:
        -y[1];

subject to Cover:
        3*y[1] + 4*y[2] + 2*y[3] >= 6.001;

Violation = -1.000000  #カット生成できなかった。

y [*] :=
1  1.000000
2  1.000000
3  0.000000
;
```

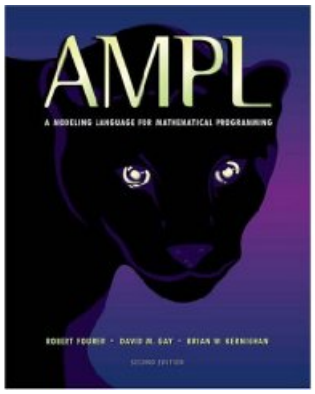
AMPL インストールガイド Win7 用 2014 年版

椎名 孝之

1. はじめに

AMPL (A Modeling Language for Mathematical Programming) は数理計画問題に対するモデリング言語であり、Bell Laboratory で開発された。AMPL では連続変数あるいは整数変数を含む、線形計画問題または非線形計画問題をモデル化し、記述することができる。AMPL で記述された数理計画モデルは、数理計画ソルバー（線形計画問題に対しては BPMPD, CPLEX, LAMPS, LOQO, lp_solve, MINOS, MOSEK, OSL, SOPT, XA, Xpress-MP などのソルバーが対応している）を用いて解くことができる。同時に、上にあげたソルバーに基づくような解法アルゴリズムを実装できることが大きな特徴である。

AMPL のホームページ http://www.ampl.com/ は AMPL に関する総合的な情報源である。AMPL の全容は、解説書「AMPL: A Modeling Language for Mathematical Programming」(by Robert Fourer, David M. Gay, and Brian W. Kernighan, Duxbury Press / Brooks/Cole Publishing Company, 2002. ISBN 0-534-38809-4）に詳しく、http://ampl.com/resources/the-ampl-book/chapter-downloads/ より上記の解説書をダウンロードすることが可能である。



2. AMPL Demo Version のインストール

AMPL Demo Version は http://ampl.com/try-ampl/download-a-demo-version/ より入手することができ、同時に CPLEX, Gurobi や MINOS のようなソルバーもダウンロードできるが、取り扱うことのできる問題の規模に制限がある。

・For linear problems, 500 variables and 500 constraints plus objectives.

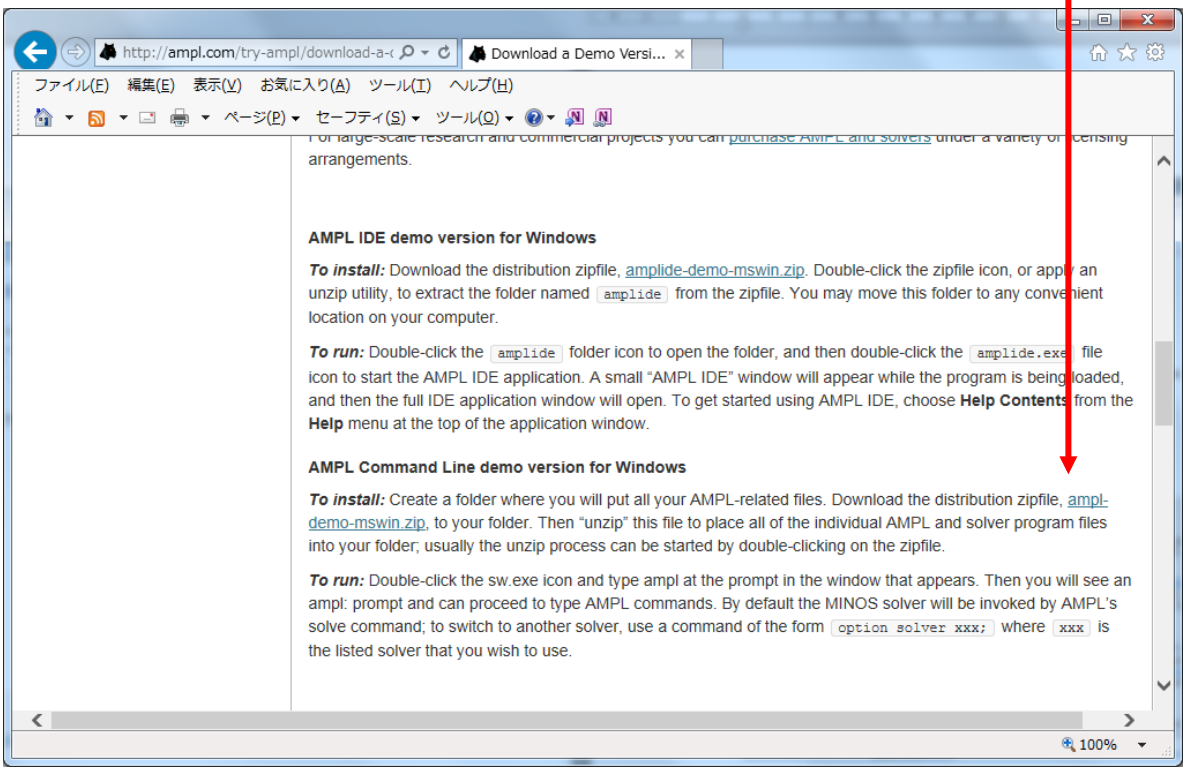
・For nonlinear problems, 300 variables and 300 constraints plus objectives.

以下に手順を示す。

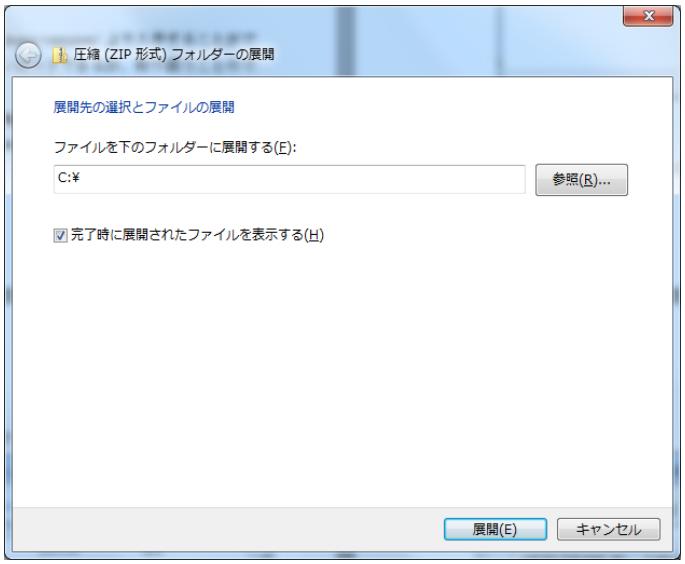
(1) AMPL IDE demo version for Windows のダウンロード

http://ampl.com/try-ampl/download-a-demo-version/

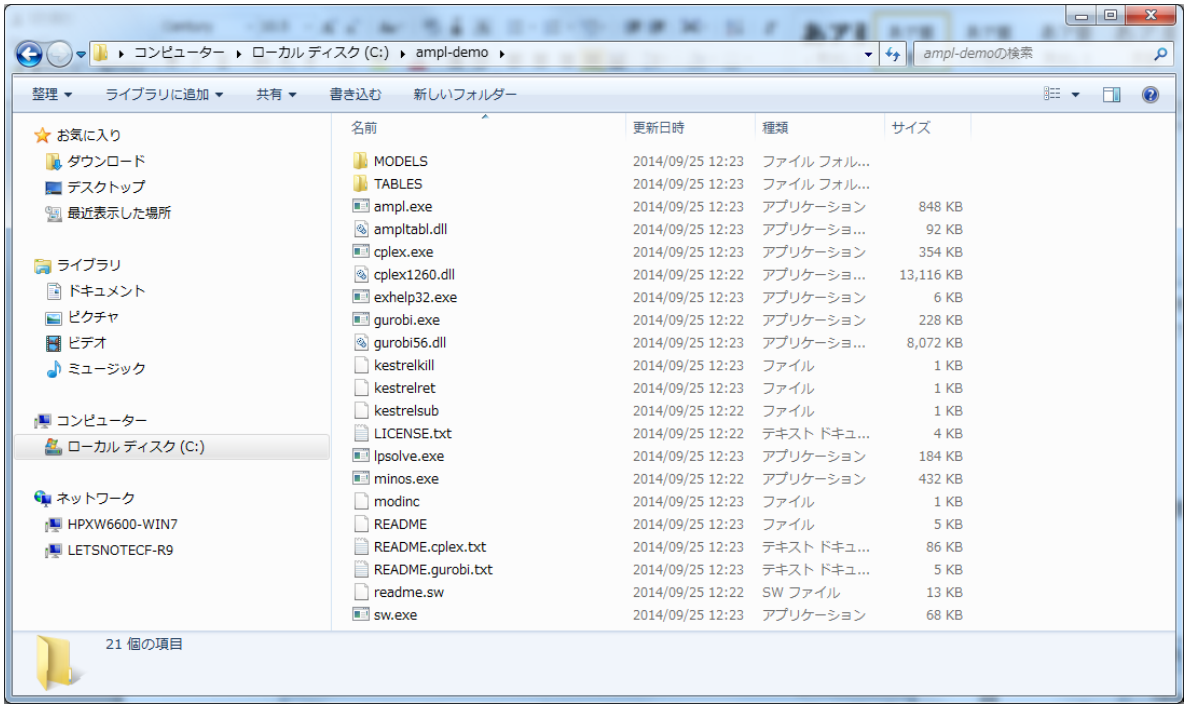
における **AMPL Command Line demo version for Windows** の ampl-demo-mswin.zip を適切なフォルダに保存後、右クリックし、「**開く(O)**」を選択する。



(2)ファイル ampl-demo-mswin.zip をすべて展開し、ファイルをディレクトリ「C:¥」に展開する。このディレクトリでなくても適当なディレクトリを指定してもよい。

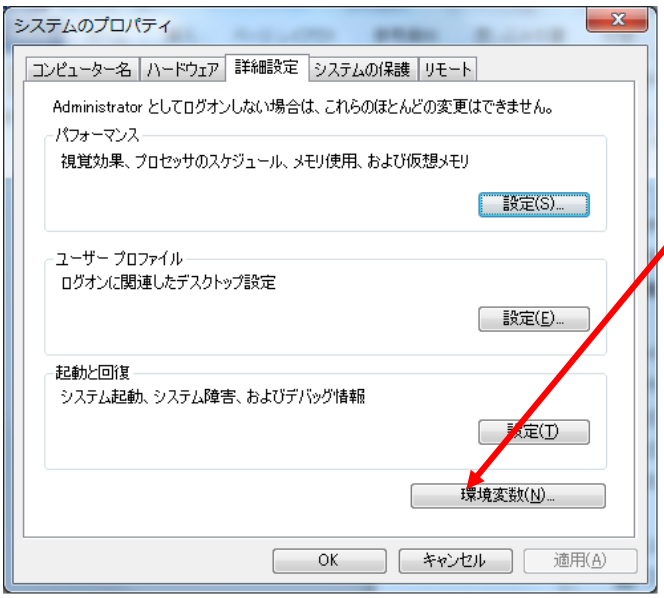


(3)「展開(E)」をクリックすると、C:¥ampl-demo というフォルダが作成された。この中の ampl.exe が AMPL の実行ファイルであり、cplex.exe がソルバ CPLEX の実行ファイルである。

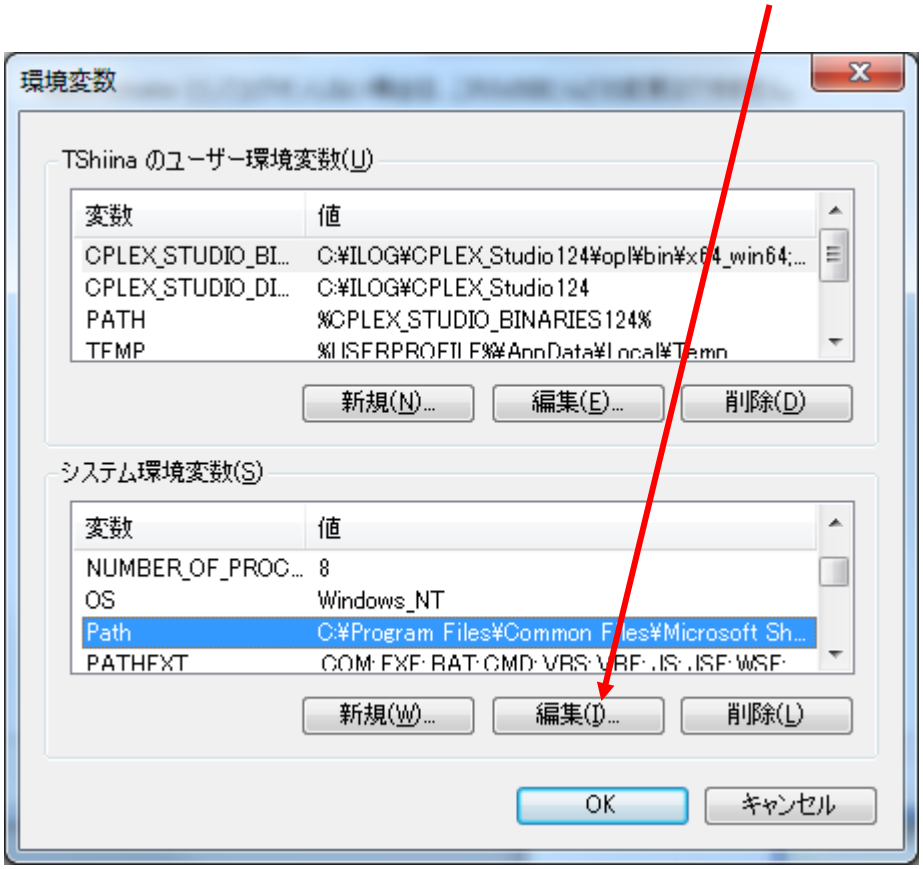


3. パスの設定

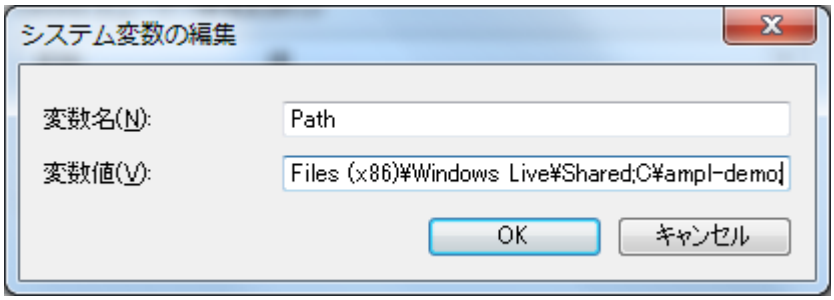
AMPL の実行にはコマンドプロンプトを使用する。注意すべき点は、コマンドプロンプトにおいて実行可能なコマンドはコマンドプロンプトが示すカレントディレクトリのプログラムだけに限られるということである。そのため、環境変数パスの設定を行う。「スタート」→「コントロールパネル」→「システムとセキュリティ」→「システム」をクリックし、「システムの詳細設定」を選択する。「環境変数(N)」をクリックし、環境変数を表示させる。



「システム環境変数(S)」の変数「Path」を選択し、「編集(I)」をクリックする。

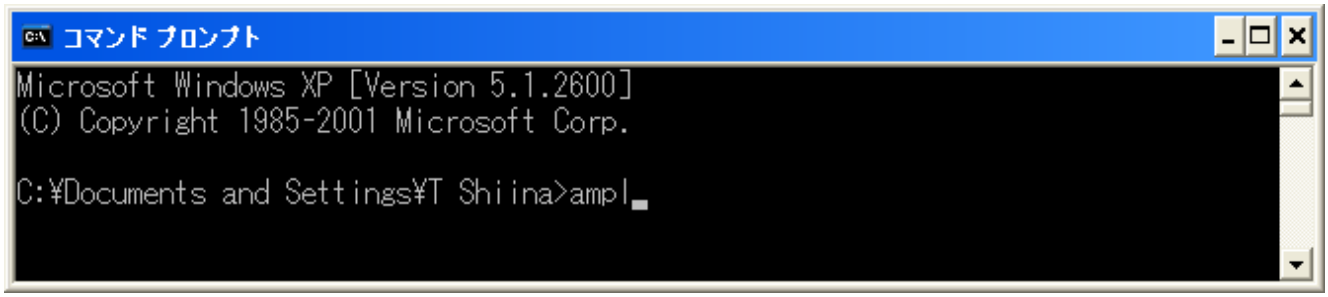


「システム変数の編集」において「変数値(V)」の最後尾に「;C:¥ampl-demo」を追加する（空白は入れない、コマンドの区切がセミコロンであることに注意)。「OK」をクリックし終了。

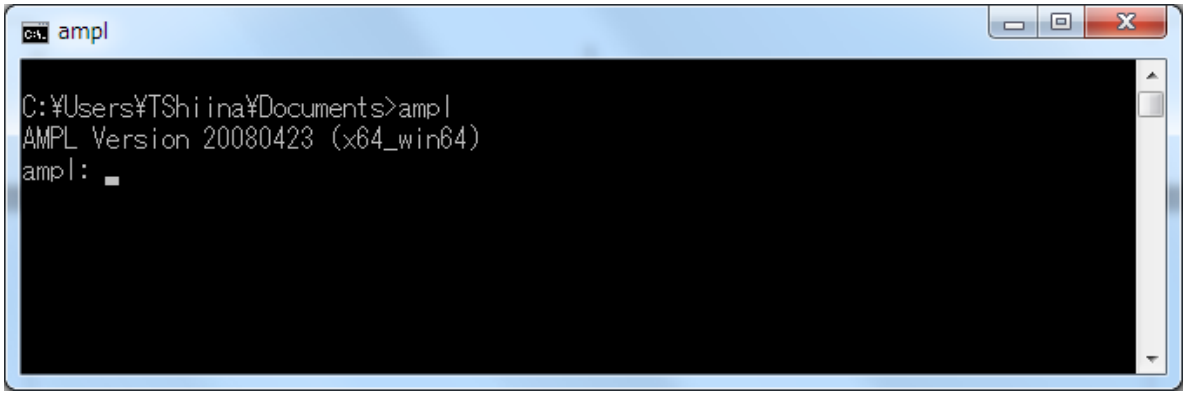


4. ampl の実行

コマンドプロンプトを開き、「ampl」と入力する。



「ampl:」と表示されたらインストールされたことが確認できた。「Ctrl」＋「c」で終了できる。



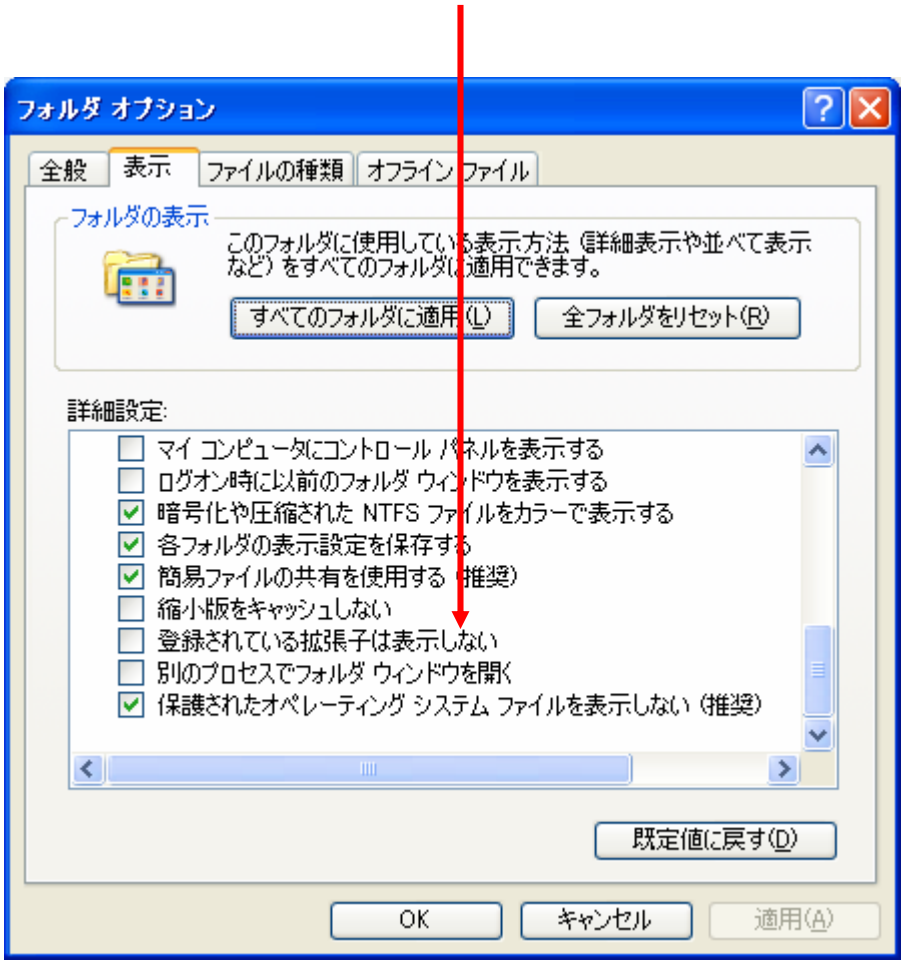
以上

AMPL 例題（ナップサック問題の連続緩和）

椎名 孝之

1. はじめに

AMPL(A Modeling Language for Mathematical Programming)をインストールした amplcml フォルダには、ampl.exe という AMPL の実行ファイルがあることを確認する。また、適当なフォルダを開いて、「ツール」→「オプション」をクリックし、次のフォルダオプションのウインドウを開く。**ここで、「登録されている拡張子は表示しない」のチェックを外す。**元々外してあれば不要です。



このように拡張子を表示することで、AMPL で使用するファイルの識別が容易になる。

AMPL を使用するとき、次のようなファイルの操作を必要とする。

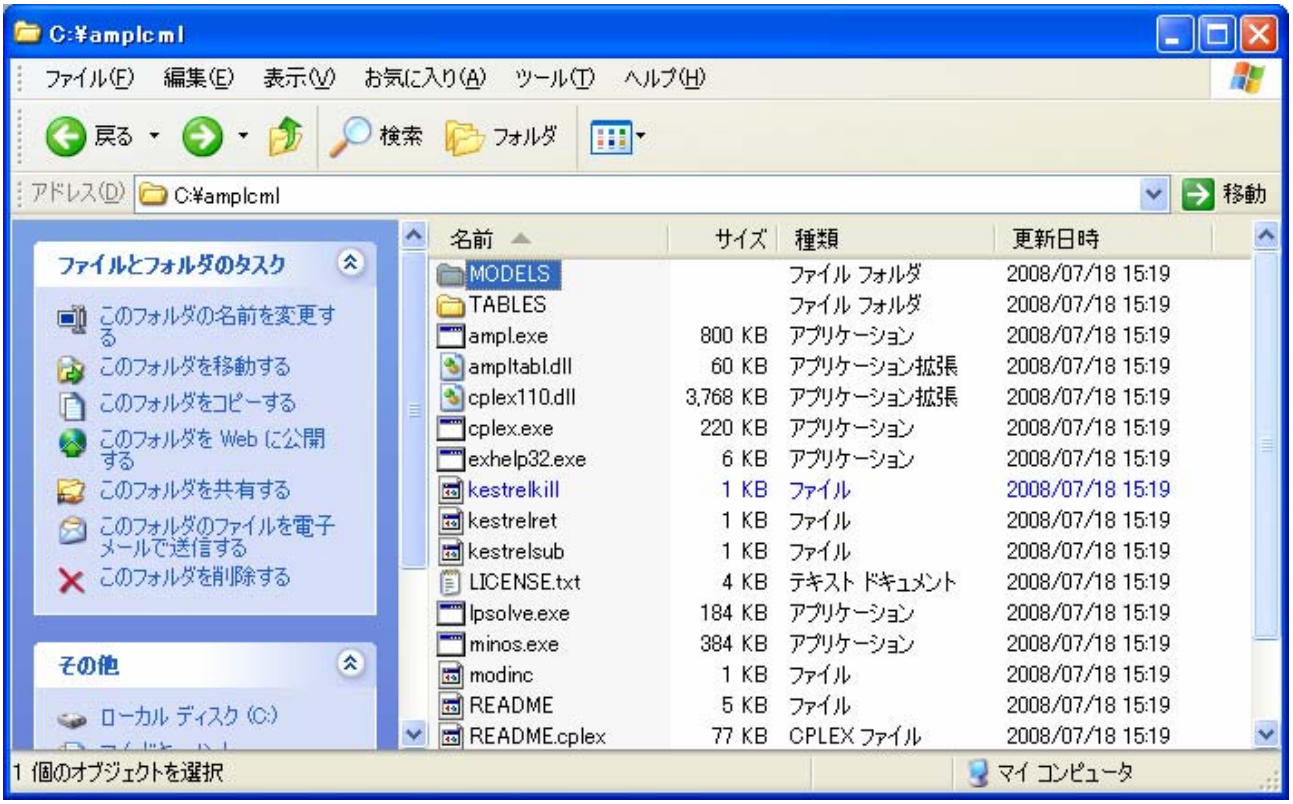
.mod ファイル： 数理計画モデルを記述したファイル

.dat ファイル： 数理計画モデルにおけるデータを記述したファイル

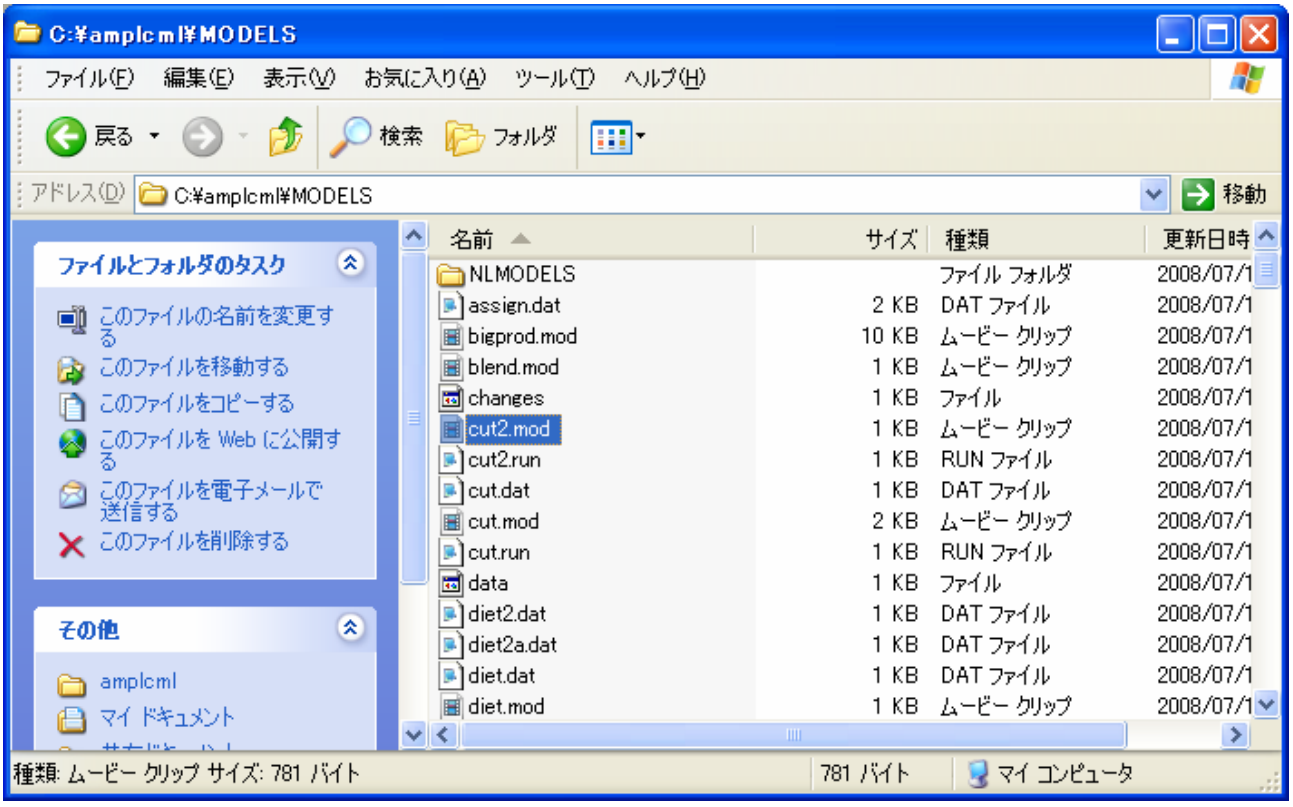
.run ファイル： AMPL のコマンドを記述したファイル

これらのファイルは、メモ帳（notepad）あるいはワードパッドなどのテキストエディタで操作できるようにファイルとの関連付けを行う。AMPL がインストールされたフォルダ C:¥amplcml の中には、MODELS と TABLES の２つのフォルダが存在する（確認されたい）。フォルダ MODELS には、数理計画モデルファイル（.mod）のファイルが保存されている。これらのファイルは AMPL 解説書「AMPL: A Modeling Language for Mathematical Programming」(by Robert Fourer, David M. Gay, and Brian

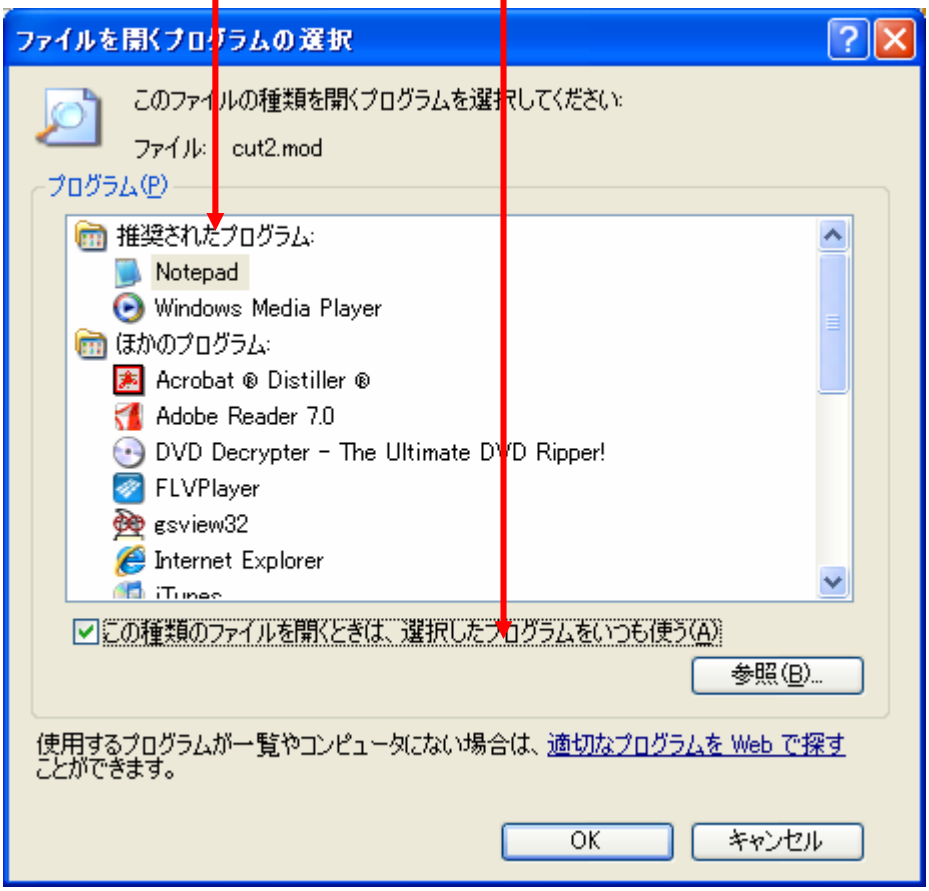
W. Kernighan, Duxbury Press / Brooks/Cole Publishing Company, 2002. ISBN 0-534-38809-4）に掲載されているものと同一である。



フォルダ MODELS の中を見てみよう。



ここには、上に示した以下の .mod ファイル、 .dat ファイル、.run ファイルの多くの例題がある。これらのファイルはAMPLで記述したテキストファイルであるが、WindowsXPなどの標準的な環境では、必ずしもテキストエディタとは関連付けられてはいない。例えば、.mod ファイルはムービークリップとして、Windows media player と関連付けられている（しかし、メディアプレーヤーで.modファイルを開いても当然ながら再生できない)。そのため、適当な.modファイルを選び、右クリックして「プログラムから開く（H)」をクリックし、「プログラムの選択（C)」を左クリックする。「ファイルを開くプログラムの選択」ウインドウにおいて、「推奨されたプログラム」あるいは「ほかのプログラム」の中からNotepadを選択し、「この種類のファイルを開くときは、選択したプログラムをいつも使う(A)」をチェックする。



これで.mod ファイルをメモ帳（Notepad）で開くことが可能になった。同様の操作を繰り返すことによって、.dat ファイルと.run ファイルをメモ帳（Notepad）で開くことができることを確認されたい。

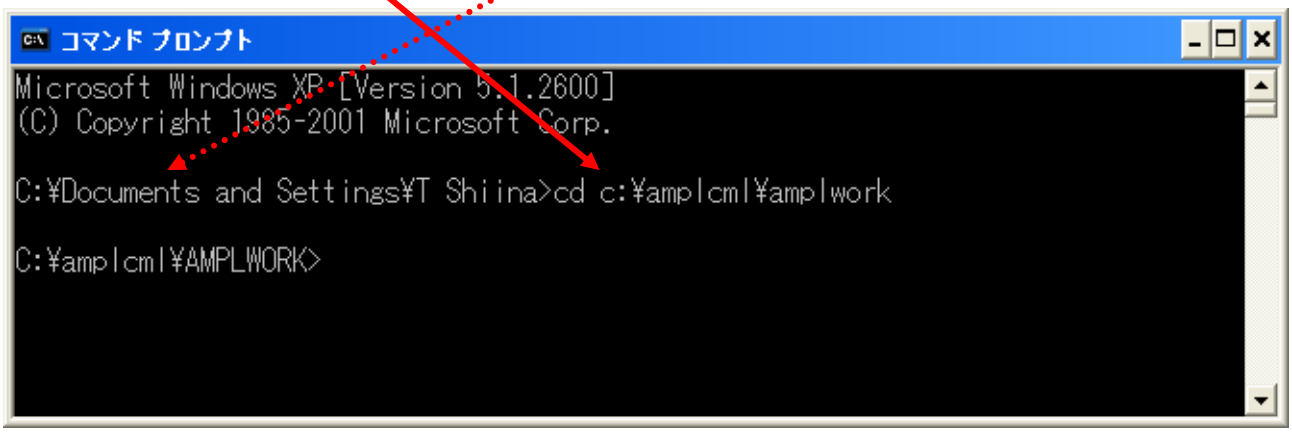
2. 作業フォルダの準備

フォルダ amplcml の下にフォルダ AMPLWORK を作成する。

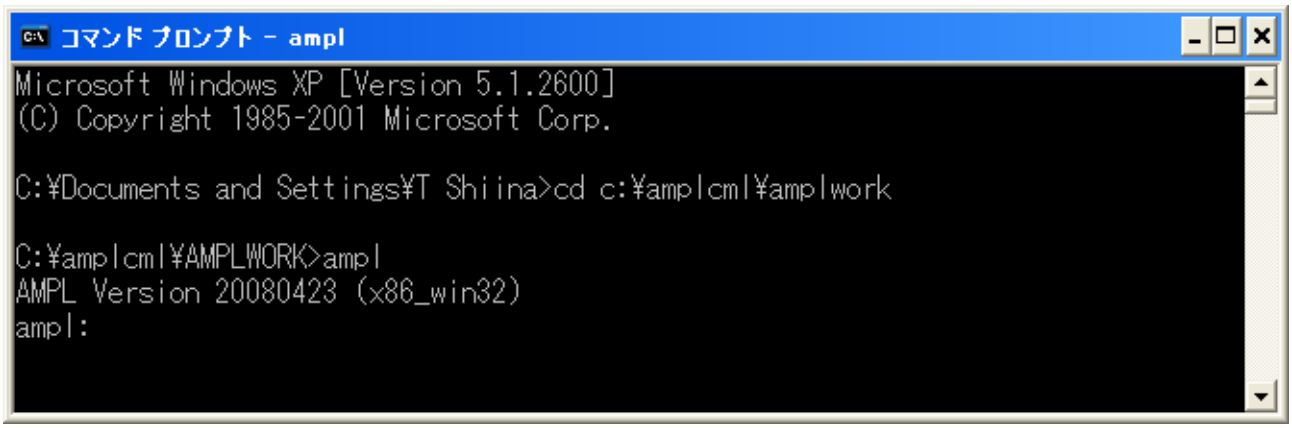
新規作成されたこのフォルダAMPLWORKを、コマンドプロンプトで開く。コマンドプロンプト（スタート→すべてのプログラム→アクセサリ）では通常マイドキュメントのフォルダが開くため、

「cd c:¥amplcml¥amplwork」

で AMPLWORK フォルダに移動する。



このフォルダでも AMPL が起動できることを確認されたい。Ctrl+c で終了できる。



3. 線形計画問題に対する適用例

次のような 3 つのファイルを作業フォルダ AMPLWORK (C:¥amplcml¥amplwork)に作成する。

knapsack-relax.mod ファイル：ナップサック問題の連続緩和問題を記述したファイル

knapsack-relax.dat ファイル：ナップサック問題のデータを記述したファイル

knapsack-.run ファイル： AMPL のコマンドを記述したファイル

以下に内容を記す。ただし「#」以降はコメントである。

ファイル knapsack-relax.mod の内容

```
param Nitem;                    #品物数を Nitem と定義
param weight{i in 1..Nitem};  #各品物に重さ（weight）を定義
param value{i in 1..Nitem};    #各品物に価値（value）を定義
param l{i in 1..Nitem};         #0-1 制約の緩和制約における下限
param u{i in 1..Nitem};         #0-1 制約の緩和制約における上限
param capacity;                  #ナップサックの容量
var x{i in 1..Nitem} >=0;              #各品物（入れる/入れない）に対応する変数
maximize Profit: sum{i in 1..Nitem}value[i]*x[i]; #目的関数 Profit
subject to Capacity_Constraint:   sum{i in 1..Nitem}weight[i]*x[i] <= capacity;
                                    #ナップサックの容量制約
subject to Upper_Lower_Bound {i in 1..Nitem}:   l[i] <= x[i] <=u[i];
                                    #0-1 制約を緩和した上下限
```

ファイル knapsack-relax.dat の内容

```
param Nitem:= 3;
param value :=
1 7
2 8
3 3 ;
param weight :=
1 3
2 4
3 2 ;
param capacity:=
6;
param l :=
1 0
2 0
3 0;
param u :=
1 1
2 1
3 1
;
```

ファイル knapsack-relax.run の内容

```
model knapsack-relax.mod ;
data knapsack-relax.dat ;
option display_round 6; #小数点以下 6 桁表示
option solver cplex;        #ソルバに CPLEX を指定、これがないと MINOS が動く
solve;                      #問題の求解
expand > knapsack-relax.sol;            #ファイル(knapsack-relax.sol)に定式化表示
display Profit > knapsack-relax.sol;    #ファイル(knapsack-relax.sol)に目的関数値
display x > knapsack-relax.sol;         #ファイル(knapsack-relax.sol)に最適解
display Capacity_Constraint.dual > knapsack-relax.sol; #ナップサック制約の双対変数
display Upper_Lower_Bound.dual > knapsack-relax.sol; #0-1 緩和制約の双対変数
```

4. ampl の実行

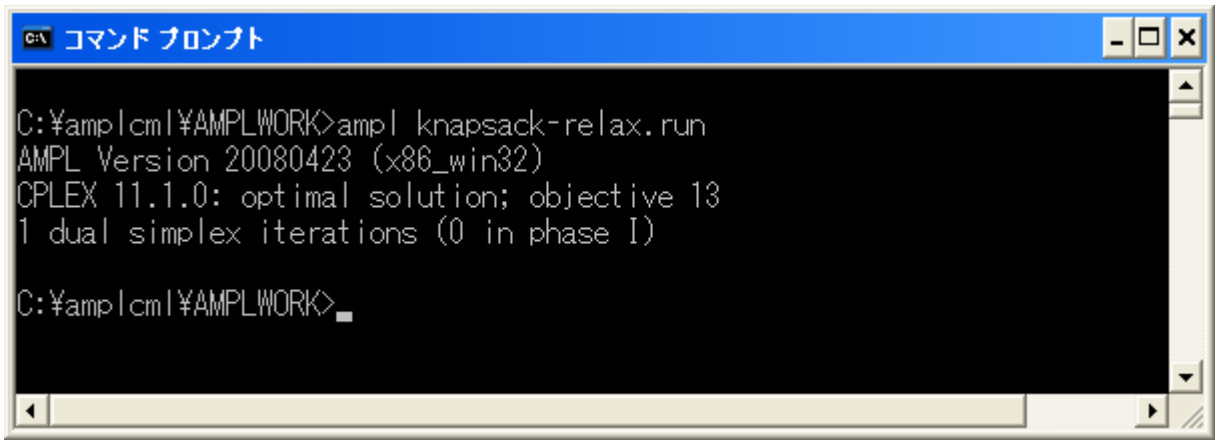
コマンドプロンプトを開き、 amplwork フォルダ（C:¥amplcml¥amplwork)において、

「**ampl knapsack-relax.run**」

と入力する。正常に起動し終了すると、次のように

C:¥amplcml¥AMPLWORK>

と表示され終了する。



5. 実行結果の確認

ファイル knapsack-relax.sol の内容を示す。ただし.sol ファイルについても、テキストエディタから開けるようにプログラムとの関連付けを行っておくことを忘れないように。

```
maximize Profit:
        7*x[1] + 8*x[2] + 3*x[3];

subject to Capacity_Constraint:
        3*x[1] + 4*x[2] + 2*x[3] <= 6;

subject to Upper_Lower_Bound[1]:
        0 <= x[1] <= 1;

subject to Upper_Lower_Bound[2]:
        0 <= x[2] <= 1;

subject to Upper_Lower_Bound[3]:
        0 <= x[3] <= 1;

Profit = 13.000000

x [*] :=
1   1.000000
2   0.750000
3   0.000000
;
Capacity_Constraint.dual = 2.000000

Upper_Lower_Bound.dual [*] :=
1    1.000000
2    0.000000
3    0.000000
;
```

以上