ソルバを用いた数理最適化入門

檀　寛成（関西大学　環境都市工学部）

September 20, 2016
(ver 2.0.0)

目 次

1 ソルバを用いた数理最適化の基礎 9

1.1 数理最適化とは 9
1.2 ソルバを用いた数理最適化のプロセス 9
1.2.1 例 1: 「原料購入 + 輸送」問題 10
1.2.2 例 2: 制約条件の追加 17
1.2.3 例 3: 整数性条件の追加 22
1.2.4 まとめ：3 つの例を通じて 27
1.3 この本の構成 27

2 数理モデルと最適化問題 29

2.1 数理モデルとは 29
2.1.1 対象のモデル化 29
2.1.2 数理モデル 32
2.2 数理最適化：はじめの一歩 34
2.2.1 概念としての最適化問題 34
2.2.2 数理最適化 35
2.2.3 数理モデルとしての最適化問題 36
2.2.4 最適化問題の実行可能性と解の性質 37
2.2.5 最適化問題の分類 39

3 最適化問題の「モデル化」と「定式化」 45

3.1 「モデル化」と「定式化」の差は？ 45
3.2 定式化・基礎 (1): 和を取る 48
3.2.1 添字を使って和を取る 48
3.2.2 0-1 変数を使って必要なものの和を取る 51
3.3 定式化・基礎 (2): 基本的な表現 53
3.3.1 基本パターン 53
3.3.2 集合に関するパターン 55
3.3.3 ナップサック制約に関するパターン 56
3.3.4 その他のパターン 57
3.4 定式化・基礎 (3): 定式化の「デザインパターン」 58
3.4.1 ネットワーク 58
3.4.2 二部グラフ 59
3.4.3 誤差 60
3.5 定式化・応用 (1): 便利な表現 61
3.5.1 最大値の最小化（最小値の最大化） 61

3.5.2 絶対値の最小化 . . . 62
3.5.3 2 通りの絶対値制約 . . . 63
3.5.4 指標関数 . . . 64
3.5.5 Special Ordered Set . . . 64
3.5.6 特殊な状況における曲線の折れ線近似 . . . 66
3.6 定式化・応用 (2):「かゆいところに手が届く」表現 . . . 69
3.6.1 最大値の最大化（最小値の最小化） . . . 69
3.6.2 差の絶対値 . . . 70
3.6.3 複数の 0-1 変数の連動 . . . 71
3.6.4 「または」の表し方 . . . 73

4 モデリング言語による最適化問題の記述 75
4.1 モデリング言語 AMPL の基礎 . . . 75
4.1.1 モデリング言語と最適化ソルバの関係 . . . 75
4.1.2 モデリング言語 AMPL の概要 . . . 75
4.1.3 モデルファイルとデータファイル . . . 76
4.1.4 集合の宣言 . . . 79
4.1.5 変数・パラメータの宣言 . . . 80
4.1.6 目的関数・制約条件の記述 . . . 86

5 ソルバによる求解と解の分析 91
5.1 ソルバによる求解 . . . 91
5.2 得られた解の最適性 . . . 91
5.2.1 連続計画問題に対する最適性の条件 . . . 92
5.2.2 （混合）整数計画問題に対する最適性の条件 . . . 92
5.3 ソルバの出力の確認 . . . 93
5.3.1 最適解が得られているかどうか . . . 93
5.3.2 目的関数の値 . . . 96
5.3.3 変数・制約の値 . . . 96
5.3.4 最適性の条件に関する情報（連続計画問題） . . . 97
5.3.5 最適性の条件に関する情報（(混合）整数計画問題） . . . 98
5.3.6 他のソルバでの出力 . . . 99
5.4 （連続最適化問題に対する）感度分析 . . . 99

6 よく知られた最適化問題 109
6.1 集合分割問題 . . . 109
6.1.1 問題の定義と定式化 . . . 109
6.1.2 モデリング言語による問題記述 . . . 109
6.1.3 データ例と最適解 . . . 110
6.2 集合パッキング問題 . . . 110
6.2.1 問題の定義と定式化 . . . 110
6.2.2 モデリング言語による問題記述 . . . 111
6.2.3 データ例と最適解 . . . 112

6.3 集合被覆問題 112
6.3.1 問題の定義と定式化 112
6.3.2 モデリング言語による問題記述 113
6.3.3 データ例と最適解 113
6.4 ナップサック問題 114
6.4.1 問題の定義と定式化 114
6.4.2 モデリング言語による問題記述 114
6.4.3 データ例と最適解 115
6.5 ビンパッキング問題 115
6.5.1 問題の定義と定式化 115
6.5.2 モデリング言語による問題記述 115
6.5.3 データ例と最適解 116
6.6 最大流問題 116
6.6.1 問題の定義と定式化 116
6.6.2 モデリング言語による問題記述 117
6.6.3 データ例と最適解 118
6.7 最短路問題 119
6.7.1 問題の定義と定式化 119
6.7.2 モデリング言語による問題記述 120
6.7.3 データ例と最適解 121
6.8 Hitchcock 型輸送問題 121
6.8.1 問題の定義と定式化 121
6.8.2 モデリング言語による問題記述 122
6.8.3 データ例と最適解 122
6.9 割当問題 123
6.9.1 問題の定義と定式化 123
6.9.2 モデリング言語による問題記述 123
6.9.3 データ例と最適解 124
6.10 巡回セールスマン問題 125
6.10.1 問題の定義と定式化 125
6.10.2 モデリング言語による問題記述 126
6.10.3 データ例と最適解 127

7 難しい問題をどう解くか 129

7.1 近似解法の利用 129
7.2 大規模問題に対する 2 つのアプローチ 130
7.2.1 大規模問題恐るるに足らず？ 130
7.2.2 大きなモデルを作るのがよいのか？ 131
7.3 制約条件を徐々に厳しくする（巡回セールスマン問題を例に） 133
7.4 制約条件を満たさなくてもよい，でもできるだけ満たしたい 139
7.5 定式化を工夫する 141
7.6 ソルバを用いるのか？用いないのか？ 142

7.6.1 ソルバを使うのが得策ではない問題 . . . 142
7.6.2 最適化問題に見えないが最適化問題として定式化できる問題 . . . 143

8 おわりに代えて 149
8.1 本書で学んだこと . . . 149
8.2 本書を超えるための「次の一歩」 . . . 149

A 数学的準備 151
A.1 本書で用いる数学記号 . . . 151
A.2 1 次関数・非線形関数 . . . 153
A.3 凸性の基礎 . . . 154
A.3.1 凸集合・凸関数 . . . 154
A.3.2 凸結合と凸包 . . . 157
A.3.3 半正定値行列と凸性 . . . 158
A.4 グラフの基礎 . . . 159
A.4.1 グラフの定義 . . . 159
A.4.2 特徴あるグラフ . . . 161

B 最適化問題の分類 (Advanced) 165

C サンプルデータ 169
C.1 集合分割問題のデータ . . . 169
C.2 集合パッキング問題 . . . 170
C.3 集合被覆問題のデータ . . . 171
C.4 ナップサック問題のデータ . . . 171
C.5 ビンパッキング問題のデータ . . . 172
C.6 最大流問題のデータ . . . 173
C.7 最短路問題のデータ . . . 173
C.8 Hitchcock 型輸送問題のデータ . . . 173
C.9 割当問題のデータ . . . 174
C.10 巡回セールスマン問題のデータ . . . 175

D ソルバの入手方法・使い方 179
D.1 GLPK の入手方法・使い方 . . . 179
D.1.1 入手方法 . . . 179
D.1.2 使い方 . . . 181
D.2 GLPK 以外のソルバの紹介 . . . 182

E 改訂予定 185

参考文献 187

索 引 192

まえがき

本稿は「ソルバ」と呼ばれるソフトウェアを用いることを念頭に置いた,「数理最適化」の入門書です.

筆者は，数理最適化（単に「最適化」とも言います）を専門とする仕事に携わり，その後大学で教鞭を執ることになりました．会社員時代には，様々な業種の皆さんと最適化に関する仕事をさせて頂きましたが，最適化という技術のポテンシャルからすると，もっと現場で出番があってよいのではないかとも思っていました．そして大学に異動してからは，最適化の理論的な側面を教える機会に恵まれました．理論的なことを理解するのは非常に大事なことですが，そこで会社員時代に現場の方から聞いた声を思い出しました：

> 「（数理）最適化って，難しいですよね」

そう，現場で必要な数理最適化の知識と，最適化の理論との間には大きなギャップがあるのです．

最適化の理論的な結果はソルバと呼ばれるソフトウェアに詰め込まれていて，我々はソルバを使うことで数理最適化のアルゴリズムを気軽に使うことができます．現場で生じる最適化に関する様々な問題は，このソルバを用いることで解決することができます．しかし，そのようなことはあまり知られていないのが現状でしょう．

この度，所属している学科のカリキュラムの改変に伴って「最適化分析」という科目が設置されることになり，筆者がこの授業を担当することになりました．そしてこの科目を，前述のギャップを埋めるような科目にしようと考えた次第です．しかしながら，そのようなことに適した入門書は私の知る限りではほとんどありません．そこで，本書を自分で書くことにしました．

本書は発展途上の原稿ですので，まだ読みづらい箇所も多くあると思いまし，予期せぬ誤りも含んでいると思います．そのような箇所を見つけたら是非筆者までご連絡ください．

2014 年 9 月

檀 寛成
dan@kansai-u.ac.jp

第1章　ソルバを用いた数理最適化の基礎

本章で学ぶ内容

- 数理最適化とは
- ソルバを用いた数理最適化の概要
  - 最適化問題の定式化
  - モデリング言語による問題の記述
  - ソルバを用いた求解と最適解の分析

1.1　数理最適化とは

「数理最適化」の世界へようこそ！

この本では,「**ソルバ**」とよばれるソフトウェアを用いた**数理最適化**について説明していきます．この本を読み始める時点で読者のみなさんに求める知識はそれほど多くはありません[1]．数理最適化の基本から応用の入り口までを，できるだけ丁寧に説明していきたいと思います．

この章では，ソルバを用いた数理最適化の基礎について説明していきます．

まず，**最適化問題**とは，以下のような特徴を持つ問題のことをいいます．

最適化問題の特徴

- 考えている問題に最大または最小にしたい**目的**がある
- （多くの場合,）考えている問題に守るべき**制約**がある

つまり，最適化問題とは，守るべき制約の下で目的を最良（最大または最小）にする状態を求める問題である，といえます．そしてこの最適な状態のことを **最適解** といいます．

1.2　ソルバを用いた数理最適化のプロセス

最適化問題を解くためには，問題の性質に応じたアルゴリズムを選択し，それを適用する必要があります．とはいっても，数学の問題を解くときのように「紙と鉛筆を準備して…」ということではあ

[1] この本を読み進める上で必要となる数学的事項については付録 A にまとめました．必要に応じて参考にしてください．

りません．最適化問題で考慮する構成要素の数が多くなると，手作業で問題を解くのは現実的ではありません．その代わりに，**ソルバ**とよばれるソフトウェアを用いて問題を解くことができます．最近のコンピュータの処理能力の向上はめざましく，またソルバの性能も一昔前に比べて格段に向上しています（7.2.1 節の **Note** も参考にしてください）．そのため，かなり大規模な問題であってもソルバで最適化問題の解を求めることができるようになっています．

ここでは，前節で紹介した最適化問題を，ソルバを用いて実際に解いてみることで，ソルバを用いた数理最適化のプロセスがどのようなものであるかを紹介したいと思います．数理最適化のプロセスは，以下のように構成されます：

数理最適化のプロセス

- **Step 1:** 実際の問題を最適化問題として定式化する
- **Step 2:** 「ソルバ」と呼ばれるソフトウエアを用い，最適解を得る
- **Step 3:** 最適解を分析し，実際の問題を解決する

もし Step 3 での分析の結果，何らかの誤りがあったことがわかれば，Step 1 に戻って問題の定式化やデータなどを修正し，Step 2 での求解，Step 3 での最適解の分析を繰り返すことになります（図 1.1）．

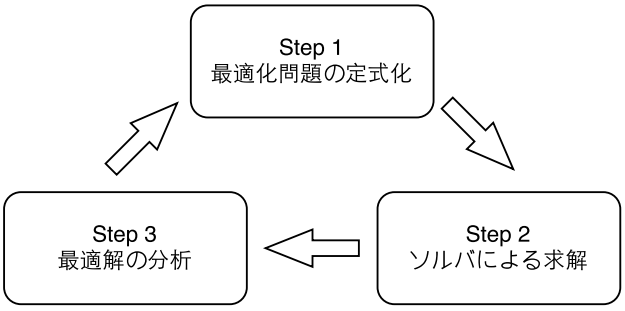

図 1.1: 数理最適化のプロセス

以下では，具体例を通じて上記のプロセスを紹介したいと思います．ここではあまり細かいことは説明せずに，直感的な理解を得ることを目指します（詳細については次章以降で学ぶことになります）．

1.2.1　例 1:「原料購入 + 輸送」問題

Step 1: 最適化問題の定式化

ここでは，次の問題を考えます（図 1.2）．

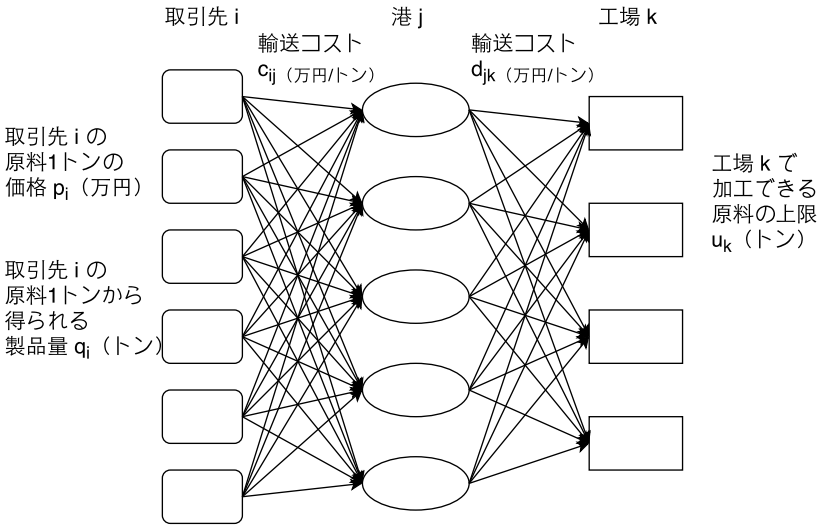

図 1.2: 「原料購入 + 輸送」問題

問題設定

ある企業では，海外にある複数の取引先 $i \in I$ から原料を買い付け，それを日本国内の複数の工場で加工して製品を製造する業務を行っています．

取引先からは原料を海路（タンカー）で日本国内に複数ある港 $j \in J$ のいずれかに運び，さらに陸路（鉄道）で国内にある複数の工場 $k \in K$ へ運んで加工します．なお，港に運ばれた原料は全ていずれかの工場に運ばなくてはなりません．ここで，取引先 $i \in I$ から港 $j \in J$ への輸送コストと港 $j \in J$ から工場 $k \in K$ への輸送コストは原料 1 トンあたりそれぞれ c_{ij}, d_{jk} 万円であるものとします．

取引先 $i \in I$ から原料を 1 トン購入するときの価格は p_i 万円です．また，原料の品質にはばらつきがあり，取引先 $i \in I$ から購入した原料 1 トンから得られる製品は q_i トンであることがわかっています．さらに，工場 $k \in K$ で加工できる原料の上限は u_k トンです．また，各工場で製造される製品の合計が f トン以上でなければならないというルールもあります．

これらの条件の下で「原料買い付けに必要な費用」と「輸送（海外から国内までの海路・国内での陸路）に必要な費用」の合計を最小にするにはどうすればよいでしょうか？

この問題は次のように定式化することができます．

$$
\left|\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{i\in I}\left(p_i\sum_{j\in J}x_{ij}\right)+\sum_{i\in I,j\in J}c_{ij}x_{ij}+\sum_{j\in J,k\in K}d_{jk}y_{jk} \\
\text{subject to} & \displaystyle\sum_{i\in I}x_{ij}=\sum_{k\in K}y_{jk} \quad (j\in J) \\
& \displaystyle\sum_{j\in J}y_{jk}\le u_k \quad (k\in K) \\
& \displaystyle\sum_{i\in I}\left(q_i\sum_{j\in J}x_{ij}\right)\ge f \quad (k\in K)
\end{array}\right. \tag{1.2.1}
$$

ここで，x_{ij} は取引先 $i \in I$ から港 $j \in J$ への輸送量，y_{jk} は港 $j \in J$ から工場 $k \in K$ への輸送量を表しています．

1.1 節で，最適化問題とは「守るべき制約の下で目的を最良（最大または最小）にする状態を求める」問題である，と説明しました．これらに対応するのが，

最適化問題の構成要素

- 目的関数
- 制約条件

の 2 つです．このことについて，詳しくは 2.2 節で説明しますが，ここではこの具体例を通じて目的関数と制約条件についてみてみます．

問題 (1.2.1) では，“minimize” に続く部分が目的関数を，“subject to” に続く部分が制約条件を表しています．

問題 (1.2.1) の目的関数は 3 つの項で構成されていますが，それぞれの項は次の量を表しています：

- 第 1 項：取引先 i からの原料購入費用の合計
- 第 2 項：取引先 i から港 j への輸送費用（海外から国内までの海路）の合計
- 第 3 項：港 j から工場 k への輸送費用（国内での陸路）の合計

そして，目的関数には “minimize”＝「最小化」という指示があることから，これら 3 項の和を最小にするような状態を求めよ，という意味になります．

一方，問題 (1.2.1) の制約条件はそれぞれ次のような意味を持っています：

- 1 番目の制約：「取引先 i から港 j に運ばれてくる原料の総量」と「港 j から工場 k に運ばれる原料の総量」が等しい（言い換えると「原料は港 j で消えてなくなることはないし，増えることもない」）
- 2 番目の制約：工場 k に運ばれる原料の総量は，工場 k で加工可能な量を上回ってはならない
- 3 番目の制約：工場 k で製造される製品量が指定された量を下回ってはならない

これが上述の問題設定を定式化したものになります．なお，ここでは定式化の細かい部分まで理解する必要はありません（1.2.2 節，1.2.3 節も同様です）．次章以降を読むことで理解できるようになるはずです．

Step 2: ソルバを用いた求解

さて，問題の定式化ができたら，次は実際に問題を解くことになります．先ほど説明したように，この本では，ソルバというソフトウェアを用いて問題を解くことにします．そのためには，上で定式化した問題を，ソルバが理解できるような形で作成する必要があります．これにはいくつかの方法がありますが，ここでは**モデリング言語**を用いて作成することにします．最適化問題 (1.2.1) を **AMPL** というモデリング言語で記述すると，次のようになります [2]．

[2] 実際は，テキストファイルとして作成することになります

```
set I;
set J;
set K;

var x{I, J} >= 0;
var y{J, K} >= 0;

param c{I, J};
param d{J, K};
param p{I};
param q{I};
param u{K};
param f;

minimize Objective:
        sum{i in I} (p[i] * sum{j in J} x[i, j])
        + sum{i in I, j in J} (c[i, j] * x[i, j])
        + sum{j in J, k in K} (d[j, k] * y[j, k]);

subject to FlowConservation {j in J}:
        sum{i in I} x[i, j] = sum{k in K} y[j, k];

subject to ProductionLimit {k in K}:
        sum{j in J} y[j, k] <= u[k];

subject to ProductionAmount:
        sum{i in I} (q[i] * sum{j in J} x[i, j]) >= f;

end;
```

モデリング言語 AMPL を用いた最適化問題の記述方法は 4 章で詳細に説明します．ただ，この記述と定式化 (1.2.1) を見比べると，その対応関係は直感的に理解できるのではないかと思います．

さて，最適化問題をソルバが理解できる形式で記述することはできましたが，これだけでは問題を解くことはできません．問題に現れるパラメータの具体的な値が必要になります．ここでは，以下のデータを用いることにします．

```
set I := 1 2 3 4 5 6;
set J := 1 2 3 4 5;
set K := 1 2 3 4;

param c: 1 2 3 4 5 :=
1 30.0 27.8 32.2 28.9 29.8
2 31.2 30.1 33.5 32.1 28.8
3 32.5 33.3 29.9 31.7 30.6
4 29.8 31.6 31.8 30.9 29.5
5 30.5 29.5 28.5 31.2 31.0
6 29.7 30.0 31.1 29.4 30.0
;

param d: 1 2 3 4 :=
```

```
1 13.2 12.8 15.0  8.8
2  2.8 10.5 11.7 10.0
3  9.9 11.2 10.9 14.2
4 14.0 12.7 17.5  7.3
5 11.1 12.0  8.8 11.5
;

param p :=
1 101.2  2 96.9 3 115.8  4 120.3  5 110.8  6 99.7;

param q :=
1 0.50  2 0.45  3 0.56  4 0.61  5 0.55  6 0.48;

param u :=
1 60.0  2 52.5  3 54.2  4 56.8;

param f := 100;

end;
```

この記述についても詳細は 4 章で説明をしますが，例えば先頭の `set I := 1 2 3 4 5 6;` は，取引先 $i \in I$ が 1 から 6 の 6 箇所あることを表しています．同様に港 $j \in J$ は 5 箇所，工場 $k \in K$ は 4 箇所になります．さらに，定式化中に現れる各パラメータに対する値を与えています．例えば，`param c` の部分を見ると，取引先 1 から港 2 までの輸送コストは 1 トンあたり $c_{12} = 29.2$ 万円であることがわかります．

さて，これでソルバで最適解を求める準備ができました．この本では最適化ソルバとして **GLPK** (GNU Linear Programming Kit) [27] を用います．GLPK は GNU というフリーのソフトを開発するプロジェクトに含まれる最適化ソルバです．フリーソフトですので，ダウンロードすることにより誰でも利用することができます．

先ほどの問題記述とデータを GLPK に与えると（与え方は付録 D で紹介します），次のような結果を得ることができます．

```
Problem:    sample_model_1
Rows:       11
Columns:    50
Non-zeros:  150
Status:     OPTIMAL
Objective:  Objective = 25760.32 (MINimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 Objective    B        25760.3
     2 FlowConservation[1]
                    NS             0            -0             =          -8.9
     3 FlowConservation[2]
                    NS             0            -0             =          -6.7
     4 FlowConservation[3]
                    NS             0            -0             =         -10.9
```

```
     5 FlowConservation[4]
                   NS             0            -0             =          -7.4
     6 FlowConservation[5]
                   NS             0            -0             =          -8.8
     7 ProductionLimit[1]
                   NU            60                          60          -3.9
     8 ProductionLimit[2]
                   B              0                        52.5
     9 ProductionLimit[3]
                   B        47.1344                        54.2
    10 ProductionLimit[4]
                   NU          56.8                        56.8          -0.1
    11 ProductionAmount
                   NL           100           100                         260

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 x[1,1]       NL             0             0                        10.1
     2 x[1,2]       NL             0             0                         5.7
     3 x[1,3]       NL             0             0                        14.3
     4 x[1,4]       NL             0             0                         7.5
     5 x[1,5]       NL             0             0                         9.8
     6 x[2,1]       NL             0             0                          20
     7 x[2,2]       NL             0             0                        16.7
     8 x[2,3]       NL             0             0                        24.3
     9 x[2,4]       NL             0             0                        19.4
    10 x[2,5]       NL             0             0                        17.5
    11 x[3,1]       NL             0             0                        11.6
    12 x[3,2]       NL             0             0                        10.2
    13 x[3,3]       NL             0             0                          11
    14 x[3,4]       NL             0             0                         9.3
    15 x[3,5]       NL             0             0                         9.6
    16 x[4,1]       NL             0             0                         0.4
    17 x[4,2]       B             60             0
    18 x[4,3]       NL             0             0                         4.4
    19 x[4,4]       B           56.8             0
    20 x[4,5]       B        47.1344             0
    21 x[5,1]       NL             0             0                         7.2
    22 x[5,2]       NL             0             0                           4
    23 x[5,3]       NL             0             0                         7.2
    24 x[5,4]       NL             0             0                         6.4
    25 x[5,5]       NL             0             0                         7.6
    26 x[6,1]       NL             0             0                        13.5
    27 x[6,2]       NL             0             0                        11.6
    28 x[6,3]       NL             0             0                        16.9
    29 x[6,4]       NL             0             0                        11.7
    30 x[6,5]       NL             0             0                        13.7
    31 y[1,1]       NL             0             0                         8.2
    32 y[1,2]       NL             0             0                         3.9
    33 y[1,3]       NL             0             0                         6.1
    34 y[1,4]       B              0             0
    35 y[2,1]       B             60             0
    36 y[2,2]       NL             0             0                         3.8
    37 y[2,3]       NL             0             0                           5
```

```
    38 y[2,4]       NL             0             0                    3.4
    39 y[3,1]       NL             0             0                    2.9
    40 y[3,2]       NL             0             0                    0.3
    41 y[3,3]       B              0             0
    42 y[3,4]       NL             0             0                    3.4
    43 y[4,1]       NL             0             0                   10.5
    44 y[4,2]       NL             0             0                    5.3
    45 y[4,3]       NL             0             0                   10.1
    46 y[4,4]       B           56.8             0
    47 y[5,1]       NL             0             0                    6.2
    48 y[5,2]       NL             0             0                    3.2
    49 y[5,3]       B        47.1344             0
    50 y[5,4]       NL             0             0                    2.8

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.DE: max.abs.err = 2.84e-14 on column 17
        max.rel.err = 8.93e-17 on column 17
        High quality

KKT.DB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

この結果を見ると，なにやら答が得られているように思います．この分析は次の項で扱いましょう．

Step 3: 最適解の検証

ソルバ GLPK で得られた結果の見方は 5 章で詳しく紹介しますが，ここではその概要を見てみたいと思います．

結果の先頭部分に `Status: OPTIMAL` という出力があることがわかると思います．これは，ソルバが最適解 (“optimal solution”) を見つけることができた，ということを表しています．その次の行には `Objective = 25760.32` という出力があります．これは，最適化における目的関数の値，すなわちこの問題では最小化された費用（原料買い付け費用 + 輸送費用）を表しています．

それ以下には `Row name` という箇所と `Column name` という箇所があります．これらは，それぞれ制約条件，変数に関する項目を記載しています（これらをなぜ `Row`, `Column` と表現するかは 5.3.3 節を参照のこと）．

ここでは変数について見てみます．`Column name` の右に `Activity` という列がありますが，これが最適な変数の値を表しています．例えば，最適解において，変数 `x[4,1]` の値は `0`，`x[4,2]` の値は `60`, ... のように読みます．すなわち，最適解においては取引先 4 から港 1 までは輸送を行わない $(x_{41} = 0)$，また取引先 4 から港 2 までは 60 トン輸送する $(x_{42} = 60)$ となるわけです．

さて,『最適解がわかった！めでたしめでたし』となるかというと，必ずしもそうではありません．最適解を分析してみると，担当者はあることに気付いたのです…（次節に続く）

1.2.2 例 2: 制約条件の追加

Step 1: 最適化問題の定式化（の修正）

実は，担当者は次のことをすっかり忘れていました．

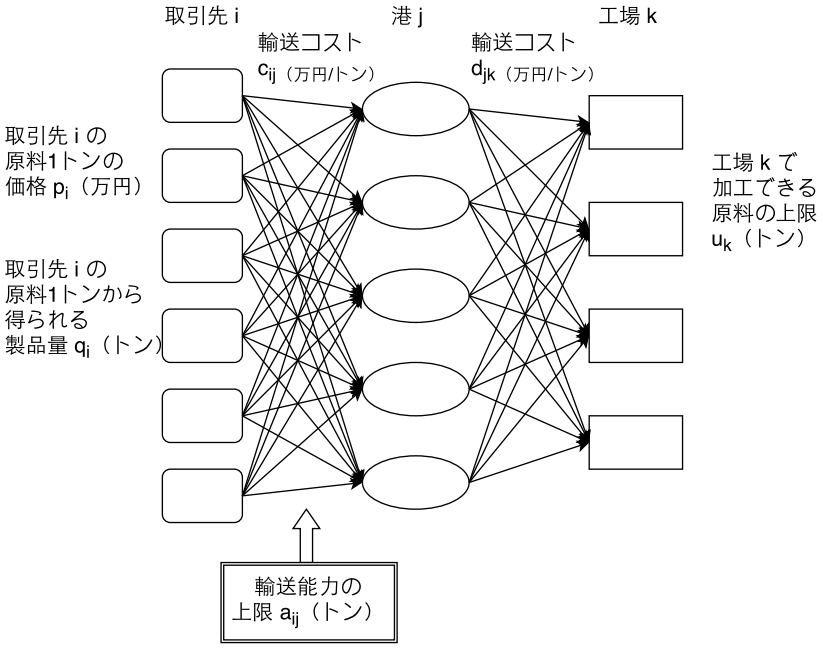

図 1.3: 「原料購入 + 輸送」問題への制約条件の追加

担当者が忘れていた条件

取引先 $i \in I$ から港 $j \in J$ までの輸送能力には上限 a_{ij} がある

つまり，取引先 $i \in I$ と港 $j \in J$ の間の輸送経路を用いて運ぶことができる原料の量には限りがある，ということです．

残念ながら，先ほどの最適解は正しくない可能性があるため，この条件を追加して改めて問題を解く必要があります．この条件を定式化すると次のようになります．

$$x_{ij} \leq a_{ij} \quad (i \in I, j \in J) \tag{1.2.2}$$

Step 2: ソルバを用いた求解

追加する制約 (1.2.2) をモデリング言語 AMPL を用いて記述すると次のようになります.

```
param a{I, J};
  :
  :
subject to TransportLimit {i in I, j in J}:
        x[i, j] <= a[i, j];
```

また，取引先 $i \in I$ から港 $j \in J$ までの輸送能力の上限値 a_{ij} は，次のように与えられるものとします.

```
param a: 1 2 3 4 5 :=
1 20.1 18.8 22.2 19.5 25.5
2 15.3 13.2 17.7 16.1 10.2
3 17.2 18.8 20.0 11.8 14.5
4 22.2 19.3 12.5 20.4 18.8
5 18.2 17.3 10.9 14.2 13.3
6 12.8 10.4 13.3 14.2 13.6
;
```

これらの設定を 1.2.1 節で示した記述に追加して解くと，次のような結果を得ることができます.

```
Problem:    sample_model_2
Rows:       41
Columns:    50
Non-zeros:  180
Status:     OPTIMAL
Objective:  Objective = 26327.80339 (MINimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 Objective    B        26327.8
     2 FlowConservation[1]
                    NS             0            -0             =      -12.2232
     3 FlowConservation[2]
                    NS             0            -0             =          -6.4
     4 FlowConservation[3]
                    NS             0            -0             =         -11.2
     5 FlowConservation[4]
                    NS             0            -0             =      -10.7232
     6 FlowConservation[5]
                    NS             0            -0             =          -9.1
     7 ProductionLimit[1]
```

```
                NU           60                        60          -3.6
 8 ProductionLimit[2]
                B       2.25893                      52.5
 9 ProductionLimit[3]
                NU          54.2                      54.2          -0.3
10 ProductionLimit[4]
                NU          56.8                      56.8       -3.42321
11 ProductionAmount
                NL           100           100                   277.679
12 TransportLimit[1,1]
                B              0                      20.1
13 TransportLimit[1,2]
                NU          18.8                      18.8       -3.43929
14 TransportLimit[1,3]
                B              0                      22.2
15 TransportLimit[1,4]
                B              0                      19.5
16 TransportLimit[1,5]
                B              0                      25.5
17 TransportLimit[2,1]
                B              0                      15.3
18 TransportLimit[2,2]
                B              0                      13.2
19 TransportLimit[2,3]
                B              0                      17.7
20 TransportLimit[2,4]
                B              0                      16.1
21 TransportLimit[2,5]
                B              0                      10.2
22 TransportLimit[3,1]
                B              0                      17.2
23 TransportLimit[3,2]
                B            4.6                      18.8
24 TransportLimit[3,3]
                B              0                        20
25 TransportLimit[3,4]
                B              0                      11.8
26 TransportLimit[3,5]
                B       0.958929                      14.5
27 TransportLimit[4,1]
                NU          22.2                      22.2       -7.06071
28 TransportLimit[4,2]
                NU          19.3                      19.3       -11.0839
29 TransportLimit[4,3]
                NU          12.5                      12.5       -6.08393
30 TransportLimit[4,4]
                NU          20.4                      20.4       -7.46071
31 TransportLimit[4,5]
                NU          18.8                      18.8       -10.4839
32 TransportLimit[5,1]
                B              0                      18.2
33 TransportLimit[5,2]
                NU          17.3                      17.3       -6.02321
34 TransportLimit[5,3]
                NU          10.9                      10.9       -2.22321
```

```
    35 TransportLimit[5,4]
                   B           14.2                        14.2
    36 TransportLimit[5,5]
                   NU          13.3                        13.3      -1.82321
    37 TransportLimit[6,1]
                   B              0                        12.8
    38 TransportLimit[6,2]
                   B              0                        10.4
    39 TransportLimit[6,3]
                   B              0                        13.3
    40 TransportLimit[6,4]
                   B              0                        14.2
    41 TransportLimit[6,5]
                   B              0                        13.6

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 x[1,1]       NL             0             0                    4.58393
     2 x[1,2]       B           18.8             0
     3 x[1,3]       NL             0             0                    5.76071
     4 x[1,4]       NL             0             0                    1.98393
     5 x[1,5]       NL             0             0                    1.26071
     6 x[2,1]       NL             0             0                    15.3679
     7 x[2,2]       NL             0             0                    8.44464
     8 x[2,3]       NL             0             0                    16.6446
     9 x[2,4]       NL             0             0                    14.7679
    10 x[2,5]       NL             0             0                    9.84464
    11 x[3,1]       NL             0             0                    5.02321
    12 x[3,2]       B            4.6             0
    13 x[3,3]       NL             0             0                        1.4
    14 x[3,4]       NL             0             0                    2.72321
    15 x[3,5]       B       0.958929             0
    16 x[4,1]       B           22.2             0
    17 x[4,2]       B           19.3             0
    18 x[4,3]       B           12.5             0
    19 x[4,4]       B           20.4             0
    20 x[4,5]       B           18.8             0
    21 x[5,1]       NL             0             0                        0.8
    22 x[5,2]       B           17.3             0
    23 x[5,3]       B           10.9             0
    24 x[5,4]       B           14.2             0
    25 x[5,5]       B           13.3             0
    26 x[6,1]       NL             0             0                     8.3375
    27 x[6,2]       NL             0             0                    2.81429
    28 x[6,3]       NL             0             0                    8.71429
    29 x[6,4]       NL             0             0                     6.5375
    30 x[6,5]       NL             0             0                    5.51429
    31 y[1,1]       NL             0             0                    4.57679
    32 y[1,2]       NL             0             0                   0.576786
    33 y[1,3]       NL             0             0                    3.07679
    34 y[1,4]       B           22.2             0
    35 y[2,1]       B             60             0
    36 y[2,2]       NL             0             0                        4.1
    37 y[2,3]       NL             0             0                        5.6
```

```
    38 y[2,4]       NL             0             0                7.02321
    39 y[3,1]       NL             0             0                    2.3
    40 y[3,2]       B        2.25893             0
    41 y[3,3]       B        21.1411             0
    42 y[3,4]       NL             0             0                6.42321
    43 y[4,1]       NL             0             0                6.87679
    44 y[4,2]       NL             0             0                1.97679
    45 y[4,3]       NL             0             0                7.07679
    46 y[4,4]       B           34.6             0
    47 y[5,1]       NL             0             0                    5.6
    48 y[5,2]       NL             0             0                    2.9
    49 y[5,3]       B        33.0589             0
    50 y[5,4]       NL             0             0                5.82321

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err = 7.11e-15 on row 6
        max.rel.err = 1.06e-16 on row 6
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.DE: max.abs.err = 2.84e-14 on column 12
        max.rel.err = 9.27e-17 on column 24
        High quality

KKT.DB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

Step 3: 最適解の検証

この結果と 1.2.1 節の結果を比べると，再計算した結果には `TransportLimit[1,1]`, ..., `TransportLimit[6,5]` という行が増えていることがわかると思います．これは追加した制約，すなわち取引先 $i \in I$ から港 $j \in J$ までの輸送能力に関する制約についての情報を表しています．

ここで `TransportLimit[3,2]` についての情報を見てみましょう．まず，`Upper bound`，すなわち輸送量の上限が 18.8 となっています．これは，上で準備したデータからも確認することができます．これに対し，`Activity` の列で実際の輸送量を確認すると 4.6 であることがわかります．すなわち，上限に対して余裕を持って輸送を行っていることになります．

次に `TransportLimit[4,1]` についても見てみましょう．この場合，輸送量の上限，実際の輸送量とも 22.2 になっていることがわかります．従って，この経路には輸送可能量に余裕のないことがわかります．

その他，いろいろ検証したところ，求めた解に誤りはなさそうです．担当者は「よかったよかった」と思っていたのですが，今度は取引先が新たな条件を提示してきたのです…（さらに続く）

1.2.3　例 3: 整数性条件の追加

Step 1: 最適化問題の定式化（の修正）

取引先から提示された新たな条件は，

新たな制約条件

各取引先 $i \in I$ からの原料の購入は 5 トン単位でしかできない

というものでした（図 1.4）．先ほどと同様，これを新たに制約条件として取り込む必要があります．

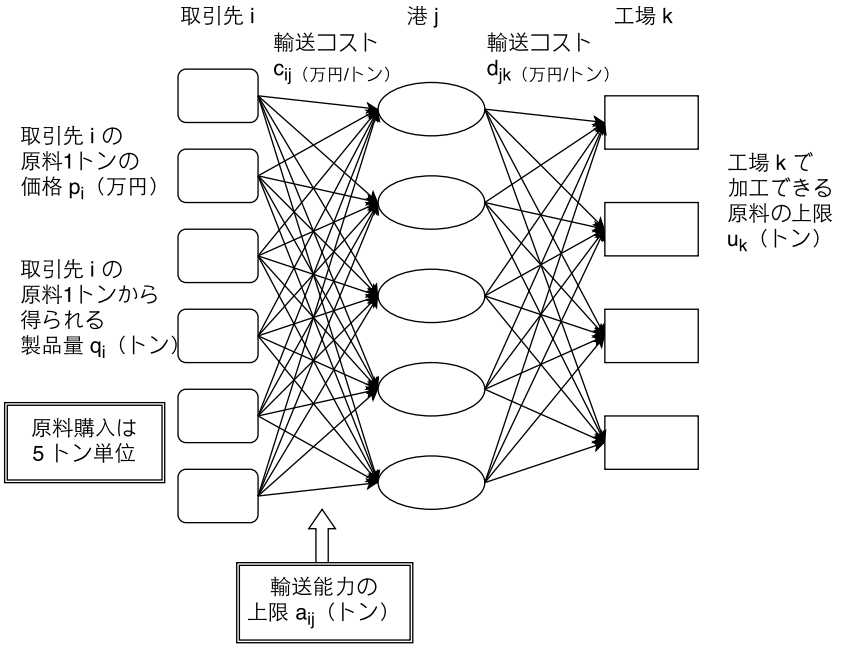

図 1.4: 「原料購入 + 輸送」問題への整数性条件の追加

この条件を定式化するためには，**整数変数** を導入する必要があります．整数変数については 2.2.5 節で詳しく説明しますが，ここでは「値として整数値 $(0, \pm 1, \pm 2, \ldots)$ のみを取る変数」と理解しておけば十分です．

ここでは，上記の条件を次のように定式化することにします．

$$\left| \quad \sum_{j \in J} x_{ij} = s z_i \quad (i \in I) \right. \tag{1.2.3}$$

ここで s は購入単位（この問題では 5 トンとなります）を表すパラメータ，z_i は取引先 i から購入する単位数とします．上述のように，z_i の値は整数でなくてはなりません．

(1.2.3) の左辺は，ある取引先 i から，それぞれの港 $j \in J$ に輸送する量の総和を表しています．すなわちこれは取引先 i から原料を購入した量に他なりません．一方，右辺は購入単位 s に取引先 i からの購入単位数（整数値）を乗じているので，$s = 5.0$ トン単位の量を表現していることになります．この制約は，これらが等しくなるということを表しているため，購入量が 5 トン単位に制限されることになります．

Step 2: ソルバを用いた求解

追加する制約 (1.2.3) を記述すると次のようになります.

```
var z{I} integer;
  :
param s;
  :
subject to UnitPurchase {i in I}:
        sum{j in J} x[i, j] == s * z[i];
```

また，s の値は次のように与えます.

```
param s := 5.0;
```

これらの設定を 1.2.2 節で示した記述に追加して解くと，次のような結果を得ることができます.

```
Problem:    sample_model_3
Rows:       47
Columns:    56 (6 integer, 0 binary)
Non-zeros:  216
Status:     INTEGER OPTIMAL
Objective:  Objective = 26380.7 (MINimum)

   No.   Row name        Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 Objective             26380.7
     2 FlowConservation[1]
                                   0            -0             =
     3 FlowConservation[2]
                                   0            -0             =
     4 FlowConservation[3]
                                   0            -0             =
     5 FlowConservation[4]
                                   0            -0             =
     6 FlowConservation[5]
                                   0            -0             =
     7 ProductionLimit[1]
                                  60                          60
     8 ProductionLimit[2]
                                   4                        52.5
     9 ProductionLimit[3]
                                54.2                        54.2
    10 ProductionLimit[4]
                                56.8                        56.8
```

```
11 ProductionAmount
                          100          100
12 TransportLimit[1,1]
                            0                       20.1
13 TransportLimit[1,2]
                         18.8                       18.8
14 TransportLimit[1,3]
                            0                       22.2
15 TransportLimit[1,4]
                         14.2                       19.5
16 TransportLimit[1,5]
                            2                       25.5
17 TransportLimit[2,1]
                            0                       15.3
18 TransportLimit[2,2]
                            0                       13.2
19 TransportLimit[2,3]
                            0                       17.7
20 TransportLimit[2,4]
                            0                       16.1
21 TransportLimit[2,5]
                            0                       10.2
22 TransportLimit[3,1]
                            0                       17.2
23 TransportLimit[3,2]
                          4.6                       18.8
24 TransportLimit[3,3]
                            0                         20
25 TransportLimit[3,4]
                            0                       11.8
26 TransportLimit[3,5]
                          5.4                       14.5
27 TransportLimit[4,1]
                         22.2                       22.2
28 TransportLimit[4,2]
                         19.3                       19.3
29 TransportLimit[4,3]
                          9.3                       12.5
30 TransportLimit[4,4]
                         20.4                       20.4
31 TransportLimit[4,5]
                         18.8                       18.8
32 TransportLimit[5,1]
                            0                       18.2
33 TransportLimit[5,2]
                         17.3                       17.3
34 TransportLimit[5,3]
                         10.9                       10.9
35 TransportLimit[5,4]
                            0                       14.2
36 TransportLimit[5,5]
                         11.8                       13.3
37 TransportLimit[6,1]
                            0                       12.8
38 TransportLimit[6,2]
```

```
                                  0                        10.4
    39 TransportLimit[6,3]
                                  0                        13.3
    40 TransportLimit[6,4]
                                  0                        14.2
    41 TransportLimit[6,5]
                                  0                        13.6
    42 UnitPurchase[1]
                                  0            -0             =
    43 UnitPurchase[2]
                                  0            -0             =
    44 UnitPurchase[3]
                                  0            -0             =
    45 UnitPurchase[4]
                                  0            -0             =
    46 UnitPurchase[5]
                                  0            -0             =
    47 UnitPurchase[6]
                                  0            -0             =

   No. Column name       Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 x[1,1]                     0             0
     2 x[1,2]                  18.8             0
     3 x[1,3]                     0             0
     4 x[1,4]                  14.2             0
     5 x[1,5]                     2             0
     6 x[2,1]                     0             0
     7 x[2,2]                     0             0
     8 x[2,3]                     0             0
     9 x[2,4]                     0             0
    10 x[2,5]                     0             0
    11 x[3,1]                     0             0
    12 x[3,2]                   4.6             0
    13 x[3,3]                     0             0
    14 x[3,4]                     0             0
    15 x[3,5]                   5.4             0
    16 x[4,1]                  22.2             0
    17 x[4,2]                  19.3             0
    18 x[4,3]                   9.3             0
    19 x[4,4]                  20.4             0
    20 x[4,5]                  18.8             0
    21 x[5,1]                     0             0
    22 x[5,2]                  17.3             0
    23 x[5,3]                  10.9             0
    24 x[5,4]                     0             0
    25 x[5,5]                  11.8             0
    26 x[6,1]                     0             0
    27 x[6,2]                     0             0
    28 x[6,3]                     0             0
    29 x[6,4]                     0             0
    30 x[6,5]                     0             0
    31 y[1,1]                     0             0
    32 y[1,2]                     0             0
    33 y[1,3]                     0             0
```

```
    34 y[1,4]                    22.2             0
    35 y[2,1]                      60             0
    36 y[2,2]                       0             0
    37 y[2,3]                       0             0
    38 y[2,4]                       0             0
    39 y[3,1]                       0             0
    40 y[3,2]                       4             0
    41 y[3,3]                    16.2             0
    42 y[3,4]                       0             0
    43 y[4,1]                       0             0
    44 y[4,2]                       0             0
    45 y[4,3]                       0             0
    46 y[4,4]                    34.6             0
    47 y[5,1]                       0             0
    48 y[5,2]                       0             0
    49 y[5,3]                      38             0
    50 y[5,4]                       0             0
    51 z[1]          *              7
    52 z[2]          *              0
    53 z[3]          *              2
    54 z[4]          *             18
    55 z[5]          *              8
    56 z[6]          *              0

Integer feasibility conditions:

KKT.PE: max.abs.err = 1.42e-14 on row 45
        max.rel.err = 1.00e-16 on row 42
        High quality

KKT.PB: max.abs.err = 2.13e-14 on row 42
        max.rel.err = 2.13e-14 on row 42
        High quality

End of output
```

Step 3: 最適解の検証

この結果の下の方に，最適解における `z[1]`, ..., `z[6]` の値が出力されています．例えば `z[1]` は 7 ですから，取引先 1 からは $5 \times 7 = 35$ トン購入していることになります．このことは，取引先 1 から各港への輸送量 `x[1,1]`, `x[1,2]`, `x[1,3]`, `x[1,4]`, `x[1,5]` の総和としても得られます（$0 + 18.8 + 0 + 14.2 + 2 = 35$）．どうやら，各取引先からの購入量が 5 トン単位になる，という制約を考慮することができているようです．

この例で追加した制約のため，変数の値は 1.2.2 節での結果から随分変わっているようです．ここではその変化を追いかけることはしませんが，表などにまとめて比較し，その理由を考えてみると面白いでしょう．

1.2.4 まとめ：3 つの例を通じて

ここでは 3 つの例を通じて，数理最適化のプロセスがどのようなものであるか，その概要を説明しました．イメージをつかむことができたでしょうか？

最後に，示した例を通じて最適解における目的関数値がどのように変化したかを見ておきましょう．

- 1.2.1 節での結果：`Objective = 25760.32`
- 1.2.2 節での結果：`Objective = 26327.80339`
- 1.2.3 節での結果：`Objective = 26380.7`

となっており，目的関数値はは少しずつ悪くなっている（費用が増えている）ことがわかります．これは，3 つの例で制約が追加されていることによるものです．一般に，制約が厳しくなると，状態は悪くなる（良くはならない）はずです（これは日常生活の中でも様々な形で経験することです）．それがこの結果にも表れています．

1.3 この本の構成

この本の構成を，前節で示した数理最適化のプロセスと対応づけて示すと，以下のようになります：

- Step 1: 最適化問題の定式化
  - 数理モデル・数理最適化問題（2 章）
  - 最適化問題の定式化技法（3 章）
- Step 2: ソルバを用いた求解
  - モデリング言語による最適化問題の記述（4 章）
  - ソルバによる求解（5.1 節）
- Step 3: 最適解の検証
  - 得られた解の最適性（5.2 節）
  - ソルバの出力の確認方法（5.3 節）
  - 感度分析（5.4 節）

さらに，上記の各プロセスへの理解を深める内容として，

- よく知られている最適化問題（6 章）
- 難しい状況への対応方法のヒント（7 章）

を紹介しています．

これらの内容を理解して頂ければ，実際の問題に最適化手法を適用する上で必要になる基本的な方法を身につけて頂けるはずです．

第2章　数理モデルと最適化問題

本章で学ぶ内容

- 数理モデルの概念
- 数理最適化の概念
- 最適化問題の一般形，解の性質，問題の数学的な分類

この章では,（数理）モデルという考え方，また数理最適化の基本的な事項について学びます.

2.1　数理モデルとは

2.1.1　対象のモデル化

まず，本書でよく出てくることになる言葉「**モデル化**」について説明します．まず「**モデル**」という言葉を広辞苑で調べてみると，次のような意味が書いてあります：

(1) 型，形式.「最新―」

(2) 模型，ひな形.「計量経済―」

(3) 模範，手本.「―地区」

(4) 美術家が制作の対象にする人.

(5) 小説・戯曲などの題材とされた実在の人物.「―小説」

(6) ファッション・モデルの略.

本書で用いる「モデル化」の「モデル」は (2) の意味です．すなわち「モデル化」とは「対象を表現する模型・ひな形を作ること」です．例えば,「車の外形を粘土で作る」ということはは「車」の「模型」を作っていることに他なりませんから，これは「モデル化」と言えます.

考察の対象をモデル化することの意味は，作成したモデルを用いて対象を理解すること にあります．例えば「粘土で作った車」（図 2.1）は，そのモデルを風洞実験に用いることで，高速走行時に車の周りの空気がどのように流れるかを知ることに用いることができます[1]．また，以下に示すように，様々な学問分野で「モデルを作り，対象を理解する」という行為が行われています：

[1]風洞実験の様子を YouTube などの動画サイトで見てみるのもよいでしょう．例えば “wind tunnel” などのキーワードで検索してみてください.

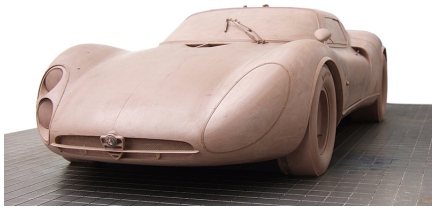

図 2.1: 粘土（クレイ）で作った車
アシスト株式会社 HP より引用 (http://www.assist-it.co.jp/sa207.html)

- 化学：面心立方格子，体心立方格子，六方最密格子
  - 物質の中での原子の配列を表したモデル（図 2.2, 2.3）
- 生物学：二重らせんモデル
  - DNA の構造を表現したモデル（図 2.4）
- 経済学：需要・供給モデル
  - 需要曲線と供給曲線の交点でものの価格が決まる

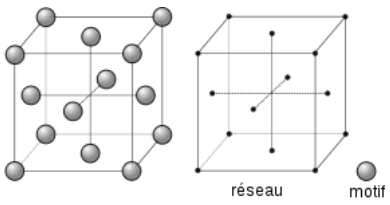

図 2.2: 面心立方格子
Wikipedia より引用
http://ja.wikipedia.org/wiki/面心立方格子構造

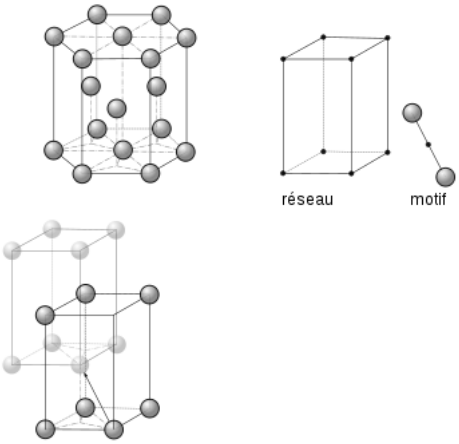

図 2.3: 六方最密格子
Wikipedia より引用
http://ja.wikipedia.org/wiki/六方最密充填構造

ここで，物理でよくありそうな次の問題を考えてみましょう：

問題 2.1.1 物理の問題（エネルギー保存則）
軽い糸に重りをつけた単振り子があります．この重りを糸がたるまないように持ち上げて，静かに離します．空気抵抗は無視します．このとき重りはどの高さまで上がるでしょうか？

(a) 離した高さまで上がらない

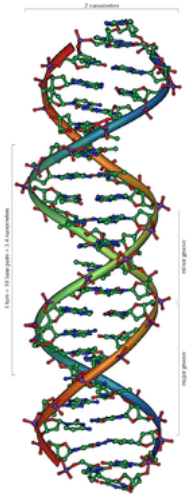

図 2.4: DNA の二重らせん構造
Wikipedia より引用
http://ja.wikipedia.org/wiki/二重らせん

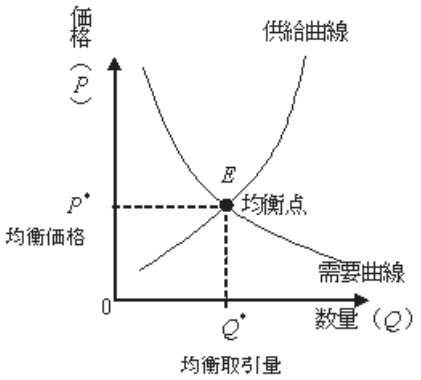

図 2.5: 需要・供給曲線
Wikipedia より引用
http://ja.wikipedia.org/wiki/需要と供給

(b) ちょうど離した高さまで上がる

(c) 離した高さよりも高く上がる □

恐らく，問題 2.1.1 に対しては (b) と答える人が多いのではないでしょうか（物理について勉強したことがある人は，エネルギー保存則という法則を勉強したはずです）．しかし，この問題をよく読むといろんなことが見えてきます．例えば，

- 「軽い糸」というが，重さは全くない糸なんてないから，糸の重さを考慮しなくてはならないのではないか？
- 「糸がたるまないように」するのなら，糸に力が掛かっていることになる．すると，糸はわずかでも伸びてしまうのではないか？
- 「空気抵抗は無視してよい」というが，そんな世界って本当に作れるのか？

…などなど．恐らく，現実世界でこの問題と同じ状況を作ることはできないでしょう．しかし，この問題で想像している現象（すなわち (b)）と，現実世界で起きる現象との差は非常に小さいため，この問題を考える意義があるわけです．

Note: 似顔絵にうぶ毛は描かない　似顔絵は，実際に存在する人の顔を「モデル化」したものであると言えます．そして似顔絵を書く際には，その人の特徴（目が大きい/細い，眉が太い/細い，鼻が大きい/小さい，etc...）を捉えて（場合によってはやや強調して）描きます．しかし，通常，人の顔にあるはずのうぶ毛は似顔絵には描きません．つまり，「モデル化」とは，必ずしも「そこにある物を忠実に再現すること」ということではないのです．モデルの目的を考え，不必要なものを削る作業もモデル化の一部なのです．

2.1.2　数理モデル

対象をモデル化する際に，数理的・数学的な道具を用いてモデル化したものを**数理モデル**といいます[2]（これに対し，先ほどの「粘土で作った車のモデル」は**物理モデル**といいます）．数理モデルの例を以下に示します．

例 2.1.1 リトルの公式 [20]
オペレーションズ・リサーチの一分野に「**待ち行列**」に関する分野があります．待ち行列とは，例えばスーパーのレジで会計を待つ時の列のように，あるサービス（この場合は会計処理）を受けるために待つ列のことです．

待ち行列における基本的な公式に「リトルの公式」があります．これをスーパーのレジでの待ち行列になぞらえて説明すると，1 分あたりにレジにやってくる人数の平均を λ（人/分），レジに並んでいる人の人数（レジで会計処理をしてもらっている人を含む）の平均を L（人），レジに並び始めてから会計処理が終わるまでに要する時間の平均を W（分）とすると

$$L = \lambda W$$

が成立するというものです．　□

例 2.1.2（都市計画における）**グラビティモデル** [3]
グラビティモデルとは，都市にある施設を作る際に，どの程度の人数がその施設を利用するかを説明するためのモデルです．

ここでは小売店舗の利用者数について考えてみます．店舗の利用者を実際に調べてみると，次のようなことが成立しているそうです．

- 店舗が魅力的ならば利用者数は増える
- 利用したい人が多ければ，実際の利用者も多い
- アクセスの便利さに応じて利用者数は変わる

このことを，W. J. Reilly という人は次のように数学的に解釈しました．

- 利用者数は店舗の売り場面積 m_1 に比例する

[2] あるいは**数学モデル**とも言います．

- 利用者数は商圏人口 m_2 に比例する
- 利用者数は店舗までの距離 r の二乗に反比例する

このとき，利用者数 F は

$$F = k\frac{m_1 m_2}{r^2} \tag{2.1.1}$$

とすることができます（k は比例定数）．式 (2.1.1) を見て,「どこかで見たことある式だな」と思った方も多いでしょう．そうです，これは物理学における万有引力の法則と全く同じ式になっています．

また上記の考え方を元にすると，2 つの店舗を結ぶ線分上にいる利用者がどちらに引き付けられるかという，勢力の均衡点を求めることができます．式 (2.1.1) からわかるように，店舗 i, j の魅力はそれぞれの売り場面積 m_i, m_j に比例し，店舗までの距離 r_i, r_j の 2 乗に反比例します．すると，均衡点では，

$$\frac{m_i}{r_i^2} = \frac{m_j}{r_j^2}$$

となります．$d = r_i + r_j$ とし，これを整理すると

$$r_i = \frac{d}{1 + \sqrt{m_j/m_i}}$$

を得ます．すなわち，店舗 i から店舗 j に向けて r_i だけ離れた点が 2 つの店舗の勢力が均衡する点になります． □

例 2.1.3 Hodgkin–Huxley（ホジキン–ハクスレイ）モデル [18] ***Hodgkin–Huxley* モデル**

生物の脳はニューロンと呼ばれる細胞でできています．そして，ニューロンは電位を変えることで情報の伝達や処理を行っています．A. L. Hodgkin と A. F. Huxley は 1952 年の論文で，このニューロンの電位変化を表す次の数理モデル（微分方程式）を提案しました．

$$\left|\begin{array}{l} C\frac{\mathrm{d}V}{\mathrm{d}t} = I - 120.0m^3h(V - 115.0) - 40.0n^4(V + 12.0) - 0.24(V - 10.613) \\ \frac{\mathrm{d}m}{\mathrm{d}t} = \frac{0.1(25-V)}{\exp\left(\frac{25-V}{10}\right)-1}(1-m) - 4\exp\left(\frac{-V}{18}\right)m \\ \frac{\mathrm{d}h}{\mathrm{d}t} = 0.07\exp\left(\frac{-V}{20}\right)(1-h) - \frac{1}{\exp\left(\frac{30-V}{10}+1\right)}h \\ \frac{\mathrm{d}n}{\mathrm{d}t} = \frac{0.01(10-V)}{\exp\left(\frac{10-V}{10}-1\right)}(1-n) - 0.125\exp\left(\frac{-V}{80}\right)n \end{array}\right.$$

（各記号の意味はここでは触れません．詳細は [18] を参照のこと.）この数理モデルはヤリイカの神経の実挙動と極めて良く一致しました．この功績により，ホジキンとハクスレイは 1963 年のノーベル生理学・医学賞を受賞しています． □

例 2.1.4 （おまけ）恋愛の微分方程式モデル [17]

数理モデルについて調べていたところ，*"Love Affairs and Differential Equations"*（「恋愛と微分方程式」）なるタイトルの論文を見つけました．この論文は連立微分方程式に興味を持たせる題材として書かれたもののようですので（著者が論文にそう書いています），ここで扱う題材としても適当でしょう．

ここではロミオ (Romeo) とジュリエット (Juliet) の恋愛事情を微分方程式で表現します．ジュリエットの気持ちはロミオの気持ちに連動して変動します．つまり，ロミオがジュリエットのことを好

き/嫌いな量に比例して，ジュリエットはロミオのことを好き/嫌いになります．一方，(原作に反して？) ロミオは気まぐれ（論文中には “fickle” と書かれています）で，ジュリエットがロミオのことを好き/嫌いな量に比例して，ロミオはジュリエットのことが嫌い/好きになります．

これを連立微分方程式で書くと，次のようになります．

$$\left| \begin{array}{l} \frac{\mathrm{d}r}{\mathrm{d}t} = -a \cdot j(t) \\ \frac{\mathrm{d}j}{\mathrm{d}t} = b \cdot r(t) \end{array} \right. \tag{2.1.2}$$

ここで $r(t), j(t)$ は，それぞれ時刻 t でのロミオ，ジュリエットの愛情の強さを表しています．また $a, b(>0)$ は比例定数です．

$a = b = 1$，$r(0) = 1, j(0) = 0$ とすると，連立微分方程式 (2.1.2) の解は $r(t) = \cos t, j(t) = \sin t$ となります．図 2.6 に二人の愛情の変化を感じ取れますか？ □

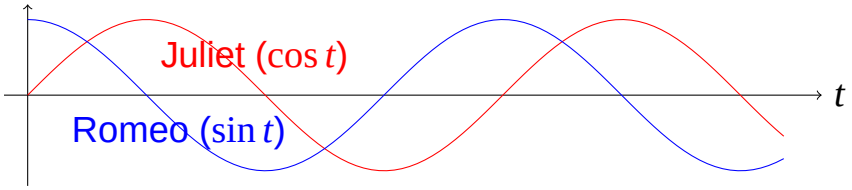

図 2.6: ロミオとジュリエットの感情の変動

例 2.1.1 では，身近にある「待ち」という現象の基本的な性質を簡潔な式で表しています．また例 2.1.2 では，基本的な仮定から，施設の利用者数を推定するためのモデルを構築しています．さらに例 2.1.3 では，生物の体内で起きている現象を数学的に表現することに成功しています（そして例 2.1.4 では恋愛までも数学的に！）．これらは，現実世界のある対象を数学的に表現している「数理モデル」に他なりません．

もちろん，2.1.1 節で説明したように，実際の現象と数理モデルとの間には差があります．しかし，その差が小さければ，数理モデルを通じて実際の現象を理解することが可能になります．

2.2　数理最適化：はじめの一歩

本節では，本書のテーマである「数理最適化」とはどのようなことを指すのかについて説明します．

2.2.1　概念としての最適化問題

ここではまず，「数理最適化」から「数理」を取り，「最適化」という言葉について考えてみます．「最適化」という考え方は社会の様々な場面で用いられています．例えば，

- カーナビで目的地までの一番短い道順を探す
- 最も儲かりそうな株の買い方を考える
- 満足度の最も高いアルバイトのシフトを作成する

などの問題は，全て「最適化」という言葉で表現することができます．

最適化の基本的な考え方は，

> 「可能な選択肢の中から目的が最も良くなる状態を求める」

という言葉に集約できます．そして，そのようなことを考える問題を「**最適化問題**」といいます（なお，この言葉は 2.2.3 節で再定義します）．上記の例では，具体的には，

- 現在地から目的地まで様々な行き方がある中で，最も短い道順を求める
- 様々な株の組合せを考え，過去の株価変動の履歴を考慮して，最も儲かりそうな組合せを求める
- 希望が叶わない日数に偏りがないようなシフトを組む

などということを考えています．日常，このようなことを考えるときに「最適化」や「最適化問題」という言葉を直接用いることは少ないかもしれませんが，これらのことは「最適化」「最適化問題」そのものなのです．

多くの場合，選択肢の範囲には制約が付きます．例えば，上記の例については，

- お金がないので，高速道路は使わない範囲で最も短い道順を探したい
- 手持ちの資金に制限があるので，購入できる株は限られている
- 全員の出勤・休みの希望を満たすことはできない

といった制約が考えられるでしょう．このことを踏まえて上記の言葉を言い換えると，

> 「制約を満たすような状態の中から目的が最も良くなる状態を求める」

ということができます．

2.2.2 数理最適化

「数理」という言葉が名詞に付くと，それは「数学的な」あるいは「数学的な手法を用いた」といった意味になります．つまり，本書で考える「**数理最適化**」とは，「数学的な手法を用いた最適化」という意味です．「数学的な手法」というのはかなり広い意味を持ちますが，本書でいう数学的な手法がどういったことを指すのかは，ページが進むにつれて明らかになってくるでしょう．

数学的な手法を用いて最適化問題を解くためには，最適化問題を数学的に表現する ことが必要になります．これは，前節で説明した「数理モデル」に他なりません．すなわち数理最適化とは，

> 数理モデルを作り，これに数理的な手法を適用することで最適解を求め，さらに得られた解を分析すること

といえます．

数理最適化は，我々が生活する社会において様々な形で役に立っています．その具体例は本書で登場することになります．

Note: 数理モデル 2 種　数理モデルは，シミュレーションに用いられるモデルと最適化に用いられるモデルの 2 種類に大別することができます．シミュレーションモデルは，実世界の現象をコンピュータ上で再現するためのモデルです．多くの場合，時間の経過とともに変化する現象を再現するため，モデル内では（例 2.1.3 の Hodgkin–Huxley モデルのように）微分方程式や差分方程式が用いられます．それに対し最適化モデルでは，実世界の問題においてコントロールできる量を変数とし，（何らかの意味で）最適な状態を導くことができる変数の量を見いだすことを目的としています．

2.2.3　数理モデルとしての最適化問題

2.1.2 節で挙げた数理モデルの例 2.1.1, 2.1.2, 2.1.3 は，実際の現象を理解するためのモデルでした．これに対し，数理最適化では，考察対象の数理モデルを作成することで対象の最適な状態を求めます．このとき，数理最適化のために作成したモデルを**最適化問題**といいます．2.2.1 節では「最適化問題」という言葉の指す範囲を説明しましたが，ここでは具体的な問題を数理モデルとして表現したもの，すなわち数理モデルとして**定式化**したものを「最適化問題」としています．この差が混乱を招くことはほとんどないと思われるため，本書ではこの言葉を両方の意味で使うことにします．

最適化問題の一般的な形は次のように書くことができます．

$$\left|\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in S \end{array}\right. \tag{2.2.1}$$

ここで，$\boldsymbol{x}$ は，問題 (2.2.1) を解いて値を決めるべき**変数**です．また $f : \mathbb{R}^n \to \mathbb{R}$ を**目的関数** (objective function) といい，$\boldsymbol{x} \in S$ を**制約条件**といいます．また，$S \subseteq \mathbb{R}^n$ を**実行可能領域** (feasible region) といい，実行可能領域に含まれる点のことを**実行可能解** (feasible solution) といいます．

問題 (2.2.1) では，目的関数 f を「最小化」する問題を最適化問題として定義していますが，問題によっては（例えば f が何らかの利得を表すような関数の場合）は，f を「最大化」することが目的となります．ここで，「f を最大化する」ということは「$-f$ を最小化する」ということと等価ですので，本書では最小化問題と最大化問題を特に区別せずに説明します（必要に応じて読み替えて下さい）．

またここでは実行可能領域を単に $S \subseteq \mathbb{R}^n$ とだけ書いていますが，（後で見るように）実際には S は関数を用いて与えられることが多くあります．例えば次のような形です．

$$\left|\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & g_i(\boldsymbol{x}) \leq 0 \quad (i = 1, \ldots, m) \\ & h_j(\boldsymbol{x}) = 0 \quad (j = 1, \ldots, l) \end{array}\right. \tag{2.2.2}$$

ここで，$g_i : \mathbb{R}^n \to \mathbb{R}$ $(i = 1, \ldots, m)$, $h_j : \mathbb{R}^n \to \mathbb{R}$ $(j = 1, \ldots, l)$ です．$g_i(\boldsymbol{x}) \leq 0$ $(i = 1, \ldots, m)$ を**不等式制約**といい，$h_j(\boldsymbol{x}) = 0$ $(j = 1, \ldots, l)$ を**等式制約**といいます．もちろん，$g_i(\boldsymbol{x}) \geq 0$ の形の不等式制約もあり得ますが，この場合は両辺を (-1) 倍して「≤ 0」の形になると考えればよいでしょう．このとき，問題 (2.2.1) における S は

$$S := \{\boldsymbol{x} \in \mathbb{R}^n \mid g_i(\boldsymbol{x}) \leq 0 \ (i = 1, \ldots, m),\ h_j(\boldsymbol{x}) = 0 \ (j = 1, \ldots, l)\} \tag{2.2.3}$$

と書くことができます．すなわち，$g_i(\boldsymbol{x}) \leq 0$ $(i = 1, \ldots, m)$, $h_j(\boldsymbol{x}) = 0$ $(j = 1, \ldots, l)$ の全てを満たすような点 $\boldsymbol{x}$ の集合が実行可能領域となります．

Note: 不等式制約で狭義の不等号は使わない 問題 (2.2.2) では不等式制約を定義するのに等式付き不等号 $\leq$ $(\geq)$ を用いています．それでは，等式のない，狭義の不等号 $<$ $(>)$ は使えるでしょうか？

結論としては，使うことはできません．というよりは,「使う意味がない」と言った方がよいかもしれません．仮に次のような問題を考えてみましょう．

$$\begin{array}{ll} \text{minimize} & x \\ \text{subject to} & x > 0 \end{array}$$

この最適化問題では，x が上から（正の値の方から）0 に近づくにつれて目的関数の値がよくなりますが，決して 0 になることはできません．このとき，最適解は理論的には存在しないことになってしまいます．しかし，実際には $x = 0$ が最適解だと考えて差し支え無い場合がほとんどでしょう．この意味で，最適化問題で狭義の不等式が使われることはありません．

Note: 制約に「または (OR)」は導入できるか？ 問題 (2.2.3) で見たように，最適化問題で複数の制約を並べると，制約はすべて「かつ (AND)」で繋がれたものとして扱われます．それでは,「または (OR)」は表現できるのでしょうか？結論から言うと，これは表現可能です．これについては 3.6.4 節で説明します．

最適化問題を取り扱うとき，すなわち「最適化問題に対する解法を構築する」あるいは「具体的な問題を最適化問題に定式化する」際には，問題 (2.2.1) において，目的関数 f や実行可能領域 S の数学的性質に注目することが重要になります．

2.2.4 最適化問題の実行可能性と解の性質

問題 (2.2.1) において，S が空集合になる場合もあり得ます．このような場合は問題が**実行不可能**であるといいます．これに対し S が空集合ではない場合，すなわち実行可能解が存在する場合は**実行可能**であるといいます．

問題が実行可能であるとき，問題の**最適解**は**大域的最適解** (global optimal solution) と**局所的最適解** (local optimal solution) に大別できます．以下，これらについて説明します．

Note: 実行可能だが最適解が存在しない場合とは？　最適化問題では，実行可能であっても最適解が存在しない場合があります．例えば次のような問題です．

$$\left|\begin{array}{ll}\text{minimize} & -x \\ \text{subject to} & x \geq 1\end{array}\right. \tag{2.2.4}$$

$$\left|\begin{array}{ll}\text{minimize} & \frac{1}{x} \\ \text{subject to} & x \geq 1\end{array}\right. \tag{2.2.5}$$

問題 (2.2.4) においては，x を大きな値にすることにより，実行可能性を満たしつつ目的関数値をいくらでも小さくすることができます．従って最適解を定めることはできません．また問題 (2.2.5) で x を大きな値にすると,（目的関数値は下に有界（上から 0 に近づく）であるものの,）目的関数値は小さくなるので，やはり最適解を定めることができません．

問題 (2.2.1) において，ある実行可能解 $\bar{\boldsymbol{x}}$ が，他の実行可能解と比べて目的関数の値を最も小さくする場合[3]，$\bar{\boldsymbol{x}}$ を問題 (2.2.1) の大域的最適解といいます．一方，ある実行可能解 $\hat{\boldsymbol{x}}$ が，$\hat{\boldsymbol{x}}$ のまわり（数学的には「**近傍**」といいます）で目的関数を最も小さくするとき，これを局所的最適解といいます[4]．直感的には，

- 大域的最適解 = 実行可能解の中で最も目的関数の値を小さくする点
- 局所的最適解 = 自分の近くに自分よりも目的関数の値を小さくするような点がない点

と考えてよいでしょう．

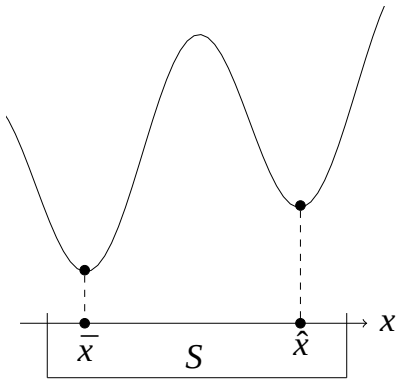

図 2.7: 大域的最適解（図中の $\bar{\boldsymbol{x}}$）・局所的最適解（図中の $\hat{\boldsymbol{x}}$）

[3] より正確には「任意の実行可能解よりも目的関数を大きくしない場合」と言うことになります（目的関数値を最も小さくする実行可能解が複数ある場合を考えています）．数学的には $f(\boldsymbol{x}) \geq f(\bar{\boldsymbol{x}})$ $(\forall \boldsymbol{x} \in S)$ と書くことができます（$\forall$ は「全ての」を意味します）．

[4] 数学的には次のように書くことができます: $\hat{\boldsymbol{x}}$ に対しある $\varepsilon > 0$ が存在し $f(\boldsymbol{x}) \geq f(\hat{\boldsymbol{x}})$ $\forall x \in S \cap \{\boldsymbol{x} \mid \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \leq \varepsilon\}$. 詳しくは [11, 12] を参照してください．

もちろん，局所的最適解よりも大域的最適解の方が優れているわけですが，問題の数学的性質によっては，大域的最適解を求めるのが非常に難しい問題もあります．そのような場合は局所的最適解を求めることになります．一方，問題の数学的性質がよいと，局所的最適解が大域的最適解になるような（局所的最適解が大域的最適解と一致するような）問題もあります．これらのことについては次節で触れます．

2.2.5 最適化問題の分類

最適化問題は，その数学的性質によって解法や解の分析のアプローチが異なります．逆に，最適化モデルを作成する際には，そのモデルがどのような問題のクラスに属するのかを意識する必要があります．そこで本節では，最適化問題を数学的な特徴に注目して分類することにします．

一般に，分類したそれぞれの問題クラスには**標準形** (standard form) があります．ある最適化問題 P が問題クラス X の標準形に帰着できる（変形できる）のであれば，問題 P は問題クラス X に属す，ということになります．ここでは，基本的な最適化問題のクラスの標準形を説明します．

線形計画問題

最適化問題の最も代表的なクラスは**線形計画問題** (linear programming problem, LP) です．線形計画問題とは，

- 目的関数: 1 次関数
- 制約条件: 1 次の等式・不等式

であるような最適化問題のことを言います．線形計画問題の標準形は以下のようになります．

$$\left|\begin{array}{ll} \text{minimize} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0} \end{array}\right. \tag{2.2.6}$$

ここで，$\boldsymbol{c} \in \mathbb{R}^n, \boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{b} \in \mathbb{R}^m$ であるものとします．これらは問題を特徴付ける**パラメータ**です．等式制約 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ は，このように書くと 1 つの制約のようにも見えますが，実際は m 個の制約 $\boldsymbol{a}_i^\top \boldsymbol{x} = b_i\ (i = 1, \ldots, m)$ であることに注意してください（$\boldsymbol{a}_i$ は行列 $\boldsymbol{A}$ の i 行目を縦ベクトルにしたもの）．同様に不等式制約 $\boldsymbol{x} \geq \boldsymbol{0}$ は $x_j \geq 0\ (j = 1, \ldots, n)$ という意味になります．なお，この制約は変数の値を非負の領域に限定するものなので，**非負制約**といいます．

ここで，A.3.1 節で述べたように，問題 (2.2.6) の実行可能領域 $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}$ は凸集合になることに注意してください．

上で説明したように，任意の LP はこの標準形 (2.2.6) で表現することができます．しかし，ソフトウェアを用いて LP（や他の問題）を解く際には，通常，標準形のことを気にする必要はありません．ですので，ここではその方法については触れないことにします[5]．

2.2.4 節では，最適化問題の解として大域的最適化と局所的最適解があることを説明しました．LP では，局所的最適解は必ず大域的最適解になります（局所的最適解でありながら大域的最適解でない，ということはありません）．すなわち，ソルバを用いて LP を解いて得られる解は大域的最適解になります．

[5]その方法については [12] などを参照してください．

2 次計画問題

LP において，目的関数が 2 次関数に代わると，2 **次計画問題** (quadratic programming problem, QP) となります．すなわち，

- 目的関数: 2 次関数
- 制約条件: 1 次の等式・不等式

であるような問題が 2 次計画問題です．標準形は以下の通りです．

$$\left|\begin{array}{ll} \text{minimize} & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0} \end{array}\right. \tag{2.2.7}$$

ここで，パラメータは $\boldsymbol{Q} \in \mathbb{R}^{n\times n}, \boldsymbol{c} \in \mathbb{R}^n, \boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{b} \in \mathbb{R}^m$ であるものとします．

特に問題 (2.2.7) において $\boldsymbol{Q}$ が半正定値行列である場合，目的関数が凸関数となることから，これを**凸 2 次計画問題** (convex quadratic programming problem, convex QP) といいます．単なる「2 次計画問題」と「凸 2 次計画問題」は厳密には区別されるべきものですが，凸 2 次計画問題には（その凸性を利用した）効率的な解法が複数提案されていることもあり，単に「2 次計画問題」と表記してある場合でも実際には「凸 2 次計画問題」のことを指していることが多くあります．本書でも，特に断りなく「2 次計画問題」と表記した場合には,「凸 2 次計画問題」のことを指すものとします．

凸 2 次計画問題は，LP 同様，局所的最適解であれば必ず大域的最適解になります．しかし，凸ではない 2 次計画問題では，局所的最適解が大域的最適解にならないことがあり得ます．従って，問題を解く際には，問題（の目的関数）が凸性を持つかどうかを検討しておく必要があります．

非線形計画問題

目的関数や制約条件を構成する等式・不等式に非線形関数が含まれるような最適化問題を**非線形計画問題** (nonlinear programming problem, NLP) といいます（すなわち，2 次計画問題は非線形計画問題の 1 つです.）．非線形計画問題の標準形は以下の通りです．

$$\left|\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & g_i(\boldsymbol{x}) \leq 0 \quad (i = 1, \ldots, m) \\ & h_j(\boldsymbol{x}) = 0 \quad (j = 1, \ldots, l) \end{array}\right. \tag{2.2.8}$$

ここで，$g_i : \mathbb{R}^n \to \mathbb{R}\ (i = 1, \ldots, m),\ h_j : \mathbb{R}^n \to \mathbb{R}\ (j = 1, \ldots, l)$ であるものとします．

問題 (2.2.8) ですが，実は問題 (2.2.2) と全く同じ形であることに気づいたでしょうか？問題 (2.2.2) は「最適化問題の一般的な形」として紹介したわけですが，NLP は最も一般的な最適化問題であるため，このようなことになります．

非線形計画問題では，文字通り，$f, g_i\ (i = 1, \ldots, m), h_j\ (j = 1, \ldots, l)$ が非線形関数であって構いません．例えば三角関数や $\log, \exp$ などを使うことができます．さらには，微分不可能な関数（例えば $f = |x|$ のような場合）や不連続な関数が含まれる場合も，NLP に分類されます．ただ，そのような問題を取り扱うことができる解法（アルゴリズム）は限定的ですので，問題に含まれる関数は連続微分可能なものに限定されることになるでしょう．

非線形計画問題では，局所的最適解が大域的最適解にならないことがあり得ます．従って，ソルバを用いて非線形計画問題を解く際には，その解が大域的最適解ではない可能性があることに注意しなくてはなりません．

凸計画問題

ここまでに見てきた LP, QP, NLP では，問題によって解の性質が次の 2 通りに分かれていました．

- 局所的最適解であれば大域的最適解である（LP, 凸 QP）
- 局所的最適解であっても大域的最適解でない場合がある (NLP)

実は，この二つを隔てているのは問題の凸性です．目的関数が凸関数であり，実行可能領域が凸集合であるような問題を**凸計画問題**といいます．凸計画問題は，次のよい性質を持つことが知られています．

定理 2.2.1 凸計画問題の局所的最適解は必ず大域的最適解である． □

これまでに説明した問題の解の性質が，定理 2.2.1 に従っていることを確認してください．注意したいのは，非線形計画問題であっても凸計画問題であることはある，ということです．そのような場合は，非線形計画問題であっても，問題を解けば大域的最適解が得られる，ということになります．

（混合）整数計画問題

さて，ここまでは最適化問題に含まれる関数の数学的性質（1 次関数，2 次関数，非線形関数，凸性）に注目して問題を分類してきましたが，ここでは問題に含まれる変数の性質に注目します．

ここまで紹介してきた問題クラスでは，暗黙のうちに，変数の値が**連続的**に変わりうるものを想定してました.「連続的」というのは，ここでは,「変数が任意の実数値を取りうる」（もちろん制約条件を満たさなくてはなりませんが）と解釈してください．このような変数のことを**連続変数**といい，全ての変数が連続変数であるような最適化問題を**連続計画問題**といいます．

これに対し，最適化問題では，値が**離散的**であるような変数を考えることができます.「離散的」というのは，一般には「とびとびである」ことを言いますが，ここでは「変数の値が整数値に限定される」という意味です．このような変数のことを**整数変数**といいます．

変数が（全て）整数変数であるような最適化問題を**整数計画問題**といいます．これは前項までに挙げた問題クラスと複合します．例えば線形計画問題において変数が整数変数である場合，その問題は**整数線形計画問題**といいます．同様に，**整数 2 次整数計画問題**，**整数非線形計画問題**という問題クラスを考えることができます．

ここで次の例題を考えてみましょう．

問題 2.2.1 2 種類の製品 A, B を製造している企業があります．従業員は 3000 人おり，A, B を 1 単位製造するのに必要な従業員数はそれぞれ 600 人，500 人です．また A, B の需要はそれぞれ 4 単位あります．さらに，A, B を 1 単位販売すると，それぞれ 11, 10 億円の利益が発生します．このとき，この企業は A, B をそれぞれ何単位製造すべきでしょうか？ □

解説 問題 2.2.1 を素直に定式化すると，次のようになります（なお，ここでは定式化の詳細はわからなくても構いません．定式化の詳細については 3 章で学びます）．

$$
\left|\begin{array}{ll}
\text{maximize} & 11x_A + 10x_B \\
\text{subject to} & 600x_A + 500x_B \leq 3000 \\
& 0 \leq x_A \leq 4 \\
& 0 \leq x_B \leq 4
\end{array}\right. \tag{2.2.9}
$$

ここで変数 x_A, x_B は製品 A, B の製造単位数を表しています.

問題 (2.2.9) は，目的関数・制約条件とも 1 次関数で表現されていますので，これは線形計画問題になります．例えば製品 A, B が種類の異なる飲料であれば，これを解けばいいでしょう（このときの製品 1 単位は例えば 「100t」になるでしょう）．飲料は「連続的」なものであるので，連続変数で表現するのに適しています．あるいは，離散的なものであっても，量が多量であるため事実上連続的なものであると見なせる場合（例えばコーヒー豆）は，やはり連続変数で表現すればよいでしょう.

しかし，これが製品 A, B が航空機であればどうでしょうか？「1.5 機」販売するということはできないわけですから，変数の値を整数値に限定する必要があるでしょう（単位は「機」になります）．すなわち，問題は次のように定式化できます.

$$\left|\begin{array}{ll} \text{maximize} & 11x_A + 10x_B \\ \text{subject to} & 600x_A + 500x_B \leq 3000 \\ & 0 \leq x_A \leq 4 \\ & 0 \leq x_B \leq 4 \\ & x_A, x_B \text{は整数} \end{array}\right. \tag{2.2.10}$$

これは変数が整数値に限定されており，目的関数・制約条件が 1 次関数であることから，整数線形計画問題になります.

ここで，問題 (2.2.9), (2.2.10) の最適解は次のようになります（もちろんソルバを用いて求めた最適解です）.

	x_A	x_B
(2.2.9)	1.66667	4
(2.2.10)	4	1

注意すべきは，問題 (2.2.9) の最適解を整数値に丸めても（近くの整数値を採用することにしても），問題 (2.2.10) の最適解にはならない，ということです．このようなことが起きうるため，整数計画問題を考える必要があります. □

さらに，連続変数と整数変数の両方を同時に持つ最適化問題を考えることもできます．その場合は，元の問題クラスに「混合整数」を付けます．すなわち，**混合整数線形計画問題**，**混合整数 2 次計画問題**，**混合整数非線形計画問題**などとなります.

整数変数の一つの使い方は，対象を数えるために使うというものです．従って，多くの場合，整数変数には非負制約が付きます．また，別の使い方としては，値を 0 か 1 に限定し，OFF/ON を表すある種の「スイッチ」のように使うことがあります（具体例は 3 章で見ることになります）．このような整数変数を特に 0-1 **変数**または**バイナリ変数**といいます．また，変数がすべて 0-1 変数であるような最適化問題を 0-1 **計画問題**といいます.

整数計画問題の場合の解の大域性/局所性は，問題の整数性を除いた問題クラスに準じます．例えば「整数線形計画問題」であれば（「整数」を取って）「線形計画問題」と同じ，すなわち局所的最適解であれば大域的最適解である，ということになります.

ここまでに出てきた最適化問題のクラスを 2.1 にまとめました．なお，問題の略称にカッコが付いているものは，あまり使われることはありません.

表 2.1: 最適化問題のクラス

問題略称	問題名（英語）	問題名（日本語）	目的関数	制約条件	変数
LP	Linear Prog. Prob.	線形計画問題	線形	線形	連続
QP	Quadratic Prog. Prob.	2 次計画問題	2 次	線形	連続
NLP	Nonlinear Prog. Prob.	非線形計画問題	非線形	非線形	連続
IP	Integer Prog. Prob.	整数計画問題	—	—	整数
(ILP)	Integer Linear Prog. Prob.	整数線形計画問題	線形	線形	整数
(IQP)	Integer Quadratic Prog. Prob.	整数 2 次計画問題	2 次	線形	整数
(INLP)	Integer Nonlinear Prog. Prob.	整数非線形計画問題	非線形	非線形	整数
MILP	Mixed Integer Linear Prog. Prob.	混合整数線形計画問題	線形	線形	連続/整数
MIQP	Mixed Integer Quadratic Prog. Prob.	混合整数 2 次計画問題	2 次	線形	連続/整数
MINLP	Mixed Integer Nonlinear Prog. Prob.	混合整数非線形計画問題	非線形	非線形	連続/整数

※ Prog. = Programming, Prob. = Problem

数理最適化？数理計画法？ 「数理最適化 (Mathematical Optimization)」のことを「数理計画（法）(Mathematical Programming)」ということがあります.「最適化」と「数理計画」，前者に比べて後者の方が堅い言葉だなぁ，と感じる方が多いのではないでしょうか．これに関して，次のような話題があります: 2010 年,「数理計画」に関する国際的に最大の学会である Mathematical Programming Society (MPS) が，Mathematical Optimization Society (MOS) に改名するという出来事がありました.「最適化」という言葉は他分野も含めて広く使われているが,「数理計画」という言葉はそうではない，というのが理由だそうです．本書もそれに倣い,「（数理）最適化」という言葉を用いることにします．一方，この変更に伴って少し困ることも出てきました．それは問題クラスの名前です．例えば「線形 計画 問題 (Linear Programming Problem)」のように，問題名に「計画」という言葉が入っているのです．これを「線形 最適化 問題」とすると，略称が 'LP' から Linear Optimization Problem = 'LO' となってしまい，広く使われてきた略称が変わってしまう，ということがあります．本書執筆時点（2014 年）では,『略称については変えない派』が主流ではないかと思いますが…さてどうなるか？

第3章　最適化問題の「モデル化」と「定式化」

本章で学ぶ内容

- 最適化問題のモデル化
- 最適化問題の定式化（モデル化との違い）
- 定式化の技法（基本）
- 定式化の技法（応用）
- よく知られた最適化問題とその定式化

この章では，最適化問題のモデル化と定式化の方法について学びます.

3.1　「モデル化」と「定式化」の差は？

ここでは，最適化問題の「モデル化」と「定式化」の差について考えます.

これらの言葉は混同されることも多いのですが，本書では以下のように定義し，一応の区別をつけておきたいと思います：

- 「モデル化」: 考察対象の目的や制約を分析し,「定式化」を意識しながら整理すること
- 「定式化」: モデル化で整理された目的や制約を最適化問題として数学的に表現すること

「一応」としたのは,「モデル化」の定義の中に「定式化」という言葉が出てくることからわかるように，両者は連動するからです（それが言葉の混同の一因になっているものと思われます）.

ここでは，有名な巡回セールスマン問題を例にとり,「モデル化」について見てみましょう.

問題 3.1.1 巡回セールスマン問題

巡回セールスマン問題 (Traveling Salesman Problem, TSP) とはつぎのような問題です [4].

> あるセールスマンが，本社のある都市を出発し，指定された都市をセールスのために訪問し，本社まで帰ってこなくてはなりません．都市の訪問順番は指定されていないため，そのセールスマンは「本社を出発して，できるだけ短い移動距離で全ての訪問都市を回り，また本社に帰ってこよう」と考えました．手元には，全ての訪問都市間（本社含む）の距離一覧があります．さて，どのような経路で全ての都市を回ればいいか，計画を立ててください.

ここでは，巡回すべき各都市（例えば図 3.1）を節点とするグラフを考えます（グラフはモデル化の有力なツールの一つです）．都市の訪問順番は指定されていないため，どの都市からどの都市へも行くことができるものと考えられます．つまり，考えるべきグラフは完全グラフになります（図 3.2. 枝の重みは都市間の距離になります）．そして，このグラフの上で，各節点をちょうど一度ずつ通過する最短の単純閉路を求めることになります（図 3.3）．これにより,「巡回セールスマン問題」は「完全グラフ上で全節点を通過する最短単純閉路を求める問題」と等価であることがわかりました．これはモデル化に他なりません．　□

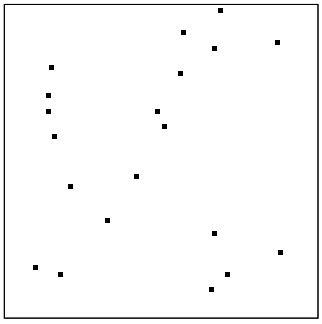

図 3.1: 都市の配置例

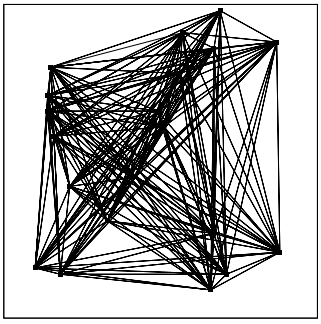

図 3.2: 想定する完全グラフ

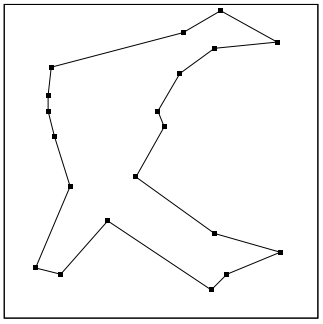

図 3.3: TSP の解答

すなわち，

- 巡回セールスマン問題での「都市」↔ （完全）グラフの「節点」
- 「都市間の距離」↔ 「枝の重み」
- 「都市をちょうど一度ずつ訪問し元の都市に戻る」↔ 「全節点を通過する単純閉路」

という対応を見い出して抽象化することにより，巡回セールスマン問題を（完全）グラフ上の問題に帰着させたわけです．

一般に，最適化問題が与えられたとき，直ちに定式化可能である（と感じられる）場合と，そうではない場合があります．前者の場合は，問題が既にモデル化されていると言えます．そのような場合ばかりだと助かるのですが，残念ながらそうはいきません．新しく取り組む場合は，後者であることがほとんどでしょう．そのような場合には，問題が持つ構造を見いだし，モデル化することが必要になります．そして，そのようなモデル化を行う際には，定式化のことを頭に置いておくことが大事です．様々な定式化の技法を知っていれば，自ずと最適化問題として扱うことができる対象を広げることにつながります．次節以降では，定式化の基本的な技法やよく出てくる表現方法について見ていきます．

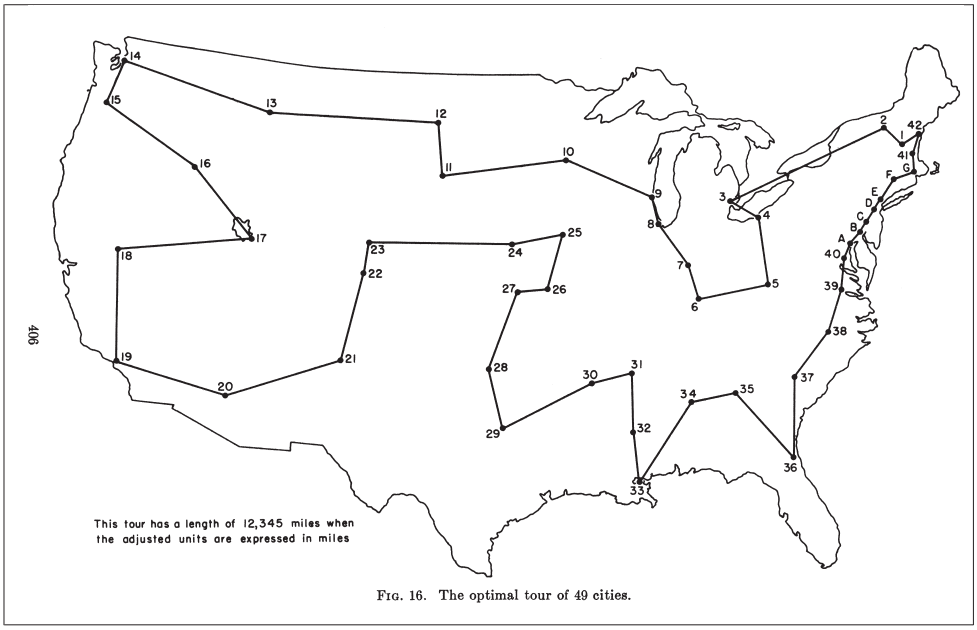

FIG. 16. The optimal tour of 49 cities.

図 3.4: アメリカの各州を対象とした TSP の解答 [8]

巡回セールスマン問題の古典的な論文では，アメリカの 48 州とワシントン D.C. の 49 都市を巡る巡回セールスマン問題を解いています．この論文では，TSP に対する厳密解法（切除平面法に基づく手法）が初めて提案されています．

Note: 問題の特徴を損なわないモデル化を　2 章でも触れましたが，問題をモデル化する際には，問題の特徴を捉えたモデルを作る必要があります．一方で，ソルバで扱うことができる最適化問題のクラスには限りがあります．問題の特徴を損なわない範囲でソルバで扱うことが最適化問題として定式化できればよいのですが，必ずしもそうならない場合もあるでしょう．そのような場合は，少なくともソルバで扱うことを断念せざるを得ません（ただし，7.5 節で紹介するようなアプローチはあり得ます）．間違っても，問題の本質を損なう形で定式化してはいけません．それは元の問題を解いたことにはならないのです！

3.2　定式化・基礎 (1): 和を取る

定式化の基本中の基本は「和を取る」ことです.「和を取る」とだけ聞くと単純なようにも思いますが，これには様々なパターンが考えられます．ここでは，それらのパターンをいくつか見ていきましょう．

3.2.1　添字を使って和を取る

ここでは次のような設定を考えます.

設定 3.2.1 あるメーカーは 3 つの工場を持っており，各工場では同一の製品を作っています．また，この製品を購入する会社が 4 つあるものとします．このとき，各工場の生産量の上限と製品を購入する各会社の発注量が表 3.1 のように与えられており，メーカー側は各会社に発注された分を納品しなくてはなりません．□

表 3.1: 各工場の生産量の上限，各会社の発注量（単位: トン）

工場	1	2	3
生産量の上限	25	20	35

会社	1	2	3	4
発注量	15	20	18	22

以下，この設定を定式化することを考えます.

定式化のための第一歩は，既知の量と未知の量を区別することです.既知の量は**パラメータ**となり，未知の量は最適化問題を解いて決定する**変数**として表現することになります．ここでは，上記文中に現れる「各工場の生産量の上限」「製品を購入する各会社の発注量」は既知の量，すなわちパラメータとして扱うことができます．一方，未知の量は文中には明示的には現れていませんが,「メーカー側は各会社に発注された分を納品しなくてはならない」という一文から,「各工場から各会社への輸送量を決めなくてはならない」ということを読み取る必要があります（実際には，問題を整理していく中でこのことに気づく必要があります）.

さて，ここでは各工場から各会社への輸送量を変数 x で表すことにします．ここで注意すべきことは,「各工場から各会社への」輸送量を考えていることです．単に x とだけすると，これは一つの量しか表すことができませんので，例えば「工場 1 から会社 1 への輸送量」と「工場 2 から会社 3 への輸送量」を区別することができません．そこで **添字** の出番です．変数の右下に小さい文字を付

すことで，同じ種類の量（ここでは「輸送量」）を表しながらも，その中での区別ができるようにします[1]．ここでは，添字として「工場」を表す添字と「会社」を表す添字の 2 つが必要になります．添字を考える際には，**集合** という概念とセットにする必要があります．要素の集まりが集合ですが，添字は集合内の要素を表す文字として利用するわけです．この問題では，集合として

- 工場の集合 $I = \{1, 2, 3\}$
- 会社の集合 $J = \{1, 2, 3, 4\}$

を考え，これらの集合の要素をそれぞれ i, j で表すことにします[2]．すると,「各工場から各会社への輸送量」は x_{ij} $(i \in I, j \in J)$ と書くことができます．

同様にパラメータも文字を用いて表現することにします．ここでは,「各工場の生産量の上限」を u で,「製品を購入する各会社の発注量」を d で表現することにします．u は各工場のことを表すわけですから，やはり添字が必要になります．変数の添字を考える際に，工場のことは添字 $i \in I$ で表現することにしましたので，ここでもそれに合わせる必要があります．すなわち，各工場の生産量の上限は u_i $(i \in I)$ とします．同様に d は各会社のことを表すので，添字 j を用いて d_j $(j \in J)$ とします．

さて，それでは上記の設定に現れる条件を定式化してみましょう．まず，メーカー側が各会社の発注量を満たすように輸送しなくてはなりません．この条件を定式化してみます．会社 1 の発注量について考えると，各工場から会社 1 に輸送する量の総和が 15 と等しくなくてはいけません（多く輸送してもいけませんし，足りなくてもいけません）．これは

$$x_{11} + x_{21} + x_{31} = 15 \tag{3.2.1}$$

と表現することができます．変数 x の 2 つの添字のうち，工場を表す前の添字が $1, 2, 3$ と変わっているのに対し，会社を表す後ろの添字は 1 のまま変わっていないことに注意してください．同様に会社 $2, 3, 4$ の発注量についても

$$x_{12} + x_{22} + x_{32} = 20 \tag{3.2.2}$$
$$x_{13} + x_{23} + x_{33} = 18 \tag{3.2.3}$$
$$x_{14} + x_{24} + x_{34} = 22 \tag{3.2.4}$$

と書くことができます．(3.2.1), (3.2.2), (3.2.3), (3.2.4) により各工場からの輸送量と各会社の発注量との関係を定式化することができました．

しかし，このやり方では，発注する会社が増える度に制約式を増やさなくてはなりません．またメーカーが工場を増設した場合には，左辺に現れる変数が増えることになります．これではいかにも不便です．そこで (3.2.1), (3.2.2), (3.2.3), (3.2.4) の右辺をパラメータ d を用いて書き換えてみます．すると

$$x_{11} + x_{21} + x_{31} = d_1 \tag{3.2.5}$$
$$x_{12} + x_{22} + x_{32} = d_2 \tag{3.2.6}$$
$$x_{13} + x_{23} + x_{33} = d_3 \tag{3.2.7}$$
$$x_{14} + x_{24} + x_{34} = d_4 \tag{3.2.8}$$

[1] 添字を右上や左下，左上に添字を付す流儀もありますが，右下が一番標準的な位置だと思います．
[2] 通常，集合は大文字で書き，その要素は小文字で書くことが多いです．

となります．これをよく観察すると，これらは共通の構造を持っていることが分かります．すなわち，会社 j のことを考えているのであれば (3.2.5), (3.2.6), (3.2.7), (3.2.8) は

$$x_{1j} + x_{2j} + x_{3j} = d_j \quad (j \in J)$$

とすることができます．ここで「$j \in J$」とは，各工場 j について共通に成立するということを表しています．さらに左辺に注目すると，これは各工場から会社 $j \in J$ への輸送量ですから，和の記号 Σ を用いると，これは

$$\sum_{i \in I} x_{ij} = d_j \quad (j \in J) \tag{3.2.9}$$

と表すことができます．Σ の下にある「$i \in I$」は，集合 I に含まれる要素 i に対して和を取る，ということです．ここでは工場が $1, 2, 3$ の 3 つですから，(3.2.9) を

$$\sum_{i=1}^{3} x_{ij} = d_j \quad (j \in J) \tag{3.2.10}$$

としても同じことを表します．しかし，この書き方には次のような問題があります．

- (3.2.10) では，工場の数が増える度に Σ の上の数字を書き換える必要があります．問題の制約が (3.2.10) だけならばまだしも，通常はこの他にも多くの制約があることも考えられ，そのような場合はこれは面倒な作業になります．一方 (3.2.9) では，集合 I の内容を書き換えれば，それだけで済みます．
- 現在は集合 I の要素である工場が数字で表現されていますが，$I = \{$東京工場, 大阪工場, 福岡工場$\}$ などのように I の要素が文字で表現される場合もあり得ます．このような場合は，(3.2.10) のような書き方をすることはできません．

このような点を踏まえ，本書では（特に断らない限り）(3.2.9) のような和の表記をすることにします．

同様に各工場の生産量の上限についても考えてみましょう．ここで定式化すべきことは，生産量の上限を超えて各工場に向けて出荷することはできない，ということです．このことを添字を使わずに表現すると，

$$x_{11} + x_{12} + x_{13} + x_{14} \leq 25$$
$$x_{21} + x_{22} + x_{23} + x_{24} \leq 20$$
$$x_{31} + x_{32} + x_{33} + x_{34} \leq 35$$

となります．上で見たように，この書き方は冗長です．パラメータ u_i $(i \in I)$ を用いて整理すると，これらは

$$x_{i1} + x_{i2} + x_{i3} + x_{i4} \leq u_i \quad (i \in I)$$

とでき，さらに Σ を用いて左辺を表記すると

$$\sum_{j \in J} x_{ij} \leq u_i \quad (i \in I) \tag{3.2.11}$$

となります．

定式化とパラメータの分離

(3.2.1), (3.2.2), (3.2.3), (3.2.4) では各会社の発注量を式の中に直接書き下していましたが，このやり方は上で見たように制約式をまとめて記述するのに不向きですし，発注量が変わったときに式中の値を書き直す必要があるため不便です（複数の制約に渡ってこの値が現れている場合には，その全てを修正する必要があります）．そこで，(3.2.5), (3.2.6), (3.2.7), (3.2.8) で行ったように，定式化中には（できるだけ）パラメータの値を記述しないようにしておくと，同じ枠組みの問題を共通して表現することができます．これを「**定式化とパラメータの分離**」などということがあります．

3.2.2 0-1 変数を使って必要なものの和を取る

ここでは，次の設定を定式化することを考えましょう：

設定 3.2.2 財布に入っている 1,000 円で，野菜・果物・菓子を買います．ただし野菜・果物・菓子それぞれに 500 円以上つかってはいけません．店で売っている野菜・果物・菓子の種類と値段は表 3.2 の通りです．各商品を買うか買わないかを決めたいのですが… □

表 3.2: 野菜・果物・おかしの種類と値段

野菜	値段
キャベツ	200
にんじん	130
ピーマン	180
じゃがいも	200
玉ねぎ	150

果物	値段
りんご	130
みかん	240
いちご	300
もも	350

菓子	値段
チョコ	130
クッキー	150
ポテトチップス	160
あめ	160
ガム	120

前項で述べたように，定式化のための第一歩は既知の量と未知の量を区別することです．ここでの既知の量は

- 各商品の値段
- 各品目内での購入金額の上限（500 円）

です．一方，未知の量は

- 各商品を **買うか買わないか**

です．ここで注意したいのは，前項での変数は「各工場から各会社までの輸送量」という **量** を表す変数であったのに対し，ここでは「買うか買わないか」という **選択** を表す変数が必要になる，ということです．このような選択を表すための変数として，最適化問題では 0-1 **変数** とよばれる変数を用いることが可能です．これは，変数が値として 0 もしくは 1 しか取らないような変数です．0-1 変数により，様々な組合せ的事象を定式化できるようになります（そのような例をこれからたくさん見ていくことになります）．

さて，この問題では品目毎での購入金額に上限がありますので，これを考えるために

- 品目の集合 $I = \{$ 野菜, 果物, 菓子 $\}$

を準備します．さらに，この品目毎に商品の集合がありますので，

- 野菜の集合 $J_{\text{野菜}} = \{$ キャベツ, にんじん, ピーマン, じゃがいも, 玉ねぎ $\}$
- 果物の集合 $J_{\text{果物}} = \{$ りんご, みかん, いちご, もも $\}$
- 菓子の集合 $J_{\text{菓子}} = \{$ チョコ, クッキー, ポテトチップス, あめ, ガム $\}$

という集合を定義します．

さて，今回の設定に現れるパラメータは

- 品目 $i \in I$ の商品 $j \in J_i$ の値段: p_{ij}
- 各品目の購入金額の上限 u
- 購入金額全体の上限 v

とできます．ここで注意したいのは，これらのパラメータの添字です．まず，品目 $i \in I$ の商品 $j \in J_i$ の値段を p_{ij} としましたが，このように書くことができるのは，各品目の商品の集合を，共通の文字 J とそれに付す添字 $i \in I$ で表現しているからです．このように，添字の付いた集合を用いると，定式化がやりやすくなる場合があります．もし,「野菜の集合 J」「果物の集合 K」「菓子の集合 L」などと文字を変えて表現した場合，このように共通の文字 p で商品の値段を表現することはできません．一方，各品目の購入金額の上限は u で，購入金額全体の上限は v で表すことにしています．これらの文字には添字がありません．u については，いずれの品目についても購入金額の上限は共通（500円）しているため，区別する必要がありません．もし，品目毎に購入金額の上限が変わるようなことがあれば，添字 $i \in I$ を付すことになるでしょう．また v については全体で一つだけの制約ですから，これについても添字を付す必要はありません．

次に変数を考えます．上で述べたように，ここでは購入の選択（買うか買わないか）を表す 0-1 変数 x を導入します．すなわち，

$$x_{ij} = \begin{cases} 1, & \text{品目 } i \in I \text{ の商品 } j \in J_i \text{ を購入する} \\ 0, & \text{品目 } i \in I \text{ の商品 } j \in J_i \text{ を購入しない} \end{cases}$$

という変数を準備します．

ここからは，各品目内での商品購入金額について考えます．例えば，野菜のうち，キャベツに関する購入金額は，

$$\text{キャベツに関する購入金額} = \begin{cases} 200\,(\text{円}), & \text{キャベツを購入する} \\ 0\,(\text{円}), & \text{キャベツを購入しない} \end{cases}$$

となります（$p_{\text{野菜, キャベツ}} = 200$ に注意してください）．これは，先ほど準備した 0-1 変数を用いると，

$$\text{キャベツに関する購入金額} = p_{\text{野菜, キャベツ}} \cdot x_{\text{野菜, キャベツ}}$$

とすることができます（4.1.3 節でも説明しますが，モデルにはいわゆる `if` 文を用いることはできません！）．そしてこの表現を野菜の各商品に適用すると，

$$\begin{aligned} \text{野菜の購入金額} = & p_{\text{野菜, キャベツ}} \cdot x_{\text{野菜, キャベツ}} + p_{\text{野菜, にんじん}} \cdot x_{\text{野菜, にんじん}} + p_{\text{野菜, ピーマン}} \cdot x \\ & + p_{\text{野菜, じゃがいも}} \cdot x_{\text{野菜, じゃがいも}} + p_{\text{野菜, 玉ねぎ}} \cdot x_{\text{野菜, 玉ねぎ}} \end{aligned}$$

とすることができます．さらに前項で学んだ，添字による和の表現を用いると，これは

$$\text{野菜の購入金額} = \sum_{j \in J_{\text{野菜}}} p_{\text{野菜, j}} \cdot x_{\text{野菜, j}}$$

とできます．さらに，新たに，品目 $i \in I$ の購入金額を表す変数 y_i を導入すると，

$$y_i = \sum_{j \in J_i} p_{ij} x_{ij} \quad (i \in I)$$

とすることができます．これを用いると，各品目の購入金額の上限に関する制約と，全体での購入金額での上限はそれぞれ

$$y_i \leq u \quad (i \in I) \tag{3.2.12}$$

$$\sum_{i \in I} y_i \leq v \tag{3.2.13}$$

ここで注意したいのは，式に添字が付く場合と付かない場合があるということです．(3.2.12) の左辺に注目すると，添字 i が付された変数 y_i が現れています．これは式全体が i で添字付けされていることに他なりません．一方，(3.2.13) の左辺にも y_i が現れていますが，これについては添字 $i \in I$ について和を取っていますので，結果として左辺は添字付けされていないことになります．また右辺も添字付けされていませんので，結果として (3.2.13) は添字付けされないことになります．

3.3 定式化・基礎 (2): 基本的な表現

3.2 節で述べたように,「和を取る」というのは定式化の基本です．ベンチマーク問題集 MIPLIB2010 [35]（下記 **Note** も参照のこと）では，各問題の制約条件に現れる和の形を分類しています．ここでは，そのパターンを通じて目的関数・制約条件に現れる基本的な表現を学びます [3].

Note: ベンチマーク問題集 MIPLIB　一般に,「ベンチマーク」とは比較のための指標となるもののことを言います．数理最適化においては，複数のソルバの性能を比較するための「ベンチマーク」問題集があります．その中で最も有名なのは，混合整数計画問題 (MIP) のためのベンチマーク問題集 MIPLIB でしょう．MIPLIB のサイトに記載があるように，最初の問題集は 1992 年に作成され，何度かの改変を経て，2014 年現在は MIPLIB2010 が最新の問題集になります．MIPLIB2010 には様々なパターンの問題が総計 361 問収録されています.（※ MIPLIB については 7.2.1 節でも触れます.）

なお，本節での制約式の説明においては，x が変数を表し，それ以外の文字はパラメータを表しています.

3.3.1 基本パターン

ここで取り上げる制約は最も典型的な制約と言ってよいでしょう.

[3] なお，MIPLIB2010 で紹介されている形を少しアレンジして紹介しているものがあります.

変数の上下限 (Variable Bound)

これは，変数の上下限を定める制約です．

$$x_i \leq a_k x_k + b \text{ or } x_i \geq a_k x_k + b, \quad x_i, x_k : \text{整数変数 または 連続変数}, \ a_k, b \in \mathbb{R} \tag{3.3.1}$$

変数 x_i の上下限を一次式 $a_k x_k + b$ で与えています．上下限が変数 x_k の値に連動することに注意してください．このとき $a_k = 0$ とすると，

$$x_i \leq b \text{ or } x_i \geq b$$

となり，これは一番シンプルな形の上下限制約になります．

個数 (Cardinality)

これは，0-1 変数のうち，値が 1 となるような変数の個数[4] を定める制約です．

$$\sum_{i \in I} x_i = b, \quad x_i : \text{0-1 変数}, \ b \in \mathbb{N} \tag{3.3.2}$$

(3.3.2) を設けることにより，値が 1 になる x_i の個数をちょうど b 個に制約することができます．なお，値が 0 になる x_i の個数をちょうど b 個になるように制約するには

$$\sum_{i \in I} (1 - x_i) = b, \quad x_i : \text{0-1 変数}, \ b \in \mathbb{N} \tag{3.3.3}$$

あるいは

$$\sum_{i \in I} x_i = |I| - b, \quad x_i : \text{0-1 変数}, \ b \in \mathbb{N} \tag{3.3.4}$$

とすればよいでしょう．

集約 (Aggregation)

2 つの変数をパラメータで重み付けした和（これを線形和といいます）があるパラメータと等しくするための制約です．

$$a_i x_i + a_k x_k = b, \quad x_i, x_k : \text{整数変数 または 連続変数}, \ a_i, a_k, b \in \mathbb{R} \tag{3.3.5}$$

(3.3.5) では 2 つの変数の場合を扱っていますが，より一般的に

$$\sum_{i \in I} a_i x_i = b, \quad x_i : \text{整数変数 または 連続変数}, \ a_i \in \mathbb{R} \ (i \in I)$$

などとすることもできるでしょう．

[4] “cardinality” は通常「基数（= 集合の要素の個数）」「濃度」と訳しますが，ここでは直感的に意味が伝わるように「個数」としました．

3.3.2 集合に関するパターン

ここでは，集合に関するいくつかの制約を紹介します．

$$\sum_{i\in I} a_{ij}x_i = 1\ (j \in J), \quad x_i : \text{0-1 変数},\ a_{ij} : \text{0-1 パラメータ} \tag{3.3.6}$$

$$\sum_{i\in I} a_{ij}x_i \le 1\ (j \in J), \quad x_i : \text{0-1 変数},\ a_{ij} : \text{0-1 パラメータ} \tag{3.3.7}$$

$$\sum_{i\in I} a_{ij}x_i \ge 1\ (j \in J), \quad x_i : \text{0-1 変数},\ a_{ij} : \text{0-1 パラメータ} \tag{3.3.8}$$

これらの制約は非常によく似ており，等号・不等号（の向き）のみが異なります．(3.3.6) は**集合分割制約**，(3.3.7) は**集合パッキング制約**，(3.3.8) は**集合被覆制約**とよばれる制約です [5]．

これらの制約を理解するために，例を挙げてみます．次のような 5 つの集合があるものとしましょう:

- 集合 $1 = \{1, 3\}$，集合 $2 = \{2, 3\}$，集合 $3 = \{1, 2\}$，集合 $4 = \{3\}$，集合 $5 = \{1\}$

上に挙げた制約により，これら 5 つの集合と，集合 $J = \{1, 2, 3\}$ との関係を表すことができます．集合 I を $I = \{1, 2, 3, 4, 5\}$ とし，集合 $i \in I$ が要素 $j \in J$ を含むとき $a_{ij} = 1$，そうでないとき $a_{ij} = 0$ とします．すると，a_{ij} $(i = 1, 2, 3, 4, 5;\ j = 1, 2, 3)$ は次のようになります．

$i \setminus j$	1	2	3
1	1	0	1
2	0	1	1
3	1	1	0
4	0	0	1
5	1	0	0

(3.3.6) は，集合 $i \in I$ により，集合 J を分割するための制約です．例えば $x_3 = x_4 = 1, x_1 = x_2 = x_5 = 0$ とすると，これは (3.3.6) を満たしています（下記表を参照）．

$i \setminus j$	1	2	3	x_i
1	1	0	1	0
2	0	1	1	0
3	1	1	0	1
4	0	0	1	1
5	1	0	0	0
$\sum_{i\in I} a_{ij}x_i$	1	1	1	−

これは，集合 $J = \{1, 2, 3\}$ が集合 $3 = \{1, 2\}$ と集合 $4 = \{3\}$ に **分割** されることを表しています．

(3.3.7) は，集合 $i \in I$ により集合 J を閉じ込める（= パックする）ための制約です．例えば

[5] MIPLIB のサイト [35] では，(3.3.6), (3.3.7), (3.3.8) は少し違う形で紹介されています．

$x_3 = 1, x_1 = x_2 = x_4 = x_5 = 0$ とすると，これは (3.3.7) を満たしています（下記表を参照）.

$i \setminus j$	1	2	3	x_i
1	1	0	1	0
2	0	1	1	0
3	1	1	0	1
4	0	0	1	0
5	1	0	0	0
$\sum_{i \in I} a_{ij} x_i$	1	1	0	−

これは，集合 $J = \{1, 2, 3\}$ が集合 $3 = \{1, 2\}$ を **パッキング** していることを表しています.

(3.3.8) は，集合 $i \in I$ により集合 J をカバー（被覆）するための制約です．例えば $x_1 = x_2 = 1, x_3 = x_4 = x_5 = 0$ とすると，これは (3.3.8) を満たしています（下記表を参照）.

$i \setminus j$	1	2	3	x_i
1	1	0	1	1
2	0	1	1	1
3	1	1	0	0
4	0	0	1	0
5	1	0	0	0
$\sum_{i \in I} a_{ij} x_i$	1	1	2	−

これは，集合 $1 = \{1, 3\}$，集合 $2 = \{2, 3\}$ が 集合 $J = \{1, 2, 3\}$ を **カバー（被覆）** していることを表しています.

(3.3.6), (3.3.7), (3.3.8) の 3 パターンの制約が現れる問題はそれぞれ 6.1, 6.2, 6.3 節で取り上げます.

3.3.3　ナップサック制約に関するパターン

ここではナップサック制約とその派生形を紹介します．まず，基本となるナップサック制約は次のようなものです.

$$\sum_{i \in I} a_i x_i \leq b, \quad x_i : \text{0-1 変数}, \ a_i, b \in \mathbb{N} \tag{3.3.9}$$

この制約は，次のような状況を表しています: 容量が b のナップサック（袋）に，大きさ a_i $(i \in I)$ の品物 i を詰め込みます．そのとき，ナップサックの中に品物 $i \in I$ を詰め混むときは $x_i = 1$，そうでないときは $x_i = 0$ とします．このとき，制約 (3.3.9) は詰め込む品物がナップサックからあふれないということを表しています．この制約は 6.4 節で説明する**ナップサック問題**とよばれる最適化問題に現れます.

ナップサック制約の派生形として，次のようなものが考えられます．

$$\sum_{i\in I} a_i x_i = b, \quad x_i : \text{0-1 変数}, \ a_i, b \in \mathbb{N} \tag{3.3.10}$$

$$\sum_{i\in I} x_i \le b, \quad x_i : \text{0-1 変数}, \ b \in \mathbb{N} \tag{3.3.11}$$

$$\sum_{i\in I} a_i x_i \le b, \ x_i \ge 0 \quad x_i : \text{整数変数}, \ a_i, b \in \mathbb{N} \tag{3.3.12}$$

$$\sum_{i\in I} a_i x_i \le a_k x_k, \quad x_i, x_k : \text{0-1 変数}, \ a_i, a_k \in \mathbb{N} \tag{3.3.13}$$

(3.3.10) は等式ナップサック制約 (Equality Knapsack) です．(3.3.9) が等式になった制約であり，ナップサックに余りがないように詰め込まなくてはならない，ということを表しています．

(3.3.11) は，不変ナップサック制約 (Invariant Knapsack) です．品物の大きさが不変（全て 1）である場合を表しています．

(3.3.12) は整数ナップサック制約 (Integer Knapsack) です．(3.3.9) は品物をナップサックに入れるか入れないかの選択しかできませんでしたが，この制約では同じ品物を複数個入れることが可能になります．

(3.3.13) は，ビンパッキング (Bin Packing) 制約と呼ばれます[6]．ここでは，ナップサックだった入れ物が “bin”=「容器」に代わります（「瓶」ではないので，あしからず）．ナップサック制約では，ナップサックは常にあるものとされていましたが，この制約ではビン k は使う場合と使わない場合に分けて考えます．すなわち，ビン k を利用する場合は 0-1 変数 x_k が 1 に，そうでないときは 0 になるものとします．するとビン k を使う場合は右辺が a_k となり，これを b と見なせば (3.3.9) と同じ形になります．一方ビン k を使わない場合は右辺が 0 となり，自動的に $x_i = 0$ $(i \in I)$（品物 i は詰め込まない）となります．

3.3.4 その他のパターン

ここでは，その他のパターンについて紹介します．

混合 0-1 制約 (Mixed Binary)

これは，0-1 変数と連続変数の線形和に関する以下のような制約です．

$$\sum_{i\in I} a_i x_i + \sum_{j\in J} p_j s_j \left\{ \begin{array}{c} \le \\ = \\ \ge \end{array} \right\} b, \quad x_i : \text{0-1 変数}, \ s_j : \text{連続変数}, \ a_i, p_j \in \mathbb{R} \tag{3.3.14}$$

その他の制約

MIPLIB では，上記以外の制約を “General: All other constraint types” として分類しています．上記のパターンは典型的な制約ではありますが，制約が複雑になると，上記のパターンに当てはまらないものも出てきます．

[6] MIPLIB のサイト [35] では，$\sum_{i\in I} a_i x_i + a_k x_k \le a_k$ という形で紹介されています．$a_k x_k$ を右辺に移項し，$(1 - x_k)$ を改めて x_k とすると (3.3.13) を得ます．

3.4　定式化・基礎 (3): 定式化の「デザインパターン」

「**デザインパターン**」という言葉を聞いたことがあるでしょうか？この言葉は建築分野で最初に用いられました [1]．これは，建築物に現れる典型的なパターンを整理し，新しい建築物を設計する際に生かそうとする試みでした．その後，デザインパターンという言葉はプログラミングの世界でも使われるようになります [13].

ここでは，最適化問題に現れる典型的な制約条件をパターンとしてまとめ，最適化問題を定式化する際の「デザインパターン」としたいと思います[7].

3.4.1　ネットワーク

デザインパターン「ネットワーク」は，グラフ上の流れ（これを**フロー**といいます）を考える場合に使うことのできるパターンです.

グラフ上のフローを考える場合，**フロー保存則**というルールを適用することがよくあります．これは，

> グラフ上のフローは各節点で保存される

という単純なものです．具体的に言うと，グラフが水道管網を表しており，枝の上を水が流れていく場合,「節点で水漏れしない」ということがフロー保存則に対応します．言い換えれば,「節点に流れ込む量と節点から流れ出る量が等しい」ということができます．直感的にもイメージしやすいルールだと思います．ただし注意すべきこととして，フローの始点となる節点（これを**ソース**といいます）と終点となる節点（**シンク**といいます）ではこのルールは適用されません.

グラフ $G = (V, E)$ 上でのフロー保存則を定式化すると，次のようになります.

$$\sum_{(i,v)\in E} x_{iv} = \sum_{(v,j)\in E} x_{vj}, \quad (v \in V,\ v \neq s, t) \tag{3.4.1}$$

$$\sum_{(s,j)\in E} x_{sj} = f \left(= \sum_{(j,t)\in E} x_{jt} \right) \tag{3.4.2}$$

$$x_{ij} \geq 0, \quad ((i,j) \in E) \tag{3.4.3}$$

ここで s はソース，t はシンクを表していて，x_{ij} は枝 $(i,j) \in E$ の上を流れるフローの量，f はフローの全量とします.

(3.4.1) は，ソースあるいはシンクでない節点 $i \in V$，すなわちにフローを中継する節点に適用される式です．(3.4.1) の左辺は節点 i から出て行く枝を流れるフローの和を，右辺は節点 i に入ってくる枝を流れるフローの和を表しています．これらが等しくなるということが，フロー保存則に他なりません．フローの量は，それが連続的に分割可能なもの（例えば水など）であれば連続変数で表現しますし，分割できないものであれば整数変数で表現することになります.

(3.4.2) の左側の等号は，ソース s から流れ出るフローの量が f であることを表しています．そして，フロー保存則の下ではフローがどこかに消える（「水漏れ」する）ようなことはないため，このフローは全てシンク t に到達することになります．それが (3.4.2) の右側の等号です.

[7]ここで挙げるデザインパターンは，私の研究室で指導した学生の卒業論文 [19] で作成されたものを一部改変したものです．[19] で作成されたデザインパターンには 3.4 節で扱うものの他に「選択」というパターンがありましたが，これは 3.3 節で出てきています（(3.3.2) に相当）ので，ここでは省略します.

最後に (3.4.3) は，フローの量が非負であることを表している制約です．もしこの制約がなければ，フローの量が負になってしまう可能性があります．

その他に，各枝にフローの容量に関する制約が付加される場合があります．例えば水道管の直径に応じて流す事が可能な容量を定めるような場合です．その場合は，枝 $(i,j) \in E$ の容量を c_{ij} (≥ 0) とすると，

$$x_{ij} \leq c_{ij} \tag{3.4.4}$$

とすることができます．

ここで紹介した制約を用いると，例えば 6.6 節で説明する**最大流問題**とよばれる最適化問題を定式化することができます．

3.4.2 二部グラフ

デザインパターン「二部グラフ」は，問題に二部グラフ的な構造が現れる場合に用いることができるパターンです．

グラフ $G = (V, E)$ が二部グラフであるものとし，節点集合 V は集合 I と J に分割できるものとします．すなわち $I \cup J = V, I \cap J = \emptyset$ であり，全ての枝 $e \in E$ の端点の片方は I に，もう片方は J に属するものとします．

このパターンでは，節点 $i \in I$ と節点 $j \in J$ の組 (i,j) に何らかの量を対応づける変数 x_{ij} を考えます．例えば次のようなものが考えられます．

設定 3.4.1 x_{ij} は節点 $i \in I$ から節点 $j \in J$ へのフローの量を表すものとします．このとき，(通常，) この変数には非負制約 $x_{ij} \geq 0$ が必要になります． □

設定 3.4.2 x_{ij} は節点 $i \in I$ と節点 $j \in J$ を何らかの意味で対応づける 0-1 変数であるものとします．すなわち $x_{ij} = 1$ のとき i と j が対応，$x_{ij} = 0$ のとき i と j は対応しないものとします． □

設定 3.4.1, 3.4.2 のいずれの場合であっても，このパターンで重要になるのは次の式です．

$$\sum_{j \in J} x_{ij} \quad (i \in I) \tag{3.4.5}$$

$$\sum_{i \in I} x_{ij} \quad (j \in J) \tag{3.4.6}$$

まず，設定 3.4.1 の下で (3.4.5), (3.4.6) がどのような意味を持つかについて説明します．このとき，(3.4.5) で定められる量は節点 i から流れ出るフローの量の合計を表しており，(3.4.6) で定められる量は節点 j に流れ込むフローの量の合計を表しています．これを用いると，6.8 節で示す **Hitchcock 型輸送問題**と呼ばれる，最も典型的な輸送問題を定式化することができます．

次に設定 3.4.2 の下での (3.4.5), (3.4.6) の意味について説明します．ここでは $|I| = |J|$，すなわち I に含まれる節点数と J に含まれる節点数が等しいものとします．このとき，

$$\sum_{j \in J} x_{ij} = 1 \quad (i \in I) \tag{3.4.7}$$

$$\sum_{i \in I} x_{ij} = 1 \quad (j \in J) \tag{3.4.8}$$

とすると，これは $i \in I$ がいずれかの $j \in J$ と対応するようになります．これを利用すると，6.9 に挙げる**割当問題**と呼ばれる問題を定式化することができます．

3.4.3　誤差

デザインパターン「誤差」は，主に**最小 2 乗問題**と呼ばれる問題に現れます．この問題は次のようなものです．

問題 3.4.1 時刻 t_i $(i \in I)$ における何らかの観測値 v_i $(i \in I)$ があるものとします．そして，観測値 v_i は本来はある関数 $f(t;\boldsymbol{a})$ に従う，すなわち $v_i = f(t;\boldsymbol{a})$ となることが知られているものとします．ここで $\boldsymbol{a}$ は関数 f のパラメータです．しかし，観測の際に誤差が生じることがあるため，必ずしも $v_i = f(t;\boldsymbol{a})$ とはなりません．このとき，関数と観測値の誤差の 2 乗和（2 乗したものの総和）が最小になるように関数 f のパラメータ $\boldsymbol{a}$ を決めて下さい．　□

問題をもう少し具体的に理解するために，図 3.5 に例を準備しました．図中の黒丸が観測時刻と

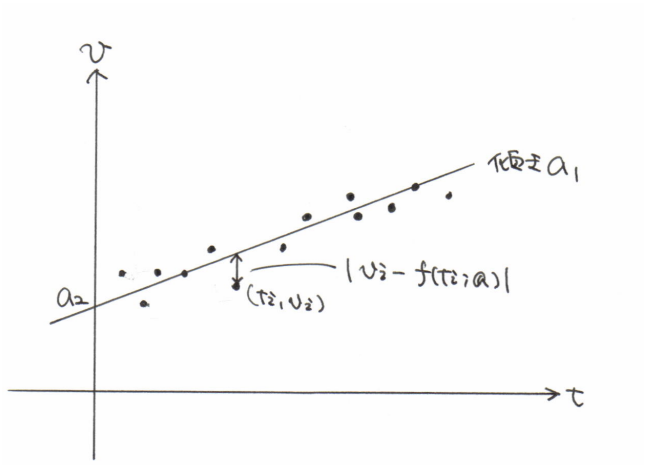

図 3.5: デザインパターン・誤差

観測値の組を表しているものとします．このとき，本来は観測値は $f(t;\boldsymbol{a}) = a_1 t + a_2$ という直線（$\boldsymbol{a} = (a_1, a_2)^\top$ とする．a_1 が直線の傾き，a_2 が縦軸の切片を表しているものとします）に従うはずなのですが，そうはなっていません．そこで，誤差の 2 乗和が最も小さくなるように $\boldsymbol{a} = (a_1, a_2)^\top$ を定める，という問題です．

問題 3.4.1 中の「誤差の 2 乗和」を $E(\boldsymbol{a})$ とすると，これは次のように書くことができます．

$$E(\boldsymbol{a}) = \sum_{i \in I} (v_i - f(t_i;\boldsymbol{a}))^2 \tag{3.4.9}$$

これを用いると，最小 2 乗問題は次のように書くことができます．

$$\left| \begin{array}{ll} \text{minimize} & E(\boldsymbol{a}) = \displaystyle\sum_{i \in I} (v_i - f(t_i;\boldsymbol{a}))^2 \end{array} \right. \tag{3.4.10}$$

問題 (3.4.10) で注意すべきことは，変数が t_i や v_i ではなく，関数のパラメータ $\boldsymbol{a}$ であることです．ここでは $\boldsymbol{a}$ には制約を設けていませんが，例えばパラメータが全て非負である必要があるならば $\boldsymbol{a} \geq \boldsymbol{0}$ のような制約を設けることもできます．

また，ここまでは「誤差の 2 乗和」を最小にすることを考えて来ましたが,「誤差の絶対値の和」を最小にする問題を考えることができます[8]．そのときは，(3.4.9) を次のように修正する必要があります．

$$E(\boldsymbol{a}) = \sum_{i \in I} |v_i - f(t_i; \boldsymbol{a})| \tag{3.4.11}$$

これを最小化すればよいのですが，(3.4.11) には絶対値記号 $|\cdot|$ が含まれているため，このままでは多くのソルバでは直接扱うことができません．しかし，少し表現を修正すると，扱うことができる問題となります．これについては 3.5.2 節で説明します．

3.5 定式化・応用 (1): 便利な表現

3.5.1 最大値の最小化（最小値の最大化）

ここでは次のような最適化問題を考えたいものとします[9]．

$$\left| \begin{array}{ll} \text{minimize} & \max\{x_1, x_2, \ldots, x_m\} \\ \text{subject to} & \ldots\text{(もしあれば他の制約)} \end{array} \right. \tag{3.5.1}$$

すなわち，変数 $x_1, x_2, \ldots, x_m$ のうち最大の値となるものを最小にしたい，という問題です．関数 $f(\boldsymbol{x}) = \max\{x_1, x_2\}$ のグラフを図 3.6 に示します．これをみると分かるように，この関数は微分不可能な点（$x_1 = x_2$ となるような点）を持つため，これを直接書き下すことができない場合があります[10]．

この問題 (3.5.1) は次の定式化と同値です．

$$\left| \begin{array}{ll} \text{minimize} & y \\ \text{subject to} & x_i \leq y \quad (i = 1, 2, \ldots, m) \\ & \ldots\text{(もしあれば他の制約)} \end{array} \right. \tag{3.5.2}$$

制約 $x_i \leq y$ $(i = 1, 2, \ldots, m)$ により，新たに導入した変数 y の値は x_i $(i = 1, 2, \ldots, m)$ の最大値以上になります．一方で，目的関数で y を最小化しようとするため，y の値は x_i $(i = 1, 2, \ldots, m)$ の最大値より大きな値になることはなく，最大値と等しくなることがわかります．(3.5.2) に現れる目的関数・制約条件はいずれも微分可能（かつ線形）であるため，ソルバで扱うことができる形になっています．

ここでは，単純に m 個の変数 $x_1, x_2, \ldots, x_m$ の最大値を最小にするという設定にしましたが，(3.5.1), (3.5.2) を見ればわかるように，これは特に変数である必然性はありません．$x_1, x_2, \ldots$ に相当する量が何らかの関数で決まっているような場合も同様の方法が適用可能です．

---

[8] 単に「誤差の和」とすることはまずありません．誤差にはプラスとマイナスがあることを忘れないで下さい！

[9] 目的関数に 'min-max' と並ぶので，これを「ミンマックス」型の目的関数ということがあります．

[10] モデリング言語・ソルバによっては，これを直接表現できる場合もあります．そのような場合は本節は読み飛ばして下さい．

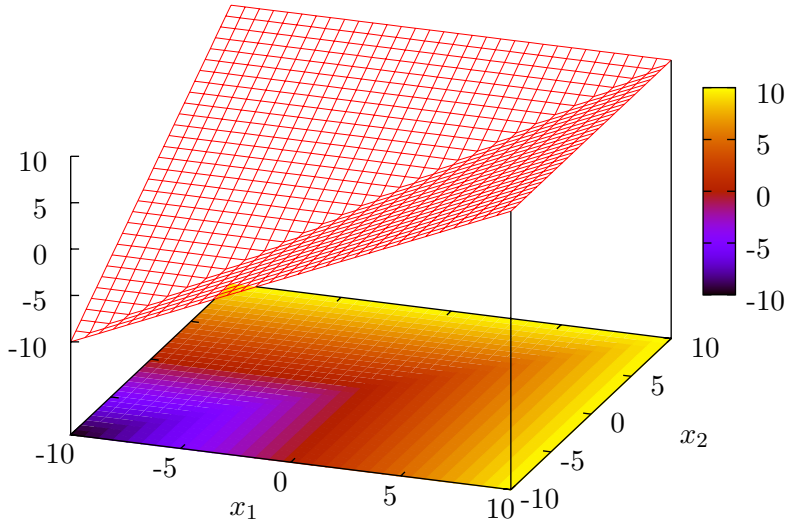

図 3.6: 関数 $f(\boldsymbol{x}) = \max\{x_1, x_2\}$ のグラフ

また，ここでは「最大値の最小化」で説明しましたが,「最小値の最大化」も同様の考え方で定式化することができます [11]．しかし,「最大値の最大化」「最小値の最小化」には同じ手法は適用できません．これには別の定式化を考える必要があります．これは 3.6.1 節で説明します．

3.5.2　絶対値の最小化

ここでは次のような最適化問題を考えるものとします．

$$\left|\begin{array}{ll} \text{minimize} & |x| \\ \text{subject to} & \ldots\text{(何らかの制約)} \end{array}\right. \tag{3.5.3}$$

3.5.1 節と同様，(3.5.3) を直接定式化することはできません [12]．関数 $|x|$ は $x = 0$ で折れ曲がっており，微分不可能なためです（例 A.2.1 でも取り上げました）．そこで，新たに変数 y を導入し，(3.5.3) と等価な問題として次の問題を考えます．

$$\left|\begin{array}{ll} \text{minimize} & y \\ \text{subject to} & -y \leq x \leq y \\ & \ldots \end{array}\right. \tag{3.5.4}$$

(3.5.4) の制約条件を見ると，$-y \leq y$ なる関係が成立しなくてはならないため，$y \geq 0$ となることがわかります．さらに目的関数の効果により y の値は小さくなろうとします．このとき，$x \geq 0$ であれば制約 $x \leq y$ が働き，結果として $x = y$ となります．あるいは $x < 0$ であれば $x = -y$ となるはず

[11]「最小値の最大化」の場合は「マックスミン（マキシミン）」型の目的関数，ということになります．
[12]モデリング言語・ソルバによっては可能な場合もあります．

です．よって，いずれの場合であっても y の値が x の絶対値と等しくなります．

ここで，3.4.3 節で取り上げた「誤差の絶対値の和」を最小化する問題について説明します．これは次のような問題です．

$$\left| \text{minimize} \quad E(\boldsymbol{a}) = \sum_i |v_i - f(t_i; \boldsymbol{a})| \right. \tag{3.5.5}$$

問題 (3.5.5) は，上で述べた工夫を応用することで，次のように書くことができます．

$$\left| \begin{array}{l} \text{minimize} \quad E(\boldsymbol{a}) = \displaystyle\sum_i y_i \\ \\ \text{subject to} \quad -y_i \le v_i - f(t_i; \boldsymbol{a}) \le y_i \quad (i \in I) \end{array} \right. \tag{3.5.6}$$

問題 (3.5.6) は絶対値記号 $|\cdot|$ が含まれていないため，通常のソルバで取り扱うことができるようになります（ただし，ソルバによっては関数 f の線形性が要求される場合があることに注意してください）．

3.5.3 2 通りの絶対値制約

絶対値が現れる制約条件には，次の 2 通りがあります．

$$|x| \le a \tag{3.5.7}$$

$$|x| \ge b \tag{3.5.8}$$

ここで $a \ge 0, b \ge 0$ とします（そうでない場合，(3.5.7), (3.5.8) を満たすような x は存在しません）．前項でも述べたように，(3.5.7) は

$$|x| \le a \quad \Leftrightarrow \quad -a \le x \ \underline{\text{かつ}}\ x \le a$$

とすることで，絶対値を使わない形で表現することができます．一方，(3.5.8) は

$$|x| \ge b \quad \Leftrightarrow \quad -b \ge x \ \underline{\text{または}}\ x \ge b$$

となります．2.2.3 節で述べたように，最適化問題で制約条件を複数設けることは「または」ではなく「かつ」の意味ですから，いくつかの制約を単純に並べるだけでは (3.5.8) を表現することはできません．そこで，(3.5.8) を定式化するためには，0-1 変数 z を用いて，次のように書きます．

$$zb - (1-z)M \le x \ \underline{\text{かつ}}\ x \le -(1-z)b + zM \tag{3.5.9}$$

ここで M は十分大きな正の定数を表しているものとします．最適化問題の定式化では，このように十分大きな定数を導入することで定式化がうまくいく場面が多くあります．この分野では，そのような定数を表現する際に慣用的に M を用い，これを俗に **big-M** と言います．本書もそれを踏襲することにします．

さて，$z = 0$ と $z = 1$ の場合にわけて (3.5.9) を見てみると，次のようになります．

$z = 0$ のとき: $-M \le x$ かつ $x \le -b \quad \Leftrightarrow \quad -M \le x \le -b$
$z = 1$ のとき: $b \le x$ かつ $x \le M \quad \Leftrightarrow \quad b \le x \le M$

$z = 0$ は x の値が負の場合に対応しています．$-M$ は十分小さな値になるため，実質的には有効な制約とはなりません（逆に言えば，有効な制約とならない程度に M の値を大きくしておく必要があります）．するとこの制約は $x \leq -b$ となり，$|x| \geq b$ の片側（x の値が負の場合）に対応します．$z = 1$ の場合も同様で，これは x の値が正の場合に対応します．

3.5.4　指標関数

ここでは，変数 x の値の正負に応じて 0-1 変数 y の値を切り替える方法について考えます．例えば次のような状況です．

設定 3.5.1 $x \geq 0$ ならば $y = 1$，$x < 0$ ならば $y = 0$. □

これを実現するためには，big-M を用いることで次のように書くことができます．

$$-(1-y)M \leq x \leq yM \tag{3.5.10}$$

$y = 1$ の場合，(3.5.10) は $0 \leq x \leq M$ となります．M が（x の取りうる値よりも）十分大きければ，これは実質的に $0 \leq x$ となり，$x \geq 0$ と $y = 1$ が対応していることがわかります．一方 $y = 0$ の場合，(3.5.10) は $-M \leq x \leq 0$ となり，これは実質的に $x \leq 0$ となりますから，$x \leq 0$ と $y = 0$ が対応することがわかります．

ただし，この方法では $x = 0$ の場合に y が 0 にも 1 にもなることができます．これが問題になる場合は，例えば次のような定式化が考えられます．

$$-(1-y)M + y\epsilon \leq x \leq yM \tag{3.5.11}$$

ここで ϵ は非常に小さな正の定数とします[13]．$y = 1$ の場合 (3.5.11) は $\epsilon \leq x \leq M$ となり，$y = 0$ の場合は $-M \leq x \leq 0$ となりますから，$x = 0$ の場合は必ず $y = 0$ となります（同様に，(3.5.11) の最右辺から $(1-y)\epsilon$ を引けば，$x = 0$ の場合は必ず $y = 1$ となります）．しかし，x の値が $0 < x < \epsilon$ となる場合はどちらにも含まれないため，エラーとなってしまうため，ϵ の設定に注意する必要があります．

3.5.5　Special Ordered Set

最適化問題でよく使われる制約として，順番が与えられた変数の組に対する “Special Ordered Set” (SOS) という制約が知られています [5]．

順番を持つ（順番に意味がある）m 個の変数 $z_1, z_2, \ldots, z_m$ に対し，k 個の連続した変数のみが非零（通常は正の値），残りの変数が 0 であるとき，これを SOS-k と呼ぶことにします．

この制約は広く知られており，モデリング言語がこの制約固有の表現を備えている場合もあります．ここではその表現に頼らず，自ら SOS-k に相当する表現を定式化する手法を紹介します．

よく用いられるのは SOS-1, SOS-2 です．以下ではこれらの制約の典型的な利用例を紹介します．

[13] 余談: “big-M” とは言いますが，“small-ϵ” とは言いません．一般的に数学では ϵ は「微小な正の値」を表現するという（暗黙の）約束がある（例: いわゆる「ϵ-δ 論法」での ϵ など）ためだと思われます．また，このような ϵ に相当する量を，big-M を用いて $1/M$ と表現する場合もあります．

SOS-1

SOS-1 は「変数 1 つだけが非零の値を取り，残りの変数は 0」という状況を意味します．これを表現するために，m 個の 0-1 変数 $z_1, z_2, \ldots, z_m$ を導入します．そして，次の制約を設けることにします．

$$\sum_{i=1}^{m} z_i = 1 \tag{3.5.12}$$

この制約を利用すれば，例えば次のような設定を表現することができます．

設定 3.5.2 m 個の変数 $y_1, y_2, \ldots, y_m$ のうち，1 個だけが正の値をとることができ，それ以外の変数は 0 でなくてはならない． □

これは，まさに SOS-1 に含まれる状況です．このような状況は，制約 (3.5.12) と

$$0 \le y_i \le M z_i \quad (i \in \{1, 2, \ldots, m\}) \tag{3.5.13}$$

という制約を同時に設けることにより表現することができます．ここで M はいわゆる big-M（十分大きな正の定数）です．$z_i = 0$ であれば (3.5.13) は $0 \le y_i \le 0 \Rightarrow y_i = 0$ となります．一方，$z_i = 1$ であれば $0 \le y_i \le M$ となりますが，M が十分大きければ（y_i が取りうる値よりも大きく設定されていれば），これは事実上 $0 \le y_i$ という制約となります．そして SOS-1 制約により $z_i = 1$ となるような i が 1 個に制限されるため，上記の設定の内容が実現されます．

SOS-2

SOS-2 は,「連続する 2 つの変数が非零であり，残りの変数は 0」という状況になります．ここでも，SOS-1 と同様，$z_1, z_2, \ldots, z_m$ が 0-1 変数であるものとします．ここでは，連続した 2 個の変数が非零であることが許されるので，

$$\sum_{i=1}^{m} z_i = 2 \tag{3.5.14}$$

という制約が必要になります．しかし (3.5.14) だけでは「連続した」という条件を満たさない解が選ばれる可能性がありますので，これを排除する制約が必要になります．ここでは，

連続した 2 個の変数が 1 $\quad\Leftrightarrow\quad$ 2 個以上離れた変数が同時に 1 になることはない

と考え，次のような制約を課します．

$$z_i + z_j \le 1 \quad (i = 1, \ldots, m-2;\ j = i+2, \ldots, m) \tag{3.5.15}$$

例えば $m = 5$ の場合，(3.5.15) は

$$z_1 + z_3 \le 1,\ z_1 + z_4 \le 1,\ z_1 + z_5 \le 1,\ z_2 + z_4 \le 1,\ z_2 + z_5 \le 1,\ z_3 + z_5 \le 1$$

となります．

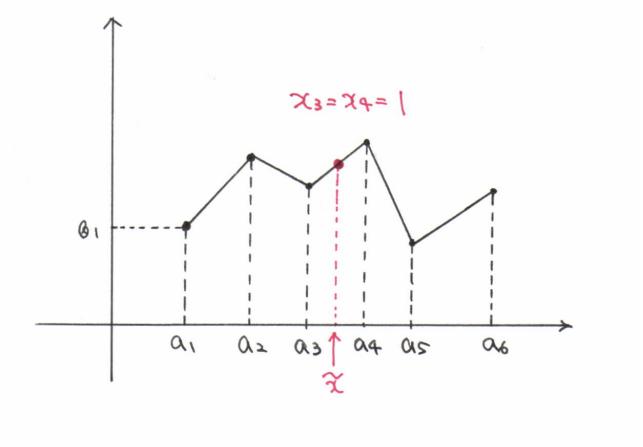

図 3.7: 区分線形関数

SOS-2 を利用する典型的な例として，**区分線形関数**の表現が挙げられます．区分線形関数とは，(a_i, b_i) $(i = 1, 2, \ldots, m)$ を折れ線状に結んで得られる関数です（図 3.7）（ただし $a_1 < a_2 < \cdots < a_m$ とします）．このとき，制約 (3.5.14), (3.5.15) の他に次の制約を準備します．

$$0 \leq \lambda_i \leq z_i \quad (i = 1, 2, \ldots, m) \tag{3.5.16}$$

$$\sum_{i=1}^{m} \lambda_i = 1 \tag{3.5.17}$$

(3.5.16), (3.5.17) に現れる変数 λ_i $(i = 1, 2, \ldots, m)$ は，点 (a_i, b_i) の重みを表す変数です．区分線形関数上の点は，隣り合う 2 個の点の凸結合で表現することができます．SOS-2 を用いることによりこの「隣り合う 2 個」という概念を表現します．このとき，$z_i = 0$ となるような $i \in \{1, 2, \ldots, m\}$ については (3.5.16) より重み λ_i が 0 となります．一方，$z_i = z_{i+1} = 1$ となるような $i \in \{1, 2, \ldots, m-1\}$ が 1 個だけ存在し，対応する重み λ_i, λ_{i+1} が非零の値を持つことができます．そして (3.5.17) より，重み $\lambda_i, \lambda_i + 1$ は総和が 1 になるように分割されます [14]．この重みを用いることにより，点 $(a_i, b_i), (a_{i+1}, b_{i+1})$ の凸結合，すなわち区分線形関数上の点を表すことができます．具体的には，

$$x = \sum_{i=1}^{m} \lambda_i a_i \tag{3.5.18}$$

$$f = \sum_{i=1}^{m} \lambda_i b_i \tag{3.5.19}$$

とすることで，考えている点 x での区分線形関数の値 f を得ることができます．

3.5.6　特殊な状況における曲線の折れ線近似

ここでは，ある特殊な状況において，曲線を折れ線近似する方法を説明します．

[14] $\lambda_i = 1, \lambda_{i+1} = 0$ または $\lambda_i = 0, \lambda_{i+1} = 1$ のように片方だけが重みを持つ可能性はあります．

まず次の問題を考えてみましょう.

$$\left|\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & \text{（変数 } x \text{ や他の変数を用いた制約）} \end{array}\right. \tag{3.5.20}$$

ここで，目的関数 $f(x)$ は図 3.8 のような非線形関数であるものとします．この問題は非線形関数

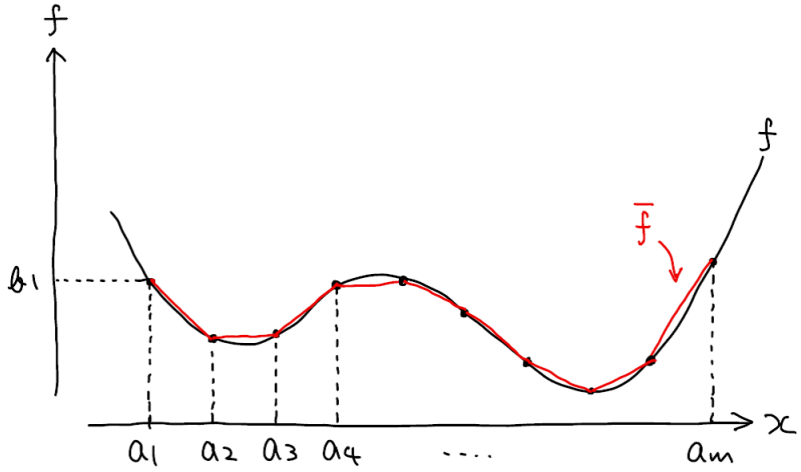

図 3.8: 非線形関数の区分線形関数による近似

f が含まれているため NLP になります．さらに，整数変数が存在した場合は MINLP になります．NLP/MINLP を直接扱えるソルバは現在はまだ多くありません．そこで，この問題を普通のソルバでも扱えるような問題クラスに変換するため，目的関数上に（実行可能領域上で）いくつか点を取り，それらを結ぶことで $f(x)$ を折れ線で近似することにします．

これは，3.5.5 節で扱った SOS-2 を用いれば表現可能になることはすぐにわかると思います．具体的には，目的関数上に点 (a_i, b_i) $(i = 1, 2, \ldots, m)$ を設定したとき，関数 f を折れ線で近似した $\bar{f}$ を目的関数とする問題は次のように書くことができます．

$$\left|\begin{array}{ll} \text{minimize} & \bar{f} = \displaystyle\sum_{i=1}^{m} \lambda_i b_i \\ \text{subject to} & x = \displaystyle\sum_{i=1}^{m} \lambda_i a_i \\ & \displaystyle\sum_{i=1}^{m} z_i = 2 \\ & z_i + z_j \le 1 \quad (i = 1, \ldots, m-2;\ j = i+2, \ldots, m) \\ & 0 \le \lambda_i \le z_i \quad (i = 1, 2, \ldots, m) \\ & \displaystyle\sum_{i=1}^{m} \lambda_i = 1 \\ & z_i \in \{0, 1\} \quad (i = 1, 2, \ldots, m) \\ & \text{（変数 } x \text{ や他の変数を用いた制約）} \end{array}\right. \tag{3.5.21}$$

ここで，(3.5.21) の目的関数が (3.5.19) と対応し，(3.5.21) の $1,2,3,4,5$ 番目の制約がそれぞれ (3.5.18), (3.5.14), (3.5.15), (3.5.16), (3.5.17) に対応していることに注意してください．このとき，この表現のために必要になる制約は全て線形なので，問題は MILP となります．これであれば，多くのソルバで扱うことができます．

それでは，最小化する目的関数が図 3.9 の $\tilde{f}$ のような場合はどうでしょうか？見ての通り，$\tilde{f}$ は凸

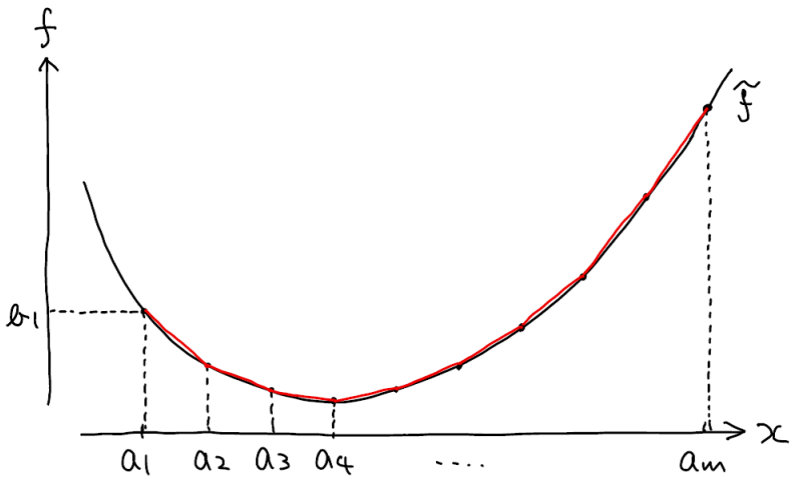

図 3.9: 凸な非線形関数の区分線形関数による近似

関数です．ここまでこの本を読んで来られた読者の中には,「凸関数の最小化」と聞くと，なんだか問題の性質がよさそうだ，ということに気付いて頂ける方もおられるでしょう．

SOS-2 を用いた折れ線近似の本質は,「連続した 2 個の変数 z_i, z_{i+1} が非零になる」ということです．そのために，(3.5.15) に対応する制約が必要になります．ではこの制約がないとどうなるでしょうか？この場合，連続しない 2 個の変数が非零になる可能性があります（ただし，(3.5.14) という制約があるので，3 個以上の変数が非零になることはありません）．

連続しない 2 個の変数が非零になる場合を考えると，例えば図 3.10 のようになります．ここでは (a_1, b_1) と (a_4, b_4) の重み λ_1, λ_4 がそれぞれ $1/2$ である場合を図示しました．このとき，(3.5.18), (3.5.19) に相当する点 $P = (x, f)$ は関数 $\tilde{f}$ を近似した折れ線上にはないことがわかります．

しかし，このようなことが実際に生じるでしょうか？ここで注意したいのは，我々が考えているのは関数 $\tilde{f}$ を近似した折れ線で表されるグラフの「最小化」であるということです．図 3.10 からわかるように，点 P は折れ線よりも上方に現れますから，最適化計算の結果，このような点 P が最適解となることはあり得ません．つまり，最小化の効果により,「連続した 2 個の変数が非零になる」ことを表した制約 (3.5.14), (3.5.15) がなくとも，最適化の結果得られる点は自然と関数 $\tilde{f}$ を近似した折れ線の上に現れることになります．

これをまとめると，最小化する目的関数を図 3.10 の $\bar{\bar{f}}$ のような折れ線関数で近似するときに解く

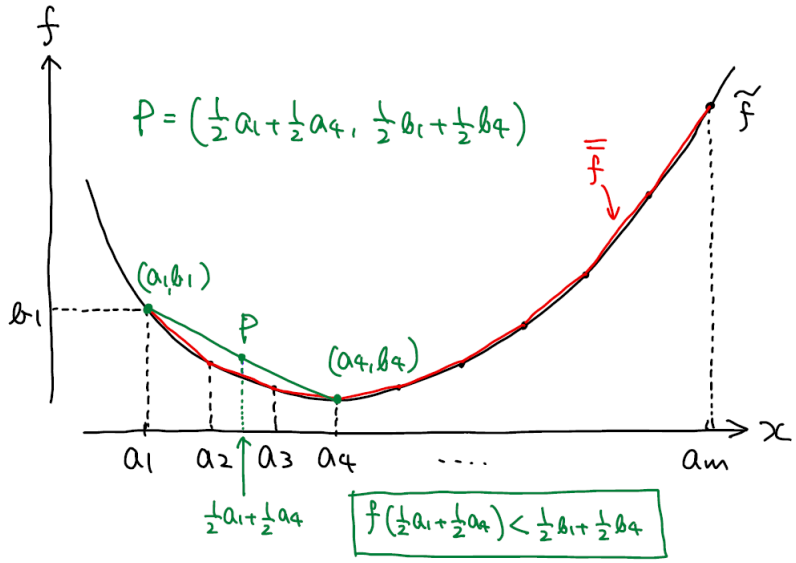

図 3.10: 連続しない 2 個の変数が非零になる場合

べき最適化問題は次のようになります.

$$
\left|\begin{array}{ll}
\text{minimize} & \bar{f} = \displaystyle\sum_{i=1}^{m} \lambda_i b_i \\
\text{subject to} & x = \displaystyle\sum_{i=1}^{m} \lambda_i a_i \\
& 0 \le \lambda_i \quad (i = 1, 2, \ldots, m) \\
& \displaystyle\sum_{i=1}^{m} \lambda_i = 1 \\
& \text{(変数 } x \text{ や他の変数を用いた制約)}
\end{array}\right. \tag{3.5.22}
$$

問題 (3.5.22) からは整数変数 z_i が消えることになります. もし他の変数が整数変数でなければ, 問題 (3.5.22) は LP になることに注意してください.

3.6 定式化・応用 (2): 「かゆいところに手が届く」表現

3.6.1 最大値の最大化 (最小値の最小化)

3.5.1 節では「最大値の最小化 (最小値の最大化)」の定式化について説明しましたが, ここでは「最大値の最大化 (最小値の最小化)」の定式化について説明します.

ここで考える問題は次のようなものです.

$$
\left|\begin{array}{ll}
\text{maximize} & \max\{x_1, x_2, \ldots, x_m\} \\
\text{subject to} & \ldots \text{(もしあれば他の制約)}
\end{array}\right. \tag{3.6.1}
$$

3.5.1 節でも説明したように，この目的関数は微分不可能な点を持つため，ソルバで直接扱えない場合があります．

ここでは，(3.6.1) を次のように書き直すことにします．

$$\left|\begin{array}{ll} \text{maximize} & y \\ \text{subject to} & x_i + M(1-\delta_i) \geq y \quad (i = 1, 2, \ldots, m) \\ & \displaystyle\sum_{i=1}^{m} \delta_i = 1 \\ & \delta_i \in \{0, 1\} \quad (i = 1, 2, \ldots, m) \\ & \ldots (\text{もしあれば他の制約}) \end{array}\right. \tag{3.6.2}$$

ここで，M はいわゆる big-M，すなわち十分大きな正の定数です（具体的には，x_i $(i = 1, 2, \ldots, m)$ が取りうる最大値に設定すれば十分です）．

(3.6.2) をみるとわかるように，ここでは 0-1 変数 δ_i $(i = 1, 2, \ldots, m)$ を導入しています．2 番目の制約式より，δ_i $(i = 1, 2, \ldots, m)$ はちょうど 1 個だけが 1 になり，残りが 0 になるということがわかります．このとき，1 番目の制約式は，

- $\delta_i = 0$ のとき: $x_i + M \geq y$
- $\delta_i = 1$ のとき: $x_i \geq y$

となります．$\delta_i = 0$ のときは，big-M の効果により，この制約式が事実上意味のないものになっていることがわかります．一方 $\delta_i = 1$ のときは y が x_i を上回らないような制約になっています．そして，この制約に現れる変数 y は目的関数で最大化する対象になっています．すると，y は $\delta_i = 1$ となった i に対して $x_i = y$ となることになります．

問題はどの $i \in \{1, 2, \ldots, m\}$ が選ばれるかということですが，目的関数を最大にするわけですから，x_i が最大になるような i が選ばれることになります．

（3.5.1 節と同様,）ここでは問題を単純化するために変数 $x_1, x_2, \ldots, x_m$ の中で最大のものを最大化するという設定にしましたが，$x_1, x_2, \ldots$ に相当する量が何らかの関数で決まっているような場合も同様の方法が適用可能です．また,「最小値の最小化」も同様の考え方で実現可能です．

3.6.2　差の絶対値

ここでは，差の絶対値を表現する方法を学びます．

設定 3.6.1 変数 x, y に対し，変数 z を $z = |x - y|$ となるようにしたい． □

このことを表現するために，まず 0-1 変数 w を導入して次の制約を設けます．

$$-M(1-w) \leq x - y \leq Mw \tag{3.6.3}$$

ここで M は 3.5.3 節で説明した big-M です．(3.6.3) により，$x - y \geq 0$ のとき $w = 1$，$x - y \leq 0$ のとき $w = 0$ となります（$x - y = 0$ のときは $w = 1$ でも $w = 0$ でもどちらでも構いません）．なお，これは (3.5.10) と同じ形であることに注意してください．

次に以下の 2 つの制約を設けます.

$$-M(1-w)+(x-y)\le z\le M(1-w)+(x-y) \tag{3.6.4}$$
$$-Mw+(y-x)\le z\le Mw+(y-x) \tag{3.6.5}$$

ここで $w=1$ の場合，すなわち $x-y\ge 0$ の場合を考えると，(3.6.4), (3.6.5) は

$$x-y\le z\le x-y \tag{3.6.6}$$
$$-M+(y-x)\le z\le M+(y-x) \tag{3.6.7}$$

となります．M が十分大きければ (3.6.7) は実質的に意味の無い制約になります．一方，(3.6.6) から，$z=x-y$ となり，(3.6.3) と整合性が取れます.

$w=0$ の場合も同様に考えると，(3.6.4), (3.6.5) は

$$-M+(x-y)\le z\le M+(x-y) \tag{3.6.8}$$
$$y-x\le z\le y-x \tag{3.6.9}$$

となり，(3.6.8) は実質的に意味がなくなり，(3.6.9) より $z=y-x$ となるので，やはり (3.6.3) と整合性が取れます.

3.6.3 複数の 0-1 変数の連動

ここでは複数の 0-1 変数を連動させる制約を考えてみることにします．例えば，0-1 変数 x,y,z が表 3.3 のような関係にあるものとします．これに対して，(3.6.10) のような制約を考えてみます.

表 3.3: 0-1 変数 x,y,z の関係例 (1)

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

表 3.4: (3.6.10) における x,y,z の関係

x	y	z
0	0	0
0	1	0
1	0	0
1	1	0 or 1

$$\frac{x+y}{2}\ge z \tag{3.6.10}$$

(3.6.10) において，x,y の値の組合せに対して z が取りうる値をまとめたものが表 3.4 です．本当であれば $x=y=1$ のときに $z=1$ に固定したいのですが，$z=0$ となる場合も考えられます．そこで，次の制約を考えます.

$$2x+2y\le z+3 \tag{3.6.11}$$

上と同様に，(3.6.11) において，x,y の値の組合せに対して z が取りうる値をまとめると表 3.5 となります．表 3.4 と表 3.5 は矛盾しておらず，しかもこれらの両方を満たすものを考えるとこれは表

表 3.5: (3.6.11) における x, y, z の関係

x	y	z
0	0	0 or 1
0	1	0 or 1
1	0	0 or 1
1	1	1

表 3.6: 0-1 変数 x, y, z の関係例 (2)

x	y	z
0	0	1
0	1	0 or 1
1	0	0
1	1	1

3.3 になります．すなわち，(3.6.10) と (3.6.11) を制約とすると，表 3.3 の関係を実現することができます．

しかし，このような制約を思いつくのは簡単ではありません．また，同一の変数の関係を表現する制約は一意ではありません（実際，表 3.3 の関係を表現する異なる制約を以下で示します）．そこで，表 3.3 のような関係を機械的に表現する方法を紹介します．

図 3.11 のように，一辺の長さが 1 の立方体を xyz- 空間に配置し，表 3.3 で実行可能な点を調べます．そして，実行可能な点の凸包を考え，その「表面」を構成する平面の式を求めます．

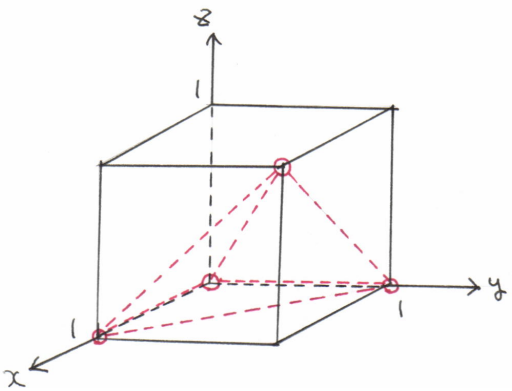

図 3.11: 立方体上の実行可能な点

ここでは $\{(0,0,0),(0,1,0),(1,0,0),(1,1,1)\}$ の凸包を考えます．この凸包の表面を構成する平面の式は

$$z = 0,\ x - z = 0,\ y - z = 0,\ x + y - z = 1$$

です．そして，これらの式の $=$ を適切に $\leq$ または $\geq$ に代えることにより，凸包全体を含む不等式に変更することができます．ここでは

$$z \geq 0,\ x - z \geq 0,\ y - z \geq 0,\ x + y - z \leq 1$$

とすると，それぞれの不等式が凸包を含むようになります．そして，これらの不等式が求める制約式に他なりません．ただし $z \geq 0$ は自明な式（z は値として 0 または 1 しか取らない）ですから，除いて構いません．よって求める制約は

$$x - z \geq 0,\ y - z \geq 0,\ x + y - z \leq 1 \tag{3.6.12}$$

となります.

ここでは表 3.3 のように x, y の値が決まれば z の値が一意に決まる場合を考えましたが，例えば表 3.6 のように，x, y の値が決まっても z の値が一意に定まらない場合も扱うことができます．表 3.6 に対する制約を求める場合は，$\{(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,1,1)\}$ の凸包の表面を構成する方程式を求めます．これは

$$x=0,\ y=1,\ z=1,\ x+y+z=1,\ x-y+z=1,\ x+y-z=1$$

になります．そして，これらの方程式を凸包を含むように不等式に変更すると

$$x\geq 0,\ y\leq 1,\ z\leq 1,\ x+y+z\geq 1,\ x-y+z\leq 1,\ x+y-z\leq 1$$

となります．そして自明な不等式を除くと，表 3.6 に対する制約は

$$x+y+z\geq 1,\ x-y+z\leq 1,\ x+y-z\leq 1$$

となります.

3.6.4 「または」の表し方

2.2.3 節でも説明したように，複数の制約条件を設けることは，それらの制約条件は「かつ (AND)」の関係，すなわちそれらを全て満たさなくてはならない，ということになります．しかし，問題によっては複数の制約が「または (OR)」の関係にある場合，すなわち複数の制約のうち少なくとも 1 個が満たされれば良い，という状況もあるでしょう.

実は，ここまでにも「または」の表現が出てきていました．3.5.3 節では，絶対値制約 $|x|\geq b$ が「$-b\geq x$ または $x\geq b$」であることを利用し，絶対値の現れない形に書き換える，ということを行いました．そして，そのときにポイントになったのはいわゆる big-M でした.

big-M は，0-1 変数と組み合わせることで，制約を「意味のある場合」と「実質的な意味のない場合」に切り替えるときに使うことができます．以下，簡単な例で big-M の役割を（再度）確認してみます.

設定 3.6.2 制約 $g_1(\boldsymbol{x})\leq 0$ と制約 $g_2(\boldsymbol{x})\leq 0$ があり，このとき少なくとも一方が満たされていればよいものとします．これはどのように定式化すればいでしょうか？ □

これを定式化するためには，次のようにすればよいでしょう.

$$\left|\begin{array}{l} g_1(\boldsymbol{x})\leq (1-\delta_1)M \\ g_2(\boldsymbol{x})\leq (1-\delta_2)M \\ \delta_1+\delta_2\geq 1 \\ \delta_i\in\{0,1\}\quad (i=1,2) \end{array}\right. \tag{3.6.13}$$

(3.6.13) において，δ_i $(i=1,2)$ は値が 1 のときには制約 $g_i(\boldsymbol{x})\leq 0$ となる，すなわち制約が元の形と同じものになります．一方，値が 0 の場合は制約は $g_i(\boldsymbol{x})\leq M$ となるため，M の値が十分大きければ（g_1, g_2 の取りうる値よりも大きく設定されていれば）実質的な意味を持たない制約になりま

す．そして，$\delta_1 + \delta_2 \geq 1$ という制約により，δ_1, δ_2 の少なくとも一方が 1 になる，すなわち制約が元の形になることから,「または (OR)」の関係が表現できていることがわかります．

初学者が勘違いしがち（あるいは思い込みがち）なのは,「変数の値が特定の条件を満たしたら，ある制約を取り除く（追加する）ようにできる」ということです．確かに，通常のプログラムにおいてはいわゆる `if` 文を用いることによりそのような記述をすることがあるでしょう．しかしソルバを用いた最適化ではそのようなことはできません（これについては 4.1.3 節でも触れます）．そのため，上記のような工夫が必要になってきます．

(3.6.13) では 2 つの不等式制約が「または」の関係にある場合を定式化しましたが，少し考えてみると 3 つ以上の不等式制約の場合でも同様の表現が可能であることがわかります．また，等式制約 $h_1(\boldsymbol{x}) = 0, h_2(\boldsymbol{x}) = 0$ が「または」の関係にある場合は，

$$\left|\begin{array}{l} -(1-\delta_1)M \leq h_1(\boldsymbol{x}) \leq (1-\delta_1)M \\ -(1-\delta_2)M \leq h_2(\boldsymbol{x}) \leq (1-\delta_2)M \\ \delta_1 + \delta_2 \geq 1 \\ \delta_i \in \{0,1\} \quad (i=1,2) \end{array}\right. \tag{3.6.14}$$

とすればよいでしょう（理由を考えてみて下さい）．さらに，不等式制約と等式制約を混在させて，

$$\left|\begin{array}{l} g_1(\boldsymbol{x}) \leq (1-\delta_1)M \\ g_2(\boldsymbol{x}) \leq (1-\delta_2)M \\ -(1-\bar{\delta}_1)M \leq h_1(\boldsymbol{x}) \leq (1-\bar{\delta}_1)M \\ -(1-\bar{\delta}_2)M \leq h_2(\boldsymbol{x}) \leq (1-\bar{\delta}_2)M \\ \delta_1 + \delta_2 + \bar{\delta_1} + \bar{\delta_2} \geq 1 \\ \delta_i \in \{0,1\} \quad (i=1,2), \quad \bar{\delta}_i \in \{0,1\} \quad (i=1,2) \end{array}\right. \tag{3.6.15}$$

とすることも可能です．

さらに，これを工夫すると，排他的論理和 (XOR) も実現可能です．不等式制約 $g_1(\boldsymbol{x}) \leq 0, g_2(\boldsymbol{x} \leq 0)$ が排他的論理和の関係にある，すなわちどちらか片方のみが満たされるようにするためには，

$$\left|\begin{array}{l} g_1(\boldsymbol{x}) \leq (1-\delta_1)M \\ g_2(\boldsymbol{x}) \leq (1-\delta_2)M \\ \delta_1 + \delta_2 = 1 \\ \delta_i \in \{0,1\} \quad (i=1,2) \end{array}\right. \tag{3.6.16}$$

とし，δ_1 と δ_2 の和がちょうど 1 になるようにすればよいでしょう．

ここで見たように,「または (OR)」（や排他的論理和 (XOR)）の関係は，big-M を用いることにより実現できることがわかりました．ただ，注意しておくべきこととして，次の事が挙げられます．

- big-M を多用しない
- big-M の値を大きくしすぎない

ソルバから見ると，big-M は求解の上で嬉しくない性質を持っています．ですので,「使わなくて良い場面では使わない」「使う場合もいたずらに値を大きくしない」ということを心がけて下さい．

第4章　モデリング言語による最適化問題の記述

本章で学ぶ内容

- モデリング言語による最適化問題の記述（モデルファイルの作成）
- よく知られた最適化問題のモデル記述の例

この章では，定式化した最適化問題をモデリング言語で記述する方法について学びます．

4.1　モデリング言語 AMPL の基礎

4.1.1　モデリング言語と最適化ソルバの関係

3 章で学んだ知識により，考えている問題を最適化問題として定式化できたものとします．次にやるべきことは，定式化した問題をソルバに解かせることです．そのためには，ソルバが理解できる形で問題を記述する必要があります．初期の最適化ソルバでは，ある特定のフォーマットに従って問題のパラメータ等を記述したファイルを準備し，ソルバに読み込ませることで，ソルバが問題を認識していました．これは機械的な表現であったため，計算機に認識させるのは容易であった反面，ユーザがこのファイルを直接作成するには大きな困難が伴っていました．しかし近年では，モデルを比較的自由に記述できるモデリング言語が登場し，ユーザが問題を定義するのが以前と比べて格段に楽になりました．

4.1.2　モデリング言語 AMPL の概要

モデリング言語は特定のソルバ固有のものもありますし，複数のソルバで認識できるものもあります．ここでは AMPL [10] [1] というモデリング言語の基礎を学びます．その理由は以下の通りです．

- AMPL で記述された最適化問題を解くことができるソルバが多数存在する
- AMPL を元にして作成されたモデリング言語が存在する
- AMPL を元にして作成されたモデリング言語でなくとも，モデル記述の考え方は AMPL と共通する部分が多い
- AMPL で記述された最適化問題を他の形式に変換できるフリーのソフトウェアが存在する [2]

[1] 参考文献に挙げたサイトから [10] の PDF ファイルがダウンロード可能です．
[2] 最適化ソルバ GLPK [27] の一つの機能として変換機能が実装されています．

4.1.3　モデルファイルとデータファイル

AMPL（や多くのモデリング言語）では，最適化問題を定義するために「モデルファイル」と「データファイル」の 2 種類のファイルを用います．これらは次のように使い分けます．

- モデルファイル: 最適化問題の構造を定義する
- データファイル: モデルファイルで定義された構造に現れる集合やパラメータの具体的な値を定義する

例えば次の最適化問題を AMPL で記述することにします．

$$\left|\begin{array}{ll} \text{minimize} & -x_1 - x_2 \\ \text{subject to} & 3x_1 + 2x_2 \leq 12 \\ & x_1 + 2x_2 \leq 8 \\ & x_1 \geq 0,\ x_2 \geq 0 \end{array}\right. \tag{4.1.1}$$

このとき，モデルファイルとデータファイルはそれぞれ次のようになります．

AMPL 記述例 1: 問題 (4.1.1) に対するモデルファイル

```
set I;
set J;
var x{I} >= 0;
param a{I, J};
param b{J};
param c{I};
minimize Objective:
    sum{i in I} c[i] * x[i];
subject to IneqConst{j in J}:
    sum{i in I} a[i, j] * x[i] <= b[j];
end;
```

AMPL 記述例 2: 問題 (4.1.1) に対するデータファイル

```
set I := 1 2;
set J := 1 2;
param a: 1 2 :=
1 3 1
2 2 2
;
param b :=
1 12
2 8
;
param c :=
1 -1
2 -1
;
```

このように，モデルファイルでは最適化問題の構造を定義し，データファイルではその構造に現れる集合・パラメータ等の値を定義しています．

一方で，上記の最適化問題と等価な問題は次のように書くこともできます．

AMPL 記述例 3: AMPL 記述例 1, 2 と等価な記述

```
var x1 >= 0;
var x2 >= 0;
minimize Objective:
    - x1 - x2;
subject to IneqConst1:
    3 * x1 + 2 * x2 <= 12;
subject to IneqConst2:
    x1 + 2 * x2 <= 8;
end;
```

一見，こちらの記述の方が単純に見えます．しかし，問題を拡張する場合はどうでしょうか？変数が増える場合，前者であればモデルファイルの改良は必要ありませんが，後者はモデルファイルを改良する必要があります．もちろん，前者の場合はデータファイルを変更する必要がありますが，データは機械的に準備できることも多く，前者の場合が有利だと言えるでしょう．あるいは，同じ構造でデータが異なるものを何回も解く場合（このような状況はよくあります）を想像しても，前者の方が有利だと言えそうです．

後者のやり方が有利になるのは，

- 問題の規模が（手作業で書き下せる程度に）小規模で，
- かつその問題を 1 回しか解くことがない

ような場合です．

そこで本書では，モデルファイルとデータファイルを分けて書く方法を中心に解説します（このやり方を覚えておけば，仮に後者の記述方法が必要となった場合でもすぐに対応できると思います）．

なお，初学者が注意しなければならないこととして，

> モデルファイルは「プログラム」ではない

が挙げられます．一部のモデリング言語・ソルバではそのような記述を受け付けるものもあるかもしれませんが，AMPL ではそれは想定されていません．よく見かける誤りの例は，変数 x の値が 0 以上の時に変数 y の値を 1 に，そうでないときは 0 にしたい，というときに，

AMPL 記述例 4: よくある誤りの例

```
var x;
var y;
if(x >= 0) y == 1;
else       y == 0;
```

などと書いてしまうケースです．一般に，最適化ソルバにおいて，変数の値に応じて別の変数の値を定めるということはできません．これは，最適化ソルバの要求する原則である，

> 最適化問題は解く前に決定していなくてはならない

に反するためです．

Note: モデルとデータはどちらが重要か？ 上で説明したように，最適化問題を解く際にはモデルとデータが必要になります．それでは，モデルとデータはどちらが重要なのでしょうか？
対象の問題をモデル化するためには 3 章で説明したような特別な知識が必要になります．それに対し，データというのはどこかに蓄積されていてそれを取り出すだけ，というような印象があるかもしれません．しかし，多くの場合において，状況はそれほど単純ではないでしょう．例えば次のようなことが想定されます．

- あると思っていたデータがない or 一部しかない
  - 捏造するわけにはいかないし…
- 関係ある他のデータを処理して必要なデータを作成しなくてはならない
  - 誰が加工するのか？
- データはあるが利用できない
  - 個人情報，守秘義務，etc...

いくら素晴らしいモデルを作成しても，データがなくては「宝の持ち腐れ」です．モデルを作成する際には，必要なデータが揃えられるのかどうかということも十分に確認する必要があるでしょう．

4.1.4 集合の宣言

（後で見るように,）変数やパラメータ，あるいは制約条件には添字が付くことがあります．そのような時は，添字が動く範囲を定める集合を定義する必要があります．ここでは様々な集合の定義の方法について説明します．

1 次元の集合

ここでは 1 次元の集合の定義の方法を説明します．ここでは「1 次元の集合」という言い方をしていますが，これは単に要素が集まった集合のことを差しています．これとは別に，AMPL では組になった要素が集まった集合を考えることもできます．これは次項で説明します．

モデルファイルで集合 `A` を宣言するには，

AMPL 記述例 5: 集合の宣言（モデルファイル）

```
set A;
```

とします．

これだけでは集合 `A` の要素は定義されていません（「集合の宣言」と「要素の定義」の違いに注意してください）．これを定義するためには，データファイルで

AMPL 記述例 6: 要素の定義（データファイル）

```
set A := 1 2 3 4 5;
```

などとします．これによって，$A = \{1, 2, 3, 4, 5\}$ という集合が定義されます．

集合の要素は数字である必要はありません．例えば

AMPL 記述例 7: 要素が文字列の集合（データファイル）

```
set Fruit := Apple Banana Grape Orange;
```

という集合を定義することもできます．

2 次元以上の集合

前項で定義した（1 次元の）集合は，要素を集めただけのシンプルなものでした．これとは別に，要素の組を集合にして 2 次元以上の集合を作ることができます．例えばモデルファイルで 2 次元の集合 P を宣言するには，

AMPL 記述例 8: 2 次元の集合の宣言（モデルファイル）

```
set P dimen 2;
```

とします（dimen の後ろの数字は要素の組の次元に一致させます）．

このように宣言した集合に対し，その要素は例えば次のように定義します．

AMPL 記述例 9: 2 次元集合の要素の定義（データファイル）

```
set P := (A, 1), (B, 2), (C, 3);
```

このように定義すると，例えば (A, 2) や (B, 1) のような要素は P には含まれません．

ところで，

```
set I := A B C;
set J := 1 2 3;
```

に対して，2 次元の集合

```
set K := (A, 1), (A, 2), (A, 3), (B, 1), (B, 2), (B, 3),
         (C, 1), (C, 2), (C, 3);
```

が必要になる場面があるかもしれません．しかし，これは陽に準備する必要はありません．(2 次元の）集合 K は集合 I と集合 J の直積集合ですが，直積集合はモデル化の中で自然に扱うことができるようになっていますので，特別な理由のない限り，準備する必要はありません．

4.1.5　変数・パラメータの宣言

変数とパラメータの宣言や利用の方法は概ね同じであるため，ここではまとめて説明します．

添字付けのない変数・パラメータ

添字の付いていない変数・パラメータは，モデルファイル中で次のように宣言します．

AMPL 記述例 10: モデルファイルでの添字の付いていない変数・パラメータの宣言

```
var x;   # 変数
param a; # パラメータ
```

var は変数の宣言に，param はパラメータの宣言に用います．なお，AMPL では # から行末までをコメントとして扱います．

また，パラメータについてはデータファイル内で値を定義しなくてはなりません．これは次のように書くことができます．

AMPL 記述例 11: データファイルでのパラメータの値の定義

```
param a := 12.34;
```

変数には上下限制約を課す場合が多くあります．典型的な例は非負制約>= 0 でしょう．例えば変数 x に宣言時に非負制約を課すためには

AMPL 記述例 12: 変数宣言時に非負制約を課す

```
var x, >= 0;
```

と書けば実現されます（この場合，var x の後の , はなくても構いません.）また，上下限制約を同時に課すためには

AMPL 記述例 13: 変数宣言時に上下限制約を課す

```
var x, >= 0, <= 10;
```

などと書くことができます．さらに，上下限制約をパラメータで課すことも可能です．例えば次のように書くことで，変数 x には 0 <= x <= 10 という上下限制約を課すことができます．

AMPL 記述例 14: 変数宣言時にパラメータを用いて上下限制約を課す（モデルファイル）

```
param lo;  # lower limit = 下限
param up;  # upper limit = 上限
var x, >= lo, <= up;
```

AMPL 記述例 15: 変数宣言時にパラメータを用いて上下限制約を課す（データファイル）

```
param lo = 0;
param up = 10;
```

ただし，変数 x の宣言よりも前に上下限を定めるパラメータ lo，up を宣言し，かつその値をデータファイル内で定める必要があることに注意してください．

また，変数が 0-1 変数や整数変数である場合もあります．変数 x が 0-1 変数であるときは，

AMPL 記述例 16: 0-1 変数の宣言（モデルファイル）

```
var x, binary;
```

とすることで 0-1 変数であることを宣言できます．また変数 x が整数変数の場合は

AMPL 記述例 17:　整数変数の宣言（モデルファイル）

```
var x, integer;
```

と宣言します.

添字付けのある変数・パラメータ

変数やパラメータには添字が付くことがあります．例えば，添字 $i \in I$ で添字づけられた変数 x_i とパラメータ a_i はモデルファイルで次のように宣言します.

AMPL 記述例 18: モデルファイルでの添字の付いた変数・パラメータの宣言

```
set I;
var x{I};
param a{I};
```

このように，変数・パラメータの宣言時は集合だけを意識すればよく，添字は登場しません．添字を意識する必要があるのは，式の記述（次項で説明）の時です.

このように宣言されたパラメータ a にデータファイルで値を与えるには，次のように書きます.

AMPL 記述例 19: 添字付きパラメータの値の定義例 1（データファイル）

```
set I := 1 2 3 4 5;
 :
param a :=
1 10   # a[1] = 10
2 0.2  # a[2] = 0.2
3 30   # a[3] = 30
4 0.4  # a[4] = 0.4
5 50   # a[5] = 50
;
```

上記の様に添字と値を併記したものを列挙することで値が定義されます．ここでは見やすくするために改行を入れましたが,

AMPL 記述例 20: 添字付きパラメータの値の定義例 2（データファイル）

```
set I := 1 2 3 4 5;
 :
param a :=
1 10 2 0.2 3 30 4 0.4 5 50;
```

のように改行を入れなくても構いません．さらに，添字に [] を付し,

AMPL 記述例 21: 添字付きパラメータの値の定義例 3（データファイル）

```
set I := 1 2 3 4 5;
 :
param a :=
[1] 10
[2] 0.2
[3] 30
[4] 0.4
[5] 50
;
```

とすることもできます.

ここで紹介した書き方は，集合の要素が文字列で与えられている場合も同じです．例えば，

AMPL 記述例 22: 添字付きパラメータの値の定義例 4（データファイル）

```
set Fruit := Apple Banana Grape Orange;
 :
param a :=
Apple 10
Banana 0.2
Grape 30
Orange 0.4
;
```

などとすることが可能です.

さて，変数・パラメータは 2 次元以上の添字を持つこともあります．例えば，添字 $i \in I$ と添字 $j \in J$ を持つ変数 x とパラメータ a を宣言するには，

AMPL 記述例 23: 2 次元の添字付けがされた変数・パラメータの宣言（モデルファイル）

```
set I;
set J;
var x{I, J};
param a{I, J};
```

とすることで実現できます．この場合，変数 x，パラメータ a は，集合 I と集合 J の直積集合上で定義されます．すなわち，データファイルに

```
set I := A B C;
set J := 1 2 3;
```

と書かれていれば，変数 x，パラメータ a の添字は

```
(A, 1), (A, 2), (A, 3), (B, 1), (B, 2), (B, 3), (C, 1), (C, 2), (C, 3)
```

となります（前節で触れたように，集合の直積はこのように自然に実現されます）．また，3 次元以

上の場合も同様に宣言・定義することができます.

一方，AMPL 記述例 8, 9 のように集合とその要素が宣言・定義されている場合，変数 x・パラメータ a が

```
var x{P};
param a{P};
```

などと宣言されていれば，変数 x・パラメータ a の添字の範囲は (A, 1), (B, 2), (C, 3) に限定され，(A, 2) や (B, 1) のような添字を取ることはありません（前述の通りです）.

次に，これらのパラメータの値をデータファイルで定義する方法について説明します．まず，添字が直積集合で定義されている場合には，次のように書く方法があります.

AMPL 記述例 24: 2 次元のパラメータの宣言例（モデルファイル）

```
set I;
set J;
param a{I, J};
```

AMPL 記述例 25: 2 次元のパラメータの値の定義例 1（データファイル）

```
set I := A B C;
set J := 1 2 3;
param a: 1 2 3 :=
A 11 2.1 0.31
B 12 2.2 0.32
C 13 2.3 0.33
;
```

データファイルでは，パラメータ a の値が 2 次元状に並んでいます．各行の先頭には A, B, C が並んでおり，これはパラメータ a の最初の添字が集合 I を動くことと対応しています．またパラメータ a の 2 番目の添字は集合 J を動きますが，これはデータファイルでの param a: 1 2 3 := の 1 2 3 の部分にその要素が現れています（a の後ろの : を忘れないようにしてください）．結果として，行列のような形式でパラメータの値が指定されており，例えば a[A, 1] = 11, a[B, 3] = 0.32 のようになっています.

一方，これと等価な表現として次のような書き方もあります.

AMPL 記述例 26: 2 次元のパラメータの値の定義例 2（データファイル）

```
set I := A B C;
set J := 1 2 3;
param a :=
[A, 1] 11 [A, 2] 2.1 [A, 3] 0.31
[B, 1] 12 [B, 2] 2.2 [B, 3] 0.32
[C, 1] 13 [C, 2] 2.3 [C, 3] 0.33
;
```

このように要素を明示的に示した上で値を書く，という方法もあります.

パラメータの添字が 2 つの集合の直積集合上を動く場合はどちらの方法を使っても構いませんが，上述の `set P := (A, 1), (B, 2), (C, 3);` のような場合には，後者の方法で指定することになります．

さらに，3 次元以上の集合を宣言し，その集合の要素を添字とするようなパラメータを定義することもできます．例えばモデルファイルで

AMPL 記述例 27: 3 次元のパラメータの宣言例（モデルファイル）

```
set I;
set J;
set K;
param a{I, J, K};
```

と宣言したとき，データファイルで集合 I，J，K の内容が

AMPL 記述例 28: 集合の定義例（データファイル）

```
set I = i1 i2 i3;
set J = j1 j2 j3;
set K = k1 k2 k3;
```

と定義されていたら，パラメータ a は

```
(i1, j1, k1), (i1, j1, k2), (i1, j1, k3),
(i1, j2, k1), (i1, j2, k2), (i1, j2, k3),
(i1, j3, k1), (i1, j3, k2), (i1, j3, k3),
(i2, j1, k1), (i2, j1, k2), (i2, j1, k3),
(i2, j2, k1), (i2, j2, k2), (i2, j2, k3),
(i2, j3, k1), (i2, j3, k2), (i2, j3, k3),
(i3, j1, k1), (i3, j1, k2), (i3, j1, k3),
(i3, j2, k1), (i3, j2, k2), (i3, j2, k3),
(i3, j3, k1), (i3, j3, k2), (i3, j3, k3)
```

の 27 通りの添字を取るようなパラメータとなります．

このパラメータ a に値を与える方法として，先ほどの AMPL 記述例 26 のように添字を直接指定する方法があります．

AMPL 記述例 29: 3 次元の添字を持つパラメータの値の定義例 1（データファイル）

```
param a :=
[i1, j1, k1] 0.1  [i1, j1, k2] 0.2  [i1, j1, k3] 0.3
[i1, j2, k1] 0.4  [i1, j2, k2] 0.5  [i1, j2, k3] 0.6
[i1, j3, k1] 0.7  [i1, j3, k2] 0.8  [i1, j3, k3] 0.9
[i2, j1, k1] 1.0  [i2, j1, k2] 1.1  [i2, j1, k3] 1.2
[i2, j2, k1] 1.3  [i2, j2, k2] 1.4  [i2, j2, k3] 1.5
[i2, j3, k1] 1.6  [i2, j3, k2] 1.7  [i2, j3, k3] 1.8
[i3, j1, k1] 1.9  [i3, j1, k2] 2.0  [i3, j1, k3] 2.1
[i3, j2, k1] 2.2  [i3, j2, k2] 2.3  [i3, j2, k3] 2.4
[i3, j3, k1] 2.5  [i3, j3, k2] 2.6  [i3, j3, k3] 2.7
;
```

一方，2 次元の直積集合上で宣言されたパラメータの値を定義する際には AMPL 記述例 25 のように行列形式でパラメータの値を表記することが可能でしたが，3 次元以上のパラメータについてもこの表記法を流用することができます．次のように表記します．

AMPL 記述例 30: 3 次元の添字を持つパラメータの値の定義例 2（データファイル）

```
param a :=
 [i1, *, *]: k1 k2 k3 :=
   j1  0.1 0.2 0.3
   j2  0.4 0.4 0.6
   j3  0.7 0.8 0.9
 [i2, *, *]: k1 k2 k3 :=
   j1  1.0 1.1 1.2
   j2  1.3 1.4 1.5
   j3  1.6 1.7 1.8
 [i3, *, *]: k1 k2 k3 :=
   j1  1.9 2.0 2.1
   j2  2.2 2.3 2.4
   j3  2.5 2.6 2.7
  ;
```

`[i1, *, *]` という記述で，先頭の添字が `i1` であるような場合の値を定義することを宣言した上で，残り 2 次元分の添字は AMPL 記述例 25 で見たように行列形式で添字と対応する値を並べています．`i2`，`i3` の場合も同様に繰り返して書くことで，パラメータの値を定義しています．

ここでは添字が 3 次元までのパラメータについて扱いましたが，4 次元以上のパラメータについても同様に記述することができます．

4.1.6　目的関数・制約条件の記述

上で宣言した集合・変数・パラメータを用いると，目的関数や制約条件を書き下すことができるようになります．

目的関数を宣言する時には，考えている問題が最小化問題の場合は

AMPL 記述例 31: 目的関数の宣言（モデルファイル）

```
minimize Objective: [... 目的関数 ...];
```

とします．最後のセミコロンを忘れないで下さい．ここで Objective は目的関数に付ける名前で，自由に付けることができます．また考えている問題が最大化問題の場合は minimize ではなく maximize を用います．

制約条件には，添字の付かないものと添字の付くものがあります．添字の付かない制約条件の宣言は，

AMPL 記述例 32: 添字なし制約条件の宣言（モデルファイル）

```
subject to Const1: [... 制約条件 ...];
```

のように subject to から記述を始め,（目的関数と同様）最後にセミコロンを書きます．Const1 は制約条件の名前で，自由に付けることができます．一方，添字の付く制約条件の宣言ですが，例えば制約が添字 $i \in I$ で添字付けされている場合は

AMPL 記述例 33: 添字付き制約条件の宣言（モデルファイル）

```
subject to Const1 {i in I}: [... 制約条件 ...];
```

のように宣言します．

さて，目的関数の定義のためには「式」を準備する必要がありますし，制約条件の定義のためには「式 $\{\leq, \geq, =\}$ 式」を準備する必要があります．この「式」の記述の方法を見ていきます．

まず，式が添字付けされていない場合について説明します．一番単純な式は線形式でしょう．パラメータ a, b と変数 x を用いて線形式 $ax + b$ を定義するためには，a * x + b と書きます．演算子 * は省略することができません．よって，目的関数が a * x + b であれば

AMPL 記述例 34: 目的関数の例（モデルファイル）

```
minimize Objective: a * x + b;
```

とすればよいですし，a * x + b が 0 以上であるという制約条件を設けるには

AMPL 記述例 35: 添字なし制約条件の宣言（モデルファイル）

```
subject to Const1: a * x + b >= 0;
```

と書くことができます．不等式制約には <= または >= を，等式制約には == を用います（実は，比較演算子は <=，>=，== だけではなく <，>，= も使用可能です．本書では，一般のプログラミング言語で用いられる比較演算子に合わせて前者を用いて説明することにします）．

また，モデルの最後には

AMPL 記述例 36: モデルファイルの終端の記述（モデルファイル）

```
end;
```

を書きます（無くてもファイルの終端を end; と解釈してくれるようですが，その場合は警告のメッセージが出ます）．

さて，目的関数や制約条件の中では添字付き変数やパラメータなどの和を取ることがよくあります．AMPL では和を取る演算子 sum が準備されています．例えば，

```
set I;
var x{I};
param a{I};
```

と宣言されているときに，$\sum_{i \in I} a_i x_i$ を計算するには，

AMPL 記述例 37: 和を取る演算子の利用例（モデルファイル）

```
sum {i in I} a[i] * x[i]
```

と書くことで実現できます．添字 i は前もって宣言する必要はなく，使用時に {i in I} と書くことで，集合 I の要素 i と認識されます．式中で変数・パラメータの添字を書く際には [] を利用することに注意してください．ここでは集合と要素（添字）の文字を（i と I で）合わせていますが，別にその必要はありません．例えば

```
set Fruit;
var x{Fruit};
param a{Fruit};
 :
sum {j in Fruit} a[j] * x[j]
```

とすることもできます．

式中に添字を 2 つ以上持つ変数・パラメータが現れる場合は，どの添字について和を取っているのか，どの添字が和を取らずに残っているのかに注意する必要があります（3.2.1 節でも説明しました）．例えば，モデルファイルに次のような宣言が書かれている状況を想定します．

```
set I;
set J;
var x{I, J};
param a{I, J};
param b;
```

このとき，次の制約について考えてみます．

```
sum {i in I} a[i, j] * x[i, j] <= b
```

この式に現れる変数 x，パラメータ a は両方とも 2 つの添字 i，j を持っています．それに対し，和を取っている添字は i in I についてのみです．すなわち，この式全体は添字 j in J で添字付けされています．上で見たように，制約条件の宣言は subject to から始める必要がありますが，この場合は次のように宣言する必要があります．

```
subject to Const1 {j in J}: sum {i in I} a[i, j] * x[i, j] <= b
```

このように制約式全体が j in J で添字付けされていることを宣言に含める必要があります．

一方で，同じ宣言に対して次のような制約を考えることも可能です．

```
sum {i in I, j in J} a[i, j] * x[i, j] <= b
```

このように，2 つ以上の添字に対して和を取る場合は，それを , で区切って列挙することで実現できます．この場合，式中に現れる添字 i，j の両方についての和を取っていますので，式全体には添字がついていません．従って，制約条件としては

```
subject to Const1: sum {i in I, j in J} a[i, j] * x[i, j] <= b
```

というように，制約の宣言時には上の例のように添字を書く必要はありません．

さて，添字を用いて和を取るときに，様々な制限を付けたい場合があると思います．例えば集合・変数・パラメータ等が次のように宣言・定義されているときを考えます．

AMPL 記述例 38: 集合・変数・パラメータの宣言例（モデルファイル）

```
set I;
var x{I};
param a{I};
param b;
```

AMPL 記述例 39: 集合の要素の定義例（データファイルの一部）

```
set I := 1 2 3 4 5;
```

このとき，次のような制約を記述することが可能です．

AMPL 記述例 40: 和を取る範囲を限定する例 1（モデルファイル）

```
subject to Const1: sum {i in I: i <= 3} a[i] * x[i] <= b;
```

この制約は，

```
subject to Const1: a[1] * x[1] + a[2] * x[2] + a[3] * x[3] <= b;
```

と等価になります．sum の直後の {} 内で，添字 i in I で和をとることを宣言していますが，その後に : に続けて和を取る範囲を限定する条件文を追加しています．

条件文は複数列挙することも可能です．例えば，上記 AMPL 記述例 38, 39 の設定の下で，モデルファイルに

AMPL 記述例 41: 和を取る範囲を限定する例 2（モデルファイル）

```
subject to Const2: sum {i in I: i >= 2 && i <= 4} a[i] * x[i] <= b;
```

とすればこれは

```
subject to Const2: a[2] * x[2] + a[3] * x[3] + a[4] * x[4] <= b;
```

と等価になります．また

AMPL 記述例 42: 和を取る範囲を限定する例 3（モデルファイル）

```
subject to Const3: sum {i in I: i <= 2 || i == 5} a[i] * x[i] <= b;
```

とすると，これは

```
subject to Const3: a[1] * x[1] + a[2] * x[2] + a[5] * x[5] <= b;
```

と等価になります．

さて，このような条件指定は，パラメータを介しても指定可能です．例えば，次のようなモデルファイル・データファイルがあるものとします．

AMPL 記述例 43: 集合・変数・パラメータの宣言例（モデルファイル）

```
set I;
var x{I};
param a{I};
param b;
param d{I};
```

AMPL 記述例 44: 集合の要素の定義例（データファイルの一部）

```
set I := 1 2 3 4 5;
param d :=
[1] 1 [2] 0 [3] 1 [4] 0 [5] 1;
```

このとき，

AMPL 記述例 45: 和を取る範囲を限定する例 4（モデルファイル）

```
subject to Const4: sum {i in I: d[i] == 1} a[i] * x[i] <= b;
```

とすると，これは

```
subject to Const4: a[1] * x[1] + a[3] * x[3] + a[5] * x[5] <= b;
```

と等価になります．

ただし，条件式に変数を含めることはできません．これは，最初に述べた大原則

> 最適化問題は解く前に決定していなくてはならない

に反することになるためです．

第5章　ソルバによる求解と解の分析

本章で学ぶ内容

- ソルバによる最適化問題の求解
- ソルバで得られた解の分析方法
- ケーススタディ

この章では，ソルバを用いた最適化問題の求解，並びに得られた解を分析する方法について学びます．

5.1　ソルバによる求解

ソルバを用いて最適解を求めるためには，4 章で学んだ知識を元に，解きたい問題のモデルファイル・データファイルを準備します．そしてこれらをソルバで処理します．

ソルバの利用方法は，ソルバによって大きく異なります．ソルバ固有の GUI を持つものが多くありますし，多くのソルバはコマンドラインから起動することができます．その具体的な方法は付録 D を参照してください．

ソルバで問題を解くと，以下の情報を得ることができます：

- 最適化計算のログ
- 最適解（ないしは暫定解）
- Karush-Kuhn-Tucker 条件（後述）の残差（連続計画問題の場合）
- 上界と下界（後述）（(混合）整数計画問題の場合）
- 最適解における制約条件の値
- 最適解における制約条件の双対変数（後述）の値

以下では，これらの情報の意味について説明します．

5.2　得られた解の最適性

ソルバは「最適性の条件」を調べることにより，得た解が最適解（に十分近い）かどうかを調べています．調べている条件は，連続計画問題と（混合）整数計画問題で異なります．以下，その条件を説明します．

5.2.1　連続計画問題に対する最適性の条件

連続計画問題で調べている最適性の条件は**Karush-Kuhn-Tucker 条件**（KKT 条件）といいます．連続計画問題の一般的な形は問題 (2.2.2)（ないしは問題 (2.2.8)）で与えられます．この問題に対する KKT 条件を考えるためには，まず**Lagrange 関数**を定義する必要があります．(2.2.2) の Lagrange 関数は

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}) + \sum_{j=1}^{l} \mu_j h_j(\boldsymbol{x}) \tag{5.2.1}$$

で定義されます．ここで λ_i $(i = 1, \ldots, m)$, μ_j $(j = 1, \ldots, l)$ は**Lagrange 乗数**と呼ばれるものです．この Lagrange 関数 (5.2.1) を用いると，KKT 条件は次のように書くことができます．

$$\left| \begin{array}{l} \nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \nabla f(\boldsymbol{x}) + \displaystyle\sum_{i=1}^{m} \lambda_i \nabla g_i(\boldsymbol{x}) + \sum_{j=1}^{l} \mu_j \nabla h_j(\boldsymbol{x}) = \boldsymbol{0} \\ g_i(\boldsymbol{x}) \leq 0,\ \lambda_i \geq 0,\ g_i(\boldsymbol{x})\lambda_i = 0 \quad (i = 1, \ldots, m) \\ h_j(\boldsymbol{x}) = 0 \quad (j = 1, \ldots, l) \end{array} \right. \tag{5.2.2}$$

ソルバは，条件 (5.2.2) を満たすような変数 $\boldsymbol{x}$ と Lagrange 乗数 $\boldsymbol{\lambda}, \boldsymbol{\mu}$ の値を求めることを目標にしています．

KKT 条件を細かく説明することは本書の範囲を超えますのでここでは割愛しますが，大事なことは条件が等式・不等式で書かれているということです．すなわち，ある $\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}$ の値を定めれば，それを (5.2.2) の各式にそれを代入し，等式・不等式の違反量（これを**残差**といいます）を調べることにより，現在の $\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}$ が最適解にどの程度近いのかを計ることができます．

実は，KKT 条件は「最適性の（1 次の）必要条件」であり，十分条件ではありません．つまり，KKT 条件を満たすような $\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}$ を求めても，理論的にはこれが最適解ではない可能性がある，ということになります．ただし，（最適化問題を構成する関数が微分可能であるような）凸計画問題であれば KKT 条件は最適性の必要十分条件になることが知られています．

また，非線形最適化問題の場合も，ソルバは KKT 条件を満たすような $\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}$ を求めようとしますが，これは局所的最適解を求めようとしていることになり，必ずしも大域的最適解を求めることにはなりません．一般に，非線形最適化問題の大域的最適解を求めること，あるいは得られた局所的最適解が大域的最適解であることを保証することは非常に難しく，ソルバを用いても大域的最適解が求められる問題は限定的です．そのため，通常のソルバでは KKT 条件を満たすような点を求め，それで計算を終了します [1]．

5.2.2　（混合）整数計画問題に対する最適性の条件

ソルバで（混合）整数計画問題を解くときには，ソルバ内では**緩和問題**が解かれます．これは元の問題の制約条件を何らかの意味で緩和したものです．例えば，変数 x が 0-1 変数であるとき，これを $0 \leq \bar{x} \leq 1$ であるような変数 $\bar{x}$ で置き換えた問題は，元の制約条件（変数の 0-1 性）を緩和していることになります．

[1] ただ，問題の性質が悪くなければ，KKT 条件を満たすような $\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}$ を求めれば，それが十分によい局所的最適解（あるいは大域的最適解）になる場合も多くあります．

ここで，元の問題が最小化問題であれば，

「緩和問題の最適解における目的関数値」≤「元の問題の最適解における目的関数値」

となることに注意します．すなわち，緩和問題を解けば,「元の問題の最適解における目的関数値」の**下界値**を得る事ができます．そして，様々な緩和問題を解き，その中で最も良い下界値（= 最小化問題であれば最大の下界値）が「最適解に最も近い下界値 (♣)」であると言えます．

一方，ソルバは，整数変数の組合せを固定した問題も解いています．変数を固定するわけですから，これは制約条件を強めていることになります．すなわち，この問題を解いて得られた（実行可能）解は,「元の問題の最適解における目的関数値」の**上界値**を与えてくれます．そして，整数変数の組合せを変えながら問題を繰り返し解き，最も良い上界値（= 最小化問題であれば最小の上界値）が「最適解に最も近い上界値 (♠)」であると言えます．

そして，(♣) = (♠) となるような解，すなわち上界値と下界値の**ギャップ**を見いだせば，それが最適解に他ならないということになります．ソルバで（混合）整数計画問題を解くと，内部ではこのような計算が行われ，最適解を求めています．

5.3 ソルバの出力の確認

5.1 節で述べたように，ソルバで最適化計算を実行すると，様々な情報を得ることができます．ここでは，その出力をどのような点に注意して確認すべきかについて説明します．

なお，出力の方法はソルバによって（場合によっては）大きく異なります．以下に説明する情報がどのように出力されるのかについては，利用するソルバのマニュアル等を参考にしてください．以下では，フリーのソルバである GLPK の出力を例に取りながら説明します．

5.3.1 最適解が得られているかどうか

まず，何より先に確認すべき点は，最適解が得られているかどうかです．一般に，ソルバが計算を終了した場合（バグなどでの不正終了は除く），次のいずれかの状態になっているはずです．

(i) 最適解を得ている

(ii) 最適解を見つけることはできていないが暫定解を得ている

(iii) 問題が実行不可能であるとわかった

(iv) 問題が有界でないことがわかった

(v) 問題についての特別な情報が得られていない

以下，これらの状態について説明します．

ソルバが最適解を得ている場合

まず,「(i) 最適解を得ている」状態です．これは文字通り「最適解が得られている」わけですから，（言うまでもありませんが）理想的な状態です．GLPK では，標準出力（Windows 環境ではコマンドプロンプト，Mac/Unix 環境ではコンソール）に

```
OPTIMAL SOLUTION FOUND           （※連続計画問題の場合）
INTEGER OPTIMAL SOLUTION FOUND   （※（混合）整数計画問題の場合）
```

と出力されます[2]．また解ファイルを生成している場合には，そこにも

```
Status:     OPTIMAL           （※連続計画問題の場合）
Status:     INTEGER OPTIMAL   （※（混合）整数計画問題の場合）
```

という出力があります．このような出力が得られた場合は，ソルバは最適解を求めることができた，ということになります．

ソルバが暫定解を得て終了している場合

ソルバを実行するときに，計算時間に上限を設けることがあります．その時間内に最適解を求めることができなかった場合，計算終了までに得られた最良の実行可能解を**暫定解**として出力することがあります．また，解いている問題が非常に大規模な問題で，計算途中でメモリ不足に陥った場合もこの出力を行うことがあります．

例えば，GLPK で混合整数計画問題を解くとき，60 秒の時間制限を設けて問題を解いたものの，最適解を求めることができなかった場合には，標準出力に

```
TIME LIMIT EXCEEDED; SEARCH TERMINATED
Time used:   60.0 secs
```

と表示されます．また解ファイルには

```
Status:     INTEGER NON-OPTIMAL
```

と表示され，最適解でない (NON-OPTIMAL) ということがわかるようになっています．

問題が実行不可能な場合

ソルバに問題を解かせたものの，それが実行不可能である場合もあり得ます（多くの場合は，モデルやデータに間違いのある場合でしょう）．ソルバがそれを検知した場合は，そのことを教えてくれます．例えば，実行不可能な連続計画問題を GLPK で解いた場合，標準出力には

```
PROBLEM HAS NO PRIMAL FEASIBLE SOLUTION
```

と出力され（“feasible solution”= 実行可能解）[3]，解ファイルには

[2] GLPK では連続計画問題と（混合）整数計画問題で出力を変えていますが，他のソルバでは同一の出力をする場合もあります．

[3] “primal” について：最適化問題を解く際に，元の問題を「**主問題** (primal problem)」とし，これに対する「**双対問題** (dual problem)」を考えることがあります．ここでいう primal はその主問題のことを指しています．本書では双対問題の持つ意味についての説明は割愛します．多くの最適化問題の教科書（例えば [11, 12]）に記述がありますのでそちらを参考にしてください．

```
Status:     UNDEFINED
```

と出力されます．また実行不可能な（混合）整数計画問題を解いた場合は，標準出力に

```
PROBLEM HAS NO INTEGER FEASIBLE SOLUTION
```

と出力され，解ファイルには

```
Status:     INTEGER EMPTY
```

と出力されます．

問題が有界でない場合

最適化問題によっては，最小化問題で目的関数値をいくらでも小さくできる（最大化問題であればいくらでも大きくできる）ような場合があります（これも前節と同様，多くの場合はモデルやデータに間違いのある場合でしょう）．このような場合，問題が**有界でない**といいます．このような問題をGLPK で解くと，標準出力に

```
PROBLEM HAS UNBOUNDED SOLUTION
```

と表示されます（連続/（混合）整数計画問題とも）．また解ファイルには

```
Status:     UNDEFINED           （※連続計画問題の場合）
Status:     INTEGER UNDEFINED   （※（混合）整数計画問題の場合）
```

と出力されます．

問題に関する特別な情報が得られていない場合

設定した計算時間の上限，あるいは使用可能なメモリ量の範囲内では，意味のある情報が得られない場合があり得ます．例えば計算時間の上限に達している場合，標準出力に

```
TIME LIMIT EXCEEDED; SEARCH TERMINATED
```

という表示がされますが，さらに解ファイルに

```
Status:     UNDEFINED
```

と出力されていれば，これは指定した時間の範囲内ではこの問題に関する意味のある情報（例えば暫定解）が得られていない，ということになります．

このように，ソルバの計算が終了した場合であっても，必ずしも最適解を得られているわけではないことに注意しましょう．次節以降では，最適解が得られている場合に確認すべき情報について説明します．

5.3.2　目的関数の値

ソルバによって最適解や暫定解が得られているときに，まず気にすべきことは目的関数の値でしょう．一般に，最適解における目的関数の値は解ファイルを見れば確認することができます．例えばGLPK では

```
Objective:  f = 365.22 (MINimum)
```

のように表記されます（f はモデルファイル内で目的関数に付けた名前です）．

問題を解く前に，最適解において目的関数の値がどの程度になるのかの見当を付けておくとよいでしょう．そうしておけば，もし目的関数の値が想定できないような値になっていた場合，モデル・データに誤りがある（あるいは自分の見当に誤りがある）ことにすぐに気づくことができます．

5.3.3　変数・制約の値

最適化問題において，制約のことを ‘Row’（行），変数のことを ‘Column’（列）で表現することがあります．これは，LP の標準形 (2.2.6) において，制約条件 $\boldsymbol{Ax} = \boldsymbol{b}$ に現れる係数行列 $\boldsymbol{A}$ の行が制約に，列が変数に対応していることによるものです．

GLPK の解ファイルの先頭部分には，

```
Rows:       441
Columns:    419 (400 integer, 400 binary)
Non-zeros:  2264
Status:     INTEGER OPTIMAL
Objective:  f = 365.22 (MINimum)
```

のような出力が見られます．これは，‘Rows’= 制約の数が 441，‘Columns’= 変数の数が 419（そのうち 400 個が ‘binary’= 0-1 整数変数，$419 - 400 = 19$ 個が連続変数）ということがわかります．またこの問題は MILP なのですが，問題を標準形 (2.2.6)（ただし変数の一部は整数変数）としたときの係数行列 $\boldsymbol{A}$ に現れる非零要素の数が 2264 であることも記載されています．

通常,「最適解」とは変数の値のことを指します．その意味で，変数の値をチェックするのは非常に大事なことです．ただ，問題が大規模になると，その全てを一つ一つチェックするのは現実的ではないかもしれません．一方で，変数には，問題の中で主要な意味を持つものがあるはずです．まずはそういった変数に注目してチェックするのがよいでしょう．

上で説明したように，変数の値は ‘Column(s)’ というキーワードで解ファイルの中に書かれている場合があります．あるいは直接的に ‘Variable(s)’（変数）と書かれている場合もあります．

ここで GLPK の解ファイルの例を見ておきましょう．

```
   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 x[A,1]       NL             0             0                       < eps
     2 x[A,2]       NL             0             0                           3
     3 x[A,3]       B             15             0
     4 x[A,4]       B             45             0
     5 x[B,1]       NL             0             0                           1
```

一番左の列 No. は変数番号を，Column name は変数名を表します．添字付きの変数の場合は [A,1] のように添字が併記されます．そして Activity の列が変数の値を表しています．また，Lover bound, Upper bound は変数の上下限を表しています．すなわち，変数の値はこの上下限を満たしている必要があります．St, Marginal については 5.3.4 節で説明します．その他，詳細については [28] を参照してください．

また，制約に関する情報は以下のように表示されます．

```
   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 Objective    B            630
     2 C1[A]        NU            60                         60            -4
     3 C1[B]        B             35                         45
     4 C1[C]        B             35                         50
     5 C2[1]        NS            35            35            =             8
```

Row name はモデルファイル中で目的関数・制約条件に付けた名前を表しています．1 行目は目的関数を表しており，2 行目以降が制約条件に関する情報です．また，Activity の列が最適解における制約条件の値を表しており，これが上下限（Lover bound, Upper bound）を満たしている必要があります．

5.3.4 最適性の条件に関する情報（連続計画問題）

5.2.1 節で説明したように，連続計画問題においては，最適解において KKT 条件 (5.2.2) が成立していなくてはなりません．GLPK で LP を解いた際の解ファイルを見てみると，以下のように KKT 条件に関する情報も記載されています．

```
Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err = 2.76e-13 on row 7
        max.rel.err = 1.01e-16 on row 69
        High quality

KKT.PB: max.abs.err = 3.91e-27 on row 37
        max.rel.err = 3.91e-27 on row 37
        High quality

KKT.DE: max.abs.err = 7.24e-14 on column 24
        max.rel.err = 1.12e-16 on column 24
        High quality

KKT.DB: max.abs.err = 1.69e-16 on column 74
        max.rel.err = 1.69e-16 on column 74
        High quality
```

KKT.PE, KKT.PB, KKT.DE, KKT.DB は，問題を LP の標準形 (2.2.6) に直したときの KKT 条件の違反量を表しています．この違反量が何を表しているのかについては [28] に譲りますが，大事なのはいずれのパートにも High quality とあることです．これは，違反量が十分に小さいということを意味します．実際，違反量として書かれている量はいずれも 10^{-10} より小さな値になっています．これは数値誤差程度の値であり，十分な精度で KKT 条件を満たしていること示しています．

GLPK の解ファイルにおいて，St は制約（変数の上下限制約を含む）の状態を表しています．具体的には，表 5.1 のような意味を持っています．例えば，$-1 \leq 2x - 1 \leq 3$ という不等式制約がある

表 5.1: 制約の状態

St	意味
B	不等式制約に余裕がある
NL	不等式制約が下限で満たされている
NU	不等式制約が上限で満たされている
NS	等式制約が満たされている

ものとします．もし最適解において $x = 1$ であれば，この不等式は上下限のいずれにも余裕がありますので，St 列には B と表記されます．あるいは $x = 0$ であればこの制約が下限に達しているので NL，$x = 2$ であれば NU となります．さらに等式制約であれば（最適解が求まっていれば）NS と表記されます．

さらに，解ファイルには Marginal という列があります．ここには値が記載されている場合とされてない場合がありますが，よく見てみると，値が書かれているのは St 列が N で始まっている場合です．実はこの列は，KKT 条件に現れているラグランジュ乗数の値を表しています．ソルバはこの値を用いて KKT 条件の値を計算し，最適性の条件が十分に満たされているかどうかを調べています．

5.3.5　最適性の条件に関する情報（(混合）整数計画問題）

5.2.1 節で説明したように,（混合）整数計画問題の最適性の条件は，問題の上界値と下界値が一致すること，すなわち，上界値と下界値のギャップが 0 になることでした．GLPK の場合，これに関する情報は，求解時に標準出力に表示されます．

次の例を見てみましょう．

```
      0: obj =   1.326940000e+03  infeas =  1.425e+02 (1)
*    39: obj =   1.242369474e+03  infeas =  0.000e+00 (1)
*   149: obj =   2.578842105e+02  infeas =  6.939e-17 (1)
OPTIMAL SOLUTION FOUND
Integer optimization begins...
+   149: mip =     not found yet >=              -inf        (1; 0)
+   653: >>>>>   5.148000000e+02 >=   2.845515789e+02  44.7% (70; 1)
+  1367: >>>>>   5.077500000e+02 >=   2.906205263e+02  42.8% (157; 3)
+  2098: >>>>>   4.571100000e+02 >=   2.919126316e+02  36.1% (233; 12)
 （中略）
+708263: mip =   3.652200000e+02 >=   3.593894737e+02   1.6% (12184; 55913)
+734862: mip =   3.652200000e+02 >=   3.606594737e+02   1.2% (9352; 63694)
+764640: mip =   3.652200000e+02 >=   3.624505263e+02   0.8% (5302; 76914)
+791854: mip =   3.652200000e+02 >=     tree is empty   0.0% (0; 106121)
INTEGER OPTIMAL SOLUTION FOUND
```

4 行目で OPTIMAL SOLUTION FOUND と書いてありますが，これで最適解を得たと思ってはいけません．これは一番最初の緩和問題が解けた，ということを意味しています．整数変数を意識した最適化計算は Integer optimization begins... から始まっています．そして，右の方に現れている 44.7%, 1.6% などの数値が上界値と下界値のギャップを表しています [4]．これが 0.0% になれば,（混

[4]GLPK ではギャップを「(上界値 − 下界値) / 上界値」で定義しています．この場合でも,「ギャップ = 0」⇔「上界値と下界値が一致」ですので，本質的な差はありません．

合）整数計画問題において最適解を求めることができた，ということになります（実際，上記の例でも 0.0% の行の直後に INTEGER OPTIMAL SOLUTION FOUND という行があります）.

5.3.6 他のソルバでの出力

上で挙げたような出力は，形式が異なることはあれども，全てのソルバで同様の出力がなされているはずです．その他には，計算に要した時間などもどこかに表示されているものと思われます.

5.4 （連続最適化問題に対する）感度分析

連続最適化問題の制約条件のうち，最適解において等号が成立している制約条件を，その最適解において**有効な制約条件**といい，そうでない制約条件を**有効でない制約条件**といいます．別の言い方をすると，不等式制約のうち余裕があるものは「有効でない制約条件」であり，余裕のない不等式制約と全ての等式制約は「有効な制約条件」ということになります[5].

ここで，有効な/有効でない制約条件と目的関数値の関係について考えます．有効でない（不等式）制約については，最適解を拘束していないため，目的関数に直接の影響は及ぼしていません．一方，有効な制約条件は最適解を拘束しているため，この制約条件が少し変わると目的関数値も変わるものと考えられます．この関係を調べることを**感度分析**と言います.

実は,「制約条件が少し変わったときに目的関数値に及ぼす影響の度合い」を表しているのが，5.3.3, 5.3.4 節で扱った，解ファイル中の Marginal の値です．以下，例を通じてこのことを確認してみましょう.

例 5.4.1 Hitchcock 型輸送問題（問題 6.8.1，不等式制約について）
ここでは *6.8* 節で扱う *Hitckcock* 型輸送問題を先取りし，この問題の 不等式制約 を例として取り上げます．問題の定義と定式化は *6.8.1* 節を，モデルは *6.8.2* 節を参照してください．またデータは表 *5.2* のものを用います．これは *C.8* 節に示したものと同じです．これらを用いて GLPK により解を

表 5.2: 各工場の供給可能量 a_i，各都市の需要 b_j，製品 1 単位あたりの輸送費用 c_{ij}

輸送費用	都市 1	2	3	4	供給可能量
工場 A	4	1	3	3	60
B	9	2	7	10	45
C	8	3	10	9	50
需要	35	15	35	45	–

求めると，次のような解ファイルを得ます（一部抜粋）.

```
Problem:    Hitchcock
Rows:       8
Columns:    12
Non-zeros:  36
Status:     OPTIMAL
Objective:  Objective = 630 (MINimum)
```

[5]等式制約は常に有効なため，これを「有効な制約条件」に含めない（すなわち余裕のない不等式制約のみを「有効な制約条件」とする）教科書もあります.

```
   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------ ------------ ------------
     1 Objective    B            630
     2 C1[A]        NU            60                        60           -4
     3 C1[B]        B             35                        45
     4 C1[C]        B             35                        50
     5 C2[1]        NS            35           35            =            8
     6 C2[2]        NS            15           15            =            2
     7 C2[3]        NS            35           35            =            7
     8 C2[4]        NS            45           45            =            7

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------ ------------ ------------
     1 x[A,1]       NL            0            0                      < eps
     2 x[A,2]       NL            0            0                          3
     3 x[A,3]       B            15            0
     4 x[A,4]       B            45            0
     5 x[B,1]       NL            0            0                          1
     6 x[B,2]       B            15            0
     7 x[B,3]       B            20            0
     8 x[B,4]       NL            0            0                          3
     9 x[C,1]       B            35            0
    10 x[C,2]       NL            0            0                          1
    11 x[C,3]       NL            0            0                          3
    12 x[C,4]       NL            0            0                          2
```

不等式制約 C1 は工場 A, B, C の供給可能量の上限 $(60, 45, 35)$ を定めている制約条件です．ここで，工場 A については上限いっぱいの量を供給されることが求められています（Activity = Upper bound）．一方，工場 B, C については供給量に余裕のある状況であることがわかります（Activity < Upper bound）．そして，前者の場合は Marginal に値が記載されているのに対し，後者には Marginal の値が記載されていないことがわかります．

これは，KKT 条件 (5.2.2) のうち不等式制約に対する部分，

$$g_i(\boldsymbol{x}) \leq 0,\ \lambda_i \geq 0,\ g_i(\boldsymbol{x})\lambda_i = 0 \quad (i = 1, \ldots, m)$$

に対応しています．すなわち，$g_i(\boldsymbol{x})$ と λ_i の積が 0 にならなくてはならないことに注意すると，不等式制約で等号が成立している場合 $(g_i(\boldsymbol{x}) = 0)$ は *Lagrange* 乗数 λ が 0 より大きな値を持つことができ，等号が成立していない場合 $(g_i(\boldsymbol{x}) < 0)$ には *Lagrange* 乗数は 0 になる，ということです．ただし，ここでは不等式制約に対する Marginal の値が負の値で表示されていることに注意してください．一般に，最適化ソルバでは内部で独自に標準形を定め，入力された問題を標準形に変換しています（例えば目的関数は「最小化」で統一するか「最大化」とするのか，あるいは不等式制約は ≤ 0 の形にするか ≥ 0 の形にするか，など）．そして，ソルバはそれに合わせた各種出力を行うため，値の正負については (5.2.2) で示した形と異なる出力が得られる場合があるため，扱いに注意が必要です．ただし，どのソルバであっても絶対量は（通常）一致します．

それでは，この値がどのような意味を持つのかを確認するために，工場 A の供給可能量の上限を 60 から 59.9 に変えて問題を解いてみます．その結果は次のようになります（一部抜粋）．

```
Problem:    prob_ex_transport
Rows:       8
Columns:    12
Non-zeros:  36
```

```
Status:     OPTIMAL
Objective:  Objective = 630.4 (MINimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 Objective    B          630.4
     2 C1[A]        NU          59.9                       59.9            -4
     3 C1[B]        B           35.1                         45
     4 C1[C]        B             35                         50
     5 C2[1]        NS            35            35             =             8
     6 C2[2]        NS            15            15             =             2
     7 C2[3]        NS            35            35             =             7
     8 C2[4]        NS            45            45             =             7

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 x[A,1]       NL            0             0                       < eps
     2 x[A,2]       NL            0             0                           3
     3 x[A,3]       B          14.9             0
     4 x[A,4]       B            45             0
     5 x[B,1]       NL            0             0                           1
     6 x[B,2]       B            15             0
     7 x[B,3]       B          20.1             0
     8 x[B,4]       NL            0             0                           3
     9 x[C,1]       B            35             0
    10 x[C,2]       NL            0             0                           1
    11 x[C,3]       NL            0             0                           3
    12 x[C,4]       NL            0             0                           2
```

目的関数値が 630 から 630.4 に変化していることがわかります．この変化の絶対量 $|630-630.4| = 0.4$ は，制約条件 `C1[A]` の定数部分の変化量 $60.0-59.9=0.1$ に，この制約条件の `Marginal` の値の絶対値 4 を乗じた値 $0.1\times 4 = 0.4$ に一致していることがわかります．また，`C1[A]` の定数部分の変化は制約条件を厳しくする方向の変化であるため，目的関数値は悪くなっている（この問題は最小化問題）ことがわかります．

あるいは工場 A の供給可能量の上限を 60 から 60.1 に変えると，結果は次のようになります（一部抜粋）．

```
Problem:    prob_ex_transport
Rows:       8
Columns:    12
Non-zeros:  36
Status:     OPTIMAL
Objective:  Objective = 629.6 (MINimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 Objective    B          629.6
     2 C1[A]        NU          60.1                       60.1            -4
     3 C1[B]        B           34.9                         45
     4 C1[C]        B             35                         50
     5 C2[1]        NS            35            35             =             8
     6 C2[2]        NS            15            15             =             2
     7 C2[3]        NS            35            35             =             7
     8 C2[4]        NS            45            45             =             7
```

```
   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 x[A,1]       NL            0             0                       < eps
     2 x[A,2]       NL            0             0                           3
     3 x[A,3]       B          15.1             0
     4 x[A,4]       B            45             0
     5 x[B,1]       NL            0             0                           1
     6 x[B,2]       B            15             0
     7 x[B,3]       B          19.9             0
     8 x[B,4]       NL            0             0                           3
     9 x[C,1]       B            35             0
    10 x[C,2]       NL            0             0                           1
    11 x[C,3]       NL            0             0                           3
    12 x[C,4]       NL            0             0                           2
```

やはり，目的関数値の変化の絶対量 $|630 - 629.6| = 0.4$ は $0.1 \times 4 = 0.4$（制約条件 `C1[A]` の定数部分の変化量 $60.0 - 59.9 = 0.1$ × `C1[A]` の `Marginal` の値）になっています．

このように，`Marginal` の値は，その制約が変化したときに目的関数値に与える影響を表しています．その観点から，この値を**潜在価格**や**シャドウ・プライス**ということがあります．ここでは工場 A の供給可能量の上限に関する潜在価格（シャドウ・プライス）が 4 であるわけですが，これは例えば次のような解釈が可能です：

- 工場 A が老朽化して供給可能量が 1 単位減少すると，費用が 4 だけ増加します．
- 工場 A に設備投資して供給可能量を 1 単位増やすための費用は 4 以下に抑える必要があります．

□

例 5.4.2 Hitchcock 型輸送問題（問題 6.8.1，等式制約について）

例 *5.4.1* では *Hitckcock* 型問題の不等式制約を取り上げましたが，ここでは等式制約 を取り上げます．

上で示したように，*6.8.2* 節で示したモデル，表 *5.2* で示したデータ（*C.8* 節で示したものと同じ）で問題を解くと，等式制約（`C2`）に関する出力は以下のようになります（一部抜粋，再掲）．

```
   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     5 C2[1]        NS           35            35             =             8
     6 C2[2]        NS           15            15             =             2
     7 C2[3]        NS           35            35             =             7
     8 C2[4]        NS           45            45             =             7
```

これを潜在価格の観点から分析すると，

> 都市 2 の `Marginal` が都市 $1, 3, 4$ と比べて小さい
> ⇒ 各都市で需要が 1 単位増えたときに輸送費用の増加が一番小さいのは都市 2
> ⇒ 製品を売り込む場合（各都市の需要を喚起する場合）は都市 2 を優先すると利幅が大きい

という考えが成立することになります．実際，各都市の需要を 1 単位ずつ増加させた場合，目的関数値は表 *5.3* のようになります．□

表 5.3: 目的関数値の変化

変化	目的関数値
変化なし	`Objective:  Objective = 630 (MINimum)`
$b_1 = 35 \to b_1 = 36$	`Objective:  Objective = 638 (MINimum)`
$b_2 = 15 \to b_2 = 16$	`Objective:  Objective = 632 (MINimum)`
$b_3 = 35 \to b_3 = 36$	`Objective:  Objective = 637 (MINimum)`
$b_4 = 45 \to b_4 = 46$	`Objective:  Objective = 637 (MINimum)`

表 5.4: 各工場の供給可能量 a_i，各都市の需要 b_j，製品 1 単位あたりの輸送費用 c_{ij}（例 5.4.1 から変更）

輸送費用	都市 1	2	3	4	供給可能量
工場 A	4	1	3	3	60
B	9	2	7	10	45
C	8	1	10	9	35
需要	35	15	35	45	–

例 5.4.3 輸送問題（問題 6.8.1，データを変更）
ここでは，例 *5.4.1* で扱った問題のデータを表 *5.4* のように変更した場合を考えてみましょう．表 *5.2* と表 *5.4* のデータで異なる点は，

- 工場 C の供給可能量が 50 から 35 に減少
- 工場 C から都市 2 への輸送費用が 3 から 1 に減少

の 2 箇所です．
表 *5.4* のデータで問題を解いた結果は次のようになります．

```
Problem:    Hitchcock
Rows:       8
Columns:    12
Non-zeros:  36
Status:     OPTIMAL
Objective:  Objective = 615 (MINimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 Objective    B            615
     2 C1[A]        NU            60                          60            -5
     3 C1[B]        B             35                          45
     4 C1[C]        NU            35                          35            -1
     5 C2[1]        NS            35            35             =             9
     6 C2[2]        NS            15            15             =             2
     7 C2[3]        NS            35            35             =             7
     8 C2[4]        NS            45            45             =             8

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
```

```
------ ------------ -- ------------ ------------ ------------ ------------
     1 x[A,1]       B             15            0
     2 x[A,2]       NL             0            0                         4
     3 x[A,3]       NL             0            0                         1
     4 x[A,4]       B             45            0
     5 x[B,1]       B              0            0
     6 x[B,2]       NL             0            0                     < eps
     7 x[B,3]       B             35            0
     8 x[B,4]       NL             0            0                         2
     9 x[C,1]       B             20            0
    10 x[C,2]       B             15            0
    11 x[C,3]       NL             0            0                         4
    12 x[C,4]       NL             0            0                         2
```

この結果では，工場 A と C の供給量が上限に達していることに注意してください．

このとき，工場 A の供給可能量を 60.1 として問題を解くと，次のような結果になります（一部抜粋）．

```
Status:     OPTIMAL
Objective:  Objective = 614.6 (MINimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------ ------------ ------------
     1 Objective    B          614.6
     2 C1[A]        NU          60.1                      60.1           -4
     3 C1[B]        B           34.9                        45
     4 C1[C]        NU            35                        35       < eps
```

これをみると，目的関数値は例 *5.4.1, 5.4.2* での分析から期待される $615 - 0.1 \times 5 = 614.5$ にはなっていません．また Marginal の値も変わっている（C1[C] も）ことがわかります．

この例からわかるのは，Marginal の値はその最適解での潜在価格であり，最適解が変わればこの値は変わりうる，ということです．言い換えると，Marginal の値（潜在価格）から目的関数値の変化を見積もる際には，適用範囲に限界がある，ということもできます．　□

例 5.4.4 最大流問題（問題 6.6.1）
ここでは，*6.6* 節で扱う最大流問題を先取りします．問題の定義と定式化は *6.6.1* 節を，モデルは *6.6.2* 節を参照してください．またデータは *C.6* 節に示したものを用います．この最大流問題を GLPK で解いた結果は次のようになります（一部抜粋）[6]．

```
Status:     OPTIMAL
Objective:  Objective = 99 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------ ------------ ------------
    12 UpperLimit[0,1]
                    B             28                        50
    13 UpperLimit[0,2]
                    NU            50                        50       < eps
    14 UpperLimit[0,3]
                    B             21                        50
    15 UpperLimit[1,4]
                    B             16                        18
```

[6] GLPK のバージョンによっては，ここに示したものとは異なる結果を得る場合がありますが，これ以降の議論は同様に成立するはずです．

```
16 UpperLimit[1,5]
                NU              12                              12              1
17 UpperLimit[2,4]
                B               18                              20
18 UpperLimit[2,5]
                NU              15                              15              1
19 UpperLimit[2,6]
                NU              17                              17              1
20 UpperLimit[3,5]
                NU              10                              10              1
21 UpperLimit[3,6]
                NU              11                              11              1
22 UpperLimit[4,7]
                NU              23                              23              1
23 UpperLimit[4,8]
                NU              11                              11              1
24 UpperLimit[5,7]
                NU              17                              17          < eps
25 UpperLimit[5,8]
                NU              19                              19          < eps
26 UpperLimit[5,9]
                B                1                              25
27 UpperLimit[6,8]
                NU              16                              16          < eps
28 UpperLimit[6,9]
                B               12                              23
29 UpperLimit[7,10]
                B               40                              50
30 UpperLimit[8,10]
                B               46                              50
31 UpperLimit[9,10]
                B               13                              50
```

これをみると，輸送量が上限に達している枝は $(0,2),(1,5),(2,5),(2,6),(3,5),(3,6),(4,7),(4,8),(5,7)$ $(5,8),(6,8)$ です．ここで，これらの枝は表 *5.5* のように 2 つに分類することができます．ここで `eps`

表 5.5: 枝の分類

`Marginal` の値	該当する枝
`< eps`	(0,2), (5,7), (5,8), (6,8)
`1`	(1,5), (2,5), (2,6), (3,5), (3,6), (4,7), (4,8)

は「*'epsilon'*= 微小な量」のことを指しています．そして，`< eps` は `Marginal` の値が考慮に値しない程小さな値であることを表しています．

このことを最大流問題の視点で解釈すると，

> `Marginal` の値が `< eps` となっている枝は，輸送量が上限に達してはいるが，他にも輸送経路の候補がある（ので，最大輸送量を変化させても目的関数に影響を及ぼさない）

ということができます．実際，枝 $(0,2)$ の輸送可能量を 50 から 51 に変えて問題を解き直しても，

```
Status:     OPTIMAL
```

```
Objective:  Objective = 99 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
    12 UpperLimit[0,1]
                    B              27                         50
    13 UpperLimit[0,2]
                    NU             51                         51         < eps
    14 UpperLimit[0,3]
                    B              21                         50
    15 UpperLimit[1,4]
                    B              15                         18
    16 UpperLimit[1,5]
                    NU             12                         12             1
    17 UpperLimit[2,4]
                    B              19                         20
    18 UpperLimit[2,5]
                    NU             15                         15             1
    19 UpperLimit[2,6]
                    NU             17                         17             1
    20 UpperLimit[3,5]
                    NU             10                         10             1
    21 UpperLimit[3,6]
                    NU             11                         11             1
    22 UpperLimit[4,7]
                    NU             23                         23             1
    23 UpperLimit[4,8]
                    NU             11                         11             1
    24 UpperLimit[5,7]
                    NU             17                         17         < eps
    25 UpperLimit[5,8]
                    NU             19                         19         < eps
    26 UpperLimit[5,9]
                    B               1                         25
    27 UpperLimit[6,8]
                    NU             16                         16         < eps
    28 UpperLimit[6,9]
                    B              12                         23
    29 UpperLimit[7,10]
                    B              40                         50
    30 UpperLimit[8,10]
                    B              46                         50
    31 UpperLimit[9,10]
                    B              13                         50
```

となり，目的関数値は変わりません（ただし，最適解（流れ）は変わっています）．

一方，Marginal の値が 1 であるような枝の輸送可能量を変えると，目的関数値が変わります．例えば枝 $(1,5)$ の輸送可能量を 12 から 13 に変えて問題を解き直すと，

```
Status:     OPTIMAL
Objective:  Objective = 100 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
    12 UpperLimit[0,1]
```

```
                B              29                      50
13 UpperLimit[0,2]
                NU             50                      50          < eps
14 UpperLimit[0,3]
                B              21                      50
15 UpperLimit[1,4]
                B              16                      18
16 UpperLimit[1,5]
                NU             13                      13             1
17 UpperLimit[2,4]
                B              18                      20
18 UpperLimit[2,5]
                NU             15                      15             1
19 UpperLimit[2,6]
                NU             17                      17             1
20 UpperLimit[3,5]
                NU             10                      10             1
21 UpperLimit[3,6]
                NU             11                      11             1
22 UpperLimit[4,7]
                NU             23                      23             1
23 UpperLimit[4,8]
                NU             11                      11             1
24 UpperLimit[5,7]
                NU             17                      17          < eps
25 UpperLimit[5,8]
                NU             19                      19          < eps
26 UpperLimit[5,9]
                B               2                      25
27 UpperLimit[6,8]
                NU             16                      16          < eps
28 UpperLimit[6,9]
                B              12                      23
29 UpperLimit[7,10]
                B              40                      50
30 UpperLimit[8,10]
                B              46                      50
31 UpperLimit[9,10]
                B              14                      50
```

となり，目的関数値は 1 増えることが確認できます. □

第6章　よく知られた最適化問題

本章では，これまでに提案され，広く使われている最適化問題とその定式化を紹介します．ここで紹介する最適化問題はいずれも汎用性が高いため，手元にある問題を定式化する上でも使える定式化が多く含まれているのではないかと思います．

6.1　集合分割問題

6.1.1　問題の定義と定式化

(3.3.6) が現れるよく知られた問題として**集合分割問題**があります．例えば次のような問題を考えてみましょう．

問題 6.1.1 あなたは航空会社の乗務員のスケジュールを作成する担当者です．I が機長の乗務案の集合，J はフライトの集合で，パラメータ a_{ij} は

$$a_{ij} = \begin{cases} 1, & \text{乗務案 } i \in I \text{ ではフライト } j \in J \text{ を担当する} \\ 0, & \text{乗務案 } i \in I \text{ ではフライト } j \in J \text{ を担当しない} \end{cases}$$

と定められているものとします．このとき，全てのフライトを滞りなく飛ばしつつ，機長の数を最小にするにはどの乗務案を採用すればよいでしょうか？

乗務案 $i \in I$ を採用する（$=$ 乗務案 i を担当する機長を 1 人準備する）ときに 1，採用しないときに 0 となるような 0-1 変数を x_i とすると，この問題は次のように定式化することができます．

$$\left| \begin{array}{ll} \text{minimize} & \displaystyle\sum_{i \in I} x_i \\ \text{subject to} & \displaystyle\sum_{i \in I} a_{ij} x_i = 1 \ (j \in J) \\ & x_i \in \{0,1\} \ (i \in I) \end{array} \right. \tag{6.1.1}$$

問題 (6.1.1) の制約条件に (3.3.6) が現れています．またここでは乗務案 $i \in I$ はどれも等価なものとして（$=$ 目的関数の中で差が無く）扱っていますが，何らかの優先度がある場合（例えば遅延に対する強さ/弱さ）には，これをパラメータ c_i として，目的関数を $\sum_{i \in I} c_i x_i$ などとすることも考えられます．□

6.1.2　モデリング言語による問題記述

集合分割問題 (6.1.1) は次のように記述することができます．

```
set I;
set J;

var x{I} binary;
param a{I, J};

minimize Objective:
 sum {i in I} x[i];

subject to Const_Partition {j in J}:
sum {i in I} (a[i, j] * x[i]) = 1;

end;
```

6.1.3　データ例と最適解

前節のモデル記述と C.1 節のデータを用いることにより求めた最適解を整理した結果を表 6.1 に示します.

最適解において $x_i = 1$ となるような i は $8, 14, 17, 19, 22, 32, 44$ の 7 個でした．すなわち，フライトの集合 J を分割して担当するのに必要な機長の数（= 最適解における目的関数の値）は 7 人であることがわかります.

表 6.1: 問題 6.1.1 の最適解と集合分割の様子（データ：C.1 節）

i	a_{ij}																			
8	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
14	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
17	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
22	0	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
32	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
44	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
$\sum_{i \in I} a_{ij} x_i$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

6.2　集合パッキング問題

6.2.1　問題の定義と定式化

(3.3.7) が現れるよく知られた問題として**集合パッキング問題**があります．例えば次のような問題を考えてみましょう.

問題 6.2.1 I を手元にある段ボール箱の集合とし，J を商品の集合とします．このとき，パラメータ a_{ij} が

$$a_{ij} = \begin{cases} 1, & \text{段ボール箱 } i \in I \text{ に商品 } j \in J \text{ を（} a_{ij} = 1 \text{ となるような他の } j \text{ と同時に）} \\ & \text{詰めることが可能} \\ 0, & \text{段ボール箱 } i \in I \text{ に商品 } j \in J \text{ を詰めることは不可能} \end{cases}$$

と定められているものとします．今，全ての商品 $j \in J$ を段ボール箱に詰め込みたいのですが，手元にある段ボール箱であれば安く済むため，できるだけ多くの商品を手元の段ボール箱 $i \in I$ に詰め込みたいと思います．ただし，段ボール箱に隙間がありすぎてもいけないので，段ボール箱 i を使うときには $a_{ij} = 1$ となるような商品 j は全て同時に詰め込まなくてはいけません．さて，どのように詰めればよいでしょうか？

段ボール箱 $i \in I$ を使う時に 1，使わない（使えない）ときに 0 となるような 0-1 変数を x_i とすると，この問題は次のように定式化することができます．

$$\left| \begin{array}{ll} \text{maximize} & \displaystyle\sum_{i \in I, j \in J} a_{ij} x_i \\ \text{subject to} & \displaystyle\sum_{i \in I} a_{ij} x_i \leq 1 \ (j \in J) \\ & x_i \in \{0, 1\} \ (i \in I) \end{array} \right. \tag{6.2.1}$$

問題 (6.2.1) の制約条件に (3.3.7) が現れています．

問題 (6.2.1) の目的関数は，$\sum_{j \in J} (\sum_{i \in I} a_{ij} x_i)$ と解釈するとよいでしょう．カッコの中は商品 j がいずれかの段ボール箱に詰められるならば 1，そうでないなら 0 になります（そしてこれは制約条件の左辺に他なりません）．さらに j について和を取ることで，段ボール箱に詰められる商品数を表すことになり，これを最大化しています． □

6.2.2 モデリング言語による問題記述

集合パッキング問題 (6.2.1) は次のように記述することができます．

```
set I;
set J;

var x{I} binary;
param a{I, J};

maximize Objective:
    sum {i in I, j in J} (a[i, j] * x[i]);

subject to Const_packing {j in J}:
    sum {i in I} (a[i, j] * x[i]) <= 1;

end;
```

6.2.3　データ例と最適解

前節のモデル記述と C.2 節のデータを用いることにより求めた最適解を整理した結果を表 6.2 に示します.

最適解において $x_i = 1$ となるような i は $4, 7, 21, 25, 35, 37, 41$ で，そのときの目的関数値は 19 でした．すなわち，20 個の商品 $j \in J$ のうち 19 個を手元の段ボールに詰め込むことができる（$= 1$ 個だけ手元の段ボールに詰め込むことができない）ということになります.

表 6.2: 問題 6.2.1 の最適解と集合パッキングの様子（データ：C.2 節）

i	a_{ij}																			
4	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
21	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
35	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
37	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1
41	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
$\sum_{i \in I} a_{ij} x_i$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1

6.3　集合被覆問題

6.3.1　問題の定義と定式化

(3.3.8) が現れるよく知られた問題として**集合被覆問題**があります．例えば次のような問題を考えてみましょう.

問題 6.3.1 I は消防署の建設候補地の集合であり，J は建設する消防署でカバーすべき地域の集合であるものとします．このとき，パラメータ a_{ij} が

$$a_{ij} = \begin{cases} 1, & \text{消防署の建設候補地 } i \in I \text{ から地域 } j \in J \text{ に所定の時間で到達可能} \\ 0, & \text{消防署の建設候補地 } i \in I \text{ から地域 } j \in J \text{ に所定の時間で到達不可能} \end{cases}$$

と定められているものとします．また建設候補地 $i \in I$ に消防署を建設する際には費用が c_i だけ必要です．このとき，費用をできるだけ抑えつつ全ての地域を所定の時間でカバーするためには，消防署をどこに建設するのがよいでしょうか？なお，消防署がカバーする地域に重なりがあってもよいものとします.

建設候補地 $i \in I$ に消防署を建設するときに 1，建設しないときに 0 となるような 0-1 変数を x_i とすると，この問題は次のように定式化することができます.

$$\left| \begin{array}{ll} \text{minimize} & \displaystyle\sum_{i \in I} c_i x_i \\ \text{subject to} & \displaystyle\sum_{i \in I} a_{ij} x_i \geq 1 \ (j \in J) \\ & x_i \in \{0, 1\} \ (i \in I) \end{array} \right. \tag{6.3.1}$$

問題 (6.3.1) の制約条件に (3.3.8) が現れています. □

6.3.2 モデリング言語による問題記述

集合被覆問題 (6.3.1) は次のように記述することができます.

```
set I;
set J;

var x{I} binary;
param a{I, J};
param c{I};

minimize Objective:
    sum {i in I} (c[i] * x[i]);

subject to Const_cover {j in J}:
    sum {i in I} (a[i, j] * x[i]) >= 1;

end;
```

6.3.3 データ例と最適解

前節のモデル記述と C.3 節のデータを用いることにより求めた最適解を整理した結果を表 6.3 に示します.

最適解において $x_i = 1$ となるような i は $1, 3, 5, 7, 14, 25, 37$ であることがわかります．これらの i に対応する a_{ij} の値を並べてみると表 6.3 のようになります．すなわち，全ての地域が建設候補地

表 6.3: 問題 6.3.1 の最適解と集合被覆の様子（データ：C.3 節）

i	a_{ij}																			
1	0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0
3	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0
5	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
14	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
37	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1
$\sum_{i \in I} a_{ij} x_i$	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1	1

$1, 3, 5, 7, 14, 25, 37$ のいずれかから所定の時間内で到達可能である（1 地区については 2 つの建設候補地から到達可能）ことがわかります.

また，建設候補地 $i \in I$ に消防署を実際に建設する場合の費用 c_i も C.3 にある通り（単位：億円）ですが，最適解における目的関数値は 13.1（億円）でした．これが所定の時間内で全地区をカバーするような建設計画のうち，最安のものになります．

6.4　ナップサック問題

6.4.1　問題の定義と定式化

制約式 (3.3.9) は，次に示す**ナップサック問題**によく現れます．

問題 6.4.1 あなたはとあるお店に忍び込んだ泥棒 (!) だとします．お店にある商品 $i \in I$ の大きさが a_i であるものとし，その価値は c_i であるものとします．あなたは大きさが b のナップサックに荷物を詰めて逃げだそうとしています．盗み出す商品の価値の合計ができるだけ大きくなるようにするには，次の問題を解けばいいでしょう．

$$\left|\begin{array}{ll} \text{maximize} & \displaystyle\sum_{i\in I} c_i x_i \\ \text{subject to} & \displaystyle\sum_{i\in I} a_i x_i \le b \\ & x_i \in \{0,1\}\ (i \in I) \end{array}\right. \tag{6.4.1}$$

ここで，変数 x_i $(i \in I)$ は，値が 1 であれば商品 i をナップサックに詰める，0 ならば詰めないことを表す 0-1 変数です．

問題 (6.4.1) の制約条件が (3.3.9) であることに注意してください．　□

6.4.2　モデリング言語による問題記述

ナップサック問題 (6.4.1) は次のように記述することができます．

```
set I;

var x{I} binary;

param b;
param c{I};
param a{I};

maximize Objective:
    sum {i in I} (c[i] * x[i]);

subject to Const_Knapsack:
    sum {i in I} (a[i] * x[i]) <= b;

end;
```

6.4.3 データ例と最適解

前節のモデル記述と C.4 節のデータを用いることにより求めた最適解を整理した結果を表 6.4 に示します.

ナップサックに詰めるべき商品の大きさの合計は，ナップサックのサイズとちょうど同じ 50 になっています．また商品価値の合計は 129 となり，これが泥棒として盗み出す最適な (!) 商品の組合せになっています.

表 6.4: 問題 6.4.1 の最適解（データ：C.4 節）

商品番号	2	5	9	13	17	18	20	21	23	2r	29	37	41	43	45	46
価値	10	7	9	7	7	7	10	10	9	6	7	7	9	8	8	8
大きさ	3	2	4	5	4	2	5	4	2	2	3	4	2	4	4	4

6.5 ビンパッキング問題

6.5.1 問題の定義と定式化

制約式 (3.3.13) は，次に示すビンパッキング問題によく現れます.

問題 6.5.1 あなたは商品 $i \in I$ の全てをあるお客さんの元に送らなくてはなりません．商品 $i \in I$ の大きさは a_i であるものとし，これを大きさの b の段ボール箱を用いて送ります（a_i $(i \in I)$ は b を超えないものとします）．そして，大きさの合計が b を超えない範囲で，一つの段ボール箱に複数の商品を詰め込むことが可能です．このとき，必要となる段ボール箱の個数が最小になるようにするための商品の組合せは，次の問題を解くことで得られます.

$$
\left|\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{j\in J} y_j \\
\text{subject to} & \displaystyle\sum_{i\in I} a_i x_{ij} \le b y_j \ (j \in J) \\
& \displaystyle\sum_{j\in J} x_{ij} = 1 \ (i \in I) \\
& x_{ij} \in \{0,1\} \ (i \in I, j \in J) \\
& y_j \in \{0,1\} \ (j \in J)
\end{array}\right. \tag{6.5.1}
$$

ここで，変数 y_j $(j \in J)$ は，値が 1 であれば段ボール箱 j を用いる，0 ならば用いないことを表す 0-1 変数です．また x_{ij} は値が 1 であれば商品 i を段ボール箱 j に詰める，0 ならば詰めないことを表します.

問題 (6.5.1) の最初の制約条件が (3.3.13) であることに注意してください． □

6.5.2 モデリング言語による問題記述

ビンパッキング問題 (6.5.1) は次のように記述することができます.

```
set I;
set J;

var x{I, J} binary;
var y{J} binary;

param b;
param a{I};

minimize Objective:
    sum {j in J} y[j];

subject to Const_BinPacking {j in J}:
    sum {i in I} (a[i] * x[i, j]) <= b * y[j];

subject to Const_AllParcelPacked {i in I}:
    sum {j in J} x[i, j] == 1;

end;
```

6.5.3　データ例と最適解

前節のモデル記述と C.5 節のデータを用いることにより求めた最適解を整理した結果を表 6.5 に示します.

必要となる段ボール箱の最小個数は 21 個であり，商品の組合せは表 6.5 のようにすればよいことがわかりました．表 6.5 をみると，各段ボール箱の「隙間」ができるだけ小さくなるような商品の組合せになっていることがわかります.

6.6　最大流問題

6.6.1　問題の定義と定式化

制約式 (3.4.1), (3.4.2), (3.4.3), (3.4.4) は，次に示す**最大流問題**によく現れます.

問題 6.6.1 図 6.1 はある運送会社の輸送網を表しています．出発地 s から到着地 t に向けて矢印に沿って輸送することができ，矢印に付されている数値はその経路の輸送可能量を表しています．ここで，図 6.1 をグラフ $G = (V, E)$ とし，枝 $(i, j) \in E$ の輸送可能量を c_{ij} とします．このとき，枝 $(i, j) \in E$ 上の輸送料を x_{ij} とすると，s から t までの最大輸送量は，次の最適化問題を解くことで

表 6.5: 問題 6.5.1 の最適解（データ：C.5 節）

箱番号	商品番号（カッコ内は商品の大きさ）	大きさの合計
1	1 (11), 35 (12), 46 (18), 50 (9)	50
2	37 (21), 40 (29)	50
3	10 (10), 44 (29), 47 (11)	50
4	6 (31), 16 (6), 20 (5), 38 (7)	49
5	7 (8), 29 (33), 33 (9)	50
6	26 (16), 30 (27), 41 (7)	50
7	23 (33), 49 (17)	50
8	24 (34), 36 (15)	49
9	11 (31), 28 (18)	49
10	25 (17), 27 (31)	48
11	22 (25), 32 (23)	48
12	4 (18), 31 (29)	47
13	12 (16), 45 (31)	47
14	2 (22), 8 (28)	50
15	3 (15), 48 (29)	44
16	14 (30), 17 (16)	46
17	9 (32), 43 (18)	50
18	13 (24), 34 (23)	47
19	21 (17), 39 (30)	47
20	15 (10), 42 (29)	39
21	5 (33), 18 (6), 19 (7)	46

得られます.

$$
\left|\begin{array}{ll}
\text{maximize} & f \\
\text{subject to} & f = \displaystyle\sum_{(s,j)\in E} x_{sj} \\
& \displaystyle\sum_{(i,v)\in E} x_{iv} = \sum_{(v,j)\in E} x_{vj} \quad (v \in V,\ v \neq s,t) \\
& 0 \leq x_{ij} \leq c_{ij} \quad ((i,j) \in E)
\end{array}\right. \tag{6.6.1}
$$

ここで，問題 (6.6.1) の制約式が (3.4.1), (3.4.2), (3.4.3), (3.4.4) であることに注意してください．□

6.6.2 モデリング言語による問題記述

最大流問題 (6.6.1) は次のように記述することができます.

```
set V;
set E dimen 2;
```

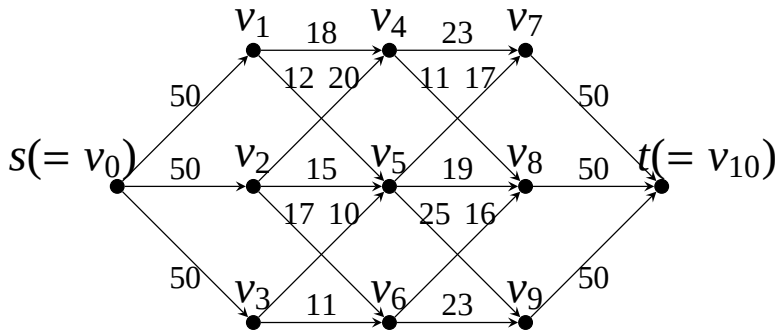

図 6.1: 問題 6.5.1 （データ：C.6 節）

```
param s;
param t;
param c{E};

var x{E} >= 0;
var f;

maximize Objective: f;

subject to FlowAmount:
    f == sum {(i, j) in E: i == s} x[i, j];

subject to FlowConservation {v in V: v != s && v != t}:
    sum {(i, v) in E} x[i, v] == sum {(v, j) in E} x[v, j];

subject to UpperLimit {(i, j) in E}:
    x[i, j] <= c[i, j];

end;
```

6.6.3　データ例と最適解

前節のモデル記述と C.6 節のデータを用いることにより求めた最適解を整理した結果を図 6.2 に示します．最適解における最大輸送量は 99 となることがわかります．

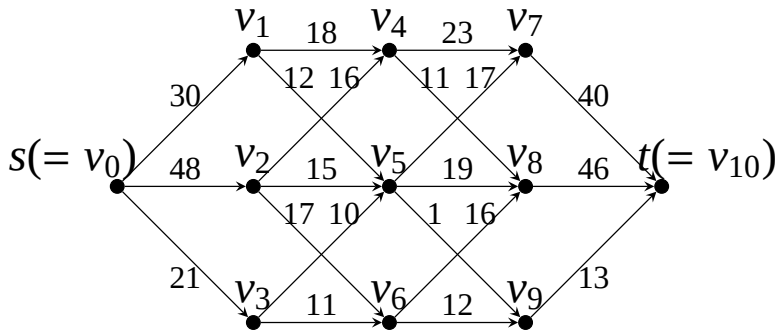

図 6.2: 問題 6.5.1 の最適解（データ：C.6 節）

6.7 最短路問題

カーナビでは，地図上の 2 点間の最短経路を求める必要があります．ここではそのような問題を考えてみましょう [1].

6.7.1 問題の定義と定式化

問題 6.7.1 有向グラフ $G = (V, A)$ があり，有向枝 $(i, j) \in A$ には長さ（重み）w_{ij} が与えられているものとします．このとき，始点 $s \in V$ と終点 $t \in V$ に対して，s から t までの最短路を求めましょう．

この問題は次のように定式化することができます．

$$
\left|\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{(i,j)\in A} c_{ij}x_{ij} \\
\text{subject to} & \displaystyle\sum_{(i,v)\in A} x_{iv} = \sum_{(v,j)\in A} x_{vj} \quad (v \in V, v \neq s, t) \\
& \displaystyle\sum_{(s,j)\in A} x_{sj} = 1 \\
& x_{ij} \in \{0,1\} \quad ((i,j) \in A)
\end{array}\right. \tag{6.7.1}
$$

ここで，変数 x_{ij} $((i, j) \in A)$ は，有向枝 (i, j) を最短路に含めるときに 1，含めないときに 0 となる 0-1 変数です．

この定式化に現れる制約条件は，*3.4.1* 節で取り上げたグラフ上のフローを表現する定式化 *(3.4.1), (3.4.2), (3.4.3)* において，フローの量を 1 としたものと解釈することができます．すなわち，最短路に含まれる有向枝にはフロー 1 が流れ，含まれない有向枝にはフローが流れない，ということです．□

[1] なお，最短路問題については 7.6.1 節にも記述がありますので，そちらも参考にしてください．

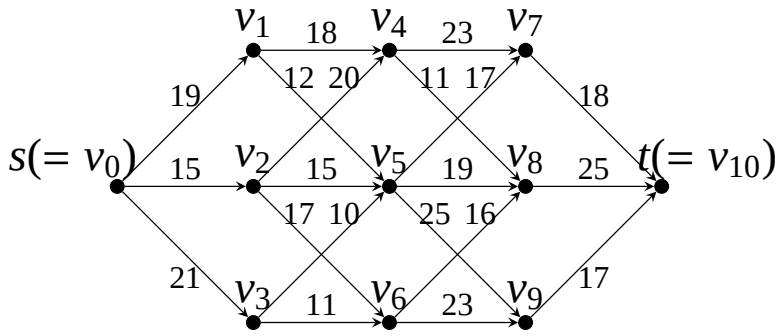

図 6.3: 問題 6.7.1 （データ：C.7 節）

6.7.2　モデリング言語による問題記述

最短路問題 (6.7.1) は次のように記述することができます.

```
set V;
set A dimen 2;

param c{A};
param s;
param t;

var x{A} binary;

minimize Objective:
sum{(i, j) in A} (c[i, j] * x[i, j]);

subject to FlowConservation {v in V: v != s && v != t}:
sum{(i, v) in A} x[i, v] == sum{(v, j) in A} x[v, j];

subject to FlowStart:
sum{(s, j) in A} x[s, j] == 1;

end;
```

6.7.3 データ例と最適解

前節のモデル記述と C.7 節のデータを用いることにより求めた最適解を整理した結果を図 6.4 に示します．最適解（最短経路：$s(=v_0), v_2, v_5, v_7, t(=v_{10})$）における経路長は 65 となります．

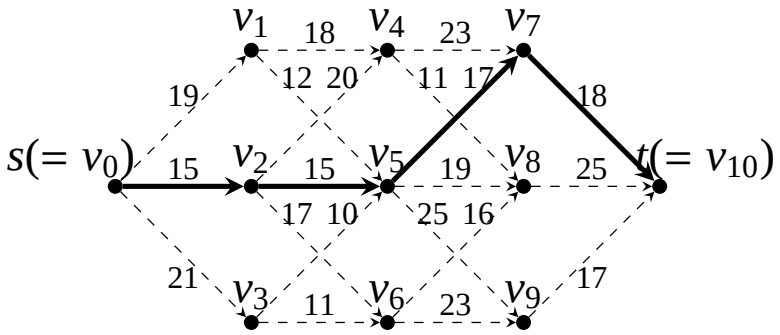

図 6.4: 問題 6.7.1 の最適解（データ：C.7 節）

6.8 Hitchcock 型輸送問題

6.8.1 問題の定義と定式化

(3.4.5), (3.4.6) が現れるよく知られた問題として，**Hitchcock 型輸送問題**が挙げられます．次の問題を考えてみましょう．

問題 6.8.1 ある単一の商品を供給側の節点 $i \in I$ から需要側の節点 $j \in J$ まで運ぶものとします．

供給側には供給可能量 s_i $(i \in I)$ が定められており，これを超えて供給することはできません．なお供給先は一つである必要はなく，供給可能量の範囲内で複数の需要側に供給することができます．

また需要側には需要量 t_j $(j \in J)$ が定められており，供給側のいずれかの節点 $i \in I$ からこれだけの量を運ぶ必要があります．なお，合計が t_j になるようにして複数の供給側から供給を受けることができます．

このとき，供給側 $i \in I$ から需要側 $j \in J$ へ商品 1 単位を運ぶ際に必要なコストを c_{ij} とします．このとき，運送に必要なコストの総和を最小にしたいと思います．

この問題は次のように定式化することができます.

$$\left|\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i\in I, j\in J} c_{ij}x_{ij} \\ \text{subject to} & \displaystyle\sum_{j\in J} x_{ij} \le s_i \quad (i \in I) \\ & \displaystyle\sum_{i\in I} x_{ij} = t_j \quad (j \in J) \\ & x_{ij} \ge 0 \quad (i \in I, j \in J) \end{array}\right. \tag{6.8.1}$$

問題 (6.8.1) の制約条件に (3.4.5), (3.4.6) が現れています. □

6.8.2　モデリング言語による問題記述

Hitchcock 型輸送問題 (6.8.1) は次のように記述することができます.

```
set I;
set J;

var x{I, J} >= 0;

param a{I};
param b{J};
param c{I, J};

minimize Objective:
        sum {i in I, j in J} (c[i, j] * x[i, j]);

subject to C1 {i in I}:
        sum {j in J} x[i, j] <= s[i];

subject to C2 {j in J}:
        sum {i in I} x[i, j] == t[j];

end;
```

6.8.3　データ例と最適解

前節のモデル記述と C.8 節のデータを用いることにより求めた最適解を表 6.6 に示します.

表 6.6 より，需要側には必要としている需要量が過不足なく輸送されることがわかります．また，供給側では，供給可能量を上回らない範囲で各需要先に輸送されることもわかります.

表 6.6: 供給側から需要側への最適な輸送量

	需要側 1	2	3	4	供給可能量
供給側 A	0	0	15	45	60
B	0	15	20	0	45
C	35	0	0	0	50
需要量	35	15	35	45	–

6.9 割当問題

6.9.1 問題の定義と定式化

(3.4.7), (3.4.8) が現れるよく知られた問題として，**割当問題**があります．次の問題を考えてみましょう．

問題 6.9.1 I は n 人の集合だとし，J は n 個の仕事の集合だとします．このとき，n 人に仕事を 1 つずつ割当てることにし，その割当て方を考えます．

人 $i \in I$ に仕事 $j \in J$ を割当てたとき，作業時間は t_{ij} であるものとして，各人の作業時間の総和を最小にしたいと思います．

この問題は次のように定式化することができます．

$$
\left|\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{i\in I, j\in J} t_{ij}x_{ij} \\
\text{subject to} & \displaystyle\sum_{j\in J} x_{ij} = 1 \quad (i \in I) \\
& \displaystyle\sum_{i\in I} x_{ij} = 1 \quad (j \in J) \\
& x_{ij} \in \{0,1\} \quad (i \in I, j \in J)
\end{array}\right. \tag{6.9.1}
$$

問題 (6.9.1) の制約条件に (3.4.7), (3.4.8) が現れています． □

6.9.2 モデリング言語による問題記述

割当問題 (6.9.1) は次のように記述することができます．

```
set I;
set J;

param t{I, J};
var x{I, J} binary;

minimize Objective:
    sum {i in I, j in J} (t[i, j] * x[i, j]);
```

```
subject to AssignmentWoker {i in I}:
    sum {j in J} x[i, j] == 1;

subject to AssignmentJob {j in J}:
    sum {i in I} x[i, j] == 1;

end;
```

6.9.3　データ例と最適解

前節のモデル記述と C.9 節のデータを用いることにより求めた最適解を表 6.7 に示します．表 6.7 より，各人の作業時間が短いような作業をうまく組み合わせていることがわかります．

表 6.7: 問題 6.9.1 の最適解（データ：C.9 節）

人 i	割り当てる仕事 j	作業時間	人 i	割り当てる仕事 j	作業時間
1	20	10	14	16	11
2	15	17	15	22	11
3	25	11	16	14	11
4	23	10	17	19	13
5	3	15	18	6	10
6	7	14	19	21	19
7	10	12	20	13	12
8	24	12	21	1	10
9	11	13	22	5	12
10	9	12	23	8	12
11	12	14	24	4	10
12	2	12	25	18	13
13	17	16		合計	312

なおここでは $|I| = |J| = n$ の場合を考えていますが，$|I| \neq |J|$ となるような場合も考えられます．例えば $|I| < |J|$（仕事の方が多い）という状況で，全ての仕事は必ず誰かに割り当てる必要があり，一人が担当できる仕事数は 2 個まで，という状況なら

$$\sum_{j \in J} x_{ij} \leq 2 \ (i \in I), \quad \sum_{i \in I} x_{ij} = 1 \quad (j \in J)$$

という制約になりますし，$|I| > |J|$（人の方が多い）という状況で全ての仕事を誰かに割り当て，一人が担当できる仕事数に上限を設けない，という状況ならば

$$\sum_{i \in I} x_{ij} = 1 \quad (j \in J)$$

という制約だけを設ければよいことになります（$i \in I$ に関する制約は不要）．

6.10 巡回セールスマン問題

6.10.1 問題の定義と定式化

巡回セールスマン問題は，3.1 節で取り上げました．ここでは，巡回セールスマン問題の定式化の一つについて取り上げます．なお，巡回セールスマン問題については 7.3 節でも取り上げます．

ここでは，セールスマンが巡回する都市の集合を $I = J = \{1, 2, \ldots, n\}$ とし，都市 $i \in I$ から都市 $j \in J$ までの距離を d_{ij} とします [2]．

ここでは次のような 0-1 変数 x_{ij} を準備します．

$$x_{ij} = \begin{cases} 1, & \text{都市 } i \text{ から都市 } j \text{ へ移動する} \\ 0, & \text{都市 } i \text{ から都市 } j \text{ へ移動しない} \end{cases}$$

また，次の変数 u_i $(i \in \{2, 3, \ldots, n\})$ を設けます．

$u_i =$ 都市 i の訪問順番（ただし都市 1 を 0 番目とする）

なお，この変数は整数変数ではなく連続変数として定義することに注意してください．また，この変数が訪問順番を表す理由は定式化を紹介した後で詳しく説明します．

これを用いると，巡回セールスマン問題は次のように定式化できます．

$$\left|\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i\in I, j\in J} d_{ij}x_{ij} \\ \text{subject to} & \displaystyle\sum_{j\in J} x_{ij} = 1 \quad (i \in I) \\ & \displaystyle\sum_{i\in I} x_{ij} = 1 \quad (j \in J) \\ & x_{ii} = 0 \quad (i \in I) \\ & u_i + 1 - (n-1)(1 - x_{ij}) \le u_j \quad (i \in I, j \in J, i \ge 2, j \ge 2, i \ne j) \\ & 1 \le u_i \le n-1 \quad (i \in I, i \ge 2) \\ & x_{ij} \in \{0, 1\} \quad (i \in I, j \in J) \end{array}\right. \tag{6.10.1}$$

まず，(6.10.1) の制約条件のうち，1, 2 番目は (3.4.5), (3.4.6) と同じ形であることに注意してください．3 番目の制約は，ある都市 $i \in I$ を出発して同一の都市に戻ることはない，という制約です [3]．

4, 5 番目の制約は読み解くのがやや難しいかもしれません（この制約式は Miller-Tucker-Zemlin (MTZ) 制約と呼ばれる制約式です）．以下，これらの制約について説明します．

まず $x_{ij} = 0$ の場合，すなわち「訪問箇所 i の次に j を訪問しない」場合を考えてみます．このとき，最初の制約式は

$$u_i + 1 - (n-1) \le u_j \tag{6.10.2}$$

[2] 通常は $d_{ij} = d_{ji}$ であるものと考えます．このような巡回セールスマン問題は**対称巡回セールスマン問題**とよばれます．これに対し，$d_{ij} \ne d_{ji}$ であるような場合は，**非対称巡回セールスマン問題**といいます．

[3] 自然に考えると $d_{ii} = 0$ $(i \in I)$ であるので，この制約がない場合，$x_{ii} = 1$ $(i \in I), x_{ij} = 0$ $(i \ne j)$ が最適解になってしまいます（目的関数値 $= 0$）．あるいはこの制約の代わりに d_{ii} $(i \in I)$ の値を（擬似的に）十分大きくしておけば，$x_{ii} = 1$ となることがなくなるため，正しい巡回路を得る事ができます．

となります．一方，$1 \leq u_i \leq n-1$, $1 \leq u_j \leq n-1$ です（j も訪問箇所を表す要素なので，u_j についても u_i と同じ式が成立します）．制約 (6.10.2) を最も「破綻」させそうなケースは，u_i ができるだけ大きく，u_j ができるだけ小さい場合，すなわち $u_i = n-1$, $u_j = 1$ の場合です．しかしその場合でも，(左辺)$= 1 =$(右辺) であり，制約式は満たされます．すなわちこの制約式は常に満たされるということになり，$x_{ij} = 0$ の場合は事実上意味を持たない制約になることがわかります．

次に $x_{ij} = 1$ の場合，すなわち「訪問箇所 i の次に j を訪問する」場合を考えます．この場合，ここで考えている制約式は

$$u_i + 1 \leq u_j \tag{6.10.3}$$

となります．つまり，$i \to j$ の順に訪問するならば，u_j の値は u_i よりも 1 以上 大きくなる，ということです．このとき，k 番目に通過する訪問箇所を i_k とし，もう一つの制約に注意すると，

$$1 \leq u_{i_1} \leq_1 u_{i_2} \leq_1 \cdots \leq_1 u_{i_{n-1}} \leq n-1$$

という大小関係が成立します（$i \geq 2$ とあるので，スタート地点 $(i = 1)$ は含まないことに注意）．ここで $a \leq_1 b$ は「b は a より 1 以上大きい」ということを表すものとします．そして，最左項 (1) と最右項 $(n-1)$ に挟まれる項の数に注意すれば，結果として

$$u_{i_1} = 1,\ u_{i_2} = 2,\ \ldots,\ u_{i_{n-1}} = n-1$$

となることがわかります．すなわち，u_i は訪問箇所 i の訪問順序を表すものに他なりません．

このようにして，各都市の訪問順序を表す変数 u_i と訪問経路を表す変数 x_{ij} の値が決定することになります．

ここで，C.10 節に挙げてあるデータは，図 3.1 に対応しています．このデータで問題を解くと，最適な巡回路として図 3.3 を得る，ということになります．

6.10.2　モデリング言語による問題記述

巡回セールスマン問題 (6.10.1) は次のように記述することができます．

```
set I;
set J;

param d{I, J};
param n;

var x{I, J} binary;
var u{I};

minimize Objective:
 sum {i in I, j in J} (d[i, j] * x[i, j]);

subject to C1 {i in I}:
sum {j in J} x[i, j] == 1;
```

```
subject to C2 {j in J}:
sum {i in I} x[i, j] == 1;

subject to C3 {i in I, j in J: i == j}:
x[i, j] == 0;

subject to C4 {i in I, j in J:
   i >= 2 && j >= 2 && i != j}:
u[i] + 1 - (n - 1) * (1 - x[i, j]) <= u[j];

subject to C5_1 {i in I: i >= 2}:
u[i] >= 1;

subject to C5_2 {i in I: i >= 2}:
u[i] <= n - 1;

end;
```

6.10.3 データ例と最適解

C.10 節に示したデータは，実は図 3.1 に対応するものです．6.10.2 節のモデルと C.10 節のデータを用いることにより，図 3.3 のような最適解を得る事ができます．

第7章　難しい問題をどう解くか

本章で学ぶ内容

- 厳密解法と近似解法
- 大規模問題に対するアプローチの方法
- 制約条件を全て書き下すことができない場合の対処
- （不必要な制約を最初から考慮しない？）
- できるだけ満たしたい制約の扱い方（制約充足問題）
- 定式化の工夫

ここまでは，最適化問題をモデル化・定式化し，それをソルバで解く，というアプローチについて説明してきました．しかし，この方法では問題をうまく解けない（ように思える）場合があります．そこで本章では，このような難しい問題に対する様々なアプローチを紹介したいと思います．

7.1　近似解法の利用

本書で説明してきたソルバ（やそこで用いられている解法）は「**厳密解法**」を念頭に置いたものでした．厳密解法とは，最適解があればそれを見つけることができる解法のことです[1]．これに対し「**近似解法**」とは，必ずしも最適解を求めることができない解法のことをいいます[2].「必ずしも最適解を求めることができない」などといわれると少し驚きますが，これは「得られる解が最適解である保証ができない」という程度の意味です．近似解法は，解の精度保証の意味では厳密解法と比べ不利ですが，厳密解法と比べ短い時間でできるだけよい解を得ることを目指す解法です．近似解法の代表的な例としては，**アニーリング法**，**遺伝アルゴリズム**，**タブー探索法**などが挙げられます [14, 23].

厳密解法では，文字通り「厳密な」最適解を得る事を目指すため，結果として解が得られない場合があります．その場合でも**暫定解**と呼ばれる，計算途中での最良な実行可能解を得る事はできますが，厳密解法は暫定解を求めるためだけに設計されているわけではないので，近似解法の方が優れた暫定解を短い時間で求められる可能性があります．

最近では，近似解法を実装した商用・非商用のソルバが数多くあります[3]．上で述べたように，これらのソルバでは精度保証はできませんが，よいソルバを用いれば厳密解にかなり近い値を得る事

[1]ただし，計算に要する時間や必要となるメモリ量などは問わないことにします．

[2]ここでの定義とは異なり，最適解からどの程度離れているかを評価できるような解法を「近似解法」ということもあります．この場合，本節で説明している「近似解法」は「**ヒューリスティック解法**（**発見的解法**）」と呼びます．

[3]ただし，近似解法を実装したソルバの場合，汎用的な問題クラスに適用できるものと，ある特定の問題（例えば巡回セールスマン問題）にのみ適用できるソルバがあります．

ができることはよくありますし，場合によっては厳密解そのものを得られる場合もあります．問題によっては近似解法を選択することで，実用上問題ない解を得られる可能性があります．

7.2　大規模問題に対する 2 つのアプローチ

問題の規模，すなわち変数の数や制約の数が大きくなると，当然問題は解きにくくなるでしょう [4]．

ここでは，そのような時にどうすればよいか，ということについて考えてみます．残念ながら，以下で説明することは全ての場合に適用可能ということではありませんが，アプローチの方法として是非頭に入れておいて欲しいことです．

7.2.1　大規模問題恐るるに足らず？

一概に「大規模な問題」と言っても，どの程度になったら大規模な問題なのか，という明確な線引きはありません．みなさんは,「大規模な問題」と聞いたときに，どの程度の変数の数・制約の数を想像しますか？

ここで，最適化問題の**疎性**について触れておきます．変数が n 個，制約条件が m 個ある最適化問題があるものとします．ここで，制約条件は全て 1 次関数であるものとしましょう．このとき，全ての変数が全ての制約条件に現れるものとすると，扱わなければならない変数の係数は nm 個になります．例えば $n = m = 1$ 万 $(= 10^4)$ のような問題であるとすれば，$nm = 1$ 億 $(= 10^8)$ となってしまいます．しかし，実際の問題ではこのようになることはまずありません．変数の数・制約の数がそれぞれ 1 万だったとしても，各制約に現れる変数の数は数個から 10 個程度であることがほとんどでしょう．すなわち，扱わなければならない係数はせいぜい 10 万個 $(= 10^5)$ 程度です．一般に，考えられる組合せのうちほんの一部だけを扱えばいい，という性質を「疎性」といいます．大規模な問題には，ほとんどの場合この疎性が見られるため，直感的に考えるよりは大規模な問題を扱うことができる場合があります．ですので,「この問題，大規模だな」と思ったとしても，

> まずはソルバに解かせてみる

というのは一つの戦略になり得ます（解けなかったら別の方法を考えるしかありませんが）．

大規模な混合整数計画問題について

ここでは，大規模な問題の中でも，混合整数計画問題に限定した話題を紹介します．

3.3 節で紹介した，混合整数計画問題のベンチマーク MIPLIB [35] では，問題の難しさに応じて ‘Easy’, ‘Hard’, ‘Open’ という分類がされています．これらの分類は表 7.1 のような基準で行われています．このうち ‘Easy’ の中には，問題名 `buildingenergy`（154978 変数，277594 制約）や `map06`, `map10`, `map14`, `map18`, `map20`（いずれも 164547 変数，328818 制約）のような，非常に大規模な問題も含まれています（図 7.2 も参照して下さい）．つまり，現在（2014 年）のソルバや計算機の性能で，この程度の問題を扱うことは現実的になっているのです．

しかし一方で,（上のような問題からすると）比較的小規模であっても ‘Open’ に分類されている問題もあります（2014 年 9 月現在．例えば `sts405`（405 変数，27270 制約）, `liu`（1156 変数，2178

[4] 問題の規模が大きくなっても例外的に解くことができる問題は存在します．しかし，一般に，これを事前に知ることは難しいことです．

表 7.1: MIPLIB における問題の分類

Easy	商用ソルバで 1 時間以内に解くことができる問題
Hard	解くことはできるが Easy には該当しない問題
Open	最適解が知られていない問題

制約）, pigeon-19（1444 変数，3307 制約） など．図 7.2 も参照）．すなわち，混合整数計画問題の場合，変数や制約の数で問題の難易度を推測することは非常に難しいと言えます．

それでもなお言いたいのは，数十万変数でも解ける問題があるわけですから,（上で述べたように）「まずはソルバに解かせてみる」という作戦は頭に入れておいてよいのではないかと思います [5]．

Note: ソルバと計算機の性能向上 混合整数計画問題のソルバとして有名な ILOG CPLEX , Gurobi はいずれも R. Bixby 教授が開発したソルバです．Bixby 教授によると，1991 年から 2012 年のおよそ 20 年の間に，混合整数計画問題を解くためのアルゴリズムは 475,000 倍 (!) のスピードアップを果たしたとのことです．一方，この期間は PC の性能向上が著しい期間でもあり，PC の計算速度は 2,000 倍速くなっています．従って，問題を解く速度は $475{,}000 \times 2{,}000 \simeq 10^9$ 程度 (!!) 速くなっています．

7.2.2 大きなモデルを作るのがよいのか？

前節では「恐るるに足らず」と書きましたが，ここでは逆のこと (!) を書きます．

（やはり,）一般的には問題の規模が大きくなるとソルバで解くのが難しくなってきます．そのような場合には,「モデルを分割できないか」ということを考えてみる必要があるでしょう．以下，例を通じてこのことを考えてみたいと思います．

例 7.2.1 大学の理工系学部の時間割を作成することを考えてみます．例えば関西大学の場合，理工系の学部・学科が 3 学部 9 学科あります．これらの学部・学科全体を同時に扱うとなると，非常に規模の大きい問題を考えなくてはなりません．

そこで，問題を分割するために，9 学科間の時間割で相互に影響を及ぼしうる要因を考えてみると，次のようなものがあることがわかりました：

- 複数の学科にまたがって開講される科目，他学科の教員に授業を依頼する科目
- 特殊教室（PC 教室，製図教室，語学用教室）を利用する科目
  - 教室の数に限りがあるため，同時に開講できる数に限りがある

[5] [15]（2008 年の論文）では「変数が 1000 個未満なら何も考えずにソルバに解かせる」というポリシー案が挙げられています．2008 年からのソルバや PC の発展を考えると，この案は数倍から 10 倍程度上方に修正してよいだろう，というのが筆者の考えです．

表 7.2: MIPLIB2010 の問題セットと難易度による分類 (2014 年 9 月現在・抜粋)

Status	Name	Rows	Cols	NZs	Int	Bin	Con
Easy	atlanta-ip	21732	48738	257532	106	46667	1965
	buildingenergy	277594	154978	788969	26287		128691
	co-100	2187	48417	1995817		48417	
	ex10	69608	17680	1162000		17680	
	go19	441	441	1885		441	
	lectsched-1	50108	28718	310792	482	28236	
	map06	328818	164547	549920		146	164401
	mspp16	561657	29280	27678735		29280	
	n3seq24	6044	119856	3232340		119856	
	neos-631710	169576	167056	834166		167056	
	ns1158817	68455	1804022	2842044		66022	173800
Hard	a1c1s1	3312	3648	10178		192	3456
	bg512142	1307	792	3953		240	552
	in	1526202	1449074	6811639		1489	1447585
	neos-847302	609	737	9566		729	8
	ns1663818	172017	124626	20433649		124626	
	probportfolio	302	320	6620		300	20
	zib02	9049868	37709944	146280582		37709944	
Open	d20200	1502	4000	189389	819	3181	
	hawaiiv10-130	1388052	685130	183263061		578444	106686
	liu	2178	1156	10626		1089	67
	nag	5840	2884	26499	35	1350	1499
	pigeon-19	3307	1444	29849		1273	171
	ramos3	2187	2187	32805		2187	
	sts405	27270	405	81810		405	
	zib01	5887041	12471400	49877768		12471400	

凡例：
Status: 問題の難易度（表 7.1），Name: 問題の名前，Rows: 制約条件の数，Cols: 変数の数，NZs: 係数行列に現れる非零要素の数，Int: 整数変数の数（0-1 変数は含まない），Bin: 0-1 変数の数，Con: 連続変数の数.
http://miplib.zib.de/miplib2010.php に掲載されている表から抜粋したものです．時間が経過すると問題の難易度が下がる（Open → Hard → Easy）可能性があることに注意してください.

すなわち，これらの科目の開講曜限を先に確定することができれば，残りの科目の時間割は各学科で個別に考えればよいことになります（ただし，各学科がどのような時間割を作成したとしても，それらを捌くことができるだけの十分な教室数が確保されているものとします）.

このような時間割の作成方法を採用した場合は，問題に現れる変数の一部を先に固定することになりますので，最終的に得られる解は，全体を同時に扱った場合の最適解よりも劣る可能性があります. しかしながら，問題を（各学科に）分割できるというメリットがあります. □

例 7.2.1 で見たように，問題の一部を固定することにより，元の問題を小さな規模の問題に分割する，という方法があります. これにより，問題の真の最適解を得ることはできなくなってしまいますが，もし元の問題が手に負えない程大きな問題であるのならば，この方法によって得られた解にも一定の意味はあるでしょう. 何より，得られた解は元の問題に対する実行可能解である，という点に注意してください.

7.3 制約条件を徐々に厳しくする（巡回セールスマン問題を例に）

ここでは，ある問題 P の最適解を求めることを考えます. このとき，問題 P が次のような性質を持っているものとします.

- 問題 P の全ての制約条件を列挙するのは難しい
- ある解が問題 P の実行可能解かどうかを調べるのは（比較的）簡単

以下では，この性質を生かした解法を考えてみます.

問題 P の全ての制約条件を列挙するのが難しいため，ここでは問題 P の制約条件を緩めた問題 P_1 を考え，これを解くことにします. もちろん，問題 P_1 を解いただけでは問題 P の最適解が得られるとは限りません. しかし，問題 P_1 を解いて得られた解が問題 P の実行可能解だったらどうでしょう？（上で述べたように，ここではこのチェックが簡単に行うことができるものとしています.）問題 P と P_1 の実行可能領域は，

「問題 P の実行可能領域」$\subseteq$「問題 P_1 の実行可能領域」

という包含関係が成立します. すなわち，問題 P の実行可能領域全体を含む問題（問題 P_1）を解いて得られた最適解が問題 P の実行可能解でもあったわけですから，これは問題 P の最適解になります.

では逆に，問題 P_1 を解いて得られた解が問題 P の実行可能解ではなかったとしましょう. このときは，問題 P_1 に問題 P の制約条件の一部を加えた問題 P_2 を考え，これを解きます. そして，問題 P_2 を解いて得られた最適解が問題 P の実行可能解かどうかを調べ，上記と同様の判断を行います. この手続きを繰り返すと，

$$\begin{aligned}\text{「問題 } P \text{ の実行可能領域」} &\subseteq \text{「問題 } P_l \text{ の実行可能領域」} \subseteq \text{「問題 } P_{l-1} \text{ の実行可能領域」} \\ &\subseteq \cdots \subseteq \text{「問題 } P_2 \text{ の実行可能領域」} \subseteq \text{「問題 } P_1 \text{ の実行可能領域」} \end{aligned} \tag{7.3.1}$$

のように，生成する問題の実行可能領域が元の問題 P の制約条件に近づいていき，最適解が求まる可能性が高まります. 最終的に，問題 P の制約条件が有限個であれば，有限回の操作で最適解を得

ることができます[6].

この方法が有効に機能する問題として，巡回セールスマン問題（問題 3.1.1）が挙げられます．巡回セールスマン問題を直接定式化し解くためには，例えば 6.10 節で取り上げた定式化を利用することができますが，ここでは上で説明した枠組みに沿った求解方法を紹介します．

まず，次の問題を考えてみましょう．

$$
\left|\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{i\in I, j\in J} d_{ij}x_{ij} \\
\text{subject to} & \displaystyle\sum_{j\in J} x_{ij} = 1 \quad (i \in I) \\
& \displaystyle\sum_{i\in I} x_{ij} = 1 \quad (j \in J) \\
& x_{ii} = 0 \quad (i \in I) \\
& x_{ij} \in \{0,1\} \quad (i \in I, j \in J)
\end{array}\right. \tag{7.3.2}
$$

ここで，図 3.1 で示した都市の配置を対象にして (7.3.2) を解くと，図 7.1 のような解を得ます．図 7.1 に描かれている線は，セールスマンが移動する経路，すなわち都市 i と j に対して $x_{ij} == 1$

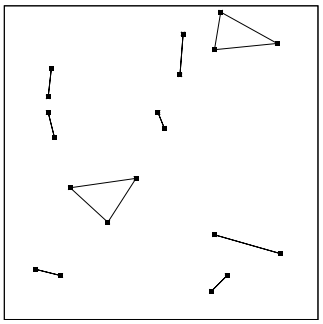

図 7.1: (7.3.2) による求解結果

となるような箇所を表しています．ここで，線分に見える箇所は，その端にある 2 つの都市を往復するような経路になっています（そうすることで，(7.3.2) の 1,2 番目の制約を満たしています）．もちろん，図 7.1 はこの問題の最適解ではありません．最適解は図 3.3 に示したように 1 本の経路（= 単純閉路）でなくてはなりません．しかし図 7.1 には小さな閉路（長さが 2 のものを含む）が複数見られます．このような閉路を**部分巡回路**といいます．では，考えられる部分順回路はどの程度あるのでしょうか？例えば図 3.1 には都市が 20 あります．これに対し，長さが 2 の部分順回路の総数は，「20 都市から 2 都市を選ぶ組合せの総数」です．これは ${}_{20}C_2 = (20 \times 19)/(2 \times 1) = 190$ 通りです．以下同様に，長さが 3 の部分順回路の総数は 1140 通り，長さが 4 なら 4845 通り，長さが 5 なら 15504 通り，長さが 6 なら 38760 通り…などとなります．これら全てを制約とするのは，いかにも筋の悪そうなアプローチです[7].

[6]ただし,「問題 P の全ての制約条件を列挙するのは難しい」のですから，全ての制約条件を列挙する前に最適解が求まることを期待しながら解くことになります．

[7]このアプローチは現実的ではないので，以下は全くの蛇足ですが…長さが 18 以下の部分順回路を考えれば十分です.「長

そこでここでは，次のような作戦を採用することにします．

- 問題 (7.3.2) に，長さが 2 の部分巡回路が生じなくなるような制約条件を追加した問題を P_1 とし，これを解きます．
- 直前に解いた問題を P_k $(k = 1, 2, \ldots)$ とするとき，その最適解に部分巡回路が含まれていたら，それを削除するような制約条件を問題 P_k に追加し，それを P_{k+1} とします．
- 以下，最適解に部分順回路が含まれなくなるまで，これを繰り返します．

それでは，この作戦を順に見ていきましょう．まず「長さが 2 の部分巡回路が生じなくなるような制約条件」ですが，これは

$$x_{ij} + x_{ji} \leq 1 \quad (i \in I, j \in J, i < j) \tag{7.3.3}$$

と書くことができます．長さが 2 の部分順回路というのは，都市 i から都市 j へ向かい（$x_{ij} = 1$），都市 j から都市 i に戻る（$x_{ji} = 1$）ということです．これを禁止するためには，x_{ij} と x_{ji} の少なくとも一方が 0 であることが必要です．式 (7.3.3) はこのことを表現しています．なお，$x_{ij} + x_{ji} = x_{ji} + x_{ij}$ であることに注意すれば，i の方が小さい場合だけ考えれば十分であることがわかります．問題 (7.3.2) にこの制約を追加した問題，すなわち問題 P_1 は次のようになります．

$$\left|\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i \in I, j \in J} d_{ij} x_{ij} \\ \text{subject to} & \displaystyle\sum_{j \in J} x_{ij} = 1 \quad (i \in I) \\ & \displaystyle\sum_{i \in I} x_{ij} = 1 \quad (j \in J) \\ & x_{ii} = 0 \quad (i \in I) \\ & x_{ij} + x_{ji} \leq 1 \quad (i \in I, j \in J, i < j) \\ & x_{ij} \in \{0, 1\} \quad (i \in I, j \in J) \end{array}\right. \tag{7.3.4}$$

問題 (7.3.4) は，問題 (6.10.1) からいくつかの制約条件（と変数）を除き，制約 (7.3.3) を追加したものです．まず，制約条件を除いた分，問題 (7.3.4) は問題 (6.10.1) の制約条件を緩めたものになっているはずです．また，問題 (6.10.1) を解いて得られる最適解は必ず単純閉路となるため，それは制約 (7.3.3) を満たしていることがわかります．すなわち，制約 (7.3.3) は問題 (6.10.1) の制約条件を厳しくするものではない，ということがわかります．従って，問題 (7.3.4) は問題 (6.10.1) の制約条件を緩めているので，上の説明に沿って考えると，問題 (6.10.1) を問題 P とするならば，問題 (7.3.4) は問題 P_1 になりうることがわかります．

実際に図 3.1 で示した都市の配置を対象にして問題 P_1（= 問題 (7.3.4)）を解くと（データは C.10 節にあります），図 7.2 のような最適解を得ます．制約 (7.3.3) のおかげで，長さが 2 の部分順回路が消えていることがわかります．しかし，図 7.1 にはなかった部分順回路が生じていることもわかります．

さが 19 の部分順回路が生じる」ということは「残りの 1 節点は単独で存在する」ということになりますが，$x_{ii} = 0$ $(i \in I)$ という制約があるので，このような状況は発生しません．また，「長さが 20 の部分順回路」は，(20 都市の巡回セールスマン問題において）まさに我々が求めたいものです．

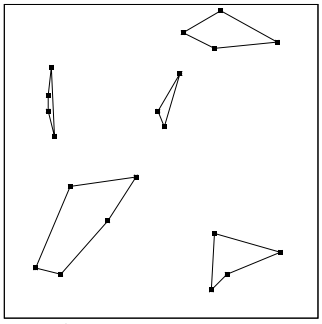

図 7.2: (7.3.4) による求解結果

そこで，問題 P_1 を解いて生じた部分順回路を削除することにします．ここでは，部分巡回路の集合を T とし，部分順回路 $t \in T$ は，

$$t : v_1, v_2, \ldots, v_p$$

とすることにします（(A.4.1) で示した表記法を用いています）[8]．すなわち，部分順回路の長さは p ということです．この部分巡回路を削除するような制約条件は，例えば

$$x_{v_1,v_2} + x_{v_2,v_3} + \cdots + x_{v_{p-1},v_p} + x_{v_p,v_1} \leq p - 1 \tag{7.3.5}$$

とすることができます．この制約を設ければ，p 本の枝 $(v_1, v_2), (v_2, v_3), \ldots, (v_{p-1}, v_p), (v_p, v_1)$ のうち，少なくとも 1 本は通過しなくなるはずです．このように，部分順回路を削除するための制約を**部分巡回路除去制約**といいます．

上で説明したように，問題 P_k $(k = 1, 2, \ldots)$ を解くたびに，生じた部分順回路制約 (7.3.5) を追加することにします．すると，問題が $P_1, P_2, \ldots$ と更新されるにつれて制約条件が増えることになるため，実行可能領域は徐々に狭まっていくことがわかります．また，元の巡回セールスマン問題の最適解においては部分巡回路が含まれてはなりませんから，ここで追加する制約条件は元の問題の実行可能領域を狭めることにはなりません．すなわち，(7.3.1) に示した関係が成立していることがわかります．そして，部分問題 P_l $(l \geq 1)$ を解いたときに得られた最適解に部分順回路が含まれなかったものとしましょう．部分順回路が含まれないということは，全都市を通過する単純閉路が得られたということです．すなわち,「元の問題よりも実行可能領域が広いにも関わらず，元の問題の実行可能解が最適解となった」ということですから，このとき得られた解は問題 P_l の最適解であるとともに，元の問題の最適解でもあります．

上でも述べたように，部分順回路除去制約として与えた不等式は，現在考えている問題 P_k の実行可能領域を狭めるものの，元の問題の実行可能領域を削ることはありません．このような不当式を**妥当不等式**といいます．またこの不等式を等式にすると平面の式が得られます．この平面のことを**切除平面**といいます．

実は，(7.3.5) よりもよい部分順回路除去制約として，

$$\sum_{i,j \in \{v_1,\ldots,v_p\}, i \neq j} x_{ij} \leq p - 1 \tag{7.3.6}$$

[8]本来であれば，P_1 を解いて生じた部分順回路は T ではなく T_1 と添字を付ける必要があるでしょう．また，部分順回路 t についても $t : v_1, v_2, \ldots, v_{p_t}, v_1$ などという表記をする必要があります．しかし，添字が煩雑になることと，添字を省略しても文意が正しく伝わるものと判断し，ここではこのような表記をすることにします．

が知られています．例えば，(7.3.5) の左辺に現れる項は全て (7.3.6) の左辺に現れることがわかります（確認してみてください）．よって，(7.3.6) が満たされれば (7.3.5) も満たされることがわかります．

さらに，上記の部分順回路 t に現れる都市 $\{v_1,\ldots,v_p\}$ に対して，訪問順序を変えた部分順回路，例えば t' : $v_1,v_2,\ldots,v_{p-2},v_p,v_{p-1},v_1$ を除去するためには

$$x_{v_1,v_2}+\cdots+x_{v_{p-2},v_p}+x_{v_p,v_{p-1}}+x_{v_{p-1},v_1}\leq p$$

という制約条件が必要になりますが，この制約条件の左辺に現れる項も全て (7.3.6) の左辺に現れるため，この部分順回路 t' も除去されることがわかります．すなわち，制約条件 (7.3.6) により，都市 $\{v_1,\ldots,v_p\}$ を通過するような全ての部分巡回路（$p!/p=(p-1)!$ 通りあります）が除去されることになります．

まとめると，P_2 として次の問題を解けばよいことがわかります．

$$\left|\begin{array}{ll}\text{minimize} & \displaystyle\sum_{i\in I,j\in J}d_{ij}x_{ij}\\ \text{subject to} & \displaystyle\sum_{j\in J}x_{ij}=1\quad(i\in I)\\ & \displaystyle\sum_{i\in I}x_{ij}=1\quad(j\in J)\\ & x_{ii}=0\quad(i\in I)\\ & x_{ij}+x_{ji}\leq 1\quad(i\in I,j\in J,i<j)\\ & \displaystyle\sum_{i,j\in V_k,i\neq j}x_{ij}\leq p_k-1\quad(k\in V_k)\\ & x_{ij}\in\{0,1\}\quad(i\in I,j\in J)\end{array}\right. \tag{7.3.7}$$

ここで，$k\in K$ は出現した部分巡回路の集合になり，V_k は部分巡回路 $k\in K$ に含まれる節点の集合，p_k は部分巡回路 $k\in K$ に含まれる枝の本数になります．

図 7.2 に対し (7.3.7) を用いて得られた結果が図 7.3 です（このときに用いたデータを C.10 節に示しています．以下同様）．図 7.3 を見ると，図 7.2 で見られた部分巡回路が除去されていることが

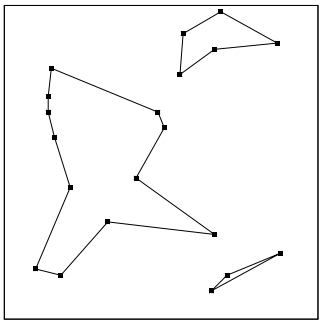

図 7.3: (7.3.7) による求解結果

わかります．しかし，依然として部分巡回路があります．

そこで，新たに出現した部分巡回路を除去する制約を追加しながら問題 (7.3.7) を繰り返し解くことになります．P_1,P_2 を解いて出現した部分巡回路を V_k として解く問題を P_3 としたとき，その結

果は図 7.4 のようになります（惜しい！）．さらに，P_1, P_2, P_3 を解いて出現した部分巡回路を V_k として解く問題を P_4 としたとき，その結果は図 7.4 のようになります．ここで，全部の都市を 1 回ずつ通過する経路（単純閉路）を得ることができました．これが最適解に他なりません．

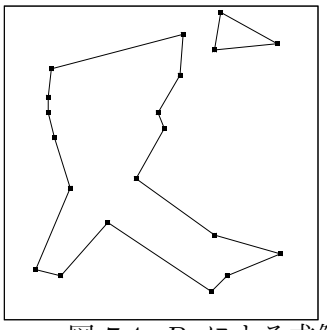

図 7.4: P_3 による求解結果

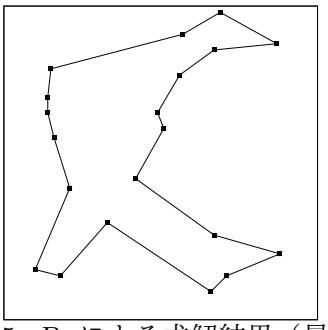

図 7.5: P_3 による求解結果（最終的な最適解）

さて，一連のプロセスを経て最適解を求めることができましたが，このとき目的関数値はどのように変化したかを調べてみましょう．表 7.3 のように，それぞれの問題における最適解における目的関

表 7.3: 目的関数値の変化

P_1	309.41
P_2	355.51
P_3	362.94
P_4	365.22

数値は単調増加（非減少）していることがわかります．これは，問題の制約条件を徐々に厳しくしながら問題を解いていることによるものです．逆に言えば，このようになっていなければ，制約を追加するプロセスに誤りがあったことになります．

Note: 部分順回路除去制約を追加する場合のもう一つの定式化 ここでは，6.10 節で取り上げた，Miller-Tucker-Zemlin (MTZ) 制約を用いた定式化 (6.10.1) から制約を減らした定式化 (7.3.2) を導入し，ここを出発点にして妥当不等式を逐次追加しながら元の問題の解を得る方法を紹介しました．

しかし，問題が対称巡回セールスマン問題である場合，都市間の距離が移動の向きに依存しないことを利用し，定式化 (6.10.1) や (7.3.2) で用いていたのとは異なる変数を設定して，妥当不等式を逐次追加する解法を構成することもできます．以下，この定式化を紹介します．

ここでは次のような変数を導入します．

$$x_e := \begin{cases} 1, & \text{枝 } e \text{ を通過する} \\ 0, & \text{枝 } e \text{ を通過しない} \end{cases}$$

この変数を用いると，上記 P_1 (7.3.4) に対応する問題として次の問題を考えることができます．

$$\left|\begin{array}{ll} \text{minimize} & \displaystyle\sum_{e\in E} d_e x_e \\ \text{subject to} & \displaystyle\sum_{e\in E(v)} x_e = 2 \quad (v \in V) \\ & x_e \in \{0,1\} \quad (e \in E) \end{array}\right. \tag{7.3.8}$$

ここで，考察対象の都市を節点として構成される完全グラフを $G = (V, E)$ とし，$E(v)$ は節点（都市）v に接続する枝の集合を表しているものとします．このとき，問題 (7.3.8) を解いて得られる解が部分順回路 t を含んでいたものとします．その部分順回路に含まれる枝の集合を E_t とすると，部分順回路 t を除去する制約は

$$\sum_{e\in E_t} x_e \le |E_t| - 1 \tag{7.3.9}$$

とすることができます．(7.3.9) が (7.3.5) に対応していることが確認できると思います．この制約を逐次追加することで，上と同様の解法を構築することができます．

7.4 制約条件を満たさなくてもよい，でもできるだけ満たしたい

ソルバを用いて最適化問題を解くと，得られる解は制約条件を必ず満たしたものになります（あるいは，問題に実行可能領域が存在しなければ，それを教えてくれます）．しかし，現実社会の問題では，

> 完全に満たされなくてもよいが，できるだけ満たしたい制約

という種類の制約が存在することがあります．例えば，シフト制の仕事のスケジュールを組む際に，

各自の休み希望日をできるだけ満たしたいが，希望の重複が多い日には（やむを得ず）何人かにシフトに入ってもらう，などの場合です．

このような場合に有効な考え方として，問題を**重み付き制約充足問題**として取り扱う，という方法が挙げられます．重み付き制約充足問題では，制約条件を**絶対制約**と**考慮制約**の 2 つに分類します[9]．

絶対制約は，必ず満たさなければならない制約です．すなわち，通常の最適化問題で設ける制約は絶対制約であるということができます．一方，考慮制約とは，できるだけ満たしたいが，やむを得ない場合（すなわち絶対制約を守るために必要な場合）は，満たさないこともあり得る制約のことです．

ソルバ（特に近似解法を用いるソルバ）によってはこれらの制約を直接扱うことができるものも存在しますが，ここでは通常のソルバで考慮制約を扱う方法について説明します．

例えば次のような最適化問題があるものとします．

$$\left|\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & g_1(\boldsymbol{x}) \leq 0 \end{array}\right. \tag{7.4.1}$$

(7.4.1) をそのままモデルファイルに記述すると，制約 $g_1 \leq 0$ は絶対制約として扱われます．一方，ある変数 h_1 を導入して次のような問題を考えたとします．

$$\left|\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) + w_1 h_1 \\ \text{subject to} & g_1(\boldsymbol{x}) \leq h_1 \\ & h_1 \geq 0 \end{array}\right. \tag{7.4.2}$$

ここで w_1 は正の値を取るパラメータです（その意味については後述します）．この問題では，$g_1 \leq 0$ という制約が $g_1 \leq h_1$ という制約に置き換わっています．そして変数 h_1 には非負制約が課され，かつ目的関数に組み込まれています．この変形は次のような意味を持っています．

- h_1 は，オリジナルの制約 $g_1 \leq 0$ の違反量を表している
- h_1 を目的関数に（正の係数 w を伴って）組み込むことにより，制約の違反量に対するペナルティの役割を果たしている
- 得られた解において $h_1 = 0$ となれば，(7.4.2) は (7.4.1) と同じ問題になっており，解も一致する
- 得られた解で $h_1 > 0$ であれば，目的関数 f とオリジナルの制約の違反量 h_1 の両方ができるだけ小さくなるような解を求めている（w は目的関数の値と制約の違反量のバランスを決めるパラメータ）

このように考えることにより,「できるだけ満たしたい」制約を実現することができます．

もちろん，この手法は複数の制約に対しても適用可能です．例えば

$$\left|\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & g_1(\boldsymbol{x}) \leq 0 \\ & g_2(\boldsymbol{x}) \leq 0 \end{array}\right. \tag{7.4.3}$$

[9]絶対制約のことを**ハード制約**，考慮制約のことを**ソフト制約**ということがあります．

という問題で，制約 $g_1 \leq 0, g_2 \leq 0$ の両方を考慮制約として扱う場合は

$$\left|\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) + w_1 h_1 + w_2 h_2 \\ \text{subject to} & g_1(\boldsymbol{x}) \leq h_1 \\ & g_2(\boldsymbol{x}) \leq h_2 \\ & h_1 \geq 0, h_2 \geq 0 \end{array}\right. \tag{7.4.4}$$

とすればよいでしょう．あるいは，

$$\left|\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & g_1(\boldsymbol{x}) = 0 \end{array}\right. \tag{7.4.5}$$

という問題において，等式制約 $g_1 = 0$ を考慮制約にするためには，

$$\left|\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) + w_1 h_1 \\ \text{subject to} & -h_1 \leq g_1(\boldsymbol{x}) \leq h_1 \\ & h_1 \geq 0 \end{array}\right. \tag{7.4.6}$$

とすることができます（(7.4.6) でよい理由を考えてみてください）．

7.5 定式化を工夫する

考えている問題を素直に定式化すると，ソルバで扱えないクラスの最適化問題になる場合があります．しかし，定式化の方法は一通りではありません．定式化の方法を変えることで，ソルバで扱えるクラスの問題になることがあります．

ここでは，次の問題を取り上げます．

問題 7.5.1 最小包含円問題

2 次元空間上の与えられた点群 (x_i, y_i) $(i = 1, 2, \ldots, n)$ を囲むような円のうち，半径が最小の円を求めよ（図 *7.6, 7.7*）．□

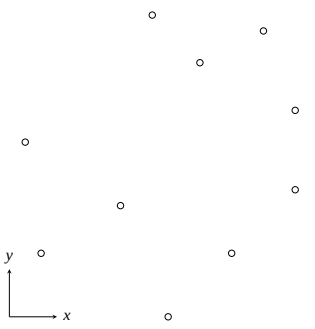

図 7.6: 最小包含円問題

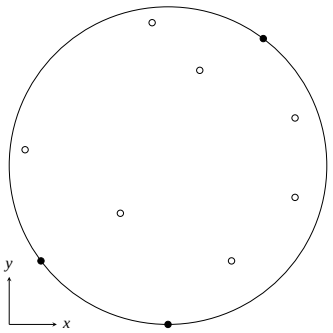

図 7.7: 最小包含円問題（答）

求める円の中心を (a, b)，円の半径を r とすると，この問題は次のように定式化できます．

$$\left|\begin{array}{ll} \text{minimize} & r^2 \\ \text{subject to} & (x_i - a)^2 + (y_i - b)^2 \leq r^2 \quad (i = 1, 2, \ldots, n) \end{array}\right. \tag{7.5.1}$$

このとき，(7.5.1) の目的関数は凸関数ですが，制約条件は凸では無いため，この問題は凸計画問題ではありません．従って，ソルバによってはこの問題を解くことはできません．

そこで，この問題を変形することを試みます [16]．(7.5.1) の制約条件を変形すると次のようになります．

$$(a^2+b^2-r^2)-2(ax_i+by_i)+(x_i^2+y_i^2)\leq 0 \quad (i=1,2,\ldots,n)$$

ここで中間変数 q を導入し $q=a^2+b^2-r^2$ とすると，

$$q-2(ax_i+by_i)+(x_i^2+y_i^2)\leq 0 \quad (i=1,2,\ldots,n)$$

となります．また，$r^2=a^2+b^2-q$ ですから，問題 (7.5.1) は次のように変形することができます．

$$\left|\begin{array}{ll} \text{minimize} & a^2+b^2-q \\ \text{subject to} & q-2(ax_i+by_i)+(x_i^2+y_i^2)\leq 0 \quad (i=1,2,\ldots,n) \end{array}\right. \tag{7.5.2}$$

(7.5.2) の変数は q,a,b ですから，目的関数は凸 2 次関数，制約条件は（一見 2 次式のようにも見えますが，ここでは x_i,y_i $(i=1,2,\ldots,n)$ は定数ですので）1 次関数です．従って，この問題は凸 2 次計画問題になり，この問題を扱うことができるソルバは増えます．

このように，定式化を見直すことで問題クラスが変わることがありますし，より扱いやすい問題になることもあります．

7.6　ソルバを用いるのか？用いないのか？

ここでは，次のようなパターンの問題について考えてみます．

- 問題は最適化問題だが，ソルバを使うのが得策ではない場合
- 一見最適化問題に見えないが，最適化問題に定式化できる場合

7.6.1　ソルバを使うのが得策ではない問題

ここでは,「ソルバを使うのが得策ではない」問題として，**最短路問題**を考えます．

6.7 節で説明したように，最短路問題は最適化問題として定式化・求解することが可能な問題です．グラフのサイズがそれほど大きくなければ，このアプローチでも十分対応できるでしょう．

しかし，グラフの規模があまりに大きくなった場合はどうでしょうか？近年，Graph 500 [29] というプロジェクトが注目されています．これは，グラフにおいて，単位時間あたりに探索できる枝の数を競うプロジェクトです [10]．ここで取り扱うグラフの規模は，節点数がなんと 10^40 個（！）にも及びます．あるいは，これほど大きくなくとも，各種 SNS での連結関係を表現したグラフ（ユーザを節点とし，フォローしているユーザ間に枝を設ける）などであれば，節点数は数億を優に超えることになります．そのようなグラフ上の 2 点に対し，ソルバを用いて最短経路を見つけるのは恐らくはかなり難しいでしょう．

[10] その他，単位消費電力あたりに探索できる枝の数を競う Green Graph 500 [30] もあります．

しかしながら，グラフ上の最短路（正確には最短路木）を求めるアルゴリズムには，効率のよいものが知られています．その代表格が**ダイクストラ法**です．この手法は**最適性の原理**に基づいた手法で，かなり大きな規模の問題に対しても非常によく機能します．さらに，ダイクストラ法に基づく工夫を行うことによって，例えば全米の道路網（節点数 2500 万，枝数 5800 万）における最短路を求めることにも成功した事例もあります [26].

実は，7.5 節で扱った最小包含円問題も同様の特徴を持つ問題です．7.5 節では，自然に定式化すると NLP だが工夫することで QP となって扱いやすくなる，という主旨の説明をしました．しかし，最小包含円問題は計算幾何の分野で古くから研究されている問題であり，効率のよいアルゴリズムが存在することが知られています [7].

これらの事例から学ぶべきことは，

- 考えている問題本当に最適化問題として定式化して解くべき問題なのか？
- 定式化したとしてソフトウェアで解けるのか？

ということになるでしょう．刃物だからといって，斧でリンゴの皮をむいてはいけないし，包丁で木を切り倒してはいけないのです．

7.6.2 最適化問題に見えないが最適化問題として定式化できる問題

ここでは,「最適化問題に見えないが最適化問題として定式化できる問題」の例として，**数独**を例に取り上げます．

数独とは，図 7.8 に示したような 9×9 のマス目に 1 から 9 の数字を埋めていくパズルです．ただし，次のルールを満たすように数字を埋めなくてはなりません：

- 各行，各列に 1 から 9 が 1 回ずつ現れる
- 太線で区切られた 3×3 のブロック内に 1 から 9 が 1 回ずつ現れる

新聞・雑誌等にもよく掲載されているのでご存知の方も多いでしょう．ここでは，ソルバを用いてこの問題を解いてみます．

…とはいっても，この問題では何を最適化するのかが明らかではないと思います．そう，この問題には「目的関数」に相当するものはありません．一方，上で説明したように，マス目に数字を埋めるときのルールは存在します．これは，最適化問題では「制約条件」に相当するものです．つまり，ここでは数独を「制約のみがある最適化問題」として定式化して解くことにします．制約を満たす解を見つければよいので，この問題は**制約充足問題**といいます（7.4 節の重み付き制約充足問題と比較してみてください）.

ここでの定式化では，変数として

$$x_{ijk} = \begin{cases} 1, & \text{マス目 } (i,j) \text{ に数字 } k \text{ が入る} \\ 0, & \text{マス目 } (i,j) \text{ に数字 } k \text{ が入らない} \end{cases}$$

を用いることにします．ここで，マス目 (i,j) とは i 行目・j 列目のマス目を指すものとします．行数・列数と数独で用いる数字はいずれも $1,2,\ldots,9$ ですから，$i \in I, j \in J, k \in K$ とするとき，集合 I, J, K はいずれも $\{1,2,\ldots,9\}$ となります．

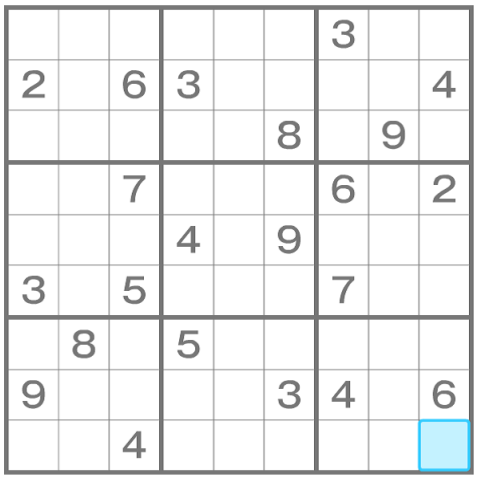

図 7.8: 数独の問題例
nikori.com「数独のおためし問題」より引用 (http://www.nikoli.com/ja/puzzles/sudoku/)

この変数を用いて数独を制約充足問題として定式化すると次のようになります：

$$
\begin{aligned}
&\text{minimize} \quad 0 \\
&\text{subject to} \quad \sum_{k\in K} x_{ijk} = 1 \quad (i \in I, j \in J) \\
&\qquad\qquad \sum_{j\in J} x_{ijk} = 1 \quad (i \in I, k \in K) \\
&\qquad\qquad \sum_{i\in I} x_{ijk} = 1 \quad (j \in J, k \in K) \\
&\qquad\qquad \sum_{m\in M, n\in N} x_{3p-m,3q-n,k} = 1 \quad (p \in P, q \in Q, k \in K) \\
&\qquad\qquad x_{ijk} = 1 \quad ((i,j,k) \in C) \\
&\qquad\qquad x_{ijk} \in \{0,1\} \quad (i \in I, j \in J, k \in K)
\end{aligned}
\tag{7.6.1}
$$

上で説明したように，ここでは特別な目的関数を設定する必要がないので，(7.6.1) では目的関数を 0 に設定しています（別に 0 である必然性もありません）．通常，数独の問題は答が一通りになるように作成してありますので，目的関数をどう設定しようとも，得られる答は変わりません．

次に，(7.6.1) に現れる制約条件を順に説明していきます．

最初の制約は，各マス目 (i,j) $(i \in I, j \in J)$ に数字 k がいずれか一つだけ入ることを表しています（もしこの制約がないと何が起こるか，考えてみてください）．

2 番目，3 番目の制約は，それぞれ i 行目，j 列目に数字 k がちょうど 1 回現れることを表しています．

4 番目の制約に現れる集合 P, Q と M, N は $P = Q = \{1,2,3\}$, $M = N = \{0,1,2\}$ と定義するものとします．まず，$p \in P, q \in Q$ は 3×3 のブロック番号を表しています．左上のブロックを $p = 1, q = 1$，その右側のブロックを $p = 1, q = 2$ …のように番号付けしていきます．ここでは，$p = q = 1$ の場合を考えることにします．このとき，$m \in M = \{0,1,2\}, n \in N = \{0,1,2\}$ に対して，$(3p - m, 3q - n) = (3 - m, 3 - n)$ は $(1,1), (1,2), \ldots, (3,3)$ の 9 通りになります．これは左上のブ

ロックに含まれるマス目に他なりません．そして，これらのマス目に対応する $x_{3p-m,3q-n,k}$ の和が 1 になるということは，このブロック内で数字 k がちょうど 1 回現れるということを意味しています．

5 番目の制約は，問題出与えられている数字を固定するのに用います．例えば図 7.8 ではマス目 $(1,7)$ に数字 3 が入ることがわかっています．このときは，集合 C に $(1,7,3)$ を加えておけば，x_{173} が 1 になる，すなわちマス目 $(1,7)$ に数字 3 が入ることになります．

問題 (7.6.1) をモデリング言語で記述すると次のようになります．

```
set C dimen 3;

set I;
set J;
set K;

set P;
set Q;

set M;
set N;

var x{I, J, K} binary;

minimize Const:
0;

subject to C_elem {i in I, j in J}:
sum {k in K} x[i, j, k] == 1;

subject to C_column {j in J, k in K}:
sum {i in I} x[i, j, k] == 1;

subject to C_row {i in I, k in K}:
sum {j in J} x[i, j, k] == 1;

subject to C_block {p in P, q in Q, k in K}:
sum {m in M, n in N} x[3*p-m, 3*q-n, k] == 1;

subject to C_fixed {(i, j, k) in C}:
x[i, j, k] == 1;

end;
```

さらに，図 7.8 に対応するデータファイルは次のようになります．

```
set I := 1 2 3 4 5 6 7 8 9;
set J := 1 2 3 4 5 6 7 8 9;
set K := 1 2 3 4 5 6 7 8 9;

set P := 1 2 3;
set Q := 1 2 3;

set M := 0 1 2;
set N := 0 1 2;
```

```
set C :=
    (1, 7, 3),
    (2, 1, 2), (2, 3, 6), (2, 4, 3), (2, 9, 4),
    (3, 6, 8), (3, 8, 9),
    (4, 3, 7), (4, 7, 6), (4, 9, 2),
    (5, 4, 4), (5, 6, 9),
    (6, 1, 3), (6, 3, 5), (6, 7, 7),
    (7, 2, 8), (7, 4, 5),
    (8, 1, 9), (8, 6, 3), (8, 7, 4), (8, 9, 6),
    (9, 3, 4);

end;
```

これらを用いて問題を解くと，次のような解ファイルを得ることができます（変数 x_{ijk} の値が 1 になっている箇所のみを抜粋）.

```
     8 x[1,1,8]     *              1              0              1
    13 x[1,2,4]     *              1              0              1
    27 x[1,3,9]     *              1              0              1
    34 x[1,4,7]     *              1              0              1
    41 x[1,5,5]     *              1              0              1
    47 x[1,6,2]     *              1              0              1
    57 x[1,7,3]     *              1              0              1
    69 x[1,8,6]     *              1              0              1
    73 x[1,9,1]     *              1              0              1
    83 x[2,1,2]     *              1              0              1
    95 x[2,2,5]     *              1              0              1
   105 x[2,3,6]     *              1              0              1
   111 x[2,4,3]     *              1              0              1
   126 x[2,5,9]     *              1              0              1
   127 x[2,6,1]     *              1              0              1
   143 x[2,7,8]     *              1              0              1
   151 x[2,8,7]     *              1              0              1
   157 x[2,9,4]     *              1              0              1
   169 x[3,1,7]     *              1              0              1
   174 x[3,2,3]     *              1              0              1
   181 x[3,3,1]     *              1              0              1
   195 x[3,4,6]     *              1              0              1
   202 x[3,5,4]     *              1              0              1
   215 x[3,6,8]     *              1              0              1
   218 x[3,7,2]     *              1              0              1
   234 x[3,8,9]     *              1              0              1
   239 x[3,9,5]     *              1              0              1
   247 x[4,1,4]     *              1              0              1
   261 x[4,2,9]     *              1              0              1
   268 x[4,3,7]     *              1              0              1
   271 x[4,4,1]     *              1              0              1
   282 x[4,5,3]     *              1              0              1
   293 x[4,6,5]     *              1              0              1
   303 x[4,7,6]     *              1              0              1
   314 x[4,8,8]     *              1              0              1
   317 x[4,9,2]     *              1              0              1
   330 x[5,1,6]     *              1              0              1
   335 x[5,2,2]     *              1              0              1
```

```
350 x[5,3,8]     *              1              0              1
355 x[5,4,4]     *              1              0              1
367 x[5,5,7]     *              1              0              1
378 x[5,6,9]     *              1              0              1
383 x[5,7,5]     *              1              0              1
388 x[5,8,1]     *              1              0              1
399 x[5,9,3]     *              1              0              1
408 x[6,1,3]     *              1              0              1
415 x[6,2,1]     *              1              0              1
428 x[6,3,5]     *              1              0              1
434 x[6,4,2]     *              1              0              1
449 x[6,5,8]     *              1              0              1
456 x[6,6,6]     *              1              0              1
466 x[6,7,7]     *              1              0              1
472 x[6,8,4]     *              1              0              1
486 x[6,9,9]     *              1              0              1
487 x[7,1,1]     *              1              0              1
503 x[7,2,8]     *              1              0              1
507 x[7,3,3]     *              1              0              1
518 x[7,4,5]     *              1              0              1
528 x[7,5,6]     *              1              0              1
535 x[7,6,4]     *              1              0              1
549 x[7,7,9]     *              1              0              1
551 x[7,8,2]     *              1              0              1
565 x[7,9,7]     *              1              0              1
576 x[8,1,9]     *              1              0              1
583 x[8,2,7]     *              1              0              1
587 x[8,3,2]     *              1              0              1
602 x[8,4,8]     *              1              0              1
604 x[8,5,1]     *              1              0              1
615 x[8,6,3]     *              1              0              1
625 x[8,7,4]     *              1              0              1
635 x[8,8,5]     *              1              0              1
645 x[8,9,6]     *              1              0              1
653 x[9,1,5]     *              1              0              1
663 x[9,2,6]     *              1              0              1
670 x[9,3,4]     *              1              0              1
684 x[9,4,9]     *              1              0              1
686 x[9,5,2]     *              1              0              1
700 x[9,6,7]     *              1              0              1
703 x[9,7,1]     *              1              0              1
714 x[9,8,3]     *              1              0              1
728 x[9,9,8]     *              1              0              1
```

この結果より，図 7.8 に対する答えは図 7.9 のようになります.
このように，一見最適化問題に見えなくとも，ソルバを用いて解くことができる問題があります.

8	4	9	7	5	2	3	6	1
2	5	6	3	9	1	8	7	4
7	3	1	6	4	8	2	9	5
4	9	7	1	3	5	6	8	2
6	2	8	4	7	9	5	1	3
3	1	5	2	8	6	7	4	9
1	8	3	5	6	4	9	2	7
9	7	2	8	1	3	4	5	6
5	6	4	9	2	7	1	3	8

図 7.9: 図 7.8 の答え

第8章　おわりに代えて

8.1　本書で学んだこと

本書では，タイトルにもあるように，ソルバというソフトウェアを用いて問題を解くことを前提にして,「数理最適化」の入門部分について学びました．具体的には，次のような項目について学びました．

- 最適化問題のモデル化と定式化
  - 定式化のための基礎知識
  - 定式化でよく用いる表現
- モデリング言語によるモデルの表現方法
- ソルバを用いた求解と得られた解の分析方法
- 最適解を得るのが難しい問題への対処法
- ケーススタディ

（以下追記予定）

8.2　本書を超えるための「次の一歩」

本書で学んだのは，数理最適化の「最初の一歩」でした．本稿だけでも一定の内容をカバーしているという自負はありますが，本書を読み終えた方には是非「次の一歩」を踏み出して頂きたいと思います．そのためのアクションとしては，例えば次のようなことが考えられます．

- 自分の周りの最適化問題を解いてみる
- 本書で取り上げなかった問題クラスについて学ぶ
  - 付録 B でそのような問題クラス（の一部）を取り上げています
- 様々な分野での最適化技法の使われ方を学ぶ
  - 機械学習，データマイニング，ビッグデータ（の処理），画像処理，言語処理など様々な分野で最適化の技法が用いられています
- 解法（の概要）について学ぶ
- ソルバをサブルーチンとして用いるような解法を学ぶ

- 必ずしも最適化を用いない問題解決方法について学ぶ

最後に挙げた項目について補足します．本書で説明したように，最適化は非常に有用な問題解決方法ではありますが，残念ながら「万能薬」ではありません（その一端を 7.6 節で紹介しました）．最適化には最適化が得意な問題領域がありますし，シミュレーションや統計にもそれぞれの得意な問題領域があります（もちろん他にも様々な「道具」があります）．それらの手法を組み合わせることで我々はより大きな問題に立ち向かうことができるはずです．

本書を読んだ方が数理的な問題に立ち向かうときに，道具の一つとして「（数理）最適化」を思い出してもらえるならば，それは筆者として望外の喜びです．

付 録 A　数学的準備

ここでは，本書で必要となる数学的事項についてまとめます．本書を読む際にわからない数学的事項がある場合，本章を参照するとよいでしょう．

A.1　本書で用いる数学記号

本書で用いる数学記号をここにまとめておきます．なお，特に断りの無い場合，本書ではベクトルは小文字の太字，行列は大文字の太字で表記します（例: ベクトル $\boldsymbol{a}$，行列 $\boldsymbol{A}$ など）．

- $\mathbb{R}$: 実数の集合．
- $\mathbb{R}^n$: 実数を成分とする n 次元ベクトルの集合．
- $\mathbb{R}^{m\times n}$: 実数を成分とする $m\times n$ 行列の集合．
- $\mathbb{N}$: 自然数の集合 $(1,2,3,\ldots)$．
- $I=\{1,2,3,4,5\}$: 集合 I の要素は $1,2,3,4,5$（$\{\ldots\}$ は集合を表します）．
- $i\in I$: i は集合 I の要素（あるいは元）．
- $|I|$: 集合 I の要素の数．
- $S=\{\boldsymbol{x}\in\mathbb{R}^n \mid (\text{条件})\}$: S は (条件) を満たすベクトル $\boldsymbol{x}\in\mathbb{R}^n$ の集合．
- $\displaystyle\sum_{i\in I} a_i$: a_i $(i\in I)$ の和．
- $\min\{x_1,x_2,\ldots,x_m\}$: x_i $(i=1,2,\ldots,m)$ の最小値．
- $\max\{x_1,x_2,\ldots,x_m\}$: x_i $(i=1,2,\ldots,m)$ の最大値．
- $P\subseteq Q$: 集合 P は集合 Q の部分集合（図 A.1）．
- $X=Y\times Z$: 集合 X は集合 Y と集合 Z の直積（あるいは直積集合）．すなわち，集合 X は集合 Y の要素 y と集合 Z の要素 z の組 (y,z) 全体．

- $\boldsymbol{a}^\top, \boldsymbol{A}^\top$: ベクトル $\boldsymbol{a}$, 行列 $\boldsymbol{A}$ を転置したベクトル，行列.

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \quad \text{ならば} \quad \boldsymbol{a}^\top = (a_1, a_2, \ldots, a_n),$$

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad \text{ならば} \quad \boldsymbol{A}^\top = \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ a_{12} & & a_{m2} \\ \vdots & & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix}.$$

- $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$: ベクトル $\boldsymbol{a}$ とベクトル $\boldsymbol{b}$ の内積．$\boldsymbol{a}^\top \boldsymbol{b}$ と同じ.
- $A \cup B$: 集合 A と集合 B の和集合，すなわち A または B に属する要素の集合.
- $\cup_{i \in I} A_i$: 集合 A_i $(i \in I)$ のいずれかに属する要素の集合.
- $A \cap B$: 集合 A と集合 B の積集合，すなわち A と B の両方に属する要素の集合.
- $\cap_{i \in I} A_i$: 集合 A_i $(i \in I)$ の全てに属する要素の集合.
- $f : \mathbb{R}^n \to \mathbb{R}^m$: f は n 次元の実数ベクトルに応じて m 次元の実数ベクトルを定める関数.
- $\boldsymbol{I}$: 単位行列.
- $\|\boldsymbol{x}\|$: $\boldsymbol{x}$ のノルム（正確には 2–ノルム）．$\boldsymbol{x} \in \mathbb{R}^n$ とすると $\|\boldsymbol{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$.
- $\frac{\partial f(\boldsymbol{x})}{\partial x_i}$: 関数 $f(\boldsymbol{x})$ $(f : \mathbb{R}^n \to \mathbb{R})$ の変数 x_i による偏導関数.
- $\nabla f(\boldsymbol{x}) = (\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_n})^\top$: 関数 $f(\boldsymbol{x})$ $(f : \mathbb{R}^n \to \mathbb{R})$ の勾配.

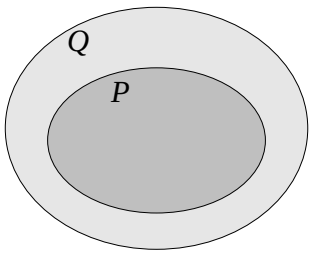

図 A.1: 包含関係 $(P \subseteq Q)$

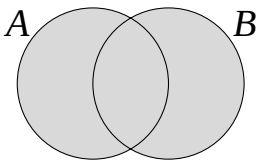

図 A.2: 和集合 $(A \cup B)$

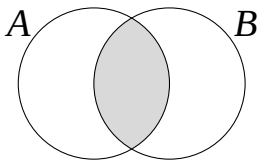

図 A.3: 積集合 $(A \cap B)$

A.2 1 次関数・非線形関数

関数 $f:\mathbb{R}^n \to \mathbb{R}$ が次数 1 以下の多項式であるとき，これを 1 **次関数**といいます．また関数 $f:\mathbb{R}^n \to \mathbb{R}^m$ を構成する関数 $f_i:\mathbb{R}^n \to \mathbb{R}$ $(i=1,2,\ldots,m)$ がそれぞれ次数 1 以下の多項式であるとき，これも 1 次関数ということにします[1]．また本書では 1 次関数でない関数を非線形関数と呼ぶことにします（次の **Note** も参照してください）．

Note: 「一次」?「線形」? 通常，1 次関数は英語で “linear function” といいます．ところがこの “linear” という言葉は「線形」と訳されることも多い単語です．例えば，“linear algebra”=「線形代数」です．これを「1 次代数」ということはまずありません．一方で，手元の『岩波 数学入門辞典』には「1 次関数」という見出しはありますが,「線形関数」という見出しはありませんところが，数理最適化分野では「区分線形関数」= “piecewise linear function” という関数を扱うことがあります（本書でも扱います）．単語に応じてよく用いられる訳が異なることに注意してください．

一般に $f:\mathbb{R}^n \to \mathbb{R}^m$ であるような 1 次関数は

$$f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \quad (\boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{b} \in \mathbb{R}^m) \tag{A.2.1}$$

という形で書くことができます．特に $m=1$ の場合を考えると

$$f(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{x} + b = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b \quad (\boldsymbol{a} \in \mathbb{R}^n, b \in \mathbb{R}) \tag{A.2.2}$$

となります $(f:\mathbb{R}^n \to \mathbb{R})$．さらに $n=m=1$ の場合は

$$f(x) = ax + b \quad (a \in \mathbb{R}, b \in \mathbb{R}) \tag{A.2.3}$$

となります $(f:\mathbb{R} \to \mathbb{R})$.

例 A.2.1 1 次関数・非線形関数の例

- $f(x) = 2x + 3$ は 1 次関数 $(f:\mathbb{R}\to\mathbb{R})$．これは (A.2.3) において $a=2, b=3$ としたもの．
- $f(\boldsymbol{x}) = x_1 - 2x_2 + 3$ は 1 次関数 $(f:\mathbb{R}^2\to\mathbb{R})$．これは (A.2.2) において $\boldsymbol{a} = (1,-2)^\top, b=3$ としたもの．
- $f(\boldsymbol{x}) = (x_1 - 2x_2 + 1, -x_1 + 3x_2 - 4)^\top$ は 1 次関数 $(f:\mathbb{R}^2\to\mathbb{R}^2)$．これは (A.2.1) において

  $$\boldsymbol{A} = \begin{bmatrix} 1 & -2 \\ -1 & 3 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ -4 \end{bmatrix}$$

  としたもの．
- $f(x) = \sin x$ は非線形関数（1 次関数ではない）$(f:\mathbb{R}\to\mathbb{R})$.

[1] 前者のみを 1 次関数として定義する教科書も少なくありません．

- $f(\boldsymbol{x}) = \log(x_1^2 + x_2^2 + 1)$ は非線形関数（1 次関数ではない）($f : \mathbb{R}^2 \to \mathbb{R}$).
- $f(x) = (x + 3, x^2)^\top$ は非線形関数（1 次関数ではない）($f : \mathbb{R} \to \mathbb{R}^2$).
- $f(x) = |x|$ は 非線形関数 ($f : \mathbb{R} \to \mathbb{R}$). 次数は 1 だが，多項式ではない.

A.3　凸性の基礎

数理最適化では,『凸』という性質が非常に重要になります．凸とは，集合について言う場合 と，関数について言う場合 があります．ここでは，それらの定義といくつかの基本的な性質について説明します.

A.3.1　凸集合・凸関数

n 次元の実数値の集合 $S \subseteq \mathbb{R}^n$ が**凸集合**（あるいは単に「S が凸」）であるとは,

$$\boldsymbol{x} \in S,\ \boldsymbol{y} \in S,\ 0 \leq \alpha \leq 1 \quad \Rightarrow \quad (1-\alpha)\boldsymbol{x} + \alpha\boldsymbol{y} \in S$$

が成立することをいいます（図 A.4）. これは，集合 S に含まれる 2 点 $\boldsymbol{x}, \boldsymbol{y}$ を結ぶ線分上の任意の点がやはり S に含まれることを意味します．直感的には「凹みのない集合」と言っても差し支えないでしょう.

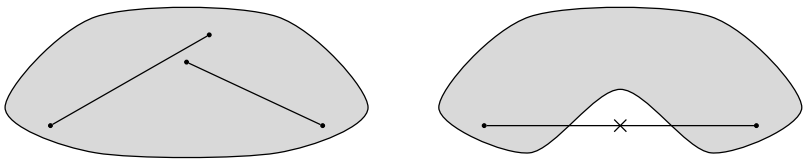

図 A.4: 凸集合の例（左：凸集合，右：凸集合でない）

また，$f : \mathbb{R}^n \to \mathbb{R}$ が**凸関数**（あるいは単に「f が凸」）であるとは,

$$\boldsymbol{x} \in \mathbb{R}^n,\ \boldsymbol{y} \in \mathbb{R}^n,\ 0 \leq \alpha \leq 1 \quad \Rightarrow \quad f((1-\alpha)\boldsymbol{x} + \alpha\boldsymbol{y}) \leq (1-\alpha)f(\boldsymbol{x}) + \alpha f(\boldsymbol{y})$$

が成立することをいいます（図 A.5）[2]. これは，直感的には「グラフ上の 2 点を結ぶ線分がグラフよりも上側にある」ということを表しています．またこれは，関数 f の「上側」の部分，つまり $\{(\boldsymbol{x}, t) \in \mathbb{R}^{n+1} \mid f(\boldsymbol{x}) \leq t\}$（これを**エピグラフ**といいます）が凸集合であることと一致します（すなわち，これを凸関数の定義と考えても構いません）.

以下に，凸集合・凸関数の簡単な例を挙げます．これらが凸集合・凸関数になる理由を考えてみましょう.

[2]凸解析では f の取りうる値として $-\infty$ と ∞ を許した場合を考えることがよくありますが（例えば [11]），本書ではそのような場合を考えないことにします．また高校の教科書などでは，このような性質のことを「下に凸」と表現することがありますが，数理最適化分野の専門書や論文などでは「下に凸」という表現は通常用いません.

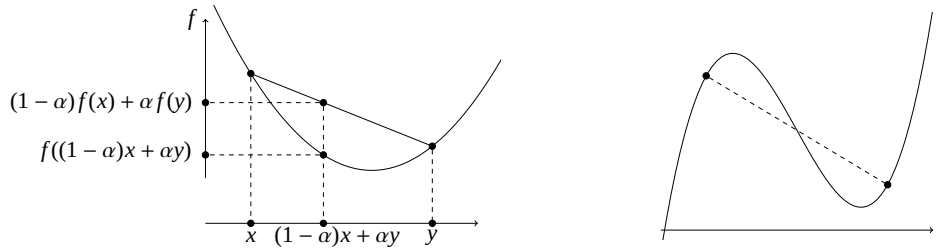

図 A.5: 凸関数の例（左：凸関数，右：凸関数でない）

例 A.3.1 凸集合の例・凸集合でない例

- n 次元空間の集合 $\{\boldsymbol{x} \in \mathbb{R}^n \mid x_i \geq 0\ (i = 1, 2, \ldots, n)\}$ は凸集合
- 1 次 不等式 で定められる n 次元空間の集合 $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}^\top \boldsymbol{x} \leq b\}$ $(\boldsymbol{a} \in \mathbb{R}^n, b \in \mathbb{R})$ は凸集合（図 A.6 は $n = 2, \boldsymbol{a} = (-1, 1)^\top, b = 1$ の場合）
- 1 次 方程式 で定められる n 次元空間の集合 $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}^\top \boldsymbol{x} = b\}$ $(\boldsymbol{a} \in \mathbb{R}^n, b \in \mathbb{R})$ は凸集合（図 A.7 は $n = 2, \boldsymbol{a} = (-1, 1)^\top, b = 1$ の場合）
- 2 次不等式で定められる n 次元空間の集合 $\{\boldsymbol{x} \in \mathbb{R}^n \mid x_1^2 + x_2^2 + \cdots + x_n^2 \leq 1\}$（半径 1 の円の境界と 内部）は凸集合（図 A.8 は $n = 2$ の場合）
- 2 次不等式で定められる n 次元空間の集合 $\{\boldsymbol{x} \in \mathbb{R}^n \mid x_1^2 + x_2^2 + \cdots + x_n^2 \geq 1\}$（半径 1 の円の境界と 外部）は 凸集合でない □

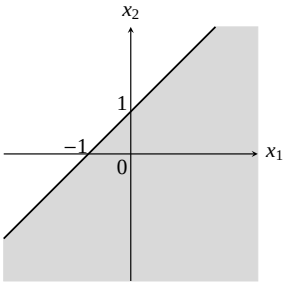

図 A.6: 凸集合の例 1

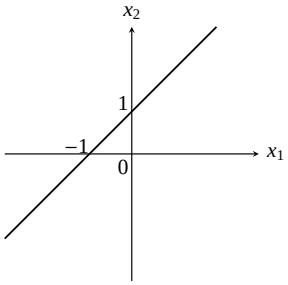

図 A.7: 凸集合の例 2

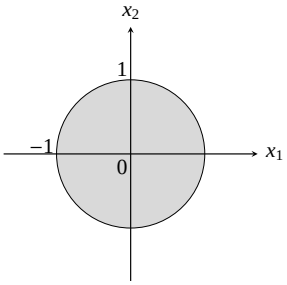

図 A.8: 凸集合の例 3

例 A.3.2 凸関数の例・凸関数でない例（図 A.9 も参照のこと）.

- $f(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{x} + b$ $(\boldsymbol{a} \in \mathbb{R}^n, b \in \mathbb{R})$ で定められる 1 次関数 $f : \mathbb{R}^n \to \mathbb{R}$ は凸関数
- $f(x) = x^2 - 2x$ で定められる非線形関数（2 次関数） $f : \mathbb{R} \to \mathbb{R}$ は凸関数
- $f(x) = e^{\alpha x}$ $(\alpha \in \mathbb{R})$ で定められる非線形関数 $f : \mathbb{R} \to \mathbb{R}$ は凸関数（図 A.9 は $\alpha = 1$ の場合）

- $f(\boldsymbol{x}) = x_1^2 + x_2^2 + \cdots + x_n^2$ で定められる非線形関数（2 次関数） $f : \mathbb{R}^n \to \mathbb{R}$ は凸関数
- $f(x) = x^3 - 1$ で定められる非線形関数（3 次関数） $f : \mathbb{R} \to \mathbb{R}$ は凸関数ではない
- $f(x) = \log(x^2 + 1)$ で定められる非線形関数 $f : \mathbb{R} \to \mathbb{R}$ は凸関数ではない　□

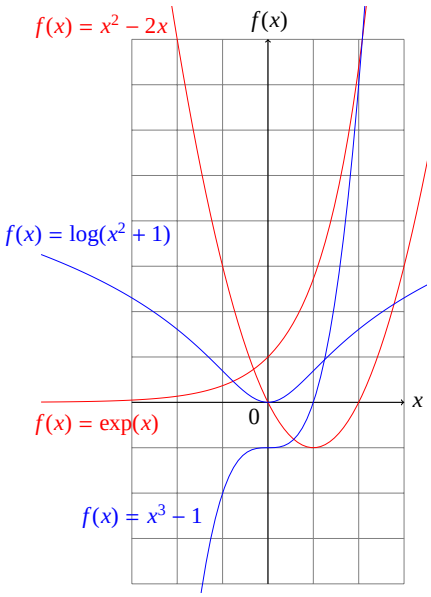

図 A.9: 凸関数・非凸関数の例

凸集合の性質

複数の凸集合については，次の定理が成立します．

定理 A.3.1 任意個の凸集合 S_i $(i \in I)$ の積集合 $\cap_{i \in I} S_i$ は凸集合である．

証明 $S = \cap_{i \in I} S_i$ とする．このとき，$\boldsymbol{x} \in S, \boldsymbol{y} \in S$ であるような $\boldsymbol{x}, \boldsymbol{y}$ は $\boldsymbol{x} \in S_i, \boldsymbol{y} \in S_i$ $(i \in I)$ である．さらに S_i $(i \in I)$ が凸集合であるから，

$$0 \leq \alpha \leq 1 \Rightarrow (1 - \alpha)\boldsymbol{x} + \alpha \boldsymbol{y} \in S_i \quad (i \in I)$$

である．よって $(1 - \alpha)\boldsymbol{x} + \alpha \boldsymbol{y} \in S$ であるので，S は凸集合である．　□

例 A.3.1 で凸集合の例として $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}^\top \boldsymbol{x} \leq b\}$, $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}^\top \boldsymbol{x} = b\}$ $(\boldsymbol{a} \in \mathbb{R}^n, b \in \mathbb{R})$ を挙げました．定理 A.3.1 で述べたように『凸集合の積集合は凸集合』ですから，集合

$$\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}_i^\top \boldsymbol{x} \leq b_i \ (i = 1, \ldots, m_i),\ \boldsymbol{a}_i^\top \boldsymbol{x} = b_i \ (i = m_i + 1, \ldots, m)\}$$

という集合も凸集合になります．

A.3.2 凸結合と凸包

n 次元空間上の k 個の点 $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k \in \mathbb{R}^n$ があるとき，$\lambda_i \geq 0$ $(i = 1, 2, \ldots, k)$, $\sum_{i=1}^{k} \lambda_i = 1$ となるような λ_i $(i = 1, 2, \ldots, k)$ を用いて

$$\boldsymbol{x} = \lambda_1 \boldsymbol{x}_1 + \lambda_2 \boldsymbol{x}_2 + \cdots + \lambda_k \boldsymbol{x}_k = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i$$

と書くことのできるベクトル $\boldsymbol{x}$ を $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k$ の**凸結合**といいます（図 ）.

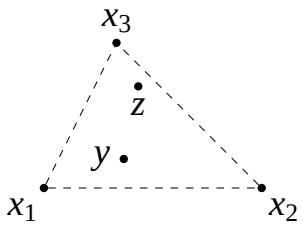

図 A.10: 凸結合の例 ($\boldsymbol{y} = 0.5\boldsymbol{x}_1 + 0.3\boldsymbol{x}_2 + 0.2\boldsymbol{x}_3, \boldsymbol{z} = 0.1\boldsymbol{x}_1 + 0.2\boldsymbol{x}_2 + 0.7\boldsymbol{x}_3$)

また，n 次元空間のある集合 S に対して，S を含む最小の凸集合を S の**凸包**といいます（図 A.12）. もちろん，S が凸集合であれば S の凸包は S そのものです．ある集合 S に対して，S の凸包は，元々 S に属する点の凸結合で構成することができます.

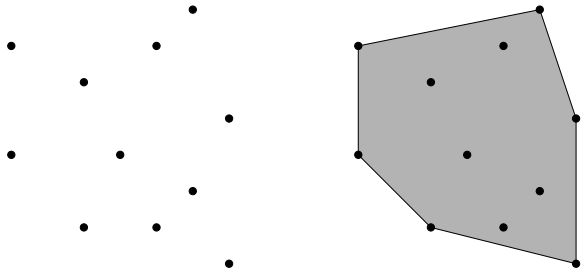

図 A.11: 凸包の例（S が点の集合の場合）

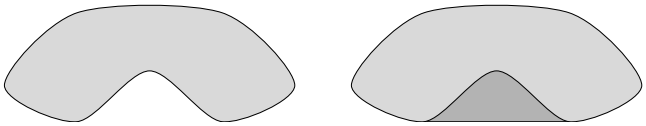

図 A.12: 凸包の例（S が大きさを持つ集合の場合）

A.3.3　半正定値行列と凸性

凸性と関係の深い行列として，**半正定値行列**があります．ある対称行列[3] $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ が半正定値行列であるとは，任意の $\boldsymbol{x} \in \mathbb{R}^n$ に対し $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \geq 0$ となることをいいます．これは，行列 $\boldsymbol{A}$ の全ての固有値が非負であることと同値です．

例 A.3.3 半正定値行列の例
次の対称行列 $\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3$ はいずれも半正定値行列です．

$$\boldsymbol{A}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \boldsymbol{A}_2 = \begin{bmatrix} 4 & -2 \\ -2 & 1 \end{bmatrix}, \quad \boldsymbol{A}_3 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 2 & -1 & 6 \end{bmatrix}$$

例えば，

$$\begin{aligned}
&\boldsymbol{x}^\top \boldsymbol{A}_1 \boldsymbol{x} = x_1^2 + x_2^2 \geq 0 \\
&\boldsymbol{x}^\top \boldsymbol{A}_2 \boldsymbol{x} = 4x_1^2 - 4x_1 x_2 + x_2^2 = (2x_1 - x_2)^2 \geq 0 \\
&\boldsymbol{x}^\top \boldsymbol{A}_3 \boldsymbol{x} = x_1^2 + x_2^2 + 6x_3^2 + 4x_1 x_3 - 2x_2 x_3 = (x_1 + 2x_3)^2 + (x_2 - x_3)^2 + x_3^2 \geq 0
\end{aligned}$$

ですので，$\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3$ が半正定値行列であることがわかります．また $\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3$ の特性方程式は

$$\begin{aligned}
&|\lambda \boldsymbol{I} - \boldsymbol{A}_1| = (\lambda - 1)^2 = 0 \\
&|\lambda \boldsymbol{I} - \boldsymbol{A}_2| = \lambda(\lambda - 5) = 0 \\
&|\lambda \boldsymbol{I} - \boldsymbol{A}_3| = (\lambda - 1)(\lambda^2 - 7\lambda + 1) = (\lambda - 1)\left(\lambda - \frac{7 + 3\sqrt{5}}{2}\right)\left(\lambda - \frac{7 - 3\sqrt{5}}{2}\right) = 0
\end{aligned}$$

ですので，$\boldsymbol{A}_1$ の固有値は 1（特性方程式の重根），$\boldsymbol{A}_2$ の固有値は $1, 3$，$\boldsymbol{A}_3$ の固有値は $1, (7 \pm 3\sqrt{5})/2$ となり，上で述べたようにいずれの固有値も非負であることが確認できます．また，$\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3$ は

$$\begin{aligned}
&\boldsymbol{A}_1 = \boldsymbol{L}_1 \boldsymbol{L}_1^\top, \quad \boldsymbol{L}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \\
&\boldsymbol{A}_2 = \boldsymbol{L}_2 \boldsymbol{L}_2^\top, \quad \boldsymbol{L}_2 = \begin{bmatrix} 2 & 0 \\ -1 & 0 \end{bmatrix}, \\
&\boldsymbol{A}_3 = \boldsymbol{L}_3 \boldsymbol{L}_3^\top, \quad \boldsymbol{L}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & -1 & 1 \end{bmatrix}
\end{aligned}$$

とすることができるので，$\boldsymbol{x}^\top \boldsymbol{A}_i \boldsymbol{x} = \|\boldsymbol{L}_i^\top \boldsymbol{x}\|^2 \geq 0 \ (i = 1, 2, 3)$ となり，やはり半正定値行列であることがわかります．　□

ここで見た半正定値行列と凸性は強いつながりがあります．それを示すのが以下の定理です．

[3] 非対称行列に対しても半正定値行列を定義することは可能ですが，最適化分野ではそのような行列を考えることはほとんどありませんので，ここでは対称行列に限定します．

定理 A.3.2 半正定値行列 $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ とベクトル $\boldsymbol{b} \in \mathbb{R}^n$ を用いて構成される関数

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{x}$$

は凸関数である．

証明 $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^n, 0 \leq \alpha \leq 1$ に対して，

$$\begin{aligned}(1-\alpha)f(\boldsymbol{x}) + \alpha f(\boldsymbol{y}) &= (1-\alpha)\left(\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{x}\right) + \alpha\left(\frac{1}{2}\boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{y} + \boldsymbol{b}^\top \boldsymbol{y}\right) \\ f((1-\alpha)\boldsymbol{x} + \alpha\boldsymbol{y}) &= \frac{1}{2}(1-\alpha)^2 \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} + \alpha(1-\alpha)\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{y} + \frac{1}{2}\alpha^2 \boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{y} \\ &\quad + (1-\alpha)\boldsymbol{b}^\top \boldsymbol{x} + \alpha \boldsymbol{b}^\top \boldsymbol{y}\end{aligned}$$

となるので，これより

$$(1-\alpha)f(\boldsymbol{x}) + \alpha f(\boldsymbol{y}) - f((1-\alpha)\boldsymbol{x} + \alpha\boldsymbol{y}) = \frac{1}{2}\alpha(1-\alpha)(\boldsymbol{x}-\boldsymbol{y})^\top \boldsymbol{A}(\boldsymbol{x}-\boldsymbol{y}) \geq 0$$

となる．よって $f(\boldsymbol{x})$ は凸関数である． □

このように，半正定値行列と凸性は深い関係があります．

A.4 グラフの基礎

最適化問題で扱う数理モデルには,「**グラフ**」と深く結びついたものがあります（なお，ここでいう「グラフ」とは，例えば折れ線グラフ・棒グラフなどの「グラフ」とは異なるものです）．ここでは，本書を理解する上で必要になるグラフの基礎的な知識，またいくつかの周辺知識について説明します．

A.4.1 グラフの定義

グラフとは,（有限個の）**節点**と，節点同士を結ぶ**枝**からなります．節点の集合を V とするとき，グラフ G は $G = (V, E)$ と書くことができます．ここで E は枝の集合で，$E \subseteq V \times V$ です．例えば図 A.13 は

$$\begin{aligned}V &= \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}\}, \\ E &= \{(v_1, v_3), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_4, v_{11}), (v_3, v_6), \\ &\qquad (v_6, v_7), (v_6, v_9), (v_6, v_{11}), (v_8, v_9), (v_9, v_{10}), (v_9, v_{11}), (v_{10}, v_{10})\}\end{aligned}$$

であるようなグラフの例です．

グラフの枝には向きのある場合とない場合があります．例えばグラフで水道管網を表す場合（水道管が枝になり，水道管の結節点が節点になります），水の流れには向きがありますので（高いところから低いところに流れる），この場合は枝に向きがあります．一方，一方通行のない道路網を考えると（この場合は道路が枝になり，交差点が節点になります），これは双方向に移動が可能ですから，枝には向きがありません．このように，向きのある枝のことを**有向枝**といい，枝が有向枝であるよう

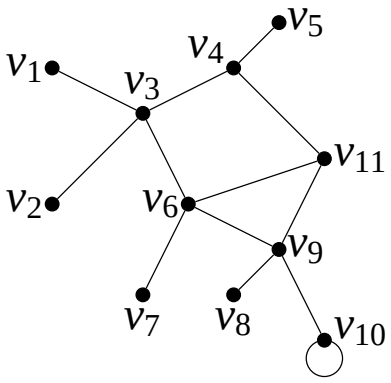

図 A.13: グラフの例

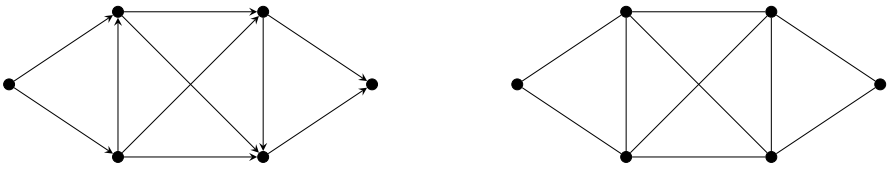

図 A.14: 有向グラフと無向グラフ

なグラフを**有向グラフ**といいます．これに対し，向きのない枝を**無向枝**といい，枝が無向枝であるようなグラフを**無向グラフ**といいます（図 A.14）[4]．

グラフ上では，ある節点から別の節点までの経路を考えることがよくあります．これを表現するためには，節点の列で表現することが可能です．これを**路**（または**道**）といいます．例えば，$(v_i, v_{i+1}) \in E$ $(i = 1, 2, \ldots, k-1)$ であるとき，

$$P:\ v_1, v_2, \ldots, v_k \tag{A.4.1}$$

とすれば，P は v_1 から v_k までの路を表していることになります（考えているグラフが有向グラフのときは枝の向きを守る必要があることに注意してください）．

表現 (A.4.1) で与えられる路 P において $v_1, v_2, \ldots, v_k$ が全て異なる時，P を**単純路**といいます．また $v_1 = v_k$ であるとき，P を**閉路**といい，始点と終点のみが同一でその他の節点が互いに異なる閉路を**単純閉路**といいます．さらに，ある節点から自分自身への枝を**自己閉路**といいます（図 A.13 で表されるグラフには自己閉路 (v_{10}, v_{10}) が見られます）．本書では，特に断らない限り，自己閉路のないグラフを考えることにします．

グラフを考える際には，節点の**次数**という概念が重要になることがあります．無向グラフの場合，ある節点の次数はその節点に接続している枝の本数を指します．また有向グラフの場合は，節点に入ってくる/節点から出ていく枝を区別し，入ってくる枝の本数を**入次数**，出て行く枝の本数を**出次数**といいます（図 A.15）．

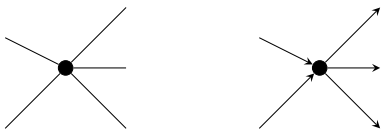

図 A.15: 節点の次数（左（無向グラフ）: 次数 5，右（有向グラフ）: 入次数 2・出次数 3）

グラフの節点や枝には何らかの値が与えられることがあります（複数与えられることもあります）．例えば上記の水道管の例であれば，水道管の長さや容量が枝に与えられる可能性がありますし，節点には入次数/出次数や流れ込んでいる水量が与えられるかもしれません．特に，枝に与えられる値が現実世界でのコスト（負荷）に相当するようなものの場合，これを枝の**重み**ということがあります．

A.4.2 特徴あるグラフ

ここでは，グラフの中でもある特徴を備えたグラフをいくつか紹介します．

木

木とは，グラフ上の任意の 2 点間に路が存在し（このようなグラフを**連結グラフ**といいます），かつ閉路を持たない場合をいいます（図 A.16）．

[4]英語では無向枝を edge，有向枝を arc ということが多いです．そのため，無向グラフは $G = (V, E)$，有向グラフは $G = (V, A)$ と書くことがよくあります．

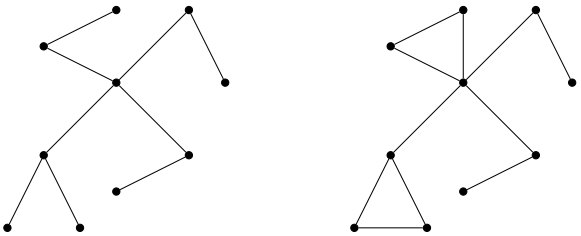

図 A.16: 木の例（左：木，右：木でない）

木を考える場合，節点が何らかの意味で優先順を持つ場合があります．例として生物の系統樹を図 A.17 に示します．実は図 A.17 は本来上下が逆なのですが，以下の説明のため，ここでは上下を逆転させて表示しています．この図は生物の進化の系統を表しており，図の最上部（本来の最下部）には「Radix（根）」という単語があり，最下部（本来の最上部）には「Plantae（植物界）」「Protista（原生生物）」「Animalia（動物界）」とあります．

このように，一番優先度の高い節点を最上部に書き，そこにつながる節点を順に下にぶら下げていく書き方をすることがあります．このとき，一番優先度の高い節点を**根**といい，最下方の節点（ぶらさげる節点のない節点）を**葉**といいます．まさに，図 A.17 の上下を反転させると（本来の向きに戻すと），これは木に他なりません．さらに，このような木の構造上に親子関係を見立て，ある節点 v に対し，v から見て根に近い方の節点 u を v の**親**，v から見て根から離れる方の節点 w を v の**子**といいます（図 A.18）．

二部グラフ

グラフ $G = (V, E)$ が**二部グラフ**であるとは，ある節点集合 V_1, V_2 が存在し，$V_1 \cup V_2 = V, V_1 \cap V_2 = \emptyset$ であり，全ての枝 $(v_i, v_j) \in E$ において端点の一方が V_1 に，他方が V_2 に属することをいいます（図 A.19）．

完全グラフ

（自己閉路の存在しない）無向グラフ $G = (V, E)$ において，異なる 2 点間を結ぶ全ての枝が存在するとき，G を**完全グラフ**といいます（図 A.20）．節点数が n（すなわち $|V| = n$）の完全グラフを K_n と書きます．K_3 は三角形になりますね．

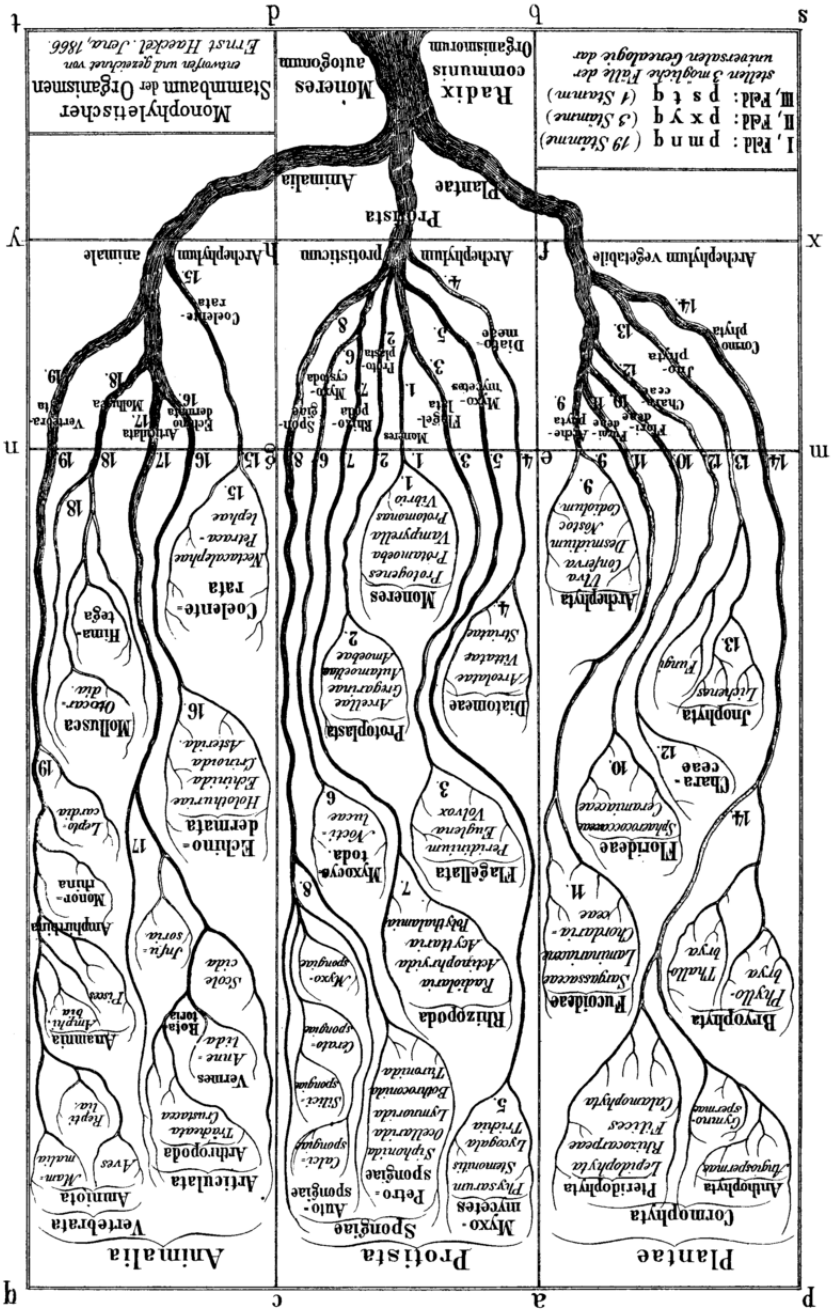

図 A.17: E. Haeckel の生物の系統樹
Wikipedia より引用 (http://ja.wikipedia.org/wiki/系統樹)

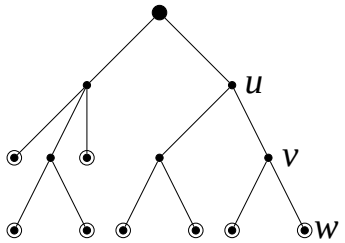

図 A.18: 根を一番上に書いた木の例（二重丸は葉を表しています）

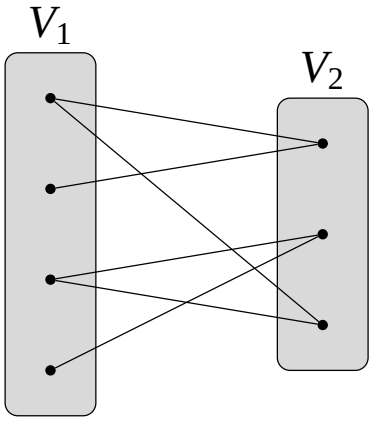

図 A.19: 二部グラフの例

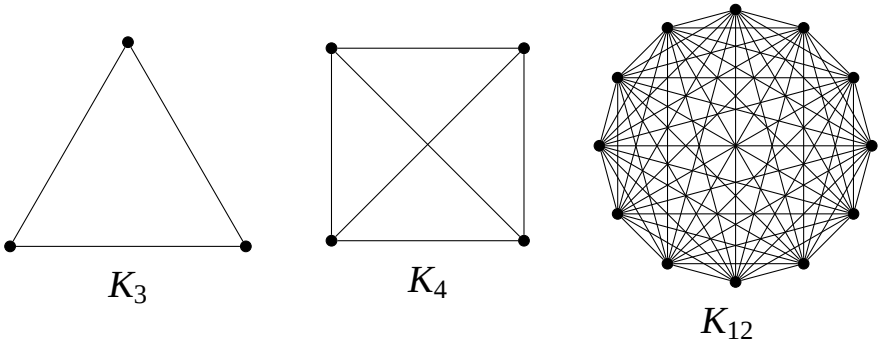

図 A.20: 完全グラフの例

付 録B　最適化問題の分類 (Advanced)

2.2.5 節では，多くのソルバで扱うことのできる問題クラスや，本書を読んでいく上で重要な問題クラスを対象に分類を行いました．本付録では，2.2.5 節では扱わなかったが，重要度の高い問題クラスを紹介したいと思います．知っておくと役に立つ場面があるでしょう．

2 次制約 2 次計画問題

目的関数が 2 次関数で，制約条件が 2 次関数を用いた不等式制約と線形の等式制約で表現されるとき，これを 2 **次制約 2 次計画問題** (quadratic constrained quadratic programming problem, QCQP) といいます．標準形は以下の通りです．

$$\left|\begin{array}{ll} \text{minimize} & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{U}_i\boldsymbol{x} + \boldsymbol{v}_i^\top \boldsymbol{x} + w_i \leq 0 \quad (i = 1, 2, \ldots, m) \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \end{array}\right. \tag{B.0.1}$$

ここで，$\boldsymbol{Q} \in \mathbb{R}^{n\times n}, \boldsymbol{c} \in \mathbb{R}^n, \boldsymbol{U}_i \in \mathbb{R}^{n\times n}, \boldsymbol{v} \in \mathbb{R}^n, w_i \in \mathbb{R}$, $\boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{b} \in \mathbb{R}^m$ です．先ほどの「2 次計画問題」と「凸 2 次計画問題」の関係と同様，単に「2 次制約 2 次計画問題」という場合でも，「凸 2 次制約凸 2 次計画問題」，すなわち $\boldsymbol{Q}, \boldsymbol{U}_i$ $(i = 1, \ldots, m)$ が半正定値行列である場合を指すことが多くあります．

錐線形計画問題

目的関数が線形関数であり，実行可能領域が線形関数と**凸錐** [1] で定められるような最適化問題は，応用領域が広範にわたることと，効率的な解法が構築可能であることから，近年広く研究が行われています．これらを**錐線形計画問題** (cone linear programming problem) といいます．その中でも，2 **次錐計画問題** (second-order cone programming problem, SOCP) や**半正定値計画問題** (semidefinite programming problem, SDP) は近年研究が大きく進んだ問題クラスです．

2 次錐計画問題には，(その名の通り) 2 次錐と呼ばれる錐が現れます．n 次元の 2 次錐 $\mathcal{Q}_n$ は次のように書くことができます．

$$\mathcal{Q}_n = \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; x_1 \geq \sqrt{x_2^2 + \cdots + x_n^2} \right\}$$

これを用いると，2 次錐計画問題の標準形は次のようになります．

$$\left|\begin{array}{ll} \text{minimize} & \sum_{i=1}^r \boldsymbol{c}_i^\top \boldsymbol{x}_i \\ \text{subject to} & \sum_{i=1}^r \boldsymbol{A}_i\boldsymbol{x}_i = \boldsymbol{b} \\ & \boldsymbol{x}_i \in \mathcal{Q}_{n_i} \quad (i = 1, \ldots, r) \end{array}\right. \tag{B.0.2}$$

[1] **錐** とは，$\boldsymbol{x} \in C, \alpha \in [0, \infty) \Rightarrow \alpha\boldsymbol{x} \in C$ となるような集合 C のことをいいます．また特に C が凸であるとき，凸錐といいます．

ここで $\boldsymbol{c}_i \in \mathbb{R}^{n_i+1}, \boldsymbol{A} \in \mathbb{R}^{m\times(n_i+1)}, \boldsymbol{b} \in \mathbb{R}^m$ です.

また，半正定値計画問題の標準形は以下の通りです [2].

$$\left|\begin{array}{ll} \text{minimize} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & \boldsymbol{F}_0 + \sum_{i=1}^n x_i \boldsymbol{F}_i \succeq \boldsymbol{O} \end{array}\right. \tag{B.0.3}$$

ここで，$\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{F}_i \in \mathbb{R}^{m\times m}$ $(i = 0, 1, \ldots, n)$ であり，記号 $\boldsymbol{A} \succeq \boldsymbol{O}$ は行列 $\boldsymbol{A}$ が半正定値行列であることを表します.

錐線形計画問題の共通の特徴として，内点法という効率的な解法を自然に構成できる，という点が挙げられます.

数理計画問題の分類と関係

最適化問題の一般的な形は (2.2.1) で与えました．表 B.1 は，(2.2.1) における f や S の数学的性質と問題クラスの関係をまとめたものです．また，既出ですが，問題クラスの略称・正式名称を 表 B.2 に示します.

表 B.1: f と S の性質による問題クラスの分類

<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="3">f</th></tr>
<tr><th>線形</th><th>(凸) 2 次</th><th>非線形</th></tr>
<tr><td rowspan="5">S</td><td>線形</td><td>LP</td><td>QP</td><td rowspan="5">NLP</td></tr>
<tr><td>(凸) 2 次</td><td colspan="2">QCQP</td></tr>
<tr><td>2 次錐</td><td>SOCP</td><td rowspan="3"></td></tr>
<tr><td>半正定値</td><td>SDP</td></tr>
<tr><td>非線形</td><td></td></tr>
</table>

表 B.2: 問題クラスの略称と正式名称

LP	Linear Programming
QP	Quadratic Programming
QCQP	Quadratic Constrained Quadratic Programming
SOCP	Second-order Cone Programming
SDP	Semidefinite Programming
NLP	Nonlinear Programming

これらの問題クラスに対し，

$$\text{LP} \subset \text{QP} \subset \text{QCQP} \subset \text{SOCP} \subset \text{SDP} \subset \text{NLP}$$

[2] 一般に，最適化問題に対して双対問題と呼ばれる問題を考えることができます（詳細は [11] などを参照してください）. この問題に対する双対問題を標準形とする場合もあります.

という関係が成立します（ただし QCQP は目的・制約とも凸であるものとします）．このうち，表 B.1 からは QCQP $\subset$ SOCP という関係は自明ではありませんが，巧妙な変数変換を行うことで QCQP を SOCP として表現することができます [2].

付 録C　サンプルデータ

ここでは，本文中で出てきた問題で用いたデータを掲載します．なお，データの表記方法は 4 章で説明した方法（モデリング AMPL での表記方法）に従います（※ただ，4 章を詳細に読まずとも，直感的に理解できる部分は少なくないと思います）．

C.1　集合分割問題のデータ

問題 6.1.1 で用いたデータは以下の通りです．

```
set I := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
         21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
         41 42 43 44 45 46 47 48 49 50 51 52 53 54 55;
set J := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20;

param a : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 :=
 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 2 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 3 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 4 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 5 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 6 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1
 7 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0
 8 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
 9 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0
10 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
11 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0
12 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
13 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
14 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
15 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
16 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0
17 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
18 1 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0
19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1
20 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 1
21 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0
22 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
23 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0
25 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0
26 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0
27 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1
28 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0
29 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1
30 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0
```

```
31 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
32 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0
33 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
34 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0
35 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0
36 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
37 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0
38 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
39 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0
40 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
41 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0
42 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
43 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0
44 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0
45 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
46 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
47 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
48 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
49 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
50 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0
51 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
52 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0
53 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 1
54 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
55 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0
;
```

C.2　集合パッキング問題

問題 6.2.1 で用いたデータは以下の通りです.

```
set I := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
         21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42;
set J := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20;

param a : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 :=
 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0
 2 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 3 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0
 4 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 5 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 6 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0
 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0
 8 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
 9 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
11 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0
12 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0
13 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0
14 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0
15 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
16 0 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0
17 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
```

```
18 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0
19 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0
21 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
22 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0
23 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0
24 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
25 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
26 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
27 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
28 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
29 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0
30 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0
31 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0
32 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0
33 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0
34 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
35 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0
36 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1
37 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1
38 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1
39 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0
40 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
41 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
42 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1
;
```

C.3 集合被覆問題のデータ

問題 6.3.1 で用いたデータのうち，`set I`, `set J`, `param a` に関するデータは，C.2 節で紹介したものと同じものを用いています．

さらに，問題 6.3.1 では以下のデータも用いています．

```
param c :=
 1 1.3  2 1.3  3 1.3  4 1.3  5 1.3
 6 2.2  7 2.2  8 2.2  9 2.2 10 2.2
11 1.9 12 1.9 13 1.9 14 1.9 15 1.9
16 3.1 17 3.1 18 3.1 19 3.1 20 3.1
21 2.4 22 2.4 23 2.4 24 2.4 25 2.4
26 1.8 27 1.8 28 1.8 29 1.8 30 1.8
31 3.2 32 3.2 33 3.2 34 3.2 35 3.2
36 2.7 37 2.7 38 2.7 39 2.7 40 2.7
41 2.8 42 2.8
;
```

C.4 ナップサック問題のデータ

問題 6.4.1 で用いたデータは以下の通りです．

```
set I := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
         21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

```
         41 42 43 44 45 46 47 48 49 50;

param b := 50;

param c :=
 1  1  2 10  3 5  4 1  5 7
 6  6  7 10  8 6  9 9 10 6
11  3 12  6 13 7 14 6 15 6
16 10 17  7 18 7 19 9 20 10
21 10 22  3 23 9 24 4 25 8
26  6 27  7 28 3 29 7 30 3
31  2 32  5 33 4 34 1 35 7
36  2 37  7 38 2 39 4 40 3
41  9 42  4 43 8 44 1 45 8
46  8 47  9 48 8 49 4 50 2
;

param a :=
 1 9  2  3  3  9  4 8  5 2
 6 6  7  5  8  9  9 4 10 5
11 7 12  7 13  1 14 8 15 5
16 9 17  4 18  2 19 8 20 5
21 4 22  8 23  2 24 7 25 10
26 2 27 10 28  2 29 3 30 9
31 9 32  8 33  5 34 6 35 5
36 4 37  4 38 10 39 8 40 9
41 2 42 10 43  4 44 8 45 4
46 4 47 10 48  5 49 9 50 4
;
```

C.5　ビンパッキング問題のデータ

問題 6.5.1 で用いたデータは以下の通りです.

```
set I := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
         21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
         41 42 43 44 45 46 47 48 49 50;
set J := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
         21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
         41 42 43 44 45 46 47 48 49 50;

param b := 50;

param a :=
 1 11  2 22  3 15  4 18  5 33
 6 31  7  8  8 28  9 32 10 10
11 31 12 16 13 24 14 30 15 10
16  6 17 16 18  6 19  7 20  5
21 17 22 25 23 33 24 34 25 17
26 16 27 31 28 18 29 33 30 27
31 29 32 23 33  9 34 23 35 12
36 15 37 21 38  7 39 30 40 29
41  7 42 29 43 18 44 29 45 31
```

```
46 18 47 11 48 29 49 17 50  9
;
```

C.6 最大流問題のデータ

問題 6.6.1 で用いたデータは以下の通りです.

```
set V := 0 1 2 3 4 5 6 7 8 9 10;
set E := (0, 1), (0, 2), (0, 3),
         (1, 4), (1, 5), (2, 4), (2, 5), (2, 6), (3, 5), (3, 6),
         (4, 7), (4, 8), (5, 7), (5, 8), (5, 9), (6, 8), (6, 9),
         (7, 10), (8, 10), (9, 10);

param s := 0;
param t := 10;

param c :=
[0, 1] 50 [0, 2] 50 [0, 3] 50
[1, 4] 18 [1, 5] 12 [2, 4] 20 [2, 5] 15 [2, 6] 17 [3, 5] 10 [3, 6] 11
[4, 7] 23 [4, 8] 11 [5, 7] 17 [5, 8] 19 [5, 9] 25 [6, 8] 16 [6, 9] 23
[7, 10] 50 [8, 10] 50 [9, 10] 50
;
```

C.7 最短路問題のデータ

問題 6.7.1 で用いたデータは以下の通りです.

```
set V := 0 1 2 3 4 5 6 7 8 9 10;
set A := (0, 1), (0, 2), (0, 3),
         (1, 4), (1, 5), (2, 4), (2, 5), (2, 6), (3, 5), (3, 6),
         (4, 7), (4, 8), (5, 7), (5, 8), (5, 9), (6, 8), (6, 9),
         (7, 10), (8, 10), (9, 10);

param s := 0;
param t := 10;

param c :=
[0, 1] 19 [0, 2] 15 [0, 3] 21
[1, 4] 18 [1, 5] 12 [2, 4] 20 [2, 5] 15 [2, 6] 17 [3, 5] 10 [3, 6] 11
[4, 7] 23 [4, 8] 11 [5, 7] 17 [5, 8] 19 [5, 9] 25 [6, 8] 16 [6, 9] 23
[7, 10] 18 [8, 10] 25 [9, 10] 17;

end;
```

C.8 Hitchcock 型輸送問題のデータ

問題 6.8.1 で用いたデータは以下の通りです.

```
set I := A B C;
set J := 1 2 3 4;

param a :=
A 60
B 45
C 50
;

param b :=
1 35
2 15
3 35
4 45
;

param c : 1 2 3 4 :=
A 4 1 3 3
B 9 2 7 10
C 8 3 10 9
;

end;
```

C.9　割当問題のデータ

問題 6.9.1 で用いたデータは以下の通りです.

```
set I := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25;
set J := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25;

param t : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 :=
 1 25 32 51 86 80 31 84 42 22 16 29 50 46 84 88 46 31 14 73 10 20 29 62 94 24
 2 18 94 40 68 97 59 19 94 24 13 42 66 80 76 17 56 84 17 92 92 75 23 12 56 29
 3 33 90 29 59 52 80 88 35 48 22 13 95 47 17 73 10 86 90 52 16 87 59 21 88 11
 4 88 89 94 70 80 86 53 46 41 68 36 46 69 69 89 27 59 89 64 30 23 12 10 31 24
 5 66 67 15 29 29 72 69 53 97 14 30 64 97 85 97 70 15 40 66 92 23 83 76 98 81
 6 19 81 81 45 19 11 14 48 87 17 68 66 92 99 72 86 49 33 74 12 16 87 14 75 89
 7 15 39 54 74 74 59 87 16 20 12 38 62 30 17 24 91 46 23 63 44 55 40 84 43 68
 8 27 14 14 41 68 85 76 31 61 53 13 88 22 84 96 13 28 23 51 23 93 50 38 12 23
 9 15 36 22 69 64 49 54 11 22 28 13 84 16 23 53 64 17 80 62 76 63 49 96 81 54
10 86 33 50 33 97 96 66 48 12 39 16 68 42 94 35 62 36 53 28 65 59 35 12 90 85
11 79 82 85 15 43 94 32 40 51 21 18 14 25 50 18 74 80 81 18 39 46 71 32 55 70
12 87 12 63 15 75 73 40 72 49 34 52 88 70 90 55 23 71 15 13 78 97 15 86 53 83
13 38 87 52 99 78 46 65 81 29 28 96 18 22 37 42 32 16 76 44 74 94 18 54 77 55
14 93 21 26 17 49 85 12 48 89 39 21 26 28 15 25 11 92 70 34 12 21 90 18 59 77
15 36 44 10 52 78 12 96 42 45 49 78 49 71 15 50 58 66 91 95 12 42 11 43 91 35
16 65 14 75 66 85 20 84 17 47 88 39 27 47 11 70 63 44 45 21 75 72 13 46 88 13
17 78 77 85 20 91 85 56 78 79 16 17 17 21 10 84 39 74 99 13 74 28 84 31 18 54
18 23 43 43 27 92 10 50 86 35 13 51 99 75 25 19 91 18 63 95 10 59 21 50 77 75
19 19 99 54 12 68 19 94 13 57 69 30 91 17 62 63 98 64 97 44 18 19 95 83 39 61
20 16 59 71 15 41 12 74 23 50 62 73 70 12 50 66 92 67 27 93 20 16 41 50 89 48
21 10 81 29 19 56 53 91 91 63 89 29 26 47 76 43 27 69 74 24 55 90 67 13 40 32
```

```
22 17 80 84 98 12 10 15 61 24 69 71 95 20 21 98 53 38 93 32 69 98 34 18 90 39
23 18 51 11 41 57 57 22 12 19 49 52 92 88 89 41 84 57 29 72 14 69 31 26 79 36
24 21 48 34 10 28 76 90 13 17 56 13 18 28 13 26 91 27 99 89 37 31 28 13 57 58
25 17 90 30 89 86 20 30 37 21 16 63 14 96 44 33 70 50 13 31 28 25 71 15 33 15
;
```

C.10 巡回セールスマン問題のデータ

問題 3.1.1，ならびに 6.10 節で用いたデータは以下の通りです．

```
set I := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20;
set J := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20;

param n := 20;

param d : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 :=
 1      0 49.74 48.38 82.49 52.63 11.18 88.51 76.56 13.04 13.89
    47.42 86.31 26.25 64.62 30.15 78.26 30.59 60.80 64.78 43.42
 2  49.74     0 35.00 77.16  8.25 56.65 50.16 86.61 43.68 63.63
     5.00 52.15 35.00 39.82 76.24 77.16 37.34 25.00 65.80 14.04
 3  48.38 35.00     0 43.27 29.07 48.02 43.19 51.88 35.85 59.48
    38.21 39.20 22.14 16.64 62.24 42.45 18.36 21.21 30.81 44.20
 4  82.49 77.16 43.27     0 70.01 77.01 56.44 25.06 69.72 89.05
    80.92 48.26 59.48 39.66 81.74  7.07 54.12 55.80 18.03 87.42
 5  52.63  8.25 29.07 70.01     0 58.18 42.43 80.95 44.72 66.40
    13.15 44.05 33.96 31.91 77.08 70.43 35.13 16.76 59.68 22.02
 6  11.18 56.65 48.02 77.01 58.18     0 90.27 68.31 13.60 12.17
    55.08 87.09 26.91 64.66 20.10 72.11 29.68 63.66 59.00 52.35
 7  88.51 50.16 43.19 56.44 42.43 90.27     0 78.16 77.20 101.02
    55.15 8.25 63.41 27.46 105.42 61.03 60.88 28.23 58.05 64.20
 8  76.56 86.61 51.88 25.06 80.95 68.31 78.16     0 65.37 79.31
    89.31 70.35 59.55 55.90 67.01 18.38 54.49 70.21 21.84 93.86
 9  13.04 43.68 35.85 69.72 44.72 13.60 77.20 65.37     0 23.85
    42.58 74.43 13.89 52.33 32.57 65.73 17.72 50.21 52.17 41.05
10  13.89 63.63 59.48 89.05 66.40 12.17 101.02 79.31 23.85    0
    61.27 98.27 37.74 76.06 20.59 84.02 41.15 73.76 71.03 56.92
11  47.42  5.00 38.21 80.92 13.15 55.08 55.15 89.31 42.58 61.27
        0 57.14 35.36 44.28 74.95 80.61 38.33 29.83 68.88  9.06
12  86.31 52.15 39.20 48.26 44.05 87.09  8.25 70.35 74.43 98.27
    57.14     0 60.54 22.67 101.18 53.00 57.43 28.16 50.70 66.07
13  26.25 35.00 22.14 59.48 33.96 26.91 63.41 59.55 13.89 37.74
    35.36 60.54     0 38.48 43.91 56.46  5.39 36.88 42.95 36.77
14  64.62 39.82 16.64 39.66 31.91 64.66 27.46 55.90 52.33 76.06
    44.28 22.67 38.48     0 78.52 41.63 34.99 16.28 34.23 52.20
15  30.15 76.24 62.24 81.74 77.08 20.10 105.42 67.01 32.57 20.59
    74.95 101.18 43.91 78.52    0 75.71 45.00 80.45 64.20 72.44
16  78.26 77.16 42.45  7.07 70.43 72.11 61.03 18.38 65.73 84.02
    80.61 53.00 56.46 41.63 75.71     0 51.08 57.31 13.60 86.56
17  30.59 37.34 18.36 54.12 35.13 29.68 60.88 54.49 17.72 41.15
    38.33 57.43  5.39 34.99 45.00 51.08     0 35.51 37.58 40.71
18  60.80 25.00 21.21 55.80 16.76 63.66 28.23 70.21 50.21 73.76
    29.83 28.16 36.88 16.28 80.45 57.31 35.51     0 48.38 38.47
19  64.78 65.80 30.81 18.03 59.68 59.00 58.05 21.84 52.17 71.03
```

```
    68.88 50.70 42.95 34.23 64.20 13.60 37.58 48.38     0 74.25
20  43.42 14.04 44.20 87.42 22.02 52.35 64.20 93.86 41.05 56.92
     9.06 66.07 36.77 52.20 72.44 86.56 40.71 38.47 74.25     0
;
```

なお，都市 $1,\ldots,20$ は x-y 平面上にあるものとし，都市間の距離はユークリッド距離で与えてあります．都市 $1,\ldots,20$ の x,y 座標は以下の通りです．

```
    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
x  57 14 42 66 16 67 10 88 56 69 14 18 49 33 87 71 51 21 67 15
y  91 66 45  9 58 86 16 21 78 98 71 14 66 31 88 14 61 42 27 80
```

また，7.3 節で P_0,P_1,P_2,P_3,P_4 を逐次に解くことになります．このうち，P_0,P_1 を解くときは上記の内容をそのまま用います．P_2,P_3,P_4 を解くときは，上記の内容に次の内容を加えたものを用います．

- P_2 用追加データ

```
set K := 1 2 3 4 5;

set V[1] := 1 10 15 6;
set V[2] := 2 11 20 5;
set V[3] := 3 18 7 12 14;
set V[4] := 4 19 8 16;
set V[5] := 9 17 13;

param p := [1] 4 [2] 4 [3] 5 [4] 4 [5] 3;
```

- P_3 用追加データ

```
set K := 1 2 3 4 5 6 7 8;

set V[1] := 1 10 15 6;
set V[2] := 2 11 20 5;
set V[3] := 3 18 7 12 14;
set V[4] := 4 19 8 16;
set V[5] := 9 17 13;
set V[6] := 1 10 15 6 9;
set V[7] := 2 5 18 7 12 14 19 3 17 13 20 11;
set V[8] := 4 16 8;

param p := [1] 4 [2] 4 [3] 5 [4] 4 [5] 3 [6] 5 [7] 12 [8] 3;
```

- P_4 用追加データ

```
set K := 1 2 3 4 5 6 7 8 9 10;

set V[1] := 1 10 15 6;
set V[2] := 2 11 20 5;
set V[3] := 3 18 7 12 14;
set V[4] := 4 19 8 16;
set V[5] := 9 17 13;
set V[6] := 1 10 15 6 9;
```

```
set V[7] := 2 5 18 7 12 14 19 3 17 13 20 11;
set V[8] := 4 16 8;
set V[9] := 1 20 11 2 5 18 7 12 14 4 16 8 19 3 17 13 9;
set V[10] := 6 15 10;

param p := [1] 4 [2] 4 [3] 5 [4] 4 [5] 3 [6] 5 [7] 12 [8] 3 [9] 17 [10] 3;
```

付 録D　ソルバの入手方法・使い方

ここでは，各ソルバの入手方法や使い方を説明します．まず，この本で利用しているソルバ GLPK [27] の入手方法や使い方を紹介し，それ以外のソルバについても簡単に説明します．

D.1　GLPK の入手方法・使い方

D.1.1　入手方法

1 章でも触れたように，GLPK はフリーのソフトウェアですので，無償で入手・利用することができます[1]．

Windows 環境

Windows 環境で利用できる GLPK を入手するには，GUSEK [32] をダウンロードするのが最も簡単だと思われます．（この本では利用していませんが,）GUSEK は GPLK を GUI から利用するためのソフトウェアです．

GUSEK は，GUSEK の HP: `http://gusek.sourceforge.net/gusek.html` にアクセスし，ここにある “Download and Install” から，`gusek_*-*-**.zip` をダウンロードすることができます（`*-*-**` には GUSEK のバージョン番号が入ります）．これを展開すると `glpsol.exe` が含まれているはずです．これが GLPK の実行形式ファイルになります．（もちろん，GUSEK 本体 `gusek.exe` も含まれています．興味のある方は是非試してみてください．）

また，ソースファイルをダウンロードし，これをコンパイルすることで実行形式ファイルを作成することも可能です．この方法であれば，GUSEK の更新ペースに関わらず，常に最新の GLPK を入手・利用することができます．

そのためには，まず，コンパイラとして C/C++ が使えるように設定された Microsoft Visual Studio がインストールされている必要があります．最新版は
`https://www.visualstudio.com/ja-jp/downloads/download-visual-studio-vs.aspx`
から入手・インストールすることが可能です．

GLPK のソースファイルを入手するためには，`http://ftp.gnu.org/gnu/glpk/` または `ftp://ftp.gnu.org/gnu/glpk/` にアクセスします（GLPK の HP `https://www.gnu.org/software/glpk/` 内にリンクが設定されています）．ここから，`glpk-*.**.tar.gz` をダウンロードします（`*.**` の部分にはバージョン番号が入ります）．特に理由のない限り，最新のものをダウンロードすればよいでしょう．

ダウンロードした `glpk-*.**.tar.gz` を展開したのち，32 ビットマシンなら `w32` というフォルダを，64 ビットマシンなら `w64` というフォルダを開きます（最近のマシンであればほとんどが 64

[1] ここで紹介する入手方法は 2016 年 9 月 14 日の情報に基づくものです．

ビットマシンのはずです）．この中にある Build_GLPK_with_VC**.bat をダブルクリックすると，コンパイル（ビルド）が行われ，同じフォルダ内に glpsol.exe が生成されるはずです．これが実行形式ファイルになります．ここで，Build_GLPK_with_VC**.bat の ** の部分には Microsoft Visual Studio の内部バージョン番号が入ります（製品名とバージョン番号の対応関係は表 D.1 を参照してください）．

表 D.1: Microsoft Visual Studio の製品名と内部バージョン番号の対応関係

製品名	バージョン番号
Visual Studio 2008	9
Visual Studio 2010	10
Visual Studio 2013	12
Visual Studio 2015	14

Mac/Unix/Linux 環境

Mac/Unix/Linux 環境で利用できる GLPK ですが，こちらは Windows 環境のように実行形式を直接入手するのではなく，自分でビルドするのが手っ取り早いようです[2]．

ビルドのためには，コンパイラとして gcc がインストールされている必要があります．

ソースファイルの入手方法は上記 Windows 環境での方法と全く同じです．glpk-*.**.tar.gz を入手したら，

```
gzip -cd glpk-*.**.tar.gz | tar xvf -
```

などとして展開すると，ディレクトリ（フォルダ）glpk-*.** が得られると思います．その後，

```
cd glpk-4.60
./configure
make
```

とすると，GLPK がビルドされます．ビルド終了後に

```
make check
```

とすると，正しくビルドができたかどうかを確認してくれます．正しくビルドされていれば，次のような表示が得られるはずです．

```
Making check in src
make[1]: Nothing to be done for ‘check’.
Making check in examples
./glpsol --version
GLPSOL: GLPK LP/MIP Solver, v*.**
```

[2]一部，コンパイルしたファイルを配布しているサイト等もあるようですが，Windows 環境の GUSEK のようなプロジェクトで運用されているかどうかを確認することができませんでした．

```
Copyright (C) 2000-2015 Andrew Makhorin, Department for Applied
Informatics, Moscow Aviation Institute, Moscow, Russia. All rights
reserved. E-mail: <mao@gnu.org>.

This program has ABSOLUTELY NO WARRANTY.

This program is free software; you may re-distribute it under the terms
of the GNU General Public License version 3 or later.
./glpsol --mps ./murtagh.mps --max
GLPSOL: GLPK LP/MIP Solver, v*.**
Parameter(s) specified in the command line:
 --mps ./murtagh.mps --max
Reading problem data from './murtagh.mps'...
Problem: OILREFI
Objective: PROFIT
74 rows, 81 columns, 504 non-zeros
600 records were read
One free row was removed
GLPK Simplex Optimizer, v*.**
73 rows, 81 columns, 474 non-zeros
Preprocessing...
67 rows, 72 columns, 419 non-zeros
Scaling...
 A: min|aij| =  3.000e-03  max|aij| =  6.600e+01  ratio =  2.200e+04
GM: min|aij| =  2.076e-01  max|aij| =  4.817e+00  ratio =  2.320e+01
EQ: min|aij| =  4.329e-02  max|aij| =  1.000e+00  ratio =  2.310e+01
Constructing initial basis...
Size of triangular part is 66
*     0: obj =  -0.000000000e+00 inf =   0.000e+00 (26)
*    43: obj =   1.260571241e+02 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.2 Mb (189297 bytes)
```

ビルドした GLPK を利用するためには，

```
make install
```

とすると，利用中の環境にインストールされます（通常は管理者権限でインストールする必要があります）．また，ディレクトリ（フォルダ）`glpk-*.**/examples/glpsol` への絶対パスを環境変数 `PATH` に追加するなどしても使えるようになります．詳しくはソースファイルに付属の `INSTALL` ファイルを読んで下さい．

D.1.2 使い方

前節で見たように，GLPK の実行ファイルは `glpsol` という名前です．Windows 環境ならコマンドプロンプト，Mac/Unix/Linux 環境であればターミナル（コンソール）にて

```
glpsol.exe    # Windows 環境の場合
glpsol        # Mac/Unix/Linux 環境の場合
```

などとしてみてください（Windows 環境での \ は半角の「¥」です）．もし

```
No input problem file specified; try glpsol --help
```

と表示されれば，コンピュータに GLPK がインストールされ，利用できる状態になっています．もし，利用しているコンピュータに glpsol(.exe) があるにも関わらず GLPK が利用できないようであれば，

- 環境変数 PATH に glpsol(.exe) があるフォルダ（ディレクトリ）への絶対パスを登録する
- コマンドプロンプト（ターミナル・コンソール）で glpsol(.exe) があるフォルダ（ディレクトリ）へ移動して GLPK を利用する
- 利用したいフォルダへ glpsol(.exe) をコピーする
  - Mac/Unix/Linux 環境の場合，glpsol のみではなく，付随して作成されているライブラリ等も合わせてコピーする必要があります（のでおすすめしません）

などの方法があります．

GLPK が利用できるようであれば，glpsol(.exe) --help を試してみると，たくさんの表示がされます．これは，glpk のオプションを表示しています．必要なときにはこれを参照するようにしましょう．ここでは，基本的なオプションだけを見ておきます．

- -m: モデルファイルを指定
- -d: データファイルを指定（データファイルがある場合）
- -o: 解ファイルの出力先を指定
- --log: ログファイルの出力先を指定

例えば，モデルファイル aaa.mod，データファイル aaa.dat があるフォルダ（ディレクトリ）にて最適化計算を実行し，同じフォルダ（ディレクトリ）に解ファイル aaa.sol，ログファイル aaa.log を出力するためには

```
glpsol.exe -m aaa.mod -d aaa.dat -o aaa.sol --log aaa.log    # Windows
glpsol -m aaa.mod -d aaa.dat -o aaa.sol --log aaa.log        # Mac/Unix/Linux
```

とすれば実行できます．

D.2　GLPK 以外のソルバの紹介

ここでは，GLPK 以外のソルバをいくつか紹介します．

ソルバ間の比較をしているサイトとして，Arizona State University の Hans D. Mittelmann 教授の “Benchmarks for Optimization Software” というサイトが有名です [24]．このサイトでは，例えば [35] などのベンチマーク問題を各ソルバで解いた結果（最適解を得られたかどうか，得られていればどの程度の時間を要したのか）を紹介しています．

問題クラスによって評価されているソルバは異なりますが，例えば次のようなソフトが利用されています（アルファベット順）:

- FICO Xpress Optimization Suite [25]（商用）

- Gurobi [31]（商用）
- IBM CPLEX Optimizer [33]（アカデミックフリー）
- lp_solve [34]（フリー）
- SCIP [37]（アカデミックフリー）

詳細についてはそれぞれのソルバの HP を参照してください.
また，日本発のソフトとして

- Numerical Optimizer [36]（商用）

を挙げておきます.

付 録E　改訂予定

- 全編に渡ってできるだけ多くの具体例・図を追加する予定
- 3.5.6 節を完成させる（下に凸な曲線の折れ線近似，凸な区分線形関数）
- 7.3 節を完成させる（巡回セールスマン問題）
- 7.6 節を完成させる
- 7 章に「ソルバと連携するプログラムの作成」を追加
- 6 章に以下の問題を追加
  - 最小費用流問題
  - （1 次元）資材切り出し問題
  - 栄養問題
  - 最適配置問題
  - 選手選択問題
  - シフト作成問題
  - 彩色問題
  - 3 次元パッキング問題
  - etc...
- 追加する予定の定式化技法
  - 複数期間モデル（在庫の引き継ぎ）
  - 大きなものから k 個取り出し
  - 先行制約
- その他適当な場所があれば扱う項目
  - Totally Unimodular
  - 表面 (facet) に関する説明（3.6.3 節）
  - 最適解を実運用で使うために注意すべきこと
- 「モデル作成チュートリアル」or「ソルバによる求解チュートリアル」の作成
- 掲載している最適化モデルの一覧

参考文献

[1] C. Alexander, A pattern language, Oxford University Press, (1977).

[2] F. Alizadeh and D. Goldfarb, Second-order cone programming, Mathematical Programming, B, **95** (2003), 3-51.

[3] 青木義次，建築計画・都市計画の数学 規模と安全の数理，数理工学社，(2006).

[4] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, The Traveling Salesman Problem, Princeton, (2006).

[5] E.M.L. Beale and J.A. Tomlin, Special Facilities in General Mathematical Programming System for Non-Convex Problems Using Ordered Sets of Variables, Proc. 5th IFORS Conference, pp. 447-454, J. R. Lawrence (ed.), Tavistock, London & Wiley, New York (1970).

[6] R. Bixby, Z. Gu, E. Rothberg, Presolve for Linear and Mixed-Integer Programming, 第 24 回 RAMP シンポジウム論文集，(2012).

[7] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry –Algorithms and Applications– (Third Ed.), *Springer*, 2008.

[8] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, Solution of a Large-Scale Traveling-Salesman Problem, Operations Research, **2** (1954), 393-410.

[9] E. D. Dolan and J. J. Moré, Benchmarking optimization software with performance profiles, Mathematical Programming, **91** (2002), 201-213.

[10] Robert Fourer, David M. Gay, and Brian W. Kernighan AMPL: A Modeling Language for Mathematical Programming, Second edition, *Duxbury Press*, (2003). Also, this book is downloadable from `http://ampl.com/resources/the-ampl-book/chapter-downloads/`.

[11] 福島雅夫，非線形最適化の基礎，朝倉書店，(2001).

[12] 福島雅夫，新版 数理計画入門，朝倉書店，(2011).

[13] E. Gamma, R Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Pub, (1995).

[14] 久保幹雄，J.P. ペドロソ，メタヒューリスティクスの数理，共立出版，(2009).

[15] 宮代隆平，ここまで解ける整数計画 –近年の発展–，第 20 回 RAMP シンポジウム論文集，(2008).

[16] 森 雅夫，松井 知己，オペレーションズ・リサーチ（経営システム工学ライブラリー），朝倉書店，(2004).

[17] S. H. Strogatz, Love Affairs and Differential Equations, Mathematics Magazine, **61** (1988), 35.

[18] 杉原正顯，杉原厚吉（編著），数理工学 最新ツアーガイド 応用から生まれつつある新しい数学，日本評論社，(2008).

[19] 杉山佳彦，数理計画問題におけるデザインパターンの提案，関西大学システムマネジメント工学科卒業論文，(2008).

[20] 滝根哲哉，伊藤大雄，西尾章治郎，岩波講座 インターネット 第 5 巻 ネットワーク設計理論，岩波書店，(2001).

[21] H. Paul Williams, Model Building in Mathematical Programming, Fourth Edition, Wiley, (1999).（本書第 3 版の日本語訳：数理計画モデルの作成法，前田英次郎監訳，小林英三訳，産業図書，(1995)（絶版））

[22] 山地憲治，システム数理工学 –意思決定のためのシステム分析–（新・電気システム工学 TKE-7），数理工学社，(2007).

[23] 柳浦睦憲，茨木俊秀，組合せ最適化 –メタ戦略を中心として–（経営科学のニューフロンティア 2），朝倉書店，(2001).

《Web サイト》

[24] Benchmarks for Optimization Software, `http://plato.asu.edu/bench.html`.（2016 年 9 月 14 日確認）

[25] FICO Xpress Optimization Suite, `http://www.fico.com/en/products/fico-xpress-optimization-suite/`.（2014 年 8 月 21 日確認）

[26] 藤澤克樹，高速かつ省電力なグラフ解析とその実応用，`http://acsi.hpcc.jp/2016/download/ACSI2016-tutorial1.pdf`.（2014 年 9 月 10 日確認）

[27] GLPK (GNU Linear Programming Kit), `https://www.gnu.org/software/glpk/`.（2014 年 8 月 21 日確認）

[28] GLPK (Wikibook), `http://en.wikibooks.org/wiki/GLPK`.

解ファイルについての解説: `http://en.wikibooks.org/wiki/GLPK/Solution_information`.（2014 年 12 月 6 日確認）

[29] Graph 500, `http://www.graph500.org`.（2016 年 9 月 10 日確認）

[30] Green Graph 500, `http://green.graph500.org`.（2016 年 9 月 10 日確認）

[31] Gurobi Optimization, `http://www.gurobi.com`.（2014 年 8 月 21 日確認）

[32] GUSEK (GLPK Under Scite Extended Kit), `http://gusek.sourceforge.net/gusek.html`.（2016 年 9 月 14 日確認）

[33] IBM CPLEX Optimizer,
http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.（2014 年 8 月 21 日確認）

[34] lp_solve reference guide, http://lpsolve.sourceforge.net.（2014 年 8 月 21 日確認）

[35] MIPLIB2010, http://miplib.zib.de.（2014 年 9 月 9 日確認）

[36] Numerical Optimizer, http://www.msi.co.jp/nuopt/.（2014 年 8 月 21 日確認）

[37] SCIP (Solving Constraint Integer Programs), http://scip.zib.de.（2014 年 8 月 21 日確認）

《本文中では参照していないが有益な参考図書》

（数理最適化全般）

[38] 穴井宏和，斎藤努，今日から使える！ 組合せ最適化 離散問題ガイドブック，講談社，(2015).

数理最適化の中でも組合せ最適化に焦点を当て，その中でもよく出てくる問題（これを「標準問題」としています）の定式化や解法が丁寧に説明されています．数理最適化を実務に使うことを目指した一冊です．なお，著者の一人（斎藤氏）による Web 上の記事も参考になります：
http://qiita.com/Tsutomu-KKE@github/items/bfbf4c185ed7004b5721

[39] 久保幹雄，J. P. ペドロソ，村松正和，A. レイス，あたらしい数理最適化　Python 言語と Gurobi で解く，近代科学社，(2012).

最適化ソフト Gurobi を用いて最適化問題を解く手法を解説しています．特に，Gurobi をプログラミング言語 Python から呼び出す，あるいは求解結果を Python で処理することに主眼が置かれていますが，様々な最適化問題の定式化に対する解説も充実しています．

（数理モデル）

[40] 藤澤克樹，梅谷俊治，応用に役立つ 50 の最適化問題，朝倉書店，(2009).

タイトルにあるように，様々な最適化問題が紹介されています．モデル化・定式化のさまざまな方法を学習するためにも使えますし，自分の考えている問題と同じ問題や近い問題が無いかを探すための辞書的な使い方もできる一冊だと思います．

[41] 久保幹雄，サプライ・チェイン最適化ハンドブック，朝倉書店，(2007).

物流・ロジスティクス等に現れる問題を最適化問題として定式化・求解・分析することに焦点を当てたハンドブックです．しかしながら，その記述は必ずしもサプライ・チェインだけにとどまるものではなく，様々な種類の最適化問題を考える上で有益な情報が多く含まれています．

[42] 日本オペレーションズ・リサーチ学会 監修，室田一雄・池上敦子・土屋 隆 編，モデリング -広い視野を求めて-（シリーズ：最適化モデリング 1），近代科学社，(2015).

日本オペレーションズ・リサーチ学会の創立 60 周年記念事業の一つとして刊行されるシリーズの 1 冊です．16 章からなり，各章ではモデル化という作業の意味が様々な角度から説明されています．

[43] 柳井浩，数理モデル（基礎数理講座 4），朝倉書店，(2009).

様々な数理モデルが紹介されている一冊です．最適化に関する記述は多くありませんが，実現象を数理的にとらえるための視点を多岐にわたって与えてくれます．

（解の分析）

（その他）

[44] 宮代隆平，松井知己，ここまで解ける整数計画，システム/制御/情報，50-9 (2006), pp. 363-368. ※http://www.tuat.ac.jp/~miya/miyashiro_matsui_ISCIE_rev5.pdf から入手可能（2014 年 9 月 9 日確認）

少し前の記事ですが，ベンチマーク問題の結果以外はほとんど現在（2014 年）でも通用する内容だと思います（タイトル・内容から推測するに，[15] はこの記事を発展させたものでしょう）．本書では直接の引用箇所はありませんが，この記事から着想を得た所が多くあります．

[45] 宮代隆平，整数計画法メモ，http://www.tuat.ac.jp/~miya/ipmemo.html．（2014 年 9 月 9 日確認）

[44] の著者の一人が作成している整数計画法に関する Web ページです．様々な情報がコンパクトにまとまっています．実際にソルバを使った計算をする時にとても頼りになるページです．

[46] 社団法人日本オペレーションズ・リサーチ学会刊行物一覧，
http://ci.nii.ac.jp/organ/journal/INT1000001610_ja.html.

国立情報学研究所が運営している，論文や図書・雑誌などの検索ができるポータルサイト CiNii http://ci.nii.ac.jp 内に収録されている，日本オペレーションズ・リサーチ学会の刊行物一覧です.（数理）最適化はオペレーションズ・リサーチ (OR) と呼ばれる研究分野の中の一領域で，本ページからも最適化に関する様々な情報が検索できます．

[47] 社団法人 日本オペレーションズ・リサーチ学会 OR 事典編集委員会，OR 事典 Wiki，
http://www.orsj.or.jp/~wiki/wiki/index.php/メインページ.

日本オペレーションズ・リサーチ学会が編纂している Web 上のオペレーションズ・リサーチ全般に関する辞典です．専門家の手による編集がなされていますので，記述内

容に対する信頼性は非常に高いと言えます．記述はオペレーションズ・リサーチ全般に及びますが，数理計画・最適化に関しても理論的な記述や応用事例が非常に充実しています．

索 引

A

AMPL 12

B

big-M 63

G

GLPK 14

H

Hitchcock 型輸送問題 59, 121
Hodgkin–Huxley モデル 33

K

Karush-Kuhn-Tucker 条件 92

L

Lagrange 関数 92
Lagrange 乗数 92

あ

アニーリング法 129
1 次関数 153
遺伝アルゴリズム 129
入次数（有向グラフの節点の） 161
枝 159
エピグラフ 154
重み（グラフの枝の） 161
重み付き制約充足問題 140
親 162

か

下界値 93
完全グラフ 162
感度分析 99
緩和問題 92
木 161
ギャップ（上界値と下界値の） 93
局所的最適解 37
近似解法 129
近傍 38
区分線形関数 66
グラビティモデル 32
グラフ 159
厳密解法 129
子 162
考慮制約 140
混合整数線形計画問題 42
混合整数 2 次計画問題 42
混合整数非線形計画問題 42

さ

最小 2 乗問題 60
最小包含円問題 141
最大流問題 59, 116
最短路問題 142
最適解 9, 37
最適化問題 9
最適化問題（概念としての） 35
最適化問題（数理モデルとしての） 36
最適性の原理 143
残差（KKT 条件の） 92
暫定解 94, 129
自己閉路 161
次数（グラフの節点の） 161
実行可能 37
実行可能解 36
実行可能領域 36
実行不可能 37
シャドウ・プライス 102
集合パッキング制約 55
集合パッキング問題 110
集合被覆制約 55
集合被覆問題 112
集合分割制約 55

集合分割問題 109
主問題 94
巡回セールスマン問題 45
上界値 93
シンク 58
錐 165
錐線形計画問題 165
数学モデル 32
数独 143
数理最適化 9, 35
数理モデル 32
整数計画問題 41
整数線形計画問題 41
整数 2 次整数計画問題 41
整数非線形計画問題 41
整数変数 22, 41
制約 9
制約充足問題 143
制約条件 36
切除平面 136
絶対制約 140
節点 159
0-1 計画問題 42
0-1 変数 42
線形計画問題 39
潜在価格 102
双対問題 94
ソース 58
疎性（最適化問題の） 130
ソフト制約 140
ソルバ 9, 10

た

大域的最適解 37
ダイクストラ法 143
対称巡回セールスマン問題 125
妥当不等式 136
タブー探索法 129
単純閉路 161
単純路 161
定式化 36
定式化とパラメータの分離 51
デザインパターン 58
出次数（有向グラフの節点の） 161
等式制約 36
凸 154
凸関数 154
凸計画問題 41
凸結合 157
凸集合 154
凸錐 165
凸 2 次計画問題 40
凸包 157

な

ナップサック問題 56, 114
2 次計画問題 40
2 次錐計画問題 165
2 次制約 2 次計画問題 165
二部グラフ 162
根 162

は

葉 162
ハード制約 140
バイナリ変数 42
発見的解法 129
パラメータ 39, 48
半正定値行列 158
半正定値計画問題 165
非線形計画問題 40
非対称巡回セールスマン問題 125
非負制約 39
ヒューリスティック解法 129
標準形（最適化問題の） 39
ビンパッキング問題 115
物理モデル 32
不等式制約 36
部分巡回路 134
部分巡回路除去制約 136
フロー 58
フロー保存則 58
閉路 161
変数 36, 48

ま

待ち行列 32

道 161
無向枝 161
無向グラフ 161
目的 9
目的関数 36
モデリング言語 12
モデル 29
モデル化 29

や

有界でない 95
有向枝 159
有向グラフ 161
有効でない制約条件 99
有効な制約条件 99

ら

離散的（変数の値が） 41
リトルの公式 32
連結グラフ 161
連続計画問題 41
連続的（変数の値が） 41
連続変数 41
路 161

わ

割当問題 60, 123