第 1 章

イントロダクション

1.1 数理最適化

数理最適化問題 (または単に最適化問題) は以下の形で表される。

$$\begin{aligned} &\text{minimize} && f_0(x) \\ &\text{subject to} && f_i(x) \leq b_i, \quad i = 1, \cdots, m \end{aligned} \tag{1.1}$$

ここで、ベクトル $x = (x_1, \cdots, x_n)$ は問題の最適化変数、$f_0 : \mathbb{R}^n \to \mathbb{R}$ は問題の目的関数、$f_i : \mathbb{R}^n \to \mathbb{R}, \quad i = 1, \cdots, m$ は (不等形の) 制約関数、定数 $b_1, \cdots, b_m$ は制約関数の限界 (または境界) である。この問題の解、または最適値はベクトル x^* で表され、制約を満たす全てのベクトルの中で最小の目的値となる。式で示すと、$\forall z$ with $f_1(z) \leq b_1, \cdots, f_m(z) \leq b_m$ の時、$f_0(z) \geq f_0(x^*)$ を満たすベクトル x^* が問題の解となる。一般的には、目的関数と制約関数が特定の形式で特徴付けることが出来る最適化問題のクラスや族について考える。重要な例として**??**節と**??**節で示す最小二乗法や線形計画問題がある。例えば線形計画問題では、もし目的関数と制約関数 $f_0, \cdots, f_m$ が線形である場合、以下を満す。

$$f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y) \tag{1.2}$$

ここで、$\forall x, y \in \mathbb{R}^n, \forall \alpha, \beta \in \mathbb{R}$ である。問題が線形ではない場合は非線形計画問題と呼ばれる (**??**で少し触れる)。本書で扱うのは凸最適化問題と呼ばれる問題であり、目的関数と制約関数が凸面である問題である。凸面であるということは以下を満たすことを言う。

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y) \tag{1.3}$$

ここで、$\forall x, y \in \mathbb{R}^n, \forall \alpha, \beta \in \mathbb{R}$ with $\alpha + \beta = 1, \alpha \geq 0, \beta \geq 0$ である。(1.2) と (1.3) を比較すると、凸面は線形性の一般化であることが分かる。

1.1.1 応用

最適化問題はある候補選択の集合から最良の選択となるようなベクトル x^* を選ぶ問題の抽象化と考えることが出来る。変数 x は選択した候補、制約関数 $f_i(x) \leq b_i$ は選択肢を制限するような不変的な要件や定義、目的関数 $f_0(x)$ は x を選択するコストである。$-f_0(x)$ として x を選択する効果や有効性と考えることもできる。この解は全ての制約を満たす選択の中で最小のコスト (または最大の有用性) を持つ選択に対応している。

例 1. ポートフォリオを例にとると、n 個の資産の集合に資本を分配して投資する際の最良の方法を探すのが最適化の目的となる。式 (1.1) で x_i は i 番目の資産への投資を表し、ベクトル $x \in \mathbb{R}^n$ は資産の集合全体への投資割り当てを示している。ここでの制約は予算の限度、投資額が非負であること、ポートフォリオ全体の期待収益額に対する最小の許容値 *(*つまり儲けは最小でどれだけいるか*)* である。目的 *(*またはコスト*)* 関数はポートフォリオ全体のリターンに対するリスク、分散の尺度が該当する。この問題の場合は、全ての不変的な制約を満す割り当てから最小のリスクとなる割り当てを選ぶ問題に対応する。

他にもデバイスサイズ (デバイスの幅、長さの最適化) やデータフィッティング (得られたデータに最も良く一致するモデルを選択する) などに応用可能であり、意思決定 (システム設計、解析、運用) を含む非常に多くの実世界の問題はこの最適化問題や多目的最適化問題の形に当てはめることができる。これらの応用の大部分で数理最適化はプロセスの監督、結果のチェック、必要に応じて問題の修正を行う人間の意思決定、システム設計、システム運用の補助を行う。意思決定を行う人間は、数理最適化によって示唆された行動を実際に実行する。

最近では数理最適化は組込みでの応用が急増している。このような組込み最適化では最適化を用いてリアルタイムでの選択と関連する操作を自動的に実行するため、ほとんど人間の手や監視は介さないで実行される。ただ、組込み最適化には非常に高い信頼性を維持しつう予測可能な時間で問題を解決する手法が必要という課題もある。

1.1.2 最適化問題の解法

最適化問題のクラスに対する解法は、クラスから特定の問題、つまり問題のインスタンスが与えられた時に与えられた精度で問題を解くアルゴリズムである。1940 年代後半以降、様々なクラスの最適化問題を解くアルゴリズムの開発、それらの特性解析、そして良いソフトウェア実装に多大な努力が払われてきた。これらのアルゴリズムの有用性は大きく変化し、特定の形の目的、制約関数や制約と変数の数、スパース性み代表される特徴的な構造のような要因に依存している。ここでいうスパース性は、問題の制約関数が少数の変数にのみ依存するような問題を言う。

たとえ目的関数や制約関数が十分滑らかであっても (多項式のように)、一般的な最適化問題は非常に解くのが困難である。そのため一般的な問題へのアプローチは計算時間が非常に長くなったり真の解が見つからない可能性などの妥協がある。しかし一般的に難しいとされる最適化問題の中でも、ある種の問題では数百〜数千の変数がある場合でも効率的に解くことが出来るアルゴリズムが存在する。その効率的に解けるアルゴリズムの中でも 2 つの重要で良く知られている例が 1.2節で述べる最小二乗法と線形計画問題である。またあまり知られていないが、本書で扱う凸最適化問題も同様に効率的に解くことが出来るアルゴリズムが存在する。

1.2 最小二乗法と線形計画問題

1.2.1 最小二乗法

最小二乗法は制約関数が無く $(m = 0)$、目的関数が $a_i^T x - b_i$ の形式の項の二乗和で表される最適化問題である。

$$\text{minimize} \quad f_0(x) = \|Ax - b\|_2^2 = \sum_{i=1}^{k} (a_i^T x - b_i)^2 \tag{1.4}$$

ここで、行列 $A \in \mathbb{R}^{k \times n}$ with $k \geq n$, A の行を表す a_i^T、ベクトル $x \in \mathbb{R}^n$ は最適化変数である。

最小二乗法の解

最小二乗法の解は線形方程式の集合を解くことに帰着する。

$$(A^T A)x = A^T b \tag{1.5}$$

ここから x について解くと、$x = (A^T A)^{-1} A^T b$ となる (正規方程式)。この問題は高精度かつ非常に高い信頼性で解くことが出来るアルゴリズムが存在し、既知の定数 $n^2 k$ に比例する時間で解くことができる。これは現在のデスクトップコンピュータを用いても 100 個の変数と 1000 個の項であれば数秒で解けてしまう。

係数行列 A が特殊な構造を持つ場合、より大きな問題を解くことが可能となる。例えば行列 A がスパース性を持つ場合、上で示した計算量 $n^2 k$ よりもはるかに速く解くことができる。

さらに大きな問題を解く場合やリアルタイム性が要求される厳しい場面では問題を解くのは困難であるが、知られている解法は非常に高効率かつ高信頼性であり、詳細を知らない人でも確実に利用できる。

最小二乗法の利用

最小二乗法は回帰分析、最適制御、多くのパラメータ推定とデータフィッティングの基礎となる。フィッティングの例で言えば、ガウスの誤差関数によって欠損した線形測定データが与えられた場合にベクトル x の最尤測定として様々な統計的解釈を与えてくれる。最小二乗法形式の最適化問題を認識するのは簡単であり、目的関数が二次形式かつ関連する二次形式が半正定値であることを確認すればよい (多分最小値を取る点で傾きが 0 になるため、他の点では二乗しているので傾きは正になる (?))。最小二乗法の基本的な形式は非常に単純である一方、いくつかの標準的なテクニックを用いて応用の幅を広げているのもある。

その 1 つとして重み付き最小二乗法がある。これは式にある基準の重みを付け、

$$\sum_{i=1}^{k} w_i (a_i^T x - b_i)^2 \tag{1.6}$$

を最小化する。ここで、$w1, \cdots, w_k$ は重みである。この重みは項 $a_i^T x - b_i$ のサイズに対する様々な懸念 (関心) を反映するように、または単に解に影響を与えるために設定される。統計的には不均一な分散を持つ誤差によって損われた線形測定データが与えられた場合のベクトル x を推定する際に生じる。

もう 1 つのテクニックとして正則化がある。これはコスト関数に追加の項を置くことでモデルに対して罰則を設ける方法である。簡単なケースでは、変数の平方和にある倍数をかけたものが追加される。

$$\sum_{i=1}^{k} (a_i^T x - b_i)^2 + \rho \sum_{i=1}^{n} x_i^2 \tag{1.7}$$

ここで、$\rho > 0$ である。追加の項は変数 x が大きい場合に罰則が加わり、結果として最初の項の和のみを解としない場合よりもより良い解となる。追加項に付くパラメータ ρ は元の目的関数を小さくしつつ、追加項を大きくしすぎないようにする適切なトレードオフを与えるようにユーザによって選ばれる。統計的には、事前分布をベクトル x に与えることで正則化が生じるようになる。

1.2.2 線形計画問題

もう 1 つの重要な最適化問題のクラスはこの線形計画問題である。この問題は全ての目的関数と制約関数が線形である問題である。

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & a_i^T x \leq b_i, \quad i = 1, \cdots, m \end{aligned} \tag{1.8}$$

ここで、ベクトル $c, a_1, \cdots, a_m \in \mathbb{R}^n$, スカラー $b_1, \cdots, b_m \in \mathbb{R}$ は目的、制約関数を記述する問題変数である。

線型計画問題の解

線形計画問題を解くための簡単な解析式は無いが、本書の後半で紹介される Dantzig's のシンプレックス法や交点法を用いて効果的に問題を解くことができる。最小二乗法とは異なりどのぐらいの算術演算で解が求まるかを与えることはできないが、特定の精度で問題を解くのに必要な演算の限度を厳密に知ることができる。実際の複雑さは $n^2m(n \geq m$ と仮定) だが、定数に関しては最小二乗法ほど特徴付けられてはいない。信頼性に関しても、使用するアルゴリズムの信頼性は非常に高いが、最小二乗法ほど信頼性が高いというわけではない。スパース性を用いることでより大きな問題が解けるという点や非常に大きな問題、リアルタイム性が求められる場面では課題が残るという点は最小二乗法の場合と同様である。

線型計画問題の利用

線型計画問題のいくつかの応用は式 (1.8) をそのまま流用することができる。他の多くの応用では式 (1.8) の形になっていないが、4 章で詳細に示す方法を用いることで等価的に変形することができ、解くことができる。

簡単な例でチェビシェフの近似法がある。

$$\text{minimize} \quad \max_{i=1,\cdots,k} |a_i^T x - b_i| \tag{1.9}$$

ここで、$x \in \mathbb{R}^n$ は変数、$a_1, \cdots, a_k \in \mathbb{R}^n, b_i, \cdots, b_k \in \mathbb{R}$ は問題を記述するためのパラメータである。チェビシェフの近似法の式 (1.9) と最小二乗法の式 (1.4) を比較すると、式の形は同じで二乗和を取るか絶対値の最大値を取るかの差であることが分かる。また、チェビシェフの近似法では絶対値を使用するので微分可能では無いが、最小二乗法は二次式なので微分可能であるという違いがある。このチェビシェフの近似法は以下の線形計画問題を使って容易に解くことができる。

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & a_i^T x - t \leq b_i, i = 1, \cdots, k \\ & -a_i^T x - t \leq b_i, i = 1, \cdots, k \end{aligned} \tag{1.10}$$

1.3 凸最適化

凸最適化問題は

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq b_i, i = 1, \cdots, m \end{aligned} \tag{1.11}$$

ここで、関数 $f_0, \cdots, f_m : \mathbb{R}^n \to \mathbb{R}$ は凸面であり、つまり

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y) \tag{1.12}$$

を満たす。また、$\forall x, y \in \mathbb{R}$, $\forall \alpha, \beta \in \mathbb{R}$ with $\alpha + \beta = 1, \alpha \geq 0, \beta \geq 0$ である。

1.3.1 凸最適化の解

凸最適化は線形計画問題と同様に解くための解析式は無いが、非常に効果的に解くことができる方法がある。内点法は実際の計算では良く機能しており、特定の場合では問題の多項式次元を越えない数の演算数で特定の精度まで解けることが証明されている。内点法では問題の構造に関係なく、各ステップにおいて $\max\{n^3, n^2m, F\}$ 回演算が必要になる。F は目的関数と制約関数の第一次と第二次導関数を評価するコストを指す。この方法は線型計画法で述べたのと同様に非常に高い信頼性で大きな問題を解くことができる。

とはいえ、一般的な凸最適化問題を解くような技術は最小二乗法や線形計画問題ほど成熟してはいない。非線形凸最適化に関しては現在も活発に研究が進められている領域であるし、今までの方法でどれが最良かというコンセンサスも得られてはいないためである。しかし、一般的な凸最適化問題はここ数年以内に解ける技術となると期待されており、二次錐計画法や幾何計画法などの凸最適化のサブクラスは、内点法がその技術にアプローチしていると言ってもいいだろう。

1.3.2 凸最適化の利用

概念的には凸最適化問題は上で述べた最小二乗法や線形計画問題のように利用することになり、凸最適化の形に問題を定式化できれば解くことができる。ただし、最小二乗法などの問題は簡単に認識することができるが、凸最適化問題と認識するのは難しく、凸最適化問題を定式化するのにはより必要トリックが多くなる。そのため、凸最適化問題を認識、または定式化するのは困難であるが、一度定式化できてしまえばその後の問題の解を技術的に求めることができるようになる。本書では読者がこの凸最適化問題の認識、定式化を行えるようになるための背景技術を与えるのが目的となる。一度凸最適化問題を認識するスキルが備わることで、非常に多くの問題が解決できることが分かるようになるだろう。

1.4 非線形最適化問題

非線形最適化問題は目的関数または制約関数が線形ではないが、凸関数であると分かっていないような問題を指す。この場合の効果的な解法はまだ存在せず、10 程度の変数の問題でさえ非常に長い時間がかかる場合がある。そのため別の側面から解法のアプローチがされているが、それぞれには何かしらの妥協点が存在する。以下では局所最適化と大域最適化について述べる。

1.4.1 局所最適化

局所最適化では全ての可能な点における最適値 x を求めず、ある局所的な領域での最適解を求める方法である。局所的な領域では最適値となるが、それが大域的な最適解になるかの保証は無い。

局所最適化は目的関数と制約関数が微分可能性のみを条件とするので、高速で大規模問題を扱うことができ、幅広い問題に適用可能である。そのため真の最適値ではないが良い点を見つけるのに価値が見出せるような場面で広く利用されている。

局所最適化の欠点は大域的な最適解を見つけられない以外にも存在する。この方法では最適化変数の初期推定値が必要となり、この初期推定値によって得られる局所最適解に大きく影響を与える。そのため特定の問題やクラスに合わせてパラメータ値を調整する必要がある。また、大域最適解から現在の局所最適解がどの程度離れているかについては全く情報が与えられていない。

局所最適化を扱うのは最小二乗法などよりも難しく、アルゴリズムの選択、アルゴリズムのパラメータの調整、適切な初期推定値の選択または生成を含んでいる。おおざっぱに言うと局所最適化は単なる技術ではなく熟練した技術 (職人技のような) が求められる。

非線形計画法での局所最適化と凸最適化での興味深い比較がある。ほとんどの局所最適化では目的関数と制約関数の微分可能性のみが必要条件なので、定式化は比較的簡単であるが、問題を解くことが局所最適化での熟練した技術となる。一方凸最適化ではその熟練した技術が定式化の部分にあたるが、問題を解くのは比較的簡単である。

1.4.2 大域最適化

大域最適化では式 1.1の真の最適解が求まる。ただし、この場合は効率性を妥協しており、最悪の場合は問題のサイズ n と m に対して指数関数的に増加する。実際には特定の問題のインスタンスに対してより高速になることが期待されるが、典型的に起きるものではない。たとえ変数が数十しかない場合でも、解くのには非常に長い時間がかかる。

この大域最適化は計算時間が重要ではなく、大域最適解を見つける価値が非常に高い、少数の変数を持つ問題に対して利用される。利用例としては、安全性が非常に重要なシステム設計における最悪の場合の解析や分析である。変数は不確実なパラメータを表しており、製造時や動作環境の条件によって変動する。目的は効用関数、制約関数はパラメータ値に関する事前知識となる。大域最適化問題の解は最悪のケースでのパラメータの値を探すことである。この値が許容できる値にであれば、システムが安全かつ信頼できると証明することが可能である。一方局所最適化で同じ問題を解いても証明することはできない。あくまで局所的な解でありその解が大域的な解ではないため、システムの安全性を証明できないためである (得られた解よりもより最悪のケースでの値が見つかる可能性があるため)。