筑波大学大学院博士課程

システム情報工学研究科修士論文

単体上における非凸関数の大域的最適化

千葉 竜介

修士（工学）

（コンピュータサイエンス専攻）

指導教員　久野 誉人

2017年3月

概要

凸関数最適化問題と比較して非凸関数の最適化は一般的に困難であると言われている．その中でも，非凸関数の一種である単調関数（Monotonic function）の最適化を扱う．単調関数とは，入力の増加に対して関数値も常に増加または減少する関数のことである．そのような単調関数について，実行可能領域を単体（Simplex）とした場合の最適化を考える．

本稿では，大域的最適化の手法としてよく利用されている分枝限定法を利用する．分枝限定法は実行可能領域の分割を繰り返し，たくさんの小さな問題を解く手法である．実行可能領域における関数値の下界と暫定解と比較することで，その領域に最適解が存在しうるかどうかを判断し，最適解が存在し得ない場合はその領域を削除して効率的に最適化を行う．したがって分枝限定法において，良い下界を与えること，効果的に実行可能領域を分割することは重要である．

単体の新たな分割方法の提案を行い，その分割手法が生成する子領域の数の上限を示す．さらに，提案した分割手法を使った分枝限定法の実験を行い，提案手法を評価する．

目 次

第1章　序論

1.1　概要

1.2　記号の定義

表記	意味
$\mathbb{R}$	実数全体の集合
$\mathbb{R}^n$	n 次元実数ベクトル全体の集合
$\mathbb{R}^n_+$	すべての要素が非負である n 次元実数ベクトル全体の集合
$\mathbb{Z}$	0 を含む自然数全体の集合
$\mathbb{Z}^n$	n 次元自然数ベクトル全体の集合
e_i $(e_i \in \mathbb{R}^n)$	i 番目の要素が 1 でそれ以外が 0 であるような単位ベクトル
x_i $(x \in \mathbb{R}^n)$	ベクトル x の i 列目の要素
$x < y$ $(x, y \in \mathbb{R}^n)$	$x_i < y_i$ $(i = 1, ..., n)$
$x \leq y$ $(x, y \in \mathbb{R}^n)$	$x_i \leq y_i$ $(i = 1, ..., n)$

1.3　構成

第 1 章では本論文の概要の説明と記号の定義を行った．第 2 章では本論文で扱う問題の定義とその性質を論じる．第 3 章では提案手法に関連する研究の説明を行い，第 4 章で提案手法の説明を行う．第 5 章では数値実験の結果を紹介し，第 6 章で結論を述べる．

第2章　単体上の単調関数最適化問題

本研究で扱う問題は以下のような単調関数 f の最小化問題である．

$$\left|\begin{array}{ll} \text{最小化} & f(x) \\ \text{条件} & \displaystyle\sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, i = 1, ..., n \end{array}\right.$$

制約条件に，x の各要素の和が 1 であることと非負であることがある．このような条件を満たす x の集合は単体と呼ばれており，様々な研究がなされている．この問題は単調関数の単体上における最適化問題であり，まずは単調関数と単体を定義し，それらの性質について論ずる．

2.1　単調関数

単調関数とは，単調増加関数または単調減少関数のことであり，入力の増加に対して関数値も常に増加または減少する関数である．

定義 1. $I \subseteq \mathbb{R}^n$ の関数 $f : I \to \mathbb{R}$ は任意の $x, x' \in I$ に対して以下が成り立つ場合に単調増加関数であるという．

$$x < x' \Rightarrow f(x) \leq f(x')$$

定義 2. $I \subseteq \mathbb{R}^n$ の関数 $f : I \to \mathbb{R}$ は任意の $x, x' \in I$ に対して以下が成り立つ場合に単調減少関数であるという．

$$x < x' \Rightarrow f(x) \geq f(x')$$

ここで I は n 次元実数空間全体や区間，離散的な空間が考えられる．

定理 1. $f : I \to \mathbb{R}$ が単調増加関数であるとき，$-f$ は単調減少関数である．逆も同様に成り立つ．

証明. f は単調増加関数であるから，任意の $x, x' \in I$ について以下が成り立つ．

$$\begin{aligned} & x < x' \Rightarrow f(x) \leq f(x') \\ \Longleftrightarrow \quad & x < x' \Rightarrow -f(x) \geq -f(x') \end{aligned}$$

これは $-f$ が単調減少関数であることを示し，f が単調増加関数であることと，$-f$ が単調減少関数であることが同値であることが証明された． □

定理 1 より，関数値を -1 倍する操作によって単調増加と単調減少が切り替わり，単調性は失われないことがわかる．さらに，最適化問題において，ある関数 f の最大化は $-f$ の最小化と同じであるため，関数値を -1 倍することで最大化問題を最小化問題に変換することができる．ここからは単調増加関数の最小化問題についてしか論じないが，この通り最大化問題は最小化問題に変換することができ，単調性は保存されることから，一般性は失わない．

2.2 単位単体

定義 3. 以下を単位単体という．

$$\Delta_n \triangleq \left\{ x \in \mathbb{R}^n_+ \;\middle|\; \sum_{i=1}^{n} x_i = 1 \right\}$$

n について具体的に考えてみると，$n = 2$ のとき線分，$n = 3$ のとき平面上の正三角形，$n = 4$ のとき三角錐となることがわかる．このように n に対して一次元減った図形になることから，先ほど定義した n 次元の単位単体 Δ_n は $n-1$ 単体と呼ばれる．

2.3 単体上の単調関数の性質

単調増加関数 $f : \mathbb{R}^n_+ \to \mathbb{R}$ の単位単体上における性質について考える．まず，単体の定義から以下が成り立つ．

定理 2. 単体上の任意の点 $x, y \in \Delta_n$ について $x < y$ が成り立つことはない．

証明. 背理法で示す．$x < y$ となるような x, y が Δ_n 中に存在すると仮定する．
$x < y$ であるから，

$$\begin{aligned} & x_i < y_i \ , i = 1, ..., n \\ \Longrightarrow \quad & \sum_{i=1}^{n} x_i < \sum_{i=1}^{n} y_i \end{aligned}$$

これは $x, y \in \Delta_n$ つまり，$\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i = 1$ に矛盾する．
よって，$x < y$ が成り立つような単体上の点 x, y は存在しないことが証明された． □

つまり，単調増加関数の唯一の性質である $x < y \Rightarrow f(x) < f(y)$ を利用できる点が単体上には存在せず，単体上だけを見ると特に性質のない任意の関数であるように見える．以上のことから，単体上における単調関数の最適化は一般的な大域的最適化問題を解くことと同程度に困難であると予想される．

しかし，実行可能領域以外の領域の情報を利用し，単体上における下界を得ることができる．

定理 3. 以下のような要素を持つ $x_{LB} \in \mathbb{R}^n_+$ は単体の任意の部分領域 $A \subseteq \Delta_n$ における単調増加関数 $f : \mathbb{R}^n_+ \to \mathbb{R}$ の関数値の下界 $f(x_{LB})$ を与える．

$$(x_{LB})_i = \min_{y \in A} y_i \ , \ i = 1, ..., n$$

証明. x_{LB} の生成方法より，A の任意の要素 x について $x_{LB} \leq x$ が成り立つ．
また，f は単調増加関数であったから，$\forall x, y \in \mathbb{R}^n_+, x < y \Rightarrow f(x) < f(y)$ が成り立ち，

$$\forall x \in A, f(x_{LB}) \leq f(x)$$

よって，$f(x_{LB})$ は単体の任意の部分領域 A における単調増加関数 f の下界となる． □

第3章　関連研究

3.1　分枝限定法

分枝限定法とは，大域的最適化の汎用アルゴリズムであり，広く利用されている．実行可能領域を分割し，それぞれの分割された領域に対して下界値・上界値を求め，それらや暫定最適値を比較することで最適解が存在し得ない領域を削除し，探索する領域を絞り込む．以上を繰り返して効率的に探索をする最適化手法である．

最小化問題を解く場合の一般的なアルゴリズムとして分枝限定法の説明を行う．分枝限定法は大きく分けて分枝・限定の 2 種類の操作から成る．分枝操作では，実行可能領域から領域 S を選択し，$S_1 \cup S_2 \cup ... \cup S_n = S, S_1 \cap S_2 \cap ... \cap S_n = \phi$ を満たす n 個の部分領域に分割する．分割の方法や分割数はアルゴリズムによって異なり，問題に対して適した手法を考えることが重要である．次に限定操作では，それぞれの部分領域について下界値を求める．その下界値が，他の領域の上界値よりも大きい場合や，暫定最適値よりも大きい場合はその部分領域に最適解が存在し得ないため，その領域の探索を打ち切る．ある領域の探索を打ち切り，削除することを刈り込みと呼ぶ．領域の刈り込みを効率的に行うには，よい下界，上界を得ること，探索中に素早くよい暫定最適値を見つけることが重要である．以上の操作をすべての領域を刈り込むまで繰り返し実行し，その時点の最適値を問題の最適値として出力する．

3.2　単体のグリッド化

de Klerk [2] などにより，単体をグリッド化して探索を行う研究がされている．ここでグリッド化とは，単体の内部に格子を考え，格子の交差点だけを抽出し，単体の内部に等間隔に存在する点群を得ることである．グリッド化によって実行可能領域は小さくなり，解きやすくなる．まずは単体のグリッド化の定義を行う．

$m \in \mathbb{Z}$ を利用して，n 次元の単位単体の各辺を m 分割して得られるグリッド化された単位単体は以下のように表せる．

$$\{\, x \in \Delta_n \mid mx \in \mathbb{Z}^n \,\}$$

これまでは単位単体のみを扱ってきたが，正の実数 c を利用して，各辺の長さが c であるような単体 $c\Delta_n$ をグリッド化したもの $G(c, n, m)$ を以下のように定義する．

定義 4.

$$G(c, n, m) \triangleq c\Delta_n \cap \{\, x \in \mathbb{R}^n \mid mx \in \mathbb{Z}^n \,\}$$

ここで，$c = 1$ としたとき，$G(1, n, m)$ は n 次元の単位単体の各辺を m 等分してグリッド化したものと同じであることがわかる．

TODO: $n = 3, m = 5$ のときのグリッド化された単体の図を描く

定理 4. グリッド化された単位単体 $G(1, n, m)$ が持つ点の数は二項係数を利用した以下で与えられることがわかっている．

$$|G(1, n, m)| = \binom{n + m - 1}{m}$$

これは m を固定した場合の n の多項式であり，最適化手法としてグリッド化した単体上を全探索しても n の多項式オーダーでアルゴリズムが完了することがわかる．しかし，もちろん n, m を大きくした場合の $\binom{n+m-1}{m}$ はとても大きい数であり，例えば $n = 10, m = 100$ の場合，生成される点の数はおよそ 8.5×10^{12} であり，全探索は難しいと容易に想像できる．

3.3 単体の分割

TODO: わかりやすい図を描く

分枝限定法の分枝操作では実行可能領域をいくつかの部分領域に分割するが，本問題の実行可能領域は単体であるから，単体を分割する手法を考える．さらに，分枝操作は繰り返し行われるため，分割された部分領域も同様にすべて単体となっている必要がある．

外崎ら [3] による単体の分割手法は，グリッド化された単体を一次元少ない単体と，一列少ない単体の二つに分割する方法である．この手法を具体的に説明する．

$mb \in \mathbb{Z}^n$ を満たすような非負ベクトル $b \in \mathbb{R}^n_+$ を利用して c を次のように与える．

$$c \equiv 1 - \sum_{j=1}^{n} b_j > 0$$

このとき次元のインデックスの部分集合 $K = \{j_1, ..., j_k\} \subset \{1, ..., n\}$ を選択すると，単体上の最小化問題である元の問題 2 の部分問題を以下のように定義することができる．

$$P(K, b) \left| \begin{array}{ll} \text{最小化} & f(x) \\ \text{条件} & \displaystyle\sum_{j=1}^{n} x_j = 1 \\ & mb_j \leq mx_j \in \mathbb{Z}, \quad j \in K \\ & x_j = b_j, \qquad\qquad j \notin K \end{array} \right.$$

K に選択されなかった次元は b の値に固定され，K に選択された次元の変数のみを変数とするような問題となる．この問題は以下の問題と等価である．

$$
\left|\begin{array}{lll}
\text{最小化} & f(x) & \\
\text{条件} & y \in G(c,k,m) & \\
& y_i = x_{j_i} - b_{j_i}, & j = 1,...,k \\
& x_j = b_j, & j \notin K
\end{array}\right.
$$

この問題は各辺の大きさが c である k 次元のグリッド化された単体上の問題である．さらに定義より，$G(c,k,m)$ は各辺を mc 個に分割したグリッドの集合であり，定理 4 より以下が成り立つ．

$$
|G(c,k,m)| = \binom{k+mc-1}{mc}
$$

次に，K から一つのインデックス j_k を選択し $P(K,b)$ を二つの部分問題に分割する．

$$
P(K, b + e_{j_k}/m) \left|\begin{array}{lll}
\text{最小化} & f(x) & \\
\text{条件} & \displaystyle\sum_{j=1}^{n} x_j = 1 & \\
& mb_j \le mx_j \in \mathbb{Z}, & j \in K\backslash\{j_k\} \\
& mb_{j_k} + 1 \le mx_{j_k} \in \mathbb{Z} & \\
& x_j = b_j, & j \notin K
\end{array}\right.
$$

$$
P(K\backslash\{j_k\}, b) \left|\begin{array}{lll}
\text{最小化} & f(x) & \\
\text{条件} & \displaystyle\sum_{j=1}^{n} x_j = 1 & \\
& mb_j \le mx_j \in \mathbb{Z}, & j \in K\backslash\{j_k\} \\
& x_j = b_j, & j \notin K \\
& x_{j_k} = b_{j_k} &
\end{array}\right.
$$

これらはそれぞれ以下の 2 問題と同値である．

$$
\left|\begin{array}{lll}
\text{最小化} & f(x) & \\
\text{条件} & y \in G(c - 1/m, k, m) & \\
& y_i = x_{j_i} - b_{j_i}, & i = 1,...,k-1 \\
& y_k = x_{j_k} - b_{j_k} - 1/m & \\
& x_j = b_j, & j \notin K
\end{array}\right.
$$

$$
\left|\begin{array}{ll}
\text{最小化} & f(x) \\
\text{条件} & y \in G(c, k-1, m) \\
& y_i = x_{j_i} - b_{j_i}, \qquad i = 1, ..., k-1 \\
& x_j = b_j, \qquad\qquad j \notin K \\
& x_{j_k} = b_{j_k}
\end{array}\right.
$$

$P(K, b + e_{j_k}/m)$ は $P(K, b)$ からグリッドが一列減った問題であり，$P(K\backslash\{j_k\}, b)$ は次元が 1 次元減った問題である．以上のようにグリッド化された単体を，一列減った単体と一次元減った単体に分割することができた．もともと単体には一列という概念は存在しないが，グリッド化をしたことによってこのような操作が可能になった．さらに，これらの定義域のグリッド化された単体のグリッド数について見てみると，以下がわかる．

$$
|G(c - 1/m, k, m)| = \binom{k + mc - 2}{mc - 1}
$$

$$
|G(c, k-1, m)| = \binom{k + mc - 2}{mc}
$$

二項係数の性質として以下が知られている．

$$
\binom{k + mc - 1}{mc} = \binom{k + mc - 2}{mc - 1} + \binom{k + mc - 2}{mc}
$$

ゆえに，以下が成り立つ．

$$
|G(c, k, m)| = |G(c - 1/m, k, m)| + |G(c, k-1, m)|
$$

この分枝操作を繰り返すと，すべてのグリッド化された単体はただ一点のみを含む単体になり，その総数は $\binom{k+mc-1}{mc}$ となる．

元の問題 2 から分枝操作を繰り返すことで，問題をノードに持つ二分木 T を考えることができ，その木の葉は一点のみを含む単体となる．この二分木のノード数について以下のことが言える．

定理 5. T の総ノード数は $2\binom{n+m-1}{m} - 1$ である

証明. 帰納的に示す．$r = \binom{n+m-1}{m}$ とおく．$q < r$ であるような q に対して，q を葉の総数とする二分木のノード数は $2q - 1$ であると仮定する．
次に T の根を親に持つような二つの部分木を T_1, T_2 とし，それらの葉の数を r_1, r_2 とすると，$r = r_1 + r_2$ が成り立つ．さらに，$r_1, r_2 \leq r - 1$ であるから仮定より，T_1, T_2 のノード数は $2r_1 - 1, 2r_2 - 1$ である．
よって，T のノード数は

$$
(2r_1 - 1) + (2r_2 - 1) + 1 = 2(r_1 + r_2) - 1 = 2r - 1
$$

□

3.4 単調関数の最適化

TODO: 書く
Tuy による，単調関数の最適化.

第4章　提案手法

単体上の単調関数を分枝限定法で最適化する．分枝限定法において，領域の分割方法，下界の良さはアルゴリズムの速度に対して大変重要であり，本研究では単調関数の下界に定理3で得られるものを利用し，分割方法は3.3節で説明したもの利用する．

4.1　分割する次元の選択

TODO: わかりやすくなる絵を描く

分枝限定法において，分割された部分問題が良い関数値を持つことは，その部分領域が良い下界を持つことにつながり，その下界によってその領域を削除しやすくなる．さらに，良い暫定解を得ることにもつながるため，部分問題が良い関数値を持つことはアルゴリズムの早い収束に対して重要である．本節では，分割された部分領域が良い関数値を持つようにするための，分割する次元の決定方法について論じる．

3.3節の $P(K,b)$ では，元の変数 x の代わりに $y \in G(c,k,m)$ となるような変数 y が取り直されている．$G(c,k,m)$ は次元のインデックス j_k を選択することで二つの部分単体 $G(c,k-1,m),\ G(c-1/m,k,m)$ に分割される．選択された j_k 次元の値について詳しく見てみると，$G(c,k-1,m)$ では j_k 次元の変数値 x_{j_k} は b_{j_k} に固定されており，これに対応する $y \in G(c,k,m)$ における変数値 y_k は最小値0となる．

本問題で扱っている関数は単調増加関数であるから，ある次元 k に着目したとき，y_k が大きくなれば関数値は単調に増加し，逆に y_k が小さくなれば関数値は単調に減少する．よって，関数値に対して大きなインパクトを与えている次元 k を探し出し，k 次元について単体を分割すると，k 次元の値が最小値に固定された部分問題 $P(K\backslash\{k\},b)$ の関数値は小さくなると予想できる．

次に，大きなインパクトを与えている次元 k の探し方を考える．$G(c,k,m)$ の頂点の集合を考えると，それらは c 倍した k 次の単位行列 I の列を取り出したものとなっている．つまり，頂点の要素は以下のようになっている．

$$(c,0,0,...,0),(0,c,0,...,0),(0,0,c,...,0),...,(0,0,0,...,c)$$

ある次元 i に着目すると，i 番目の頂点の i 番目の要素は c で，それ以外の頂点の i 番目の要素は0である．ここで，全ての頂点について関数値を調べ，最も悪い関数値を持った頂点を j 番目の頂点であるとする．ここで，関数の単調性を考えてみると，頂点の j 番目の要素のインパクトが大きいために関数値が悪くなっているのではないかと予想できる．そこで，次元

j について単体を分割し，j 次元の変数値を最小値に固定した部分領域を切り出すことで，部分問題の関数値を小さくしたい．

疑似コードを Algorithm 1 に示した．

Algorithm 1 Decide cutting dimension of simplex

```
1:  function DecideCuttingDimension(simplex)
2:      Initialize worst_value
3:      Initialize dimension
4:      for each vertex of simplex do
5:          function_value := ObjectiveFunctionValue(vertex)
6:          if function_value ≥ worst_value then
7:              worst_value := function_value
8:              dimension := GetDimension(vertex)
9:          end if
10:     end for
11:     return vertex
12: end function
```

4.2 単体の保持

分枝限定法において，分割される元の領域を親ノードとし，分割によって生成される複数の部分領域を子ノードと考えると，分割は木構造となる．ここで，根ノードは元の問題の実行可能領域であり，アルゴリズムが進行するにつれてすべての葉ノードは分割され，木は大きくなる．

3.3 節の手法は単体を二つの部分単体に分割するため，二分木の構造を持つ．根ノードを単位単体としてアルゴリズムを開始し，その時点での葉ノード全てについて分割を繰り返すため，葉ノードの数は指数的に増大する．ゆえに大きなサイズの問題を解く場合，コンピュータのメモリ上に全ての葉ノードオブジェクトを保持し続けることは困難となる．この問題を解決するために，すべての葉ノードを保持するのでなく，分割の情報を表現する二分木を保持し，葉ノードが表す部分単体はその都度二分木の情報から再生成する．

4.2.1 二分木の構成

二分木のそれぞれのノードは単体を表現し，単体を分割するときにその単体を分割した次元をノードに保存する．分割によって二つの部分単体が生成されるが，一列減った部分単体は左の子ノードに，一次元減った部分単体は右の子ノードに挿入する．以上のルールに従って二分木を構成することで，二分木に保持された情報から，ある葉ノードの単体の形状を復元することができる．

TODO: 図を描く

4.2.2 単体の再生成

二分木の情報から，葉ノードが表す単体の形状を再現する．調べたい葉ノードから根ノードまで，親ノードをたどる．その際，左右どちらの子ノードであるかの情報と分割された次元を取得し，その二つの情報から葉ノードの形状を一意に再現できる．疑似コードを Algorithm 2 に示した．

Algorithm 2 Regenerate the simplex from the binary tree

```
1:  function REGENERATESIMPLEX(node)
2:      simplex := GENERATEUNITSIMPLEX()
3:      while parent := PARENT(node) do
4:          cutting_dimension := binary_tree[parent]
5:          if node is left child node then
6:              simplex := CUTONEROW(simplex, cutting_dimension)
7:          else if node is right child node then
8:              simplex := CUTONEDIMENSION(simplex, cutting_dimension)
9:          end if
10:         node := parent
11:     end while
12:     return simplex
13: end function
```

4.3 低次元の単体の処理

アルゴリズムが進行するにつれ，分割された単体の次元は小さくなる．そして，分割された単体の情報が保持される二分木を考えてみると，木の末端に近い部分では低次元の単体しか存在せず，木の末端に近いということはそれらの数がとても多いことが容易に想像できる．よって，低次元の単体を効率良く処理することはアルゴリズムの速度に対して重要である．

外崎ら [3] による研究では単体が 1 次元 (点) になるまで分割を行っていた．例えば 2 次元の単体 (線分) を分割するとき，その単体は一次元減った単体 (点) と一点が取り除かれた単体 (線分) に分割される．下界によって刈られる場合を除き，この処理を単体の分割数回 (m 回) 繰り返すことによってやっと全ての単体が点となりアルゴリズムが終了する．分割を繰り返すことでアルゴリズムのオーバーヘッドはかさむため，低次元の単体を単純なアルゴリズムで探索し，速度を向上したい．低次元の単体一つ一つにおける実行時間の改善は小さいものであるが，本手法で生成される低次元の単体の数はとても多いため，アルゴリズム全体として速度の大幅な改善が期待される．

分割が進み，単体の次元が 2 次元になった場合は単純に全てのグリッドの関数値を評価する．この処理は，部分単体が m 分割されている場合は $O(m)$ の処理となる．疑似コードを Algorithm 3 に示した．

Algorithm 3 Line search

```
1: function LINESEARCH(line)
2:     for each point of line do
3:         function_value := OBJECTIVEFUNCTIONVALUE(point)
4:         if function_value < current_optima then
5:             current_optima := function_value
6:         end if
7:     end for
8: end function
```

4.4 アルゴリズム

疑似コードは Algorithm 4 に示した通りとなる．

4.5 並列化

本アルゴリズムの並列化について述べる．分割した実行可能領域を保持する二分木における，同一列中のノードに相互に依存性はないため，列ごとに処理が完了していることを保証すればアルゴリズムを並列化できる．Algorithm 1 によってノードを分割する次元を計算したり，Algorithm 3 によって線形探索をしている間に，他のノードの処理をすることができるためアルゴリズム全体に対して速度の向上が図れるかもしれない．

ただし，$G(1, n, m)$ を実行可能領域とする問題に対して，Algorithm 1 のオーダーは高々 $O(n)$ で，Algorithm 3 は高々 $O(m)$ であるため，それほど重い処理ではない．よって問題の大きさによっては，大きな速度の改善が見られない，もしくは改悪することも考えうる．

具体的には，Algorithm 4 における 10 - 23 行目を並列化できる．

Algorithm 4 Optimize monotonic function on simplex

```
Initialize binary_tree
Initialize current_optima
i := 0
while true do
    nodes := i th row of binary_tree
    if nodes are empty then
        break
    end if
    for each node of nodes do
        simplex = REGENERATESIMPLEX(node)
        lower_bound := CALCULATELOWERBOUND(simplex)
        if current_optima ≤ lower_bound then
            continue
        end if
        if simplex is a line then
            LINESEARCH(simplex)
            continue
        end if
        if current_optima ≥ some vertices of simplex then
            Replace current_optima
        end if
        cutting_dimension = DECIDECUTTINGDIMENSION(simplex)
        binary_tree[node] := cutting_dimension
    end for
end while
return current_optima
```

第5章　数値実験

TODO: 書く

5.1　条件

5.2　実験結果

第6章　結論

TODO: 書く

謝辞

TODO: 書く

参考文献

[1] H. Tuy, Monotonic optimization: Problems and solution approaches, SIAM Journal on Optimization 11.2 (2000), pp. 464-494.

[2] de Klerk E, The complexity of optimizing over a simplex, hypercube or sphere: a short survey, Central European Journal of Operations Research 16.2 (2008), pp. 111-125.

[3] 外崎真造, 久野誉人, 非凸 2 次制約付き配合計画問題のロバスト最適化, 筑波大学システム情報工学研究科修士学位論文 (20010)