最適化手法

1 勾配による実数値最適化

1.1 前準備

本章では多次元の実数値最適化を扱う。また、最適化の対象である数値のことを解、またはパラメータと呼び、$\boldsymbol{x}$ のように書き表す。解が取り得る集合を V とする。問題が制約なし最適化の場合、$V = \mathbb{R}^n$ となる (n は解の次元)。本資料では一貫して、目的関数 $f : V \to \mathbb{R}$ の最小化が最適化だと考える。従って問題は $\min_{\boldsymbol{x} \in V} f(\boldsymbol{x})$ と定式化される。

$\boldsymbol{x}^* \in V$ が任意の $\boldsymbol{x} \in V$ に対して $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ となるとき、$\boldsymbol{x}^*$ のことを**大域的最適解**と呼ぶ。この $\boldsymbol{x}^*$ こそ私たちの知りたい値であり、最適化手法に要求する値である。しかしながら、世の手法のほとんどは大域的最適解の出力を保証してくれない。あくまで保証してくれるのは、「適切なハイパーパラメータの基で十分に計算したときに局所最適値に収束する」ことだけである。いま、$\boldsymbol{x}$ の ϵ 近傍 B を

$$B(\boldsymbol{x}, \epsilon) = \{\boldsymbol{y} \in V | ||\boldsymbol{x} - \boldsymbol{y}|| < \epsilon\} \tag{1.1}$$

と定義する。ある解 $\boldsymbol{x}^*$ と任意の解 $\boldsymbol{x} \in B(\boldsymbol{x}^*, \epsilon) \cap V$ に対して、$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ が成り立つとき、$\boldsymbol{x}^*$ のことを**局所最適解**と言う。従って如何なる最適化手法も、周囲と比べて適した解を提供してくれるに過ぎない (それでも十分嬉しかったりする)。

1.1.1 局所最適解の特徴

定理 1.1.1(1 次の必要条件)

$\boldsymbol{x}^*$ が局所最適解であるとき、$\boldsymbol{\nabla} f(\boldsymbol{x}^*) = \boldsymbol{0}$ が成立する。

証明：任意のベクトル $\boldsymbol{\delta}$ と正の非常に小さいスカラー t を考える。このとき局所最適解の条件より、$f(\boldsymbol{x}^* + t\boldsymbol{\delta}) - f(\boldsymbol{x}^*) \geq 0$ が成立する。そこで $t \to +0$ としたとき、テイラー展開より $\boldsymbol{\nabla} f(\boldsymbol{x}^*) \cdot \boldsymbol{\delta}$ が成立する。これは $\boldsymbol{\delta} = -\boldsymbol{\nabla} f(\boldsymbol{x}^*)$ でも成り立つので、$-\boldsymbol{\nabla} f(\boldsymbol{x}^*) \cdot \boldsymbol{\nabla} f(\boldsymbol{x}^*) \geq 0$ となるが、内積の定義より、$\boldsymbol{\nabla} f(\boldsymbol{x}^*) = \boldsymbol{0}$ が得られる。 □

$\boldsymbol{\nabla} f(\boldsymbol{x}) = \boldsymbol{0}$ となるような点を**停留点**と言う。停留点であることは局所最適値である十分条件ではない。例えば関数の極大値は停留点であるが局所最適値ではない。

次に 2 次の必要条件を考えるが、本資料ではヘッセ行列 $\boldsymbol{\nabla}(\boldsymbol{\nabla} f(\boldsymbol{x}))$ を $\boldsymbol{\nabla}^2 f(\boldsymbol{x})$ のように書く (なお、前者の書き方はダイアディクの定義に従う)。

定理 1.1.2(2 次の必要条件)

$\boldsymbol{x}^*$ が局所最適解であるとき、ヘッセ行列 $\boldsymbol{\nabla}^2 f(\boldsymbol{x}^*)$ は半正定値行列となる。

証明：任意のベクトル $\boldsymbol{\delta}$ と正の非常に小さいスカラー t を考える。$t \to +0$ としたとき、$f(\boldsymbol{x}^* + t\boldsymbol{\delta}) - f(\boldsymbol{x}^*) \geq 0$ が成立する。先ほど同様テイラー展開を施し、$\boldsymbol{\nabla} f(\boldsymbol{x}^*) = \boldsymbol{0}$ を考慮すれば、$\boldsymbol{\delta} \cdot (\boldsymbol{\nabla}^2 f(\boldsymbol{x}^*)\boldsymbol{\delta}) \geq 0$ なる関係が得られる。 □

定理 1.1.3(2 次の十分条件)

$\boldsymbol{x}^* \in V$ に対して、$\boldsymbol{\nabla} f(\boldsymbol{x}^*) = \boldsymbol{0}$ かつ $\boldsymbol{\nabla}^2 f(\boldsymbol{x}^*)$ が半正定値行列であるとき、$\boldsymbol{x}^*$ は局所最適値である。

1.1.2 最適化手法の停止条件

定理 1.3 より、最適化手法は勾配がゼロでヘッセ行列が半正定値行列となるような $\boldsymbol{x}^*$ を目指す。しかしながら数値計算的上、厳密に勾配がゼロとなるような点に収束することは難しいので、別途アルゴリズムの停止条件は必要である。

考えられる停止条件として最も単純なものは

$$||\boldsymbol{\nabla} f(\boldsymbol{x})|| < \epsilon$$

であろう。ここで ϵ はハイパーパラメータであるが、浮動小数点の誤差を考慮して 10^{-16} を採用することが比較的多い。

しかしながら、これは目的関数の出力のオーダーに依存する。例えば f が何か物理量を出力する場合、採用した SI 単位によって f の値は変わるが、それによって停止条件の”厳しさ”も変わるのは、ときに好ましくない。この対策として

$$||\boldsymbol{\nabla} f(\boldsymbol{x})|| < \epsilon |f(\boldsymbol{x})|$$

なる停止条件も提案された。ただし、これは $|f(\boldsymbol{x})|$ の値が非常に小さいときに厳しすぎる停止条件となるので、代わりに

$$||\boldsymbol{\nabla} f(\boldsymbol{x})|| < \epsilon (1 + |f(\boldsymbol{x})|)$$

を採用することもある。今では、下記の条件がすべて満たされたときに停止させることが多い。各停止条件を判定する Fortran の関数を以下に示す。

Listing 1.1 停止条件

```
1 function stop_condition1(gradf, epsilon) result(cond)
2         implicit none
3         real, intent(in) :: gradf(:)
4         real, optional, intent(in) :: epsilon
```

```
5           logical :: cond
6           real :: norm_gradf
7           norm_gradf = sqrt(sum(gradf**2))
8
9           if (present(epsilon)) then
10                  cond = (norm_gradf < epsilon)
11          else
12                  cond = (norm_gradf < 1.0e-16)
13          end if
14          return
15  end function stop_condition1
16
17  function stop_condition2(gradf, f, epsilon) result(cond)
18          implicit none
19          real, intent(in) :: gradf(:), f
20          real, optional, intent(in) :: epsilon
21          logical :: cond
22          real :: norm_gradf
23          norm_gradf = sqrt(sum(gradf**2))
24
25          if (present(epsilon)) then
26                  cond = (norm_gradf < epsilon*abs(f))
27          else
28                  cond = (norm_gradf < 1.0e-16*abs(f))
29          end if
30
31          return
32  end function stop_condition2
33
34  function stop_condition3(gradf, f, epsilon) result(cond)
35          implicit none
36          real, intent(in) :: gradf(:), f
37          real, optional, intent(in) :: epsilon
38          logical :: cond
39          real :: norm_gradf
40          norm_gradf = sqrt(sum(gradf**2))
41
42          if (present(epsilon)) then
43                  cond = (norm_gradf < epsilon*(1. + abs(f)))
44          else
45                  cond = (norm_gradf < 1.0e-16*(1. + abs(f)))
46          end if
47
48          return
49  end function stop_condition3
```

1.2 最適化手法の紹介

1.2.1 最急降下法

勾配を用いた最適化手法の中で最も単純なものは**最急降下法 (SD 法)** であろう。まず、解 $\boldsymbol{x} \in V$ と十分に小さいベクトル $\boldsymbol{\delta}(\boldsymbol{x}+\boldsymbol{\delta} \in V)$ を考える。$f(\boldsymbol{x}+\boldsymbol{\delta})$ をテイラー展開し、第一項までを考慮したならば、

$$f(\boldsymbol{x}+\boldsymbol{\delta}) \simeq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x}) \cdot \boldsymbol{\delta} \tag{1.2}$$

なる関係が得られる。SD 法は解 $\boldsymbol{x}$ を $\boldsymbol{x}+\boldsymbol{\delta}$ に更新し、それを繰り返すことで最終的に局所最適解に収束することを目指す。解の更新において式 (1.2) は重要なヒントである訳だが、$\boldsymbol{\delta}$ の大きさが小さいことを考えると、$f(\boldsymbol{x}+\boldsymbol{\delta})$ が最小となるのは、$\boldsymbol{\delta}$ と $-\boldsymbol{\nabla} f(\boldsymbol{x})$ が同じ向きであるときだと分かる。従って、十分に小さい実数 α に対して $\boldsymbol{\delta} = -\alpha \boldsymbol{\nabla} f(\boldsymbol{x})$ とし、解を

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \boldsymbol{\nabla} f(\boldsymbol{x}) \tag{1.3}$$

のように更新すればよい。以上が SD 法における考え方である。

繰り返しになるが、式 (1.2) の関係を仮定しているため、式 (1.3) 中の α の値は十分に小さくなければならない。α の値を決める方法として、数学的根拠のあるものがいくつか提案されているが、多くの場合は経験則的に決めてしまう。なお、α のことを**緩和係数**というが、人工知能の分野では学習率と呼ぶことの方が多い。

緩和係数が適切に設定された場合、式 (1.3) より解は勾配がゼロとなる点に収束する。したがって、SD 法は関数の局所最適解ではなく停留点に収束することを保証する。この保証はいささか不十分に聞こえるかもしれないが、SD 法が出力する停留点は局所最適解であることがほとんどなので、実用上の問題はない。

式 (1.3) より、SD 法はヘッセ行列を考えていないことが分かる。それゆえ局所最適解の収束を保証しなかった訳であるが、ヘッセ行列を考えない分だけ解の更新に要する計算コストが低いというメリットもある。それゆえ SD 法はいまでも利用されている。しかしながら、関数 f によっては探索方向に問題があることも指摘されており、解の更新の計算コストが低いといっても解が収束するまでの時間が短いとは限らないことも事実である。

最急降下法

1. 初期解 $\boldsymbol{x}_k \in \mathbb{R}^n$ と α を設定 (ただし $k = 0$)。
2. $\boldsymbol{x}_k$ が停止条件を満たす場合、$\boldsymbol{x}_k$ を出力して終了。
3. $\boldsymbol{d}_k \leftarrow -\alpha \boldsymbol{\nabla} f(\boldsymbol{x}_k)$。
4. $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \boldsymbol{d}_k$。
5. $k \leftarrow k + 1$。ステップ 2 に戻る。

以下に SD 法の Fortran コード例 (反復計算 1 回分) を示す。23 行目で grad というサブルーチンを呼び出しているが、これは関数 f の勾配をもとめるサブルーチンである (Listing1.3 参照)。

Listing 1.2　SD 法

```
1  subroutine sd(f, x, alpha, hb)
2          implicit none
3          real, intent(in) :: alpha
4          real, optional, intent(in) :: hb
5          real, intent(inout) :: x(:)
6          real :: h
7          real :: df(size(x))
8
9          interface
10         function f(x)
11                 implicit none
12                 real, intent(in) :: x(:)
13                 real :: f
14         end function f
15         end interface
16
17         if(present(hb)) then
18                 h = hb
19         else
20                 h = 0.001
21         end if
22
23         call grad(f, x, df, h)
24
25         x = x - alpha*df
26         return
27 end subroutine sd
```

Listing 1.3　勾配計算のためのサブルーチン

```
1  subroutine grad(f, x, df, hb)
2          real, intent(in) :: x(:)
3          real, optional, intent(in) :: hb
4          real, intent(out) :: df(size(x))
5          real :: dx(size(x)), h
6          integer :: i
7          interface
8                  function f(x)
9                          real, intent(in) :: x(:)
10                         real :: f
11                 end function f
12         end interface
13
14         if (present(hb)) then
15                 h = hb
16         else
17                 h = 0.001
18         end if
19
20         i = 1
21         do while(i <= size(x))
22                 dx = 0.; dx(i) = h
23                 df(i) = (f(x + dx) - f(x))/h
24                 i = i + 1
25         end do
26
27         return
28 end subroutine grad
```

1.2.2　ニュートン法

ニュートン法は最急降下法と違い、ヘッセ行列も加味する。2 次の項まで加味したテイラー展開より、

$$f(\boldsymbol{x} + \boldsymbol{\delta}) \simeq f(\boldsymbol{x}) + \boldsymbol{\nabla} f(\boldsymbol{x}) \cdot \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta} \cdot (\boldsymbol{\nabla}^2 f(\boldsymbol{x}) \boldsymbol{\delta})$$

なる関係が得られる。最急降下法のときと同様、$f(\boldsymbol{x} + \boldsymbol{\delta})$ がより減少するような向き $\boldsymbol{\delta}$ を知りたい。それは、上式を $\boldsymbol{\delta}$ で微分した結果より、

$$\boldsymbol{\delta} = -(\boldsymbol{\nabla}^2 f(\boldsymbol{x}))^{-1} \boldsymbol{\nabla} f(\boldsymbol{x})$$

だと分かる。ニュートン法ではこの向きに解を更新する。ただし、これはヘッセ行列が半正定値であることを仮定している。局所最適値付近では問題ないが、解の探索中は成り立たないこともある。そのようなとき、上式の更新方向は降下方向にならなかったりする。これはニュートン法の欠点であるため、修正された手法がいくつか提案されている。

ニュートン法

1. 初期解 $\boldsymbol{x}_k \in \mathbb{R}^n$ と α を設定 (ただし $k = 0$)。
2. $\boldsymbol{x}_k$ が停止条件を満たす場合、$\boldsymbol{x}_k$ を出力して終了。
3. $\boldsymbol{d}_k \leftarrow -\alpha(\boldsymbol{\nabla}^2 f(\boldsymbol{x}_k))^{-1}\boldsymbol{\nabla} f(\boldsymbol{x}_k)$。
4. $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \boldsymbol{d}_k$。
5. $k \leftarrow k + 1$。ステップ 2 に戻る。

最急降下法のときもそうだが、関数の勾配やヘッセ行列は一般的に数値差分で求められる。そのため次元 n が大きいとき、計算コストの面でヘッセ行列の数値計算には苦労する。多自由度な問題に対しニュートン法を利用することはお勧めしない。

以下はニュートン法の Fortran コード例である。また、ヘッセ行列を計算するためのサブルーチンを Listing1.5 に記載した。

Listing 1.4　ニュートン法

```
subroutine newton(f, x, alpha, hb)
        implicit none
        real, intent(in) :: alpha
        real, optional, intent(in) :: hb
        real, intent(inout) :: x(:)
        real :: h
        real :: ddf(size(x), size(x)), df(size(x))

        interface
        function f(x)
                implicit none
                real, intent(in) :: x(:)
                real :: f
        end function f
        end interface

        if(present(hb)) then
                h = hb
        else
                h = 0.001
        end if

        call hessian(f, x, ddf, h)
        call grad(f, x, df, h)

        x = x - alpha*matmul(ddf, df)
        return
end subroutine newton
```

Listing 1.5　ヘッセ行列を求めるためのサブルーチン

```
subroutine hessian(f, x, ddf, hb)
        real, intent(in) :: x(:)
        real, optional, intent(in) :: hb
        real, intent(out) :: ddf(size(x), size(x))
        real :: h, df(size(x)), dx(size(x)), dfp(size(x))
        integer :: i, j

        interface
                function f(x)
                        real, intent(in) :: x(:)
                        real :: f
                end function f
        end interface

        if (present(hb)) then
                h = hb
        else
                h = 0.001
        end if

        i = 1
        do while(i <= size(x))
                call grad(f, x, df, h)

                j = 1
                do while(j <= size(x))
                        dx = 0.; dx(j) = h
                        call grad(f, x + dx, dfp, h)
                        ddf(i, j) = (dfp(i) - df(i))/h
                        ddf(j, i) = ddf(i, j)
                        j = j + 1
```

```
32                    end do
33                    i = i + 1
34            end do
35            return
36    end subroutine hessian
```

2 線形計画問題

2.1 シンプレックス法

線形計画問題とは目的関数や制約条件が線形の最適化問題のことであり、

$$\min_{\boldsymbol{x}} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \quad \text{s.t.} \begin{cases} \boldsymbol{w}_{LE}^{(i)\mathrm{T}}\boldsymbol{x} \le b_i, & i \in \{1,2,...\} \\ \boldsymbol{w}_{GE}^{(j)\mathrm{T}}\boldsymbol{x} \ge b_j, & j \in \{1,2,...\} \\ \boldsymbol{w}_{EQ}^{(k)\mathrm{T}}\boldsymbol{x} = b_k, & k \in \{1,2,...\} \end{cases} \tag{2.1}$$

のように定式化できる。もしくは行列を用いることで制約条件を纏め、

$$\min_{\boldsymbol{x}} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \quad \text{s.t.} \begin{cases} A_{\mathrm{LE}}\boldsymbol{x} \le \boldsymbol{b}_{\mathrm{LE}} \\ A_{\mathrm{GE}}\boldsymbol{x} \ge \boldsymbol{b}_{\mathrm{GE}} \\ A_{\mathrm{EQ}}\boldsymbol{x} = \boldsymbol{b}_{\mathrm{EQ}} \end{cases} . \tag{2.2}$$

のように書いてもよい。ここで、任意の i に対して $x_i \le b_i$ であるとき、$\boldsymbol{x} \le \boldsymbol{b}$ と表記する。式 (2.2) の $=$ や $\ge$ についても同様である。式 (2.2) のような形式を**一般形**と言う。

定理 2.1.1

式 (2.2) の制約条件を満たす実行可能領域は凸多角形である。

証明：制約条件 $A_{\mathrm{LE}}\boldsymbol{x} \le \boldsymbol{b}_{\mathrm{LE}}$ について、これを満たす $\boldsymbol{x}$ の集合を S_{LE} とする。$\boldsymbol{x}, \boldsymbol{y} \in S_{\mathrm{LE}}$ に対して $\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y}$ を考える $(0 \le \alpha \le 1)$。これは $\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y} \le \boldsymbol{b}_{\mathrm{LE}}$ を満たすため $\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y} \in S_{\mathrm{LE}}$ である。従って $A_{\mathrm{LE}}\boldsymbol{x} \le \boldsymbol{b}_{\mathrm{LE}}$ を満たす解の集合は凸集合である。同様の流れで $A_{\mathrm{GE}}\boldsymbol{x} \ge \boldsymbol{b}_{\mathrm{GE}}$ を満たす解の集合 S_{GE} も凸集合である。明らかに $S_{\mathrm{LE}} \cap S_{\mathrm{GE}}$ は凸集合であり、制約 $A_{\mathrm{EQ}}\boldsymbol{x} = \boldsymbol{b}_{\mathrm{EQ}}$ はアフィン空間を意味している。従ってアフィン空間と $S_{\mathrm{LE}} \cap S_{\mathrm{GE}}$ の共通部分もまた凸集合である。制約式はいずれも超平面なので、この凸集合は凸多角形を成す。 □

線形計画問題の目的関数は凸関数である。従って凸集合の実行可能領域における凸関数の最適値探査であるため、線形計画問題の局所最適解はそのまま大域的最適解になる。

2.1.1 標準形

一般形は単純で私たちにとって問題を想像しやすい形となっている。しかしながら、本節で紹介するシンプレックス法は**標準形**と呼ばれる定式化を採用している。線形計画問題における標準形を以下に示す。

$$\min_{\boldsymbol{x}} \ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \quad \text{s.t.} \begin{cases} A\boldsymbol{x} = \boldsymbol{b} \\ \boldsymbol{x} \ge 0 \end{cases} . \tag{2.3}$$

このように、標準形の制約条件は変数が正であることと $A\boldsymbol{x} = \boldsymbol{b}$ の等式制約で構成されている。

不等式制約条件から等式制約条件を導くには別の変数を導入すればよい。例えば $A_{\mathrm{LE}}\boldsymbol{x} \le \boldsymbol{b}_{\mathrm{LE}}$ の場合を考えよう。いま $\boldsymbol{x} \in \mathbb{R}^n$、$\boldsymbol{b} \in \mathbb{R}^m$、$A_{\mathrm{LE}} \in \mathbb{R}^{m\times n}$ とする。ここに新しい変数 $\boldsymbol{y}_{\mathrm{LE}} (\ge \boldsymbol{0}) \in \mathbb{R}^m$ を導入し、$A_{\mathrm{LE}}\boldsymbol{x} + \boldsymbol{y}_{\mathrm{LE}} = \boldsymbol{b}_{\mathrm{LE}}$ なる制約であると考えることにする。このような変数 $\boldsymbol{y}_{\mathrm{LE}}$ のことを**スラック変数**と言う。

$A_{\mathrm{GE}}\boldsymbol{x} \ge \boldsymbol{b}_{\mathrm{GE}}$ の場合も同様である。例えば $\boldsymbol{b}_{\mathrm{GE}} \in \mathbb{R}^l$、$A_{\mathrm{GE}} \in \mathbb{R}^{l\times n}$ としたとき、新たな変数 $\boldsymbol{y}_{\mathrm{GE}} (\ge \boldsymbol{0}) \in \mathbb{R}^l$ を導入し、$A_{\mathrm{GE}}\boldsymbol{x} + \boldsymbol{y}_{\mathrm{GE}} = \boldsymbol{b}_{\mathrm{GE}}$ なる制約であると考えることにする。このような変数 $\boldsymbol{y}_{\mathrm{GE}}$ のことを剰余変数と言う。

$x_i (i = 1, ..., n)$ のうち本来制約を受けていなかった変数に対しても、別途変数を用意することで式 (2.3) の制約条件に落とし込む。x_i が制約なしである場合、$x_i' (> 0)$ 及び $x_i'' > 0$ なる新しい変数を用意して $x_i = x_i' - x_i''$ に置き換える。

以上より一般形から標準形に変換できることがわかった。以下は一般形を標準形に変換する例題である。

例題 2.1.1

下記線形計画問題の一般形を標準形に変換せよ。

$$\min_{\boldsymbol{x}} \ \begin{bmatrix} 5 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{s.t.} \begin{bmatrix} 2 & 3 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \le \begin{bmatrix} 5 \\ 11 \end{bmatrix}$$

まず初めに $2x_1 + 3x_2 + y_1 = 5$ 並びに $4x_1 + x_2 + y_2 = 11$ となるようなスラック変数 y_1 と y_2 を導入する。これにより、問題は

$$\min_{\boldsymbol{x}} \ \begin{bmatrix} 5 & 4 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} \quad \text{s.t.} \begin{bmatrix} 2 & 3 & 1 & 0 \\ 4 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}, \quad \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \ge \boldsymbol{0}$$

と書き換えられる。変数のうち x_1 と x_2 は制約なしなので、$x_1 = x_1' - x_1''$ 並びに $x_2 = x_2' - x_2''$ なる非負の変数を導入する。すると上式は

$$\min_{\boldsymbol{x}} \begin{bmatrix} 5 & -5 & 4 & -4 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1' \\ x_1'' \\ x_2' \\ x_2'' \\ y_1 \\ y_2 \end{bmatrix} \quad \text{s.t.} \begin{bmatrix} 2 & -2 & 3 & -3 & 1 & 0 \\ 4 & -4 & 1 & -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_1' \\ x_1'' \\ x_2' \\ x_2'' \\ y_1 \\ y_2 \end{bmatrix}, \quad \begin{bmatrix} x_1' \\ x_1'' \\ x_2' \\ x_2'' \\ y_1 \\ y_2 \end{bmatrix} \geq \mathbf{0}$$

となり、晴れて標準形に変換された。

2.1.2 基底解と実行可能基底解

ここからは 2.1.1 節の次元数を忘れ、$\boldsymbol{x} \in \mathbb{R}^n$、$\boldsymbol{b} \in \mathbb{R}^m$、$A \in \mathbb{R}^{m \times n}$ と定義し直す。また、本節では $n > m$ であること、並びに $\mathrm{rank} A = m$ であることを仮定する。後者の仮定は、冗長な制約条件がないことを意味している。

式 (2.3) の変数 x_i のうち m 個を選択し、それによってできるベクトルを $\boldsymbol{x}_B \in \mathbb{R}^m$ とする。また、$\boldsymbol{x}_B$ に含まれない変数が成すベクトルを $\boldsymbol{x}_N \in \mathbb{R}^{n-m}$ とする。$\boldsymbol{x}_B$ と $\boldsymbol{x}_N$ に合わせて $\boldsymbol{x}$ の要素の順番を、$\boldsymbol{x} = (\boldsymbol{x}_B \ \boldsymbol{x}_N)^{\mathrm{T}} \in \mathbb{R}^n$ となるようにを並び換える。また、A の行ベクトルの順番も $\boldsymbol{x}_B$ と $\boldsymbol{x}_N$ に合わせて並び換える。$\boldsymbol{x}_B$ 中の変数に対応する行ベクトル要素を行列 $B \in \mathbb{R}^{m \times m}$ に纏め、$\boldsymbol{x}_N$ に対応する行ベクトルを $N \in \mathbb{R}^{(n-m) \times (n-m)}$ に纏めたとき、制約条件は

$$\begin{bmatrix} B & N \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_B \\ \boldsymbol{x}_N \end{bmatrix} = \boldsymbol{b} \tag{2.4}$$

のように書くことができる。

B が正則であるとき、$\boldsymbol{x}_B$ は

$$\boldsymbol{x}_B = B^{-1}\boldsymbol{b} - B^{-1}N\boldsymbol{x}_N \tag{2.5}$$

となる。$\boldsymbol{x}_B$ は $\boldsymbol{x}_N$ に依存するが、$\boldsymbol{x}_N = \mathbf{0}$ としたときの解 $\boldsymbol{x}_B = B^{-1}\boldsymbol{b}$、つまり $\boldsymbol{x} = (B^{-1}\boldsymbol{b}\ \mathbf{0})^{\mathrm{T}}$ は特別に**基底解**という名前がついている。更に $\boldsymbol{x}_B = B^{-1}\boldsymbol{b} \geq \mathbf{0}$ が成り立つならば、基底解は式 (2.3) の制約条件を満たすので、この $\boldsymbol{x}$ を**実行可能基底解**と言う。基底解に対して B のことを**基底行列**、$\boldsymbol{x}_B$ 中の各変数のことを**基底変数**、N を**非基底行列**、$\boldsymbol{x}_N$ 中の各変数のことを**非基底変数**と言う。

命題 2.1.1

実行可能基底解は有限個しかない。

証明：(一般的に実行可能解は無数に存在するが、一方で)$\boldsymbol{x}$ の分割方法は高々 ${}_nC_m$ 通りであるから、実行可能基底解は必ず有限個である。 □

ここからは最適値が存在する問題に限定して議論する。例えば $f(x) - x^2 (x > 0)$ を最小化する x という問題は、目的関数を限りなく最小化できるため、最適解が存在しない。また、$x > 0$ かつ $x < -1$ という制約の場合はそもそも実行可能解が存在しない。最適値が存在するとき、定理 2.1.1 より実行可能領域は凸多面体を成す。

実行可能基底解の性質について議論するために、

$$\begin{cases} 2x_1 + x_2 \leq 6 \\ x_1 + 2x_2 \leq 6 \\ x_i \geq 0 \ (i = 1, 2) \end{cases} \tag{2.6}$$

で表される制約条件を例題として扱おう。このときの実行可能解を図 2.1 に示す。これを標準形で書き換えると

$$\begin{cases} 2x_1 + x_2 + x_3 = 6 \\ x_1 + 2x_2 + x_4 = 6 \\ x_i \geq 0 \ (i = 1, 2, 3, 4) \end{cases} \tag{2.7}$$

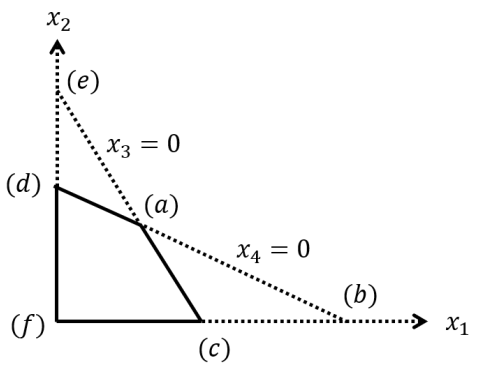

図 2.1　式 (2.6) もしくは式 (2.7) の制約条件。実行可能解は実線で表された凸多面体に内包される。点 $(a) - (f)$ は式 $(a) - (f)$ の基底解に対応。

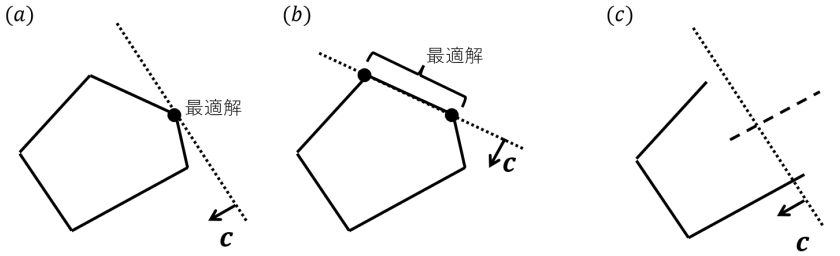

図 2.2　線形計画問題の最適解。

となる。この基底解は容易に求められ、

$$[x_1 \quad x_2 \quad x_3 \quad x_4] = [2 \quad 2 \quad 0 \quad 0] \tag{a}$$

$$[x_1 \quad x_2 \quad x_3 \quad x_4] = [6 \quad 0 \quad -6 \quad 0] \tag{b}$$

$$[x_1 \quad x_2 \quad x_3 \quad x_4] = [3 \quad 0 \quad 0 \quad 3] \tag{c}$$

$$[x_1 \quad x_2 \quad x_3 \quad x_4] = [0 \quad 3 \quad 3 \quad 0] \tag{d}$$

$$[x_1 \quad x_2 \quad x_3 \quad x_4] = [0 \quad 6 \quad 0 \quad -6] \tag{e}$$

$$[x_1 \quad x_2 \quad x_3 \quad x_4] = [0 \quad 0 \quad 6 \quad 6] \tag{f}$$

である。図 2.1 に示す通り、基底解は 2 本の直線が交わる点である。また、$(a)-(f)$ のうち実行可能基底解は $(a)(c)(d)(f)$ であり、これらは凸多面体の頂点である。

さて、図 2.2 に示す通り目的関数の等高線は凸多面体に対する超平面になる。したがって線形計画問題の最適解について以下 3 つのことが言える。

- 図 2.2(a) の場合、つまり最適値をとる目的関数と多面体が頂点で接するとき、ある 1 つの実行可能基底解が最適解となる。
- 図 2.2(b) の場合、つまり最適値をとる目的関数と多面体が面で接するとき、最適解は無数に存在する。ただしこのときも幾つかの実行可能基底解は最適解に含まれる。
- 図 2.2(c) の場合は最適解がなく、本節の議論の対象外である。
- 図 2.2 に図持していないが、実行可能解が存在しない場合はこういった多面体が存在しない場合である。

定理 2.1.2

線形計画問題が最適解を持つならば、実行可能基底解のなかに最適解が必ず存在する。

2.1.3　シンプレックス法

定理 2.1.2 より線形計画問題を見つける際は有限個の実行可能基底解を探索すればよいことが分かる。本節で紹介する**シンプレックス法**は、より小さな目的関数値をもつ実行可能基底解を次々に生成し、最終的に最適基底解に到達しようとする反復法である。

そこで、標準形の問題に対して実行可能基底解を与える分割 $\boldsymbol{x} = (\boldsymbol{x}_B \ \boldsymbol{x}_N)^{\mathrm{T}}$ 並びに $A = (B \ N)$ が既に 1 つ与えられているとする (実行可能基底解を得る一般的な方法は後述する)。なお、あくまで実行可能基底解を与える分割が既知なだけで、$\boldsymbol{x}$ は実行可能基底解でない。つまり、今は $\boldsymbol{x}_N \neq 0$ である。B は正則なため、$\boldsymbol{x}_B$ は式 (2.5) より求まる。式 (2.2) の目的関数について $\boldsymbol{c}$ も $(\boldsymbol{c}_B \ \boldsymbol{c}_N)^{\mathrm{T}}$ のように分割したとき ($\boldsymbol{c}_B \in \mathbb{R}^m$、$\boldsymbol{c}_N \in \mathbb{R}^{n-m}$)、目的関数値 z は

$$z = \boldsymbol{c}_B^{\mathrm{T}}\boldsymbol{x}_B + \boldsymbol{c}_N^{\mathrm{T}}\boldsymbol{x}_N = \boldsymbol{c}_B^{\mathrm{T}}B^{-1}\boldsymbol{b} + (\boldsymbol{c}_N^{\mathrm{T}} - \boldsymbol{c}_B^{\mathrm{T}}B^{-1}N)\boldsymbol{x}_N = z_0 + \sum_{j=m+1}^{n}(c_j - z_j)x_j \tag{2.8}$$

となる。ここで

$$z_0 = \boldsymbol{c}_B^{\mathrm{T}}B^{-1}\boldsymbol{b}, \quad z_j = \boldsymbol{c}_B^{\mathrm{T}}B^{-1}\boldsymbol{a}_j$$

であり、$\boldsymbol{a}_j$ は A の第 j 列ベクトルである。z_0 はこの分割における実行可能基底解 $(\boldsymbol{x}_B \ \boldsymbol{0})$ に対する目的関数値である。式 (2.8) より、z は x_j に対して $c_j - z_j$ の感度で変化する。この $(c_j - z_j)$ を x_j の**相対コスト係数**と言う。

分割 $(\boldsymbol{x}_B \ \boldsymbol{x}_N)^{\mathrm{T}}$ および $(B \ N)$ に対して $j \in \{m+1, ..., n\}$ の相対コスト係数がいずれも非負である場合、$x_j \geq 0$ 並びに式 (2.8) より $z \geq z_0$ となる。従って全ての相対コスト係数が非負であるとき、この分割に対する実行可能基底解 $\boldsymbol{x} = (B^{-1}\boldsymbol{b} \ \boldsymbol{0})^{\mathrm{T}}$ は最適解であり、最適値 $z_0 = \boldsymbol{c}_B^{\mathrm{T}}B^{-1}\boldsymbol{b}$ を返す。

一方で負の相対コスト係数 $(c_p - z_p) < 0$ となる $p \in \{m+1, ..., n\}$ が存在する場合を考える。x_p 以外の $x_j \in \{m+1, ..., n\}$ をゼロとし、$x_p = \Delta (> 0)$ とする。このときの解

$$\boldsymbol{x}_\Delta = \begin{bmatrix} \boldsymbol{x}_b \\ x_{m+1} \\ \cdot \\ \cdot \\ \cdot \\ x_p \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} B^{-1}\boldsymbol{b} \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ \Delta \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \tag{2.9}$$

に対する目的関数値は

$$\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}_\Delta = z_0 + (c_p - z_p)\Delta \leq z_0 \tag{2.10}$$

となる。目的に従い目的関数は減少するため、実行可能基底解 $(B^{-1}\boldsymbol{b}\ \boldsymbol{0})^{\mathrm{T}}$ から式 (2.9) のような解の更新を考える。ただし、更新後も制約条件 $A\boldsymbol{x} = \boldsymbol{b}$ は満たされないといけないので、$\boldsymbol{x}_N$ から $\boldsymbol{x}'_N = (0\ 0 ... \Delta ... 0)^{\mathrm{T}}$ と変えたのならば、式 (2.5) より $\boldsymbol{x}_B$ も $\boldsymbol{x}_B = B^{-1}\boldsymbol{b}$ から

$$\boldsymbol{x}'_B = B^{-1}\boldsymbol{b} - B^{-1}\boldsymbol{a}_p\Delta \tag{2.11}$$

に変えなければならない。

また、$\boldsymbol{x}'_B \geq 0$ でなければならないことも重要である。いま B は実行可能基底解を与えるような分割における基底行列であるため、$\boldsymbol{x}_B = B^{-1}\boldsymbol{b} \geq 0$ である。一方で $B^{-1}\boldsymbol{a}_p \leq 0$ の場合、任意の正の実数 Δ において式 (2.11) の $\boldsymbol{x}'_B$ は制約条件 $\boldsymbol{x}'_B \geq 0$ を満たす。式 (2.10) より、Δ の増加とともに目的関数値は減少するため、Δ の値に上限がないならば目的関数値にも下限がないことになる。従って $B^{-1}\boldsymbol{a}_p \leq 0$ なる p が存在する場合、この線形計画問題は有界な解を持たない。この判定は計算終了条件として重要である。

p に対して $(B^{-1}\boldsymbol{a}_p)_i > 0$ となるような $i \in \{1, ..., m\}$ が存在する場合、線形計画問題は有界な解を持たないとは限らない。Δ の増加とともに目的関数値は減少することを思い出すと、式 (2.11) の $\boldsymbol{x}'_B$ が非負の制約条件を満たす最大の Δ

$$\Delta = \min\left\{ \frac{(B^{-1}\boldsymbol{b})_i}{(B^{-1}\boldsymbol{a}_p)_i} \middle|\ (B^{-1}\boldsymbol{a}_p)_i > 0 \quad i \in \{1, ..., m\} \right\} \tag{2.12}$$

で解を更新すれば効率がよいと分かる。このとき新しい解 $\boldsymbol{x}'$ は

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{x}'_B \\ \boldsymbol{x}'_N \end{bmatrix} = \begin{bmatrix} B^{-1}\boldsymbol{b} - B^{-1}\boldsymbol{a}_p\Delta \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ \Delta \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \tag{2.13}$$

となる。$i = r (\in \{1, ..., m\})$ のとき式 (2.12) の右辺が最小となる場合、つまり $\Delta = (B^{-1}\boldsymbol{b})_r / (B^{-1}\boldsymbol{a}_p)_i$ である場合、式 (2.13) の x'_r はゼロとなる。したがって $\boldsymbol{x}'$ は少なくとも $n - m$ 個のゼロ要素を持つ。

シンプレックス法は次の解の更新時に、$n - m$ 個のゼロである変数が非基底変数となるように再度分割処理を施す。これは r 番目の要素と p 番目の要素を入れ替えるだけで済む。このとき A や $\boldsymbol{c}$ の並び変えも必要であることに注意しなければならない ($\boldsymbol{b}$ は不要)。再分割により基底行列は $(\boldsymbol{a}_1\ \boldsymbol{a}_2 ... \boldsymbol{a}_m)$ から $(\boldsymbol{a}_1\ \boldsymbol{a}_2 ... \boldsymbol{a}_{r-1}\ \boldsymbol{a}_p\ \boldsymbol{a}_{r+1} ... \boldsymbol{a}_m)$ に変わるが、再分割後の基底行列も正則であることは保証されている (証明は省く)。このような基底変数の入れ替えに伴う処理を**ピボット演算**と言う。以下にシンプレックス法のアルゴリズムを纏めた。

シンプレックス法

入力として実行可能基底解を与えるような分割が既に得られているとする。

1. $j \in \{m+1, ..., n\}$ に対し相対コスト係数 $c_j - z_j = c_j - \boldsymbol{c}_B^{\mathrm{T}} B^{-1}\boldsymbol{a}_j$ を計算する。
2. 任意の j に対して $c_j - z_j \geq 0$ ならば、$(B^{-1}\boldsymbol{b}\ \boldsymbol{0})^{\mathrm{T}}$ を最適解として出力し、計算を終了する。
3. 2. で計算が終了しなかった場合、$c_p - z_p < 0$ となる $p \in \{m+1, ..., n\}$ を一つ選択する。
4. $B^{-1}\boldsymbol{a}_p \leq \boldsymbol{0}$ の場合、この問題は有界な解を持たないと判断し、計算を終了する。
5. 4. で計算が終了しなかった場合、$\Delta \in \mathbb{R}$ に $\min\left\{(B^{-1}\boldsymbol{b})_i / (B^{-1}\boldsymbol{a}_p)_i \middle|\ (B^{-1}\boldsymbol{a}_p)_i > 0 \quad i \in \{1, ..., m\}\right\}$ を代入する。また、右辺について最小となる要素番号を $r \in \{1, ..., m\}$ とする。
6. 解 $\boldsymbol{x} = (B^{-1}\boldsymbol{b}\ \boldsymbol{0})$ を $\boldsymbol{x}' = (B^{-1}\boldsymbol{b} - B^{-1}\boldsymbol{a}_p\Delta\ 0 ... \Delta ... 0)^{\mathrm{T}}$ に更新する (ここで p 番目要素の値が Δ)。
7. $\boldsymbol{x}' = (x'_1 ... x'_n)^{\mathrm{T}}$ を $(x'_1 ... x'_{r-1}\ x'_p\ x'_{r+1} ... x'_m\ 0 ... 0)^{\mathrm{T}}$ となるように入れ替える。
8. 7. に伴い $\boldsymbol{c} = (b_1 ... c_n)^{\mathrm{T}}$ も $(c_1 ... c_{r-1}\ c_p\ c_{r+1} ... c_{p-1}\ c_r\ c_{p+1} ... c_n)^{\mathrm{T}}$ となるように入れ替える。
9. 7. に伴い A も $(\boldsymbol{a}_1\ \boldsymbol{a}_2 ... \boldsymbol{a}_n)$ から $(\boldsymbol{a}_1 ... \boldsymbol{a}_{r-1}\ \boldsymbol{a}_p\ \boldsymbol{a}_{r+1} ... \boldsymbol{a}_{p-1}\ \boldsymbol{a}_r\ \boldsymbol{a}_{p+1} ... \boldsymbol{a}_n)^{\mathrm{T}}$ となるように入れ替える。
10. $\boldsymbol{x}'$ とピボット演算後の A を初期条件として、1. に戻る。

ピボット演算により変数の順番が変わるので、プログラム上ではそれを追跡する処理も必要となる。

シンプレックス法アルゴリズムの 3. について、相対コスト係数が負であるならば p の選択は任意である。考え得る選択の一つとして、相対コスト係数が最小となる p を選択することである。つまり、

$$p = \mathrm{argmin}\,\{c_j - z_j | j \in \{m+1, ..., n\}\} \tag{2.14}$$

となるように選択する。もちろん上式の最小値探査が計算コスト的に困難である場合は他の指針に従うべきであろう。

2.1.4 理論的な収束性

前述の通り、相対コスト係数 $c_p - z_p$ が負となる非基底変数 x_p が見つかった場合、目的関数値は $|c_p - z_p|\Delta$ だけ減少する。現段階の解 $\boldsymbol{x} = (B^{-1}\boldsymbol{b}\ \boldsymbol{0})^{\mathrm{T}}$ に対して、制約条件より $B^{-1}\boldsymbol{b} \geq \boldsymbol{0}$ が成立する。そのため有界な解が存在する問題ならば、式 (2.12) より $\Delta \geq 0$ だと言える。とくに $B^{-1}\boldsymbol{b} > 0$ であるならば $\Delta > 0$ となり、解の更新によって目的関数値は狭義に減少する。線形計画問題の場合、解に対して目的関数値は一意に定まるので、シンプレックス法の反復計算中常に $B^{-1}\boldsymbol{b} > 0$ が成立するならば、更新されていく実行可能基底解の中に同一のものは存在しないはずである。実行可能基底解は有限個であることを思い出すと、このときシンプレックス法は有限回の反復計算で最適解に到達する。実行可能基底解 $(\boldsymbol{x}_B\ \boldsymbol{x}_N)^{\mathrm{T}} = (B^{-1}\boldsymbol{b}\ \boldsymbol{0})^{\mathrm{T}}$ が $B^{-1}\boldsymbol{b} > \boldsymbol{0}$ を満たすとき、この解のことを非退化であると言う。逆に非退化でない解のことを退化していると言う。

定理 2.1.3

与えられた線形計画問題の初期実行可能基底解を求めることができ、しかも計算の途中に現れる実行可能基底解が全て非退化である場合、シンプレックス法は有限回の反復で最適解を見つける (もしくは有界な解がないことを識別して計算を終了する)。

退化の例として以下の問題を考える。

$$\min_{\boldsymbol{x}} \begin{bmatrix} -3 & -2 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \quad \text{s.t.} \quad \begin{bmatrix} 2 & 1 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}, \quad \boldsymbol{x} \geq \boldsymbol{0}. \tag{2.15}$$

この問題の実行可能基底解として $\boldsymbol{x}_B = (x_1, x_2, x_4)^{\mathrm{T}}$、$\boldsymbol{x}_B = (x_1, x_3, x_4)^{\mathrm{T}}$、$\boldsymbol{x}_B = (x_1, x_4, x_5)^{\mathrm{T}}$ が考えられるが、結局はいずれも $(3, 0, 0, 4, 0)^{\mathrm{T}}$ であり、これは退化している。また、(x_1, x_2)-平面上で見たとき、この実行可能基底解は 3 本の直線の交わる点に対応している (図 2.3 の (3, 0))。

シンプレックス法の反復計算中に退化した実行可能基底解が現れると、式 (2.12) の Δ はゼロになる。そこでピボット演算をした場合、アルゴリズムの性質上基底変数と非基底変数の組み合わせは変わるが、解 $\boldsymbol{x}$ は実質的に動かず目的関数値も変化しない。また、基底に入る変数 p が一意に定まらず、同時に基底から出る変数 r も一意に定まらない場合、それらの選び方によっては何回かのピボット演算で同じ分割に戻ってしまう可能性もある。このような現象を**循環**と言う。循環が生じたときのシンプレックス法は同じ計算を何度も繰り返すことになり、最適解に到達できない。

循環という危険性を持っている点で、シンプレックス法は欠点のある手法と思われるかもしれない。ところが現実の問題で循環に出会うことはまずない。確かに退化が生じることは稀でないが、退化は循環の必要条件であって十分条件でなく、退化が生じたからと言って循環に神経質にならなくてもよい。

とはいえ運悪く循環が生じたときは困るので、いくつか循環を回避するための手法が研究された。その中でも最も有名なのが**最小添字規則**である。これは p や r の選択時、候補の中から最も添字小さいものを選ぶという規則であり。こうするだけでシンプレックス法は循環を確実に回避できる。ただし p の選択に関しては、前述の通り式 (2.14) に従う選択の方が反復回数を抑えることができるので、最小添字規則がそのまま使われることは少ない。

2.1.5 ビッグ M 法

シンプレックス法は入力として実行可能基底解を与える分割を必要とする。これまでの問題では自明な実行可能基底解が存在したが、現実の問題で手計算によって実行可能基底解が得られることは稀であり、何か対策が必要である。本節ではプログラム実装が容易なビッグ M 法を紹介する。

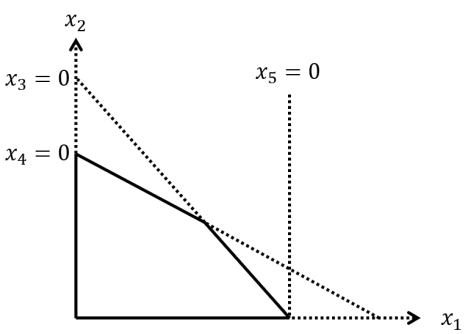

図 2.3　式 (2.15) の実行可能領域。

ここで、対象の線形計画問題は既に標準形に変換されているとする。また $\boldsymbol{b} \geq 0$ だと仮定する。仮に $b_i < 0$ であったとしても、両辺に (-1) を掛ければよいため、この仮定は一般性を欠かない。ビッグ M 法では元の線形計画問題 (2.3) の代わりに、

$$\min_{(\boldsymbol{x}\ \boldsymbol{y})^{\mathrm{T}}} \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} + M\boldsymbol{d}^{\mathrm{T}}\boldsymbol{y} \quad \text{s.t.} \quad A\boldsymbol{x} + \boldsymbol{y} = \boldsymbol{b}, \quad \boldsymbol{x} \geq 0, \quad \boldsymbol{y} \geq 0 \tag{2.16}$$

を考える。ここで $\boldsymbol{d} \in \mathbb{R}^m$ は全ての要素が 1 なベクトルで、$M\ in \mathbb{R}$ はハイパーパラメータである。また $\boldsymbol{y}$ は**人為変数**と言い、$\boldsymbol{x}$ と同様に変数として考える。この線形計画問題は明らかに実行可能基底解 $(\boldsymbol{x}\ \boldsymbol{y})^{\mathrm{T}} = (\boldsymbol{0}\ \boldsymbol{b})^{\mathrm{T}}$ を持つ。

M の値を十分に大きく設定した場合、目的関数に対して $M\boldsymbol{d}^{\mathrm{T}}\boldsymbol{y}$ が支配的になり、シンプレックス法はまず第 2 項を減少させようと働く。その結果 $\boldsymbol{y} = \boldsymbol{0}$ となれば、線形計画問題 (2.16) は (2.3) と実質的に同じになる。従って、最終的に得られる解から $\boldsymbol{x}$ に関わる要素を抽出すれば、所望の最適解が得られる訳である。

M の値が十分に大きくないとき、(2.16) の最適解に関して $\boldsymbol{y} = \boldsymbol{0}$ とならずに収束してしまうことがある。このときは M の値を更に大きくして再計算しなければならない。M を大きくし続けても $\boldsymbol{y} = \boldsymbol{0}$ とならずに収束するならば、元の問題 (2.3) は実行可能解を持たないと判断し、計算を終了する。

2.1.6 python コード例

シンプレックス法の実装は数値計算の中だと比較的難しいと個人的に感じている。一方でシンプレックス法は研究しつくされた分野であり、既に多くのライブラリが開発されている。本節では scipy を利用したコード例を紹介する。

扱う例題は

$$\min\ -3x_1 - 2x_2 \quad \text{s.t. } 2x_1 + x_2 \leq 6,\ x_1 + 2x_2 \leq 6,\ \boldsymbol{x} \geq 0$$

とする。scipy のモジュールは前もって標準形に変換することも実行可能基底解を用意することも要求しない。上式の目的関数と制約条件を入力するだけで解が得られる仕様となっている。モジュールの使い方は scipy のマニュアルに譲り、本資料では python コードを記すだけとした。

Listing 2.1 Python によるシンプレックス法コード例

```
1  import numpy as np
2  from scipy.optimize import linprog
3
4  c = np.array([-3., -2.])
5  Aub = np.array([
6          [2., 1.],
7          [1., 2.]])
8  bub = np.array([6.,6.])
9  bounds = np.array([0., None])
10 result = linprog(c, Aub, bub, bounds = bounds, method = "simplex")
11 x = result.x
```