数学・計算機科学

北原 鉄朗

平成 13 年 9 月 13 日

目 次

第 1 章　線形代数 **5**
1.1　線形空間 . . . 5
1.2　線形独立と線形従属 . . . 5
1.3　基底と次元 . . . 5
1.4　正規直交系 . . . 6
1.5　線形写像 . . . 6
1.6　行列式 . . . 7
1.7　逆行列 . . . 8
1.8　連立一次方程式 . . . 9
1.9　固有値と固有ベクトル . . . 10
1.10 特別な行列 . . . 11
1.11 行列の対角化 . . . 12

第 2 章　微分方程式 **13**
2.1　1 階常微分方程式 . . . 13
2.2　2 階線形常微分方程式 . . . 14

第 3 章　解析学 **15**
3.1　級数の和 . . . 15
3.2　微分 . . . 16

第 4 章　複素解析 **17**
4.1　複素微分 . . . 17
4.2　複素積分 . . . 17
4.3　級数 . . . 19
4.4　留数 . . . 19
4.5　定積分 . . . 20

第 5 章　信号処理 **21**
5.1　連続時間のフーリエ変換 . . . 21
5.2　離散フーリエ変換 . . . 21
5.3　z 変換 . . . 22

第 6 章　エントロピー **23**
6.1　エントロピーの定義 . . . 23

第 7 章　数理論理学 **25**
7.1　古典命題論理 . . . 25
7.2　古典述語論理 . . . 26
7.3　直観主義命題論理 . . . 26
7.4　様相論理 . . . 27

第 8 章　数理計画法 **29**
8.1　用語 . . . 29

第 9 章　データ構造とアルゴリズム **31**
9.1　アルゴリズム概論 . . . 31
9.2　線形リスト . . . 32
9.3　スタック . . . 32
9.4　待ち行列 (キュー) . . . 33
9.5　木構造 (概論) . . . 33
9.6　順序木 . . . 33
9.7　2 分木 . . . 33
9.8　2 分探索木 . . . 34
9.9　AVL 木 . . . 35
9.10 B 木 . . . 36
9.11 半順序の付いた木 . . . 36
9.12 集合 . . . 37
9.13 優先度付き待ち行列 . . . 37
9.14 ソート法 . . . 38
9.15 探索 . . . 40
9.16 文字列照合 . . . 41

第 10 章 形式言語 **43**
10.1 文法と言語 . . . 43
10.2 文脈自由文法 . . . 44
10.3 有限オートマトン . . . 44
10.4 プッシュダウン・オートマトン . . . 45

第 11 章 オペレーティングシステム **47**
11.1 プロセスの相互交渉 . . . 47
11.2 排他制御 (概論) . . . 47
11.3 セマフォア . . . 48
11.4 セマフォアの応用 . . . 49
11.5 モニタ . . . 50
11.6 モニタの応用 . . . 51
11.7 マルチプログラミング (概論) . . . 51
11.8 オーバレイ . . . 52
11.9 セグメンテーション . . . 52
11.10ページング . . . 52

第 12 章 数値解析 **53**
12.1 ニュートン法 53

第 13 章 パターン認識の基礎 **55**
13.1 パターン認識概論 55
13.2 学習 55
13.3 ノンパラメトリックな学習 56
13.4 パラメトリックな学習 57
13.5 特徴空間の変換 57

第 14 章 人工知能の基礎 **59**
14.1 探索 59
14.2 導出原理 61
14.3 ホーン節 62

第1章　線形代数

1.1　線形空間

定義 1.1.1 (線形空間) 空でない集合 V 上に, 和 $\boldsymbol{x}+\boldsymbol{y}$ とスカラー倍 $a\boldsymbol{x}$ が定義され, 次の条件を満たすとき, V を線形空間 (lineary space) という. ここで, $\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}\in V$, $a,b\in\mathbf{K}$ である.

(1) (可換則) $\boldsymbol{x}+\boldsymbol{y}=\boldsymbol{y}+\boldsymbol{x}$;
(2) (結合則) $(\boldsymbol{x}+\boldsymbol{y})+\boldsymbol{z}=\boldsymbol{x}+(\boldsymbol{y}+\boldsymbol{z})$;
(3) 零ベクトルが存在する,
すなわち, $\exists\mathbf{0}\in V:\boldsymbol{x}+\mathbf{0}=\boldsymbol{x}$;
(4) 逆ベクトルが存在する,
すなわち, $\exists\boldsymbol{x'}\in V:\boldsymbol{x}+\boldsymbol{x'}=\mathbf{0}$;
(5) $a(\boldsymbol{x}+\boldsymbol{y})=a\boldsymbol{x}+a\boldsymbol{y}$;
(6) $(a+b)\boldsymbol{x}=a\boldsymbol{x}+b\boldsymbol{y}$;
(7) $(ab)\boldsymbol{x}=a(b\boldsymbol{x})$;
(8) $1\boldsymbol{x}=\boldsymbol{x}$.

定義 1.1.2 (部分空間) 線形空間 V の部分集合 W が, 線形空間となるとき, W を V の部分空間という.

定理 1.1.1 線形空間 V の部分集合 W が V の部分空間であるための必要十分条件は,

$$\forall a,b\in\mathbf{K},\forall\boldsymbol{x},\boldsymbol{y}\in W:a\boldsymbol{x}+b\boldsymbol{y}\in W$$

が成り立つことである.

1.2　線形独立と線形従属

定義 1.2.1 (線形独立) 線形空間 V の部分空間 $\{\boldsymbol{x}_1,\cdots,\boldsymbol{x}_n\}$ が, $\forall a_1,\cdots,a_n\in\mathbf{K}$ に対して,

$$a_1\boldsymbol{x}_1+\cdots+a_n\boldsymbol{x}_n=0\Longrightarrow a_1=\cdots=a_n=0$$

を満たすとき, $\{\boldsymbol{x}_1,\cdots,\boldsymbol{x}_n\}$ は線形独立 (linearly independent) という.

定義 1.2.2 (線形従属) 線形独立の否定を線形従属 (lineraly dependent) という.

例 1.2.1 $\boldsymbol{x}:=(1,-1,4)^t$, $\boldsymbol{y}:=(2,3,-1)^t$, $\boldsymbol{z}:=(0,-5,9)^t$ が線形独立かどうかを調べる.
$a\boldsymbol{x}+b\boldsymbol{y}+c\boldsymbol{z}=0$ とおくと,

$$\begin{cases}a+2b=0\\-a+3b-5c=0\\4a-b+9c=0\end{cases}$$

$a=2,b=-1,c=-1$ とすれば上の連立方程式が成立するので, $\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$ は, 線形従属である.

1.3　基底と次元

定義 1.3.1 (基底) 線形空間 V の部分空間 $\{\boldsymbol{b}_1,\cdots,\boldsymbol{b}_n\}$ が次の 2 つの条件を満たすとき, これを V の基底という.

(1) $\{\boldsymbol{b}_1,\cdots,\boldsymbol{b}_n\}$ は線形独立である.
(2) V の任意の元は, $\{\boldsymbol{b}_1,\cdots,\boldsymbol{b}_n\}$ の線形結合である. すなわち,

$$\forall\boldsymbol{x}\in V,\exists a_1,\cdots,a_n\in\mathbf{K}:\boldsymbol{x}=a_1\boldsymbol{b}_1+\cdots+a_n\boldsymbol{b}_n.$$

定義 1.3.2 (次元) 線形空間 V の基底を構成するベクトルの個数を V の次元といい, $\dim V$ で表す.

1.4　正規直交系

定義 1.4.1 (直交) 内積空間のベクトル $\boldsymbol{x}, \boldsymbol{y}$ に対して,

$$(\boldsymbol{x}, \boldsymbol{y}) = 0$$

が成り立つとき, $\boldsymbol{x}$と$\boldsymbol{y}$は直交するといい, $\boldsymbol{x} \perp \boldsymbol{y}$ と表す.

定義 1.4.2 (正規直交系) 内積空間のベクトル $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$ が次の条件を満たすとき, 正規直交系という:

(1) $||\boldsymbol{v}_i|| = 1$;
(2) $\boldsymbol{v}_i \perp \boldsymbol{v}_j \quad (i \neq j)$.

グラムシュミットの正規直交化法

線形独立なベクトル $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$ から正規直交系 $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\}$ を以下の要領で作る:

$$\begin{aligned}
\boldsymbol{u'}_1 &:= \boldsymbol{v}_1, \qquad \boldsymbol{u}_1 := \boldsymbol{u'}_1/||\boldsymbol{u'}_1||; \\
\boldsymbol{u'}_2 &:= \boldsymbol{v}_2 - (\boldsymbol{v}_2, \boldsymbol{u}_1)\boldsymbol{u}_1, \qquad \boldsymbol{u}_2 := \boldsymbol{u'}_2/||\boldsymbol{u'}_2||; \\
\boldsymbol{u'}_3 &:= \boldsymbol{v}_3 - (\boldsymbol{v}_3, \boldsymbol{u}_1)\boldsymbol{u}_1 - (\boldsymbol{v}_3, \boldsymbol{u}_2)\boldsymbol{u}_2, \\
&\qquad\qquad\qquad\qquad \boldsymbol{u}_3 := \boldsymbol{u'}_3/||\boldsymbol{u'}_3||; \\
&\vdots \\
\boldsymbol{u'}_n &:= \boldsymbol{v}_n - \sum_{k=1}^{n-1} (\boldsymbol{v}_n, \boldsymbol{u}_k)\boldsymbol{u}_k, \qquad \boldsymbol{u}_n := \boldsymbol{u'}_n/||\boldsymbol{u'}_n||.
\end{aligned}$$

例 1.4.1 $\mathbf{R}^3$ の基底 $\boldsymbol{v}_1 := (1,0,0)^t, \quad \boldsymbol{v}_2 := (1,1,0)^t, \boldsymbol{v}_3 := (1,1,1)^t$ から正規直交規定を作る.

$$\begin{aligned}
\boldsymbol{u}_1 &= \boldsymbol{v}_1 \quad (\because ||\boldsymbol{v}_1|| = 1); \\
\boldsymbol{u'}_2 &= \boldsymbol{v}_2 - (\boldsymbol{v}_2, \boldsymbol{u}_1)\boldsymbol{u}_1 = (1,1,0)^t - 1 \cdot (1,0,0)^t \\
&= (0,1,0)^t; \\
\boldsymbol{u}_2 &= (0,1,0)^t; \\
\boldsymbol{u'}_3 &= \boldsymbol{v}_3 - (\boldsymbol{v}_3, \boldsymbol{u}_1)\boldsymbol{u}_1 - (\boldsymbol{v}_3, \boldsymbol{u}_2)\boldsymbol{u}_2 \\
&= (1,1,1)^t - 1 \cdot (1,0,0)^t - 1 \cdot (0,1,0)^t \\
&= (0,0,1)^t; \\
\boldsymbol{u}_3 &= (0,0,1)^t.
\end{aligned}$$

$$\therefore \ \{(1,0,0)^t, (0,1,0)^t, (0,0,1)^t\}.$$

1.5　線形写像

定義 1.5.1 (線形写像) V, V' を線形空間とする. 写像 $f : V \to V'$ が,

$$f(\boldsymbol{x} + \boldsymbol{y}) = f(\boldsymbol{x}) + f(\boldsymbol{y}), \quad f(a\boldsymbol{x}) = af(\boldsymbol{x})$$

を満たすとき, 写像 f を V から V' の線形写像 (linear mapping) という. ここで, $\boldsymbol{x}, \boldsymbol{y} \in V$, a はスカラーである.

定理 1.5.1 写像 $f : V \to V'$ が線形写像であることの必要十分条件は,

$$f(a\boldsymbol{x} + b\boldsymbol{y}) = af(\boldsymbol{x}) + bf(\boldsymbol{y})$$

が成り立つことである. ここで, $\boldsymbol{x}, \boldsymbol{y} \in V$, a, b はスカラーである.

像　$\mathrm{Im}f := \{f(\boldsymbol{x}) \mid \boldsymbol{x} \in V\}$.

核　$\mathrm{Ker}f := \{x \in V \mid f(\boldsymbol{x}) = 0\}$.

定義 1.5.2 (行列表現) $\boldsymbol{b}_1 := \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}, \quad \boldsymbol{b}_2 := \{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n\}$ をそれぞれ線形空間 V, V' の基底とするとき,

$$(f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_n)) = (\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n)P$$

を満たす行列 P を基底 $\boldsymbol{b}_1, \boldsymbol{b}_2$ による f の行列表現という.

例 1.5.1 $f : \mathbf{R}^2 \to \mathbf{R}^3$ が,

$$f((x_1, x_2)^t) = (x_1, 2x_1 + 3x_2, 2x_1 - 3x_2)^t$$

で定義され, $\mathbf{R}^2, \mathbf{R}^3$ の基底がそれぞれ

$$\{(1,1)^t, (2,-1)^t\}, \quad \{(1,1,-1)^t, (1,0,1)^t, (1,-2,4)^t\}$$

である. これらの基底による f の行列表現 P を求める.

$$f((1,1)^t) = (1,5,-1)^t, \quad f((2,-1)^t) = (2,1,7)^t$$

だから,

$$\begin{pmatrix} 1 & 2 \\ 5 & 1 \\ -1 & 7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & -2 \\ -1 & 1 & 4 \end{pmatrix} P.$$

$$\therefore \ P = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & -2 \\ -1 & 1 & 4 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 2 \\ 5 & 1 \\ -1 & 7 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & -3 & -2 \\ -2 & 5 & 3 \\ 1 & -2 & -1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 5 & 1 \\ -1 & 7 \end{pmatrix}$$
$$= \begin{pmatrix} 11 & -13 \\ 20 & 22 \\ -8 & -7 \end{pmatrix}.$$

定義 1.5.3 (基底変換行列) $\boldsymbol{b} := \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$ を線形空間 V の基底とする. $\boldsymbol{b}' := \{\boldsymbol{x'}_1, \cdots, \boldsymbol{x'}_n\}$ を線形空間 V の $\boldsymbol{b}$ とは異なる基底とするとき,

$$(\boldsymbol{x'}_1, \cdots, \boldsymbol{x'}_n) = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n)T$$

を満たす行列 T を基底 $\boldsymbol{b}$ から $\boldsymbol{b}'$ への基底変換行列という.

例 1.5.2 $\mathbf{R}^2$ の基底 $\{(-1,2)^t, (1,0)^t\}$ を別の $\mathbf{R}^2$ の基底 $\{(1,1)^t, (2,-1)^t\}$ に変換する基底変換行列 T を求める.

$$\begin{pmatrix} 1 & 2 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 2 & 0 \end{pmatrix} T$$

$$\therefore\ T = \begin{pmatrix} -1 & 1 \\ 2 & 0 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 2 \\ 1 & -1 \end{pmatrix}$$
$$= \begin{pmatrix} 1/2 & -1/2 \\ 3/2 & 3/2 \end{pmatrix}.$$

1.6 行列式

定義 1.6.1 (行列式) n 次正方行列:

$$A := \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$$

に対して, 要素を同じ列から選ばないようにして各行から 1 つずつ抜き取り, 全体の符号を掛け合わせ, すべての組み合わせに対して和をとってできたスカラーを行列式といい, $\det(A)$ または $|A|$ と表す.

計算手順 (1 次正方行列の場合)

$$|a_{11}| = a_{11}.$$

計算手順 (2 次正方行列の場合)

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}.$$

計算手順 (3 次正方行列の場合)

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}.$$

計算手順 (4 次以上の正方行列の場合)

定理 1.6.1(後述) をもちいて,

$$\begin{vmatrix} a_{11} & & * \\ & \ddots & \\ O & & a_{nn} \end{vmatrix}$$

の形に変形し, 定理 1.6.2(後述) を適用する.

定理 1.6.1 *1.* 行列のある行を $k(\neq 0)$ した行列の行列式は, もとの行列の行列式の k に等しい.
2. 行列のある行に他の行のスカラー倍を加えた行列の行列式は, もとの行列の行列式に等しい.

3. 行列の隣り合う 2 行を入れ替えた行列の行列式は, もとの行列の行列式の -1 倍に等しい.

定理 1.6.2

$$\begin{vmatrix} a_{11} & & * \\ & \ddots & \\ O & & a_{nn} \end{vmatrix} = a_{11} \cdots a_{nn}.$$

例 1.6.1

$$A := \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

の行列式を求める.

1 行めの -2 倍, -3 倍, -4 倍をそれぞれ 2 行め, 3 行め, 4 行めにくわえて,

$$\det A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & -2 & -7 \\ 0 & -2 & -8 & -10 \\ 0 & -7 & -10 & -13 \end{vmatrix}.$$

2 行めの -2 倍, -7 倍をそれぞれ 3 行め, 4 行めにくわえて,

$$\det A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & -2 & -7 \\ 0 & 0 & -4 & 4 \\ 0 & 0 & 4 & 36 \end{vmatrix}.$$

3 行めの $+1$ 倍をそれぞれ 4 行めにくわえて,

$$\det A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & -2 & -7 \\ 0 & 0 & -4 & 4 \\ 0 & 0 & 0 & 40 \end{vmatrix}.$$

よって, $\det A = 1 \cdot (-1) \cdot (-4) \cdot 40 = 160$.

1.7　逆行列

定義 1.7.1 n 次の正方行列 A に対して,

$$AB = BA = I$$

を満たす n 次の正方行列 B が存在するとき, A は正則または可逆であるという. このとき, B を A の逆行列といい, A^{-1} で表す.

計算手順 (2 次正方行列の場合)

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}^{-1} = \frac{1}{\Delta} \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix}.$$

ここで, $\Delta := a_{11}a_{22} - a_{12}a_{21}$.

計算手順 (3 次以上の正方行列の場合)

1. n 次正方行列 A に対して, 行列 $(A|I)$ を考える. ここで, I は n 次の基本行列.
2. $(A|I)$ に行列の基本変形 (後述) を行い, $(I|X)$ の形に変形する.
3. 2. で得られた行列 X が求める逆行列 A^{-1} である.

行列の基本変形

行の基本変形 (elementary transformation of row) は, 以下の操作とそれらを組み合わせたものである:

1. ある行をスカラー倍 (0 以外) する.
2. ある行を他の行にくわえる.
3. 2 つの行を入れ替える.

例 1.7.1

$$A := \begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & 0 \\ 3 & 0 & 3 \end{pmatrix}$$

の逆行列を求める.

$$(A|I) = \left(\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 2 & 2 & 0 & 0 & 1 & 0 \\ 3 & 0 & 3 & 0 & 0 & 1 \end{array}\right).$$

1 行目の -2 倍, -3 倍をそれぞれ 2 行目, 3 行目にくわえて,

$$\left(\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & -2 & -6 & -2 & 1 & 0 \\ 0 & -6 & -6 & -3 & 0 & 1 \end{array}\right).$$

2 行目を $-1/2$ 倍して,

$$\left(\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 3 & 1 & -1/2 & 0 \\ 0 & -6 & -6 & -3 & 0 & 1 \end{array}\right).$$

2 行目の -2 倍, 6 倍をそれぞれ 1 行目, 3 行目にくわえて,

$$\left(\begin{array}{ccc|ccc} 1 & 0 & -3 & -1 & 1 & 0 \\ 0 & 1 & 3 & 1 & -1/2 & 0 \\ 0 & 0 & 12 & 3 & -3 & 1 \end{array}\right).$$

3 行目を $1/12$ 倍して,

$$\left(\begin{array}{ccc|ccc} 1 & 0 & -3 & -1 & 1 & 0 \\ 0 & 1 & 3 & 1 & -1/2 & 0 \\ 0 & 0 & 1 & 1/4 & -1/4 & 1/12 \end{array}\right).$$

3 行目の 3 倍, -3 倍をそれぞれ 1 行目, 2 行目にくわえて,

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & -1/4 & 1/4 & 1/4 \\ 0 & 1 & 0 & 1/4 & 1/4 & -1/4 \\ 0 & 0 & 1 & 1/4 & -1/4 & 1/12 \end{array}\right).$$

よって,

$$A^{-1} = \left(\begin{array}{ccc} -1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & -1/4 \\ 1/4 & -1/4 & 1/12 \end{array}\right).$$

定義 1.7.2 (階数) 任意の n 次正方行列 A が, 基本変形を繰り返すことによって

$$\left(\begin{array}{cc} I_r & O \\ O & O \end{array}\right) \qquad (I_r : r \text{ 次の基本行列})$$

と変形されるとき, r を行列 A の階数 (rank) といい, $\mathrm{rank}A$ で表す.

1.8 連立一次方程式

計算手順

連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の両辺に対して, 行列の基本変形を用いて

$$\left(\begin{array}{cc} I & * \\ O & O \end{array}\right)\boldsymbol{x} = \boldsymbol{b'}$$

の形に変形する.

例 **1.8.1** 連立方程式:

$$\left(\begin{array}{ccc} 4 & -13 & -13 \\ 3 & -8 & -8 \\ 1 & -2 & -2 \end{array}\right)\left(\begin{array}{c} x \\ y \\ z \end{array}\right) = \left(\begin{array}{c} -55 \\ -36 \\ -10 \end{array}\right)$$

を解く.

行列に基本変形を用いると,

$$\left(\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{array}\right)\left(\begin{array}{c} x \\ y \\ z \end{array}\right) = \left(\begin{array}{c} -4 \\ 3 \\ 0 \end{array}\right).$$

すなわち,

$$\begin{cases} x = -4 \\ y + z = 3 \end{cases}$$

よって,

$$\left(\begin{array}{c} x \\ y \\ z \end{array}\right) = \left(\begin{array}{c} -4 \\ t \\ 3 - t \end{array}\right) = \left(\begin{array}{c} -4 \\ 0 \\ 3 \end{array}\right) + t\left(\begin{array}{c} 0 \\ 1 \\ -1 \end{array}\right).$$

ただし, t は任意の実数.

1.9　固有値と固有ベクトル

定義 1.9.1 (固有値と固有ベクトル) A を正方行列とする.

$$A\boldsymbol{x} = \lambda\boldsymbol{x}$$

を満たす**0**でないベクトル$\boldsymbol{x}$とスカラー λ が存在するとき, λ を A の固有値といい, $\boldsymbol{x}$を固有値 λ に対する固有ベクトルという.

定義 1.9.2 (固有多項式) A を正方行列とする. z を未知数とする. A から定まる多項式:

$$F(z) := |A - zI|$$

を A の固有多項式または特性多項式といい, 方程式:

$$F(z) = 0$$

を A の固有方程式または特性方程式という.

定理 1.9.1 A を正方行列とし, F をその固有多項式とする. このとき,

$$\lambda\text{が } A \text{ の固有値} \iff F(\lambda) = 0$$

が成り立つ.

例 1.9.1

$$A := \begin{pmatrix} 3 & 1 & 2 \\ 1 & 3 & 1 \\ 2 & 1 & 3 \end{pmatrix}$$

の固有値と固有ベクトルを求める.

$$\begin{aligned}
|A-\lambda I| &= \begin{vmatrix} 3-\lambda & 1 & 2 \\ 1 & 3-\lambda & 1 \\ 2 & 1 & 3-\lambda \end{vmatrix} \\
&= -\begin{vmatrix} 1 & 3-\lambda & 1 \\ 3-\lambda & 1 & 2 \\ 2 & 1 & 3-\lambda \end{vmatrix} \\
&= -\begin{vmatrix} 1 & 3-\lambda & 1 \\ 0 & 1-(3-\lambda)^2 & \lambda-1 \\ 0 & 2\lambda-5 & -\lambda+1 \end{vmatrix} \\
&= -\begin{vmatrix} 1-(3-\lambda)^2 & \lambda-1 \\ 2\lambda-5 & 1-\lambda \end{vmatrix} \\
&= -((1-(3-\lambda)^2)(-\lambda+1)-(\lambda-1)(2\lambda-5)) \\
&= -(\lambda-1)(\lambda^2-8\lambda+13) \\
&= -(\lambda-1)(\lambda-(4+\sqrt{3}))(\lambda-(4-\sqrt{3}))
\end{aligned}$$

$$\therefore\ |A-\lambda I| = 0 \iff \lambda = 1, 4\pm\sqrt{3}.$$

i) $\lambda = 1$ のとき

$$A - \lambda I = \begin{pmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{pmatrix}$$

だから, $\boldsymbol{x} := (x_1, x_2, x_3)^t$ は,

$$\begin{pmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

を満たす. 行列の基本変形を用いて,

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\therefore\ \begin{cases} x_1 + x_3 = 0 \\ x_2 = 0 \end{cases}$$

$$\therefore\ (x_1, x_2, x_3)^t = t(1, 0, -1)^t.$$

よって, 固有ベクトルは, $(1, 0, -1)^t$.

ii) $\lambda = 4+\sqrt{3}$ のとき

$$A - \lambda I = \begin{pmatrix} -1-\sqrt{3} & 1 & 2 \\ 1 & -1-\sqrt{3} & 1 \\ 2 & 1 & -1-\sqrt{3} \end{pmatrix}$$

だから, $\boldsymbol{x} := (x_1, x_2, x_3)^t$ は,

$$\begin{pmatrix} -1-\sqrt{3} & 1 & 2 \\ 1 & -1-\sqrt{3} & 1 \\ 2 & 1 & -1-\sqrt{3} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

を満たす. 行列の基本変形を用いて,

$$\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1-\sqrt{3} \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\therefore\ \begin{cases} x_1 - x_3 = 0 \\ x_2 + (1-\sqrt{3})x_3 = 0 \end{cases}$$

$$\therefore\ (x_1, x_2, x_3)^t = t(1, \sqrt{3}-1, 1)^t.$$

よって, 固有ベクトルは, $(1, \sqrt{3}-1, 1)^t$.

iii) $\lambda = 4-\sqrt{3}$ のとき

$$A - \lambda I = \begin{pmatrix} -1-\sqrt{3} & 1 & 2 \\ 1 & -1-\sqrt{3} & 1 \\ 2 & 1 & -1-\sqrt{3} \end{pmatrix}$$

だから, $\boldsymbol{x} := (x_1, x_2, x_3)^t$ は,

$$\begin{pmatrix} -1+\sqrt{3} & 1 & 2 \\ 1 & -1+\sqrt{3} & 1 \\ 2 & 1 & -1+\sqrt{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

を満たす. 行列の基本変形を用いて,

$$\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1+\sqrt{3} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\therefore \begin{cases} x_1 - x_3 = 0 \\ x_2 + (1+\sqrt{3})x_3 = 0 \end{cases}$$

$$\therefore \ (x_1, x_2, x_3)^t = t(1, \sqrt{3}-1, 1)^t.$$

よって, 固有ベクトルは, $(1, \sqrt{3}-1, 1)^t$.

1.10 特別な行列

定義 1.10.1 行列 $A := (a_i j)$ に対して,

1. $A^t := (a_{ji})$ を A の転置行列 (transposed matrix) という.
2. $\overline{A} := (\overline{a_i j})$ を共役行列 (conjugate matrix) という. ここで, $\overline{a_i j}$ は, $a_i j$ の共役複素数である.
3. $A^* := (\overline{A})^t = \overline{A^t} = (\overline{a_{ji}})$ を A の随伴行列 (adjoint matrix) という.

定理 1.10.1 1. $(A^t)^t = A, \quad (A^*)^* = A$;
2. $(A+B)^t = A^t + B^t$;
3. $(aA)^t = aA^t$;
4. $(AB)^t = B^t A^t, \quad (AB)^* = B^* A^*$.

定義 1.10.2 1. A が正規行列 (normal matrix) $\iff AA^* = A^*A$.
2. A がユニタリ行列 (unitary matrix) $\iff AA^* = A^*A = I$.
3. A が自己共役行列 (self-adjoint matrix) またはエルミート行列 $\iff A^* = A$.
4. A が反エルミート行列 (anti-Hermitian matrix) $\iff A^* = -A$.

特に, 成分がすべて実数のとき,

1. A が直交行列 (orthogonal matrix) $\iff AA^t = A^tA = I$.
2. A が対称行列 (symmetric matrix) $\iff A^t = A$.
3. A が交代行列 (alternate matrix) $\iff A^t = -A$.

1.11　行列の対角化

定義 1.11.1 (対角行列) 正方行列 A に対して, ある正則な行列 P が存在して, A が

$$P^{-1}AP = \begin{pmatrix} d_1 & & O \\ & \ddots & \\ O & & d_n \end{pmatrix}$$

の形に変形できるとき, A は対角化可能であるといい, 右辺の形の行列を一般に対角行列という.

定理 1.11.1 $\lambda_1, \cdots, \lambda_n$ を n 次正方行列 A の固有値という. このとき, あるユニタリ行列 U が存在して, A は

$$U^{-1}AU = \begin{pmatrix} \lambda_1 & & * \\ & \ddots & \\ O & & \lambda_n \end{pmatrix}$$

の形に変形できる. 右辺の形の行列を一般に三角行列という.

定理 1.11.2 A が正規行列のとき,

$$A \text{ が三角行列} \Longrightarrow A \text{ が対角行列}$$

である.

対角化手順

A が正規行列であれば, 固有値 $\lambda_1, \cdots, \lambda_n$, 固有ベクトル $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n$ を求め, $U := (\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n)$ とおけば, U はユニタリ行列となり,

$$U^{-1}AU = \begin{pmatrix} \lambda_1 & & O \\ & \ddots & \\ O & & \lambda_n \end{pmatrix}$$

となる.

例 1.11.1

$$A := \begin{pmatrix} 12+2i & -6+4i \\ -6+4i & 3+8i \end{pmatrix}$$

を対角化するためのユニタリ行列 U を求める.

A が正規行列であれば, $U^{-1}AU$ を三角行列にすれば対角行列になる.

$$\begin{aligned} |A-\lambda I| &= \begin{vmatrix} (12-\lambda)+2i & -6+4i \\ -6+4i & (3-\lambda)+8i \end{vmatrix} \\ &= ((12-\lambda)+2i)((3-\lambda)+8i) - (-6+4i)^2 \\ &= \lambda^2 - (15+10i)\lambda + 150i \\ &= (\lambda-15)(\lambda-10i) \end{aligned}$$

よって, A の固有値は $\lambda = 15, 10i$.

i) $\lambda = 15$ のとき

固有ベクトル $\boldsymbol{u}_1 := (x+yi, z+wi)^t$ は,

$$(A-\lambda I)\boldsymbol{u}_1 = \begin{pmatrix} -3+2i & -6+4i \\ -6+4i & -12+8i \end{pmatrix} \begin{pmatrix} x+yi \\ z+wi \end{pmatrix} = \boldsymbol{0}$$

を満たす.

$$\begin{cases} (-3x-6z)+(2y+4w)i = 0 \\ (-6x-12z)+(4y+8w)i = 0 \end{cases}$$

$$\therefore \ x+2z=0, \quad y+2w=0.$$

$$\therefore \ \boldsymbol{u}_1 = (-2,1)^t/\sqrt{5}.$$

ii) $\lambda = 10i$ のとき

同様に, 固有ベクトル $\boldsymbol{u}_2 = (1,2)^t/\sqrt{5}$ を得る.

よって,

$$U = \frac{1}{\sqrt{5}} \begin{pmatrix} -2 & 1 \\ 1 & 2 \end{pmatrix}.$$

実際, 行列 U はユニタリ行列で

$$\begin{aligned} U^{-1}AU &= U^{-1}(A\boldsymbol{u}_1, A\boldsymbol{u}_2) \\ &= U^{-1}(15\boldsymbol{u}_1, 10i\boldsymbol{u}_2) \\ &= U^{-1} \cdot \frac{1}{\sqrt{5}} \begin{pmatrix} -2\cdot 15 & 1\cdot 10i \\ 1\cdot 15 & 2\cdot 10i \end{pmatrix} \\ &= U^{-1} \frac{1}{\sqrt{5}} \begin{pmatrix} -2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 15 & 0 \\ 0 & 10i \end{pmatrix} \\ &= U^{-1}U \begin{pmatrix} 15 & 0 \\ 0 & 10i \end{pmatrix} \\ &= \begin{pmatrix} 15 & 0 \\ 0 & 10i \end{pmatrix}. \end{aligned}$$

第2章　微分方程式

2.1　1階常微分方程式

変数分離形

$g(y)dy = f(x)dx$ の形に変形できるものは，この形に変形して積分する．

例 **2.1.1** $(1+x)y + (1-y)xy' = 0$ を解く．

移項して，$(1-y)xy' = -(1+x)y$ より，

$$\frac{1-y}{y}dy = -\frac{1+x}{x}dx.$$

両辺を積分して，

$$\int \frac{1-y}{y}dy = -\int \frac{1+x}{x}dx,$$

$$\therefore \quad \log y - y = -\log x - x + C'.$$

$$\therefore \quad \log xy = y - x + C'. \quad \therefore \quad xy = Ce^{y-x}.$$

同次形その **1**

$dy/dx = f(y/x)$ は，$y/x = u$ とおく．

例 **2.1.2** $y^2 + (x^2 - xy)y' = 0$ を解く．

変形すると，

$$\frac{dy}{dx} = -\frac{y^2}{x^2-xy} = -\frac{y^2/xy}{(x^2-xy)/xy} = \frac{y/x}{1-x/y}.$$

$y/x = u$ とおき，この式の両辺を x で微分すると，

$$\begin{aligned}\frac{du}{dx} &= \frac{(dy/dx)x - y}{x^2} = \frac{1}{x}\left(\frac{dy}{dx} - \frac{y}{x}\right)\\ &= \frac{1}{x}\left(\frac{u}{1-1/u} - u\right) = \frac{1}{x}\frac{u - u(1-1/u)}{1-1/u}\\ &= \frac{1}{x}\frac{1}{1-1/u} = \frac{1}{x}\frac{u}{u-1}.\end{aligned}$$

変形して，$((u-1)/u)du = (1/x)dx$. 両辺積分して，

$$\int \frac{u-1}{u}du = \int \frac{1}{x}dx. \quad \therefore \quad u - \log u = \log x + C'.$$

$$\therefore \quad u = \log ux + C'. \quad \therefore \quad \log y = u - C'.$$

$$\therefore \quad y = Ce^{y/x}.$$

同次形その **2**

$\dfrac{dy}{dx} = f\left(\dfrac{a'x + b'y + c}{ax + by + c}\right)$ $(a'b \neq ab')$ は，$x = X + x_0, y = Y + y_0$ とおく．

同次形その **3**

$\dfrac{dy}{dx} = f\left(\dfrac{k(ax+by) + c'}{ax + by + c}\right)$ は，$ax + by = u$ とおく．

例 **2.1.3**

$$\frac{dy}{dx} = \frac{6x - 2y - 7}{3x - y + 4}$$

を解く．

$$\frac{dy}{dx} = \frac{2(3x-y) - 7}{(3x-y) + 4} = \frac{2u - 7}{u + 4} \quad (u := 3x - y)$$

より，

$$\frac{du}{dx} = 3 - \frac{dy}{dx} = \frac{3(u+4) - (2u-7)}{u+4} = \frac{u+19}{u+4}.$$

変形して，

$$\frac{u+4}{u+19}du = dx, \quad \therefore \quad \int \frac{u+4}{u+19}du = \int dx.$$

$u + 19 = w$ とおいて，$\displaystyle\int \frac{w-15}{w}dw = \int dx.$

$$\therefore \quad (u+19) - 15\log(u+19) = x.$$

$$\therefore \quad (3x - y + 19) - 15\log(3x - y + 19) = x.$$

$$\therefore \quad 3x - y - 15\log(3x - y + 19) = C.$$

線形常微分方程式

$dy/dx = P(x)y + Q(x)$ は, 両辺に $\exp\left(-\int P(x)dx\right)$ をかける.

例 **2.1.4** $t^2(dx/dt) - 2tx = 3$ を解く.

両辺を t^2 で割って, $dx/dt - (2/t)x = 3/t^2$. 両辺に $\exp\left(-\int(2/t)dt\right)$ をかけて,

$$e^{-\int(2/t)dt}\frac{dx}{dt} - e^{-\int(2/t)dt}\frac{2}{t}x = e^{-\int(2/t)dt}\frac{3}{t^2}.$$

$$e^{-\int(2/t)dt}\frac{dx}{dt} - \frac{2}{t}e^{-\int(2/t)dt}x = e^{-\int(2/t)dt}\frac{3}{t^2}.$$

$$e^{-\int(2/t)dt}\frac{dx}{dt} + \frac{d}{dt}\left(e^{-\int(2/t)dt}\right)x = e^{-\int(2/t)dt}\frac{3}{t^2}.$$

$$\frac{d}{dt}\left(e^{-\int(2/t)dt}x\right) = e^{-\int(2/t)dt}\frac{3}{t^2}.$$

$$e^{-\int(2/t)dt}x = \int e^{-\int(2/t)dt}\frac{3}{t^2} + C.$$

$$x = e^{\int(2/t)dt}\left(\int e^{-\int(2/t)dt}\frac{3}{t^2}dt + C\right)$$

ここで, $\exp\left(-\int(2/t)dt\right) = t^{-2}$, $\exp\left(\int(2/t)dt\right) = t^2$ だから,

$$\begin{aligned} x &= t^2\left(\int t^{-2}\frac{3}{t^2}dt + C\right) = t^2\left(3\frac{dt}{t^4} + C\right) \\ &= t^2\left(3\frac{1}{-3t^3} + C\right) = -\frac{1}{t} + Ct^2. \end{aligned}$$

2.2　2 階線形常微分方程式

定数係数の斉次線形常微分方程式

$d^2y/dx^2 + a(dy/dx) + by = 0$ は,

$$\left(\frac{d^2}{dx^2} + a\frac{d}{dx} + b\right)y = 0$$

と変形し, $d/dx = \lambda$ とおくと, $(\lambda^2 + a\lambda + b)y = 0$.

$\lambda^2 + a\lambda + b = 0$ の解 $\lambda = \lambda_1, \lambda_2$ が,

i) 異なる 2 実数のとき,　$y = C_1e^{\lambda_1 x} + C_2e^{\lambda_2 x}$;

ii) 重解のとき,　$y = (C_1 + C_2x)e^{\lambda_1 x}$;

iii) 異なる 2 虚数 $(\lambda_1 := \alpha + i\beta, \lambda_2 := \alpha - i\beta)$ のとき,
$y = e^{\alpha x}(C_1\cos\beta x + C_2\sin\beta x)$.

第3章　解析学

3.1　級数の和

定義 **3.1.1** (収束) 無限級数 $\sum_{n=1}^{\infty} a_n = a_1 + a_2 + \cdots$ に対して, 部分和 $S_n := a_1 + a_2 + \cdots + a_n$ の極限 $S := \lim_{n\to\infty} S_n$ が存在するとき, 級数 $\sum_{n=1}^{\infty} a_n$ は収束するといい, S を級数の和という.

次の定理の対偶より, 級数の収束の吟味は, $\lim_{n\to\infty} a_n = 0$ の場合だけ考えればよい.

定理 **3.1.1** $\sum_{n=1}^{\infty} a_n$が収束 $\Longrightarrow \lim_{n\to\infty} a_n = 0$.

注意 この逆は成立しない.

正項級数

各項がすべて正の級数を正項級数という.

例 **3.1.1** 級数 $1 + 1/2 + 1/3 + 1/4 + \cdots$ は, 発散する.

証明 部分和 S_n を考える. $2^m = n$ とおくと,

$$\begin{aligned} S_n &= 1 + \frac{1}{2} + \left(\frac{1}{3} + \frac{1}{4}\right) + \left(\frac{1}{5} + \cdots \frac{1}{8}\right) \\ &\qquad + \cdots + \left(\frac{1}{n/2+1} + \cdots \frac{1}{n}\right) \\ &> 1 + \frac{1}{2} + \left(\frac{1}{4} + \frac{1}{4}\right) + \left(\frac{1}{8} + \cdots \frac{1}{8}\right) \\ &\qquad + \cdots + \left(\frac{1}{2^m} + \cdots \frac{1}{2^m}\right) \\ &= 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \cdots + \frac{1}{2} \\ &= 1 + \frac{m}{2} \ \to \ \infty. \quad (m \to \infty) \end{aligned}$$

交項級数

奇数番目の項が正, 偶数番目の項が負の級数を交項級数という. 交項級数は,

$$|a_1| > |a_2| > \cdots \quad \text{かつ} \quad \lim_{n\to\infty} a_n = 0$$

ならば, 収束する.

例 **3.1.2** 級数 $1 - 1/2 + 1/3 - 1/4 + \cdots$ は, 収束する.

証明 部分和 S_n を考えると,

$$S_{2n} = \left(1 - \frac{1}{2}\right) + \left(\frac{1}{3} - \frac{1}{4}\right) + \cdots + \left(\frac{1}{2n-1} - \frac{1}{2n}\right)$$

は n が増加するについて増大し,

$$S_{2n} = 1 - \left(\frac{1}{2} - \frac{1}{3}\right) - \left(\frac{1}{4} - \frac{1}{5}\right) - \cdots - \frac{1}{2n} < 1$$

だから, 上に有界である. ゆえに, $\lim_{n\to\infty} S_{2n} = S$ が存在する. ここで,

$$S_{2n+1} = S_{2n} + \frac{1}{2n+1} \to S + 0 = S \quad (n \to \infty)$$

より, $\lim_{n\to\infty} S_n = S$. よって, 収束.

定義 **3.1.2** (絶対収束) 級数 $\sum a_n$ に対して, $\sum |a_n|$ が収束するとき, $\sum a_n$ は絶対収束するという.

定理 **3.1.2** 級数 $\sum a_n$が絶対収束 $\Longrightarrow \sum a_n$は収束する.

定義 **3.1.3** (条件収束) 級数 $\sum a_n$ に対して, $\sum |a_n|$ は発散するが $\sum a_n$ は収束するとき, $\sum a_n$ は条件収束するという.

例 **3.1.3** 前述の級数 $1 - 1/2 + 1/3 - 1/4 + \cdots$ は, 条件収束する.

3.2　微分

定理 3.2.1 (ライプニッツの公式) $f(x), g(x)$ が x_0 で n 回微分可能であるとき,

$$(fg)^{(n)}(x_0) = \sum_{k=0}^{n} {}_nC_k f^{(k)}(x_0) g^{(n-k)}(x_0).$$

例 3.2.1

$$\begin{aligned} & (e^x \log(1+x))^{(n)} \\ = \ & {}_nC_0 (e^x)(\log(1+x))^{(n)} + {}_nC_1 (e^x)'(\log(1+x))^{(n-1)} \\ & \qquad + \cdots + {}_nC_n (e^x)^{(n)} (\log(1+x)). \end{aligned}$$

ここで, $(e^x)' = (e^x)'' = \cdots = (e^x)^{(n)} = e^x$,
$(\log(1+x))' = 1/(1+x)$,
$(\log(1+x))'' = (1/(1+x))' = -1/(1+x)^2$,
$(\log(1+x))''' = (-1/(1+x)^2)' = 2/(1+x)^3$,
$\vdots$
$(\log(1+x))^{(k)} = (-1)^{n-1} \cdot (k-1)(k-2) \cdots 3 \cdot 2/(1+x)^k$
より, $0 \leq k \leq n-1$ に対して,

$$\begin{aligned} & {}_nC_k (e^x)^{(k)} (\log(1+x))^{(n-k)} \\ = \ & \frac{n!}{k!(n-k)!} e^x \frac{(-1)^{n-k-1}(n-k-1)!}{(1+x)^{n-k}} \\ = \ & (-1)^{n-k-1} \frac{n!}{(n-k) \cdot k!} \frac{1}{(1+x)^{n-k}} e^x. \end{aligned}$$

よって,

$$\begin{aligned} & (e^x \log(1+x))^{(n)} \\ = \ & \left\{ \log(1+x) + \frac{n}{1+x} - \frac{n(n-1)}{2(1+x)^n} \right. \\ & \qquad \left. + \cdots + (-1)^{n-1} \frac{n!}{(1+x)^n} \right\} e^x. \end{aligned}$$

定理 3.2.2 (テイラーの公式) $[x_0, x]$ で $f^{(n-1)}$ が連続で, (x_0, x) で $f^{(n)}$ が存在するならば, $\xi \in (x_0, x)$ が存在し,

$$f(x) = f(x_0) + \sum_{k=1}^{n-1} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k + R_n(x).$$

ここで, $R_n(x) := \dfrac{f^{(n)}(\xi)}{n!} (x - x_0)^n$.

定理 3.2.3 (マクローリンの公式) $[0, x]$ で $f^{(n-1)}$ が連続で, $(0, x)$ で $f^{(n)}$ が存在するならば, $\theta \in (0, 1)$ が存在し,

$$f(x) = f(0) + \sum_{k=1}^{n-1} \frac{f^{(k)}(0)}{k!} x^k + \frac{f^{(n)}(\theta x)}{n!} x^n.$$

証明 テイラーの公式で $x_0 = 0$ とおけばよい.

例 3.2.2

$$\begin{aligned} e^x &= e^0 + \frac{e^0}{1}x + \frac{e^0}{2}x^2 + \cdots + \frac{e^0}{n!}x^n + \cdots \\ &= 1 + x + \frac{1}{2}x^2 + \cdots + \frac{1}{n!}x^n + \cdots. \\ \cos x &= \cos 0 + \frac{-\sin 0}{1}x + \frac{-\cos 0}{2}x^2 + \frac{\sin 0}{6}x^3 \\ &\qquad + \frac{\cos 0}{24}x^4 + \cdots \\ &= 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 - \cdots + (-1)^n \frac{1}{(2n)!}x^{2n} + \cdots. \\ \sin x &= \sin 0 + \frac{\cos 0}{1}x + \frac{-\sin 0}{2}x^2 + \frac{-\cos 0}{6}x^3 \\ &\qquad + \frac{\sin 0}{24}x^4 + \cdots \\ &= x - \frac{1}{6}x^3 + \cdots + (-1)^n \frac{1}{(2n+1)!}x^{2n+1} + \cdots. \\ \log(1+x) &= \log 1 + \frac{1/(1+0)}{1}x + \frac{-1/(1+0)^2}{2}x^2 \\ &\qquad + \cdots + \frac{(-1)^{n-1}(n-1)!/(1+0)^n}{n!}x^n + \cdots \\ &= x - \frac{1}{2}x^2 + \cdots + (-1)^{n-1}\frac{1}{n}x^n + \cdots. \end{aligned}$$

第4章　複素解析

4.1　複素微分

定義 4.1.1 (形式的複素微分) $f(z) := u(z)+iv(z)$ ($z := x+iy$) が C^1 級であるとき,

$$\frac{\partial f}{\partial z} := \frac{1}{2}\left(\frac{\partial f}{\partial x} - i\frac{\partial f}{\partial y}\right), \quad \frac{\partial f}{\partial \bar{z}} := \frac{1}{2}\left(\frac{\partial f}{\partial x} + i\frac{\partial f}{\partial y}\right).$$

定義 4.1.2 (複素微分可能性) 関数 $f(z)$ が領域 G で定義されているとき, G の点 z_0 に対して,

$$\frac{df}{dz}(z_0) := \lim_{\alpha\to 0}\frac{f(z_0+\alpha)-f(z_0)}{\alpha}$$

が存在するとき, $f(z)$ は z_0 において複素微分可能であるという.

定理 4.1.1 (コーシー・リーマンの関係式) $f(z)$ は G で C^1 級とする. G の点 z_0 で $f(z)$ が複素微分であるための必要十分条件は,

$$\frac{\partial f}{\partial \bar{z}}(z_0) = 0$$

である. この式は, $f(z) := u(z) + iv(z)$ とおけば,

$$u_x(z_0) = v_y(z_0), \quad u_y(z_0) = -v_x(z_0)$$

と書くこともできる. また, このとき,

$$\frac{df}{dz}(z_0) = \frac{\partial f}{\partial z}(z_0)$$

が成り立つ.

定義 4.1.3 (正則関数) 領域 G において C^1 級で, 各点で複素微分可能な関数を正則関数という.

定理 4.1.2 $f(z)$ が領域 G で C^1 級であるとき, $f(z)$ が G で正則であるための必要十分条件は, G の各点でコーシー・リーマンの関係式が成り立つことである.

4.2　複素積分

定義 4.2.1 (複素積分) 路 $\gamma : z = \varphi(t) \quad (a \le t \le b)$ によって定まる曲線を C とする. 関数 $f(z)$ が C 上で連続であるとき,

$$\int_\gamma f(z)dz := \int_a^b f(\varphi(t))\frac{d\varphi(t)}{dt}dt$$

を $f(z)$ の γ 上の複素積分という.

定理 4.2.1 γ は区分的に滑らかな路とする.

1. $|f(z)| \le M$ ならば,

$$\left|\int_\gamma f(z)dz\right| \le \int_a^b |f(\varphi(t))|\left|\frac{d\varphi(t)}{dt}\right|dt \le M\int_a^b \left|\frac{d\varphi(t)}{dt}\right|dt.$$

2. γ に対して逆向きの路 $-\gamma$ について,

$$\int_{-\gamma} f(z)dz = -\int_\gamma f(z)dz.$$

定義 4.2.2 xy 平面の領域 G の境界 ∂G が有限個の区分的に滑らかな路からなるとき, G はよい領域であるという.

定理 4.2.2 (コーシーの積分定理) G がよい領域で, $f(z)$ が $\overline{G}$ で正則ならば,

$$\int_{\partial G} f(z)dz = 0$$

が成り立つ.

例 4.2.1

$$\int_0^\infty \frac{\sin x}{x}dx = \frac{\pi}{2}$$

を証明する.

$f(z) := e^{iz}/z$ とおくと, $z \neq 0$ で正則である. $0 < \varepsilon < R$ として

$$\begin{aligned}
\gamma_- &: \quad z = x \quad (-R \le x \le -\varepsilon),\\
\gamma_\varepsilon &: \quad z = \varepsilon e^{i(\pi-\theta)} \quad (0 \le \theta \le \pi),\\
\gamma_+ &: \quad z = x \quad (\varepsilon \le x \le R),\\
\gamma_R &: \quad z = Re^{i\theta} \quad (0 \le \theta \le \pi)
\end{aligned}$$

とおく. コーシーの積分定理より

$$\int_{\gamma_-} f(z)dz + \int_{\gamma_\varepsilon} f(z)dz + \int_{\gamma_+} f(z)dz + \int_{\gamma_R} f(z)dz = 0.$$

この式に対して $\varepsilon \to 0,\quad R \to \infty$ とすると

$$2iI + (-i\pi) = 0$$

となることを示せばよい. ここで, $I := \displaystyle\int_0^\infty \frac{\sin x}{x} dx.$

$$\begin{aligned}
(\text{第 1 項}) + (\text{第 3 項}) = &\int_{-R}^{-\varepsilon} \frac{e^{ix}}{x} \cdot 1 \cdot dx + \int_\varepsilon^R \frac{e^i x}{x} \cdot 1 \cdot dx \\
= &\int_R^\varepsilon \frac{e^{-ix}}{-x}(-dx) + \int_\varepsilon^R \frac{e^{ix}}{x} dx \\
= &\int_\varepsilon^R \frac{-e^{-ix} + e^{ix}}{x} dx \\
= &\, 2i \int_\varepsilon^R \frac{-e^{-ix} + e^{ix}}{2ix} dx \\
= &\, 2i \int_\varepsilon^R \frac{\sin x}{x} dx \\
\to &\quad 2iI. \quad (\varepsilon \to 0,\quad R \to \infty)
\end{aligned}$$

$$(\text{第 2 項}) = \int_{\gamma_\varepsilon} \frac{1}{z} dz + \int_{\gamma_\varepsilon} \frac{e^{iz} - 1}{z} dz.$$

$$\int_{\gamma_\varepsilon} \frac{1}{z} dz = \int_0^\pi \frac{1}{\varepsilon e^{i(\pi - \theta)}} (-i\varepsilon e^{i(\pi-\theta)}) d\theta = -i\pi.$$

$$\begin{aligned}
\left| \int_{\gamma_\varepsilon} \frac{e^{iz} - 1}{z} dz \right| &\le \int_0^\pi \left| \frac{e^{iz} - 1}{z} \right| |\varepsilon(-i) e^{i(\pi-\theta)}| d\theta \\
&= \varepsilon \int_0^\pi \left| \frac{e^{iz} - 1}{z} \right| d\theta.
\end{aligned}$$

ここで, ε が十分に小さければ $|(e^{iz} - 1)/z| \le 2$ だから,

$$\left| \int_{\gamma_\varepsilon} \frac{e^{iz} - 1}{z} dz \right| \le 2\pi\varepsilon \to 0. \quad (\varepsilon \to 0)$$

さらに,

$$(\text{第 4 項}) = \int_0^\pi \frac{e^{iRe^{i\theta}}}{Re^{i\theta}} (iRe^{i\theta}) d\theta = i \int_0^\pi e^{iR\cos\theta} e^{-R\sin\theta} d\theta$$

だから,

$$\begin{aligned}
|(\text{第 4 項})| &\le \int_0^\pi |e^{iR\cos\theta}| \cdot |e^{-R\sin\theta}| d\theta \\
&= \int_0^\pi |e^{-R\sin\theta}| d\theta \to 0. \quad (R \to \infty)
\end{aligned}$$

よって, $2iI + (-i\pi) = 0.\ \therefore\ I = \pi/2.$

定理 4.2.3 (コーシーの積分公式その 1) G がよい領域で, $f(z)$ が $\overline{G}$ で正則ならば

$$f(z) = \frac{1}{2\pi i} \int_{\partial G} \frac{f(\zeta)}{\zeta - z} d\zeta \quad (z \in G)$$

が成り立つ.

定理 4.2.4 (コーシーの積分公式その 2) G がよい領域で, $f(z)$ が $\overline{G}$ で正則ならば

$$f^{(k)}(z) = \frac{k!}{2\pi i} \int_{\partial G} \frac{f(\zeta)}{(\zeta - z)^{k+1}} d\zeta \quad (z \in G)$$

が成り立つ.

4.3 級数

定義 4.3.1 (べき級数) 複素数列 $\{a_n\}$ に対して,

$$\sum_{n=0}^{\infty} a_n(z-z_0)^n = a_0 + a_1(z-z_0) + a_2(z-z_0)^2 + \cdots$$

をべき級数といい, z_0 をその中心という.

定理 4.3.1 (テイラー級数展開) $f(z)$ が領域 $|z-z_0| < R$ で正則であるとき, $f(z)$ は次のべき級数に展開できる:

$$f(z) = \sum_{n=0}^{\infty} a_n(z-z_0)^n,$$

ただし,

$$a_n := \frac{1}{2\pi i}\int_{|\zeta-z_0|=r} \frac{f(\zeta)}{(\zeta-z_0)^{n+1}}d\zeta. \qquad (0<r<R)$$

定理 4.3.2 (ローラン展開定理) $0 \le R_1 \le R_2 \le +\infty$ とする. $f(z)$ が円環領域 $R_1 < |z-z_0| < R_2$ で正則であるとき,

$$f(z) = \underbrace{\sum_{m=1}^{\infty} \frac{a_{-m}}{(z-z_0)^m}}_{\text{主要部}} + \sum_{n=0}^{\infty} a_n(z-z_0)^n,$$

ただし,

$$a_n := \frac{1}{2\pi i}\int_{|\zeta-z_0|=r} \frac{f(\zeta)}{(\zeta-z_0)^{n+1}}d\zeta$$

$$(R_1 < r < R_2, \quad n = 0, \pm 1, \pm 2, \cdots)$$

が成り立ち, $f(z)$ のローラン級数という. 簡単のため,

$$f(z) = \sum_{n=-\infty}^{\infty} a_n(z-z_0)^n$$

と書くこともある.

注意 負べきの項がないとき, ローラン級数はテイラー級数に一致する.

よく知られた展開式

1. $1/(1-z) = 1 + z + z^2 + \cdots + z^n + \cdots \quad (|z|<1)$;
2. $1/(1+z) = 1 - z + z^2 - \cdots + (-1)^n z^n + \cdots \quad (|z|<1)$;
3. $e^z = 1 + z + (1/2!)z^2 + \cdots + (1/n!)z^n + \cdots \quad (|z|<\infty)$.

4.4 留数

定義 4.4.1 (留数) $f(x)$ が $0 < |z-z_0| < R$ で正則であるとき, $0 < r < R$ なる r について,

$$\frac{1}{2\pi i}\int_{|z-z_0|=r} f(z)dz$$

を $f(z)$ の z_0 における留数といい, $\mathrm{Res}[f; z_0]$ で表す.

定理 4.4.1 $f(z)$ が z_0 で N 位の極を持つとき,

$$\mathrm{Res}[f; z_0] = \frac{1}{(N-1)!}\lim_{z\to z_0}\frac{d^{N-1}}{dz^{N-1}}[(z-z_0)^N f(z)]$$

が成り立つ.

例 4.4.1

$$f(z) := \frac{z^2+z+1}{z^2(z-1)}.$$

$f(z)$ は, $z=0$ で 2 位の極, $z=1$ で 1 位の極を持つ.

$$\begin{aligned}
\mathrm{Res}[f;0] &= \lim_{z\to 0}\frac{d}{dz}\left[z^2\cdot\frac{z^2+z+1}{z^2(z-1)}\right] \\
&= \lim_{z\to 0}\frac{d}{dz}\left[\frac{z^2+z+1}{z-1}\right] = \lim_{z\to 0}\frac{z^2-2z-2}{(z-1)^2} \\
&= -2, \\
\mathrm{Res}[f;1] &= \lim_{z\to 1}(z-1)\frac{z^2+z+1}{z^2(z-1)} = \lim_{z\to 1}\frac{z^2+z+1}{z^2} \\
&= 3.
\end{aligned}$$

定理 4.4.2 $f(z) := g(z)/h(z)$ が z_0 で 1 位の極を持つとき,

$$\mathrm{Res}[f, z_0] = \frac{g(z_0)}{h'(z_0)}$$

が成り立つ.

例 4.4.2 前述の $f(z) := \dfrac{z^2+z+1}{z^2(z-1)}$ において,

$$\mathrm{Res}[f;1] = \frac{1^2+1+1}{3\cdot 1^2 - 2\cdot 1^2} = 3.$$

定理 4.4.3 (留数定理) G がよい領域で, $f(z)$ が, 特異点 $a_1, \cdots, a_m$ を除く $\overline{G}$ で正則ならば,

$$\int_{\partial G} f(z)dz = 2\pi i \sum_{k=1}^{m} \mathrm{Res}[f; a_k]$$

が成り立つ.

例 4.4.3

$$I := \int_{-\infty}^{\infty} \frac{1}{x^6+1} dx$$

を求める.

$$\begin{aligned} \gamma_x &: z = x \quad (-R \le x \le R), \\ \gamma_R &: z = Re^{i\theta} \quad (0 \le \theta \le \pi) \end{aligned}$$

とおく. $f(z) := 1/(x^6+1)$ の特異点のうち, γ_x と γ_R からなる閉曲線の内部に存在するのは, R を十分に大きく取れば, $z = i, \exp(\pi/6), \exp(5\pi/6)$ である. これらの留数は, それぞれ, $-i/6, \exp(-5\pi/6)/6, \exp(-\pi/6)/6$ だから, 留数定理より,

$$\begin{aligned} &\int_{-R}^{R} \frac{1}{x^6+1} dx + \int_{\gamma_R} \frac{1}{z^6+1} dz \\ &\qquad = 2\pi i \left(-\frac{i}{6} + \frac{e^{-5\pi/6}}{6} + \frac{e^{-\pi/6}}{6} \right). \end{aligned}$$

(右辺) $= 2\pi/3$,

$$\begin{aligned} \left| \int_{\gamma_R} \frac{1}{z^6+1} dz \right| &\le \int_{\gamma_R} \frac{1}{|z^6|+1} |dz| = \int_0^{\pi} \frac{1}{R+1} R d\theta \\ &= \frac{(\pi-0)R}{R+1} \to 0 \quad (R \to \infty) \end{aligned}$$

より, $R \to \infty$ とすれば, $I = 2\pi/3$ を得る.

4.5　定積分

i) $f(z)$ が,
(1) 実軸上に極を持たない,
(2) $z \to \infty$ のとき $zf(z) \to 0$,
(3) 上反平面で有限個の極 a_k を除いて正則ならば,

$$\int_{-\infty}^{\infty} f(x)dx = 2\pi i \sum_{k} \mathrm{Res}[f; a_k].$$

例 4.5.1

1. $\displaystyle\int_{-\infty}^{\infty} \frac{1}{x^6+1} dx = \frac{2}{3}\pi.$
2. $\displaystyle\int_{-\infty}^{\infty} \frac{1}{x^4+1} dx = \frac{\pi}{\sqrt{2}}.$
3. $\displaystyle\int_{-\infty}^{\infty} \frac{1}{(x^2+1)^3} dx = \frac{3}{16}\pi.$
4. $\displaystyle\int_{-\infty}^{\infty} \frac{1}{(x^2+1)^2} dx = \frac{\pi}{2}.$
5. $\displaystyle\int_{-\infty}^{\infty} \frac{1}{(x^2+a^2)(x^2+b^2)} dx = \frac{\pi}{ab(a+b)}.$

ii) $f(z)$ が,
(1) 実軸上に極を持たない,
(2) $z \to \infty$ のとき $f(z) \to 0$,
(3) 上反平面で有限個の極 a_k を除いて正則ならば,

$$\int_{-\infty}^{\infty} f(x)e^{ix} dx = 2\pi i \sum_{k} \mathrm{Res}[f(z)e^{iz}; a_k],$$

$$\int_{-\infty}^{\infty} f(x) \cos x dx = \mathrm{Re} \left[2\pi i \sum_{k} \mathrm{Res}[f(z)e^{iz}; a_k] \right],$$

$$\int_{-\infty}^{\infty} f(x) \sin x dx = \mathrm{Im} \left[2\pi i \sum_{k} \mathrm{Res}[f(z)e^{iz}; a_k] \right].$$

例 4.5.2

1. $\displaystyle\int_{0}^{\infty} \frac{x \sin x}{x^2+1} dx = \frac{\pi}{2e}.$
2. $\displaystyle\int_{0}^{\infty} \frac{x \sin x}{(x^2+1)^2} dx = \frac{\pi}{4e}.$
3. $\displaystyle\int_{-\infty}^{\infty} \frac{x \sin x}{x^4+1} dx = \frac{i\pi}{e^{1/\sqrt{2}}} \sin \frac{1}{\sqrt{2}}.$
4. $\displaystyle\int_{0}^{\infty} \frac{x \sin 2x}{x^4+1} dx = \frac{i\pi}{2e^{1/\sqrt{2}}} \sin \sqrt{2}.$

第5章　信号処理

5.1　連続時間のフーリエ変換

定義 5.1.1 (たたみ込み) 2 つの信号 $x(t)$ と $h(t)$ に対して,

$$x(t) * h(t) := \int_{-\infty}^{+\infty} x(\tau)h(t-\tau)d\tau$$

をたたみ込みという.

定義 5.1.2 (デルタ関数) デルタ関数 $\delta(t)$ を次式で定義する:

$$\delta(t) := \begin{cases} +\infty & (t=0) \\ 0 & (t \neq 0) \end{cases}$$

かつ任意の $\epsilon > 0$ に対して,

$$\int_{-\epsilon}^{\epsilon} \delta(t)dt = 1.$$

このとき, ある時刻 t_0 の連続時間信号 $x(t)$ は,

$$x(t_0) = x(t_0) * \delta(t_0) = \int_{-\infty}^{+\infty} x(t)\delta(t_0 - t)dt$$

と表すことが出来る.

定義 5.1.3 連続時間信号 $x(t)$ のフーリエ変換 $X(\omega)$ は

$$X(\omega) := \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt$$

で定義される. この式を変形して得られる式:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(\omega)e^{j\omega t}d\omega$$

を逆フーリエ変換と呼ぶ.

連続時間信号 $x(n)$ のフーリエ変換および逆フーリエ変換を $\mathcal{F}[x(n)], \mathcal{F}^{-1}[x(n)]$ で表すことがある.

定理 5.1.1 $\mathcal{F}[x(t) * h(t)] = X(\omega)H(\omega)$.

5.2　離散フーリエ変換

単位インパルス

$$\delta := \begin{cases} 1 & (n=0) \\ 0 & (n \neq 0) \end{cases}$$

任意の離散時間信号は, 単位インパルスを使って

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n-k),$$

ただし,

$$\delta(n-k) := \begin{cases} 1 & (n=k) \\ 0 & (n \neq k) \end{cases}$$

と表すことができる.

標本化

連続時間信号 $x_a(t)$ の標本化された信号 $x(t)$ は, インパルス列:

$$\delta_s(t) := \sum_{n=-\infty}^{+\infty} \delta(t - nT)$$

を用いて

$$x(t) = x_a(t)\delta_s(t) = \sum_{n=-\infty}^{+\infty} x_a(t)\delta(t-nT)$$

と表すことができる.

定理 5.2.1 (シャノンの標本化定理) 連続時間信号 $x_a(t)$ が帯域 $\Omega_c[\mathrm{rad}]$ に制限されているとき, 標本化周波数 $\Omega_s[\mathrm{rad/sec}]$ が

$$\Omega_s > 2\Omega_c$$

を満たせば, 標本 $x(n)$ から連続時間信号 $x_a(t)$ を復元できる.

5.3　z 変換

定義 5.3.1 (z 変換) 信号 $x(n)$ に対して,

$$X(z) := \sum_{n=0}^{\infty} x(n) z^{-n}$$

を $\{x(n)\}$ の **z 変換** (z-transform) といい, $\mathcal{Z}\{x(n)\}$ または $X(z)$ で表す.

定義 5.3.2 (収束領域) $X(z)$ が収束する z の範囲を $X(z)$ の収束領域という.

第6章　エントロピー

6.1　エントロピーの定義

定義 6.1.1 (エントロピー) 完全事象系*:*

$$X := \begin{pmatrix} x_1 & \cdots & x_n \\ p_1 & \cdots & p_n \end{pmatrix}$$

に対して,

$$S(p) := -\sum i = 1^n p_i \log p_i$$

を系 (X, p) のエントロピーという.

定義 6.1.2 $(X, p), (Y, q)$ を完全事象系とする. このとき, 複合事象系 $(X \times Y, r)$ のエントロピーは,

$$S(X \times Y) := -\sum_{i,j} p(x_i, y_j) \log p(x_i, y_j)$$

で与えられる.

定義 6.1.3 (条件付きエントロピー) 条件付き確率 $p(x|y)$ に対するエントロピー*:*

$$S(X|Y) := -\sum_{x \in X, y \in Y} p(x, y) \log p(x|y)$$

を条件付きエントロピーと呼ぶ.

定義 6.1.4 (相対エントロピー) p, q を事象の集合 X に対する確率分布とするとき,

$$S(p|q) := \sum_{i=1}^{n} p_i \log p_i / q_i \quad (\text{ただし } p_i \neq 0 \Longrightarrow q_i \neq 0)$$

を相対エントロピーと呼ぶ.

定義 6.1.5 (相互エントロピー) 完全事象系 (X, p), (Y, q) に対して,

$$I(X, Y) := \sum_{x \in X, y \in Y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

を相互エントロピーと呼ぶ.

第7章　数理論理学

7.1　古典命題論理

命題（真偽の確定していることがら）と論理記号 $\neg, \wedge, \vee, \supset$ のみを扱う論理体系.

ヒルベルト流公理系

- 公理

  $(\supset_1)$ $A \supset (B \supset A)$;
  $(\supset_2)$ $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$;
  $(\supset_3)$ $((A \supset B) \supset A) \supset A$;
  $(\wedge_1)$ $A \wedge B \supset A$;
  $(\wedge_2)$ $A \wedge B \supset B$;
  $(\wedge_3)$ $(A \supset B) \supset ((A \supset C) \supset (A \supset B \wedge V))$;
  $(\vee_1)$ $A \supset A \vee B$;
  $(\vee_2)$ $B \supset A \vee B$;
  $(\vee_3)$ $(A \supset C) \supset ((B \supset C) \supset (A \vee B \supset C))$;
  $(\neg_1)$ $(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$;
  $(\neg_2)$ $\neg A \supset (A \supset B)$.

- 推論規則（分離規則）

  $A, A \supset B$ が真なら, B も真である.

LK による証明

証明する論理式を A とすると, 式 "$\to A$" に対して以下の推論規則を逆に適用することで, 以下の公理を導く.

定義 7.1.1 (式) 論理式 $A_1, \cdots, A_m, B_1, \cdots, B_n$ を

$$A_1, \cdots, A_m \to B_1, \cdots, B_n$$

の形に繋げたものを式といい, $A_1 \wedge \cdots \wedge A_m \supset B_1 \vee \cdots \vee B_n$ を意味する. なお, m, n は 0 でもよい.

定義 7.1.2 (推論規則) 式 $S_1, \cdots, S_n, S$ を

$$\frac{S_1 \cdots S_n}{S}$$

の形にならべたものを推論規則といい, $S_1, \cdots, S_n$ が正しいときに S も正しいことを意味する.

- **LK** の公理：$A \to A$.

- **LK** の推論規則

増左	$\dfrac{\Gamma \to \Delta}{D, \Gamma \to \Delta}$	増右	$\dfrac{\Gamma \to \Delta}{\Gamma \to \Delta, D}$
減左	$\dfrac{D, D, \Gamma \to \Delta}{D, \Gamma \to \Delta}$	減右	$\dfrac{\Gamma \to \Delta, D, D}{\Gamma \to \Delta, D}$
換左	$\dfrac{\Gamma, C, D, \Pi \to \Delta}{\Gamma, D, C, \Pi \to \Delta}$	換右	$\dfrac{\Gamma \to \Delta, C, D, \Pi}{\Gamma \to \Delta, D, C, \Pi}$
$\supset$ 左	$\dfrac{\Gamma \to \Delta, A \quad B, \Pi \to \Sigma}{A \supset B, \Gamma, \Pi \to \Delta, \Sigma}$	$\supset$ 右	$\dfrac{A, \Gamma \to \Delta, B}{\Gamma \to \Gamma, A \supset B}$
$\wedge$ 左	$\dfrac{A, \Gamma \to \Delta}{A \wedge B, \Gamma \to \Delta}$	$\wedge$ 右	$\dfrac{\Gamma \to \Delta, A \quad \Gamma \to \Delta, B}{\Gamma \to \Delta, A \wedge B}$
$\vee$ 左	$\dfrac{A, \Gamma \to \Delta \quad B, \Gamma \to \Delta}{A \vee B, \Gamma \to \Delta}$	$\vee$ 右	$\dfrac{\Gamma \to \Delta, A}{\Gamma \to \Delta, A \vee B}$
$\neg$ 左	$\dfrac{\Gamma \to \Delta, A}{\neg A, \Gamma \to \Delta}$	$\neg$ 右	$\dfrac{A, \Gamma \to \Delta}{\Gamma \to \Delta, \neg A}$

例 7.1.1 *1.* $A \supset B \supset A \supset A$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \to A}{A \to A, B}}{\to A, A \supset B} \quad A \to A}{A \supset B \supset A \to A, A}}{A \supset B \supset A \to A}}{\to A \supset B \supset A \supset A}$$

2. $\neg\neg A \supset A$.

$$\dfrac{\dfrac{\dfrac{A \to A}{\to A, \neg A}}{\neg\neg A \to A}}{\to \neg\neg A \supset A}$$

7.2　古典述語論理

述語 (変数の内容が確定しないと真偽が確定しないことがら) と論理記号 $\neg, \wedge, \vee, \supset, \forall, \exists$ を扱う論理体系.

古典述語論理の公理系

ヒルベルト流の古典命題論理の公理系に, 次の限定記号の公理と限定記号の推論を追加する:

- 限定記号の公理
  $(\forall_1)\ \forall x A(x) \supset A(t)$;
  $(\exists_1)\ A(t) \supset \exists x A(x)$.
- 限定記号の推論
  $(\forall_2)\ C \supset A(a) \Longrightarrow C \supset \forall x A(x)$;
  $(\exists_2)\ A(a) \supset C \Longrightarrow \exists x A(x) \supset C$.

古典述語論理の **LK**

次の 4 つの推論を追加する:

$\forall$ 左 $\dfrac{A(t), \Gamma \to \Delta}{\forall x A(x), \Gamma \to \Delta}$　$\forall$ 右 $\dfrac{\Gamma \to \Sigma, A(a)}{\Gamma \to \Sigma, \forall x A(x)}$

$\exists$ 左 $\dfrac{A(a), \Gamma \to \Delta}{\exists x A(x), \Gamma \to \Delta}$　$\exists$ 右 $\dfrac{\Gamma \to \Sigma, A(t)}{\Gamma \to \Sigma, \exists x A(x)}$

$\forall$ 右 と $\exists$ 左 の下式には, 自由変数 a は現れない.

7.3　直観主義命題論理

排中律 ($A \vee \neg A$ が常に真) を認めない命題論理.

直観主義命題論理の公理は, 古典命題論理の公理から $(\supset_3)$ を取り除いたものである.

LJ

LJ の公理は LK と同じ. 推論規則に関しては, LK の右辺の論理式の個数を高々1 つに制限したもの. 推論規則を以下に示す (Δ は高々1 つの論理式):

増左 $\dfrac{\Gamma \to \Delta}{D, \Gamma \to \Delta}$　増右 $\dfrac{\Gamma \to}{\Gamma \to D}$

減左 $\dfrac{D, D, \Gamma \to \Delta}{D, \Gamma \to \Delta}$

換左 $\dfrac{\Gamma, C, D, \Pi \to \Delta}{\Gamma, D, C, \Pi \to \Delta}$

$\supset$ 左 $\dfrac{\Gamma \to A \quad B, \Pi \to \Delta}{A \supset B, \Gamma, \Pi \to \Delta}$　$\supset$ 右 $\dfrac{A, \Gamma \to B}{\Gamma \to A \supset}$

$\wedge$ 左 $\dfrac{A, \Gamma \to \Delta}{A \wedge B, \Gamma \to \Delta}$　$\wedge$ 右 $\dfrac{\Gamma \to A \quad \Gamma \to B}{\Gamma \to A \wedge B}$

$\vee$ 左 $\dfrac{A, \Gamma \to \Delta \quad B, \Gamma \to \Delta}{A \vee B, \Gamma \to \Delta}$　$\vee$ 右 $\dfrac{\Gamma \to A}{\Gamma \to A \vee B}$

$\neg$ 左 $\dfrac{\Gamma \to A}{\neg A, \Gamma \to}$　$\neg$ 右 $\dfrac{A, \Gamma \to}{\Gamma \to \neg A}$

7.4 様相論理

- $\Diamond A$:「A は可能である」
- $\Box A$:「A は必然である」

定義 7.4.1 次の 3 条件で定まるものを様相論理の論理式という (これらの全体を $\mathrm{WFF}^{\Box}$ で表す):

(1) $\mathrm{PVAR} \subseteq \mathrm{WFF}^{\Box}$;
(2) $A, B \in \mathrm{WFF}^{\Box} \Longrightarrow A \supset B, A \wedge B, A \vee B \in \mathrm{WFF}^{\Box}$;
(3) $A \in \mathrm{WFF}^{\Box} \Longrightarrow \Box A, \neg A \in \mathrm{WFF}^{\Box}$.

ここで, PVAR は命題変数全体の集合.

定義 7.4.2 *1.* $\neg\Box\neg A$ を $\Diamond A$ と表す;
2. $\Box(A \supset B)$ を $A \prec B$ と表す;
3. $(A \prec B) \wedge (B \prec A)$ を $A = B$ と表す.

体系 M

- 公理 : (1) $\Box A \supset A$, (2) $\Box(A \supset B) \supset (\Box A \supset \Box B)$.
- 推論規則 :
  $A, A \supset B \in M \Longrightarrow B \in M$ (分離規則);
  $A \in M \Longrightarrow \Box A \in M$

体系 S_4 体系 M に公理 (3) $\Box A \supset \Box\Box A$ を追加.

体系 Br 体系 M に公理 (4) $A \vee \Box\neg\Box A$ を追加.

体系 S_5 体系 M に公理 (5) $\Box A \vee \Box\neg\Box A$ を追加.

定理 7.4.1 $S_5 \supset S4, Br \supset M$.

定理 7.4.2 S_4 に公理 *(4)* を追加した体系は S_5 に等しい.

LK 流の推論

$$\Box \text{ 左} \quad \frac{A, \Gamma \to \Delta}{\Box A, \Gamma \to \Delta} \qquad \Box \text{ 右}_M \quad \frac{\Sigma \to A}{\Box\Sigma \to A}$$

$$\Box \text{ 右}_{S_4} \quad \frac{\Box\Sigma \to A}{\Box\Sigma \to \Box A}$$

$$\Box \text{ 右}_{S_5} \quad \frac{\Box\Sigma \to \Box\Delta, A}{\Box\Sigma \to \Box\Delta, \Box A}$$

第8章　数理計画法

8.1　用語

線形計画問題 (linear programming problem)

目的関数 (objective function):

$$z := c_1x_1 + \cdots + c_nx_n$$

を制約条件:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

とすべての変数に対する非負条件 (nonnegative condition):

$$x_j \geq 0, \quad j = 1, \cdots, n$$

の下で最小にする解 $\boldsymbol{x} := (x_1, \cdots, x_n)^t$ を求める問題として定式化される問題.

シンプレックス法 (単体法)

線形計画問題に対する最適解を求めるための手法であり, 次の 2 段階からなる. 第 1 段階で, 1 つの実行可能基底解を得る (存在しない場合は存在しないという情報を得る). 第 2 段階では, 基底変数を入れ換えて, 別の実行可能基底解を得る. このとき, 相対費用係数 (relative cost coefficient) がすべて非負ならば, その実行可能基底解は, 最適解である.

凸集合 (nonconvex set)

n 次元ユークリッド空間 E^n の部分集合 S に対して, $\boldsymbol{x}^1, \boldsymbol{x}^2$ を S に属する任意の 2 点として,

$$\lambda\boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2 \quad (0 \leq \lambda \leq 1)$$

が S に属するとき, 集合 S を凸集合という.

幾何学的には, $\boldsymbol{x}^1, \boldsymbol{x}^2$ を結ぶ (閉) 線分が S に含まれることである.

カーマーカー法

線形計画問題を解くための手法で, 内路経路法とも呼ばれる. 射影変換→内接円上で最小化→逆変換の繰り返しによって多項式オーダーで解を収束させることができる.

シミュレーテッド・アニーリング

一般的な逐次近似法は, 初期の実行可能解より常に解が改善されるアルゴリズムであり, いったん極小値の穴に入ってしまうと, そこから出ることはできない. そこで, 逐次に近似していく過程において解の改善だけでなく改悪も許すことで, 極小値から脱出し最適解に近づけることを考える. この手法をシミュレーテッド・アニーリングという.

第9章　データ構造とアルゴリズム

9.1　アルゴリズム概論

アルゴリズムが正しい条件

(1) どんな入力に対しても, 答えを与える場合にはその答えが正しいこと;
(2) どんな入力に対しても, 有限の時間で答えを与えること.

アルゴリズムの正しさの証明

上記 2 つの条件を別々に証明する.

(1) ループ不変条件 (くり返しのなかで, 回るたびに常に成立する変数間の関係式) の成立を示す.
  (a) 繰り返しが始まる直前にこの条件が成立していること;
  (b) この条件が成立しているときに繰り返しを 1 回進めると, 再び条件が成立すること
  を示せばよい.
(2) 単調減少する変数の値とその下限を示せばよい.

例 **9.1.1** (ユークリッドの互除法) 次のプログラムが正しいことを示す:

```
function gcd(m, n: integer): integer;
  var r: integer;
begin
  repeat
    r := m mod n;
    m := n;
    n := r;
  until r = 0;
  gcd := m;
end.
```

(1) について. 繰り返し直前, 求める最大公約数を d とすると,

$$d = gcd(m, n) \tag{9.1}$$

が成り立つ. このとき, 整数 $m', n'(m', n'$ は互いに素) を使って, $m = dm'$, $n = dn'$ と表すことができる.

1 回繰り返しが回った直後, m, n はそれぞれ元の n, r. ここで, r は m と n を割ったときの余りで, $m = qn + r$ ($q \in \mathbf{Z}$) と表すことができる.

$$\begin{cases} r = m - qn = dm' - q \cdot dn' = d(m' - qn') \\ n = dn' \end{cases}$$

で, $m' - qn'$ と n' は互いに素だから, 最大公約数は d. よって, 式 (9.1) が成り立つ. これは, 任意の回数まわしても常に成り立つ.

有限回まわして $r = 0$ になったとき, $m = qn$ が成り立つ. この n が m に代入されるから, 最大公約数は m. よって, 計算が有限時間で停止した場合, 与える答えは正しい.

(2) について. n は単調減少し, 下限は 0 である.

9.2　線形リスト

要素

リストの各要素は, 同じ型の値を持つ.

構造

リストの要素間の関係は線形.

操作

- procedure FindRight(L: LIST);
- procedure FindLeft(L: LIST);
- procedure Findith(L: LIST; i: integer);
- function Retrieve(L: LIST): Element;
- procedure Update(L: LIST; e: Element);
- procedure InsertLeft(L: LIST; e: Element);
- procedure InsertRight(L: LIST; e: Element);
- procedure Delete(L: LIST);
- function Size(L: LIST);
- function CurPos(L: LIST);
- procedure Create(L: LIST);

実現

(i) 配列のべた詰め

- InsertRight, InsertLeft, Delete——末尾の要素に対しては一定時間. その他は後続する要素をすべて移動するため $O(n)$.
- その他——一定時間.

(ii) 一方向連結リスト

- InsertLeft, Delete, FindLeft, Fineith, Size, CurPos——先頭からたどる必要があるため $O(n)$.
- その他——一定時間

(iii) 一方向連結リスト (ポインタは対応する要素の左隣を指す)

- InsertLeft, Delete——一定時間.
- その他——(ii) と同じ.

(iv) 双方向リスト

- FindLeft——一定時間.
- その他——(ii) と同じ.

連結リストは, 通常レコードとポインタで実現されるが, 配列とインデックスで実現することもできる.

9.3　スタック

要素・構造

線形リストと同じ

操作

先頭要素のみ操作可能.

- procedure PushDown(S: STACK; e: Element);
- procedure PopUp(S: STACK);
- function Retrieve(S: STACK): Element;
- function Empty (S: STACK): Boolean;
- procedure Create(S: STACK);

実現

配列を使用.

- 配列の長さで制約を受ける.
- 配列は先頭に対する処理は一定時間なので最適.

グラフの探索

深さ優先探索法 (depth-first search) で利用.

9.4 待ち行列 (キュー)

要素・構造

線形リストと同じ.

操作

削除・参照は先頭, 挿入は末尾の要素に対してのみ可能.

- procedure EnQueue(Q: QUEUE; e: Element);
- procedure DeQueue(Q: QUEUE);
- function Retrieve(Q: QUEUE);
- function Empty(S: STACK): Boolean;
- procedure Create(Q: QUEUE);

実現

(i) 配列——配列の最初の方にたまる空きが無駄になる.
(ii) 循環配列——(i) の欠点が解消.

グラフの探索

幅優先探索法 (breadth-first search) で利用.

9.5 木構造 (概論)

木の帰納的定義

木 (根付き木, rooted tree) とは, 有限集合 T(各要素を節点 (node) という) のうち, 次の条件を満たすものである:

(1) 根 (root) と呼ばれる節点が, 1 つだけ指定されている;
(2) 根以外の接点は, $m(\geq 0)$ 個の共通部分を持たない有限集合 $T_1, \cdots, T_m$ に分割でき, $T_1, \cdots, T_m$ は再び木である (これらを部分木 (subtree) という).

表現法

(i) 節点と辺の集合の対 (接点の集合, 辺の集合, 根)

$$(\{A, B, C, D, E\}, \{(A, B), (B, D), (B, E), (A, C)\}, A)$$

(ii) 入れ子の括弧による表現 (根 (部分木) ... (部分木))

$$(A, (B, (D), (E)), C)$$

9.6 順序木

各節点の子に対して, 順序を区別する. この順序を兄弟 (siblings) と呼ぶ.

9.7 2 分木

各節点の子の個数を高々2 とし,「左の子」と「右の子」を区別.

注意 片方の子が空の 2 分木と 1 つの子しか持たない順序木は異なる.

操作

- function FindParent (n: Node): Node;
- function FindLeftChild (n: Node): Node;
- function FindRightChild (n: Node): Node;
- function FindRoot(T: BinaryTREE): Node;
- function Retrieve(n: Node): Label;
- function EmptyNode(n: Node): Boolean;

9.8　2 分探索木

2 分木において, 接点のラベルに対して次の制限を加えたものである:

(1) 節点のラベルの型が線形順序を持つ.
(2) 各節点のラベルが, 左部分木のすべての節点のラベルより大きく, 右部分木のすべての節点のラベルより小さい.

(通りがけ順にたどることで, ソートされたラベルのリストが得られる.)

関数 member(x, A)

```
fun member(x, Empty) = false
  | member(x, Node(y, L, R)) =
      if x = y then true
        else if x < y then member(x, L)
        else member(x, R);
```

関数 insert(x, A)

根から順番にラベルと挿入すべきデータとの大小関係を比較して, それに基づいて反復的に下へたどる. 葉にたどり着いたらその子として導入.

```
fun insert(x, Empty) = Node(x, Empty, Empty)
  | insert(x, Node(y, L, R)) =
      if x = y then Node(y, L, R)
        else if x < y then
          Node(y, insert(x, L), R)
        else Node(y, L, insert(x, R));
```

関数 delete(x, A)

i) 削除すべき節点が子を持たないとき, 削除のみ行う.
ii) 削除すべき節点がただ 1 つの子を持つとき, 親と子をくっつける.
iii) 削除すべき節点が 2 つの子を持つとき, 左部分木の最大の節点で置き換える.

```
fun delete(x, Empty) = Empty
  | delete(x, Node(y, L, R)) =
      if x < y then Node(y, delete(x, L), R)
        else if x > y then
          Node(y, L, delete(x, R))
        else if L = Empty then R
        else if R = Empty then L
        else let val (z, R1) = deletemin(R)
             in  Node(z, L, R1)
             end;
```

関数 deletemin(A)

```
fun deletemin(Node(y, Empty, R)) = (y, R)
  | deletemin(Node(y, L, R) =
      let val (x, L1) = deletemin(L)
      in  (z, Node(y, L1, R)
      end;
```

9.9 AVL 木

2 分木において, 次の制限を加えたものである:

1. すべての節点において, 左部分木と右部分木の高さの差が高々1.

探索が最悪でも $O(\log n)$.

挿入

まず, 通常の 2 分探索木と同様に挿入し, 上の条件を満たさなくなったら, 木を再構成する:

i) 根 A の左の子 B の 左部分木 の高さが h, 右部分木の高さが $h-1$, A の右部分木の高さが $h-1$ の場合.
  (a) B を根とし, A を B の右の子にする.
  (b) 元々の B の右部分木を A の左部分木にする.
  (**1** 重回転 (single rotation) という.)

ii) 根 A の左の子 B の 右部分木 の高さが h, 左部分木の高さが $h-1$, A の右部分木の高さが $h-1$ の場合.
  (a) B の右の子 C を根にする.
  (b) B を C の左の子, A を C の右の子にする.
  (c) 元々の C の左部分木を B の右部分木, C の右部分木を A の左部分木にする.
  (**2** 重回転 (double rotation) という.)

右が高くなる場合も, 同様に考える.

$O(\log n)$.

削除

まず, 通常の 2 分探索木と同様に削除し, 条件を満たさなくなったら, 1 重回転・2 重回転によって木を再構成する.

定理 9.9.1 *n 個の節点をもつ AVL 木の高さは $O(\log n)$ である.*

証明 高さ h の AVL 木の節点の個数が, 最小の場合でも h の指数関数のオーダーになることを示せばよい.

高さ h の AVL 木のうちで, 節点の個数が最小のものの 1 つは, 葉以外のすべての節点において, 左部分木の高さが右部分木の高さより 1 大きいものである (**Fibonacci** 木). 高さ h の AVL 木の最小の節点の個数を N_h とすると,

$$N_h = N_{h-1} + N_{h-2} + 1 \quad (h \geq 2),$$

ここで, $N_0 = 1,\ N_1 = 2$ である. この漸化式を解くと,

$$N_h = \frac{1}{\sqrt{5}} \left\{ \left(\frac{1+\sqrt{5}}{2} \right)^{h-3} - \left(\frac{1-\sqrt{5}}{2} \right)^{h+3} \right\} - 1.$$

$\left| \frac{1-\sqrt{5}}{2} \right| < 1$ より, h を十分に大にとれば, $\left(\frac{1-\sqrt{5}}{2} \right)^{h+3} \sim 0$. また, -1 も無視できるので,

$$N_h \sim \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^{h+3}.$$

よって, 節点の個数 N_h は, 高さ h の指数関数のオーダーで増加する. これは, 逆に AVL 木の高さは, 節点の個数に対して, 高々対数のオーダーでしか増えないことを示している.

9.10　B 木

$m(\geq 3)$ 木において, 次の制限を加えたものである[1] :

1. 各節点 (葉以外) のこの個数は $\lceil m/2 \rceil$ 以上, m 以下 (ただし, 根の子は 2 個以上);
2. 根から葉までの深さは, どの葉についても等しい.

データはすべて葉に昇順に格納され, 葉以外の節は, ((子の個数) -1) 個の要素からなる配列を持ち, k 番目の要素には k 番目の子より大きく, $(k+1)$ 番目の子より小さい値を持つ.

挿入

1. 挿入すべき葉の位置を求める.
2. 求めた位置の親の子の個数が m 以上の場合, 親を 2 つに分割した上で挿入する.
3. 分割によって新しくできた親に対して, 2. を繰り返す.

$O(\log n)$.

削除

1. 削除すべき葉を削除する.
2. 削除した葉の親の子の個数が $\lceil m/2 \rceil$ を下回った場合, 隣 (左右どちらでもよい) と同数の子を持つように, 隣から子をもらってくる.
3. それによってとなりの子の個数が $\lceil m/2 \rceil$ を下回った場合, 両者を合体する.
4. 合体した場合, その親に対して 2. ~ 3. を繰り返す.

$O(\log n)$.

[1] $m = 3$(各節点の子の個数が 2 ~ 3) の B 木を **2-3** 木, $m = 4$(各節点の子の個数が 2 ~ 4) の B 木を **2-3-4** 木という.

9.11　半順序の付いた木

定義

2 分木において, 節点のラベルに対して次の制限を加えたものである:

1. 節点のラベルの型が線形順序を持つ.
2. 各節点のラベルが, 左右の子のラベルより小さいか等しい.

関数 Deletemin(Q)

1. 根節点を削除.
2. 最下段で最も右の葉を根に移す.
3. この節点 (新しい根) とその左右の子を比べ, 子より小さければ入れ換える.
4. 入れ換えた子に対して, 3. を繰り返す.
   (sift down 操作)

手続き Insert(x, Q)

1. x を最下段のできるだけ左に挿入.
2. 挿入した節点とその親を比べ, 親より大きければ入れ換える.
3. 入れ換えた親に対して, 3. を繰り返す.
   (sift up 操作)

配列による実現

1. $h[1]$ には根を入れる.
2. $h[i]$ の左の子を $h[2i]$, 右の子を $h[2i+1]$ に入れる.

このとき, $h[i] \leq h[2i]$, $h[i] \leq h[2i+1]$ $(i = L, L+1, \cdots, R/2)$ を満たし, ヒープと呼ぶ.

9.12 集合

要素

各要素は, 同じ型の値を持つ.

構造

重複はなく, 順序を区別しない. 要素数は有限.

操作

- function Union(A, B: SET): SET;
- function Intersection(A, B: SET): SET;
- function Difference(A, B: SET): SET;
- procedure Insert(x: Element; A: SET);
- procedure Delete(x: Element; A: SET);
- procedure Create(A: SET);
- function Member(x: Element; A: SET): Boolean;
- function Min(A: SET): Element;

実現

(i) ビットベクトル

- 集合演算を論理演算命令を使って容易に実現できる.
- Min の実現は難しい.
- Member, Insert, Delete——一定時間.
- Union, Intersection, Difference——$O(n)$, 一語に収まる場合は一定時間.

(ii) 連結リスト

- Union, Intersection, Difference——ソートされていれば 2 つのリストの長さの和, ソートされていなければ積.
- Insert——ソートされていれば $O(n)$, ソートされていなければ一定時間.

9.13 優先度付き待ち行列

要素の削除参照は, 一番優先度の高い要素に対して行われる. 操作は, Insert と Deletemin がある.

実現

ハッシュ法 (Deletemin が難しい) 以外であれば, どれでも可能.

(i) 連結リスト

- リストをソートする / しないに関わらず, Insert / Deletemin のどちらかが $O(n)$.

(ii) 半順序木

- Insert / Deletemin ともに $O(\log n)$.

9.14　ソート法

単純交換法 (バブルソート)

```
begin
  for i := 1 to n-1 do
    for j := n downto i+1 do
      if A[j-1] > A[j] then
        exchange(j-1, j)
end
```

```
   { 48  (65) (10)  36   83   5   21   72 }
-> { 48  [10]([65])(36)  83   5   21   72 }
-> { 48   10  [36] [65] (83) (5)  21   72 }
-> { 48   10   36   65  [5]([83])(21)  72 }
-> { 48   10   36   65   5  [21]([83])(72)}
-> {(48) (10)  36   65   5   21  [72] [83]}
-> {[10]([48])(36)  65   5   21   72 | 83 }
-> { 10  [36] [48] (65) (5)  21   72 | 83 }
-> { 10   36   48  [5]([65])(21)  72 | 83 }
-> { 10   36  (48) (5) [21] [65]  72 | 83 }
->                 ......
-> {  5   10   21  36   48   65   72   83 }
```

$O(n^2)$.

単純選択法

```
begin
  for i := 1 to n-1 do
    begin
      min := i;
      for j := i+1 to n do
        if A[j] < a[min]
          exchange(j, min)
    end
end
```

```
   {(48)  65    10   36   83    (5)   21    72 }
-> {[5]  (65)  (10)  36   83   [48]   21    72 }
-> { 5 | [10] ([65]) 36   83    48   (21)   72 }
-> { 5    10 | [21]  36  (83)  (48)  [65]   72 }
-> { 5    10    21 | 36  [48] ([83]) (65)   72 }
-> { 5    10    21   36   48 | [65] ([83]) (72)}
-> { 5    10    21   36   48    65 | [72]  [83]}
-> { 5    10    21   36   48    65    72    83 }
```

$O(n^2)$. 比較に比べて交換の手間が大きい場合に有用.

単純挿入法

```
begin
  for i := 2 to n do
    begin
      temp := A[i];
      A[0] := A[i];
      j := i - 1;
      while A[i]  < A[j]
        begin
          A[j+1] := A[j];
          j := j - 1
        end;
      A[j+1] := temp
    end
end
```

```
   { 48 |(65)  10   36   83    5   21   72 }
-> { 48  [65]|(10)  36   83    5   21   72 }
-> {[10]  48   65 |(36)  83    5   21   72 }
-> { 10  [36]  48   65 |(83)   5   21   72 }
-> { 10   36   48   65  [83]| (5)  21   72 }
-> { [5]  10   36   48   65   83 |(21)  72 }
-> {  5   10  [21]  36   48   65   83 |(72)}
-> {  5   10   21   36   48   65  [72]  83 }
```

$O_{ave}(n^2)$, $O_{max}(n^2)$, $O_{min}(n)$. データがほとんど整列済みの場合に有用 (線形時間).

シェルソート

$h(>1)$ おきにとったデータに対して, 挿入法と同様の方法である程度整列された状態にする (場合によってはこれを繰り返す). その上で, 挿入法を適用.

h の取り方で計算量変化. $O(n^{1.25})$ も実現可能.

クイックソート

```
procedure qsort(L, R: integer);
var i: integer;
begin
  if L < R then
    begin
      i := partition(L, R);
      qsort(L, i-1);
      qsort(i+1, R)
    end
end

function partition(L, R: integer): integer;
var i, j: integer;
var center: item;
begin
  i := L;
  j := R;
  center := A[(L+R) div 2];
  repeat
    while A[i] < x do i := i + 1;
    while A[j] > x do j := j - 1;
    if i <= j then
      begin
        exchange(i, j);
        i := i + 1;
        j := j - 1
      end
  until i > j;
  partition := j + 1;
end
```

```
   {(48)  65   10  /36/   83    5   (21)  72 }
-> {[21] (65)  10  /36/   83   (5)  [48]  72 }
-> { 21   [5]  10  /36/   83  [65]   48   72 }
-> {(21)  /5/  10 | 36 | (83) /65/  (48)  72 }
-> {/5/  [21]  10 | 36 | [48] /65/  [83]  72 }
-> { 5 | /21/ (10)| 36 |  48 | 65 | /83/ (72)}
-> { 5 | [10] /21/| 36 |  48 | 65 | [72] /83/}
-> { 5 |  10 | 21 | 36 |  48 | 65 |  72 | 83 }
```

$O_{ave}(n \log n), \quad O_{max}(n^2).$

ヒープソート

まず, ヒープ列を構成する. $A[n/2+1], \cdots, A[n]$ はすべて葉なので, $A[n/2]$ から $A[n/2-1], A[n/2-2], \cdots, A[1]$ と順番にふるい落とし $(\mathrm{sift})^2$ を行う.

次に, このヒープ列からソート列を構成する. ヒープ列の根 $A[1]$ は最小なので $A[n]$ と交換する ($A[n]$ は決定). すると, 配列 $A[1], \cdots, A[n-1]$ がヒープ列でなくなる[3] ので, ふるい落としによってヒープ列を構成する. このとき, 根 $A[1]$ はヒープ列 $A[1], \cdots, A[n-1]$ のなかで最小なので, $A[n-1]$ と交換する. これを繰り返す.

```
begin
  L := n div 2;
  R := n;
  repeat                        (* ヒープ列の構成 *)
    sift;
    L := L - 1
  until L <= 1;
  repeat                        (* ソート列の構成 *)
    exchange(A[1], A[R]);
    R := R - 1;
    sift
  until R <= 1
end

begin                           (*  ふるい落とし *)
  i := L;
  x := A[i];
  while 2 * i <= R do
    begin
      j = 2 * i;
      if (j < R) and (A[j] < A[j + 1]) then
        j := 2 * i + 1;
      if x <= A[j] then goto 13;
      A[i] := A[j];
      i := j;
      j := 2 * i;
    end;
13: A[i] := x
end
```

[2] ヒープ $A[L+1], \cdots, A[R]$ が与えられたとき, $A[L], A[L+1], \cdots, A[R]$ をヒープにすること. いま, ふるい落としの対象要素を $A[i]$ とすると, $\min(A[2i], A[2i+1])$ と $A[i]$ とを比較して, $A[i] > \min(A[2i], A[2i+1])$ なら交換し, 交換した相手を $A[i]$ とおいて, $A[i]$ が葉になるまで繰り返す.

[3] $A[2], \cdots, A[n-1]$ はヒープ列のままである.

9.15　探索

線形探索

　配列やリストに格納されたデータを前から順番に探索する.
挿入：一定時間
探索：$O(n)$

2 分探索

1. データを配列に小さい順にならべる.
2. 与えられたデータと配列の中央のデータを比較し, 与えられたデータの方が小さければ左半分を, 大きければ右半分を探索対象とする.
3. 新しい探索対象に対して, 再帰的に 2. を繰り返す.

挿入：$O(n)$
探索：$O(\log n)$

2 分探索木を用いる方法

挿入：$O(\log n)$
探索：平均 $O(\log n)$. 最悪で $O(n)$.

AVL 木

　探索は最悪でも $O(\log n)$.

- ほとんどの場合, 完全 2 分木より 1 高い程度におさえられる.
- 挿入で約 47%, 削除で約 21%の確率で, 回転が必要.

→ 探索が挿入・削除に比べて多い場合に最適.

B 木

　2 次記憶上で実現するのに最適.
　2-3 分木を 2 分木で表現した場合, AVL 木の制約がゆるいものに相当.
→ AVL 木に比べて再構成が少なくて済む.

ハッシュ法

- サイズ B の配列 T (ハッシュ表という) を用意し, ハッシュ関数 $h : \text{Element} \to \{0, \cdots, B-1\}$ を使って, 挿入・削除・探索を行う. Element 型の要素 x をキー, $h(x)$ をキー x のハッシュ値という.
- いくつかのキーが同じハッシュ値を持つ場合, 衝突 (collision) するといい, 同じハッシュ値を持つキーのクラスをバケット, そのハッシュ値をバケット番号という.
- 平均時間計算量: 一定.

(i) オープンハッシュ法, チェイン法, 連鎖法

- 同じバケット中の要素を連結リストにし, そのヘッダをハッシュ表に格納する.
- 挿入：$T[h(x)]$ をヘッダとする連結リストに x があれば何もしない, なければ挿入.
- 削除：$T[(h(x)]$ をヘッダとする連結リストに x があれば削除, なければ何もしない.
- 探索：$T[h(x)]$ をヘッダとする連結リストに x があるかを調べる.
- 代表的なハッシュ関数 :
  $h(n) := (n^2 \text{ div } C) \bmod B$. ここで, $BC^2 \sim K^2$ となるように C を定める.
- 平均時間計算量 : $O(1 + N/B)$.
  (登録要素数) $\geq 2B$ となったら倍のサイズのハッシュ表を作ることにすれば, 平均時間計算量は常に 3 以下.
- Min の時間計算量のよいアルゴリズムはない.

(ii) クローズドハッシュ法, 開番地法, オープン法

- ハッシュ関数 h のほかに再ハッシュ関数 $h_1, h_2, \cdots$ を用意する. 要素 x を格納するときに $T[h(x)]$ がふさがっていれば再ハッシュ関数を用いて別の場所へ格納する.
- 挿入：x が辞書に含まれているかを調べ, 含まれていれば何もしない, 含まれていなければ '削除状態' または '空状態' が見つかるまで再ハッシュし登録.
- 削除：$T[h(x)]$ が '空状態' なら何もしない, x なら '削除状態' に書き換える, 別の要素または '削除状態' なら再ハッシュ.

- 探索: $T[h(x)]$ が '空状態' なら false, x なら true, 別の要素または '削除状態' なら再ハッシュ.
- 代表的なハッシュ関数

(1) 線形検査法 $h_i(x) := (h(x) + i) \bmod B$. ($h(x)$ に集中しやすい)

(2) $h_i(x) := (h(x) + ci) \bmod B$, c と B は互いに素. (c 個めごとに団子状態)

(3) 2 次関数検査法 $h_i(x) := (h(x) + i^2) \bmod B$.

(4) $h_i(x) := (h(x) + d_i) \bmod B$, d_i : ランダムな順列.

- 登録要素数がバケット数 B の 90%以上になったとき, 表中にある要素を探す平均時間計算量は 2.56 以上, 表中にない要素を探す平均時間計算量は 10 以上.
- Min の時間計算量のよいアルゴリズムはない.

トライ

データの値自身によって木の分岐を決める方法.

例えばデータが文字列なら, 各節点は文字の種類の分だけ子を持ち, それぞれの子が各文字に対応する. 根の子がデータの 1 文字目を表し, さらにその子が次の文字を表す. 以下同様.

長所: 探索・挿入の計算量がデータの個数によらない (データ長に比例).

短所: 記憶域の無駄が大きい. (解決策: 3 段目までトライ, それ以降 2 分探索木など.)

9.16 文字列照合

テキスト文字列 $text := u_1 \cdots u_m$ の中に含まれる照合文字列 $pattern := v_1 \cdots v_n$ の開始位置を求める.

単純法

```
begin
  k := 0;
  found = false;
  while (k < m) and (found = false)
    begin
      k := k + 1;
      j := 1;
      while (text[k+j-1] = pattern[j])
                and (found = false)
        begin
          j := j + 1;
          if j > n then found = true
        end
    end;
  if found = false then k := -1
end
```

$O(mn)$.

KMP 法

照合文字列の繰り返しに着目し, 以下の配列に基づいて照合の一部を省略.

$next[j] = k \iff v_1 \cdots v_k$ と $v_{j-(k-1)} \cdots v_j$ が一致
$\Longrightarrow$ $u_{i-(k-1)} \cdots u_i$ と $v_{j-(k-1)} \cdots v_j$ が一致すれば, $u_{i-(k-1)} \cdots u_i$ と $v_1 \cdots v_k$ が一致する
$\Longrightarrow u_{i+1}$ と v_{j+1} が不一致なら, u_{i+1} と v_{k+1} を調べればよい.

照合文字列の各文字について何文字一致しているかをあらかじめ表にしておく.

例 9.16.1 $text$:= "ABCABCABA", $pattern$:= "ABCABA". $next[1] = 0$, $next[2] = 0$, $next[3] = 0$, $next[4] = 1$, $next[5] = 2$, $next[6] = 1$ より,
$text[4..5]$ と $pattern[4..5]$ が一致
$\Longrightarrow$ $text[4..5]$ と $pattern[1..2]$ が一致.

```
AABABABBABAAB   (BABAA)
B.
 B.
  BABAA
    ..BA
       BABAA
```

$O(m+n)$.

Boyer-Moore のアルゴリズム

照合文字列を 末尾から 照合. 次の方針に従い, 不一致時にできるだけ照合を省略.

(1) 不一致時の テキスト側の 文字が, 照合文字列にあるか, あるならどこにあるかで, 何文字ずらすかを変える.
   1. あらかじめすべての文字に対して, その文字列が照合文字列に含まれているか, 含まれているなら末尾から数えて何番目にあるかを表す表を作成する.
      例 never

n	e	v	r	その他
4	1	2	5	5

   2. 不一致時, テキスト側の文字に対する 1. の表の値の分だけずらす.

(2) 不一致時, すでに一致した部分の繰り返しが 照合文字列にあるか, あるならどこにあるかで何文字ずらすかを変える.
   v_j で不一致になったとする.
   i) $v_{j+1} \cdots v_m$ が $v_{(j+1)-s} \cdots v_{m-s}$ と一致し, $v_j \neq v_{j-s}$ ならば, s 文字ずらす.
   ii) $v_{j+1} \cdots v_m$ の後半 $(m-s)$ 文字 $v_{s+1} \cdots v_m$ が, $v_1 \cdots v_m$ の前半 $(m-s)$ 文字 $v_1 \cdots v_{m-s}$ と一致するとき, s 文字ずらす.
   iii) 上記以外のとき, m 文字ずらす.

あらかじめ照合文字列の各文字に対して, 不一致になったら何文字ずらすかを表にする.

$O(n+m)$. うまくいくと, $O(n/m)$.

第10章　形式言語

10.1　文法と言語

定義 10.1.1 (文法) 文法 G は，次の 4 組 $G := (N, \Sigma, P, S)$ からなる:

(1) N : 非終端記号 (nonterminal symbol) の有限集合;
(2) Σ : 終端記号 (terminal symbol) の有限集合;
(3) P : 生成規則 (production rule) の有限集合;
(4) S : 開始記号 (start symbol), $S \in N$.

定義 10.1.2 (文脈依存文法) 文法 $G := (N, \Sigma, P, S)$ が **1 型文法または文脈依存文法** (context-sensitive grammar; CSG) であるとは, 任意の生成規則 $\alpha \to \beta \in P$ に対して, $|\alpha| \leq |\beta|$ が成り立つことである.

定義 10.1.3 (文脈自由文法) 文法 $G := (N, \Sigma, P, S)$ が **2 型文法または文脈自由文法** (context-free grammar; CFG) であるとは, 任意の生成規則 $\alpha \to \beta \in P$ に対して, α がただ 1 つの非終端記号であることである.

定義 10.1.4 (正規文法) 文法 $G := (N, \Sigma, P, S)$ が **3 型文法または正規文法** (regular grammar) であるとは, 任意の生成規則が $A \to aB$ または $A \to a$ $(A, B \in N, a \in \Sigma)$ からなっていることである.

定義 10.1.5 (言語) 文法 $G := (N, \Sigma, P, S)$ によって生成される言語を

$$\{w \mid w \in \Sigma^* \text{ かつ } S \overset{*}{\Longrightarrow} w\}$$

と定義し, $L(G)$ で表す.

例 10.1.1 $G := (\{S\}, \{0, 1\}, \{S \to 0S1,\ S \to 01\}, S)$.
$S \Longrightarrow 0S1 \Longrightarrow 00S11 \Longrightarrow \cdots \Longrightarrow 0^{n-1}S1^{n-1} \Longrightarrow 0^n1^n$. 逆に, $L(G)$ はこの列しか含まない. よって, $L(G) = \{0^n1^n \mid n \geq 1\}$.

例 10.1.2 $G := (\{S, B, C\}, \{a, b, c\}, P, S)$. ここで, P は以下の生成規則の集合である:

(1) $S \to aSBC$;
(2) $S \to aBC$;
(3) $CB \to BC$;
(4) $aB \to ab$;
(5) $bB \to bb$;
(6) $bC \to bc$;
(7) $cC \to cc$.

$$\begin{aligned}
S &\overset{*}{\Longrightarrow} a^{n-1}S(BC)^{n-1} \quad (\because (1)) \\
&\Longrightarrow a^n(BC)^n \quad (\because (2)) \\
&\overset{*}{\Longrightarrow} a^nB^nC^n \quad (\because (3)) \\
&\Longrightarrow a^nbB^{n-1}C^n \quad (\because (4)) \\
&\overset{*}{\Longrightarrow} a^nb^nC^n \quad (\because (5)) \\
&\Longrightarrow a^nb^ncC^{n-1} \quad (\because (6)) \\
&\Longrightarrow a^nb^nc^n \quad (\because (7)).
\end{aligned}$$

ゆえに, $L(G)$ は $a^nb^nc^n$ $(n \geq 1)$ を含む. また, 逆に $L(G)$ はこの列しか含まない (証明省略). よって, $L(G) = \{a^nb^nc^n \mid n \geq 1\}$.

定義 10.1.6 (帰納性) 文法 G が帰納的 (recursive) であるとは, 任意の語 w について, w が G によって生成されるか否かを決定するアルゴリズムが存在することをいう.

定理 10.1.1 文脈依存文法は, 帰納的である.

例 10.1.3 例 10.1.2 で取り上げた文法 G において, $w := abac$ が $L(G)$ に属するかどうかを考える.
生成規則を高々n 回適用して得られる列の集合を T_n とする. ただし, 長さが 4 より長いものは考えない.

$$\begin{aligned}
T_0 &= \{S\}, \\
T_1 &= \{S, aSBC, aBC\}, \\
T_2 &= \{S, aSBC, aBC, abC\}, \\
T_3 &= \{S, aSBC, aBC, abC, abc\}, \\
T_4 &= \{S, aSBC, aBC, abC, abc\}, \\
&\vdots
\end{aligned}$$

$T_4 = T_3$ で, $abac \notin T_3$. よって, $abac \notin L(G)$.

10.2　文脈自由文法

定理 10.2.1 (Chomsky 標準形定理) 任意の文脈自由言語は, 生成規則が $A \to BC$ または $A \to a$ だけからなる文法によって生成できる.

例 10.2.1 $G := (\{S, A, B\}, \{a, b\}, P, S)$. ここで, $P := \{S \to bA,\ A \to a,\ A \to aS,\ A \to bAA,\ S \to aB,\ B \to b,\ B \to bS,\ B \to aBB\}$.

$A \to a$ と $B \to b$ はすでに標準形になっている.

まず, $S \to bA$ を $S \to C_1A$ と $C_1 \to b$ に,
$A \to aS$ を $A \to C_2S$ と $C_2 \to a$ に,
$A \to bAA$ を $A \to C_3AA$ と $C_3 \to b$ に,
$S \to aB$ を $S \to C_4B$ と $C_4 \to a$ に,
$B \to bS$ を $B \to C_5S$ と $C_5 \to b$ に,
$B \to aBB$ を $B \to C_6BB$ と $C_6 \to a$ に, それぞれ置き換える.

次に, $A \to C_3AA$ を $A \to C_3D_1$ と $D_1 \to AA$ に, $B \to C_6BB$ を $B \to C_6D_2$ と $D_2 \to BB$ にそれぞれ置き換える.

よって, 最終的に得られる生成規則は, 次の通り:

$$S \to C_1A \quad A \to C_2S \quad A \to C_3D_1$$

$$S \to C_4B \quad B \to C_5S \quad B \to C_6D_2$$

$$C_1 \to b \quad C_2 \to a \quad C_3 \to b$$

$$C_4 \to a \quad C_5 \to b \quad C_6 \to a$$

$$D_1 \to AA \quad D_2 \to BB \quad A \to a \quad B \to b$$

定理 10.2.2 (Greibash 標準形) すべての文脈自由言語は, 生成規則が $A \to a\alpha$ の形だけからなる文法によって生成できる. ここで, A は非終端記号, a は終端記号, α は非終端記号の列 (空でもよい) である.

定義 10.2.1
1. CFG $G := (N, \Sigma, P, S)$ において, $L(G)$ のなかに 2 つ以上の異なった最左導出を有する語が存在するとき, G は曖昧 (ambiguous) であるという.
2. CFL を生成するどの文法も曖昧なら, その言語を本質的に曖昧 (inherently ambiguous) という.

例 10.2.2
1. 例 10.2.1 の文法を考える. 語 $aabbab$ は次の 2 つの最左導出がある:
   $S \Longrightarrow aB \Longrightarrow aaBB \Longrightarrow aabB \Longrightarrow aabbS \Longrightarrow aabbaB \Longrightarrow aabbab$;
   $S \Longrightarrow aB \Longrightarrow aaBB \Longrightarrow aabSB \Longrightarrow aabbAB \Longrightarrow aabbaB \Longrightarrow aabbab$.
   よって, 曖昧である.
2. 言語 $L(G) := \{w \mid w$ の a と b の個数は同じ $\}$ は本質的に曖昧ではない. たとえば, $L(G)$ は, 曖昧でない文法 $G_1 := (\{S, A, B\}, \{a, b\}, P, S)$, $P := \{S \to aBS,\ S \to aB,\ S \to bAS,\ S \to bA,\ A \to bAA,\ A \to a,\ B \to aBB,\ B \to b\}$ によって生成される.

10.3　有限オートマトン

定義 10.3.1 (有限オートマトン) (決定性) 有限オートマトン ((deterministic) finite automaton, automata とも書く) とは, 次の 5 組からなるシステム $M := (K, \Sigma, \delta, q_0, F)$ である:

(1) K : 状態 (state) の有限集合;
(2) Σ : 入力アルファベット (input alphabet) の有限集合;
(3) δ : 遷移関数 (transition function), $K \times \Sigma$ から K への写像[1] ;
(4) q_0 : 初期状態 (starting state), $q_0 \in K$;
(5) F : 終了状態 (final state), $F \subset K$.

定義 10.3.2 有限オートマトン $M := (K, \Sigma, \delta, q_0, F)$ において, ある入力系列 $x \in \Sigma^*$ に対して $\delta(q_0, x) \in F$ が存在するとき, x は M によって受理 (accept) されるという.

定義 10.3.3 (非決定性有限オートマトン) 有限オートマトンにおいて, 遷移関数の値域を K から「K の部分集合」に置き換えたものを非決定性有限オートマトン (non-deterministic finite automaton) という. すなわち, 決定性有限オートマトンの遷移関数の値が, 1 個の状態であるのに対し, 非決定性有限オートマトンでは, 0 個以上の状態の集合となる.

定理 10.3.1 L を非決定性有限オートマトンによって受理される集合とするとき, L を受理する決定性オートマトンが存在する.

定理 10.3.2 正規文法 $G := (N, \Sigma, P, S)$ に対して, 有限オートマトン $M := (K, \Sigma, \delta, S, F)$ が存在して, $T(M) = L(G)$.

定理 10.3.3 有限オートマトン M に対して, 正規文法 G が存在し, $L(G) = T(M)$.

[1] $\delta(q_i, a) = q_j$ は, 現在の状態が q_i のときにアルファベット a が入力されると, 状態が q_j に遷移することを表している.

例 10.3.1 $G := (\{S,B\},\{0,1\},P,S)$, $P := \{S \to 0B,\ B \to 0B,\ B \to 1S,\ B \to 0\}$.

G から $T(M) = L(G)$ を満たす非決定性有限オートマトン $M := (\{S,B,A\},\{0,1\},\delta,X,\{A\})$ を構成する.

1. $\delta(S,0) = \{B\}$ ($\because$ S を左辺に, 0 を右辺に持つ規則は $S \to 0B$ のみ);
2. $\delta(S,1) = \emptyset$ ($\because$ S を左辺に, 1 を右辺に持つ規則は存在せず);
3. $\delta(B,0) = \{B,A\}$ ($\because$ $B \to 0B,\ B \to 0$);
4. $\delta(B,1) = \{S\}$ ($\because$ $B \to 1S$);
5. $\delta(A,0) = \delta(A,1) = \emptyset$.

次に, これと同等な決定性オートマトン $M_1 := (K,\{0,1\},\delta',[S],F)$ を構成する.

$$K := \{\emptyset,[S],[A],[B],[A,S],[A,B],[B,S],[A,B,S]\},$$
$$F := \{[A],[A,S],[A,B],[A,B,S]\},$$

$$\begin{array}{ll} \delta'([S],0) = [B], & \delta'([S],1) = \emptyset, \\ \delta'([B],0) = [A,B], & \delta'([B],1) = [S], \\ \delta'([A,B],0) = [A,B], & \delta'([A,B],1) = [S], \\ \delta'(\emptyset,0) = \delta'(\emptyset,1) = \emptyset. & \end{array}$$

最後に, いま構成した有限オートマトン M_1 から文法 $G_1 := (K,\{0,1\},P_1,[S])$ を構成する.

$$\begin{array}{lll} [S] \to 0[B], & [S] \to 1\emptyset, & \\ [B] \to 0[A,B], & [B] \to 1[S], & [B] \to 0, \\ [A,B] \to 0[A,B], & [A,B] \to 1[S], & [A,B] \to 0, \\ \emptyset \to 0\emptyset, & \emptyset \to 1\emptyset. & \end{array}$$

10.4 プッシュダウン・オートマトン

定義 10.4.1 (pd-オートマトン) プッシュダウン・オートマトン (pushdown automaton; **pd-オートマトン**と略す) とは, 次の 7 組からなるシステム $M := (K,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ である:

(1) K : 状態 (state) の有限集合;
(2) Σ : 入力アルファベット (input alphabet) の有限集合;
(3) Γ : **pd-アルファベット** (pushdown alphabet) の有限集合;
(4) δ : 遷移関数 (transition function), $K \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ から $K \times \Gamma^*$ の有限部分集合の写像[2] ;
(5) q_0 : 初期状態 (initial state), $q_0 \in K$;
(6) Z_0 : **pd-初期記号** (start symbol), $Z_0 \in \Gamma$, Z_0 がはじめに pd-記憶部に置かれている;
(7) F : 最終状態 (final state) の集合, $F \subset K$.

定義 10.4.2 pd-オートマトン $M := (K,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ が決定性 (deterministic) であるとは, 次の条件を満たすことをいう:

(1) $q \in K, Z \in \Gamma$ において, $\delta(q,\varepsilon,Z) \neq \emptyset$ ならば, $\forall a \in \Gamma : \delta(q,a,Z) = \emptyset$;
(2) すべての $q \in K, Z \in \Gamma, a \in \Sigma \cup \{\varepsilon\}$ に対して, $\delta(q,a,Z)$ は 2 個以上の要素を含まない.

例 10.4.1 $\{ww^R \mid w \in \{0,1\}^*\}$ を受理する非決定性 pd-オートマトン

$$M = (\{q_1,q_2\},\{0,1\},\{R,B,G\},\delta,q_1,R,\emptyset)$$

(1) $\delta(q_1,0,R) = \{(q_1,BR)\}$;
(2) $\delta(q_1,1,R) = \{(q_1,GR)\}$;
(3) $\delta(q_1,0,B) = \{(q_1,BB),(q_2,\varepsilon)\}$;
(4) $\delta(d_1,0,G) = \{(q_1,BG)\}$;
(5) $\delta(q_1,1,B) = \{(q_1,GB)\}$;
(6) $\delta(q_1,1,G) = \{(q_1,GG),(q_2,\varepsilon)\}$;
(7) $\delta(q_2,0,B) = \{(q_2,\varepsilon)\}$;
(8) $\delta(q_2,1,G) = \{(q_2,\varepsilon)\}$;
(9) $\delta(q_1,\varepsilon,B) = \{(q_2,\varepsilon)\}$;
(10) $\delta(q_2,\varepsilon,B) = \{(q_2,\varepsilon)\}$.

このオートマトンに列 001100 を入力させたときの計算状況を以下に示す:

入力	計算状況
ε	$(q_1,R) \to (q_2,\varepsilon)$
0	(q_1,BR)
00	(q_1,BBR) $(q_2,R) \to (q_2\varepsilon)$
001	$(q_1,GBBR)$
0011	$(q_1,GGBBR)$ (q_2,BBR)
00110	$(q_1,BBGGBR)$ (q_2,BR)
001100	$(q_1,BBGGBBR)$ $(q_2,BBGGR)$
	$(q_2,R) \to (q_2,\varepsilon)$

[2] $\delta(q_i,a,X) = \{(q_j,Y)\}$ は, 現在の状態が q_i で pd-記憶部の先頭のアルファベットが X のときに, アルファベット a が入力されると, pd-記憶部の X が Y に書き換えられ, 状態が q_j に遷移することを表している. 非決定性 (後述) の場合, 入力列を受理できるように自由に選択してよい.

定理 10.4.1 次の 3 つは同値である[3] :

1. L が文脈自由言語である;
2. ある pd-オートマトン M_1 が存在し, $L = N(M_1)$;
3. ある pd-オートマトン M_2 が存在し, $L = T(M_2)$.

例 10.4.2 $M := (\{q_0, q_1\}, \{0,1\}, \{X, Z_0\}, \delta, q_0, Z_0, \emptyset)$;

$\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$,　$\delta(q_1, 1, X) = \{(q_1, \varepsilon)\}$,
$\delta(q_0, 0, X) = \{(q_0, XX)\}$,　$\delta(q_1, \varepsilon, X) = \{(q_1, \varepsilon)\}$,
$\delta(q_0, 1, X) = \{(q_1, \varepsilon)\}$,　$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$

に対して, $N(M)$ を生成する文脈自由文法 $G := (N, \Sigma, P, S)$ を作る.

$$N := \{S, [q_0, X, q_0], [q_0, X, q_1], [q_1, X, q_0], [q_1, X, q_1], [q_0, Z_0, q_0], [q_0, Z_0, q_1], [q_1, Z_0, q_0], [q_1, Z_0, q_1]\},$$

$\Sigma := \{0,1\}$ とする[4] .

M は初期状態が (q_0, Z_0) で, (q_1, ε) で終了するから, 最初の生成規則 $S \to [q_0, Z_0, q_1]$ を得る.

(q_0, Z_0) では, 入力記号 0 を受け付けて (q_0, XZ_0) となり ($\because\ \delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$), 非終端記号 $[q_0, X, q_i]$ から導出される記号列によって (q_i, Z_0) となる $(i = 0, 1)$. そして, $[q_i, Z_0, q_1]$ で (q_1, ε) となり受理される. よって, ここから得られる生成規則は,

$$[q_0, Z_0, q_1] \to 0[q_0, X, q_0][q_0, Z_0, q_1],$$

$$[q_0, Z_0, q_1] \to 0[q_0, X, q_1][q_1, Z_0, q_1].$$

同様に考えて, $\delta(q_0, 0, X) = \{(q_0, XX)\}$ から,

$$[q_0, X, q_0] \to 0[q_0, X, q_0][q_0, X, q_0],$$

$$[q_0, X, q_0] \to 0[q_0, X, q_1][q_1, X, q_0],$$

$$[q_0, X, q_1] \to 0[q_0, X, q_0][q_0, X, q_1],$$

$$[q_0, X, q_1] \to 0[q_0, X, q_1][q_1, X, q_1]$$

を得る.

また, $\delta(q_0, 1, X) = \{(q_1, \varepsilon)\}$ より, M は (q_0, X) において 1 を受け付けて終了するので, $[q_0, X, q_1] \to 1$. 同様に,

$$[q_1, Z_0, q_1] \to \varepsilon \quad (\because\ \delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}),$$

$$[q_1, X, q_1] \to \varepsilon \quad (\because\ \delta(q_1, \varepsilon, X) = \{(q_1, \varepsilon)\}),$$

$$[q_1, X, q_1] \to 1 \quad (\because\ \delta(q_1, 1, X) = \{(q_1, \varepsilon)\})$$

を得る. ただし, $[q_1, X, q_0], [q_0, X, q_0]$ から終端記号列を生成しないので, これらを含む生成規則を除外する. 結局, 生成規則は以下の通り:

$S \to [q_0, Z_0, q_1]$　　$[q_0, X, q_1] \to 1$
$[q_0, Z_0, q_1] \to 0[q_0, X, q_1][q_1, Z_0, q_1]$　　$[q_1, X, q_1] \to \varepsilon$
$[q_0, X, q_1] \to 0[q_0, X, q_0][q_0, X, q_1]$　　$[q_1, X, q_1] \to 1$
$[q_1, Z_0, q_1] \to \varepsilon$

例 10.4.3 言語 $L(G) := \{0^n 1^n \mid n \geq 1\}$ を受理する *pd*-オートマトンを構成する.

この言語の文法は,

$$G = (\{S\}, \{0,1\}, \{S \to 0S1,\ S \to 01\}, S)$$

と表すことができる. これを *Greibach* 標準形に直すと,

$$G = (\{S, B\}, \{0,1\}, \{S \to 0SB,\ S \to 0B,\ B \to 1\},\ S)$$

となる. 求める *pd*-オートマトンを

$$M := (\{q_0, q\}, \{0,1\}, \{Z_0, S, B\}, \delta, q_0, Z_0, \emptyset)$$

とおくと, 遷移関数 δ を以下の要領で求める:

G において S が開始記号だから,

$$\delta(q_0, 0, Z_0) = \delta(q, 0, S),\quad \delta(q_0, 1, Z_0) = \delta(q, 1, S).$$

以下, $X \to a\alpha,\ X \to a\beta, \cdots$ という生成規則があれば, 遷移関数を $\delta(q, a, X) = \{(q, \alpha), (q, \beta), \cdots\}$ と定義する.
$S \to 0SB,\ S \to 0B$ より, $\delta(q, 0, S) = \{(q, SB), (q, B)\}$.
$B \to 1$ より, $\delta(q, 1, B) = \{(q, \varepsilon)\}$.
ここで, 定義されていない遷移関数は空集合をとる.

[3] $T(M)$ は, 最終状態で受理される言語を, $N(M)$ は, 記憶内容を空にすることで受理される言語を表す.

[4] 非終端記号 $[q_i, X, q_j]$ は, この非終端記号から導出される終端記号列 w が, pd-オートマトン M が $(q_i, X\alpha)$ の状態で読み取りが開始され, 読み取りが終了したときの M の状態が q_j, 記憶部が α(先頭の X が消去された状態) であることを表す.

第11章　オペレーティングシステム

11.1　プロセスの相互交渉

- 協調 (cooperation)

  複数のプロセスが互いに協調して情報交換を行いながら, 共通の目的のために処理を進めるもので, 互いに他のプロセスを必要とする場合.

- 競合 (competition)

  資源を取り合うなど, 互いに他のプロセスの存在を意識しなければならないが, 自分自身の処理の進行には不要な場合.

- 干渉 (interference)

  相手のプロセスの中断など, 他のプロセスに影響を与える場合.

11.2　排他制御 (概論)

排他制御が満たすべき条件

(1) クリティカルセクション[1] を実行しているプロセスが 1 つもないとき, クリティカルセクションには射る洋弓をしたプロセスは ただちに許可 されなければならない.
(2) 2 つ以上のプロセスがクリティカルセクションに入ろうと争っているとき, その選択は無期限に延長されてはならない (デッドロック防止条件).
(3) いかなるプロセスも, あるプロセスがそのクリティカルセクションに入るのを無期限に妨げることはできない (公平条件).

[1] 分割してはならないプログラムの部分 (実行の途中で他のプロセスが実行されると矛盾が起こる可能性がある部分).

フラグ (実現の基本的な考え方)

誰か (いずれかのプロセス) がクリティカルセクションに入っているというフラグ (旗) を設けること. いずれかのプロセスがクリティカルセクションに入るとフラグが上げられ, 出ると降ろされる. 他のプロセスは, フラグが上がっているときにはクリティカルセクションに入らないようにする.

問題点 フラグの上げ下げ自身が, 排他制御と同じ問題である.

ユニプロセッサシステムでの実現

クリティカルセクションを割込み禁止にすればよい.

マルチプロセッサシステムでの実現

次のハードウェア命令を用いる:

- **TS** 命令 (Test and Set) : M 番地のメモリ内容をレジスタ R に移し, M 番地のメモリに 1 を入れる (フラグとして使われる);
- **SW** 命令 (Swap) : M 番地のメモリ内容とレジスタ R の値を入れ換える.

TS 命令, SW 命令ともに, 分割不可能で, 同時には 1 個の CPU でしか実行されないことを ハードウェアが 保証する.

11.3　セマフォア

排他制御の一方法で, イベント変数 (event variable) の一種.

イベント変数

イベントの発生やその履歴を伝えるためにアクセスされる, プロセス間の共通変数. 次の 2 つの要素からなる.

(1) イベントの履歴を記録するデータ構造部分 (セマフォアでは整数 S);
(2) プロセスの待ち行列.

セマフォアの命令

- **P 命令**

  条件 $S \geq 1$ が満たされるまでプロセスを一時停止 (**wait 命令**).

```
P(S) (* S : セマフォア *)
begin
  if S >= 1 then
    begin
      S := S - 1;
      P 命令を実行したプロセスはそのまま続行
    end
  else
      P 命令を実行したプロセスを待ち行列に
        入れて待ち状態に移す
end
```

- **V 命令**

  一時停止させられたプロセスを起動 (**signal 命令**).

```
V(S)
begin
  if |L(S)| >= 1 then
              (* |L(S)| : S の待ち行列に
                 入っているプロセスの個数 *)
    begin
      待ち行列をプロセスを 1 つ取り出す;
      取り出したプロセスを実行可能状態に;
      V 命令を実行したプロセスも実行可能状態に
    end
  else
    begin
      S := S + 1;
      V 命令を実行したプロセスは続行
    end
end
```

AND 命令

デッドロック[2] を避けるため, P 命令と V 命令を次のように拡張する:

- **P 命令**

```
P_and(S1, S2, ..., Sn)
begin
  if S1 >= 1 and ... and Sn >= 1 then
    for i := 1 to n do
      Si := Si - 1;
  else
    begin
      現在のプロセスを Si < 1 を満たす最初の
          Si の待ち行列に入れる;
      再起動アドレスを P 命令の先頭に設定する
    end
end
```

- **V 命令**

```
V(S1, S2, ..., Sn)
begin
  for i := 1 to n do
    begin
      Si := Si + 1;
      Si の待ち行列のすべてのプロセスを
        実行可能状態にする
    end
  V 命令を要求したプロセスを実行可能列に入れる
end
```

[2] プロセス $1, 2, \cdots, N$ がリソース $R_1, R_2, \cdots, R_N$ をそれぞれ保持しながら, $R_2, R_3, \cdots, R_1$ を要求する (要求の環) ができたときに, すべてのプロセスが互いに他のプロセスのリソース開放を待ち, そのプロセスも先に進めなくなる現象.

11.4 セマフォアの応用

プロデューサ / コンシューマ問題

プロデューサがメッセージをバッファに入れ, コンシューマがバッファからメッセージを取り出す (取り出されたメッセージはバッファからなくなる). プロデューサは, バッファがいっぱいのときはメッセージを入れるのを停止し, バッファが空くのを待ち, コンシューマは, 空のバッファからメッセージを取り出してはならない.

解法 2 つのセマフォア S, M を用意する. S はバッファの空きスロットの個数を数えるのに用い, M はメッセージの個数を数えるのに用いる.

```
var S, M: SEMAPHORE;
    BUFFER: array[0..N-1] of MESSAGE;
S := N;
M := 0;

(* プロデューサ *)
var I: integer;
I := 0;
while true do
  begin
    P(S);
    BUFFER[I] := メッセージ;
    V(M);
    I := (I + 1) mod N;
  end

(* コンシューマ *)
var J: integer;
J := 0;
while true do
  begin
    P(M);
    メッセージ := BUFFER[J];
    V(S);
    J := (J + 1) mod N;
  end
```

この解を複数プロデューサ / 複数コンシューマ問題に拡張するには, プロデューサの共通変数 I とコンシューマの共通変数 J を排他制御しなければならない.

リーダ / ライタ問題

ライタはレコードにデータを書き込み, リーダはそれを読み出す. 不用意にレコードの書き込み / 読み出しを行うと項目間に矛盾が生じる.

座席予約システムなどで現れる.

解法 現在レコードをアクセスしているリーダの個数を数え, もし 0 でなければライタの書き込みを禁止する. リーダは, ライタが書き込み中であればアクセスを延期する.

```
var W: SEMAPHORE;
W := 1;

(* ライタ *)
while true do
  begin
    P(W);
    書き込み;
    V(W)
  end

(* リーダ *)
var M: SEMAPHORE;
var R: integer;
M := 1;
R := 0;
while true do
begin
  P(M);
  if R = 0 then P(W);
  R := R + 1;
  V(M);
  読み出し;
  P(M);
  R := R - 1;
  if R = 0 then V(W);
  V(M);
end
```

ダイニング・フィロソファ問題

5 人のフィロソファが食卓についている．各フィロソファは, 考えることと食べることを繰り返す．テーブルの中央にはスパゲッティがあり, 自分の前に両脇に置かれたフォークを 2 本とって食べる．食卓にはフォークが 5 本しか置かれていないため, 自分の両隣が食べていないときのみ食べることができる．

(オペレーティングシステムのリソース競合[3] の抽象化)

```
var FORK: array [0..4] of SEMAPHORE;
FORK := {1, 1, 1, 1, 1};

(* プロセス i のプログラム *)
while true do
  begin
    think;
    P_and(FORK[(i+1) mod 5], FORK[i]);
    eat;
    V(FORK[(i+1) mod 5], FORK[i])
  end
```

[3] たとえば, プロセス A が磁気テープユニットとディスクを, プロセス B がディスクとプリンタを, プロセス C がプリンタとカードリーダ, プロセス D がカードリーダとプロッタ, プロセス E がプロッタと磁気テープユニットを要求した場合など．

11.5　モニタ

目的

リソースの管理と割り当てを円滑に記述するための寄稿を提供すること．

構造

- リソース自身;
- ローカルデータ (リソースの状態を保持している変数);
- スケジューラ;
- 待ち行列 (条件変数);
- プロシージャ(プロセスがリソースを確保・開放・初期化などをするためのもの);
- 初期化コード (モニタが定義されたときに実行され, ローカルデータを初期化).

モニタのプロシージャは, どのプロセスからでも呼ぶことができるが, 同時には 1 個のプロセスしかモニタのプロシージャを実行できない．

命令

- **wait** 命令
  (1) 待ち行列にプロセス (プロセス制御ブロック; PCB) を登録する;
  (2) モニタの排他制御をはずす;
  (3) 制御をディスパッチャへ渡す．
- **signal** 命令
  (1) 待ち行列に入っているプロセスの 1 つがただちに起動される;
  (2) 起動されたプロセスのモニタ内手続きの実行が終了する．

多くの場合, 待ち状態の理由は 1 つ以上あるので, どの理由で俟っているかを区別するため, 条件 (condition) という型の変数を導入する:

```
condvariable.wait
condvariable.signal
```

11.6 モニタの応用

プロデューサ / コンシューマ問題

- 資源
  大きさ N のバッファ.
- ローカル変数 `firstcounter`
  メッセージが入っているスロットの位置.
- ローカル変数 `lastpointer`
  バッファ内の空きスロットの位置.
- ローカル変数 `count`
  バッファに含まれているメッセージの個数
- 待ち行列 `nonempty`
  メッセージが入るまで俟つための待ち行列
- 待ち行列 `nonfull`
  スロットがあくまで俟つための待ち行列
- プロシージャ`append`
  プロデューサがバッファにメッセージを入れるときに呼ぶ. `count` $= N$ ならば, メッセージを入れて `count` を 1 増やし, `nonempty.signal` を起動する.
- プロシージャ`remove`
  待っているコンシューマがあれば, その中の 1 つが呼び, メッセージを取り出す.

リーダ / ライタ問題

ダイニング・フィロソファ問題

モニタは AND 条件を容易に記述できるので, 簡単に解くことができる.

11.7 マルチプログラミング (概論)

マルチプログラミングとは

主記憶に複数個のプログラムを格納し, 入出力待ちが生じたときにプログラムを切り替えることによってコンピュータの利用効率を向上させる方式.

マルチタスキングともいう.

記憶保護

(i) ロック / キー機構
主記憶を同一の長さのブロックに分割し, それぞれに鍵をかける方式. ユーザプログラムにはキーが与えられ, 各ブロックのロックの値と一致した場合のみアクセスが許される. 一致しない場合には記憶保護違反割り込みが生じる.

(ii) 下限レジスタ機構
下限レジスタ (lower limit register) という特別なハードウェアレジスタが, ユーザ領域の下限を示す. ユーザプログラムから主記憶へのアクセスは, すべて下限レジスタの値と比較され, 下限レジスタよりも小さければ, 記憶保護違反割り込みが発生する.

リエントラントプログラム (再入可能プログラム)

複数のプロセスで同時に共用可能なプログラム. 以下の条件を満たさなければならない:

(1) プログラムがピュア (実行中はそのプログラムに書き込み / 修正をしない).
(2) リエントラントプログラムとそれを呼ぶプログラムは互いにアクセス可能 (パラメータを渡すことも可能).
(3) 個々のプロセスは, 記憶保護される.

マルチプログラミングの形式

次の条件の組み合わせで, 4 つの形式が生じる:

(1) 分割を固定するか否か.
(2) プログラムのスワップイン / スワップアウトを許すか否か.

11.8　オーバレイ

プログラムを

(1) ルートセグメント：常に主記憶に存在する部分 (オーバレイを制御する);
(2) オーバレイセグメント：セグメントの呼び出し要求にしたがってスワップイン / スワップアウトされる部分

に分割する.

- オーバレイ構造：セグメント間の呼び出し関係に基づいて作られる木構造.
- リンケージエディタ：オーバレイ構造を作る.

11.9　セグメンテーション

プログラムをセグメントと呼ばれる主記憶領域にロードされる論理的な単位で, 論理アドレスを割り振っておき, それを実行時に, 複数のセグメントレジスタで修飾して物理アドレスを生成する方式.

セグメンテーションの長所

1. プロシージャの共用が容易.
2. 仮想記憶を実現可能.
3. セグメントの大きさを動的に増減可能.

セグメンテーションの短所

1. フラグメンテーションが生じる (メモリコンパクションが必要).
2. セグメントの管理が困難.

11.10　ページング

ロック / キー機構を動的再配置機能を備えるように拡張したもので,

ページ：仮想アドレス空間を分割した各領域

から

ページフレーム : 物理アドレス空間を分割した各領域

への変換 (mapping) 機構.

ページとページフレームはすべて同じ大きさで, ページは任意のページフレームに割り当ててよい.

ページングの機能

1. 動的再配置機能
   (∵ ページのページフレームへの割り当てを実行時に任意に変更可能)
2. (外部) フラグメンテーションの解決
   (∵ 固定の大きさのページを用い, ページはいかなるページフレームにも割り当て可能)
   ただし, 内部フラグメンテーション[4] は解決されず.
   (ページを小さくすれば解決できるが, 主記憶 / 二次記憶間のページの転送量・ページテーブルの大きさが増加.)
3. 仮想記憶の実現

ページングの長所

1. メモリフラグメンテーション問題を解決
   (メモリコンパクションが不要).
2. ジョブのアドレス空間が, 主記憶の大きさに制限されない.
3. 主記憶をより有効に使える (結果として, CPU の利用効率向上).

ページングの短所

1. ハードウェアによるサポートが必要.
2. 実行時のオーバーヘッドが少なくない.
3. ページテーブルがかなりの主記憶領域をしめる.
4. ページ管理のアルゴリズムが複雑.

[4] プログラムの大きさが, ページの大きさの正数倍でないために, ページ内に空き領域ができること.

第12章　数値解析

12.1　ニュートン法

ニュートン法とは

非線形方程式 $f(x)=0$ の解 α を以下の手順で求める:

1. 解 α の近くの値を出発値 $x^{(0)}$ とする;
2. $x^{(k+1)} := x^{(k)} - f(x^{(k)})/f'(x^{(k)})$　$f'(x^{(k)}) \neq 0$;
3. 停止条件が満たされれば終了し, そうでなければ $k := k+1$ として, 2. に戻る.

例 **12.1.1** $f(x) := 1/x - a = 0$　$(a>0)$ をニュートン法で解く.

$f'(x) = -1/x^2$ より反復式は,

$$x^{(k+1)} = x^{(k)} - \frac{1/x - a}{-1/x^2} = x^{(k)} + x(1-ax)$$

収束性

ニュートン法の反復は, $f(x)$ が解 α の近傍で滑らかな関数で, かつ $f'(\alpha) \neq 0$ のとき, 出発値 $x^{(0)}$ が α に十分近ければ

$$k^{(k)} \to \alpha \quad (k \to \infty)$$

となる. また, ある数以上の k に対しては

$$|x^{(k+1)} - \alpha| \leq C|x^{(k)} - \alpha|^2 \quad (C:\text{定数}); \qquad (12.1)$$

$$|x^{(k+1)} - x^{(k)}| \sim |x^{(k)} - \alpha|$$

が成り立つ. 式 (12.1) が成り立つとき, $x^{(k)}$ は **2** 次収束するという.

停止条件

次の式を用いる場合が多い:

i) $|x^{(k+1)} - x^{(k)}| < \varepsilon$;
ii) $|f(x^{(k+1)} - f(x^{(k)})| < \varepsilon$.

減速法

ニュートン法の 2. の式のかわりに

$$x^{(k+1)} := x^{(k)} - \mu^{(k)} \frac{f(x^{(k)})}{f'(x^{(k)})}$$

を用いる. $\mu^{(k)}$ は次の方法で定める:

まず, $\mu^{(k)} := 1$ として $x^{(k+1)}$ を計算し,

$$|f(x^{(k+1)})| < \left(1 - \frac{\mu^{(k)}}{2}\right)|f(x^{(k)})|$$

を満たすまで $\mu^{(k)}$ を繰り返し $1/2$ 倍する.

第13章　パターン認識の基礎

13.1　パターン認識概論

認識系の構成

入力
　|
[前処理部]
　|
[特徴抽出部]
　|
[識別演算部] ← [識別辞書]（←あわせて識別部と呼ぶ）
　|
対応するクラスを出力

特徴空間

- 特徴ベクトル (feature vector)：抽出した特徴を組にしたベクトル.
- 特徴空間 (feature space)：特徴ベクトルによって張られる空間.
- クラスタ (cluster)：特徴空間上におけるクラスごとにまとまった塊.

識別系

(i) 最も単純な識別系：特徴空間上の各点のクラス名をすべて識別辞書として格納する.
→識別辞書の作成は人手によるしかないため, 現実的でない.

(ii) 最近傍決定則 (nearest neighbor rule), **NN** 法 (NN rule)：各クラスの代表的なパターン (プロトタイプと呼ぶ) を用意する. 入力パターンは, 特徴空間上でこれらのプロトタイプと比較され, 最も距離の近いプロトタイプ (最近傍 (nearest neighbor) と呼ぶ) の属するクラスを識別結果として出力する.

(iii) k-**NN** 法 (k-nearest neighbor rule)：入力パターンに最も近い k 個のプロトタイプをとり, その中で最も多数を占めたクラスを識別結果として出力する.

13.2　学習

クラスを正しく分離するための決定境界を自動的に求めること.

特徴ベクトルをクラスに対応させるのに用いられる関数を識別関数 (discriminant function) といい, 線形な識別関数を線形識別関数 (linear discriminant function) と呼ぶ. いま, クラス ω_i の線形識別関数を $g_i(x) := w_{i0} + w_{i1}x_1 + \cdots + w_{id}x_d$ とすると, 学習とは, 重みベクトル $\boldsymbol{w}_i := (w_{i0}, w_{i1}, \cdots, w_{id})^t$ を適切に求めることである (非線形の場合でも同様に定義する).

パラメトリックな学習とノンパラメトリックな学習

- パラメトリックな学習 (parametric learning)：学習パターンを用いて確率密度関数のパラメータ推定を行い, 識別部を構成する方法.
- ノンパラメトリックな学習 (nonparametric learning)：学習パターンから直接識別関数を求める方法.

現実のパターン認識の問題において, 確率密度関数が既知ということはまずない. 実際のパターン認識は複雑な分布をし, 正規分布などの単純な確率密度関数はまず当てはまらない. パラメータの個数を増やせば複雑な確率密度関数を任意の精度で近似できるが, あまり現実的でない. よって, ノンパラメトリックな学習の方が実用性は高い.

教師付き学習と教師なし学習

- 教師付き学習 (supervised learning)：学習パターンがその所属クラス名とともに与えられる学習法.
- 教師なし学習 (unsupervised learning)：クラスのラベルが付いていないパターンを用いて学習を行う方法.

線形識別関数と非線形識別関数

- 線形識別関数 (linear discriminant function)
$g(x) := \boldsymbol{w}^t\boldsymbol{x}$.
長所：頑健.

- **2 次識別関数** (quadric discriminant function)
  $g(x) := \boldsymbol{w}^t\boldsymbol{x} + \boldsymbol{x}^t\boldsymbol{W}\boldsymbol{x}$.
  線形識別関数の問題に帰着可能.
  長所：線形に比べ, 自由度が高い.

- 区分的線形識別関数
  線形分離不可能な分布において, 識別関数を複数の線形識別関数の組で表す. クラス ω_i の識別関数 $g_i(\boldsymbol{x})$ は, L_i 個の副次識別関数 $g_i^{(l)}(\boldsymbol{x})$ $(l := 1, \cdots, L_i)$ で表される:

$$\begin{aligned} g_i(\boldsymbol{x}) &:= \max_l\{g_i^{(l)}(\boldsymbol{x})\}; \\ g_i^{(l)}(\boldsymbol{x}) &:= w_{i0}^{(l)} + \sum_{j=1}^{d} w_{ij}^{(d)}\boldsymbol{x}_j. \end{aligned}$$

  長所：どのような複雑な決定境界も任意の精度で近似可能. (→ 有限個の学習パターンを完全に分離できる)
  短所：パーセプトロンの学習規則を適用できない[1] .

- ニューラルネットワーク：入力層と出力層の間に中間層を設け, 非線形名処理を導入することで, 非線形識別関数を実現. 区分的線形識別関数と極限において等価.

頑健性 (robustness)

- 2 次識別関数は, 線形識別関数よりも多くのパラメータを持つため, 最適な識別関数へ漸近するには, より多くのパターンを必要とする.

- 自由度の高い非線形モデルを使用する際, 学習パターンでうまく識別できたからといって, その性能がそのまま未知パターンにも当てはまるとはいえない.

		学習パターン	未知のパターン
非線形モデル	自由度高	○	△
線形モデル	推定値のばらつき小	△	○

[1] パーセプトロンの学習規則は, $\boldsymbol{x}$の任意の関数 $\varphi_1(\boldsymbol{x}), \cdots, \varphi_d(\boldsymbol{x})$ の線形結合で表される関数 (Φ 関数 (Φ function) と呼ぶ) に対して適用可能である.

13.3　ノンパラメトリックな学習

パーセプトロンの学習規則

識別関数, 教師信号[2] ともに 2 値で, 全学習パターンに対して出力と教師信号が一致するまで重みの修正を繰り返す.
長所：線形分離可能であれば, 必ず誤認識 0 に到達.
短所：線形分離不可能な場合には収束しない.

(1) 重みベクトル $\boldsymbol{w}$ の初期値を適当に設定する.
(2) 学習パターンを 1 つ選ぶ.
(3) 識別を行い, ω_i を ω_j と間違ったとき, 次式により重みベクトルを修正する (ρ : 正の定数):

$$\begin{cases} \boldsymbol{w}_i' := \boldsymbol{w}_i + \rho\boldsymbol{x} \\ \boldsymbol{w}_j' := \boldsymbol{w}_j - \rho\boldsymbol{x} \end{cases}$$

(4) (2), (3) をすべての学習パターンに対して繰り返す.
(5) すべての学習パターンを正しく識別できたら終了, 間違えたら (2) に戻る.

中間層を持たないニューラルネットワークに相当.

Widrow-Hoff の学習規則

識別関数の出力を連続値とし, 教師信号との二乗誤差の総和を最小化する.
長所：線形分離不可能な場合でも収束を保証.
短所：線形分離可能な場合に得られる重みは, 必ずしも誤認識 0 の重みではない.

重みベクトルの修正法として,

$$\boldsymbol{w}_i' := \boldsymbol{w}_i - \rho(\boldsymbol{w}_i^t\boldsymbol{x}_p - b_{ip})\boldsymbol{x}_p$$

を用いる.

教師信号 b_{ip} を

$$b_{ip} := \begin{cases} 1 & (\boldsymbol{x}_p \in \omega_i) \\ 0 & (\boldsymbol{x}_p \in \omega_i) \end{cases}$$

とすると, パーセプトロンの学習規則と同じ式を得る. よって, パーセプトロンの学習規則は, Widrow-Hoff の学習規則の特別な場合である.

[2] 教師信号：各学習パターン $\boldsymbol{x}_p$ に対する, 各クラス ω_i の識別関数の望ましい出力値. b_{ip} で表す.
教師ベクトル：各クラス ω_i の教師信号 b_{ip} からなるベクトル. $(b_{1p}, \cdots, b_{cp})^t$.

誤差逆伝播法

ニューラルネットワークに適用.

ユニット i からの出力を g_{ip}, ユニット j への入力を h_{jp} とすると,

$$h_{ip} = \sum_i w_{ij} g_{ip}.$$

また, g_{ip} は非線形関数 f_j を使って $g_{jp} = f_j(h_{jp})$ と表せる. 出力層の l 番目のユニットに対する教師信号 b_{lp} と実際の出力との二乗誤差 J_p は,

$$J_p = \frac{1}{2} \sum_l (g_{lp} - b_{lp})^2.$$

最急降下法 $(w'_{ij} := w_{ij} - \rho(\partial J_p / \partial w_{ij}))$ を用いる.

$$\frac{\partial J_p}{\partial w_{ij}} = \frac{\partial J_p}{\partial h_{ip}} \cdot \frac{\partial h_{ip}}{\partial w_{ij}}.$$

ここで, $\partial J_p / \partial h_{jp} = \varepsilon_{jp}$ とおくと, $\partial J_p / \partial w_{ij} = \varepsilon_{jp} g_{ip}$.

$$\therefore \quad w'_{ij} = w_{ij} - \rho \varepsilon_{jp} g_{ip}.$$

次に ε_{jp} を求める.

$$\varepsilon_{jp} = \frac{\partial J_p}{\partial h_{jp}} = \frac{\partial J_p}{\partial g_{ip}} \cdot \frac{\partial g_{ip}}{\partial h_{jp}} = \frac{\partial J_p}{\partial g_{jp}} \cdot f'_j(h_{jp}).$$

$\partial J_p / \partial g_{jp}$ は, ユニット j が出力層にあるときは $\partial J_p / \partial g_{jp} = g_{jp} - b_{jp}$, 中間層にあるときは $\partial J_p / \partial g_{jp} = \sum_k \varepsilon_{kp} w_{jk}$.

非線形関数 f_j としてシグモイド関数:

$$S(u) := \frac{1}{1 + \exp(-u)}$$

を用いるとすると, $S'(u) = S(u)(1 - S(u))$ より,

$$f'_j(h_{jp}) = g_{jp}(1 - g_{jp}).$$

よって,

$$\varepsilon_{jp} = \begin{cases} (g_{jp} - b_{jp}) g_{jp}(1 - g_{jp}) & \text{(ユニット j が出力層にある時)} \\ \left(\sum_k \varepsilon_{kp} w_{jk} \right) g_{jp}(1 - g_{jp}) & \text{(ユニット j が中間層にある時)} \end{cases}$$

この手法は, ある重みで計算された出力と教師信号との差に基づいて ε_{jp} を計算し, 出力層から順に重みを修正する.

13.4 パラメトリックな学習

ベイズ決定則

認識対象とするパターンが, 既知の確率密度関数 $p(\boldsymbol{x}|\omega_i)$ に基づいて生起すると考える.

- $P(\omega_i)$: クラス ω_i の生起確率 (事前確率), $\sum_i P(\omega_i) = 1$;
- $p(\boldsymbol{x})$: $\boldsymbol{x}$ の生起確率, $p(\boldsymbol{x}) = \sum_i P(\omega_i) p(\boldsymbol{x}|\omega_i)$;
- $P(\omega_i|\boldsymbol{x})$: $\boldsymbol{x}$が生起されたとき, そのクラスが ω_i である確率 (事後確率), $\sum_i P(\omega_i|\boldsymbol{x}) = 1$

とおき, $P(\omega_i|\boldsymbol{x})$ を最大にするクラス ω_i が$\boldsymbol{x}$の属するクラスだと考える.

ここで,

$$P(\omega_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\omega_i)}{p(\boldsymbol{x})} P(\omega_i)$$

が成り立ち, $\boldsymbol{x}$ を固定して考えると, $p(\boldsymbol{x})$ は定数だから, 識別関数 $g_i(\boldsymbol{x})$ は,

$$g_i(\boldsymbol{x}) := p(\boldsymbol{x}|\omega_i) P(\omega_i)$$

と定義できる.

確率密度関数の形はわかるが, パラメータが未知の場合, パラメータを推定する (この意味でパラメトリックと呼ばれる).

13.5 特徴空間の変換

正規化 (normalization)

各特徴量のスケールの取り方でその特徴量の重み付けが変化するという恣意性を回避するために行われる.
1 つの方法 : パターン相互の距離を最小化する. これは, 各軸の分散を等しくすることに等しい.

KL 展開 (KL expansion)

特徴ベクトルの分布を最も良く近似する部分空間 (subspace) を求める (次元を圧縮する) 方法. 主成分分析と数学的にほぼ等価.

(i) 分散最大基準または (ii) 平均二乗誤差最小基準による.

一般に, 分散最大基準による変換と平均二乗誤差最小基準による変換は一致しないが, 平均二乗誤差最小基準による変換を求める際に, 原点を分布の重心に移動することで, 2 つの変換は一致する.

- 固有次元数 (intrinsic dimensionality)：累積寄与率 (cumulative proportion) が, 最初の m 個の特徴量でほぼ 100%に達するとき, m を固有次元数という.

線形判別法 (linear discriminant method)[3]

各クラスのパターン分布の分離度を最大にする部分空間を求める方法.

- フィッシャーの線形判別法 (Fisher's linear discriminant method)：2 クラスに対する線形判別. 特徴空間上の 2 クラスのパターンの分布から, この 2 クラスを識別するのに最適な 1 次元軸を求める.
  (i) クラス内変動・クラス間変動比最大基準
  (ii) クラス内分散・クラス間分散比最大基準
  (iii) マハラノビス汎距離[4] .

KL 展開と線形判別法

KL 展開は, 表現または圧縮のための次元圧縮 であり, 線形判別法は, 判別のための次元圧縮である.

KL 展開は, 識別は全く考慮されていないが, 以下の理由で広く用いられる:

1. 文字認識と音声認識などは, 通常高次元の特徴ベクトルを必要とし, 次元の呪い から逃れる手段として次元削減が不可欠である.
2. はじめに選ばれた特徴には, 相関の高い 特徴の組が含まれている可能性がある. 相関の高い特徴は, 冗長な情報を含み, 逆行列の計算誤差が大きくなる場合がある.

[3] 統計の分野では, 判別分析 (discriminant analysis) と呼ばれる.
[4] マハラノビスの汎距離 (Mahalanobis generalized distance)：共分散行列の等しい 2 つの分布の平均間距離を表す量. 2 つの分布の平均間のユークリッド距離 (Euclidean distance) が等しくても, 分散が小さければ小さいほど, マハラノビス汎距離は大きくなる.

第14章　人工知能の基礎

14.1　探索

探索とは

探索とは, 多くの可能性の中から特定の解を見つけ出すための戦略. 問題の状態空間を有向グラフで表したとき, 問題の解決は, その有向グラフの出発節点 (初期状態を表す) から目標節点へ至る順路を見つけ出すことである.

横形探索，幅優先探索 (breadth-first search)

1. 出発節点を OPEN に置く.
2. OPEN が空 $\Longrightarrow$ 失敗.
3. OPEN の最初に位置する節点 (n とする) を取り除き, これを CLOSED に置く.
4. n が目標節点 $\Longrightarrow$ 終了.
5. n の, OPEN または CLOSED にない子節点を, OPEN の後ろに 付け加える. (子節点から n へポインタをつける)
6. 2. に戻る.

複数の経路が存在するとき, 横形探索を使えば最短の経路を得られる.

縦形探索，深さ優先探索 (depth-first search)

1. 出発節点を OPEN に置く.
2. OPEN が空 $\Longrightarrow$ 失敗.
3. OPEN の最初に位置する節点 (n とする) を取り除き, これを CLOSED に置く.
4. n が目標節点 $\Longrightarrow$ 終了.
5. n の子節点を OPEN の先頭に 付け加える. (子節点から n へポインタをつける)
6. 2. に戻る.

コストを考慮する探索

1. 出発節点を OPEN に置く.
2. OPEN が空 $\Longrightarrow$ 失敗.
3. OPEN の最初に位置する節点 (n とする) を取り除き, これを CLOSED に置く.
4. n が目標節点 $\Longrightarrow$ 終了.
5. n のすべての子節点 m について, 出発節点からのコスト $\hat{g}(m)$ を求め, OPEN に置く.
6. OPEN をコストの小さい順に並べ換える.
7. 2. に戻る.

ここで, $\hat{g}(m)$ は出発節点から節点 m までのコストの推定値で, 探索対象が木であれば, これはコストの真値 $g(m)$ と一致する. 有向グラフのとき, 一般に $g(n) \leq \hat{g}(m)$ である.

分岐限定法 (branch and bound)

有向グラフの探索において, 循環路を作らないように, n の子節点 m が OPEN あるいは CLOSE に含まれているかに応じて $\hat{g}(m)$ を評価する方法.

以下, 節点 n からその子節点 m へのコストを $c_{n,m}$, 出発節点から n 経由での m までのコストを $\hat{g}(m|n)$ とする.

i) m が OPEN にも CLOSED にも含まれていない場合
$\hat{g}(m) := \hat{g}(m|n)$ とする.

ii) m が OPEN に含まれている場合
すでに得られている $\hat{g}(m)$ と $\hat{g}(m|n)$ を比較し, $\hat{g}(m|n) < \hat{g}(m)$ なら, m からのポインタを n に付け替え, $\hat{g}(m|n)$ を新たな $\hat{g}(m)$ とする.

iii) m が CLOSED に含まれている場合
常に $\hat{g}(m|n) \geq g(m)$ なので, n の子節点としての m は無視する.

- 経路の存在が保証されれば, かならず最適経路を見つけることができる.
- 枝のコストが均一ならば, 分岐限定法は横形探索と一致する.

山登り法 (hill-climbing)

n の子節点 m に対してヒューリスティック関数 $\hat{h}(m)$ を求め, $\hat{h}(m)$ が最も小さい節点を選ぶ[1] .

- 最適順路の発見を保証しない.
- $\hat{h}(m)$ の与え方によっては, 目標節点を見つけられない (収束しない) こともある.

最良優先探索 (best-first search)

1. 出発節点を OPEN に置く.
2. OPEN が空 $\Longrightarrow$ 失敗.
3. OPEN の最初に位置する節点 (n とする) を取り除き, これを CLOSED に置く.
4. n が目標節点 $\Longrightarrow$ 終了.
5. n のすべての子節点 m が OPEN にも CLOSE にも含まれていなければ, m を OPEN に入れる (子節点から n へポインタをつける).
6. OPEN を $\hat{h}(n)$ の小さい順に並べ換える.
7. 2. に戻る.

- 最適順路の発見を保証しない.
- グラフが有限であれば, 探索は必ず成功する.

A アルゴリズム (algorithm A)

$\hat{f}(m) = \hat{g}(n) + \hat{h}(n)$ を分岐限定法と同様の考え方で評価する.

まず, n の子節点 m について, $\hat{f}(m|n) = \hat{g}(n) + c_{n,m} + \hat{h}(m)$ を計算する.

i) m が OPEN にも CLOSED にも含まれていない場合
$\hat{g}(m) := \hat{g}(m|n)$ とする.

ii) m が OPEN に含まれている場合
$\hat{f}(m|n) < \hat{f}(m)$ なら, m からのポインタを n は付け替え, $\hat{f}(m|n)$ を新たな $\hat{f}(m)$ とする.

iii) m が CLOSED に含まれている場合
$\hat{f}(m|n) < \hat{f}(m)$ なら, m からのポインタを n は付け替え, $\hat{f}(m|n)$ を新たな $\hat{f}(m)$ とし, m を OPEN に戻す.

探索は必ず成功するが, 最適順路を発見できる保証はない.

[1] ヒューリスティック関数 $\hat{h}(m)$: 節点 m から目標節点までの最適順路のコストの予測値.

A*アルゴリズム (algorithm A*)

A アルゴリズムにおいて, すべての節点 n に対して $hath(n) \leq h(n)$ を満たすヒューリスティック関数を用いる. このとき, 必ず最適順路を見つけることができる.

$\hat{h}(n) \equiv 0$ ならば, A*アルゴリズムは分岐限定法と一致.

14.2 導出原理

述語論理式 $P(x)$ の恒真性, すなわち $\forall x P(x)$ を証明するには, $\neg P(x)$ が恒偽式であること (充足不能性) を示せばよい.

スコーレム標準形 (Skolem standard form)

$$\forall x_1 \cdots \forall x_m [C_1 \wedge \cdots \wedge C_n].$$

ここで, C_i は節 ($C_i := p_{i1} \vee \cdots \vee p_{ij}$; p_{ij} はリテラル).

以下の要領で, 任意の述語論理式をスコーレム標準形に変換する:

1. 含意記号 "$\rightarrow$" を除去する.
2. 否定記号を素式の直前にもってくる.
3. 束縛変数を標準化する (各限量記号について, その束縛変数の名前が他の限量記号のものと重複しないように名前を付け替える).
4. スコーレム関数を導入して存在記号を除去する ($\forall x_1 \cdots \forall x_n \exists y \forall z_1 \cdots \forall z_m P(\cdot)$ において, y は $x_1, \cdots x_n$ に応じて決まり, $z_1, \cdots, z_m$ には依存しないので, $y = f(x_1, \cdots, x_n)$ とおける).
5. 冠頭形に変換する (減量記号を前に移す).
6. 母式を連言標準形に変換する.

例 14.2.1

$$\begin{aligned}
& \forall x \exists y [P(x,y) \rightarrow Q(x)] \\
& \qquad \wedge \neg (\forall x \exists y [(P(x,y) \rightarrow \neg R(y)) \wedge \forall z R(z)]) \\
= \; & \forall x \exists y [\neg P(x,y) \vee Q(x)] \\
& \qquad \wedge \neg (\forall x \exists y [(\neg P(x,y) \vee \neg R(y)) \wedge \forall z R(z)]) \\
= \; & \forall x \exists y [\neg P(x,y) \vee Q(x)] \\
& \qquad \wedge \exists x \forall y [\neg (\neg P(x,y) \vee \neg R(y)) \vee \neg (\forall z R(z))] \\
= \; & \forall x \exists y [\neg P(x,y) \vee Q(x)] \\
& \qquad \wedge \exists x \forall y [(P(x,y) \wedge R(y)) \vee \exists z (\neg R(z))] \\
= \; & \forall x_1 \exists y_1 [\neg P(x,y) \vee Q(x_1)] \\
& \qquad \wedge \exists x_2 \forall y_2 [(P(x_2,y_2) \wedge R(y_2)) \vee \exists z (\neg R(z))] \\
= \; & \forall x_1 [\neg P(x_1, f(x_1)) \vee Q(x_1)] \\
& \qquad \wedge \forall y_2 [(P(a,y_2) \wedge R(y_2)) \vee \neg R(g(y_2))] \\
= \; & \forall x_1 \forall y_2 [(\neg P(x_1, f(x_1)) \vee Q(x_1)) \\
& \qquad \wedge ((P(a,y_2) \wedge R(y_2)) \vee \neg R(g(y_2)))] \\
= \; & \forall x_1 \forall y_2 [(\neg P(x_1, f(x_1)) \vee Q(x_1)) \\
& \qquad \wedge (P(a,y_2) \vee \neg R(g(y_2))) \wedge (R(y_2) \vee R(g(y_2)))]
\end{aligned}$$

節集合

述語論理式をスコーレム標準形:

$$\forall x_1 \cdots \forall x_m [C_1 \wedge \cdots \wedge C_n]$$

で表したとき, 各節からなる集合 $C := \{C_1, \cdots, C_n\}$ を節集合 (clause set) という.

導出

素式 p を含む節 C_i とその否定形 $\neg p$ を含む節 C_j から, 新しい節 C_{ij} を作り出す操作を導出という.

$$\begin{array}{ccc}
C_i = p \vee q_1 \vee \cdots q_m & & C_j = \neg p \vee r_1 \vee \cdots \vee r_n \\
\qquad \downarrow & & \qquad \swarrow \\
C_{ij} = q_1 \vee \cdots \vee q_m \vee r_1 \vee \cdots \vee r_n & &
\end{array}$$

このとき, C_i と C_j を親節 (parent clause), C_{ij} を導出節 (resolvent clause) という.

導出原理

節集合 C より適当な節 C_i と C_j を選び, これらを親節とする導出節 C_{ij} を求め, C に加える ($C := C \supset \{C_{ij}\}$). この操作を繰り返して空節が得られれば, C は充足不能である. 逆も成立する.

導出の制御戦略

- 横形優先戦略
- 線形導出

導出グラフ (resolution graph)

導出によって充足不能性を証明する様子を表すグラフ. 特に, 空節を根に持つ木であるとき, 導出反駁木 (refutation tree) という.

14.3　ホーン節

ホーン節とホーン集合

- ホーン節：

  “$B \leftarrow A_1, \cdots, A_n$” または “$\leftarrow A_1, \cdots, A_n$”

  の形で表される節で, それぞれ, $\neg A_1 \vee \cdots \vee \neg A_n \vee B$, $\neg A_1 \vee \cdots \vee \neg A_n$ を意味する. $n = 0$ でもよい.

- ホーン集合：ホーン節からなる集合.

ホーン集合に対する導出戦略：**SNL, SLD**

1. 最初の導出の一方の親節として, ゴール節を選ぶ.
2. 次の導出を行う:

$$\begin{array}{cc} \leftarrow B, C_1, \cdots, C_m & B \leftarrow A_1, \cdots, A_n \\ \quad\downarrow & \quad\swarrow \\ \leftarrow A_1, \cdots, A_n, C_1, \cdots, C_m & \end{array}$$

3. 得られた節を一方の親節として, 導出を繰り返す.

例 14.3.1 動物を治療できるのは獣医である. 獣医になるには資格が必要である. 太郎は獣医の資格を持っている. 犬は動物である. ジョンは犬である. 太郎はジョンを治療できることを証明したい.

次の述語を用意する:

$$\begin{array}{rcl} T(x,y) & : & x \text{ は } y \text{ を治療できる;} \\ V(x) & : & x \text{ は獣医である;} \\ Q(x) & : & x \text{ は獣医の資格を持っている;} \\ A(x) & : & x \text{ は動物である;} \\ D(x) & : & x \text{ は犬である.} \end{array}$$

$$\begin{array}{cc} \leftarrow T(Taro, John) & T(x_1, y_1) \leftarrow V(x_1), A(y_1) \\ \quad\downarrow & \quad\swarrow \\ \leftarrow V(Taro), A(John) & V(x_2) \leftarrow Q(x_2) \\ \quad\downarrow & \quad\swarrow \\ \leftarrow Q(Taro), A(John) & Q(Taro) \leftarrow \\ \quad\downarrow & \quad\swarrow \\ \leftarrow A(John) & A(x_3) \leftarrow D(x_3) \\ \quad\downarrow & \quad\swarrow \\ \leftarrow D(John) & D(John) \leftarrow \\ \quad\downarrow & \quad\swarrow \\ \square & \end{array}$$

前向き推論と後向き推論

- 前向き推論 (forward reasoning) : 事実 P と推論規則 $P \to Q$ から結論 Q を得る.
- 後向き推論 (backward reasoning) : 結論 Q の否定 $\neg Q$ を仮定し, $\neg Q$ と $P \to Q$ から $\neg P$ を得る. 事実 P が存在すれば矛盾が生じるので $\neg Q$ が棄却され, Q が成り立つ.

SNL は, 後向き推論と等価である.