オペレーションズ・リサーチ

Operations Research: Models, Algorithms, and Implementations

劉子昂

2025-11-25

目次

Preface

OR を学ぶための教科書です.

注意点

随時更新していきますので, 読む際は必ずブラウザをリロードしてください.

Python

すべてのモデル, アルゴリズムについて，Python での実装例を示します. 講義では Python プログラミングについては扱いませんが，Python の基礎的な知識があると理解が深まります.

以下の手順で, 教材に掲載されているコードを Google Colab 上で簡単に実行できます. ぜひ試してみてください.

1. Google アカウントにログインする
2. Google Colab にアクセスする
3. 「+新しいノート」をクリックする
4. コードをコピーして, セルに貼り付ける
5. セルを実行する

フィードバック

継続的に改善していきますので, 誤字脱字, 内容の不備, わかりにくい箇所などを見つけたら，劉子昂までご連絡ください.

内容

回	内容
1	ガイダンス

回	内容
2	在庫モデル：EOQ モデル
3	在庫モデル：新聞売り子問題
4	在庫モデル：安全在庫
5	AHP
6	PERT 1
7	PERT 2
8	PERT 3
9	包絡分析法
10	動的計画法
11	待ち行列 1
12	待ち行列 2
13	待ち行列 3

講義

本授業では、オペレーションズ・リサーチ（OR）の中の代表的な手法である PERT、在庫理論、待ち行列理論、動的計画法、階層分析法、及び包絡分析法の数理を理解し、具体的な問題への応用を学ぶ。

講義情報

- 講義名：オペレーションズ・リサーチ B
- 曜日：水曜日
- 時限：2 時限目（10:50〜12:30）
- 教室：西館 W202
- 担当教員：劉 子昂
- Google Classroom: Link

出席

不定期に出席を取ります。

出席は WebClass 上で取ります。以下の時間に注意してください。

- 出席扱い：10:50 - 11:09
- 遅刻扱い：11:10 - 11:29
- 欠席扱い：11:30 -

成績評価

- 期末試験（100%）
- 4 回以上の欠席は単位取得不可。
- 講義中の加点問題に正解した場合、試験の点数に加点します。

到達目標

各分野について、下記の事項を目標して講義を行う。

- **PERT** の計算と解析方法を理解し、プロジェクトの評価を行うことができる。
- **在庫モデル**を理解し、自分で式を構築及び解析することができる。
- **待ち行列理論**の重要な式や定理を理論的に導出し、それらを適切に解釈することができる。
- **動的計画法**の基本的な考え方を理解し、簡単な問題への適用ができる。
- **階層分析法**による意思決定の手法を理解し、一対比較行列からウェイトと整合性を計算することができる。
- **包絡分析法**における CCR モデルを理解し、得られた結果を解釈することができる。

さらに、これらの手法を用いて比較的簡単な現象をモデル化し、解析することができる。

OR は重要

経営工学において、最も重要な学問分野の一つ。

日本経営工学会によると、「解決すべき課題の数理モデルを構築し、最適な手法を求めるオペレーションズ・リサーチ（OR）という分野は、経営工学の主要なテーマとなっています」。

海外では、管理科学（Management Science）と OR は同義語として使われることもよくある。

OR の全体像

- 線形計画法
- 整数計画法
- 非線形計画法
- **動的計画法**
- グラフ理論・ネットワーク
- シミュレーション
- **在庫モデル**
- **待ち行列**
- **多基準意思決定分析**
- **プロジェクトマネジメント**

- …

OR は難しい？

基礎知識が必要

微分積分、線形代数、確率、統計の基礎知識が必要です。これらの基礎が不十分な場合、授業についていけないです。基礎知識が不十分な場合、必ず復習してください。

この講義では、以下の工夫をしています。

- 付録に必要な基礎知識のまとめがあります。随時更新しますので、参考にしてください。
- 講義資料には例題、図、演習問題を多く用意しています。
- プログラミングの実装例も示します。

数式が多い

OR は、問題を数理的にモデル化し、解析する学問です。数式をたくさん使い、証明も多いです。数式を読むのが苦手な人は、慣れるまで大変かもしれません。

この講義では、以下の工夫をしています。

- 証明は省略なく丁寧に行います。
- わかりにくいところをコラムで補足します。

何を学ぶ？

- モデル：現実の問題を数理的に表現したもの
- モデリング：現実の問題を数理モデルで表現すること
- 解：問題の答え
- 最適化：最適解を見つけること
- アルゴリズム：問題を解く手順

単に，結論・定理を覚えるだけでなく，モデリング，証明，アルゴリズムの理解が重要です。

Python について

- すべてのアルゴリズム、モデルを Python で実装します。
- Python の基礎については、この講義では説明しません。
- Python のコードを理解できなくても、授業内容の理解には支障ありません。
- Youtube などでは、2-3 時間で Python 入門の動画がたくさんあります。事前に学習しておくことをお勧めします。

授業時間外の学習

本授業の準備・復習等の授業時間外学習は、4 時間を標準とする

必要なもの

- 本講義では, 受講者自身のノート PC を用いて演習を行います. 毎週必ずノート PC を持参してください.

私語

- 講義中の私語は厳禁です.
- 注意してもやめない場合は、減点を行います。

記号

集合

A 集合。大文字で表す。
A B 集合 A と集合 B の差集合。

線形代数

記号	意味
$\mathbf{x}$, $\mathbf{y}$	ベクトル。太字の小文字で表す。
$\mathbf{A}$, $\mathbf{B}$	行列。太字の大文字で表す。
$\mathbf{I}$	単位行列。

確率統計

X: 確率変数。大文字で表す。

$\mathbb{P}(X = x)$: 確率変数 X が x をとる確率

$\mathbb{E}[X]$: 確率変数 X の期待値

$F_X(x)$: 確率変数 X の累積分布関数

$f_X(x)$: 確率変数 X の確率密度関数

$p_X(x)$: 確率変数 X の確率質量関数

$\phi(z)$: 標準正規分布の確率密度関数

関数

A は要素の間に順序が定義された集合とする。

$\max A$: 集合 A の最大値。

$\min A$: 集合 A の最小値。

$(x)^+$: x と 0 のうち大きい方。すなわち、$(x)^+ = \max(x, 0)$。

$(x)^-$: x と 0 のうち小さい方の絶対値。すなわち、$(x)^- = \max(-x, 0) = -\min(x, 0)$。

例題

例 0.1 (差集合)**.** $A = \{1, 2, 3\}$、$B = \{2, 3, 4\}$ とする。このとき、$A \setminus B = \{1\}$ である。

例 0.2 (最大値と最小値)**.** $A = \{1, 0, -1\}$ とする。このとき、$\max A = 1$、$\min A = -1$ である。

例 0.3 (正の部分)**.** $(10)^+ = 10$、$(-30)^+ = 0$ である。

例 0.4 (負の部分)**.** $(10)^- = 0$、$(-30)^- = 30$ である。

第I部

在庫モデル

第1章　在庫管理とは

予備知識

- 微分積分
- 確率統計

商店・工場・倉庫などで、原材料・部品・製品などを適切に管理することを**在庫管理**（Inventory Management）という。一般的に、在庫管理の目的は、顧客の需要を満たしつつ、在庫に関わる費用を最小化することである。

ノート

豊田自動車が提唱した**ジャストインタイム**（Just In Time, JIT）は、生産方式としてよく知られている。
JIT とは必要なものを、必要な時に、必要な量だけ生産することである。
JIT の目的は、在庫を最小限に抑え、効率的な生産を実現することである。
アメリカの研究者らは、その生産方式を体系化し、**リーン生産方式**（Lean Manufacturing）という概念を提唱した。

在庫量が多すぎると、保管費用がかかる。逆に、在庫量が少なすぎると、欠品が発生し、顧客の需要を満たせなくなる。在庫管理は次の二つの問題を決定する。

1. どのくらいの量を発注するか？（発注量）
2. いつ発注するか？（発注時期）

科学的在庫管理（Scientific Inventory Management）では、これらの問題に答えるために、次の手順で在庫管理を行う。

1. 在庫システムを数学モデルとして定式化する。
2. 最適な発注量と発注時期を決定する。

練習 1.1. 前回スーパーに行ったときに買った商品（例えば、牛乳、卵など）について考える。需要と在庫の観点から、次の質問に答えよ。

1. どのくらいの量を買ったか？
2. なぜその量を買ったのか？
3. どのタイミング・頻度でその商品を買うか？

1.1　在庫量

需要（demand） ある期間に顧客が購入したい商品の量。通常、d で表す。
手持ち在庫（on-hand inventory） ある時点で、実際に手元にある在庫の量。OH で表す。
バックオーダー（backorder） 手持ち在庫がなく、満たせない需要。BO で表す。
在庫量（inventory level） ある時点での在庫の量。I で表す。

一般、BO と OH は次のように表される。

$$OH = I^+ = \max(0, I)$$

$$BO = I^- = \max(0, -I)$$

例 1.1. 手持ち在庫が $OH = 50$、需要が $d = 30$ のとき、在庫量は次のように計算される。

$$I = 50 - 30 = 20$$

手持ち在庫が $OH = 50$、需要が $d = 70$ のとき、在庫量は次のように計算される。

$$I = 50 - 70 = -20$$

このとき、手持ち在庫は

$$OH = I^+ = \max(0, I) = 0$$

となる。バックオーダーは

$$BO = I^- = \max(0, -I) = 20$$

となる。

1.2 在庫モデルの分類

在庫モデルは、次のような要素で分類される。

需要（demand） 需要が決定論的 (Deterministic) か確率的（Stochastic）か。

観測（review） 在庫量を連続観測 (Continuous Review) するか、周期観測 (Periodic Review) するか。連続観測の場合、在庫量が連続的に観測でき、いつでも発注が可能である。周期観測の場合、一定の期間（例えば 1 週間）ごとに在庫量を観測する。

リードタイム（lead time） 発注から納品までの期間。調達期間とも呼ばれる。リードタイムが決定論的か確率的か。また、リードタイムが 0 かどうか。在庫モデルを単純化するために、リードタイムを 0 とし、発注から納品までの期間を無視することもある。

バックオーダー（backorder） バックオーダーが許容されるかどうか。需要が手持ち在庫を上回った場合、バックオーダーが許容されると、欠品が発生しても、後で需要を満たすことができる。バックオーダーが許容されない場合、欠品が発生すると、上回った需要は失われ、機会損失が発生する。

計画期間（planning horizon） 単一期間 (Single Period) か、複数期間 (Multi Period) か、無限 (Infinite) か。

以下の表に、需要と観測に基づく、古典的な在庫モデルを示す。

在庫モデル	需要	観測
EOQ モデル	決定論的	連続観測
Wagner-Whitin	決定論的	周期観測
安全在庫	確率的	連続観測
新聞売り子問題	確率的	周期観測

1.3 在庫の費用

ここでは、在庫に関わる費用を紹介する。

発注費用（ordering cost） 発注量に関わらず、1 回の発注にかかる費用。調達費用、固定費用（fixed cost）などとも呼ばれる。通常、1 回の発注にかかる費用を K とする。

購入費用（purchase cost） 商品を購入するためにかかる費用。通常、単位あたりの購入費用を c とする。

欠品費用（stockout cost） 需要が手持ち在庫を上回った場合に発生する費用。通常、単位あたりの欠品費用を p とする。

保管費用（holding cost） 在庫を保管するためにかかる倉庫費用、保険費用、税金、機会費用など。通常、単位時間あたりの 1 単位あたりの保管費用を h とする。

例 1.2 (発注費用と購入費用)**.** 毎回の発注量を Q、1 回の発注にかかる費用を K、単位あたりの購入費用を c とする。1 回の発注にかかる総費用は、次のように計算される。

$$K + cQ$$

となる。

例 1.3 (欠品費用)**.** 在庫量を I、需要を d、単位あたりの欠品費用を p とする。欠品費用は次のように計算される。

$$p(d - I)^+$$

$p = 10$、$I = 50$、$d = 70$ のとき、欠品費用は次のように計算される。

$$p(d - I)^+ = 10(70 - 50)^+ = 200$$

$p = 10$、$I = 50$、$d = 20$ のとき、欠品費用は次のように計算される。

$$p(d - I)^+ = 10(20 - 50)^+ = 0$$

例 1.4 (在庫量が一定の保管費用)**.** 1 日あたり 1 単位の在庫を保管するために、h の費用がかかるとする。30 日間、50 単位の在庫を保管するための総保管費用を計算せよ。

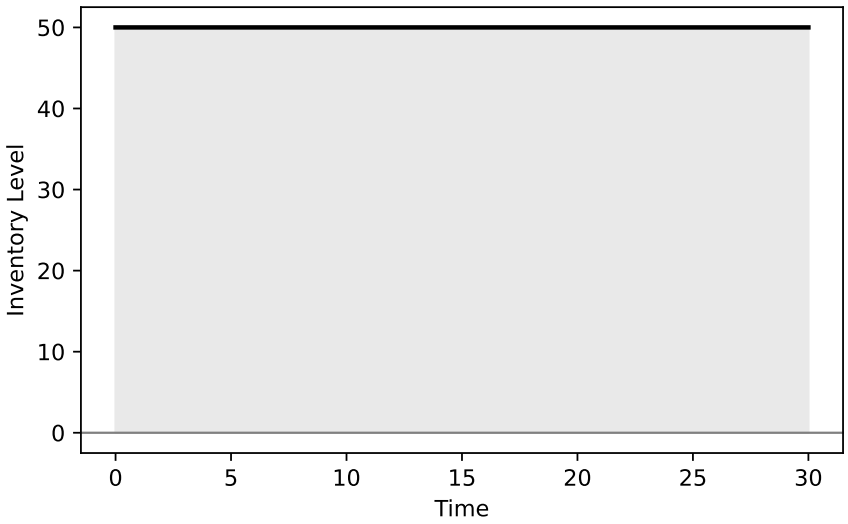
保管費用は次のように計算される。

$$30 \times 50 \times h = 1500h$$

下の図では、横軸が時間、縦軸が在庫量を表す。

```
import matplotlib.pyplot as plt
import numpy as np

t = np.linspace(0, 30, 1000)
inventory = np.full_like(t, 50)
```

```
# Plotting the inventory level
plt.fill_between(t, inventory, color="lightgray", alpha=0.5, label="Inventory Level")
plt.plot(t, inventory, label="Inventory Level", color="black", linewidth=2)
plt.xlabel("Time")
plt.ylabel("Inventory Level")
plt.axhline(0, color="gray", linewidth=1)
plt.tight_layout()
plt.show()
```



一般的に、保管費用は次の式で計算される。

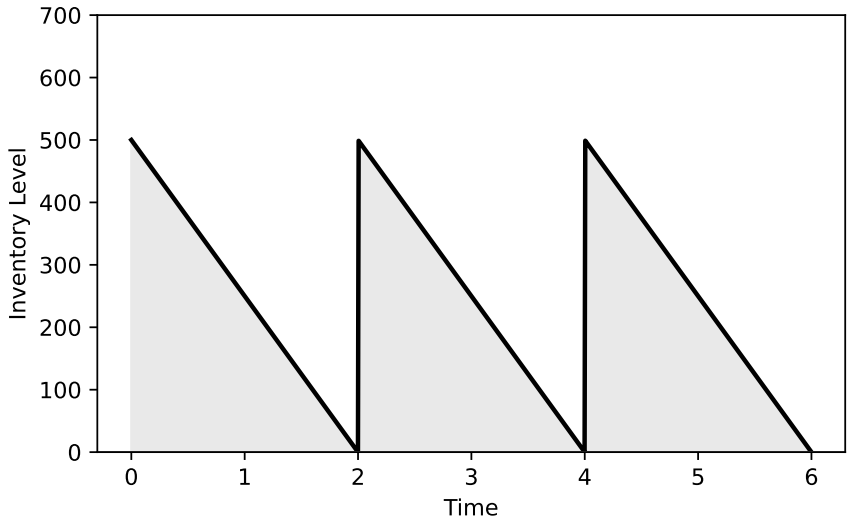
$$\text{保管費用} = \text{面積} \times h$$

例 1.5 (在庫が時間とともに変化する保管費用). 通常、在庫量が定数ではなく、時間とともに変化する。ここでは、在庫量が時間とともに線形に減少し、0 になると在庫が補充される場合を考える。毎回の発注量を 500 とする。

下の図に示すように在庫量が時間とともに変化するとする。6 日間の保管費用を計算せよ。

```
# Parameters
d = 250  # Demand rate
Q = 500  # Order quantity
T = Q / d  # Cycle length
t = np.linspace(0, 2.999 * T, 1000)
```

```
# Inventory level over time
inventory = np.maximum(0, Q - (d * t) % Q)

# Plotting the inventory level
plt.fill_between(t, inventory, color="lightgray", alpha=0.5, label="Inventory Level
plt.plot(t, inventory, label="Inventory Level", color="black", linewidth=2)
plt.xlabel("Time")
plt.ylabel("Inventory Level")
plt.axhline(0, color="gray", linewidth=1)
plt.ylim(bottom=0, top=Q + 200)
plt.tight_layout()
plt.show()
```



保管費用は 面積 $\times\, h$ で計算される。それぞれの三角形の面積は $\frac{1}{2} \times 500 \times 2$ であるため、6 日間の保管費用は次のように計算される。

$$\frac{2 \times 500}{2} \times 3 \times h$$

1.4　在庫方策

確率的在庫モデルにおいて、一つ重要な概念は**在庫方策**（inventory policy）である。在庫方策は、在庫の状況に応じて、在庫管理のルールを定めるものである。代表的な在庫方策を以下に示す。

1. (r, Q) 方策：在庫量を連続的に観測し、在庫量が発注点 r 以下になったときに発注量 Q を発注する方式である。**発注点方式**とも呼ばれる。
2. BSP 方策(Base Stock Policy)：在庫量を定期的に観測し、在庫量が基準在庫 S 以下になったときに、在庫量を S まで補充する方式である。**定期発注方式**とも呼ばれる。
3. (s, S) 方策：在庫量を定期的に観測し、在庫量が発注点 s 以下になったときに、在庫量を補充点 S まで補充する方式である。

一部の確率的在庫モデルにに対し、これらの在庫方策は**最適**であることが知られている。その場合、在庫方策が持つパラメータを最適化することで、在庫の期待コストを最小化することができる。

1.5 練習問題

1. 在庫管理の目的は、_**を満たしつつ、**_を最小化することである。
2. 在庫管理の決定すべき二つの問題は、_**と**_である。
3. 在庫量 $I = -30$, 需要 $d = 10$ のとき、手持ち在庫 OH とバックオーダー BO を計算せよ。
4. 1 回の発注にかかる費用 $K = 1000$、単位あたりの購入費用 $c = 50$、発注量 $Q = 200$ のとき、3 回の発注にかかる総費用を計算せよ。

1.6 挑戦

1. Python に関する入門書やビデオを参考にして、Python の基礎を学び、簡単なプログラムを書けるようにせよ。

YouTube で “Python 入門” などのキーワードで検索すると、多くの 2-3 時間入門のビデオが見つかる。

第2章　経済的発注量

経済的発注量（EOQ: Economic Order Quantity）モデルは、最も基本的な在庫管理モデルの一つである。Harris (1990) がこのモデルを最初に提案した。

EOQ モデルは、単位時間あたりの需要量は決定論的で、一定であると仮定する。すなわち、需要量は事前に分かっており、時間とともに変化しない。単位時間あたりの需要量は需要率（demand rate）と呼ばれ、記号 d で表される。リードタイムは 0 とし、発注から納品までの時間はないと仮定する。一回の発注量を Q とし、一定であるとする。欠品は許せないとする。全ての需要は満されなければならない。また、EOQ モデルでは、在庫量は連続的に観測され、いつでも発注が可能であるとする。

在庫に関わる費用は、1 回あたりの発注費用 K、単位時間あたりの 1 単位の保管費用 h と、購入単価 c がある。

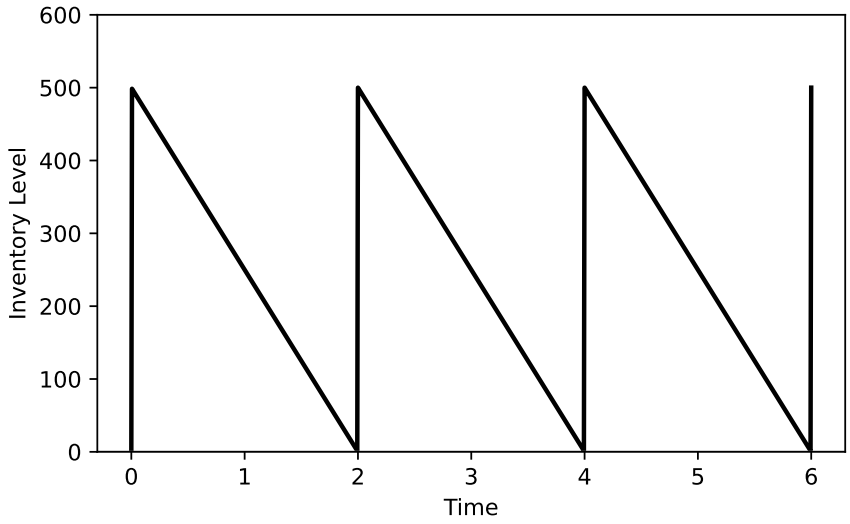
EOQ モデルの最適解は次の二つの性質を持つ (Snyder と Shen 2019)：

1. Zero-inventory ordering (ZIO). 在庫量が 0 のときに発注を行う。リードタイムは 0 であるため、在庫量が 0 でないときに発注すると、保管費用が発生する。
2. Constant order sizes. 発注量は一定である。需要率 d が一定であり、在庫量が 0 のときに発注を行うため、最適発注量も一定である。

以上の性質から、在庫量の時間的変化は下図のようになる。

```
import matplotlib.pyplot as plt
import numpy as np

# Parameters
d = 250  # Demand rate
Q = 500  # Order quantity
T = Q / d  # Cycle length
t = np.linspace(0, 3 * T, 1000)  # Time from 0 to 3 cycles

# Inventory level over time
```

```
inventory = np.maximum(0, Q - (d * t) % Q)
inventory[0] = 0

# Plotting the inventory level
plt.plot(t, inventory, label="Inventory Level", color="black", linewidth=2)
plt.xlabel("Time")
plt.ylabel("Inventory Level")
plt.axhline(0, color="gray", linewidth=1)
plt.ylim(bottom=0, top=Q + 100)
plt.tight_layout()
plt.show()
```



発注の間隔を**サイクル** (cycle) と呼び、サイクル期間は

$$T = \frac{Q}{d}$$

で与えられる。

例 2.1. A 社は、毎月 250 個の需要がある商品を取り扱っている。一回の発注量は 500 個とし、サイクル期間は

$$T = \frac{500}{250} = 2\text{ヶ月}$$

となる。

2.1 記号

記号	意味
d	単位時間あたりの需要量
K	1 回あたりの発注費用
h	単位時間あたりの 1 単位の保管費用
c	購入単価
Q	発注量
T	サイクル期間
$g(Q)$	平均コスト

2.2 コスト関数

ここでは、1 サイクルあたりのコストを考える。

発注費用：発注は 1 回だけ行うため、発注費用は K である。

購入費用：Q 個の商品を単価 c で購入するため、購入費用は cQ である。

保管費用：サイクル期間 T は $\frac{Q}{d}$ であるため、1 サイクルあたりの保管費用は

$$\frac{TQ}{2}h = \frac{hQ^2}{2d}$$

となる。

以上より、1 サイクルあたりのコストは次のように表される。

$$K + cQ + \frac{hQ^2}{2d}$$

平均コストは、これをサイクル期間 T で割ったものとして定義される。したがって、平均コスト $g(Q)$ は次のように表される。

$$\begin{aligned} g(Q) &= \frac{1}{T}\left(K + cQ + \frac{hQ^2}{2d}\right) \\ &= \frac{d}{Q}\left(K + cQ + \frac{hQ^2}{2d}\right) \\ &= \frac{Kd}{Q} + cd + \frac{hQ}{2} \end{aligned}$$

以上より、平均コストは発注量 Q の関数として次のように表される。

$$g(Q) = \frac{Kd}{Q} + cd + \frac{hQ}{2}$$

2.3　最適発注量

EOQ モデルの目的は、平均コスト $g(Q)$ を最小化する発注量 Q を求めることである。

平均コストの導関数 $g'(Q)$ が 0 となる点を求めることで、最適発注量 Q^* を求めることができる。

$$g'(Q) = -\frac{Kd}{Q^2} + \frac{h}{2} = 0$$

これを解くと、最適発注量

$$Q^* = \sqrt{\frac{2Kd}{h}}$$

を得る。これを **EOQ 公式**（EOQ formula）と呼ぶ。Q^* を経済的発注量と呼ぶ（経済的は最適という意味である）。

二階導関数 $g''(Q)$ を求めて、最適発注量が最小値を与えることを確認する。

$$g''(Q) = \frac{2Kd}{Q^3} > 0$$

$g''(Q) > 0$ であるため、Q^* は最小値を与える。

最適発注量 Q^* を次の定理にまとめる。

定理 2.1. *EOQ モデルにおいて、最適発注量 Q^* は*

$$Q^* = \sqrt{\frac{2Kd}{h}} \tag{2.1}$$

で与えられる。

Q^* を用いて、最適なサイクル期間 T^* を求めることができる。

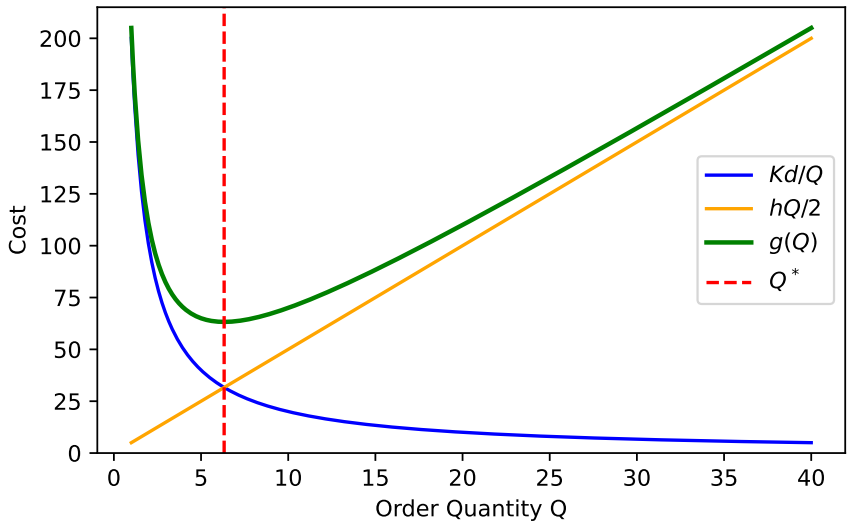
$$T^* = \frac{Q^*}{d} = \sqrt{\frac{2K}{hd}} \tag{2.2}$$

式 2.1 と 式 2.2 から、以下の性質がわかる。

1. h の増加に伴い、Q^* は減少する。保管費用が高い場合は、少量で高い頻度で発注することが望ましい。
2. K の増加に伴い、Q^* は増加する。発注費用が高い場合は、多量で低い頻度で発注することが望ましい。
3. d の増加に伴い、Q^* は増加する。
4. c は Q^* に影響しない。購入単価は最適発注量に影響しない。

次の図は、購入単価を $c = 0$ とするとき、平均コスト $g(Q)$、発注コスト $\frac{Kd}{Q}$、保管コスト $\frac{hQ}{2}$ のグラフを示す。

```
# Parameters
K = 40  # Order cost
h = 10  # Holding cost
d = 5  # Demand rate
Q = np.linspace(1, 40, 400)  # Order quantity from 1 to 40

# Cost calculations
order_cost = K * d / Q
holding_cost = (h * Q) / 2
average_cost = order_cost + holding_cost
optimal_Q = np.sqrt(2 * K * d / h)

# Plotting the costs
plt.plot(Q, order_cost, label=r"$Kd/Q$", color="blue")
plt.plot(Q, holding_cost, label=r"$hQ/2$", color="orange")
plt.plot(Q, average_cost, label=r"$g(Q)$", color="green", linewidth=2)
plt.axvline(optimal_Q, color="red", linestyle="--", label=r"$Q^*$")
plt.xlabel("Order Quantity Q")
plt.ylabel("Cost")
plt.ylim(bottom=0)
plt.legend()
plt.tight_layout()
plt.show()
```



平均コストが最小となる発注量 Q^* は、発注コストと保管コストの交差点である。すなわち、発注コストと保管コストを等しくする発注量は最適な発注量 Q^* である。この性質は以下の式からわかる。

$$\frac{Kd}{Q^*} = \frac{hQ^*}{2} \Longrightarrow Q^* = \sqrt{\frac{2Kd}{h}}$$

また、この図からもわかるように、Q の増加に伴い、平均発注コストは減少し、平均保管コストは増加する。逆もまた然りである。

例 2.2. ある電気量販店では、毎月 250 台の PC が販売されている。発注費用は 5000 円、保管費用は 1 台あたり月 150 円、購入単価は 10 万円とする。このとき、最適発注量 Q^* は次のように求められる。

$$Q^* = \sqrt{\frac{2 \cdot 5000 \cdot 250}{150}}$$

Excel では、下記のように計算できる。

```
=SQRT(2 * 5000 * 250 / 150)
```

Python では、次のように `eoq(K, d, h)` 関数を定義し、最適発注量を計算できる。

```
def eoq(K, d, h):
    """
    Calculate the Economic Order Quantity (EOQ).
```

```
    Parameters:
    K (float): Order cost
    d (float): Demand rate
    h (float): Holding cost

    Returns:
    float: Optimal order quantity Q*
    """
    return np.sqrt(2 * K * d / h)


if __name__ == "__main__":
    K = 5000  # Order cost
    d = 250  # Demand rate (units per month)
    h = 150  # Holding cost (per unit per month)

    Q_star = eoq(K, d, h)
    T_star = Q_star / d
    print(f"Optimal Order Quantity (Q*): {Q_star:.2f}")
    print(f"Optimal Cycle Time (T*): {T_star:.2f}")
```

```
Optimal Order Quantity (Q*): 129.10
Optimal Cycle Time (T*): 0.52
```

PC の場合は、注文量が整数である必要があるため、$g(129)$ と $g(130)$ を比較して最適発注量を決定する。

2.4 リードタイム*

EOQ モデルでは、リードタイムは 0 と仮定している。リードタイムが $L > 0$ の場合も、最適発注量 Q^* も変換せず、L 期間前に Q^* を発注すればよい。

ここでは、r を発注点(reorder point)とする。在庫量が r になったときに発注を行う。リードタイム L の間に需要が dL 個あるため、発注点は次のように表される。

$$r = dL$$

例 2.3. 上の例で、リードタイムが一週間とし、一か月を 4 週間とすると、リードタイムは $L = 1/4$ となる。したがって、発注点は次のように求められる。

$$r = dL = 250 \times \frac{1}{4} = 62.5$$

PC の在庫量が 63 台になったときに発注を行う。

2.5　他の EOQ モデル*

- バックオーダーを考慮した EOQ モデル
- 数量割引(quantity discount)を考慮した EOQ モデル
  - 総量割引(all-units discount)
  - 増分割引(incremental discount)

2.6　文献案内

オペレーションズ・リサーチに関する教科書の多くは、EOQ モデルを取り扱っている。モデルの分類から、「deterministic continuous-review inventory models」などの章で説明されていることが多い。

リードタイムを考慮した EOQ モデルについては、Snyder と Shen (2019) で説明されている。

数量割引を考慮した EOQ モデルについては、Snyder と Shen (2019) 、Camm ほか (2022) で説明されている。

バックオーダーを考慮した EOQ モデルについては、Snyder と Shen (2019) 、Camm ほか (2022) 、Hillier と Lieberman (2025) で説明されている。

2.7　練習問題

練習 2.1. ある会社は、毎月 300 個の需要がある商品を取り扱っている。一回の発注量は 600 個とし、サイクル期間 T を求めよ。

解答 2.1. 需要率は $d = 300$、発注量は $Q = 600$ である。したがって、サイクル期間 T は以下のように求められる。

$$T = \frac{Q}{d} = \frac{600}{300} = 2\text{ヶ月}$$

練習 2.2. ある工場は、鋼材を毎日 16 トン消費し、年間 250 日稼働している。鋼材の購入単価は 1 トンあたり 1100 ドル、1 回の発注にかかる固定費は 5500 ドル、保管費は鋼材 1 トンあたり年間は 275 ドルである。EOQ モデルにより以下を求めよ。

1. 最適な発注量
2. サイクル期間
3. 年間平均コスト

解答 2.2. 需要率は $d = 4000$、発注費用は $K = 5500$、保管費用は $h = 275$、購入単価は $c = 1100$ である。

```python
import numpy as np

K = 5500  # Order cost
d = 4000  # Demand rate (units per year)
h = 275  # Holding cost (per unit per year)
c = 1100  # Purchase cost (per unit)

Q_star = np.sqrt(2 * K * d / h)
T_star = Q_star / d
g_avg = (K * d / Q_star) + (c * d) + (h * Q_star / 2)
print(f"Optimal Order Quantity (Q*): {Q_star:.2f} tons")
print(f"Optimal Cycle Time (T*): {T_star:.2f} years")
print(f"Average Annual Cost (g(Q*)): {g_avg:.2f} dollars")
```

```
Optimal Order Quantity (Q*): 400.00 tons
Optimal Cycle Time (T*): 0.10 years
Average Annual Cost (g(Q*)): 4510000.00 dollars
```

第3章　安全在庫

需要 D がある確率分布に従うと仮定する。リードタイムを L とし、既知の定数とする。発注費用を K、単位あたりの保管費用を h とする。在庫量が連続的に観測され、いつでも発注が可能であるとする**連続観測**の場合を考える。

在庫管理には，(r, Q) 方策が用いられるとする。在庫量が発注点 r 以下になったときに、発注量 Q を発注する。この場合、与えられたサービスレベルを満たすように、発注点 r と発注量 Q を決定することが目的である。

> **ノート**
>
> リードタイム $L = 0$, 需要 d が一定の場合，EOQ モデルに帰着する。最適発注量は $Q^* = \sqrt{2Kd/h}$, 最適な発注点は $r^* = 0$ である。

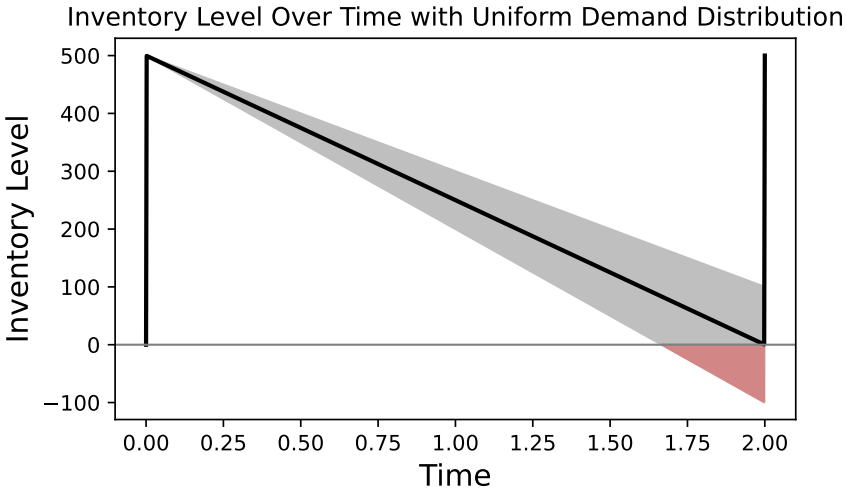
例 3.1 (思考実験)**.** 毎日の需要 D が連続一様分布 $U(200, 300)$ に従うと仮定する。毎日の平均需要は $\mu = 250$ である。以下の問題を考えよ。

1. 二日間の需要が従う分布を求めよ。
2. 初期在庫量は 500 であるとき, 欠品が発生する確率を求めよ。
3. 100% のサービスレベルを達成するために必要な初期在庫量を求めよ。

次の図は、在庫量の時間的変化を示す。灰色の領域は、需要の範囲を示す。赤い領域は、在庫量が 0 以下になったときの欠品を示す。黒い線は平均需要に基づく在庫量の変化を示す。

```
import scipy.stats as stats
import numpy as np
import matplotlib.pyplot as plt

# Parameters
d_mean = 250  # Mean demand rate
d_max = 300  # Max demand rate
d_min = 200  # Min demand rate
Q = 500  # Order quantity
T = Q / d_mean  # Average cycle length
```

```
# Simulate over multiple cycles to show repeated pattern
n_cycles = 1
t = np.linspace(0, n_cycles * T, 1000)

# Inventory levels: linear depletion over time
inventory_mean = Q - d_mean * (t % T)
inventory_mean[0] = 0
inventory_max = Q - d_max * (t % T)
inventory_min = Q - d_min * (t % T)

# Plotting
plt.plot(t, inventory_mean, label="Mean Demand", color="black", linewidth=2)
plt.fill_between(
    t, inventory_min, inventory_max, color="gray", alpha=0.5, label="Demand Range"
)

# Highlight when inventory drops below 0 (shortage)
plt.fill_between(
    t,
    inventory_max,
    0,
    where=(inventory_max < 0),
    color="red",
    alpha=0.3,
    label="Shortage",
)

# Aesthetics
plt.axhline(0, color="gray", linewidth=1)
plt.xlabel("Time", fontsize=14)
plt.ylabel("Inventory Level", fontsize=14)
plt.title("Inventory Level Over Time with Uniform Demand Distribution")
plt.tight_layout()
plt.show()
```



3.1 近似解法

> **ノート**
>
> この問題の定式化および厳密解法は、ここでは説明しない。Snyder と Shen (2019) の「Fundamentals of Supply Chain Theory」などの文献を参照されたい。以下は (r, Q) の近似解法を紹介する。

単位期間あたりの需要を D とし、D は正規分布 $N(\mu, \sigma^2)$ に従うと仮定する。

3.1.1 発注量 Q

D の平均需要 μ を EOQ モデルの需要率とみなすと、発注量 Q は次のように求めることができる (Camm ほか 2022)。

$$Q = \sqrt{\frac{2K\mu}{h}}$$

得られた発注量 Q は、最適解ではなく、近似解であることに注意されたい。

例 3.2. 単位期間あたりの需要 D が正規分布 $N(100, 20^2)$ に従うと仮定する。発注費用 $K = 200$、単位あたりの保管費用 $h = 5$ のとき、発注量 Q を求めよ。

$$Q = \sqrt{\frac{2K\mu}{h}} = \sqrt{\frac{2 \cdot 200 \cdot 100}{5}} \approx 89.44$$

Python では、以下のように計算できる。

```
import math

K = 200
h = 5
mu = 100
Q = math.sqrt((2 * K * mu) / h)
print(f"order quantity: {Q:.2f}")
```

```
order quantity: 89.44
```

3.1.2　発注点 r

リードタイム期間中に発生する需要は $D_L \sim N(\mu_L, \sigma_L^2)$ とし、正規分布の再生性により、

$$\mu_L = \mu L, \quad \sigma_L^2 = \sigma^2 L$$

になる。すなわち、リードタイム期間中の平均需要は $\mu_L = \mu L$、標準偏差は $\sigma_L = \sigma\sqrt{L}$ である。

> **正規分布の再生性**
>
> $X_1, X_2, \ldots, X_n$ が独立に同一の正規分布 $N(\mu, \sigma^2)$ に従うならば、$Y = X_1 + X_2 + \cdots + X_n$ は正規分布 $N(n\mu, n\sigma^2)$ に従う。

例 3.3. 単位期間あたりの需要 D が正規分布 $N(100, 20^2)$ に従うと仮定する。リードタイム $L = 4$ のとき、リードタイム期間中の平均需要 μ_L と標準偏差 σ_L を求めよ。

$$\mu_L = \mu L = 100 \cdot 4 = 400$$

$$\sigma_L = \sigma\sqrt{L} = 20\sqrt{4} = 40$$

これにより, リードタイム期間中の需要 D_L は正規分布 $N(400, 40^2)$ に従うことがわかる。

発注点 r を決めるためには、**サービスレベル**（service level）を考える。ここでは、サービスレベルを、リードタイム期間中に需要を満たす確率と定義する。サービスレベルを α とし、$0 < \alpha < 1$ とする。

与えられたサービスレベル α に対して、D_L が発注点 r 以下になる確率（欠品が発生しない確率、つまり、サービスレベル）が α になるように発注点 r を決定する。

$$P(D_L \leq r) = \alpha$$

もし、発注点 $r = \mu_L$ とすると、$P(D_L \leq \mu_L) = 0.5$ となる。すなわち、50% の確率で欠品が発生することになる。

> **ノート**
>
> $$P(D_L \leq \mu_L) = P\left(\frac{D_L - \mu_L}{\sigma_L} \leq 0\right) = \Phi(0) = 0.5$$

したがって、サービスレベル $\alpha > 0.5$ の場合、発注点 r は平均需要 μ_L より大きくなる必要がある。$r - \mu_L$ を**安全在庫**（safety stock）と呼び、s と表す。

$$s = r - \mu_L$$

この式を変形すると、発注点 r は次のように表される。

$$r = \mu_L + s$$

従って、サービスレベル $P(D_L \leq r) = \alpha$ は次のように表される。

$$P(D_L \leq \mu_L + s) = \alpha$$

与えられたサービスレベル α に対して、安全在庫 s を求めることを考える。ここで、$s \geq 0$ とする。

$$P(D_L - \mu_L \leq s) = \alpha \tag{3.1}$$

$$P\left(\frac{D_L - \mu_L}{\sigma_L} \leq \frac{s}{\sigma_L}\right) = \alpha \tag{3.2}$$

$$\Phi\left(\frac{s}{\sigma_L}\right) = \alpha \tag{3.3}$$

$$\frac{s}{\sigma_L} = \Phi^{-1}(\alpha) \tag{3.4}$$

$$s = \sigma_L \Phi^{-1}(\alpha) \tag{3.5}$$

$$s = \sigma\sqrt{L}\Phi^{-1}(\alpha) \tag{3.6}$$

ここで、$\Phi(\cdot)$ は標準正規分布の累積分布関数であり、$\Phi^{-1}(\alpha)$ はその逆関数である。したがって、発注点 r は次のように表される。

$$r = \mu_L + s = \mu L + \sigma\sqrt{L}\Phi^{-1}(\alpha)$$

$\Phi^{-1}(\alpha)$ は標準正規分布表、Excel、Python などを用いて求めることができる。

例 3.4. 単位期間あたりの需要 D が正規分布 $N(100, 20^2)$ に従うと仮定する。リードタイム $L = 4$、発注点 $r = 500$ のとき、安全在庫 s とサービスレベル α を求めよ。

リードタイム期間中の平均需要は $\mu_L = 400$ である。したがって、安全在庫 s は次のように求められる。

$$s = r - \mu_L = 500 - 400 = 100$$

サービスレベル α は次のように求められる。

$$\alpha = P(D_L \leq r) = \Phi\left(\frac{r - \mu_L}{\sigma_L}\right) = \Phi\left(\frac{100}{40}\right) = \Phi(2.5) \approx 0.99379$$

発注点 $r = 500$ のとき、安全在庫 s は 100、サービスレベル α は約 99.379% である。

例 3.5. リードタイム $L = 4$、平均需要 $\mu = 100$、需要の標準偏差 $\sigma = 20$、サービスレベル $\alpha = 0.95$ のとき、発注点 r と安全在庫 s を求める。

リードタイム期間中の平均需要と標準偏差は次のように計算される。

$$\mu_L = \mu L = 100 \cdot 4 = 400 \tag{3.7}$$
$$\sigma_L = \sigma\sqrt{L} = 20\sqrt{4} = 40 \tag{3.8}$$
$$\tag{3.9}$$

標準正規分布表から $\Phi^{-1}(0.95) \approx 1.64485$ を得る。これを用いて安全在庫 s と発注点 r を求める。

$$s = \sigma_L \Phi^{-1}(0.95) \approx 40 \cdot 1.64485 \approx 65.79 \tag{3.10}$$
$$r = \mu_L + s \approx 400 + 65.79 \approx 465.79 \tag{3.11}$$

したがって、発注点 r は約 465.79、必要な安全在庫 s は約 65.79 となる。

Python では、以下のように計算できる。

```python
from scipy.stats import norm

L = 4
mu = 100
sigma = 20
alpha = 0.95

mu_L = mu * L
sigma_L = sigma * (L ** 0.5)

s = sigma_L * norm.ppf(alpha)
r = mu_L + s

print(f"reorder point: {r:.2f},  safety stock: {s:.2f}")
```

```
reorder point: 465.79,  safety stock: 65.79
```

3.2 欠品費用を考慮する場合*

また、欠品費用も考慮する場合、バックオーダーを考慮した EOQ モデルを用いて、発注量 Q は次のように求めることができる (Hillier と Lieberman 2025)。

$$Q = \sqrt{\frac{2K\mu}{h}}\sqrt{\frac{p+h}{p}}$$

p は単位あたりの欠品費用である。

3.3　文献案内

3.4　練習問題

練習 3.1. ある会社は、A 商品を販売している。(r, Q) 方策に基づいて在庫管理を行っている。毎日の需要 D は正規分布 $N(150, 30^2)$ に従う。商品を発注するための固定費用は $K = 300$、単位あたりの保管費用は $h = 4$, 購入単価は $c = 20$ である。リードタイムは $L = 4$ 日である。この会社は，90% のサービスレベルを目標としている。このとき、発注量 Q、発注点 r、必要な安全在庫 s を求めよ。

解答 3.1. 発注量 Q は次のように求められる。

$$Q = \sqrt{\frac{2K\mu}{h}} = \sqrt{\frac{2 \cdot 300 \cdot 150}{4}} = 150$$

リードタイム期間中の平均需要と標準偏差は次のように計算される。

$$\mu_L = \mu L = 150 \cdot 4 = 600$$

$$\sigma_L = \sigma\sqrt{L} = 30\sqrt{4} = 60$$

標準正規分布表から $\Phi^{-1}(0.9) \approx 1.28155$ を得る。これを用いて安全在庫 s と発注点 r を求める。

$$s = \sigma_L \Phi^{-1}(0.9) \approx 60 \cdot 1.28155 \approx 76.89$$

$$r = \mu_L + s \approx 600 + 76.89 \approx 676.89$$

したがって、発注量 Q は 150、発注点 r は約 676.89、必要な安全在庫 s は約 76.89 となる。

```
import math
from scipy.stats import norm
K = 300
h = 4
mu = 150
sigma = 30
L = 4
alpha = 0.9
Q = math.sqrt((2 * K * mu) / h)
mu_L = mu * L
sigma_L = sigma * (L ** 0.5)
s = sigma_L * norm.ppf(alpha)
r = mu_L + s
print(f"order quantity: {Q:.2f}, reorder point: {r:.2f},  safety stock: {s:.2f}")
```

```
order quantity: 150.00, reorder point: 676.89,  safety stock: 76.89
```

第4章　Wagner-Whitin モデル

記号	意味
$\mathcal{T}$	期間の集合、$\mathcal{T} = \{1, 2, \dots, T\}$
K	1 回あたりの発注費用
h	単位あたりの保管費用
d_t	第 t 期の需要量
q_t	第 t 期の発注量
x_t	第 t 期の在庫量
y_t	第 t 期に発注する場合は 1、しない場合は 0
M	非負の大きな数

Wagner-Whitin モデルは次のように定式化される。

$$\text{minimize} \quad \sum_{t=1}^{T}(Ky_t + hx_t) \tag{4.1}$$

$$\text{subject to} \quad x_t = x_{t-1} + q_t - d_t \qquad \forall t \in \mathcal{T} \tag{4.2}$$

$$q_t \geq 0 \qquad \forall t \in \mathcal{T} \tag{4.3}$$

$$q_t \leq My_t \qquad \forall t \in \mathcal{T} \tag{4.4}$$

$$x_t \geq 0 \qquad \forall t \in \mathcal{T} \tag{4.5}$$

$$y_t \in \{0, 1\} \qquad \forall t \in \mathcal{T} \tag{4.6}$$

第5章　新聞売り子問題

これまで紹介した EOQ 在庫モデルは、需要が決定論的であると仮定していた。ここからは、需要が確率的であると仮定した在庫モデルを紹介する。

新聞売り子問題（Newsvendor Problem）は, 古典的な確率的在庫モデルの一つである。新聞は次の日には売れなくなるため, 新聞売り子問題は最も単純な perishable 在庫モデルとして知られている。新聞に限らず, 食品や花などの生鮮品の在庫管理にも応用される。

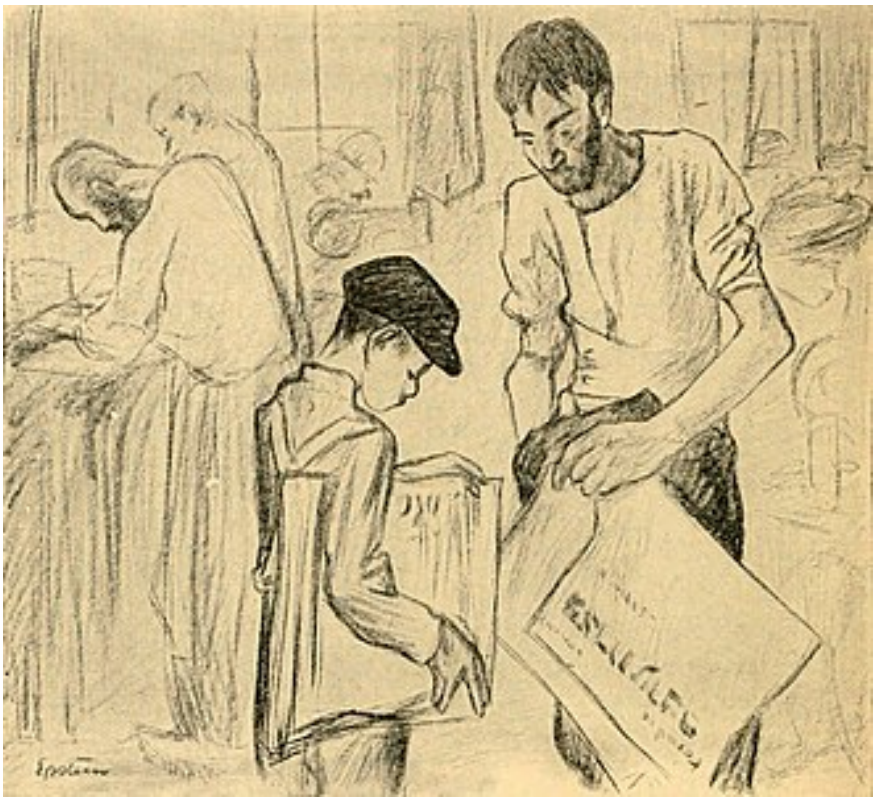


図 5.1: Jacob Epstein, Buying a Newspaper. The Spirit of the Ghetto, 1902, public domain, Wikimedia Commons

新聞売り子が新聞を仕入れ, 販売する問題を考える。新聞 1 部の欠品費用を p, 保管費用を h とする。新聞売子問題において, p を**在庫不足費用** (underage cost)，h を**在庫超過費用** (overage cost)とも呼ぶ。$p0$，$h0$ とし, 初期在庫は 0 とする。

新聞の需要 D を確率変数とするとき, 新聞売り子はどれだけの新聞を発注すればよいかという問題である。

例 5.1 (思考実験)**.** 需要 D が 一様分布 $U(100, 300)$ に従うとする。以下のそれぞれの場合, 新聞売り子がどれだけの新聞を発注するか考えよ。

- 在庫超過費用 $h = 1000$, 在庫不足費用 $p = 0.1$
- 在庫超過費用 $h = 0.1$, 在庫不足費用 $p = 1000$
- 在庫超過費用 $h = 10$, 在庫不足費用 $p = 10$

5.1　記号

記号	意味
h	1 個当たりの在庫超過費用
p	1 個当たりの在庫不足費用
D	需要(確率変数)
d	需要の観測値
$f_D(d)$	需要 D の確率密度関数
$F_D(d)$	需要 D の累積分布関数
S	発注量
$g(S, d)$	発注量 S, 需要の観測値 d に対するコスト
$g(S)$	$g(S, d)$ の期待値，$g(S) = \mathbb{E}[g(S, D)]$

5.2　定式化

新聞売り子が S 部の新聞を仕入れ, 需要 D が d であったとする。このとき, 新聞売り子のコスト $g(S, d)$ は以下のように表される。

$$g(S, d) = h(S - d)^+ + p(d - S)^+ \tag{5.1}$$

例 5.2. 発注量 $S = 100$, 需要 D の観測値 $d = 120$, 在庫超過費用が $h = 10$, 在庫不足費用が $p = 5$ のとき, コスト $g(S, d)$ は以下のように求められる。

$$g(100, 120) = 10(100 - 120)^+ + 5(120 - 100)^+ = 100$$

観測値 d が分かれば, コスト $g(S,d)$ を計算できる。しかし, 需要 D が確率変数であることを思い出そう。ここからは, 需要 D が連続型確率変数であると仮定する。

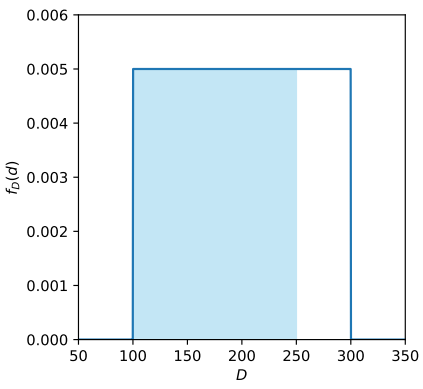
需要 D の確率密度関数を $f_D(d)$, 累積分布関数を $F_D(d)$ とする。発注量を S としたとき, 需要 $D \leq S$ である確率は, 累積分布関数 $F_D(S)$ で与えられる。

$$P(D \leq S) = F_D(S) = \int_0^S f_D(d)dd \tag{5.2}$$

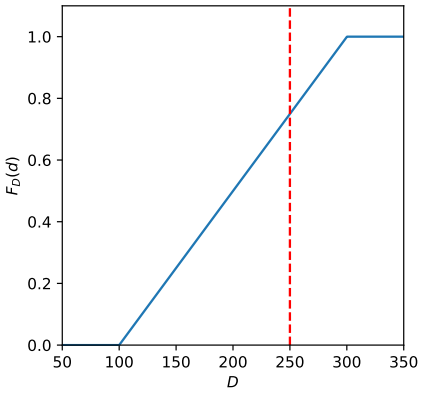
例 5.3. 需要 D が一様分布 $U(100, 300)$ に従うとき, 確率密度関数 $f_D(250)$ と累積分布関数 $F_D(250)$ は以下のように表される。

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import uniform

# 一様分布のパラメータ
a = 100
b = 300
# 需要の範囲
d = np.linspace(a - 50, b + 50, 1000)
# plot pdf with shaded area
plt.figure(figsize=(4, 4))
plt.plot(d, uniform.pdf(d, a, b - a))
plt.fill_between(
    d,
    0,
    uniform.pdf(d, a, b - a),
    where=(d <= 250),
    color="skyblue",
    alpha=0.5,
    label="F(250)",
)
plt.xlabel(r"$D$")
plt.ylabel(r"$f_D(d)$")
plt.xlim(a - 50, b + 50)
plt.ylim(0, 0.006)
plt.show()

# plot cdf
```

```
plt.figure(figsize=(4, 4))
plt.plot(d, uniform.cdf(d, a, b - a))
plt.axvline(250, color="r", linestyle="--", label="d=250")
plt.xlabel(r"$D$")
plt.ylabel(r"$F_D(d)$")
plt.xlim(a - 50, b + 50)
plt.ylim(0, 1.1)
plt.show()
```



(a) 確率密度関数 $f_D(d)$



(a) 累積分布関数 $F_D(d)$

需要 $D \leq 250$ である確率は, 累積分布関数 $F_D(250)$ で与えられる。

$$P(D \leq 250) = F_D(250) = \int_{100}^{250} f_D(d)dd = 0.75$$

需要 $D > 250$ である確率は $1 - F_D(250)$ で与えられる。

$$P(D > 250) = 1 - F_D(250) = 1 - \int_{100}^{250} f_D(d)dd = 0.25$$

D が確率変数であるため, コストも確率変数となる。新聞売り子問題の目的は, コストの期待値を最小化することである。

ここで, コストの期待値を $g(S) = \mathbb{E}[g(S, D)]$ とする。$g(S, D)$ は 式 5.1 で与えられるため, コストの期待値 $g(S)$ は以下のように表される。

$$
\begin{aligned}
g(S) &= \mathbb{E}[g(S, D)] && (5.3)\\
&= \mathbb{E}[h(S-D)^+ + p(D-S)^+] && (5.4)\\
&= \mathbb{E}[h(S-D)^+] + \mathbb{E}[p(D-S)^+] && (5.5)\\
&= h\mathbb{E}[(S-D)^+] + p\mathbb{E}[(D-S)^+] && (5.6)\\
& && (5.7)
\end{aligned}
$$

この式により, コストの期待値 $g(S)$ は，$(S-D)^+$ の期待値かける在庫超過費用 h と，$(D-S)^+$ の期待値かける在庫不足費用 p の和であることが分かる。

期待値の線形性(Linearity of Expectation)

X と Y を確率変数，a と b を定数とする。このとき，

$$\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

$$\mathbb{E}[aX+b] = a\mathbb{E}[X] + b$$

が成り立つ。

需要 D が連続型確率変数であるため，$\mathbb{E}[(S-D)^+]$ と $\mathbb{E}[(D-S)^+]$ は以下のように表される。

$$\mathbb{E}[(S-D)^+] = \int_0^\infty (S-d)^+ f_D(d)dd = \int_0^S (S-d) f_D(d)dd \quad (5.8)$$

$$\mathbb{E}[(D-S)^+] = \int_0^\infty (d-S)^+ f_D(d)dd = \int_S^\infty (d-S) f_D(d)dd \quad (5.9)$$

連続型確率変数の期待値

X を連続型確率変数，$f_X(x)$ を X の確率密度関数とする。このとき，X の期待値 $\mathbb{E}[X]$ は以下のように表される。

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x)dx$$

$\mathbb{E}[g(X)]$ は以下のように表される。

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x)dx$$

したがって, コストの期待値 $g(S)$ は以下のように表される。

$$g(S) = \mathbb{E}[g(S, D)] \tag{5.10}$$
$$= h\mathbb{E}[(S-D)^+] + p\mathbb{E}[(D-S)^+] \tag{5.11}$$
$$= h\int_0^S (S-d)f_D(d)dd + p\int_S^\infty (d-S)f_D(d)dd \tag{5.12}$$

例 5.4. 在庫超過費用が $h = 10$, 在庫不足費用が $p = 5$ のとき, 発注量 $S = 150$ に対するコストの期待値 $g(150)$ は以下のように求められる。

$$g(150) = 10\int_0^{150} (150-d)f_D(d)dd + 5\int_{150}^\infty (d-150)f_D(d)dd$$

5.3　最適化

以上の議論から, 新聞売り子問題は, コストの期待値 $g(S)$ を最小化する発注量 S を求める問題に帰着される。手順は以下の通りである。

1. $g(S)$ の 1 階微分を求める。
2. $dg(S)/dS = 0$ を解く。
3. 2 階微分を求め，$g(S)$ が凸関数であることを確認する。

式 5.2 を用い，$g(S)$ の 1 階微分は以下のように求める。

$$\frac{dg(S)}{dS} = h\int_0^S f_D(d)dd - p\int_S^\infty f_D(d)dd \tag{5.13}$$
$$= hF_D(S) - p(1-F_D(S)) \tag{5.14}$$
$$\tag{5.15}$$

よって，$dg(S)/dS = 0$ から,

$$hF_D(S) - p(1-F_D(S)) = 0 \tag{5.16}$$
$$F_D(S) = \frac{p}{h+p} \tag{5.17}$$

になる。2 階微分は

$$\frac{d^2g(S)}{dS^2} = hf_D(S) + pf_D(S) \tag{5.18}$$

$$= (h+p)f_D(S) > 0 \tag{5.19}$$

である。したがって, コスト関数 $g(S)$ は凸関数であり，1 階微分が 0 になる点は最小値を与える。

> **ヒント**
>
> $p > 0$，$h > 0$，$f_D(S) > 0$ より，$(h+p)f_D(S) > 0$ が成り立つ。

コスト関数 $g(S)$ を最小化するための最適発注量 S^* は

$$S^* = F_D^{-1}\left(\frac{p}{h+p}\right)$$

となる。ここで，F_D^{-1} は需要 D の累積分布関数の逆関数である。

> **逆関数(Inverse Function)**
>
> ある関数 $y = f(x)$ に対し, 次の条件を満たす関数 $x = f^{-1}(y)$ を $f(x)$ の逆関数と呼ぶ。
>
> $$f(f^{-1}(y)) = y, \quad f^{-1}(f(x)) = x$$

定理 5.1. 新聞売り子問題における最適発注量 S^* は,

$$S^* = F_D^{-1}\left(\frac{p}{h+p}\right)$$

で与えられる。

5.3.1 正規分布の場合

需要 D が正規分布 $N(\mu, \sigma^2)$ に従うとき, 新聞売り子問題の最適発注量 S^* は以下の式を満たす。

$$F_D(S^*) = \Phi\left(\frac{S^* - \mu}{\sigma}\right) = \frac{p}{h+p}$$

ここで,

$$z = \Phi^{-1}\left(\frac{p}{h+p}\right)$$

とおくと,

$$S^* = \sigma z + \mu$$

で与えられる。標準正規分布表を用いて z を調べることができる。

> **標準正規分布(Standard Normal Distribution)**
>
> 平均 0, 分散 1 の正規分布を**標準正規分布**（standard normal distribution)と呼ぶ。標準正規分布に従う確率変数を Y とすると，$Y \sim N(0,1)$ と表される。標準正規分布の確率密度関数を $\phi(z)$, 累積分布関数を $\Phi(z)$ と表す。
> 与えられた $X \sim N(\mu, \sigma^2)$ の累積分布関数 $F_X(x)$ の値を求めるには, 以下のように変換する。
>
> $$\begin{aligned} F_X(x) &= P(X \le x) && (5.20)\\ &= P\left(\frac{X-\mu}{\sigma} \le \frac{x-\mu}{\sigma}\right) && (5.21)\\ & && (5.22)\\ &= P\left(Y \le \frac{x-\mu}{\sigma}\right) && (5.23)\\ & && (5.24)\\ &= \Phi\left(\frac{x-\mu}{\sigma}\right) && (5.25) \end{aligned}$$
>
> 与えられた z に対し，$\Phi(z)$ の値は**標準正規分布表**（standard normal table)を用いて調べることができる。

5.4　臨界率

以上の議論から, 最適発注量 S^* は以下の式を満たす。

$$F_D(S) = P(D \le S) = \frac{p}{h+p}$$

ここで，$F_D(S) = P(D \le S)$ は需要 D が発注量 S 以下である確率を表す。言い換えると, 欠品が発生しない確率を表す。この確率のことは**サービスレベ**

ル（service level）と呼ぶ。定理 5.1 は，サービスレベルを $p/(h+p)$ に等しくする発注量 S が最適であることを示している。

この $p/(h+p)$ は**臨界率**（critical ratio）と呼ばれる。

注釈 5.1.

- 在庫不足費用 p の増加に伴い，サービスレベル $p/(h+p)$ は増加し，最適発注量 $S^* = F_D^{-1}(p/(h+p))$ も増加する。
- 在庫超過費用 h の増加に伴い，サービスレベル $p/(h+p)$ は減少し，最適発注量 $S^* = F_D^{-1}(p/(h+p))$ も減少する。
- 直感的に，在庫不足費用が増加すると，欠品を避けるために発注量が増加し，サービスレベルも上昇する。一方，在庫超過費用が増加すると，過剰在庫を避けるために発注量が減少し，サービスレベルも低下する。

5.5 例題

例 5.5. 需要 D が正規分布 $N(100, 25)$ に従う新聞売り子問題を考える。在庫超過費用が $h = 10$，在庫不足費用が $p = 40$ のとき，最適発注量 S^* を求める。

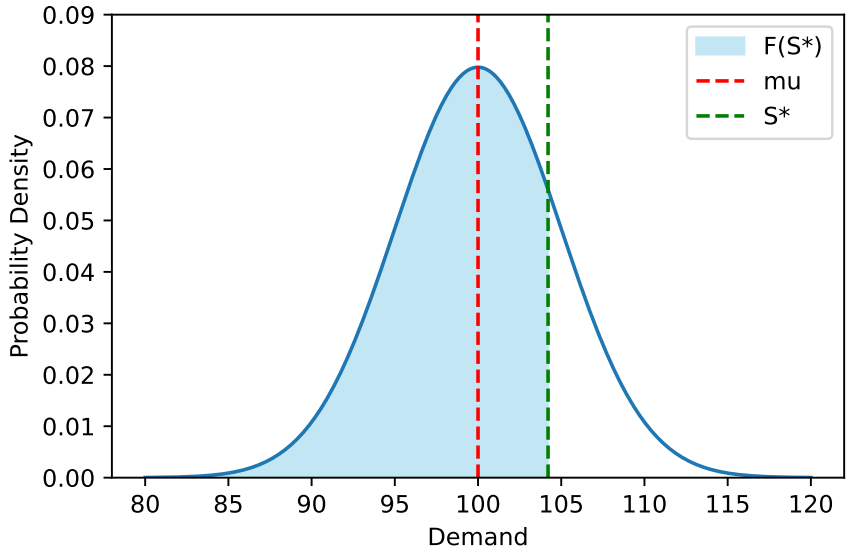
定理 5.1 より，S^* は以下の式で与えられる。

$$S^* = F_D^{-1}\left(\frac{p}{h+p}\right) = F_D^{-1}\left(\frac{40}{10+40}\right) = F_D^{-1}(0.8)$$

言い換えると，$F_D(S^*) = 0.8$ を満たす S^* を求めればよい。これを図で表すと，面積が 0.8 になるような S^* を求めることに相当する。

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm

# 正規分布のパラメータ
mu = 100
sigma = 5

# 需要の範囲
d = np.linspace(mu - 4 * sigma, mu + 4 * sigma, 1000)
S_star = norm.ppf(0.8, loc=mu, scale=sigma)

# plot the normal distribution
```

```
plt.plot(d, norm.pdf(d, mu, sigma))
plt.fill_between(
    d,
    0,
    norm.pdf(d, mu, sigma),
    where=(d <= S_star),
    color="skyblue",
    alpha=0.5,
    label="F(S*)",
)
plt.xlabel("Demand")
plt.ylabel("Probability Density")
plt.axvline(mu, color="r", linestyle="--", label="mu")
plt.axvline(S_star, color="g", linestyle="--", label="S*")
plt.ylim(0, 0.09)
plt.legend()
plt.show()
```



下では，Excel，Python, 標準正規分布表を用いて S^* を求める方法を示す。

5.5.1　標準正規分布表

標準正規分布表を調べると，$\Phi(z) = 0.8$ のとき，z は約 0.84 である。したがって，S^* は

$$S^* = z\sigma + \mu \approx 0.84 \times 5 + 100 = 104.2$$

となる。

5.5.2 Excel

Excel では，NORM.INV(確率, 平均, 標準偏差) 関数を用いて $F_D^{-1}(0.8)$ を求めることができる。

```
=NORM.INV(0.8, 100, 5)
```

5.5.3 Python

Python では，SciPy ライブラリの ppf() 関数を用いて, 逆関数を求めることができる。

```
from scipy.stats import norm

# 正規分布のパラメータ
mu = 100
sigma = 5

# 在庫超過費用と在庫不足費用
h = 10
p = 40

# 臨界率
critical_ratio = p / (h + p)

# 最適発注量
S_star = norm.ppf(critical_ratio, loc=mu, scale=sigma)
print(f"Optimal order quantity S*: {S_star:.2f}")
```

```
Optimal order quantity S*: 104.21
```

5.6 再定式化

新聞 1 部の仕入れ価格を c, 販売価格を r, 残存価額を v, 欠品費用を p, 保管費用を h とする。ここで, 以下の条件を満たすとする。

- $r > c$. 販売価格は仕入れ価格より高い。
- $r > v$. 販売価格は残存価値より高い。

新聞売り子が S 部の新聞を仕入れ, 需要 D が d であったとする。このとき, 新聞売り子の利益 $\pi(S, d)$ は以下のように表される。

5.6.1　コスト関数

$$\pi(S,d) = r\min\{d,S\} - cS + v\max\{0,S-d\} \tag{5.26}$$
$$- h\max\{0,S-d\} - p\max\{0,d-S\} \tag{5.27}$$

第一項は販売利益, 第二項は仕入れコスト, 第三項は残存価値, 第四項は保管コスト, 第五項は欠品コストである。

以下は，$\max\{0,x\} = x^+$ を用いて書き換えた形である。整理すると, 利益は以下のように表される。

$$\pi(S,d) = r\min\{d,S\} - cS + (v-h)(S-d)^+ - p(d-S)^+$$

第一項を以下のように書き換えることができる。

$$r\min\{d,S\} = rd - r(d-S)^+$$

> **ノート**
>
> - $d < S$ の場合，$r\min\{d,S\} = rd$ となる。
> - $d >= S$ の場合，$r\min\{d,S\} = rd - r(d-S) = rS$ となる。

したがって, 利益は以下のように書き換えられる。

$$\pi(S,d) = rd - cS + (v-h)(S-d)^+ - (p+r)(d-S)^+$$

利益の最大化は, コストの最小化に帰着される。したがって, コスト関数 $g(S,d) = -\pi(S,d)$ は以下のように表される。

$$g(S,d) = cS - rd + (h-v)(S-d)^+ + (p+r)(d-S)^+$$

D が確率変数であるため, コストの期待値 $g(S)$ は以下のように表される。

$$g(S) = \mathbb{E}[g(S, D)] \tag{5.28}$$
$$= \int_0^\infty g(S, d) f_D(d) dd \tag{5.29}$$
$$= cS - r\mathbb{E}[D] + (h - v)\mathbb{E}[(S - D)^+] + (p + r)\mathbb{E}[(D - S)^+] \tag{5.30}$$
$$= cS - r\mu + (h - v) \int_0^\infty (S - d)^+ f_D(d) dd + (p + r) \int_0^\infty (d - S)^+ f_D(d) dd \tag{5.31}$$
$$= cS - r\mu + (h - v) \int_0^S (S - d) f_D(d) dd + (p + r) \int_S^\infty (d - S) f_D(d) dd \tag{5.32}$$

5.6.2 最適化

$g(S)$ の 1 階微分は以下のように求める。

$$\frac{dg(S)}{dS} = c + (h - v)F_D(S) - (p + r)(1 - F_D(S)) \tag{5.33}$$
$$\tag{5.34}$$

よって，$dg(S)/dS = 0$ から，

$$c + (h - v)F_D(S) - (p + r)(1 - F_D(S)) = 0 \tag{5.35}$$
$$F_D(S) = \frac{p + r - c}{h + p + r - v} \tag{5.36}$$

になる。2 階微分は

$$\frac{d^2 g(S)}{dS^2} = (h - v) f_D(S) + (p + r) f_D(S) \tag{5.37}$$
$$= (h - v + p + r) f_D(S) \tag{5.38}$$

である。したがって, コスト関数 $g(S)$ は凸関数であり，1 階微分が 0 になる点は最小値を与える。

コスト関数 $g(S)$ を最小化するための最適発注量 S^* は

$$S^* = F_D^{-1}\left(\frac{p+r-c}{h+p+r-v}\right)$$

となる。ここで，F_D^{-1} は需要 D の累積分布関数の逆関数である。

新聞売り子問題において, より一般的に, 在庫超過費用(overage cost)と在庫不足費用(underage cost)を考慮する。

$$C_o = h + c - v, \quad C_u = p + r - c$$

5.6.3　臨界率

在庫超過費用 C_o は, 在庫が余ったときのコストである。1 部の在庫超過に対し, 保管コストと仕入れコストが発生するが, 残存価額が得られないため, $C_o = h + c - v$ となる。

在庫不足費用 C_u は, 在庫が不足したときのコストである。1 部の在庫不足に対し, 欠品コスト p と失われた販売機会の利益 $r - c$ が発生するため, $C_u = p + r - c$ となる。

従って,

$$\begin{aligned} S^* &= F_D^{-1}\left(\frac{p+r-c}{h+p+r-v}\right) \\ &= F_D^{-1}\left(\frac{C_u}{C_o + C_u}\right) \end{aligned}$$

が得られる。

5.7　初期在庫を考慮した新聞売り子問題*

新聞売り子の初期在庫を I とする。$I \leq S^*$ の場合, 最適発注量は $S^* - I$ となる。すなわち, 在庫量を S^* にすればよい。

また, $g(S)$ は凸関数であるため, $I > S^*$ の場合, 何も発注しないことが最適である。

したがって, 最適発注量は

$$Q = \begin{cases} S^* - I, & \text{if } I \leq S^*, \\ 0, & \text{if } I > S^*. \end{cases}$$

となる。

このような発注方式を **Base Stock Policy** (BSP) と呼ぶ。BSP は, 各期間の在庫量を観測し, 在庫量が S^* に引き上げられるように発注する方式である。新聞売り子問題において，BSP は最適な方策であると知られている。

5.8 文献案内

Arrow, Harris, と Marschak (1951) は, 新聞売り子問題を初めて定式化した。

Scarf (1959) の論文では，(s, S) 方策が発注費用を考慮した複数期間の新聞売り子問題において最適であることを示している。

実際には, 需要の分布が不明であることが多い。Huber ほか (2019) は, データ駆動新聞売り子問題(Data-Driven Newsvendor Problem)に関する研究が行われている。

Qin ほか (2011) は, 新聞売り子問題に関する研究をレビューした。

5.9 練習問題

練習 5.1. ある弁当屋では，1 個 500 円で弁当を仕入れ，1 個 800 円で販売している。売れ残った弁当を廃棄する場合，1 個当たり 10 円の廃棄費用がかかる。弁当の需要 D は, 正規分布 $N(50, 8^2)$ に従うとする。このとき, 最適な弁当の発注量を求めよ。

解答 5.1. 弁当 1 個当たりの在庫超過費用 h, 在庫不足費用 p は以下のように求められる。

$$h = 10 + 500 = 510, \quad p = 800 - 500 = 300$$

したがって，S^* は以下のように求められる。

$$S^* = F_D^{-1}\left(\frac{p}{h+p}\right) = F_D^{-1}\left(\frac{300}{510+300}\right) \approx F_D^{-1}\left(0.37\right)$$

標準正規分布表を調べると，$\Phi(z) = 0.37$ のとき，z は約 -0.33 である。したがって，S^* は

$$S^* = -0.33 \times \sigma + \mu = -0.33 \times 8 + 50 \approx 47.36$$

となる。

Python を用いて計算すると, 以下のようになる。

```
from scipy.stats import norm

# 正規分布のパラメータ
mu = 50
sigma = 8
# 在庫超過費用と在庫不足費用
h = 510
p = 300
# 臨界率
critical_ratio = p / (h + p)
# 最適発注量
S_star = norm.ppf(critical_ratio, loc=mu, scale=sigma)
print(f"Optimal order quantity S*: {S_star:.4f}")
```

```
Optimal order quantity S*: 47.3530
```

第6章　まとめ

- 在庫管理の目的は，**発注量**と**発注時期**を決定し, 顧客の需要を満たしつつ, コストを最小化することである.
- 科学的在庫管理では, 在庫モデルを定式化し, 発注量と発注時期を決定する.
- **在庫方策**は, 在庫の状況に応じて, 発注量と発注時期を決定するルールである．(s, S) 方策，(r, Q) 方策，BSP 方策などがある.
- 安全在庫の章では, コストを最小化するのではなく, サービスレベルを満たすための在庫方策を決定する方法を学んだ.

紹介した在庫モデルを分類すると, 次の表のようになる.

モデル名	分類
EOQ モデル	決定論的連続観測モデル
安全在庫	確率的連続観測モデル
Wagner-Whitin モデル	多期間決定論的周期観測モデル
新聞売り子問題	単一期間確率的周期観測モデル

第II部

多基準意思決定分析

第7章　階層分析法

予備知識

- 線形代数

AHP は, 複数の代替案の中から一つの代替案を選択するための手法である.

7.1　代替案

代替案の集合 $X = \{x_1, x_2, \dots, x_n\}$ とする. 代替案の重要度を

$$\mathbf{w} = (w_1, w_2, \dots, w_n)^T$$

で表す. ここで, w_i は代替案 x_i の重要度を表し, $\sum_{i=1}^{n} w_i = 1$ を満たす. AHP では, $\mathbf{w}$ を計算できるものとする. 計算する方法は後述する.

w_i が大きいほど, 代替案 x_i が望ましいとする. $w_i > w_j$ ならば, 代替案 x_i は代替案 x_j より望ましい. このとき, $x_i \succ x_j$ と表す.

例 7.1. 旅行先の代替案

$$X = \{\text{北海道}, \text{沖縄}, \text{九州}\}$$

とし, 各代替案の重要度を $\mathbf{w} = (0.3, 0.5, 0.2)^T$ とするとき,

$$\text{沖縄} \succ \text{北海道} \succ \text{九州}$$

である.

このように, それぞれの代替案の重要度が与えられるならば, 望ましい代替案の選択は容易である. しかし, 実際には, 代替案の重要度を直接決定することは難しい.

7.2　記号

記号	意味
$\mathbf{A}$	一対比較行列
X	代替案の集合
x_i	代替案 i
a_{ij}	一対比較値
$\mathbf{w}$	重要度ベクトル

7.3　一対比較

すべての代替案を並べて比較することは難しい. そこで，AHP では，2 つの代替案を比較する**一対比較** (pairwise comparison) を用いる. その理由として，$\{x_1, x_2, \ldots, x_n\}$ の中から最も望ましい代替案を選ぶことは難しいが，2 つの代替案 x_i と x_j のどちらが望ましいかを決定することは比較的容易であることが挙げられる.

AHP では，x_i は x_j よりどれくらい望ましいかを**一対比較値**で表す. 正式には, 一対比較値 a_{ij} は x_i が x_j より a_{ij} 倍望ましいことを表す.

例えば，$a_{ij} = 3$ ならば，x_i は x_j より 3 倍望ましいことを表す. 例 7.1 で，$a_{12} = 3$ ならば，x_1 (北海道)は x_2 (沖縄)より 3 倍望ましいことを表す.

Saaty は, 一対比較値を以下の尺度で表すことを提案した.

Table: The fundamental scale of absolute numbers (Saaty, 2008)

Importance	Definition	Explanation
1	Equal	Two activities contribute equally to the objective
2	Weak or slight	-
3	Moderate	Experience and judgment slightly favor one activity over another
4	Moderate plus	-
5	Strong	Experience and judgment strongly favor one activity over another
6	Strong plus	-

Importance	Definition	Explanation
7	Very strong	An activity is favored very strongly over another activity
8	Very, very strong	-
9	Extreme	The evidence favoring one activity over another is of the highest possible order

7.4 一対比較行列

すべての代替案の組み合わせについて一対比較を行い, その結果を**一対比較行列** (pairwise comparison matrix) にまとめる. 一対比較行列は $\mathbf{A} = [a_{ij}]_{n\times n}$ と定義され, 以下のように表される。

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

理論的には,

$$a_{ij} \approx \frac{w_i}{w_j} \quad \forall i, j$$

が想定されている.

> ヒント
>
> 意思決定者が一対比較を行うとき，$\mathbf{w} = (w_1, w_2, \ldots, w_n)^T$ が知らないが, 合理的な意思決定者ならば，$a_{ij} = w_i/w_j$ を満たすと考えられる.

意思決定者が完全に合理的であるならば，$a_{ij} = w_i/w_j$ を満たす. このとき, 一対比較行列 $\mathbf{A}$ は以下のようになる.

$$\mathbf{A} = \begin{bmatrix} w_1/w_1 & w_1/w_2 & \cdots & w_1/w_n \\ w_2/w_1 & w_2/w_2 & \cdots & w_2/w_n \\ \vdots & \vdots & \ddots & \vdots \\ w_n/w_1 & w_n/w_2 & \cdots & w_n/w_n \end{bmatrix}$$

このとき，$\mathbf{A}$ は完全に**整合性** (consistency) を持つと言う．$\mathbf{A}$ は, 以下の性質が成り立つ.

- $a_{ii} = 1, \forall i$
- $a_{ij} = 1/a_{ji}, \forall i, j$
- $a_{ik} = a_{ij}a_{jk}, \forall i, j, k$

ヒント

$a_{ij} = w_i/w_j$ とするとき, これらの性質は, 以下のように導かれる.

$$a_{ii} = \frac{w_i}{w_i} = 1$$

$$a_{ij} = \frac{w_i}{w_j} = \frac{1}{\frac{w_j}{w_i}} = \frac{1}{a_{ji}}$$

$$a_{ik} = \frac{w_i}{w_k} = \frac{w_i}{w_j} \times \frac{w_j}{w_k} = a_{ij}a_{jk}$$

例 7.2. A さんがクッキー(x_1)はチョコレート(x_2)より 3 倍好きで, チョコレート(x_2)はアイスクリーム(x_3)より 2 倍好きならば, クッキーはアイスクリームより 6 倍好きである. すなわち，$a_{12} = 3$，$a_{23} = 2$ ならば，$a_{13} = 6$ である. また，A さんがクッキーはチョコレートより 3 倍好きであるならば, チョコレートはクッキーの 1/3 倍好きである. すなわち，$a_{12} = 3$ ならば, $a_{21} = 1/3$ である.

例 7.3. 旅行先の代替案 $X = \{$北海道, 沖縄, 九州$\}$ とし, 以下の一対比較を行ったとする.

- x_1 (北海道)は x_2 (沖縄)より 3 倍望ましい
- x_1 (北海道)は x_3 (九州)より 6 倍望ましい
- x_2 (沖縄)は x_3 (九州)より 2 倍望ましい

このとき, 一対比較行列 $\mathbf{A}$ は以下のようになる.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 6 \\ 1/3 & 1 & 2 \\ 1/6 & 1/2 & 1 \end{bmatrix}$$

7.5　重要度ベクトル

一対比較行列 $\mathbf{A}$ が与えられたとき, 重要度ベクトル $\mathbf{w}$ を計算する方法を説明する.

まず, 意思決定者が完全に合理的であるならば, $a_{ij} = w_i/w_j$ を満たすため, $\mathbf{A}$ は

$$\mathbf{A} = \begin{bmatrix} w_1/w_1 & w_1/w_2 & \cdots & w_1/w_n \\ w_2/w_1 & w_2/w_2 & \cdots & w_2/w_n \\ \vdots & \vdots & \ddots & \vdots \\ w_n/w_1 & w_n/w_2 & \cdots & w_n/w_n \end{bmatrix}$$

であるから, $\mathbf{A}$ の各列は比例している. このとき, 重要度ベクトル $\mathbf{w}$ は, $\mathbf{A}$ の任意の列を正規化することで求められる.

例 7.3 の一対比較行列 $\mathbf{A}$ を考える.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 6 \\ 1/3 & 1 & 2 \\ 1/6 & 1/2 & 1 \end{bmatrix}$$

このとき, $\mathbf{A}$ の第 3 列を正規化すると, 重要度ベクトル $\mathbf{w}$ は以下のようになる.

$$w_1 = \frac{6}{6+2+1} = \frac{6}{9} \approx 0.67$$
$$w_2 = \frac{2}{6+2+1} = \frac{2}{9} \approx 0.22$$
$$w_3 = \frac{1}{6+2+1} = \frac{1}{9} \approx 0.11$$

7.6 階層

AHP では, 複数の**評価基準**（criterion）を用いて代替案を評価することができる. 例えば, 旅行先の代替案を評価するとき, 費用, 距離, 環境などの評価基準を用いることができる.

> ヒント
>
> 意思決定者が直接代替案を一対比較することは難しいが, 評価基準を用いて代替案を評価することは比較的容易であると考えられる.

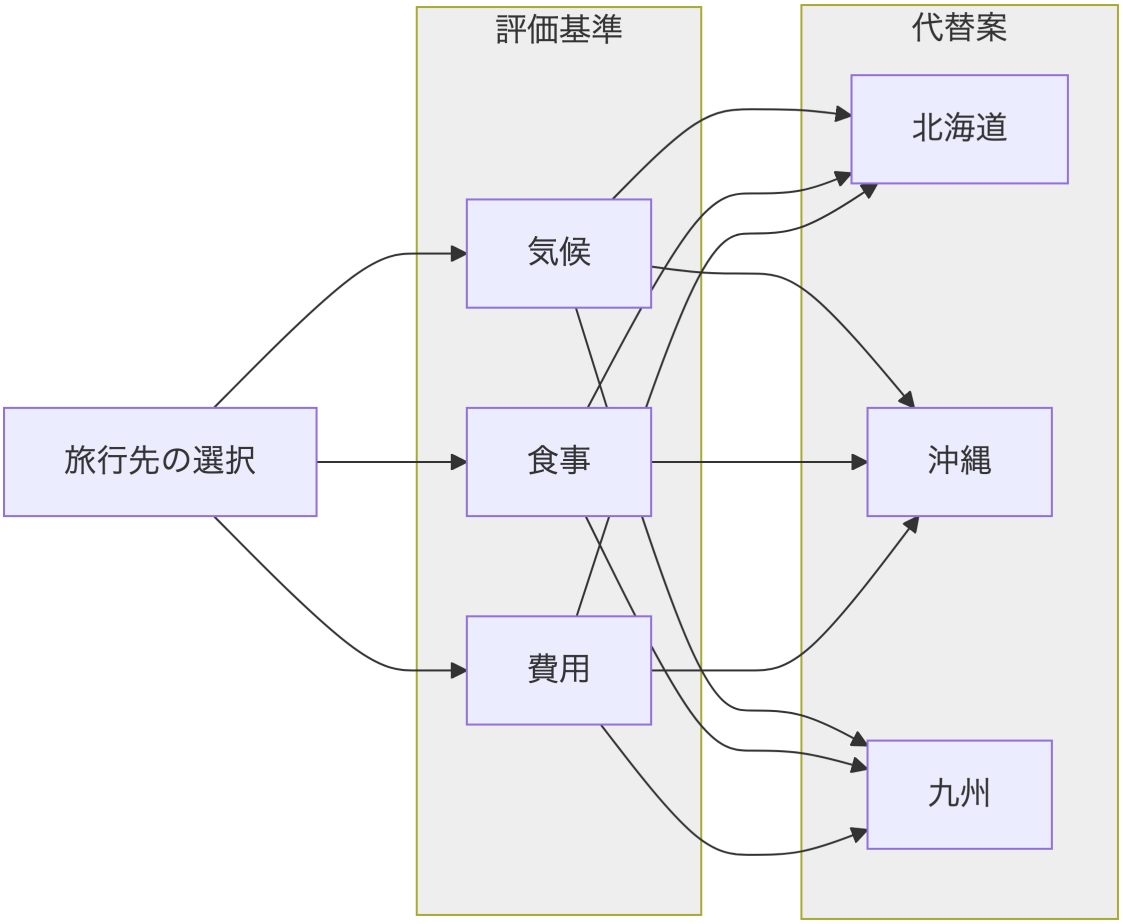
AHP では, 評価基準を用いて代替案を評価するために, **階層** (hierarchy) を用いる. 階層は, 以下の要素から構成される.

- 目的 (goal)
- 評価基準 (criteria)

- 代替案 (alternatives)

正式に, 評価基準の集合を $C = \{c_1, c_2, \dots, c_m\}$ とする.

例えば, 旅行先の代替案を $X = \{$北海道, 沖縄, 九州$\}$ とし, 評価基準 $C = \{$気候, 食事, 費用$\}$ とする. このとき, 以下のような階層を考える.



このような階層構造を用いることで, 評価基準に基づいて代替案を評価することができる. 各評価基準 $c_k \in C$ を用いて, 代替案の一対比較行列 $\mathbf{A}^{(k)}$ を構築する.

例えば, $A^{(1)}$, $A^{(2)}$, $A^{(3)}$ をそれぞれ評価基準 c_1, c_2, c_3 に対応する一対比較行列とする.

$$\mathbf{A}^{(1)} = \begin{bmatrix} 1 & 1/3 & 3 \\ 3 & 1 & 9 \\ 1/3 & 1/9 & 1 \end{bmatrix}, \quad \mathbf{A}^{(2)} = \begin{bmatrix} 1 & 1 & 4 \\ 1 & 1 & 4 \\ 1/4 & 1/4 & 1 \end{bmatrix}, \quad \mathbf{A}^{(3)} = \begin{bmatrix} 1 & 1/2 & 1/6 \\ 2 & 1 & 1/3 \\ 6 & 3 & 1 \end{bmatrix}$$

ヒント

以上の一対比較行列から見ると, この人は以下のような評価をしていることがわかる.

- 気候に関しては, 沖縄が最も望ましく, 次に北海道, 最後に九州である
- 食事に関しては, 北海道と沖縄が同じくらい望ましく, 九州が最も望ましくない
- 費用に関しては, 九州が最も望ましく, 次に沖縄, 最後に北海道である

重要度ベクトル $\mathbf{w}^{(k)}$ をそれぞれの一対比較行列 $\mathbf{A}^{(k)}$ から計算する.

$$\mathbf{w}^{(1)} = \begin{bmatrix} 3/13 \\ 9/13 \\ 1/13 \end{bmatrix}, \quad \mathbf{w}^{(2)} = \begin{bmatrix} 4/9 \\ 4/9 \\ 1/9 \end{bmatrix}, \quad \mathbf{w}^{(3)} = \begin{bmatrix} 1/9 \\ 2/9 \\ 6/9 \end{bmatrix}$$

各評価基準 $c_k \in C$ において, 代替案の重要度ベクトル $\mathbf{w}^{(k)}$ が計算できた. 次に, 評価基準はどれくらい重要かを評価する. 評価基準の一対比較行列 $\mathbf{B}$ を構築する.

例えば, 以下のような一対比較行列 $\mathbf{B}$ を考える.

- 評価基準 c_1 (気候)は c_2 (食事)より 1/4 倍重要である
- 評価基準 c_1 (気候)は c_3 (費用)より 2 倍重要である
- 評価基準 c_2 (食事)は c_3 (費用)より 8 倍重要である

ヒント

この人は, 食事を最も重要視しており, 次に気候, 最後に費用を重要視していることがわかる.

$$\mathbf{B} = \begin{bmatrix} 1 & 1/4 & 2 \\ 4 & 1 & 8 \\ 1/2 & 1/8 & 1 \end{bmatrix}$$

同様に, 重要度ベクトル $\mathbf{v}$ を一対比較行列 $\mathbf{B}$ から計算する.

$$\mathbf{v} = \begin{bmatrix} 2/11 \\ 8/11 \\ 1/11 \end{bmatrix}$$

最終的に, 代替案の重要度ベクトル $\mathbf{w}$ は以下のように計算される.

$$\mathbf{w} = \sum_{k=1}^{m} v_k \mathbf{w}^{(k)} = \frac{2}{11} \begin{bmatrix} 3/13 \\ 9/13 \\ 1/13 \end{bmatrix} + \frac{8}{11} \begin{bmatrix} 4/9 \\ 4/9 \\ 1/9 \end{bmatrix} + \frac{1}{11} \begin{bmatrix} 1/9 \\ 2/9 \\ 6/9 \end{bmatrix} \approx \begin{bmatrix} 0.38 \\ 0.47 \\ 0.16 \end{bmatrix}$$

```
import numpy as np

w1 = np.array([3/13, 9/13, 1/13])
w2 = np.array([4/9, 4/9, 1/9])
w3 = np.array([1/9, 2/9, 6/9])
v = np.array([2/11, 8/11, 1/11])
w = v[0] * w1 + v[1] * w2 + v[2] * w3
w
```

```
array([0.37529138, 0.46930847, 0.15540016])
```

このとき, 重要度ベクトル $\mathbf{w}$ から, 望ましい代替案は x_2 (沖縄)であることがわかる.

> **階層の一般化**
>
> 今回は, 目標, 評価基準, 代替案の 3 層の階層を考えたが, 評価基準の下にさらに評価基準を追加するなど, より複雑な階層を考えることもできる.

7.7　重要度ベクトルの推定

しかし, 人間の評価は必ずしも合理的でないため, 一般には $a_{ij} = w_i/w_j$ を満たさない. このとき, 他の方法で一対比較行列 $\mathbf{A}$ から重要度ベクトル $\mathbf{w}$ を計算する必要がある. 計算する方法として, 主に**幾何平均法** (geometric mean method) と**固有ベクトル法** (eigenvector method) の 2 つがある.

ヒント

幾何平均法と固有ベクトル法のどちらが優れているかについては議論がある．AHP の創始者である Saaty は固有ベクトル法が優れていると主張しているが, 幾何平均法が優れているとする出張する研究もある．他にも normalized columns method という方法もあるが, 理論的根拠がないと言われている．

7.7.1 幾何平均法

Crawford と Williams (1985) は, 幾何平均法を提案した. 幾何平均法では, 重要度ベクトル $\mathbf{w}$ の各要素 w_k を以下のように計算する．

$$w_k = \frac{\left(\prod_{j=1}^{n} a_{kj}\right)^{1/n}}{\sum_{i=1}^{n}\left(\prod_{j=1}^{n} a_{ij}\right)^{1/n}} \quad \forall k$$

ヒント

$\mathbf{A}$ の各行の幾何平均を計算し, 正規化することで, 重要度ベクトル $\mathbf{w}$ を求めている．

例 7.3 の一対比較行列 $\mathbf{A}$ を考える．

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 6 \\ 1/3 & 1 & 2 \\ 1/6 & 1/2 & 1 \end{bmatrix}$$

このとき, 重要度ベクトル $\mathbf{w}$ は以下のようになる. これは, 理論から計算した重要度ベクトルと一致する．

$$w_1 = \frac{(1 \times 3 \times 6)^{1/3}}{(1 \times 3 \times 6)^{1/3} + (1/3 \times 1 \times 2)^{1/3} + (1/6 \times 1/2 \times 1)^{1/3}} \approx 0.67$$

$$w_2 = \frac{(1/3 \times 1 \times 2)^{1/3}}{(1 \times 3 \times 6)^{1/3} + (1/3 \times 1 \times 2)^{1/3} + (1/6 \times 1/2 \times 1)^{1/3}} \approx 0.22$$

$$w_3 = \frac{(1/6 \times 1/2 \times 1)^{1/3}}{(1 \times 3 \times 6)^{1/3} + (1/3 \times 1 \times 2)^{1/3} + (1/6 \times 1/2 \times 1)^{1/3}} \approx 0.11$$

一般に, 一対比較行列 $\mathbf{A}$ が整合性を持つならば, 幾何平均法で計算した重要度ベクトル $\mathbf{w}$ は理論から計算した重要度ベクトルと一致する. 興味があれば, 証明してみよう.

7.7.2　固有ベクトル法*

> 💡 ヒント
>
> 固有ベクトル法は直感的に理解することが難しいため, 講義では扱わない.

一対比較行列 $\mathbf{A}$ が整合性を持つとき,

$$\mathbf{Aw} = \begin{bmatrix} w_1/w_1 & w_1/w_2 & \cdots & w_1/w_n \\ w_2/w_1 & w_2/w_2 & \cdots & w_2/w_n \\ \vdots & \vdots & \ddots & \vdots \\ w_n/w_1 & w_n/w_2 & \cdots & w_n/w_n \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} nw_1 \\ nw_2 \\ \vdots \\ nw_n \end{bmatrix} = n\mathbf{w}$$

が成り立つ. 式 $\mathbf{Aw} = n\mathbf{w}$ より，$\mathbf{w}$ は $\mathbf{A}$ の固有ベクトル，n は $\mathbf{A}$ の固有値である.

Saaty は, 最大固有値 λ_{max} に対応する固有ベクトルを重要度ベクトルとして用いることを提案した. すなわち, 重要度ベクトル $\mathbf{w}$ を求めるには, 以下の式を解く.

$$\mathbf{Aw} = \lambda_{max}\mathbf{w}$$

例 7.3 の一対比較行列が与えられたとする.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 6 \\ 1/3 & 1 & 2 \\ 1/6 & 1/2 & 1 \end{bmatrix}$$

このとき，$\mathbf{A}$ の固有値は，$\det(\mathbf{A} - \lambda\mathbf{I}) = 0$ を解くことで求められる.

$$\begin{aligned}\det(\mathbf{A}-\lambda\mathbf{I}) &= \det\begin{bmatrix}1-\lambda & 3 & 6\\ 1/3 & 1-\lambda & 2\\ 1/6 & 1/2 & 1-\lambda\end{bmatrix}\\ &= -\lambda^3+3\lambda^2\\ &= -\lambda^2(\lambda-3)\end{aligned}$$

$\lambda^2(\lambda-3)=0$ より，$\lambda=0,3$ が得られる. この例において，$\mathbf{A}$ の最大固有値は $\lambda_{max}=3$ であり, それ以外の固有値は $\lambda=0$ である. また，$\mathbf{A}$ は 3×3 行列であるため，$\lambda_{max}=n$ が成り立つ. これは特別な例ではなく, 一般に 命題 7.1 が成り立つ.

命題 7.1. 整合性のある $n\times n$ 行列 $\mathbf{A}$ の最大固有値は $\lambda_{max}=n$ であり, それ以外の固有値は $\lambda=0$ である.

行列 $\mathbf{A}$ は整合性を持たないとき，$\lambda_{max}>n$ が成り立つ. 命題 7.2 は λ_{max} の性質を述べている.

命題 7.2 (Saaty)**.** $\mathbf{A}$ を一対行列とする. $\mathbf{A}$ が整合性を持つならば, $\lambda_{max}=n$ である. $\mathbf{A}$ が整合性を持たないならば，$\lambda_{max}>n$ である.

7.8 整合性*

計算された重要度ベクトル $\mathbf{w}$ が一対比較行列 $\mathbf{A}$ とどれくらい整合しているかを評価する指標として, 様々な指標が提案されている..

7.8.1 CI

Saaty は, 一対比較行列 $\mathbf{A}$ の整合性を評価する指標として**整合度** (consistency index, CI) を提案した.

$$CI=\frac{\lambda_{max}-n}{n-1}$$

7.8.2 GCI

Aguarón が**平均幾何整合度** (geometric consistency index, GCI) を提案した.

一対比較値 a_{ij} と重要度ベクトル $\mathbf{w}$ から, 以下のように e_{ij} を定義する.

$$e_{ij} = a_{ij}\frac{w_j}{w_i}$$

$a_{ij} = w_i/w_j$ を満たすならば，$e_{ij} = 1$ である．

GCI は以下のように定義される．

$$GCI = \frac{2}{(n-1)(n-2)}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\left(\ln e_{ij}\right)^2$$

7.9　文献案内

Saaty (1977) は初めて AHP を提案している. この論文では, 固有ベクトル法と整合性の指標 CI を提案している．

Crawford と Williams (1985) は幾何平均法を提案している．

Brunelli (2015) は AHP を丁寧に解説した書籍である. 深く理解したい人にお勧めする．

7.10　用語

English	Japanese
multi-criteria decision making (MCDM)	多基準意思決定
Analytic Hierarchy Process (AHP)	階層分析法
Consistency index	整合度
pairwise comparison matrix	一対比較行列
alternative	代替案
criterion	評価基準
priority vector	重要度ベクトル

7.11　練習問題

練習 7.1. ある会社が新しい工場の建設場所を選ぶために，3 つの都市を代替案として検討している．

- 都市 A は都市 B よりも 2 倍望ましい
- 都市 A は都市 C よりも 4 倍望ましい
- 都市 B は都市 C よりも 2 倍望ましい

このとき, 一対比較行列を構築し, 重要度ベクトルを計算せよ.

解答 7.1. 一対比較行列 $\mathbf{A}$ は以下のようになる.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 1/2 & 1 & 2 \\ 1/4 & 1/2 & 1 \end{bmatrix}$$

$\mathbf{A}$ は整合性を持つため, $\mathbf{A}$ の任意の列を正規化することで, 重要度ベクトル $\mathbf{w}$ を求めることができる.

$$w = \begin{bmatrix} 4/7 \\ 2/7 \\ 1/7 \end{bmatrix}$$

練習 7.2. ある学生が大学を選ぶために, 3 つの大学を代替案として検討している.

- 大学 A は大学 B よりも 3 倍望ましい
- 大学 A は大学 C よりも 5 倍望ましい
- 大学 B は大学 C よりも 2 倍望ましい

このとき, 一対比較行列を構築し, 重要度ベクトルを計算せよ.

解答 7.2. 一対比較行列 $\mathbf{A}$ は以下のようになる.

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 5 \\ 1/3 & 1 & 2 \\ 1/5 & 1/2 & 1 \end{bmatrix}$$

$\mathbf{A}$ は整合性を持たないため, 幾何平均法で重要度ベクトル $\mathbf{w}$ を求める.

$$w_1 = \frac{(1 \times 3 \times 5)^{1/3}}{(1 \times 3 \times 5)^{1/3} + (1/3 \times 1 \times 2)^{1/3} + (1/5 \times 1/2 \times 1)^{1/3}} \approx 0.65$$

$$w_2 = \frac{(1/3 \times 1 \times 2)^{1/3}}{(1 \times 3 \times 5)^{1/3} + (1/3 \times 1 \times 2)^{1/3} + (1/5 \times 1/2 \times 1)^{1/3}} \approx 0.23$$

$$w_3 = \frac{(1/5 \times 1/2 \times 1)^{1/3}}{(1 \times 3 \times 5)^{1/3} + (1/3 \times 1 \times 2)^{1/3} + (1/5 \times 1/2 \times 1)^{1/3}} \approx 0.12$$

```
import numpy as np

A = np.array([[1, 3, 5], [1/3, 1, 2], [1/5, 1/2, 1]])
w1 = (np.prod(A[0, :]))**(1/3)
w2 = (np.prod(A[1, :]))**(1/3)
w3 = (np.prod(A[2, :]))**(1/3)
w = np.array([w1, w2, w3])
w = w / np.sum(w)
w
```

```
array([0.64832901, 0.22965079, 0.12202019])
```

練習 7.3. ある学生が就職先を選ぶために，3 つの企業を代替案として検討している. 評価基準として, 給与, 勤務地, 企業の評判を考える.

- 給与に関しては, 企業 A は企業 B よりも 2 倍望ましく, 企業 A は企業 C よりも 3 倍望ましく, 企業 B は企業 C よりも 2 倍望ましい.
- 勤務地に関しては, 企業 A は企業 B と同じくらい望ましく, 企業 A は企業 C よりも 4 倍望ましく, 企業 B は企業 C よりも 4 倍望ましい.
- 企業の評判に関しては, 企業 B は企業 A の 2 倍望ましく, 企業 C は企業 B の 4 倍望ましく, 企業 C は企業 A の 8 倍望ましい.

また, この学生は, 評価基準として, 給与は勤務地の 2 倍, 勤務地は企業の評判の 2 倍重要, 給与は企業の評判の 4 倍重要と考えている.

このとき，AHP を用いて重要度ベクトルを計算し, 最も望ましい企業を選べ.

解答 7.3. 給与に関する一対比較行列 $\mathbf{A}^{(1)}$ は以下のようになる.

$$\mathbf{A}^{(1)} = \begin{bmatrix} 1 & 1 & 4 \\ 1 & 1 & 4 \\ 1/4 & 1/4 & 1 \end{bmatrix}$$

勤務地に関する一対比較行列 $\mathbf{A}^{(2)}$ は以下のようになる.

$$\mathbf{A}^{(2)} = \begin{bmatrix} 1 & 2 & 6 \\ 1/2 & 1 & 3 \\ 1/6 & 1/3 & 1 \end{bmatrix}$$

企業の評判に関する一対比較行列 $\mathbf{A}^{(3)}$ は以下のようになる.

$$\mathbf{A}^{(3)} = \begin{bmatrix} 1 & 1/2 & 1/8 \\ 2 & 1 & 1/4 \\ 8 & 4 & 1 \end{bmatrix}$$

重要度ベクトル $\mathbf{w}^{(k)}$ をそれぞれの一対比較行列 $\mathbf{A}^{(k)}$ から計算する.

$$\mathbf{w}^{(1)} = \begin{bmatrix} 4/9 \\ 4/9 \\ 1/9 \end{bmatrix}, \quad \mathbf{w}^{(2)} = \begin{bmatrix} 6/10 \\ 3/10 \\ 1/10 \end{bmatrix}, \quad \mathbf{w}^{(3)} = \begin{bmatrix} 1/11 \\ 2/11 \\ 8/11 \end{bmatrix}$$

評価基準の一対比較行列 $\mathbf{B}$ は以下のようになる.

$$\mathbf{B} = \begin{bmatrix} 1 & 2 & 4 \\ 1/2 & 1 & 2 \\ 1/4 & 1/2 & 1 \end{bmatrix}$$

重要度ベクトル $\mathbf{v}$ を一対比較行列 $\mathbf{B}$ から計算する.

$$\mathbf{v} = \begin{bmatrix} 4/7 \\ 2/7 \\ 1/7 \end{bmatrix}$$

最終的に, 代替案の重要度ベクトル $\mathbf{w}$ は以下のように計算される.

$$\mathbf{w} = \sum_{k=1}^{m} v_k \mathbf{w}^{(k)} = \frac{4}{7} \begin{bmatrix} 4/9 \\ 4/9 \\ 1/9 \end{bmatrix} + \frac{2}{7} \begin{bmatrix} 6/10 \\ 3/10 \\ 1/10 \end{bmatrix} + \frac{1}{7} \begin{bmatrix} 1/11 \\ 2/11 \\ 8/11 \end{bmatrix} \approx \begin{bmatrix} 0.44 \\ 0.37 \\ 0.20 \end{bmatrix}$$

```
import numpy as np

w1 = np.array([4/9, 4/9, 1/9])
w2 = np.array([6/10, 3/10, 1/10])
w3 = np.array([1/11, 2/11, 8/11])
v = np.array([4/7, 2/7, 1/7])
w = v[0] * w1 + v[1] * w2 + v[2] * w3
w
```

```
array([0.43838384, 0.36565657, 0.1959596 ])
```

このとき, 重要度ベクトル $\mathbf{w}$ から, 望ましい代替案は企業 A であることがわかる.

第III部

ネットワーク最適化

第8章　グラフ理論

グラフ（graph）は、**点**（vertex）の集合 V と辺（edge）の集合 E から構成され、$G = (V, E)$ で表される。

グラフは、**有向グラフ**（directed graph）と**無向グラフ**（undirected graph）に分けられる。

8.1　無向グラフ

無向グラフは、辺の方向を持たないグラフである。辺は、2 つの点の集合として表される。

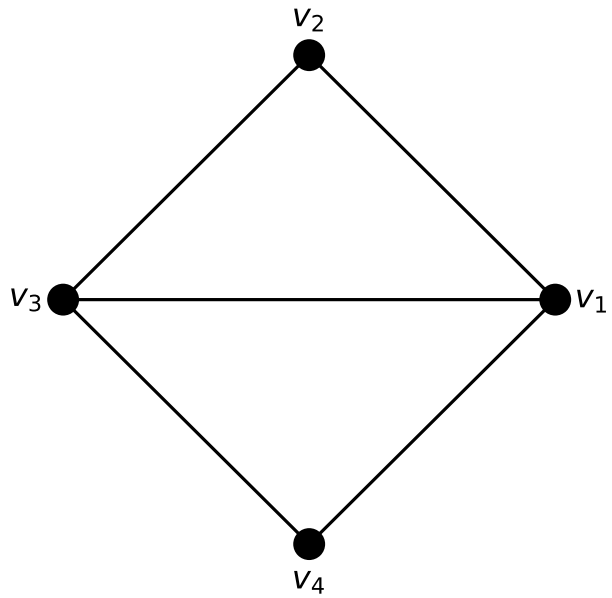
例 8.1 (無向グラフの例)**.** グラフ

$$G = (v_1, v_2, v_3, v_4, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_1\}, \{v_1, v_3\}\})$$

は、点の集合 $V = \{v_1, v_2, v_3, v_4\}$ と辺の集合 $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_1\}, \{v_1, v_3\}\}$ からなる。

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# グラフの作成(頂点を番号で定義)
G = nx.Graph()
G.add_edges_from([(1, 2), (2, 3), (3, 4), (4, 1), (1, 3)])

# 頂点ラベルを LaTeX 形式に変換
labels = {i: rf"$v_{i}$" for i in G.nodes()}

# レイアウト(円形)
pos = nx.circular_layout(G)
```

```
# ラベルの位置を少し外側へ移動
label_pos = {}
for k, (x, y) in pos.items():
    r = np.sqrt(x**2 + y**2)  # 原点からの距離
    scale = 1.15  # 外側に押し出す係数(調整可)
    label_pos[k] = (x / r * scale, y / r * scale)

plt.figure(figsize=(4, 4))
# ノード描画
nx.draw_networkx_nodes(G, pos, node_color="black", node_size=120, edgecolors="black
# エッジ描画
nx.draw_networkx_edges(G, pos, edge_color="black", width=1.2)
# 外側にラベル描画
nx.draw_networkx_labels(G, label_pos, labels, font_size=12, font_weight="regular")
plt.axis("off")
plt.show()
```



グラフ $G = (V, E)$ において、$e = \{v, u\} \in E$ を満たすとき、点 v と u は**隣接**（adjacent）しているといい、e は v と u を**接続**（incident）しているという。

点 v に接続している辺の数を、v の**次数**（degree）という。例 8.1 の場合、v_1 の次数は 3、v_2 の次数は 2 である。

8.2 有向グラフ

有向グラフは、辺に方向があるグラフである。辺は、2 つの点の順序対として表される。

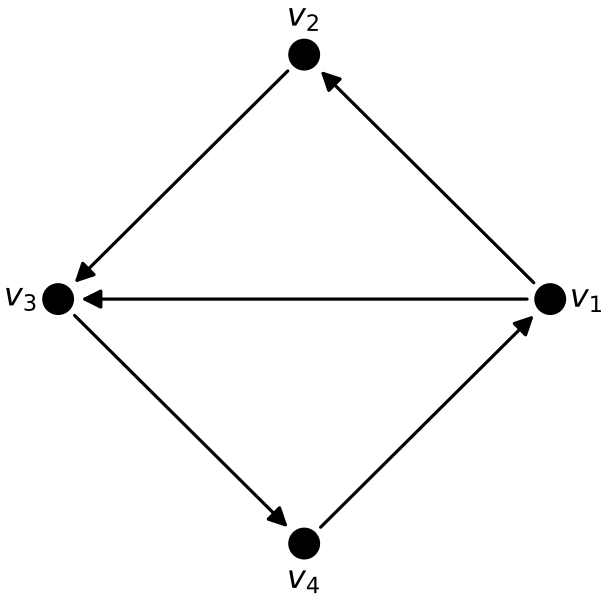
例 8.2 (有向グラフの例)**.** グラフ

$$G = (v_1, v_2, v_3, v_4, \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1), (v_1, v_3)\})$$

は、点の集合 $V = \{v_1, v_2, v_3, v_4\}$ と辺の集合 $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1), (v_1, v_3)\}$ からなる。

```python
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# グラフの作成(頂点を番号で定義)
G = nx.DiGraph()
G.add_edges_from([(1, 2), (2, 3), (3, 4), (4, 1), (1, 3)])

# 頂点ラベルを LaTeX 形式に変換
labels = {i: rf"$v_{i}$" for i in G.nodes()}

# レイアウト(円形)
pos = nx.circular_layout(G)

# ラベルの位置を少し外側へ移動
label_pos = {}
for k, (x, y) in pos.items():
    r = np.sqrt(x**2 + y**2)  # 原点からの距離
    scale = 1.15  # 外側に押し出す係数(調整可)
    label_pos[k] = (x / r * scale, y / r * scale)

plt.figure(figsize=(4, 4))

# ノード描画
nx.draw_networkx_nodes(G, pos, node_color="black", node_size=120, edgecolors="black")

# エッジ描画
nx.draw_networkx_edges(G, pos, edge_color="black", width=1.2, arrowsize=15)
```

```
# 外側にラベル描画
nx.draw_networkx_labels(G, label_pos, labels, font_size=12, font_weight="regular")

plt.axis("off")
plt.show()
```



グラフ $G = (V, E)$ において、点 v から出る辺の数を**出次数**（outdegree）といい、点 v に入る辺の数を**入次数**（indegree）という。例 8.2 の場合、v_1 の出次数は 2、入次数は 1 である。

8.3　パス

$G = (V, E)$ において、**パス**（path）P とは、点の組

$$P = (v_{i_1}, v_{i_2}, \dots, v_{i_k})$$

であって、各 $j = 1, 2, \dots, k-1$ に対して $(v_{i_j}, v_{i_{j+1}}) \in E$ を満たすものをいう。パスの**長さ**（length）は、パスに含まれる辺の数である。

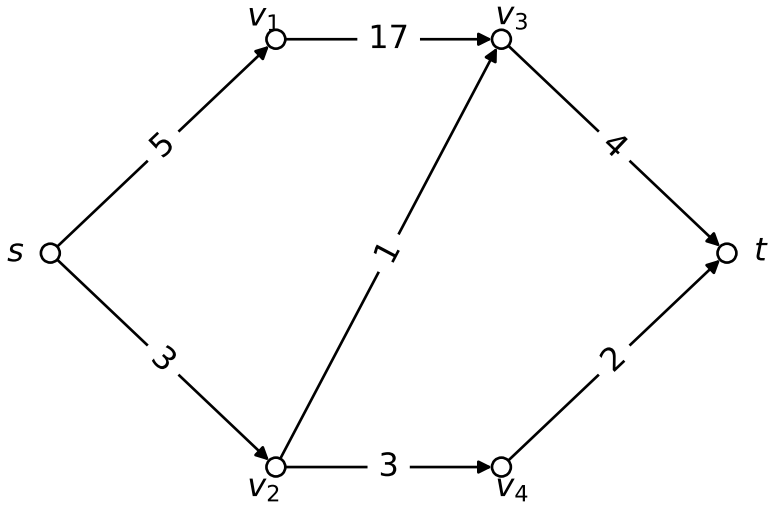
第9章　最短路問題

有向グラフ $G = (V, E)$ を考える。各辺 $(v, u) \in E$ に非負の重み $w(v, u)$ が与えられているとする。このとき、点 $s \in V$ から点 $t \in V$ への**最短路**（shortest path）とは、s から t へのパスのうち、重みの和が最小となるパスである。最短路問題とは、s から t への最短路を求める問題である。

例 9.1. 次の図は、最短路問題の例を示している。

```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np

# グラフの作成(頂点を番号で定義)
G = nx.DiGraph()
edges = [
    ("s", "v1", 5),
    ("s", "v2", 3),
    ("v1", "v3", 17),
    ("v2", "v3", 1),
    ("v2", "v4", 3),
    ("v3", "t", 4),
    ("v4", "t", 2),
]
G.add_weighted_edges_from(edges)

labels = {
    node: (rf"$v_{node[1:]}$" if node.startswith("v") else rf"${node}$")
    for node in G.nodes()
}

for layer, nodes in enumerate(nx.topological_generations(G)):
    for node in nodes:
        G.nodes[node]["layer"] = layer
```

```
pos = nx.multipartite_layout(G, subset_key="layer")

label_pos = {}
for k, (x, y) in pos.items():
    r = np.sqrt(x**2 + y**2) if np.sqrt(x**2 + y**2) > 0 else 1
    scale = 1.1
    label_pos[k] = (x * scale, y * scale)

# Plotting
nx.draw_networkx(
    G,
    pos=pos,
    with_labels=False,
    node_color="white",
    node_size=50,
    edgecolors="black",
)
nx.draw_networkx_labels(G, label_pos, labels, font_size=12)
nx.draw_networkx_edge_labels(
    G,
    pos,
    edge_labels={(u, v): d["weight"] for u, v, d in G.edges(data=True)},
    font_color="black",
    font_size=12,
)
plt.axis("off")
plt.show()
```


v_1
17
v_3
5
4
s
1
t
3
2
3
v_2
v_4

第10章　PERT/CPM

プロジェクトは、多くの作業からなる。プロジェクトを完遂できるように、作業のスケジュールを作成する必要がある。その方法として、**PERT**（Program Evaluation and Review Technique）と **CPM**（Critical Path Method）がある。現在では、PERT と CPM が統合され、**PERT/CPM** として知られている (Camm ほか 2022; Hillier と Lieberman 2025)。

> **ノート**
>
> PERT/CPM は「基本情報技術者試験」や「応用情報技術者試験」などで頻出のトピックのようです. 他にも, 線形計画法, 在庫問題, ゲーム理論, データ分析, データベース, データ構造とアルゴリズムなど, 様々な経営工学関係のトピックが出題されます. 経営工学を学ぶ学生は, 受けてみると良いでしょう.

10.1　プロジェクト

ここで、ある作業の開始前に完了しなければならない作業を**先行作業**（Immediate Predecessor）と呼ぶ。また、ある作業の完了後に開始できる作業を**後続作業**（Immediate Successor）と呼ぶ。

まとめると、プロジェクトは、次の要素で構成される。

- 作業：プロジェクトを構成する仕事。活動、アクティビティとも呼ばれる。
- 先行関係：それぞれの作業の先行作業を定義する関係。
- 作業時間：各作業に必要な時間。

例 10.1. 学生の田中さんと佐藤さんが協力し、ある授業のレポートを作成することになった。このレポートを作成するためには、下の表に示すように、いくつかの作業を行う必要がある。

作業	作業内容	先行作業	時間(日)
A	課題の理解	-	2
B	データ収集	A	3
C	データ分析	B	4
D	文献調査	A	2
E	レポート作成	C, D	5

10.2 プロジェクト・ネットワーク

プロジェクトをネットワークで表現したものを**プロジェクト・ネットワーク**(Project Network)と呼ぶ。プロジェクト・ネットワークには、**AOA**(Activity on Arrow)や **AON**(Activity on Node)という 2 種類の表現方法がある。

AOA では、作業を辺で表現し、先行関係を点で表現する。AON では、作業を点で表現し、先行関係を辺で表現する。AON のほうが理解と作成が容易で、実務でも AON がよく使われる (Camm ほか 2022; Hillier と Lieberman 2025; Eiselt と Sandblom 2022)。これは以降、AON に基づいて説明する。

プロジェクト・ネットワークは $G = (V, E)$ という有向グラフで表される。ここで、V は始点 s、終点 t と各作業を表す点の集合であり、E は先行関係を表す辺の集合である。辺 $(v, u) \in E$ では、v が u の先行作業であることを意味する。

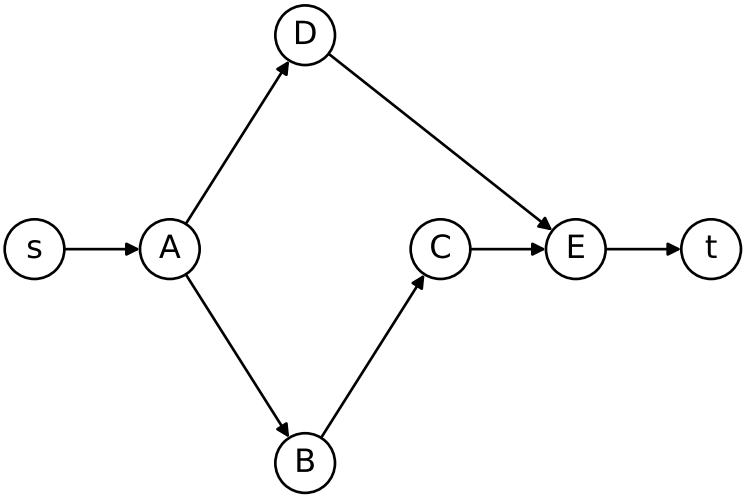
辺 $(s, v) \in E$ は、作業 v が先行作業を持たないことを意味し、辺 $(v, t) \in E$ は、作業 v が後続作業を持たないことを意味する。作業 v の先行作業の集合を $\mathcal{P}(v)$ と表す。作業 v の後続作業の集合を $\mathcal{S}(v)$ と表す。

作業点 $v \in V$ には、作業時間 $\tau(v)$ が与えられている。始点 s と終点 t の作業時間は $\tau(s) = \tau(t) = 0$ とする。

例 10.1 のプロジェクト・ネットワークは次の図のようになる。

```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np

# グラフの作成(頂点を番号で定義)
G = nx.DiGraph()
edges = [
    ("s", "A"),
    ("A", "B"),
```

```
    ("A", "D"),
    ("B", "C"),
    ("C", "E"),
    ("D", "E"),
    ("E", "t"),
]
G.add_edges_from(edges)

for layer, nodes in enumerate(nx.topological_generations(G)):
    for node in nodes:
        G.nodes[node]["layer"] = layer

pos = nx.multipartite_layout(G, subset_key="layer")

label_pos = {}
for k, (x, y) in pos.items():
    r = np.sqrt(x**2 + y**2) if np.sqrt(x**2 + y**2) > 0 else 1
    scale = 1.12
    label_pos[k] = (x * scale, y * scale)

# Plotting
nx.draw_networkx(
    G,
    pos=pos,
    with_labels=True,
    node_color="white",
    node_size=500,
    edgecolors="black",
)
plt.axis("off")
plt.show()
```

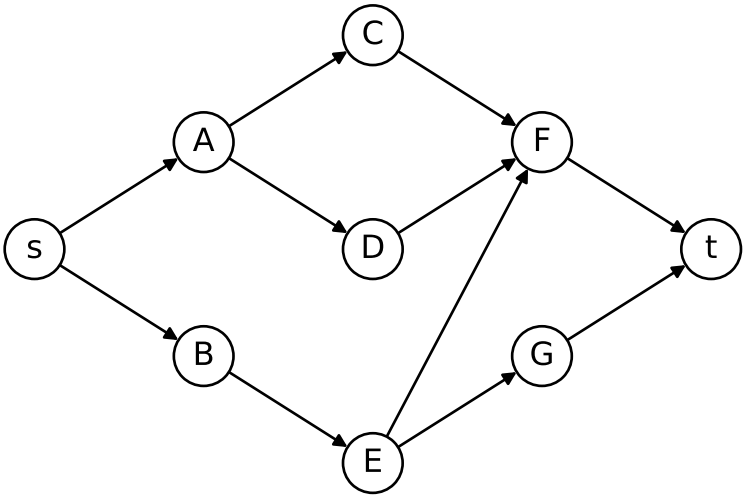


例 10.2. 以下の作業リストに基づいて、プロジェクト・ネットワークを作成せよ。

作業	先行作業	時間(日)
A	-	2
B	-	3
C	A	4
D	A	2
E	B	5
F	C, D, E	9
G	E	2

```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np

# グラフの作成(頂点を番号で定義)
G = nx.DiGraph()
edges = [
    ("s", "A"),
    ("s", "B"),
    ("A", "C"),
    ("A", "D"),
    ("C", "F"),
```

```
    ("D", "F"),
    ("B", "E"),
    ("E", "G"),
    ("E", "F"),
    ("G", "t"),
    ("F", "t"),
]
G.add_edges_from(edges)

for layer, nodes in enumerate(nx.topological_generations(G)):
    for node in nodes:
        G.nodes[node]["layer"] = layer

pos = nx.multipartite_layout(G, subset_key="layer")

label_pos = {}
for k, (x, y) in pos.items():
    r = np.sqrt(x**2 + y**2) if np.sqrt(x**2 + y**2) > 0 else 1
    scale = 1.12
    label_pos[k] = (x * scale, y * scale)

# Plotting
nx.draw_networkx(
    G,
    pos=pos,
    with_labels=True,
    node_color="white",
    node_size=500,
    edgecolors="black",
)
plt.axis("off")
plt.show()
```



10.3　クリティカルパス

プロジェクト・ネットワークにおいて、始点 s から終点 t までの経路を**パス**と呼ぶ。パスの長さは、そのパス上の作業時間の合計である。パス $P = (s, v_{i_1}, v_{i_2}, \ldots, v_{i_k}, t)$ の長さは、

$$\sum_{v \in P} \tau(v) = \tau(v_{i_1}) + \tau(v_{i_2}) + \cdots + \tau(v_{i_k})$$

である。

最も長いパスを**クリティカルパス**（Critical Path）と呼ぶ。クリティカルパスに含まれる作業を**クリティカル作業**（Critical Activity）と呼ぶ。クリティカル作業が遅延すると、プロジェクト全体の完了が遅延する。

クリティカルパスを求める問題は、最長路問題に帰着できるが、ここで述べる方法は、プロジェクトにおける様々な有用な情報も提供する。

10.3.1　最早開始時刻と最早終了時刻

作業点 $v \in V$ の最も早く開始できる時間を**最早開始時刻**（Earliest Start Time）と呼び、$ES(v)$ と表す。v の最も早く終了できる時間を**最早終了時刻**（Earliest Finish Time）と呼び、$EF(v)$ と表す。始点 s の最早開始時刻は $ES(s) = 0$ とする。

v の最早終了時刻は、最早開始時刻に作業時間を加えたものである。

$$EF(v) = ES(v) + \tau(v)$$

u の最早開始時刻は、u の先行作業 $v \in \mathcal{P}(u)$ の中で最早終了時刻 $EF(v)$ が最大のものである。

$$ES(u) = \max_{v \in \mathcal{P}(u)} EF(v)$$

で与えられる。

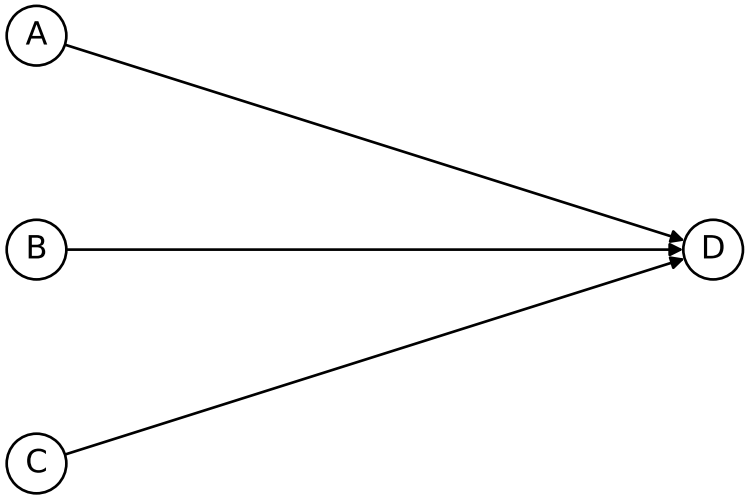
例 10.3. 作業点 D の先行作業を A, B, C とする. そのとき, $\mathcal{P}(D) = \{A, B, C\}$ である. $EF(A) = 2$, $EF(B) = 3$, $EF(C) = 6$ とすると, D の最早開始時刻は,

$$ES(D) = \max\{EF(A), EF(B), EF(C)\} = \max\{2, 3, 6\} = 6$$

である.

```
import matplotlib.pyplot as plt
import networkx as nx

# グラフの作成(頂点を番号で定義)
G = nx.DiGraph()
edges = [
    ("A", "D"),
    ("B", "D"),
    ("C", "D"),
]
G.add_edges_from(edges)

# A, B, C を縦に並べる

pos = {
    "A": (0, 1),
    "B": (0, 0),
    "C": (0, -1),
    "D": (2, 0),
}
# Plotting
nx.draw_networkx(
```

```
    G,
    pos=pos,
    with_labels=True,
    node_color="white",
    node_size=500,
    edgecolors="black",
)
plt.axis("off")
plt.show()
```



最早開始時刻と最早終了時刻を求めるアルゴリズムは下の通りである。ここで、点を**トポロジカルオーダー** (topological order) で処理するとは、v のすべての先行作業 $u \in \mathcal{P}(v)$ が v より前に処理されるように点を順序付けることである。

Algorithm 10.1. ***Forward Pass Algorithm***

1. $ES(s) \leftarrow 0$, $EF(s) \leftarrow 0$.
2. ***For each*** $v \in V \setminus \{s\}$ *in topological order* ***do***
    1. $ES(v) \leftarrow \max_{u \in \mathcal{P}(v)} EF(u)$.
    2. $EF(v) \leftarrow ES(v) + \tau(v)$.

例 10.2 のプロジェクト・ネットワークに対して、最早開始時刻と最早終了時刻を求めると、次の表のようになる。

v	$\mathcal{P}(v)$	$\tau(v)$	$ES(v)$	$EF(v)$
s	-	0	0	0
A	s	2	0	2
B	s	3	0	3
C	A	4	2	6
D	A	2	2	4
E	B	5	3	8
F	C, D, E	9	8	17
G	E	2	8	10
t	F, G	0	17	17

作業 F を開始するためには、作業 C、D、E のすべてが完了している必要がある。そのため、作業 F の開始時間は、作業 C、D、E の中で最も遅い完了時間に依存する。作業 C、D、E の完了時間はそれぞれ 6 日、4 日、8 日であるため、作業 F の最も早い開始時間は 8 日となる。

10.3.2 最遅開始時刻と最遅終了時刻

作業点 v の遅くとも始めないといけない時間を**最遅開始時刻**（Latest Start Time）と呼び、$LS(v)$ と表す。v の遅くとも終わらせないといけない時間を**最遅終了時刻**（Latest Finish Time）と呼び、$LF(v)$ と表す。

終点 t の最遅終了時刻は $LF(t) = EF(t)$ とする。

v の最遅開始時刻は、最遅終了時刻から作業時間を引いたものである。

$$LS(v) = LF(v) - \tau(v)$$

u の最遅終了時刻は、u の後続作業 $v \in \mathcal{S}(u)$ の中で $LS(v)$ が最小のものである。

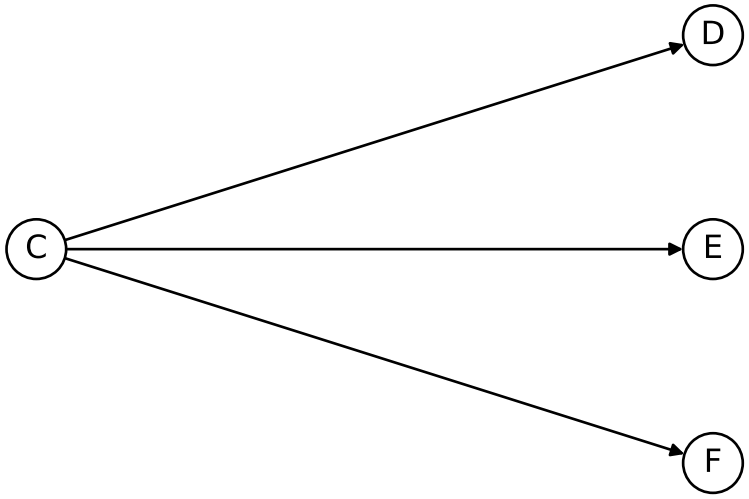
$$LF(u) = \min_{v \in \mathcal{S}(u)} LS(v)$$

例 10.4. 作業点 C の後続作業を D, E, F とする. そのとき, $\mathcal{S}(C) = \{D, E, F\}$ である. $LS(D) = 6$, $LS(E) = 3$, $LS(F) = 8$ とすると, C の最遅終了時刻は,

$$LF(C) = \min\{LS(D), LS(E), LS(F)\} = \min\{6, 3, 8\} = 3$$

である.

```
import matplotlib.pyplot as plt

import networkx as nx

# グラフの作成(頂点を番号で定義)
G = nx.DiGraph()
edges = [
    ("C", "D"),
    ("C", "E"),
    ("C", "F"),
]
G.add_edges_from(edges)

pos = {
    "C": (0, 0),
    "D": (2, 1),
    "E": (2, 0),
    "F": (2, -1),
}
# Plotting
nx.draw_networkx(
    G,
    pos=pos,
    with_labels=True,
    node_color="white",
    node_size=500,
    edgecolors="black",
)
plt.axis("off")
plt.show()
```



例 10.2 のプロジェクト・ネットワークに対して、最遅開始時刻と最遅終了時刻を求めると、次の表のようになる。

v	$\mathcal{S}(v)$	$\tau(v)$	$LS(v)$	$LF(v)$
s	A, B	0	0	0
A	C, D	2	2	4
B	E	3	0	3
C	F	4	4	8
D	F	2	6	8
E	F, G	5	3	8
F	t	9	8	17
G	t	2	15	17
t	-	0	17	17

最遅開始時刻と最遅終了時刻を求めるアルゴリズムは下の通りである。ここで、点を逆トポロジカルオーダー(reverse topological order)で処理するとは、v のすべての後続作業 $u \in \mathcal{S}(v)$ が v より前に処理されるように点を順序付けることである。

Algorithm 10.2. ***Backward Pass Algorithm***

1. $LF(t) \leftarrow EF(t)$, $LS(t) \leftarrow EF(t)$.
2. ***For each*** $v \in V \setminus \{t\}$ *in reverse topological order* ***do***
    1. $LF(v) \leftarrow \min_{u \in \mathcal{S}(v)} LS(u)$.
    2. $LS(v) \leftarrow LF(v) - \tau(v)$.

10.3.3　スラック

作業点 v の**スラック**（Slack）を $SL(v)$ と表す。スラックは、

$$SL(v) = LS(v) - ES(v) = LF(v) - EF(v)$$

で与えられる。スラックは、作業 v の開始または終了を遅らせることができる時間を示す。

例 10.2 のプロジェクト・ネットワークに対して、スラックを求めると、次の表のようになる。

v	$\tau(v)$	$ES(v)$	$EF(v)$	$LS(v)$	$LF(v)$	$SL(v)$
s	0	0	0	0	0	0
A	2	0	2	2	4	2
B	3	0	3	0	3	0
C	4	2	6	4	8	2
D	2	2	4	6	8	4
E	5	3	8	3	8	0
F	9	8	17	8	17	0
G	2	8	10	15	17	7
t	0	17	17	17	17	0

スラックが 0 の作業はクリティカル作業である。したがって、例 10.2 のプロジェクト・ネットワークにおけるクリティカルパスは、$s \to B \to E \to F \to t$ であり、このパスの長さは 17 日である。

10.4　3 点見積もり

以上の方法では、作業の所要時間が確定していることを前提としている。しかし、実際には、所要時間の推定は難しい場合が多い。そこで、作業の所要時間を確率変数として扱い、3 点見積もり（Three-Point Estimation）という方法でその作業時間の平均と分散を推定する。

3 点見積もりでは、次の 3 つの所要時間を事前に与えられたとする。

- 楽観値（Optimistic Time, a）：予想される最短の所要時間。
- 最頻値（Most Likely Time, m）：最も可能性が高い所要時間。
- 悲観値（Pessimistic Time, b）：予想される最長の所要時間。

一般に、$a \leq m \leq b$ である。

この a、m、b に基づいて、所要時間の平均 $\hat{\mu}$ と分散 $\hat{\sigma}^2$ を次のように推定する。

> **ノート**
>
> 作業の所要時間がベータ分布に従うと仮定した場合は、式 10.1 が μ と σ^2 の良い推定量となる。これらの式の詳細な導出は省略する。興味のある読者は、Pleguezuelo, Pérez, と Rambaud (2003) を参照されたい。

$$\hat{\mu} = \frac{a + 4m + b}{6}, \quad \hat{\sigma}^2 = \left(\frac{b - a}{6} \right)^2 \tag{10.1}$$

そこで、$\mu(v)$ と $\sigma^2(v)$ をそれぞれ作業点 v の所要時間の平均と分散とする。$a(v)$、$m(v)$、$b(v)$ をそれぞれ作業点 v の楽観値、最頻値、悲観値とする。このとき、すべての作業点 $v \in V$ に対して、

$$\mu(v) = \frac{a(v) + 4m(v) + b(v)}{6}, \quad \sigma^2(v) = \left(\frac{b(v) - a(v)}{6} \right)^2$$

で $\mu(v)$ と $\sigma^2(v)$ を計算する。

例 10.2 のプロジェクト・ネットワークに対して、各作業点の楽観値、最頻値、悲観値が次のように与えられたとするとき、平均と分散を求めると、次の表のようになる。

v	a	m	b	μ	σ^2
s	0	0	0	0	0
A	0.5	2	3.5	2	0.25
B	1.5	3	4.5	3	0.25
C	1.5	3	4.5	3	0.25
D	0.5	2	3.5	2	0.25
E	1	5.5	7	5	1
F	5	8	17	9	4
G	0.5	2	3.5	2	0.25
t	0	0	0	0	0

10.5　不確実性を考慮したPERT/CPM

3 点見積もりを用いて、各作業点 $v \in V$ の所要時間の平均 $\mu(v)$ と分散 $\sigma^2(v)$ を求めたとする。このとき、与えられた時間までにプロジェクトが完了する確率を求めたい。例えば,「プロジェクトが 30 日以内に完了する確率はどれくらいか？」という問いである。

プロジェクト・ネットワークにおけるパス $P = (s, v_{i_1}, v_{i_2}, \dots, v_{i_k}, t)$ の所要時間の平均 $\mu(P)$ と分散 $\sigma^2(P)$ は、それぞれ次のように推定できる。

$$\mu(P) = \sum_{v \in P} \mu(v), \quad \sigma^2(P) = \sum_{v \in P} \sigma^2(v)$$

パス P の所要時間の標準偏差 $\sigma(P)$ は、

$$\sigma(P) = \sqrt{\sigma^2(P)}$$

で与えられる。求められたパスの中で、所要時間の平均 $\mu(P)$ が最大のパスをクリティカルパス P^* とする。セクション 10.3 で述べた方法でクリティカルパスを求めることができる。

クリティカルパス P^* の所要時間を X とし、与えられた時間 d までにプロジェクトが完了する確率を $\mathbb{P}(X \leq d)$ と表す。

> **ノート**
>
> ここでは、計算を簡単にするために、クリティカルパスが最も時間がかかるパスであると仮定している。しかし, 実際には、平均値から計算されたクリティカルパスが、最も時間がかかるパスであるとは限らない。これにより, クリティカルパスをを用いて計算された確率は, 実際の確率よりも過大評価される可能性がある。
> ここで述べる方法をすべてのパスに適用し、各パスの確率を計算すれば、より正確な確率を得ることができる

$$\mathbb{P}(X \leq d) = \phi\left(\frac{d - \mu(P^*)}{\sigma(P^*)}\right)$$

ここで、$\phi(z)$ は標準正規分布の確率密度関数である。z は次のように計算し、標準正規分布表を用いて $\phi(z)$ を調べる。

$$z = \frac{d - \mu(P^*)}{\sigma(P^*)}$$

例 10.2 では、クリティカルパスは $s \to B \to E \to F \to t$ であり、このパスの所要時間の平均、分散、標準偏差は次のように計算できる。

$$\mu(P^*) = 3 + 5 + 9 = 17$$

$$\sigma^2(P^*) = 0.25 + 1 + 4 = 5.25$$

$$\sigma(P^*) = \sqrt{5.25} \approx 2.29$$

与えられた時間 $d = 20$ 日までにプロジェクトが完了する確率を求めるには、次のように計算する。

z を計算すると、

$$z = \frac{20 - 17}{2.29} \approx 1.31$$

標準正規分布表を用いて、$\phi(1.31) \approx 0.9049$ である。したがって、与えられた時間 $d = 20$ 日までにプロジェクトが完了する確率は約 90.49% である。

10.6 時間とコストのトレードオフ

プロジェクトの所要時間を短縮するために、追加のコストをかけて作業時間を短縮することができる場合がある。このような場合、決められた時間までにプロジェクトを完了するために、どの作業をどれだけ短縮すればよいかを決定する問題を考える。

10.6.1 問題の定式化

プロジェクト・ネットワーク $G = (V, E)$ に対して、各作業点 $i \in V$ に次のパラメータが与えられるとする。

- c_i: 作業 i の単位時間あたりの短縮コスト
- M_i: 作業 i の最大短縮時間
- T: プロジェクトの完了時間の上限
- τ_i: 作業 i の標準所要時間

決定変数は次の通りである。

- y_i: 作業 i の開始時間

- x_i: 作業 i の短縮時間

プロジェクトの所要時間を短縮するためのコストは

$$\sum_{i \in V} c_i x_i$$

で与えられる。j が i の後続作業である場合, 作業 j はすべての先行作業 i が完了した後に開始できるため,

$$y_j \geq y_i + \tau_i - x_i, \quad \forall (i,j) \in E$$

がある. また、各作業の最大短縮時間 M_i により、すべての作業 $i \in V$ について、

$$0 \leq x_i \leq M_i$$

である。プロジェクト全体の完了時間は終点 t の終了時間 y_t であり、これが上限 T 以下である必要があるから、

$$y_t \leq T$$

である。始点 s の開始時間は 0 とするから、

$$y_s = 0$$

である。

このとき、この問題は次の線形計画問題として定式化できる。

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in V} c_i x_i & \\
\text{subject to} \quad & y_j \geq y_i + \tau_i - x_i, & \forall (i,j) \in E \\
& 0 \leq x_i \leq M_i, & \forall i \in V \\
& y_t \leq T & \\
& y_s = 0 &
\end{aligned}$$

10.6.2 例題

プロジェクト・ネットワークはグラフ $G = (V, E)$ で表す。作業の集合は

$$V = \{s, t, A, B, C, D, E\}$$

であり、辺の集合は

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

である。ここで、s は始点、t は終点である。プロジェクト全体の所要時間の上限を $T = 10$ とする。

作業時間、コスト、最大短縮時間は次の表のように与えられる。

作業	τ_i	c_i	M_i
A	7	100	3
B	3	150	1
C	6	200	2
D	3	150	2
E	2	250	1

この問題を次のように定式化できる。

$$
\begin{aligned}
\text{minimize} \quad & 100x_A + 150x_B + 200x_C + 150x_D + 250x_E \\
\text{subject to} \quad & y_A \geq 0 \\
& y_C \geq 0 \\
& y_B \geq y_A + 7 - x_A \\
& y_D \geq y_C + 6 - x_C \\
& y_E \geq y_B + 3 - x_B \\
& y_E \geq y_D + 3 - x_D \\
& y_t \geq y_E + 2 - x_E \\
& 0 \leq x_A \leq 3 \\
& 0 \leq x_B \leq 1 \\
& 0 \leq x_C \leq 2 \\
& 0 \leq x_D \leq 2 \\
& 0 \leq x_E \leq 1 \\
& y_t \leq 10 \\
& y_s = 0
\end{aligned}
$$

この線形計画問題は、最適化ソルバーを用いて解くことができる。以下に Gurobi を用いた実装例を示す。

```
!pip install gurobipy
from gurobipy import Model, GRB

# データ定義
V = ["s", "t", "A", "B", "C", "D", "E"]
E = [("s", "A"), ("s", "C"), ("A", "B"), ("C", "D"), ("B", "E"), ("D", "E"), ("E",

tau = {"A": 7, "B": 3, "C": 6, "D": 3, "E": 2}
cost = {"A": 100, "B": 150, "C": 200, "D": 150, "E": 250}
M = {"A": 3, "B": 1, "C": 2, "D": 2, "E": 1}

T = 10

# モデル作成
m = Model("Project_Crashing")
m.setParam("OutputFlag", 0)
```

```
# 変数定義
x = {i: m.addVar(lb=0, ub=M[i], name="x_{}".format(i)) for i in M}
y = {i: m.addVar(lb=0, name="y_{}".format(i)) for i in V}

# 目的関数
m.setObjective(sum(cost[i] * x[i] for i in M), GRB.MINIMIZE)

# 制約
for i, j in E:
    if i == "s":
        m.addConstr(y[j] >= 0)
    elif i in tau:
        m.addConstr(y[j] >= y[i] + tau[i] - x[i])

# プロジェクト終了制約
m.addConstr(y["t"] <= T)

# 開始ノード固定
m.addConstr(y["s"] == 0)

# 最適化
m.optimize()

# 結果表示
print("Objective:", m.objVal)
for i in x:
    print("x_{} =".format(i), x[i].X)
for i in y:
    print("y_{} =".format(i), y[i].X)
```

```
Objective: 350.0
x_A = 1.0
x_B = 0.0
x_C = 0.0
x_D = 0.0
x_E = 1.0
y_s = 0.0
y_t = 10.0
y_A = 0.0
```

```
y_B = 6.0
y_C = 0.0
y_D = 6.0
y_E = 9.0
```

この結果から、作業 A を 1 日、作業 E を 1 日短縮することで、最小コスト 350 でプロジェクトを 10 日以内に完了できることがわかる。

10.7　文献案内

10.8　練習問題

練習 10.1. 次の作業リストに基づいて、以下の問いに答えよ。

作業	先行作業	時間(日)
A	-	4
B	-	3
C	A	2
D	A	5
E	B	6
F	C, D	4
G	E	3

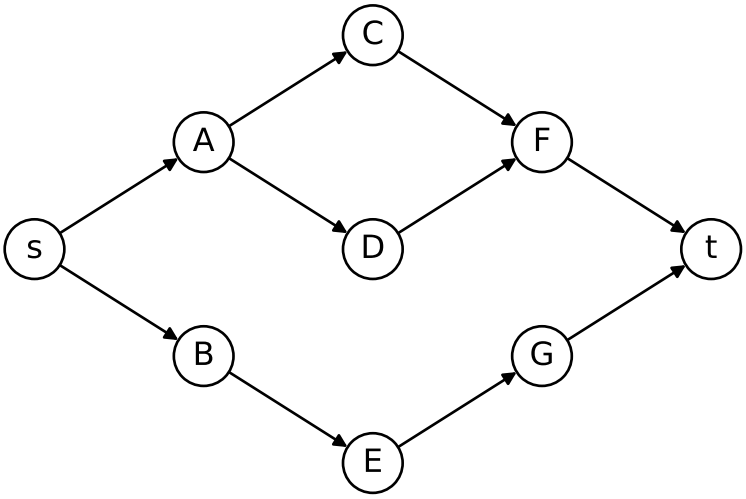
1. プロジェクト・ネットワークを AON で書け。
2. 各作業の最早開始時刻、最早終了時刻、最遅開始時刻、最遅終了時刻、スラックを求めよ。
3. このプロジェクトのクリティカルパスを求めよ。

解答 10.1.

1. プロジェクト・ネットワークは次のようになる。

```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np

# グラフの作成(頂点を番号で定義)
G = nx.DiGraph()
edges = [
    ("s", "A"),
    ("s", "B"),
```

```
    ("A", "C"),
    ("A", "D"),
    ("C", "F"),
    ("D", "F"),
    ("B", "E"),
    ("E", "G"),
    ("F", "t"),
    ("G", "t"),
]
G.add_edges_from(edges)
for layer, nodes in enumerate(nx.topological_generations(G)):
    for node in nodes:
        G.nodes[node]["layer"] = layer
pos = nx.multipartite_layout(G, subset_key="layer")
label_pos = {}
for k, (x, y) in pos.items():
    r = np.sqrt(x**2 + y**2) if np.sqrt(x**2 + y**2) > 0 else 1
    scale = 1.12
    label_pos[k] = (x * scale, y * scale)
# Plotting
nx.draw_networkx(
    G,
    pos=pos,
    with_labels=True,
    node_color="white",
    node_size=500,
    edgecolors="black",
)
plt.axis("off")
plt.show()
```



2~3. 省略

第IV部

データ包絡分析

第11章　はじめに

予備知識

- 線形代数
- 線形計画法

データ包絡分析（DEA: Data Envelopment Analysis）は, 分析対象の効率性を評価するための手法である．DEA では, 分析対象のことを **DMU**（Decision Making Unit）と呼ばれ, 複数の**入力**（input）から複数の**出力**（output）への変換を行うものである.

下の表は, いくつかの DMU の例を示している.

DMU	入力	出力
店舗	店員数, 面積	売上高, 顧客数
大学	教員数, 研究費	卒業生数, 論文数, 特許数
病院	医師数, 看護師数, ベッド数	患者数, 手術件数, 治癒率

11.1　1入力1出力

n 個の DMU があり，DMU_j $(j = 1, 2, \dots, n)$ は 1 つの入力 x_j と 1 つの出力 y_j を持つとする. そのとき，DMU_j の効率性は

$$\frac{y_j}{x_j}$$

で評価できる. 単位入力あたりの出力を表している. この値が大きいほど, 効率的であると評価できる.

例 11.1 (1 入力 1 出力の問題例)．ある会社は, 以下の 6 つの店舗を運営している. 効率的である店舗はどれか.

店舗	A	B	C	D	E	F
店員数(x)	2	3	3	4	2	5
売上高(y)	1	3	2	3	2	2

DMU$_j$ の店員数を x_j, 売上高を y_j とすると, 店舗 j の効率性は y_j/x_j で計算できる. 結果から, B, E が最も効率的であることがわかる.

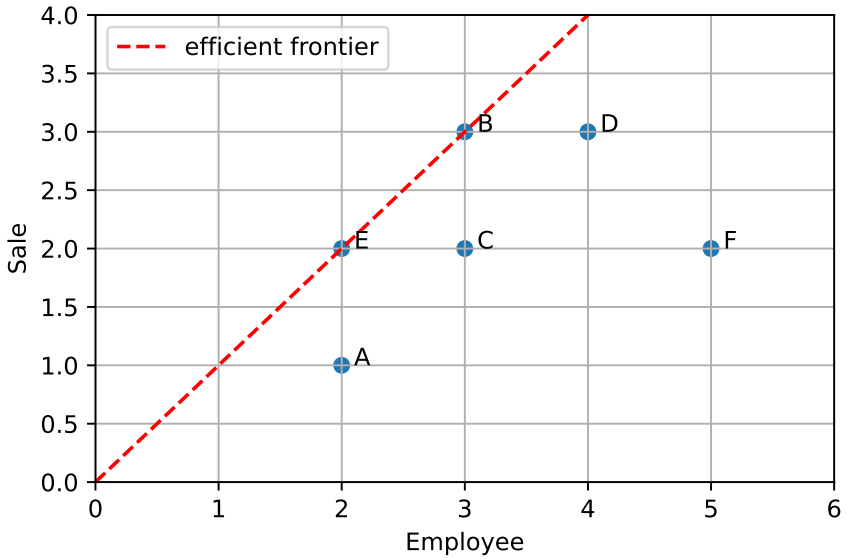
```
import matplotlib.pyplot as plt
import pandas as pd

data = {
    "store": ["A", "B", "C", "D", "E", "F"],
    "employee": [2, 3, 3, 4, 2, 5],
    "sale": [1, 3, 2, 3, 2, 2],
}
df = pd.DataFrame(data)
df["theta"] = df["sale"] / df["employee"]
print(df)
```

```
  store  employee  sale     theta
0     A         2     1  0.500000
1     B         3     3  1.000000
2     C         3     2  0.666667
3     D         4     3  0.750000
4     E         2     2  1.000000
5     F         5     2  0.400000
```

横軸に店員数, 縦軸に売上高をとって, 点 (x_j, y_j) をプロットした散布図を下の図に示す. 原点から点 (x_j, y_j) を結ぶ直線の傾きは, y_j/x_j に等しい.

```
plt.scatter(df["employee"], df["sale"])
for i in range(len(df)):
    plt.text(df["employee"][i] + 0.1, df["sale"][i], df["store"][i])
plt.plot([0, 4], [0, 4], color="red", linestyle="--", label="efficient frontier")
plt.xlabel("Employee")
plt.ylabel("Sale")
plt.xlim(0, 6)
plt.ylim(0, 4)
plt.legend()
plt.grid()
```

```
plt.show()
```



店舗 B と E は，**効率的**（efficient）と言える．また，原点から店舗 B（または店舗 E）を結ぶ直線を**効率的フロンティア**（efficient frontier）と呼ぶ．

11.2 2 入力 1 出力

例 11.2 (2 入力 1 出力の問題例)．ある会社は，以下の 6 つの店舗を運営している．効率的である店舗はどれか．

店舗	A	B	C	D	E	F
店員数	4	7	8	4	2	5
面積	3	3	1	2	4	2
売上高	1	1	1	1	1	1

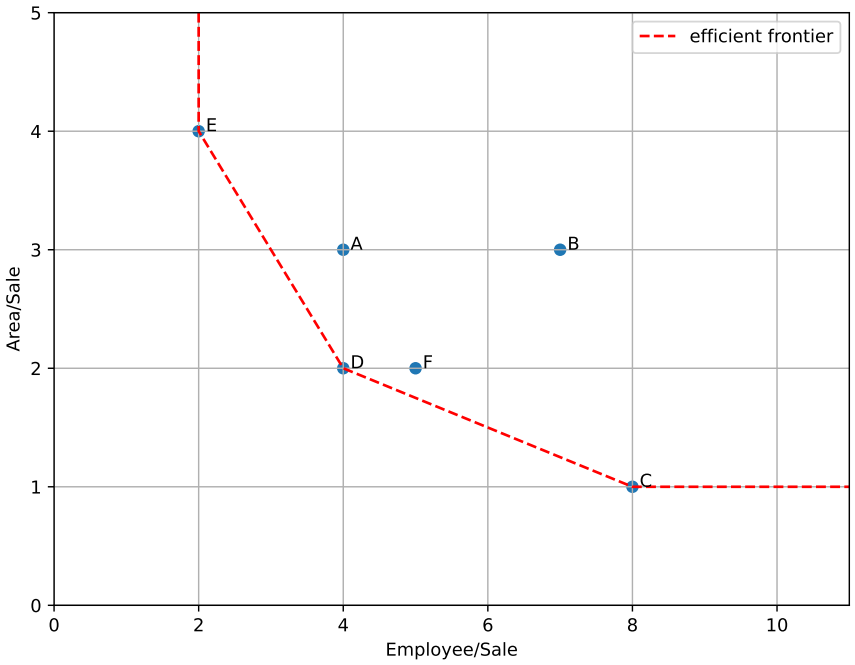
DMU_j の店員数を x_j，面積を y_j，売上高を z_j とする．それぞれの店舗に対して，次の 2 つの比率を計算する．

$$\frac{x_j}{z_j}, \quad \frac{y_j}{z_j}$$

これらの値が小さいほど，効率的であると評価できる．

x_j/z_j と y_j/z_j を横軸と縦軸にとった散布図を下の図に示す．

```
import matplotlib.pyplot as plt
import pandas as pd

data = {
    "store": ["A", "B", "C", "D", "E", "F"],
    "employee": [4, 7, 8, 4, 2, 5],
    "area": [3, 3, 1, 2, 4, 2],
    "sale": [1, 1, 1, 1, 1, 1],
}
df = pd.DataFrame(data)
plt.figure(figsize=(8, 6))
plt.scatter(df["employee"], df["area"])
for i in range(len(df)):
    plt.text(df["employee"][i] + 0.1, df["area"][i], df["store"][i])
plt.plot(
    [2, 2, 4, 8, 11],
    [5, 4, 2, 1, 1],
    color="red",
    linestyle="--",
    label="efficient frontier",
)
plt.xlabel("Employee/Sale")
plt.ylabel("Area/Sale")
plt.xlim(0, 11)
plt.ylim(0, 5)
plt.legend()
plt.grid()
plt.show()
```



DEA では, 任意の両点の線分上の点が実現可能であると仮定する. 例えば, 点 D と点 E を用いて, $(wx_D + (1-w)x_E, wy_D + (1-w)y_E)$ で表される点は, 実現可能であると仮定する.

C, D, E は効率的で, これらの点を結ぶ折れ線が効率的フロンティアとなる.

11.3 1 入力 2 出力

例 11.3 (1 入力 2 出力の問題例). ある会社は, 以下の 6 つの店舗を運営している. 効率的である店舗はどれか.

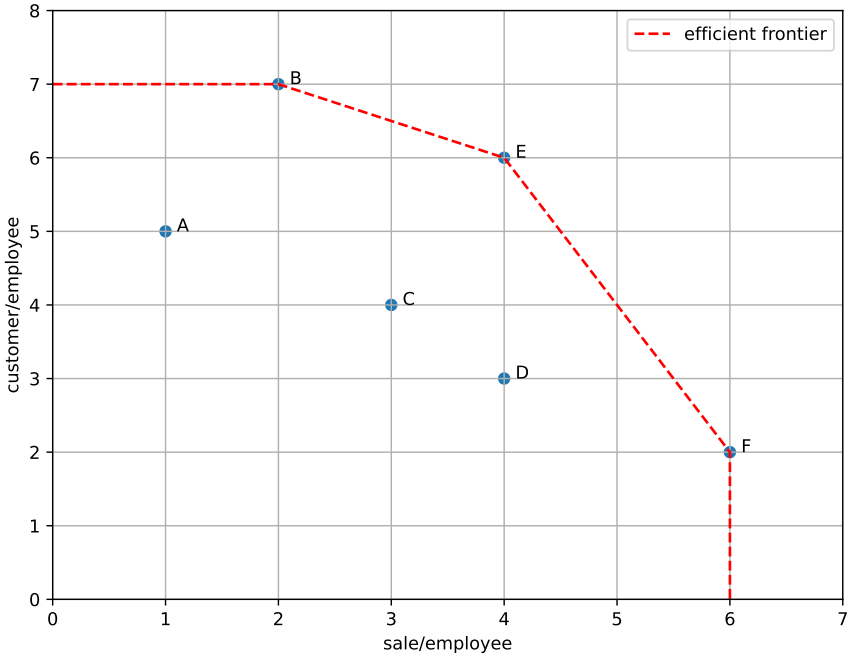
店舗	A	B	C	D	E	F
店員数	1	1	1	1	1	1
売上高	1	2	3	4	4	6
顧客数	5	7	4	3	6	2

DMU_j の店員数を x_j, 売上高を y_j, 顧客数を z_j とする. それぞれの店舗に対して, 次の 2 つの比率を計算する.

$$\frac{y_j}{x_j}, \quad \frac{z_j}{x_j}$$

これらの値が大きいほど, 効率的であると評価できる.

```
import matplotlib.pyplot as plt
import pandas as pd

data = {
    "store": ["A", "B", "C", "D", "E", "F"],
    "employee": [1, 1, 1, 1, 1, 1],
    "sale": [1, 2, 3, 4, 4, 6],
    "customer": [5, 7, 4, 3, 6, 2],
}
df = pd.DataFrame(data)
plt.figure(figsize=(8, 6))
plt.scatter(df["sale"], df["customer"])
for i in range(len(df)):
    plt.text(df["sale"][i] + 0.1, df["customer"][i], df["store"][i])
plt.plot(
    [0, 2, 4, 6, 6],
    [7, 7, 6, 2, 0],
    color="red",
    linestyle="--",
    label="efficient frontier",
)
plt.xlabel("sale/employee")
plt.ylabel("customer/employee")
plt.xlim(0, 7)
plt.ylim(0, 8)
plt.legend()
plt.grid()
plt.show()
```



店舗 B，E，F は効率的で, これらの点を結ぶ折れ線が効率的フロンティアとなる.

11.4 用語集

English	Japanese
Data Envelopment Analysis (DEA)	データ包絡分析
Decision Making Unit (DMU)	意思決定主体 ;意思決定単位
Reference Set	参照集合
Efficient	効率的
Efficient Frontier	効率的フロンティア

11.5 練習問題

練習 11.1 (1 入力 1 出力). 次の表のような入力と出力が与えられたとき, 効率的な DMU を求めよ. 効率性を $[0, 1]$ の範囲で評価せよ.

DMU	A	B	C	D	E
入力	4	7	8	5	6

DMU	A	B	C	D	E
出力	2	7	9	4	5

解答 11.1.

```
import pandas as pd

data = {
    "DMU": ["A", "B", "C", "D", "E"],
    "input": [4, 7, 8, 5, 6],
    "output": [2, 7, 9, 4, 5],
}
df = pd.DataFrame(data)
df["efficiency"] = df["output"] / df["input"]
print(df)
```

```
  DMU  input  output  efficiency
0   A      4       2    0.500000
1   B      7       7    1.000000
2   C      8       9    1.125000
3   D      5       4    0.800000
4   E      6       5    0.833333
```

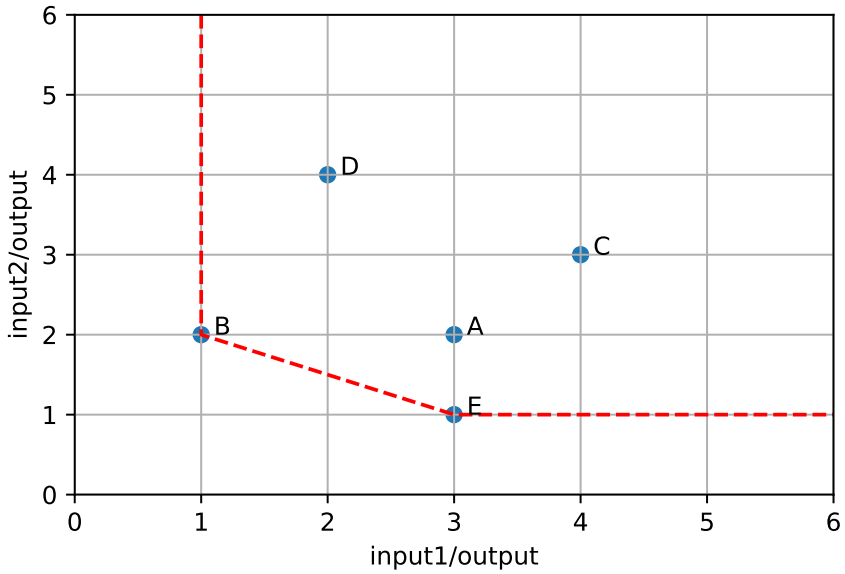
練習 11.2 (2 入力 1 出力)**.** 次の表のような入力と出力が与えられたとき, 効率的な DMU を求めよ.

DMU	A	B	C	D	E
入力 1	6	6	12	4	15
入力 2	4	12	9	8	5
出力	2	6	3	2	5

解答 11.2. 下の図より，DMU B，D が効率的であることがわかる.

```
import pandas as pd
import matplotlib.pyplot as plt

data = {
    "DMU": ["A", "B", "C", "D", "E"],
    "input1": [6, 6, 12, 4, 15],
```

```
    "input2": [4, 12, 9, 8, 5],
    "output": [2, 6, 3, 2, 5],
}
df = pd.DataFrame(data)
x = df["input1"] / df["output"]
y = df["input2"] / df["output"]
plt.scatter(x, y)
for i in range(len(df)):
    plt.text(x[i] + 0.1, y[i], df["DMU"][i])
# 効率的フロンティアの描画 B-E
plt.plot(
    [1, 1, 3, 6],
    [6, 2, 1, 1],
    color="red",
    linestyle="--",
    label="efficient frontier",
)
plt.xlabel("input1/output")
plt.ylabel("input2/output")
plt.xlim(0, 6)
plt.ylim(0, 6)
plt.grid()
plt.show()
```



第12章　CCRモデル

CCR モデルは，1978 年に Charnes, Cooper, and Rhodes によって提案されたもので，DEA の基本的なモデルである．

n 個の DMU を評価する. 各 DMU は m 個の入力と s 個の出力を持つとする．DMU_j $(j = 1, 2, \ldots, n)$ の i 番目の入力を x_{ij}，r 番目の出力を y_{rj} と表す. このとき, 入力データと出力データはそれぞれ次のような行列で表される．

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix}$$

$$\mathbf{Y} = \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{s1} & y_{s2} & \cdots & y_{sn} \end{pmatrix}$$

入力の重み $\mathbf{v}$ と出力の重み $\mathbf{u}$ を

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix}, \quad \mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_s \end{pmatrix}$$

とし，DMU_o の効率性 θ は

$$\theta = \frac{\sum_{r=1}^{s} u_r y_{ro}}{\sum_{i=1}^{m} v_i x_{io}} = \frac{u_1 y_{1o} + u_2 y_{2o} + \cdots + u_s y_{so}}{v_1 x_{1o} + v_2 x_{2o} + \cdots + v_m x_{mo}}$$

として計算できる．

CCR モデルの考え方は，DMU_o の効率性を最大化するような重み $\mathbf{u}$ と $\mathbf{v}$ を選ぶことである．DMU_o は最適な重みを用いたとき, 他の DMU との比較を行う．

12.0.1　CCR モデルの定式化

CCR モデルは次の分数計画問題として定式化できる．

$$
\begin{aligned}
\text{maximize} \quad & \theta = \frac{\sum_{r=1}^{s} u_r y_{ro}}{\sum_{i=1}^{m} v_i x_{io}} \\
\text{subject to} \quad & \frac{\sum_{r=1}^{s} u_r y_{rj}}{\sum_{i=1}^{m} v_i x_{ij}} \leq 1, && \forall j = 1, \dots, n \\
& u_r \geq 0, && \forall r = 1, \dots, s \\
& v_i \geq 0, && \forall i = 1, \dots, m
\end{aligned}
$$

この分数計画問題は，θ を最大化する重み $\mathbf{u}$ と $\mathbf{v}$ を求めるものである. 制約条件は, すべての DMU の効率性が 1 以下であることを保証している．

> **ノート**
>
> 厳密に, 制約条件は
>
> $$\frac{u_r}{\sum_{i=1}^{m} v_i x_{io}} \geq \epsilon > 0, \quad \forall r = 1, \dots, s$$
>
> $$\frac{v_i}{\sum_{i=1}^{m} v_i x_{io}} \geq \epsilon > 0, \quad \forall i = 1, \dots, m$$
>
> とする必要がある．

この分数計画問題において, 任意の $\alpha > 0$ に対して，$(\mathbf{u}^*, \mathbf{v}^*)$ を最適解とすると，$(\alpha\mathbf{u}^*, \alpha\mathbf{v}^*)$ も最適解となる. したがって, この分数計画問題は無限に多くの最適解を持つ. この問題を解決するために, 分母を 1 に固定することで, 次の線形計画問題に変換できる．

$$
\begin{aligned}
\text{maximize} \quad & \theta = \sum_{r=1}^{s} u_r y_{ro} \\
\text{subject to} \quad & \sum_{i=1}^{m} v_i x_{io} = 1 \\
& \sum_{r=1}^{s} u_r y_{rj} - \sum_{i=1}^{m} v_i x_{ij} \leq 0, \qquad \forall j = 1, \ldots, n \\
& u_r \geq 0, \qquad \forall r = 1, \ldots, s \\
& v_i \geq 0, \qquad \forall i = 1, \ldots, m
\end{aligned}
$$

定義 12.1.

1. $\theta^* = 1$ かつ最適解 $\mathbf{u}^* > \mathbf{0}$，$\mathbf{v}^* > \mathbf{0}$ を満たすとき，DMU_o は効率的である．
2. $\theta^* < 1$ のとき，DMU_o は非効率的である．

$\theta^* < 1$ のとき, 少なくとも 1 つの $j \in \{1, 2, \ldots, n\}$ に対して, 次の式が成り立つ．

$$\sum_{r=1}^{s} u_r^* y_{rj} - \sum_{i=1}^{m} v_i^* x_{ij} = 0$$

そうでなければ，θ^* を増加させることができる. また，$\theta^* < 1$ から，$j \neq o$ であることがわかる．

正式に，DMU_o における $\theta^* < 1$ のとき, 次の集合を DMU_o の**参照集合**（reference set）と呼ぶ．

$$\mathcal{R}_o = \{j \mid \sum_{r=1}^{s} u_r^* y_{rj} - \sum_{i=1}^{m} v_i^* x_{ij} = 0\}$$

参照集合 $\mathcal{R}_o$ は，DMU_o を非効率的にしている DMU の集合である．

12.0.2 Python による CCR モデルの実装

以下に，Python で CCR モデルを解く関数を示す．Gurobi をソルバーとして使用している．

```
!pip install gurobipy
from gurobipy import Model, GRB
import numpy as np

def ccr(inputs, outputs, dmu_index):
```

```
    num_dmu = inputs.shape[1]
    num_inputs = inputs.shape[0]
    num_outputs = outputs.shape[0]
    model = Model("DEA_CCR")
    # 変数の定義
    u = [model.addVar(name=f"u_{r}", lb=0) for r in range(num_outputs)]
    v = [model.addVar(name=f"v_{i}", lb=0) for i in range(num_inputs)]
    # 目的関数の定義
    model.setObjective(
        sum(u[r] * outputs[r, dmu_index] for r in range(num_outputs)),
        GRB.MAXIMIZE,
    )
    # 制約条件の定義
    model.addConstr(
        sum(v[i] * inputs[i, dmu_index] for i in range(num_inputs)) == 1,
        "input_normalization",
    )
    for j in range(num_dmu):
        model.addConstr(
            sum(u[r] * outputs[r, j] for r in range(num_outputs))
            - sum(v[i] * inputs[i, j] for i in range(num_inputs))
            <= 0,
            f"theta_constraint_{j}",
        )
    # 最適化の実行
    model.optimize()
    if model.status == GRB.OPTIMAL:
        theta = model.objVal
        output_weights = [u[r].X for r in range(num_outputs)]
        input_weights = [v[i].X for i in range(num_inputs)]
        ref_set = []
        for j in range(num_dmu):
            constr = model.getConstrByName(f"theta_constraint_{j}")
            if abs(constr.Slack) == 0:
                ref_set.append(j)
        return theta, output_weights, input_weights, ref_set
    else:
        return None
```

`ccr()` 関数を用いて, すべての DMU について CCR モデルを解く関数を以

下に示す.

```
def solve_all_ccr(inputs, outputs):
    inputs = np.atleast_2d(inputs)
    outputs = np.atleast_2d(outputs)

    num_dmu = inputs.shape[1]
    dmu_names = [chr(65 + i) for i in range(num_dmu)]

    results = []
    u_list = []
    v_list = []
    ref_sets = []

    for i in range(num_dmu):
        theta, u_val, v_val, ref_set = ccr(inputs, outputs, i)
        results.append(theta)
        u_list.append(u_val)
        v_list.append(v_val)
        ref_sets.append([dmu_names[j] for j in ref_set])

    for i in range(num_dmu):
        name = dmu_names[i]
        theta = results[i]
        u_val = u_list[i]
        v_val = v_list[i]
        ref_set = ref_sets[i]

        formatted_u = ", ".join(f"{w:.4f}" for w in u_val)
        formatted_v = ", ".join(f"{w:.4f}" for w in v_val)
        formatted_ref_set = ", ".join(ref_set)

        print(f"DMU {name}")
        print(f"  theta            : {theta:.4f}")
        print(f"  Output Weights u : [{formatted_u}]")
        print(f"  Input Weights  v : [{formatted_v}]")
        print(f"  Reference Set    : [{formatted_ref_set}]")
        print("-" * 50)
```

12.1　CCR モデルの例題

12.1.1　1 入力 1 出力の CCR モデル

DMU	A	B	C	D	E	F
Input	2	3	3	4	2	5
Output	1	3	2	3	2	2

入力に対する重みを v, 出力に対する重みを u とする．DMU_A の効率性を評価する CCR モデルは次のようになる．

$$
\begin{aligned}
\text{maximize} \quad & u \\
\text{subject to} \quad & 2v = 1 \\
& u - 2v \le 0 && (A) \\
& 3u - 3v \le 0 && (B) \\
& 2u - 3v \le 0 && (C) \\
& 3u - 4v \le 0 && (D) \\
& 2u - 2v \le 0 && (E) \\
& 2u - 5v \le 0 && (F) \\
& u \ge 0 \\
& v \ge 0
\end{aligned}
$$

$2v = 1$ より，$v = 0.5$ であるから, この等式を用いて, この線形計画問題は簡単に解ける. 最適値は $\theta^* = 0.5$, 重みは $u^* = 0.5$，$v^* = 0.5$ である．

$\theta^* = 0.5 < 1$ から，DMU_A は非効率的であることがわかる. また，u^* と v^* を代入し, 制約条件 (B) と (E) が等号で成り立っているため，DMU_A の参照集合は $\mathcal{R}_A = \{B, E\}$ となる. さらに,

$$\theta = \frac{u^* y_A}{v^* x_A} = 0.5$$

であるから，A の出力 y_A を 2 倍するか, 入力 x_A を 0.5 倍すれば, 効率的になることがわかる．

同様にして, すべての DMU について CCR モデルを解くと, 次の結果が得られる．

```
X = np.array([[2, 3, 3, 4, 2, 5]])
Y = np.array([1, 3, 2, 3, 2, 2])
solve_all_ccr(X, Y)
```

```
DMU A
  theta            : 0.5000
  Output Weights u : [0.5000]
  Input Weights  v : [0.5000]
  Reference Set    : [B, E]
--------------------------------------------------
DMU B
  theta            : 1.0000
  Output Weights u : [0.3333]
  Input Weights  v : [0.3333]
  Reference Set    : [B, E]
--------------------------------------------------
DMU C
  theta            : 0.6667
  Output Weights u : [0.3333]
  Input Weights  v : [0.3333]
  Reference Set    : [B, E]
--------------------------------------------------
DMU D
  theta            : 0.7500
  Output Weights u : [0.2500]
  Input Weights  v : [0.2500]
  Reference Set    : [B, E]
--------------------------------------------------
DMU E
  theta            : 1.0000
  Output Weights u : [0.5000]
  Input Weights  v : [0.5000]
  Reference Set    : [B, E]
--------------------------------------------------
DMU F
  theta            : 0.4000
  Output Weights u : [0.2000]
  Input Weights  v : [0.2000]
  Reference Set    : [B, E]
```

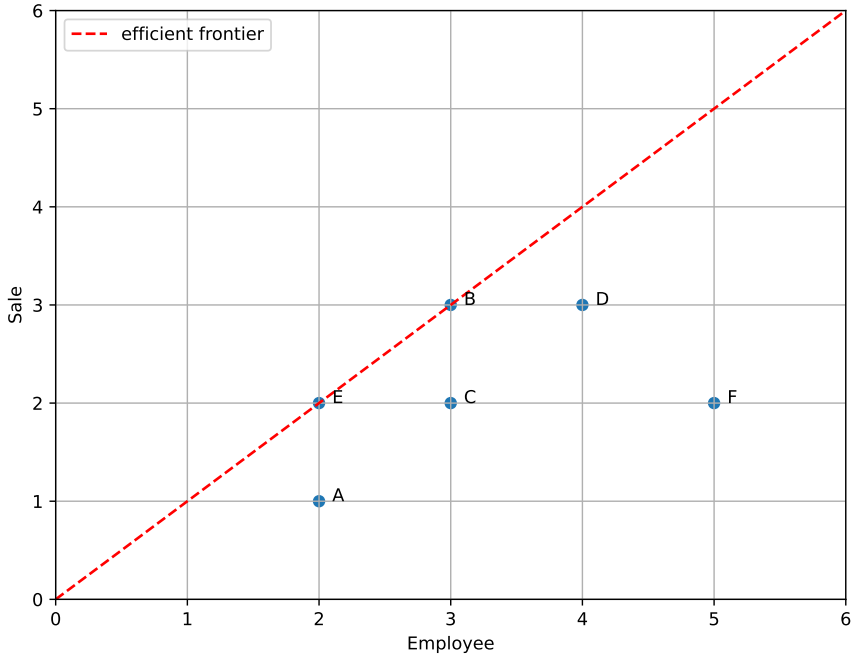
$\{B, E\}$ が他の DMU の参照集合となっていることもわかる.

1 入力 1 出力の分数計画問題において, 目的関数は

$$\frac{uy_o}{vx_o} = \frac{u}{v}\frac{y_o}{x_o}$$

となり，y_o/x_o に比例する. したがって，1 入力 1 出力の CCR モデルの最適化問題を解くことは, すべての DMU について y_j/x_j を計算し，$[0, 1]$ に正規化することと同じである. 今回の例題では，DMU B と E の $\theta^* = 1$ で最大であるため，$u/v = 1$ となる.

```
import matplotlib.pyplot as plt
import pandas as pd

data = {
    "store": ["A", "B", "C", "D", "E", "F"],
    "employee": [2, 3, 3, 4, 2, 5],
    "sale": [1, 3, 2, 3, 2, 2],
}
df = pd.DataFrame(data)
df["theta"] = [0.5, 1.0, 0.67, 0.75, 1.0, 0.4]
df["u"] = [0.5, 0.33, 0.33, 0.25, 0.5, 0.2]
df["v"] = [0.5, 0.33, 0.33, 0.25, 0.5, 0.2]
plt.figure(figsize=(8, 6))
plt.scatter(df["employee"], df["sale"])
for i in range(len(df)):
    plt.text(df["employee"][i] + 0.1, df["sale"][i], df["store"][i])
plt.plot([0, 6], [0, 6], color="red", linestyle="--", label="efficient frontier")
plt.xlabel("Employee")
plt.ylabel("Sale")
plt.xlim(0, 6)
plt.ylim(0, 6)
plt.legend()
plt.grid()
plt.show()
```



12.1.2 2 入力 1 出力の CCR モデル

DMU	A	B	C	D	E	F
Input 1	4	7	8	4	2	10
Input 2	3	3	1	2	4	1
Output	1	1	1	1	1	1

入力 1 に対する重みを v_1, 入力 2 に対する重みを v_2, 出力に対する重みを u とする. DMU_A の効率性を評価する CCR モデルは次のようになる.

$$
\begin{aligned}
\text{maximize} \quad & u \\
\text{subject to} \quad & 4v_1 + 3v_2 = 1 \\
& u - 4v_1 - 3v_2 \leq 0 \qquad (A) \\
& u - 7v_1 - 3v_2 \leq 0 \qquad (B) \\
& u - 8v_1 - 1v_2 \leq 0 \qquad (C) \\
& u - 4v_1 - 2v_2 \leq 0 \qquad (D) \\
& u - 2v_1 - 4v_2 \leq 0 \qquad (E) \\
& u - 10v_1 - 1v_2 \leq 0 \qquad (F) \\
& u \geq 0 \\
& v_1 \geq 0 \\
& v_2 \geq 0
\end{aligned}
$$

ソルバーでこの線形計画問題を解くと, 最適値は $\theta^* = 0.8571$, 重みは $u^* = 0.8571$, $v_1^* = 0.1429$, $v_2^* = 0.1429$ である. また, u^*, v_1^*, v_2^* を制約条件に代入し, A の参照集合は $\mathcal{R}_A = \{D, E\}$ となる.

12.1.3　2 入力 2 出力の CCR モデル

DMU	A	B	C	D	E	F	G	H	I	J	K	L
Input 1	20	19	25	27	22	55	33	31	30	50	53	38
Input 2	151	131	160	168	158	255	235	206	244	268	306	284
Output 1	100	150	160	180	94	230	220	152	190	250	260	250
Output 2	90	50	55	72	66	90	88	80	100	100	147	120

DMU_A の効率性を評価する CCR モデルは次のようになる.

$$
\begin{aligned}
\text{maximize} \quad & 100u_1 + 90u_2 \\
\text{subject to} \quad & 20v_1 + 151v_2 = 1 \\
& 100u_1 + 90u_2 - 20v_1 - 151v_2 \leq 0 \\
& 150u_1 + 50u_2 - 19v_1 - 131v_2 \leq 0 \\
& 160u_1 + 55u_2 - 25v_1 - 160v_2 \leq 0 \\
& 180u_1 + 72u_2 - 27v_1 - 168v_2 \leq 0 \\
& 94u_1 + 66u_2 - 22v_1 - 158v_2 \leq 0 \\
& 230u_1 + 90u_2 - 55v_1 - 255v_2 \leq 0 \\
& 220u_1 + 88u_2 - 33v_1 - 235v_2 \leq 0 \\
& 152u_1 + 80u_2 - 31v_1 - 206v_2 \leq 0 \\
& 190u_1 + 100u_2 - 30v_1 - 244v_2 \leq 0 \\
& 250u_1 + 100u_2 - 50v_1 - 268v_2 \leq 0 \\
& 260u_1 + 147u_2 - 53v_1 - 306v_2 \leq 0 \\
& 250u_1 + 120u_2 - 38v_1 - 284v_2 \leq 0 \\
& u_r \geq 0, \quad \forall r = 1, 2 \\
& v_i \geq 0, \quad \forall i = 1, 2
\end{aligned}
$$

第Ⅴ部

動的計画法

第13章　動的計画法

> **学習目標**
>
> - 動的計画法による最短路問題とナップサック問題の解法を理解する。
> - 動的計画法の基本的な考え方を理解する。
> - *ベルマン方程式の意味を理解する。

> **Principle of Optimality**: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.
>
> **最適性の原理**（Principle of Optimality）：最適な方策は、初期状態と初期決定がどんなものであれ、その結果得られる次の状態に関して、以降の決定が必ず最適方策になっているという性質をもつ。
>
> – Richard Bellman, 1957

動的計画法（dynamic programming, DP）は Richard E. Bellman によって提案された，**多段階決定過程**（multi-stage decision process）に対する最適化手法である．

> **ノート**
>
> 動的計画法を algorithmic paradigm の一つとして扱う場合もあるが，本書では最適化手法として説明する．
> 大学では，情報，制御工学，経営工学など，様々な分野で動的計画法が教えられている．

多段階決定過程は，複数の段階にわたって行われる一連の意思決定を行う問題である．各段階 t において，意思決定者は現在の**状態**を観測し，意思決定を行う．その意思決定は，将来の状態に影響を与える．目的として，状態に関するある目的関数を最小化（または最大化）することである．動的計画法は，多段階

決定過程における**最適方策**（optimal policy）を求めるための一般的な手法である．

最適化, 制御, オペレーションズ・リサーチなどの分野において, 多数の応用例が知られている．

- 在庫管理問題
- 自動運転
- ロボット制御
- 生物学
- 経済学・金融工学
- …

さらに, 現在では, 強化学習の基礎理論としても重要な役割を果たしている．

ここでは, 動的計画法の基本的な考え方を理解するために，**最短路問題**（shortest path problem）と**ナップサック問題**（knapsack problem）を取り上げる．

動的計画法の名前の由来

> I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, “Where did the name, dynamic programming, come from?” The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word “research”. I’m not using the term lightly; I’m using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word “programming”. I wanted to get

> across the idea that this was dynamic, this was multistage, this was time-varying. I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities.
>
> — Richard Bellman, Eye of the Hurricane: An Autobiography (1984, page 159)

13.1 最短路問題

多段階の有向グラフ $G = (V, E)$ における最短路問題を考える。各辺 $(v, u) \in E$ に非負の重み $w(v, u)$ が与えられているとする。始点 $s \in V$ から終点 $t \in V$ への最短路を求める問題である。s と t 以外の頂点を段階 $D = \{1, 2, \dots, n\}$ に分けることができるとする。結果として，n 回の意思決定を行い，最適解は $(s, v_1, v_2, \dots, v_n, t)$ の形になる。

$P^* = (s, v_1, v_2, \dots, v_n, t)$ は最短路であれば，任意の $1 \leq k \leq n$ に対して，パス $(v_k, v_{k+1}, \dots, v_n, t)$ も v_k から t への最短路である。これは，もし $(v_k, v_{k+1}, \dots, v_n, t)$ が最短路でないとすると，より短いパスが存在し，P^* も最短路でなくなるためである。この性質は**最適性の原理**（Principle of Optimality）と呼ばれる。

点 v から点 t までの最短路の長さ（またはコスト）を $d(v)$ と定義する。最適性の原理により，点 $v \in V \setminus \{t\}$ に対して，

$$d(v) = \min_{u \in \mathcal{P}(v)} \{w(v, u) + d(u)\}$$

が成り立つ。ここで，$\mathcal{P}(v) = \{u \mid (v, u) \in E\}$ とする。

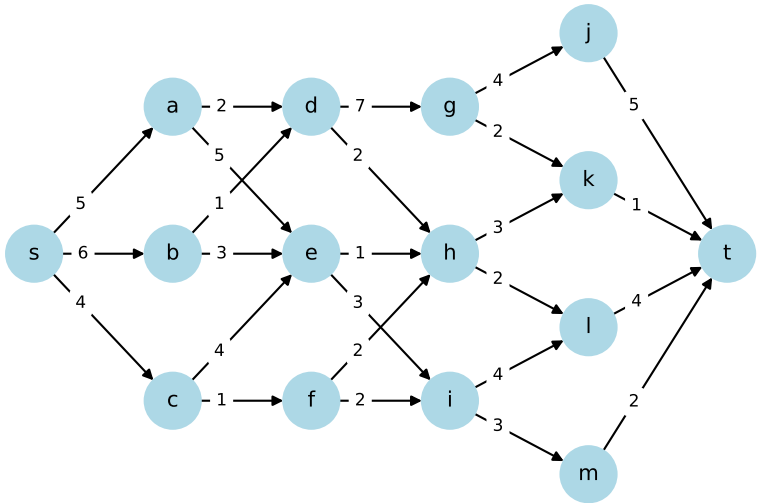
例 13.1. 下のグラフにおいて，$d(a)$，$d(b)$，$d(c)$ が与えられたとき，$d(s)$ を計算せよ。

- $d(a) = 8$
- $d(b) = 7$

- $d(c) = 7$

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.DiGraph()
edges = [
    ("s", "a", 5),
    ("s", "b", 6),
    ("s", "c", 4),
    ("a", "d", 2),
    ("a", "e", 5),
    ("b", "d", 1),
    ("b", "e", 3),
    ("c", "e", 4),
    ("c", "f", 1),
    ("d", "g", 7),
    ("d", "h", 2),
    ("e", "h", 1),
    ("e", "i", 3),
    ("f", "h", 2),
    ("f", "i", 2),
    ("g", "j", 4),
    ("g", "k", 2),
    ("h", "k", 3),
    ("h", "l", 2),
    ("i", "l", 4),
    ("i", "m", 3),
    ("j", "t", 5),
    ("k", "t", 1),
    ("l", "t", 4),
    ("m", "t", 2),
]
G.add_weighted_edges_from(edges)

for layer, nodes in enumerate(nx.topological_generations(G)):
    for node in nodes:
        G.nodes[node]["layer"] = layer

pos = {
```

```
    "s": (0, 0),
    "a": (1, 1),
    "b": (1, 0),
    "c": (1, -1),
    "d": (2, 1),
    "e": (2, 0),
    "f": (2, -1),
    "g": (3, 1),
    "h": (3, 0),
    "i": (3, -1),
    "j": (4, 1.5),
    "k": (4, 0.5),
    "l": (4, -0.5),
    "m": (4, -1.5),
    "t": (5, 0),
}

nx.draw(G, pos, with_labels=True, node_size=700, node_color="lightblue", font_size=10)
edge_labels = nx.get_edge_attributes(G, "weight")
nx.draw_networkx_edge_labels(
    G, pos, edge_labels=edge_labels, label_pos=0.3, font_size=8, rotate=False
)
plt.show()
```



```
# find shortest path from s to t
source = "s"
target = "t"
shortest_path = nx.shortest_path(G, source=source, target=target, weight="weight")
shortest_path_length = nx.shortest_path_length(
    G, source=source, target=target, weight="weight"
)
shortest_path, shortest_path_length
```

```
(['s', 'c', 'f', 'h', 'k', 't'], 11)
```

$$
\begin{aligned}
d(s) &= \min\{w(s,a) + d(a), w(s,b) + d(b), w(s,c) + d(c)\} \\
&= \min\{5 + 8, 6 + 7, 4 + 7\} \\
&= \min\{13, 13, 11\} \\
&= 11
\end{aligned}
$$

動的計画法による最短路問題の解法を以下に示す。

Algorithm 13.1.

- ***Input***: 有向グラフ $G = (V, E)$, 重み関数 $w : E \to \mathbb{R}_{\geq 0}$, 始点 $s \in V$, 終点 $t \in V$
- ***Output***: s から t への最短路の長さ $d(s)$

1. $d(t) \leftarrow 0$
2. $d(v) \leftarrow \infty$ *for all* $v \in V \setminus \{t\}$
3. $k \leftarrow n - 1$ *(段階数)*
4. *while* $k \geq 0$ *do*
    1. *for each* $v \in D_k$ *do*
        1. $d(v) \leftarrow \min_{u \in \mathcal{P}(v)} \{w(v,u) + d(u)\}$
    2. $k \leftarrow k - 1$
5. *return* $d(s)$

```
def dp_shortest_path(G):
    # Initialize distances
    d = {v: float("inf") for v in G.nodes()}
    d["t"] = 0  # Distance to target is 0

    # Get layers in topological order
```

```
    layers = list(nx.topological_generations(G))

    # Process layers in reverse order
    for layer in reversed(layers[:-1]):  # Exclude the last layer (target)
        for v in layer:
            d[v] = min(
                (G[v][u]["weight"] + d[u] for u in G.successors(v)),
            )
    return d["s"]


if __name__ == "__main__":

    shortest_path_length_dp = dp_shortest_path(G)
    print(shortest_path_length_dp)
```

```
11
```

まず，$d(t) = 0$ とし，第 4 段の点 j，k，l，m について，$d(j)$，$d(k)$，$d(l)$，$d(m)$ を計算する。

$$
\begin{aligned}
d(j) &= \min\{w(j,t) + d(t)\} = \min\{5 + 0\} = 5 \\
d(k) &= \min\{w(k,t) + d(t)\} = \min\{1 + 0\} = 1 \\
d(l) &= \min\{w(l,t) + d(t)\} = \min\{4 + 0\} = 4 \\
d(m) &= \min\{w(m,t) + d(t)\} = \min\{2 + 0\} = 2
\end{aligned}
$$

次に，第 3 段の点 g，h，i について，$d(g)$，$d(h)$，$d(i)$ を計算する。

$$
\begin{aligned}
d(g) &= \min\{w(g,j) + d(j), w(g,k) + d(k)\} = \min\{4 + 5, 2 + 1\} = \min\{9, 3\} = 3 \\
d(h) &= \min\{w(h,k) + d(k), w(h,l) + d(l)\} = \min\{3 + 1, 2 + 4\} = \min\{4, 6\} = 4 \\
d(i) &= \min\{w(i,l) + d(l), w(i,m) + d(m)\} = \min\{4 + 4, 3 + 2\} = \min\{8, 5\} = 5
\end{aligned}
$$

第 2 段の点 d，e，f について，$d(d)$，$d(e)$，$d(f)$ を計算する。

$$d(d) = \min\{w(d,g) + d(g), w(d,h) + d(h)\} = \min\{7+3, 2+4\} = \min\{10, 6\} = 6$$
$$d(e) = \min\{w(e,h) + d(h), w(e,i) + d(i)\} = \min\{1+4, 3+5\} = \min\{5, 8\} = 5$$
$$d(f) = \min\{w(f,h) + d(h), w(f,i) + d(i)\} = \min\{2+4, 2+5\} = \min\{6, 7\} = 6$$

第 1 段の点 a，b，c について，$d(a)$，$d(b)$，$d(c)$ を計算する。

$$d(a) = \min\{w(a,d) + d(d), w(a,e) + d(e)\} = \min\{2+6, 5+5\} = \min\{8, 10\} = 8$$
$$d(b) = \min\{w(b,d) + d(d), w(b,e) + d(e)\} = \min\{1+6, 3+5\} = \min\{7, 8\} = 7$$
$$d(c) = \min\{w(c,e) + d(e), w(c,f) + d(f)\} = \min\{4+5, 1+6\} = \min\{9, 7\} = 7$$

最後に, 始点 s について，$d(s)$ を計算する。

$$\begin{aligned} d(s) &= \min\{w(s,a) + d(a), w(s,b) + d(b), w(s,c) + d(c)\} \\ &= \min\{5+8, 6+7, 4+7\} \\ &= \min\{13, 13, 11\} \\ &= 11 \end{aligned}$$

最短路 P^* は，$d(s)$ の計算過程を逆にたどることで求めることができる。点 v に対して, 次の点 u を

$$u = \arg \min_{u' \in \mathcal{P}(v)} \{w(v,u') + d(u')\}$$

と定義する。始点 s から始めて, 次の点を順にたどることで, 最短路 P^* を構築できる。この問題において, 最短路は $P^* = (s, c, f, h, k, t)$ であり, その長さは 11 である。

例 13.2. 動的計画法による最長路問題のアルゴリズムを考え, 例題の最長路を求めよ。点 v から点 t までの最長路の長さ $d(v)$ は,

$$d(v) = \max_{u \in \mathcal{P}(v)} \{w(v,u) + d(u)\}$$

と与えられる。

13.2 0/1 ナップサック問題

0/1 ナップサック問題は, 有限個の品物からなる集合 $I = \{1, 2, \dots, n\}$ と, ナップサックの容量 W が与えられたとき, 以下の最適化問題を解くことである。

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in I} v_i x_i \\ \text{subject to} \quad & \sum_{i \in I} w_i x_i \leq W \\ & x_i \in \{0, 1\} \quad \forall i \in I \end{aligned}$$

ここで, v_i は品物 i の価値, w_i は品物 i の重量, x_i は品物 i をナップサックに入れるかどうかを示す 0/1 変数である。目的は, ナップサックの容量を超えないように品物を選び, その価値の総和を最大化することである。

例 13.3. ここでは, 0/1 ナップサック問題の例題を示す. ナップサックの容量 $W = 15$ のとき, 最適解を求めよ。

品物 i	重量 w_i	価値 v_i
1	12	4
2	2	2
3	1	1
4	1	2

$\mathbf{x} = (x_1, x_2, x_3, x_4)$ とすると, 最適解は $\mathbf{x}^* = (1, 1, 0, 1)$ であり, 価値の総和は 8 である。

品物 $1, 2, \dots, n$ を取るか取らないかを順に決定していく多段階決定過程として 0/1 ナップサック問題を定式化できる。段階 i において, 品物 i を取るか取らないかを決定する。$v_i(w)$ を, 品物 $i, i+1, \dots, n$ から選んで, 利用可能な容量が w のときの最大価値と定義する。すなわち, $v_i(w)$ は, 以下の最適化問題の最適値である。

$$\begin{aligned} \text{maximize} \quad & \sum_{j=i}^{n} v_j x_j \\ \text{subject to} \quad & \sum_{j=i}^{n} w_j x_j \leq w \\ & x_j \in \{0, 1\} \quad \forall j = i, i+1, \dots, n \end{aligned}$$

$\mathbf{x}^* = (x_1, x_2, \dots, x_n)$ は最適解であれば，$(x_i, x_{i+1}, \dots, x_n)$ は，利用可能な容量が $W - \sum_{j=1}^{i-1} w_j x_j$ のときの品物 $i, i+1, \dots, n$ に対する最適解である。これは，もし $(x_i, x_{i+1}, \dots, x_n)$ が最適解でないとすると，より価値の高い解が存在し，$\mathbf{x}^*$ も最適解でなくなるためである。

以上の性質により，$v_i(w)$ は以下の再帰式で与えられる。

$$v_i(w) = \begin{cases} v_{i+1}(w) & \text{if } w_i > w \\ \max\{v_{i+1}(w), v_{i+1}(w - w_i) + v_i\} & \text{if } w_i \leq w \end{cases}$$

また，$n+1$ 段階目では品物が存在しないため，すべての容量 w に対して $v_{n+1}(w) = 0$ となる。

ヒント

$v_{i+1}(w)$ は，すべての $w = 0, 1, \dots, W$ に対して，品物 $i+1, i+2, \dots, n$ から選んで利用可能な容量が w のときの最大価値を表す。これを用いて，$v_i(w), \forall w = 0, 1, \dots, W$ を計算する。最適性の原理により，$v_i(w)$ は部分問題の最適値である。
品物 i の重量 w_i が利用可能な容量 w を超える場合，品物 i は取れないため，$v_i(w) = v_{i+1}(w)$ となる。
$w_i \leq w$ の場合，品物 i を取らない場合と取る場合のうち，価値の高い方を選ぶ．ただし，品物 i を取る場合は，利用可能な容量が $w - w_i$ になり，価値が v_i 増加する．

Algorithm 13.2.

- ***Input**:* 品物の集合 $I = \{1, 2, \dots, n\}$，品物 i の重量 w_i，品物 i の価値 v_i，ナップサックの容量 W

- ***Output**:* 最適値 $v_1(W)$

1. *for all* $w = 0, 1, \dots, W$ *do*
    1. $v_{n+1}(w) \leftarrow 0$
2. *for* $i = n, n-1, \dots, 1$ *do*
    1. *for all* $w = 0, 1, \dots, W$ *do*
        1. *if* $w_i > w$ *then*
            1. $v_i(w) \leftarrow v_{i+1}(w)$
        2. *else*
            1. $v_i(w) \leftarrow \max\{v_{i+1}(w), v_{i+1}(w - w_i) + v_i\}$
3. *return* $v_1(W)$

```python
def dp_knapsack(weights, values, W):
    n = len(weights)
    # Initialize value table
    v = [[0 for _ in range(W + 1)] for _ in range(n + 2)]

    # Fill the table in reverse order
    for i in range(n, 0, -1):
        for w in range(W + 1):
            if weights[i - 1] > w:
                v[i][w] = v[i + 1][w]
            else:
                v[i][w] = max(v[i + 1][w], v[i + 1][w - weights[i - 1]] + values[i - 1])
    return v[1][W]


if __name__ == "__main__":
    weights = [12, 2, 1, 1]
    values = [4, 2, 1, 2]
    W = 15
    optimal_value = dp_knapsack(weights, values, W)
    print(optimal_value)  # Output: 8
```

```
8
```

容量 $W = 2$ のときの計算過程を以下に示す。

$v_5(w) = 0, \forall w$ であり，次に $i = 4$ について，

$$
\begin{aligned}
v_4(0) &= 0 \\
v_4(1) &= \max\{v_5(1), v_5(0) + 2\} = \max\{0, 0 + 2\} = 2 \\
v_4(2) &= \max\{v_5(2), v_5(1) + 2\} = \max\{0, 0 + 2\} = 2
\end{aligned}
$$

が得られる。次に $i = 3$ について，次の結果が得られる。

$$
\begin{aligned}
v_3(0) &= 0 \\
v_3(1) &= \max\{v_4(1), v_4(0)+1\} = \max\{2, 0+1\} = 2 \\
v_3(2) &= \max\{v_4(2), v_4(1)+1\} = \max\{2, 2+1\} = 3
\end{aligned}
$$

次に $i = 2$ について, 次の結果が得られる。

$$
\begin{aligned}
v_2(0) &= 0 \\
v_2(1) &= v_3(1) = 2 \\
v_2(2) &= \max\{v_3(2), v_3(0)+2\} = \max\{3, 0+2\} = 3
\end{aligned}
$$

最後に $i = 1$ について, 次の結果が得られる。

$$
\begin{aligned}
v_1(0) &= 0 \\
v_1(1) &= v_2(1) = 2 \\
v_1(2) &= v_2(2) = 3
\end{aligned}
$$

したがって, 最適値は $v_1(2) = 3$ である。最適解は, 計算過程を逆にたどることで求めることができ, $\mathbf{x}^* = (0, 0, 1, 1)$ となる。

1. $v_1(2) = v_2(2)$ より, 品物 1 は取らない。$x_1^* = 0$。
2. $v_2(2) = v_3(2)$ より, 品物 2 は取らない。$x_2^* = 0$。
3. $v_3(2) = v_4(1) + 1$ より, 品物 3 は取る。$x_3^* = 1$。利用可能な容量は $2 - 1 = 1$ となる。
4. $v_4(1) = v_5(0) + 2$ より, 品物 4 は取る。$x_4^* = 1$。

例 13.4. 容量 $W = 3$ のときの最適値と最適解を求めよ。

13.3　ベルマン方程式（確定的場合）*

動的計画法における最適性の原理は, **ベルマン方程式** (Bellman equation) として知られる再帰方程式により表現される。状態 s におけるの価値関数 $v(s)$ は, 以下のように定義される。

$$
v(s) = \max_{a \in A(s)} \{r(s, a) + \gamma v(s')\}
$$

ここで，$A(s)$ は状態 s における可能な行動の集合，$R(s,a)$ は状態 s で行動 a を取ったときの報酬，γ は割引率，s' は行動 a を取った後の次の状態である。

記号	最短路問題	0/1 ナップサック問題
s	現在の頂点	現在の品物と利用可能な容量
a	次に移動する頂点	品物を取るか取らないか
$A(s)$	現在の頂点から到達可能な頂点の集合	現在の品物を取るか取らないかの選択肢
$R(s,a)$	負の辺の重み	品物の価値
s'	次の頂点	次の品物と新しい利用可能な容量

第VI部

待ち行列理論

第14章　待ち行列とは

14.1　待ち行列モデルの基本

待ち行列モデルを分類するのに，**ケンドールの記号**（Kendall's notation）を用いる。ケンドールの記号は, 次の形式で表される。

$$A/B/C$$

ここで，A は

C はサーバ数を表す。

記号	意味
λ	平均到着率, 単位時間あたりの到着客数の平均
μ	平均サービス率, 単位時間あたりに処理できる客数の平均
ρ	利用率
s	サーバ数

$1/\lambda$ は平均到着間隔，$1/\mu$ は平均サービス時間を表す. 利用率 ρ は, 次の式で定義される.

$$\rho = \frac{\lambda}{s\mu}$$

例 14.1. あるアイスクリーム屋さんでは，1 名の店員が客の注文を受け付け, アイスクリームを提供している. 平均して，1 時間に 30 人の客が来店している. 店員は，1 時間に 60 人の客に対応できる. このとき, 平均到着率 λ は 30 人/時間, 平均サービス率 μ は 60 人/時間である. サーバ数 $s = 1$ であるため, 利用率は

$$\rho = \frac{30}{1 \times 60} = 0.5$$

となる. また，1 時間あたりの到着率が 30 人であるため, 平均到着間隔は $1/\lambda = 1/30$ 時間, 平均サービス時間は $1/\mu = 1/60$ 時間である. これを分単位で表すと, 平均到着間隔は 2 分, 平均サービス時間は 1 分となる.

記号	意味
L	平均系内客数
W	平均滞在時間
L_q	平均待ち行列長
W_q	平均待ち時間

平均系内客数 L は, 待ち行列システム内にいる客数の平均を表す. **平均滞在時間** W は, 客がシステムに到着してサービスを受け, システムを退出するまでにかかる平均時間を表す.

平均待ち行列長 L_q は, 待ち行列にいる客数の平均を表す. **平均待ち時間** W_q は, 客がシステムに到着してからサービスを受けるまでにかかる平均時間を表す.

リトルの法則（Little's Law）により, 次の式が成り立つ.

$$L = \lambda W$$

$$L_q = \lambda W_q$$

さらに, 平均滞在時間 W は, 平均待ち時間 W_q と平均サービス時間 $1/\mu$ の和であるから, 次の式で表される.

$$W = W_q + \frac{1}{\mu}$$

例 14.2. ある銀行では，1 時間に平均 20 人の客が来店し, 各客が銀行に滞在する平均時間は 12 分である. このとき, 平均系内客数 L は

$$L = \lambda W = 20 \times \frac{12}{60} = 4 \text{ 人}$$

となる. つまり, 銀行内には平均 4 人の客がいることになる. 平均滞在時間 W や平均到着率 λ が増加すると, 平均系内客数 L も増加することがわかる.

さらに, 平均サービス時間は 6 分であるとすると, 平均待ち時間 W_q と平均待ち行列長 L_q はそれぞれ

$$W_q = W - \frac{1}{\mu} = 12 - 6 = 6 \text{ 分}$$

$$L_q = \lambda W_q = 20 \times \frac{6}{60} = 2 \text{ 人}$$

となる. つまり, 客が行列で待つ平均時間は 6 分, 1 時間あたり平均 2 人の客が行列で待っていることになる.

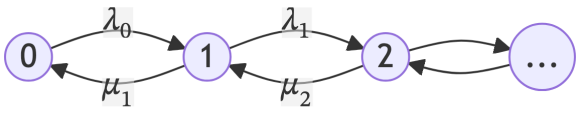
14.1.1 用語集

英語	日本語
service facility	サービス施設
server	サーバ
number of servers	サーバ数
interarrival time	到着間隔
service time	サービス時間

第15章　出生死滅過程

時刻 t における生物集団の個体数を $N(t)$ で表す. これを**状態**（state）とも呼ぶ. 状態は非負の整数であるため，$N(t) \in \{0, 1, 2, ...\}$ となる.

$N(T) = n$ のとき, 次の出生はパラメータ λ_n を持つ指数分布に従い, 次の死滅はパラメータ μ_n を持つ指数分布に従うとする．$n \geq 0$ のとき, 一つの個体が出生すると, 状態は n から $n+1$ に遷移する．$n \geq 1$ のとき, 一つの個体が死滅すると, 状態は n から $n-1$ に遷移する.



第VII部

OR キャリア

第16章　キャリア

16.1　リンク

- Informs
  - Consider a Career in Operations Research and Analytics!
- 日本オペレーションズ・リサーチ学会
  - OR キャリアセッション
  - 企業事例交流会
  - OR 実施事例
- 日本経営工学会
  - 産学連携研究交流会
- キヤノン
  - 日本オペレーションズ・リサーチ(OR)学会レポート活動の舞台裏を大公開！

16.2　関連学会

- 日本オペレーションズ・リサーチ学会
- 日本経営工学会

参考文献

Arrow, Kenneth J, Theodore Harris, と Jacob Marschak. 1951.「Optimal Inventory Policy」. *Econometrica* 19 (3): 250.

Brunelli, Matteo. 2015. *Introduction to the Analytic Hierarchy Process*. 2015th 版. SpringerBriefs in Operations Research. Cham: Springer International Publishing.

Camm, Jeffrey, James Cochran, Michael Fry, Jeffrey Ohlmann, David Anderson, Dennis Sweeney, と Thomas Williams. 2022. *An introduction to management science: Quantitative approaches to decision making*. 16th 版. Florence, AL: South-Western College Publishing.

Crawford, Gordon, と Cindy Williams. 1985.「A note on the analysis of subjective judgment matrices」. *Journal of Mathematical Psychology*, Some thoughts about the mathematics of the analytic hierarchy process, 29 (4): 387–405.

Eiselt, H A, と Carl-Louis Sandblom. 2022. *Operations research: A model-based approach*. Cham: Springer International Publishing.

Harris, Ford W. 1990.「How many parts to make at once」. *Oper. Res.* 38 (6): 947–50.

Hillier, Frederick, と Gerald Lieberman. 2025. *ISE introduction to operations research*. 11th 版. Columbus, OH: McGraw-Hill Education.

Huber, Jakob, Sebastian Müller, Moritz Fleischmann, と Heiner Stuckenschmidt. 2019.「A data-driven newsvendor problem: From data to decision」. *Eur. J. Oper. Res.* 278 (3): 904–15.

Pleguezuelo, Rafael Herrerías, José García Pérez, と Salvador Cruz Rambaud. 2003.「A note on the reasonableness of PERT hypotheses」. *Oper. Res. Lett.* 31 (1): 60–62.

Qin, Yan, Ruoxuan Wang, Asoo J Vakharia, Yuwen Chen, と Michelle M H Seref. 2011.「The newsvendor problem: Review and directions for future research」. *Eur. J. Oper. Res.* 213 (2): 361–74.

Saaty, Thomas L. 1977.「A scaling method for priorities in hierarchical structures」. *J. Math. Psychol.* 15 (3): 234–81.

Scarf, Herbert. 1959.「The optimality of (S, s) policies in the dynamic

inventory problem」.

Snyder, Lawrence V, と Zuo-Jun Max Shen. 2019. *Fundamentals of supply chain theory*. 2nd 版. Nashville, TN: John Wiley & Sons.

付録 A　微分積分

A.1　極値（Extremum）

1 変数関数 $f(x)$ が点 $x = a$ で極値をとるとき、

$$f'(a) = 0$$

が成り立つ。$f''(a) > 0$ のとき、$f(a)$ は極小値をとる。$f''(a) < 0$ のとき、$f(a)$ は極大値をとる。

A.2　凸関数（Convex Function）

1 変数 2 階微分可能な関数 $f(x)$ が凸関数であることの必要十分条件は、すべての x について

$$f''(x) \geq 0$$

が成り立つことである。

凸関数 $f(x)$ の極小値は、最小値である。

付録B　確率

B.1　確率変数

離散型確率変数 X が特定の値 x をとる確率を

$$P(X = x) = p_X(x)$$

と表すとき、$p_X(x)$ を X の**確率質量関数** (PMF) という。

連続型確率変数 X がある区間 $[a, b]$ にある値をとる確率を

$$P(a \leq X \leq b) = \int_a^b f_X(x)dx$$

と表す。$f_X(x)$ を X の**確率密度関数** (PDF) という。

確率変数 X の**累積分布関数** (CDF) を

$$F_X(x) = P(X \leq x) = \begin{cases} \sum_{k \leq x} p_X(k) & \text{if } X \text{ is discrete} \\ \int_{-\infty}^{x} f_X(t)dt & \text{if } X \text{ is continuous} \end{cases}$$

と表す。確率密度関数 $f_X(x)$ は累積分布関数 $F_X(x)$ の微分である。

$$f_X(x) = \frac{d}{dx} F_X(x)$$

B.2　確率分布

B.3　指数分布

連続型確率変数 X は**指数分布**（exponential distribution）に従うとき、$X \sim \mathrm{Exp}(\lambda)$ と表す。ここで $\lambda > 0$ は分布のパラメータである。X の確率密度関数は

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

で与えられる。平均は $E[X] = \frac{1}{\lambda}$、分散は $\mathrm{Var}(X) = \frac{1}{\lambda^2}$ である。

B.3.1 正規分布

連続型確率変数 X は**正規分布**（normal distribution）に従うとき、$X \sim N(\mu, \sigma^2)$ と表す。ここで μ は平均、σ^2 は分散である。X の確率密度関数は

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

と表す。平均は $E[X] = \mu$、分散は $\mathrm{Var}(X) = \sigma^2$ である。

X が $N(\mu, \sigma^2)$ に従うとき、$Y = aX + b$ は、$N(a\mu + b, a^2\sigma^2)$ に従う。特に、$Z = \frac{X-\mu}{\sigma}$ は標準正規分布（standard normal distribution）に従う。すなわち、$Z \sim N(0, 1)$ である。

連続型確率変数 Y が標準正規分布に従うとき、Y の累積分布関数は

$$\Phi(y) = P(Y \leq y) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y} e^{-\frac{t^2}{2}} dt$$

と表す。**標準正規分布表**から、y の値に対する $\Phi(y)$ を調べることができる。

Python では、以下のように $\Phi(y)$ を計算できる。

```
from scipy.stats import norm
def phi(y):
    return norm.cdf(y)

phi(0)  # 0.5
```

また、$\Phi(y) = 0.95$ のときの y の値を求めるには、以下のようにする。

```
from scipy.stats import norm
def phi_inverse(p):
    return norm.ppf(p)

phi_inverse(0.95)  # 約 1.64485
```

正規分布は**再生性**（reproductive property）を持つ。すなわち、$X_1, X_2, \dots, X_n$ が独立に $N(\mu_i, \sigma_i^2)$ に従うとき、$Y = \sum_{i=1}^{n} a_i X_i$ は $N\left(\sum_{i=1}^{n} a_i \mu_i, \sum_{i=1}^{n} a_i^2 \sigma_i^2\right)$ に従う。

付録C　標準正規分布表

下表は、標準正規分布 $N(0,1)$ の累積分布関数

$$\phi(z) = P(Z \leq z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{t^2}{2}} dt$$

の値を示すものである。

例 C.1. $z = 1.23$ のとき、$\phi(1.23)$ の値を求めよ。

表の行から 1.2、列から $+0.03$ を選ぶ。交差する値は 0.89065 である。したがって、$\phi(1.23) = 0.89065$ である。

例 C.2. $\phi(z) = 0.89065$ のとき、z の値を求めよ。

表の中から 0.89065 を探す。行から 1.2、列から $+0.03$ を選ぶ。したがって、$z = 1.23$ である。

例 C.3. $X \sim N(100, 15^2)$ のとき、$P(X \leq 120)$ の値を求めよ。

$$\begin{aligned} P(X \leq 120) &= \phi\left(\frac{120-100}{15}\right) && \text{(C.1)} \\ &= \phi\left(\frac{20}{15}\right) && \text{(C.2)} \\ &\approx \phi(1.33) && \text{(C.3)} \\ & && \text{(C.4)} \end{aligned}$$

表から $\phi(1.33) \approx 0.90824$ である。

例 C.4. $X \sim N(100, 15^2)$ のとき、$P(X \leq x) = 0.9$ のとき、x の値を求めよ。

$$P(X \leq x) = 0.9 \tag{C.5}$$

$$\phi\left(\frac{x-100}{15}\right) = 0.9 \tag{C.6}$$

$$\frac{x-100}{15} \approx 1.28 \tag{C.7}$$

$$x - 100 \approx 19.2 \tag{C.8}$$

$$x \approx 119.2 \tag{C.9}$$

$$\tag{C.10}$$

したがって、$x \approx 119.2$ である。

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
-3.9	0.00005	0.00005	0.00004	0.00004	0.00004	0.00004	0.00004	0.00004	0.00003	0.00003
-3.8	0.00007	0.00007	0.00007	0.00006	0.00006	0.00006	0.00006	0.00005	0.00005	0.00005
-3.7	0.00011	0.00010	0.00010	0.00010	0.00009	0.00009	0.00008	0.00008	0.00008	0.00008
-3.6	0.00016	0.00015	0.00015	0.00014	0.00014	0.00013	0.00013	0.00012	0.00012	0.00011
-3.5	0.00023	0.00022	0.00022	0.00021	0.00020	0.00019	0.00019	0.00018	0.00017	0.00017
-3.4	0.00034	0.00032	0.00031	0.00030	0.00029	0.00028	0.00027	0.00026	0.00025	0.00024
-3.3	0.00048	0.00047	0.00045	0.00043	0.00042	0.00040	0.00039	0.00038	0.00036	0.00035
-3.2	0.00069	0.00066	0.00064	0.00062	0.00060	0.00058	0.00056	0.00054	0.00052	0.00050
-3.1	0.00097	0.00094	0.00090	0.00087	0.00084	0.00082	0.00079	0.00076	0.00074	0.00071
-3.0	0.00135	0.00131	0.00126	0.00122	0.00118	0.00114	0.00111	0.00107	0.00104	0.00100
-2.9	0.00187	0.00181	0.00175	0.00169	0.00164	0.00159	0.00154	0.00149	0.00144	0.00139
-2.8	0.00256	0.00248	0.00240	0.00233	0.00226	0.00219	0.00212	0.00205	0.00199	0.00193
-2.7	0.00347	0.00336	0.00326	0.00317	0.00307	0.00298	0.00289	0.00280	0.00272	0.00264
-2.6	0.00466	0.00453	0.00440	0.00427	0.00415	0.00402	0.00391	0.00379	0.00368	0.00357
-2.5	0.00621	0.00604	0.00587	0.00570	0.00554	0.00539	0.00523	0.00508	0.00494	0.00480
-2.4	0.00820	0.00798	0.00776	0.00755	0.00734	0.00714	0.00695	0.00676	0.00657	0.00639
-2.3	0.01072	0.01044	0.01017	0.00990	0.00964	0.00939	0.00914	0.00889	0.00866	0.00842
-2.2	0.01390	0.01355	0.01321	0.01287	0.01255	0.01222	0.01191	0.01160	0.01130	0.01101
-2.1	0.01786	0.01743	0.01700	0.01659	0.01618	0.01578	0.01539	0.01500	0.01463	0.01426
-2.0	0.02275	0.02222	0.02169	0.02118	0.02068	0.02018	0.01970	0.01923	0.01876	0.01831
-1.9	0.02872	0.02807	0.02743	0.02680	0.02619	0.02559	0.02500	0.02442	0.02385	0.02330
-1.8	0.03593	0.03515	0.03438	0.03362	0.03288	0.03216	0.03144	0.03074	0.03005	0.02938
-1.7	0.04457	0.04363	0.04272	0.04182	0.04093	0.04006	0.03920	0.03836	0.03754	0.03673
-1.6	0.05480	0.05370	0.05262	0.05155	0.05050	0.04947	0.04846	0.04746	0.04648	0.04551
-1.5	0.06681	0.06552	0.06426	0.06301	0.06178	0.06057	0.05938	0.05821	0.05705	0.05592

z	−0.00	−0.01	−0.02	−0.03	−0.04	−0.05	−0.06	−0.07	−0.08	−0.09
-1.4	0.08076	0.07927	0.07780	0.07636	0.07493	0.07353	0.07215	0.07078	0.06944	0.06811
-1.3	0.09680	0.09510	0.09342	0.09176	0.09012	0.08851	0.08692	0.08534	0.08379	0.08226
-1.2	0.11507	0.11314	0.11123	0.10935	0.10749	0.10565	0.10383	0.10204	0.10027	0.09853
-1.1	0.13567	0.13350	0.13136	0.12924	0.12714	0.12507	0.12302	0.12100	0.11900	0.11702
-1.0	0.15866	0.15625	0.15386	0.15151	0.14917	0.14686	0.14457	0.14231	0.14007	0.13786
-0.9	0.18406	0.18141	0.17879	0.17619	0.17361	0.17106	0.16853	0.16602	0.16354	0.16109
-0.8	0.21186	0.20897	0.20611	0.20327	0.20045	0.19766	0.19489	0.19215	0.18943	0.18673
-0.7	0.24196	0.23885	0.23576	0.23270	0.22965	0.22663	0.22363	0.22065	0.21770	0.21476
-0.6	0.27425	0.27093	0.26763	0.26435	0.26109	0.25785	0.25463	0.25143	0.24825	0.24510
-0.5	0.30854	0.30503	0.30153	0.29806	0.29460	0.29116	0.28774	0.28434	0.28096	0.27760
-0.4	0.34458	0.34090	0.33724	0.33360	0.32997	0.32636	0.32276	0.31918	0.31561	0.31207
-0.3	0.38209	0.37828	0.37448	0.37070	0.36693	0.36317	0.35942	0.35569	0.35197	0.34827
-0.2	0.42074	0.41683	0.41294	0.40905	0.40517	0.40129	0.39743	0.39358	0.38974	0.38591
-0.1	0.46017	0.45620	0.45224	0.44828	0.44433	0.44038	0.43644	0.43251	0.42858	0.42465
-0.0	0.50000	0.49601	0.49202	0.48803	0.48405	0.48006	0.47608	0.47210	0.46812	0.46414

z	+0.00	+0.01	+0.02	+0.03	+0.04	+0.05	+0.06	+0.07	+0.08	+0.09
0.0	0.50000	0.50399	0.50798	0.51197	0.51595	0.51994	0.52392	0.52790	0.53188	0.53586
0.1	0.53983	0.54380	0.54776	0.55172	0.55567	0.55962	0.56360	0.56749	0.57142	0.57535
0.2	0.57926	0.58317	0.58706	0.59095	0.59483	0.59871	0.60257	0.60642	0.61026	0.61409
0.3	0.61791	0.62172	0.62552	0.62930	0.63307	0.63683	0.64058	0.64431	0.64803	0.65173
0.4	0.65542	0.65910	0.66276	0.66640	0.67003	0.67364	0.67724	0.68082	0.68439	0.68793
0.5	0.69146	0.69497	0.69847	0.70194	0.70540	0.70884	0.71226	0.71566	0.71904	0.72240
0.6	0.72575	0.72907	0.73237	0.73565	0.73891	0.74215	0.74537	0.74857	0.75175	0.75490
0.7	0.75804	0.76115	0.76424	0.76730	0.77035	0.77337	0.77637	0.77935	0.78230	0.78524
0.8	0.78814	0.79103	0.79389	0.79673	0.79955	0.80234	0.80511	0.80785	0.81057	0.81327
0.9	0.81594	0.81859	0.82121	0.82381	0.82639	0.82894	0.83147	0.83398	0.83646	0.83891
1.0	0.84134	0.84375	0.84614	0.84849	0.85083	0.85314	0.85543	0.85769	0.85993	0.86214
1.1	0.86433	0.86650	0.86864	0.87076	0.87286	0.87493	0.87698	0.87900	0.88100	0.88298
1.2	0.88493	0.88686	0.88877	0.89065	0.89251	0.89435	0.89617	0.89796	0.89973	0.90147
1.3	0.90320	0.90490	0.90658	0.90824	0.90988	0.91149	0.91308	0.91466	0.91621	0.91774
1.4	0.91924	0.92073	0.92220	0.92364	0.92507	0.92647	0.92785	0.92922	0.93056	0.93189
1.5	0.93319	0.93448	0.93574	0.93699	0.93822	0.93943	0.94062	0.94179	0.94295	0.94408
1.6	0.94520	0.94630	0.94738	0.94845	0.94950	0.95053	0.95154	0.95254	0.95352	0.95449
1.7	0.95543	0.95637	0.95728	0.95818	0.95907	0.95994	0.96080	0.96164	0.96246	0.96327
1.8	0.96407	0.96485	0.96562	0.96638	0.96712	0.96784	0.96856	0.96926	0.96995	0.97062

z	+0.00	+0.01	+0.02	+0.03	+0.04	+0.05	+0.06	+0.07	+0.08	+0.09
1.9	0.97128	0.97193	0.97257	0.97320	0.97381	0.97441	0.97500	0.97558	0.97615	0.97670
2.0	0.97725	0.97778	0.97831	0.97882	0.97932	0.97982	0.98030	0.98077	0.98124	0.98169
2.1	0.98214	0.98257	0.98300	0.98341	0.98382	0.98422	0.98461	0.98500	0.98537	0.98574
2.2	0.98610	0.98645	0.98679	0.98713	0.98745	0.98778	0.98809	0.98840	0.98870	0.98899
2.3	0.98928	0.98956	0.98983	0.99010	0.99036	0.99061	0.99086	0.99111	0.99134	0.99158
2.4	0.99180	0.99202	0.99224	0.99245	0.99266	0.99286	0.99305	0.99324	0.99343	0.99361
2.5	0.99379	0.99396	0.99413	0.99430	0.99446	0.99461	0.99477	0.99492	0.99506	0.99520
2.6	0.99534	0.99547	0.99560	0.99573	0.99585	0.99598	0.99609	0.99621	0.99632	0.99643
2.7	0.99653	0.99664	0.99674	0.99683	0.99693	0.99702	0.99711	0.99720	0.99728	0.99736
2.8	0.99744	0.99752	0.99760	0.99767	0.99774	0.99781	0.99788	0.99795	0.99801	0.99807
2.9	0.99813	0.99819	0.99825	0.99831	0.99836	0.99841	0.99846	0.99851	0.99856	0.99861
3.0	0.99865	0.99869	0.99874	0.99878	0.99882	0.99886	0.99889	0.99893	0.99896	0.99900
3.1	0.99903	0.99906	0.99910	0.99913	0.99916	0.99918	0.99921	0.99924	0.99926	0.99929
3.2	0.99931	0.99934	0.99936	0.99938	0.99940	0.99942	0.99944	0.99946	0.99948	0.99950
3.3	0.99952	0.99953	0.99955	0.99957	0.99958	0.99960	0.99961	0.99962	0.99964	0.99965
3.4	0.99966	0.99968	0.99969	0.99970	0.99971	0.99972	0.99973	0.99974	0.99975	0.99976
3.5	0.99977	0.99978	0.99978	0.99979	0.99980	0.99981	0.99981	0.99982	0.99983	0.99983
3.6	0.99984	0.99985	0.99985	0.99986	0.99986	0.99987	0.99987	0.99988	0.99988	0.99989
3.7	0.99989	0.99990	0.99990	0.99990	0.99991	0.99991	0.99992	0.99992	0.99992	0.99992
3.8	0.99993	0.99993	0.99993	0.99994	0.99994	0.99994	0.99994	0.99995	0.99995	0.99995
3.9	0.99995	0.99995	0.99996	0.99996	0.99996	0.99996	0.99996	0.99996	0.99997	0.99997

付録D　線形代数

D.1　固有値と固有ベクトル

$n \times n$ 行列 $\mathbf{A}$ に対し, ベクトル $\mathbf{x} \neq \mathbf{0}$ とスカラー λ が存在して,

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

が成り立つとき，$\mathbf{x}$ を $\mathbf{A}$ の**固有ベクトル** (eigenvector)，λ を $\mathbf{A}$ の**固有値** (eigenvalue) と呼ぶ.

例 D.1. 次のような $\mathbf{A}, \mathbf{x}, \lambda$ を考える.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 1/2 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \lambda = 2$$

このとき,

$$\begin{bmatrix} 1 & 2 \\ 1/2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix} = 2 \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

が成り立つため，$\mathbf{x}$ は $\mathbf{A}$ の固有ベクトル，λ は $\mathbf{A}$ の固有値である.

$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ は単位行列 $\mathbf{I}$ を用いて $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$ と書ける. このとき，$\mathbf{x} \neq \mathbf{0}$ であるため，$\det(\mathbf{A} - \lambda\mathbf{I}) = 0$ が成り立つ.

例 D.2. 次のような行列 $\mathbf{A}$ を考える.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 1/2 & 1 \end{bmatrix}$$

このとき，$\det(\mathbf{A} - \lambda\mathbf{I})$ は以下のようになる.

$$\begin{aligned}\det(\mathbf{A}-\lambda\mathbf{I}) &= \det\begin{bmatrix}1-\lambda & 2\\ 1/2 & 1-\lambda\end{bmatrix}\\ &= (1-\lambda)^2-1\\ &= \lambda^2-2\lambda\\ &= \lambda(\lambda-2)\end{aligned}$$

$\lambda(\lambda-2)=0$ より，$\lambda=0,2$ が得られる．

付録E　最適化問題