オペレーションズ・リサーチ

Operations Research: Models, Algorithms, and Implementations

劉子昂

2025-09-30

目次

Preface

OR の基礎と応用を学ぶための教科書です.

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

出席人数が多い場合は、出席を取らないことがあります。

WebClass で出席を取ります。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。

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

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 [1]。
> 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？

---

[1] 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 9.1 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。

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。

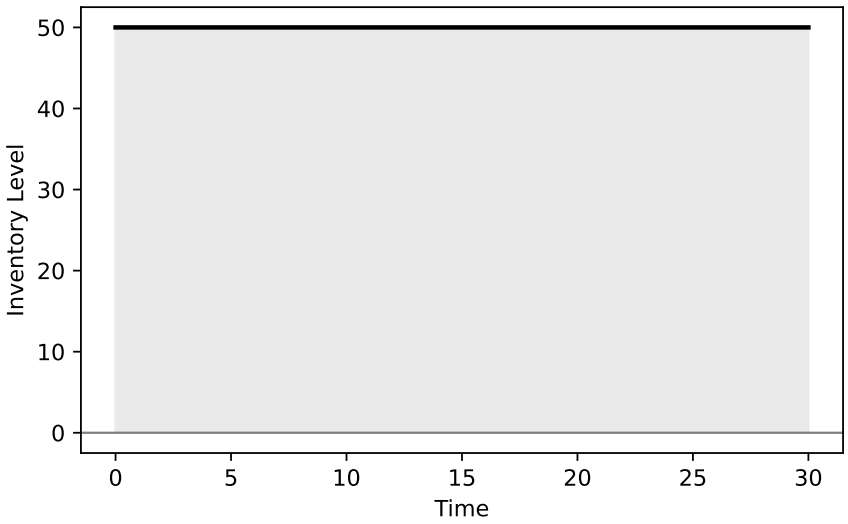
保管費用は次のように計算される。

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

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。

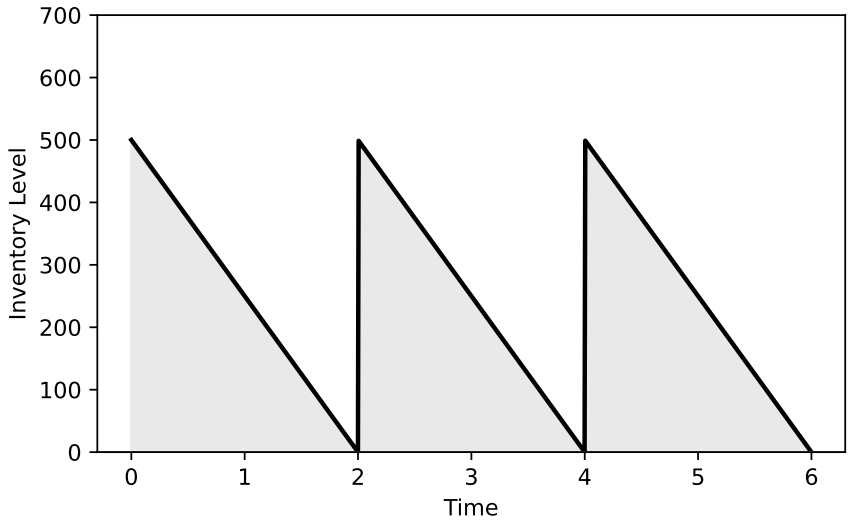
$$\text{保管費用} = \text{面積} \times h$$

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) 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、発注費用 K、保管費用 h と、購入費用（購入単価を c と表す）がある。

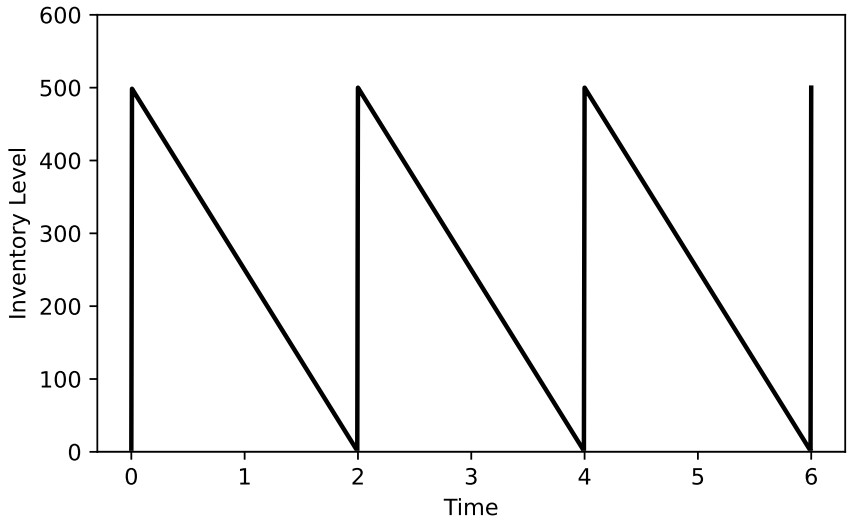
EOQ モデルの最適解は次の二つの性質を持つ (Snyder と Shen 2019)：

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 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：在庫量はサイクル期間 T の間に Q 個から 0 個まで減少するため、平均在庫量は $\frac{Q}{2}$ である。したがって、平均保管費用は $\frac{hQ}{2}$ である。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

$$\frac{hQ}{2} \cdot T = \frac{hQ^2}{2d}$$

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。

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

2.2 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。

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

注釈 2.1. 以下の性質がわかる。

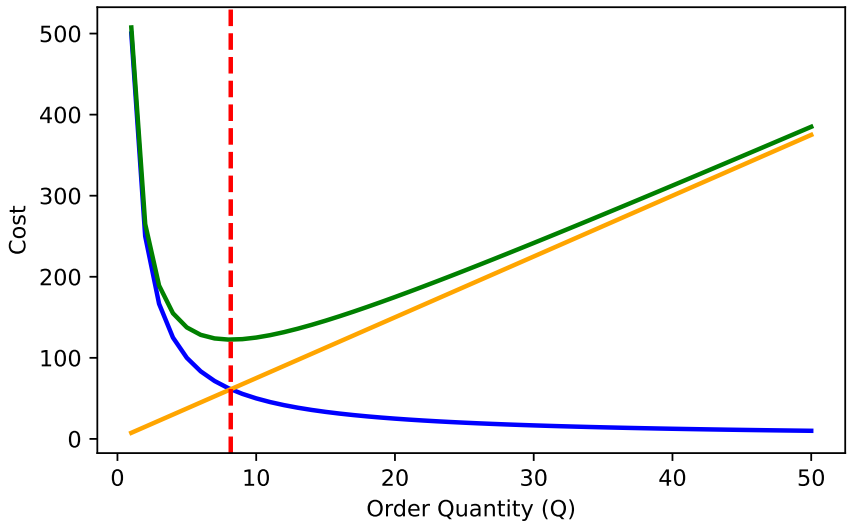
1. Q^* は c には依存しない。
2. h の増加に伴い、Q^* は減少する。保管費用が高い場合は、少量で高い頻度で発注することが望ましい。
3. K の増加に伴い、Q^* は増加する。発注費用が高い場合は、多量で低い頻度で発注することが望ましい。

Algorithm for EOQ Model

1. Input: K, d, h.
2. Compute Q^* using 式 2.1.
3. Compute T^* using 式 2.2.

次の図は、発注コスト、保管コスト、平均コストの関係を示している。購入単価を $c = 0$ とする。

```
# Parameters
K = 500  # Order cost
h = 15  # Holding cost
c = 0  # Purchase cost
Q = np.linspace(1, 50, 50)

# Average cost function
g = (K / Q) + c + (h * Q / 2)

# Plotting the costs vs order quantity
plt.plot(Q, K / Q, label="Order Cost", color="blue", linewidth=2)
plt.plot(Q, c + (h * Q / 2), label="Holding Cost", color="orange", linewidth=2)
plt.plot(Q, g, label="Average Cost", color="green", linewidth=2)
plt.axvline(
    x=np.sqrt(2 * K / h),
    color="red",
    linestyle="--",
    label="Optimal Order Quantity",
    linewidth=2,
)
plt.xlabel("Order Quantity (Q)")
plt.ylabel("Cost")
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)
    print(f"Optimal Order Quantity (Q*): {Q_star:.2f}")
```

```
Optimal Order Quantity (Q*): 129.10
```

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。

2.3 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。

2.4　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)

2.5　練習問題

1. B 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。
2. 需要率を d、発注費用を K、保管費用を h, 購入単価を c とする。EOQ モデルにおける平均コスト $g(Q)$ を発注量 Q の関数として導出せよ。
3. $g(Q)$ を最小化する発注量 Q^* を導出せよ。なぜ Q^* が最小値を与えるのか説明せよ。
4. 需要率 $d = 300$、発注費用 $K = 4000$、保管費用 $h = 20$ のとき、最適発注量 Q^* とサイクル期間 T^* を計算せよ。

2.6　実装

1. Python で EOQ モデルの最適発注量を計算する関数 `eoq(K, d, h)` を実装せよ。

第3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。

$$\text{minimize} \quad \sum_{t=1}^{T}(Ky_t + hx_t) \tag{3.1}$$

$$\text{subject to} \quad x_t = x_{t-1} + q_t - d_t \qquad \forall t \in \mathcal{T} \tag{3.2}$$

$$q_t \geq 0 \qquad \forall t \in \mathcal{T} \tag{3.3}$$

$$q_t \leq My_t \qquad \forall t \in \mathcal{T} \tag{3.4}$$

$$x_t \geq 0 \qquad \forall t \in \mathcal{T} \tag{3.5}$$

$$y_t \in \{0, 1\} \qquad \forall t \in \mathcal{T} \tag{3.6}$$

第4章　新聞売り子問題

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 を確率変数とし、確率密度関数を $f_D(d)$、累積分布関数を $F_D(d)$ とする。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。

記号	意味
h	在庫超過費用
p	在庫不足費用
D	需要(確率変数)
d	需要の観測値

4.1　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。

$$g(S,d) = h(S-d)^+ + p(d-S)^+$$

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。

$$
\begin{aligned}
g(S) &= \mathbb{E}[g(S, D)] && (4.1)\\
&= h\mathbb{E}[(S-D)^+] + p\mathbb{E}[(D-S)^+] && (4.2)\\
&= h\int_0^\infty (S-d)^+ f_D(d)dd + p\int_0^\infty (d-S)^+ f_D(d)dd && (4.3)\\
&= h\int_0^S (S-d) f_D(d)dd + p\int_S^\infty (d-S) f_D(d)dd && (4.4)
\end{aligned}
$$

4.2　最適化

$g(S)$ の 1 階微分は以下のように求める。

$$
\begin{aligned}
\frac{dg(S)}{dS} &= h\int_0^S f_D(d)dd - p\int_S^\infty f_D(d)dd && (4.5)\\
&= hF_D(S) - p(1-F_D(S)) && (4.6)\\
& && (4.7)
\end{aligned}
$$

> **ノート**
>
> $D \leq S$ である確率は、需要 D の累積分布関数 $F_D(S)$ で与えられる。
>
> $$P(D \leq S) = F_D(S) = \int_0^S f_D(d)dd$$
>
> また、$D > S$ である確率は $1 - F_D(S)$ で与えられる。
>
> $$P(D > S) = 1 - F_D(S) = \int_S^\infty f_D(d)dd$$

よって、$dg(S)/dS = 0$ から、

$$
\begin{aligned}
hF_D(S) - p(1-F_D(S)) &= 0 && (4.8)\\
F_D(S) &= \frac{p}{h+p} && (4.9)
\end{aligned}
$$

になる。2 階微分は

$$\frac{d^2g(S)}{dS^2} = hf_D(S) + pf_D(S) \tag{4.10}$$
$$= (h+p)f_D(S) \tag{4.11}$$

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

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

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。

定理 4.1. 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。

4.3 臨界率

$p/(h+p)$ は**臨界率**（critical ratio）と呼ばれる。臨界率は、在庫不足費用と在庫超過費用の比率を表す。

ここで、$F_D(S) = P(D \leq S)$ は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。したがって、最適発注量 S^* を用いることは、サービスレベルを $p/(h+p)$ に等しくすることを意味する。これは、

$$1 - \frac{p}{h+p} = \frac{h}{h+p}$$

の確率で欠品が発生することが最適であることがわかる。

注釈 4.1.

- 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。

4.4　例題

例 4.1. 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。

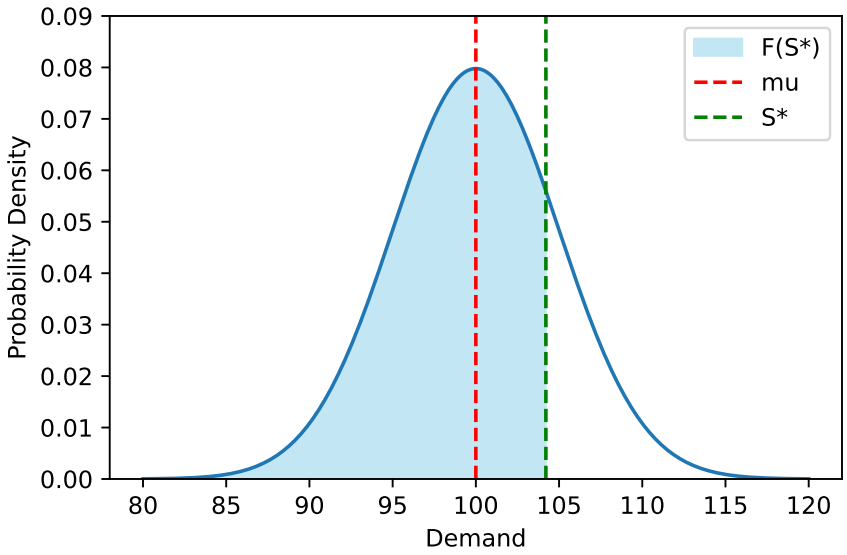
定理 4.1 より、S^* は以下の式で与えられる。

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

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。

4.4.1 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)
```

4.4.2 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
```

4.4.3　標準正規分布表

新聞売り子問題の最適発注量 S^* は、以下の式を満たす。

$$F_D(S^*) = \frac{p}{h+p}$$

ここで、D は正規分布 $N(\mu, \sigma^2)$ に従うため、

$$F_D(S^*) = P(D \leq S^*) = \Phi\left(\frac{S^* - \mu}{\sigma}\right)$$

である。したがって、S^* は以下の式を満たす。

$$\Phi\left(\frac{S^* - 100}{5}\right) = \frac{p}{h+p}$$

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^* は以下の式を満たす。

$$\frac{S^* - 100}{5} \approx 0.84$$

これを解くと、$S^* \approx 104.2$ となる。

4.5 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。

4.5.1 コスト関数

$$\pi(S,d) = r\min\{d,S\} - cS + v\max\{0,S-d\} \tag{4.12}$$
$$- h\max\{0,S-d\} - p\max\{0,d-S\} \tag{4.13}$$

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。

$$\begin{aligned}
g(S) &= \mathbb{E}[g(S,D)] && (4.14)\\
&= \int_0^\infty g(S,d) f_D(d) dd && (4.15)\\
&= cS - r\mathbb{E}[D] + (h-v)\mathbb{E}[(S-D)^+] + (p+r)\mathbb{E}[(D-S)^+] && (4.16)\\
&= cS - r\mu + (h-v)\int_0^\infty (S-d)^+ f_D(d) dd + (p+r)\int_0^\infty (d-S)^+ f_D(d) dd && (4.17)\\
&= cS - r\mu + (h-v)\int_0^S (S-d) f_D(d) dd + (p+r)\int_S^\infty (d-S) f_D(d) dd && (4.18)
\end{aligned}$$

4.5.2　最適化

$g(S)$ の 1 階微分は以下のように求める。

$$\begin{aligned}
\frac{dg(S)}{dS} &= c + (h-v)F_D(S) - (p+r)(1-F_D(S)) && (4.19)\\
& && (4.20)
\end{aligned}$$

よって、$dg(S)/dS = 0$ から、

$$\begin{aligned}
c + (h-v)F_D(S) - (p+r)(1-F_D(S)) &= 0 && (4.21)\\
F_D(S) &= \frac{p+r-c}{h+p+r-v} && (4.22)
\end{aligned}$$

になる。2 階微分は

$$\begin{aligned}
\frac{d^2 g(S)}{dS^2} &= (h-v) f_D(S) + (p+r) f_D(S) && (4.23)\\
&= (h-v+p+r) f_D(S) && (4.24)
\end{aligned}$$

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。

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

4.5.3 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。

4.6 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。

4.7　発注費用を考慮した新聞売り子問題

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。

4.8　練習問題

1. 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 $p = 40$ のとき、コスト $g(S, d)$ を求めよ。
2. 需要 D が正規分布 $N(150, 36)$ に従う新聞売り子問題を考える。在庫超過費用が $h = 5$、在庫不足費用が $p = 20$ のとき、最適発注量 S^* を求めよ。

第5章　安全在庫

これまで紹介した在庫モデルは、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。

5.1　問題設定

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。

(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、発注点 r と発注量 Q を決定変数とし、在庫の**期待コスト**（expected cost）を最小化することを目的とする。

> **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。

5.2　近似解法

以下では、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。ここで、μ は平均需要、σ は需要の標準偏差である。

5.2.1　発注量 Q

D の平均 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)。

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

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。

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。

5.2.2　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。

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。

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

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

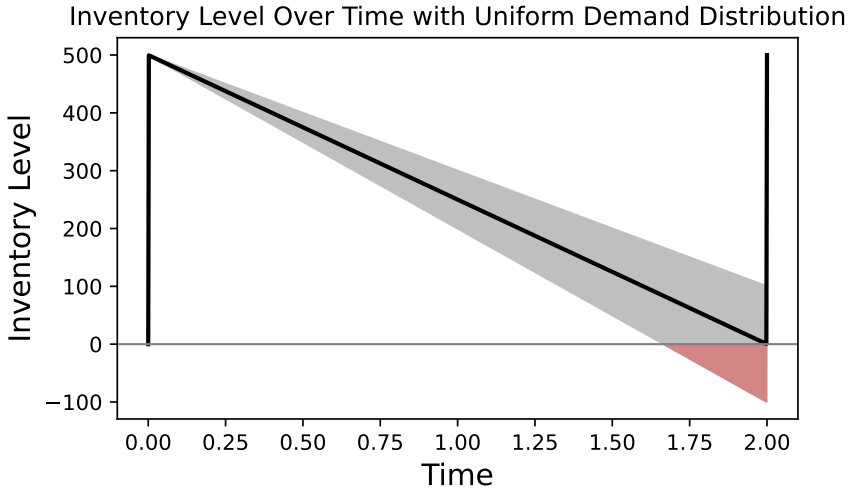
> **ノート**
>
> $$P(D_L \le \mu_L) = P\left(\frac{D_L - \mu_L}{\sigma_L} \le 0\right) = \Phi(0) = 0.5$$

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。それを**安全在庫**（safety stock）と呼び、s と表す。

例 5.1. 単位期間あたりの需要 D が連続一様分布 $U(200, 300)$ に従うと仮定する。平均需要は $\mu = 250$ である。一回の発注量を $Q = 500$ とする。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

# 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()
```



よって、発注点 r は次のように表される。

$$r = \mu_L + s$$

従って、$P(D_L \leq r) = \alpha$ は次のように表される。

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

この式を変形すると、

$$P(D_L - \mu_L \leq s) = \alpha \tag{5.1}$$

$$P\left(\frac{D_L - \mu_L}{\sigma_L} \leq \frac{s}{\sigma_L}\right) = \alpha \tag{5.2}$$

$$\Phi\left(\frac{s}{\sigma_L}\right) = \alpha \tag{5.3}$$

$$\frac{s}{\sigma_L} = \Phi^{-1}(\alpha) \tag{5.4}$$

$$s = \sigma_L \Phi^{-1}(\alpha) \tag{5.5}$$

$$s = \sigma\sqrt{L}\Phi^{-1}(\alpha) \tag{5.6}$$

ここで、$\Phi(\cdot)$ は標準正規分布の累計分布関数であり、$\Phi^{-1}(\alpha)$ 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 5.2. 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。

$$\mu_L = \mu L = 100 \cdot 4 = 400 \tag{5.7}$$
$$\sigma_L = \sigma\sqrt{L} = 20\sqrt{4} = 40 \tag{5.8}$$
$$\tag{5.9}$$

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。

$$s = \sigma_L \Phi^{-1}(0.95) \approx 40 \cdot 1.64485 \approx 65.79 \tag{5.10}$$
$$r = \mu_L + s \approx 400 + 65.79 \approx 465.79 \tag{5.11}$$

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
```

第II部

多基準意思決定分析

第6章　階層分析法

第III部

ネットワーク最適化

第7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。

7.1　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。

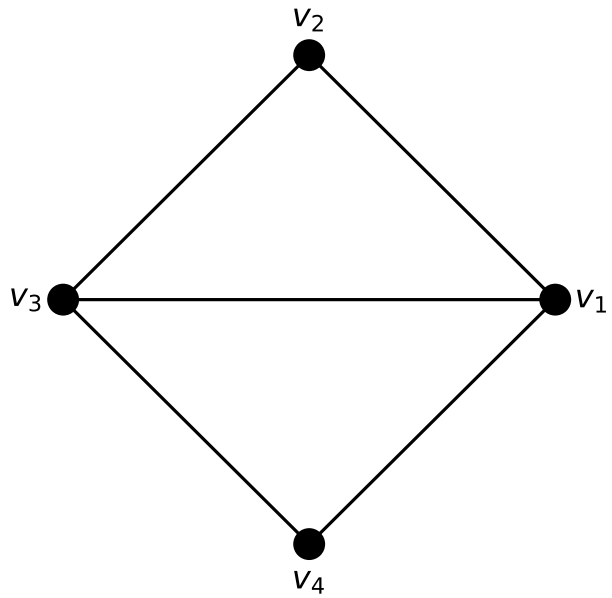
例 7.1 (無向グラフの例)**.** グラフ

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

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 7.1 の場合、v_1 の次数は 3、v_2 の次数は 2 である。

7.2 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。

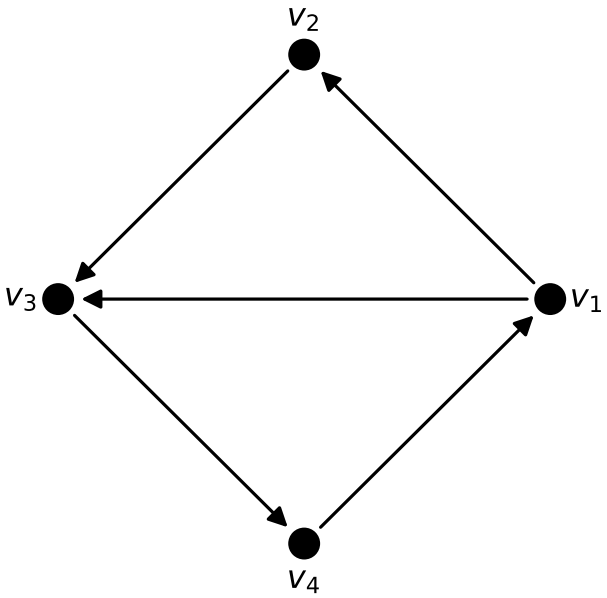
例 7.2 (有向グラフの例)**.** グラフ

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

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 7.2 の場合、v_1 の出次数は 2、入次数は 1 である。

7.3　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。

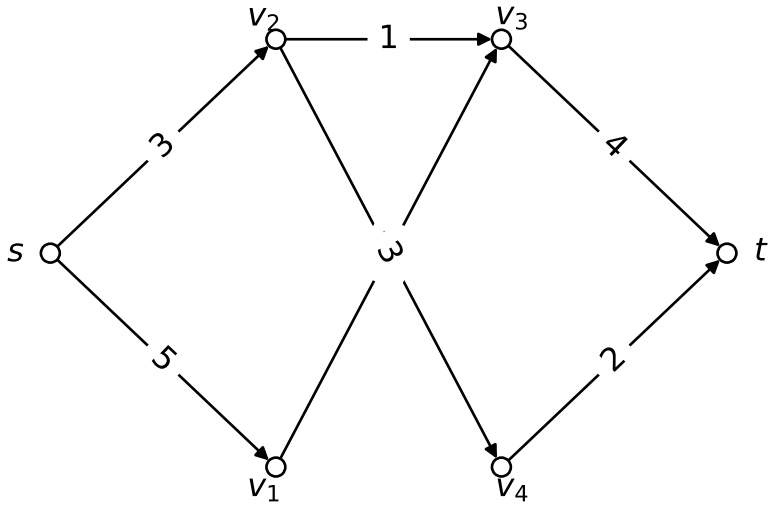
第8章　最短路問題

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 8.1. 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_2
v_3
1
3
4
s
3
t
5
2
v_1
v_4

第9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)。

9.1　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 9.1. 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

9.2 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。

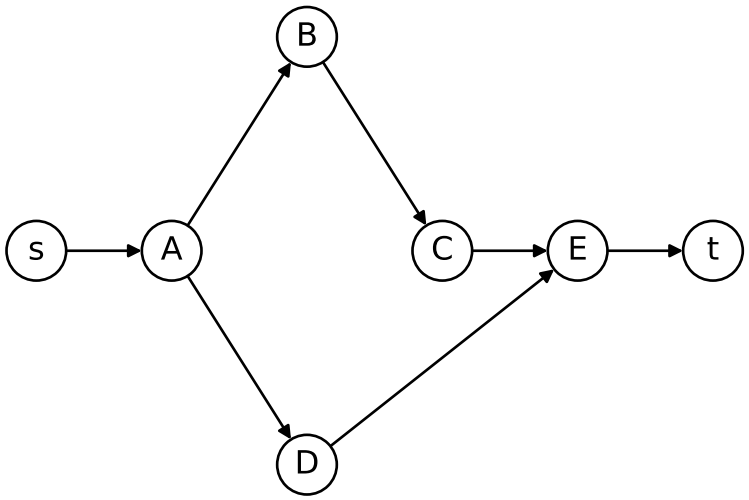
辺 $(s, v) \in E$ は、作業 v が先行作業を持たないことを意味し、辺 $(v, t) \in E$ は、作業 v が後続作業を持たないことを意味する。作業 v の先行作業の集合を $\mathcal{P}(v)$ と表す。作業 v の後続作業の集合を $\mathcal{S}(v)$ と表す。

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 9.1 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()
```

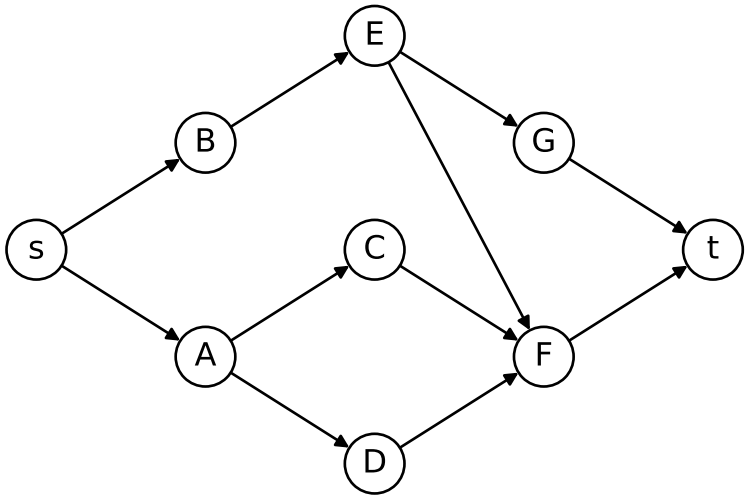


例 9.2. 以下の作業リストに基づいて、プロジェクト・ネットワークを作成せよ。

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()
```



9.3 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。

9.3.1 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。

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。

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 9.2 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。

9.3.2 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 9.2 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

9.3.3 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。スラックが 0 の作業はクリティカル作業である。

例 9.2 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

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。

9.4 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 [1]。

$$\hat{\mu} = \frac{a + 4m + b}{6}, \quad \hat{\sigma}^2 = \left(\frac{b - a}{6}\right)^2 \tag{9.1}$$

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 9.2 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

[1] 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 9.1 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。

v	a	m	b	μ	σ^2
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

9.5 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。

プロジェクト・ネットワークにおけるパス $P = (s, v_{i_1}, v_{i_2}, \ldots, v_{i_k}, t)$ 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 9.3 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[2]。

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

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。

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

[2] 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。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とに注意されたい。

例 9.2 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。

9.6 時間とコストのトレードオフ

作業時間を短縮するには、コストがかかる。最小のコストでプロジェクトを短縮する問題を考える。

9.6.1 問題の定式化

- 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
- y_i: 作業 i の開始時間
- x_i: 作業 i の短縮時間

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

9.6.2 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", "t")]

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。

参考文献

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.

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.

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.

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　1 変数関数の極値

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　凸関数の性質

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、$f''(x) \geq 0$ が成り立つことである。

$f''(x) > 0$ のとき、$f(x)$ は狭義凸関数である。

凸関数 $f(x)$ の極小値は、最小値である。

付録B　確率

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.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
```

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。

$$P(X \leq x) = P\left(\frac{X-\mu}{\sigma} \leq \frac{x-\mu}{\sigma}\right) \tag{B.1}$$

$$\tag{B.2}$$

$$= P\left(Y \leq \frac{x-\mu}{\sigma}\right) \tag{B.3}$$

$$\tag{B.4}$$

$$= \Phi\left(\frac{x-\mu}{\sigma}\right) \tag{B.5}$$

正規分布は**再生性** (reproductive property)を持つ。すなわち、$X_1, X_2, \ldots, X_n$ 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　Optimization Problems