多次元累積和の一般化とその計算効率に関する研究

Generalization of Multi-Dimensional Cumulative Sums and Their Computational Efficiency

千田研究室　後藤柊矢
Chida Laboratory　Shuya GOTO

令和 6 年 xx 月 xx 日

Abstract

This study focuses on the formalization of the theory of multidimensional cumulative sums and the development of efficient computation methods utilizing this theory. While cumulative sums are widely used to quickly compute range sums in one- and two-dimensional data arrays, research on extending this approach to higher-dimensional data and improving its computational efficiency remains insufficient. Building upon existing theories for one- and two-dimensional cumulative sums, this study proposes a generalized method that can be extended to arbitrary dimensions D.

The proposed method constructs an algorithm that achieves a computational complexity of $O(N)$ for preprocessing cumulative sums with data size N and $O(2^D)$ for query processing. By applying the principle of inclusion-exclusion, the method efficiently computes the total sum over arbitrary subregions in a multidimensional space. Additionally, leveraging bit representations for indices optimizes set operations in higher dimensions, simplifying the implementation process.

The findings of this study provide valuable solutions to both academic and practical challenges aimed at the efficient analysis of high-dimensional data. In particular, the generalization of multidimensional cumulative sums and insights into improving their computational efficiency are expected to contribute significantly to the advancement of high-dimensional data processing techniques.

目次

第 1 章

序論

1.1 背景

累積和のアルゴリズムは、データ列に対する区間和の計算を高速に行うための手法として、情報処理のさまざまな場面で広く利用されてきた。一次元累積和の基本的な考え方は、与えられたデータ列に対してあらかじめ前計算をしておくことで、後続のクエリに対して効率的な応答を可能にするというものである。これにより、区間和の計算を単純な反復処理による計算量 $O(N)$ から定数時間の計算量 $O(1)$ に削減することができる。累積和の概念は二次元へと拡張され、画像処理や地理情報システム（GIS)、データ分析など、より広範な分野においても応用が進んだ。

しかしながら、現行の累積和アルゴリズムは主に一次元または二次元のデータを対象としており、人間が視覚的に直感しやすい次元に留まっているのが現状である。三次元以上の高次元データに関する累積和アルゴリズムは十分に一般化されておらず、理論的な整備や実装の試みは限られている。

近年、データサイエンスや機械学習、ビッグデータ解析の発展に伴い、膨大な高次元データを扱う機会が増加している。例えば、画像解析では通常は二次元データが用いられるが、時間軸を加えることで三次元、さらに複数のセンサ情報を統合すれば四次元以上のデータが生成される。同様に、地理情報システム（GIS）や物理シミュレーションでは、空間的次元に時間的要素や他の物理的変数を組み合わせた多次元データが扱われることが一般的である。これらの応用分野において、任意の次元に対応する累積和アルゴリズムの理論的基盤が整備されていれば、これまで計算上の困難が伴っていた高次元データ解析の課題に対して実用的な解決策を提供できる可能性がある.

本研究では、一次元および二次元累積和の基礎理論を出発点として、それを三次元以上の高次元データに拡張するための理論の定式化と計算アルゴリズムの構築を目指した。このような理論の整備は、将来的に高次元データ解析が必要となる際に大きな利便性をもたらすだろう。

1.2 目的

本研究の目的を以下に示す.

1. **ビット表記を用いた多次元累積和のアルゴリズムを示し、その正当性を包除原理を用いて証明する:**
   高次元データの各次元の添字が複数の集合を定義し、これらの集合に対して包除原理を適用することで任意の部分領域における総和を効率的に計算する仕組みを構築する。この定式化において、添字をビット表記で表現することで、高次元空間における集合演算を効率化し、アルゴリズムの単純化および計算効率の向上を図る。
2. **多次元累積和アルゴリズムのプログラムによる実装を示す:**

この実装では、理論で示したアルゴリズムの構築手順を忠実に再現するとともに、プログラムの汎用性および可読性を考慮した設計を行う。特に、任意の次元 D に対応可能な柔軟な実装を目指す。

3. **実装したアルゴリズムの性能評価を行う:**
   データ数 N を固定し，本論文で示した多次元累積和アルゴリズムと愚直に総和を求める方法とでクエリ処理にかかる時間を測定し，本研究の有用性を示す.

第 2 章

準備

2.1 集合論

2.1.1 記法

- 集合 A の要素数を $|A|$ で表す.
- x が集合 A の元であることを $x \in A$ で表す.

2.1.2 集合の演算

定義 1 (和集合)**.** 二つの集合 A, B について，A と B のいずれか少なくとも一方に含まれるような元の集合を A と B の和集合といい， $A \cup B$ で表す．すなわち,

$$A \cup B := \{x | x \in A \text{ または } x \in B\} \tag{2.1}$$

である.

定義 2 (積集合)**.** 二つの集合 A と B 両方に含まれる元全体の集合を A と B の積集合といい，$A \cap B$ で表す．すなわち,

$$A \cap B := \{x | x \in A \text{ かつ } x \in B\} \tag{2.2}$$

である.

また，以下に示す結合法則と分配法則が成り立つことが知られている.

定理 1 (結合法則)**.** 集合 A, B, C について以下の式が成り立つ.

$$(A \cup B) \cup C = A \cup (B \cup C) \tag{2.3}$$

$$(A \cap B) \cap C = A \cap (B \cap C) \tag{2.4}$$

定理 2 (分配法則)**.** 集合 A, B, C について以下の式が成り立つ.

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \tag{2.5}$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \tag{2.6}$$

2.1.3 包除原理

有限集合 A と B に対して和集合の要素数 $|A \cup B|$ を求めるには，まずそれぞれに属する要素数 $|A|$ と $|B|$ を足し合わせ，その後 A と B の積集合 $|A \cap B|$ の要素数を引けば良い．具体的には，以下の式が成り立つ．

$$|A \cup B| = |A| + |B| - |A \cap B| \tag{2.7}$$

また，n 個の集合 $A_1, A_2, \ldots, A_n$ の和集合の要素数について，以下の式が成り立つ．

定理 3 (包除原理)**.**

$$\begin{aligned}\left|\bigcup_{i=1}^{n} A_i\right| = &\sum_i |A_i| \\ &- \sum_{i<j} |A_i \cap A_j| \\ &+ \sum_{i<j<k} |A_i \cap A_j \cap A_k| \\ &- \cdots \\ &+ (-1)^{n-1} |A_1 \cap \cdots \cap A_n|.\end{aligned} \tag{2.8}$$

これを包除原理 (Principle of Inclusion-Exclusion, PIE) という．

証明. 集合ちょうど m 個の共通部分は，右辺第 k 項において $(-1)^{k-1}\binom{m}{k}$ だけ足されるので，合計で

$$\sum_{k=1}^{m} (-1)^{k-1} \binom{m}{k}$$

となる．

ここで、二項定理より

$$\begin{aligned}0 &= (1-1)^m \\ &= \sum_{k=0}^{m} \binom{m}{k} (-1)^k \\ &= \binom{m}{0} + \sum_{k=1}^{m} \binom{m}{k} (-1)^k \\ &= 1 - \sum_{k=1}^{m} \binom{m}{k} (-1)^{k-1} \\ \therefore &\sum_{k=1}^{m} (-1)^{k-1} \binom{m}{k} = 1\end{aligned}$$

したがって、どんな m に対してもこの数え上げが常に 1 であることが分かり、包除原理が成り立つことが示された。 □

2.1.4 重み付き包除原理

包除原理を拡張し，和集合の各要素に対する重みを考慮した重み付き包除原理（Weighted Inclusion-Exclusion Principle) を導入する．

有限集合 S_1, S_2 に対して，各要素 i に重み W_i が付与されている場合，和集合の重み総和は以下のように求められる：

$$\sum_{i\in|S_1\cup S_2|} W_i = \sum_{i\in S_1} W_i + \sum_{i\in S_2} W_i - \sum_{i\in S_1\cap S_2} W_i \tag{2.9}$$

これを n 個の集合 $S_1, S_2, \ldots, S_n$ に拡張すると，次の重み付き包除原理が得られる．

定理 4 (重み付き包除原理)**.** 任意の n 個の集合 $S_1, S_2, \ldots, S_n$ に対して，各要素 i に重み W_i がある場合，和集合の重み総和について以下の式が成り立つ：

$$\begin{aligned}\sum_{i\in|\bigcup_{j=1}^{n} S_j|} W_i = &\sum_j \sum_{i\in S_j} W_i \\ &- \sum_{j_1<j_2} \sum_{i\in S_{j_1}\cap S_{j_2}} W_i \\ &+ \sum_{j_1<j_2<j_3} \sum_{i\in S_{j_1}\cap S_{j_2}\cap S_{j_3}} W_i \\ &- \cdots \\ &+ (-1)^{n-1} \sum_{i\in S_1\cap\cdots\cap S_n} W_i\end{aligned} \tag{2.10}$$

2.2 計算量

本章では、アルゴリズムの効率性を評価する指標として広く用いられる O 記法とその基本的な性質について説明する．

2.2.1 O 記法

定義 3 (O 記法)**.** 整数 N に対して定義されている関数 f, g に対し，

$$f(N) = O(g(N))$$

であるとは，ある正の実数 c と 0 以上の整数 N_0 が存在して，任意の $N \geq N_0$ に対して

$$|f(N)| \leq c|g(N)|$$

が成り立つことをいう．

多変数についても同様に定義される．

2.2.2 O 記法の性質

以下に，O 記法の基本的な性質を示す．

1. **和:**
$$O(f(n)) + O(f(n)) = O(f(n)).$$
2. **積:**
$$O(f(n))O(g(n)) = O(f(n)g(n)), \quad f(n)O(g(n)) = O(f(n)g(n)).$$
3. **定数倍:**
$$O(cf(n)) = O(f(n)) \quad (c \text{ は } 0 \text{ でない定数}).$$

2.3 群

2.3.1 群の定義

集合 G とその上の二項演算 $+ : G \times G \to G$ が群であるとは，以下の三つの条件を満たすことである．

1. **結合則:** 任意の $a, b, c \in G$ に対して，$(a + b) + c = a + (b + c)$ が成り立つ．
2. **単位元の存在:** ある $e \in G$ が存在して，任意の $a \in G$ に対して，$a + e = e + a = a$ が成り立つ．
3. **逆元の存在:** 任意の $a \in G$ に対して，ある $b \in G$ が存在して，$a + b = b + a = e$ が成り立つ．

なお本論文では，計算機上での二項演算は $O(1)$ を仮定し，演算結果のことを和と呼ぶこととする．

2.3.2 可換群

群 $(G, +)$ が可換群であるとは，任意の $a, b \in G$ に対して，$a + b = b + a$ が成り立つことである．

2.3.3 群の性質

本論文で用いる群の性質を以下に示す．

補題 1. 群 $(G, +)$ において，$-(a + b) = (-b) + (-a)$ が成り立つ．

証明. $a + b + (-b) + (-a) = a + e + (-a) = a + (-a) = e$ より，$(-b) + (-a)$ は $a + b$ の逆元である． □

補題 2. $(G, +)$ を群，$N \in \mathbb{N}$, 1 以上 N 以下の整数 i に対して $A_i \in G$ とする．このとき，

$$-(A_1 + A_2 + \ldots + A_N) = -A_N - A_{N-1} - \ldots - A_1 \tag{2.11}$$

が成り立つ．

証明. 帰納法を用いて証明する。

$N = 1$ のときは明らかに成り立つ．

$N = k$ のときに

$$-(A_1 + A_2 + \ldots + A_k) = -A_k - A_{k-1} - \ldots - A_1$$

が成り立つと仮定すると，$N = k + 1$ のとき

$$\begin{aligned} -(A_1 + A_2 + \ldots + A_{k+1}) &= -((A_1 + A_2 + \ldots + A_k) + A_{k+1}) \\ &= -A_{k+1} - (A_1 + A_2 + \ldots + A_k) \quad (\because \text{補題 1 より}) \\ &= -A_{k+1} - A_k - A_{k-1} - \ldots - A_1 \quad (\because \text{仮定より}) \end{aligned} \tag{2.12}$$

となる．したがって，$N = k + 1$ のときも成り立つ．

□

第 3 章

累積和

3.1 一次元累積和

一次元累積和は，与えられた数列に対してその区間和を前計算することで，任意の区間の和を高速に求めるための手法である．具体的には，長さ $N \in \mathbb{N}$ の数列 A に対してクエリ l, r が与えられたとき

$$\sum_{x=l}^{r} A_x \tag{3.1}$$

の値を高速に求めることができる．

3.1.1 定義

$(G, +)$ を群，0 をその単位元とし，各要素が G の元である長さ N の数列 $A = \{A_1, A_2, \ldots, A_N\}$ が与えられたとき，A の累積和 $S = \{S_0, S_1, S_2, \ldots, S_N\}$ は以下のように定義される．

$$S_0 = 0, \quad S_i = \sum_{x=1}^{i} A_x \quad (1 \leq i \leq N) \tag{3.2}$$

3.1.2 構築

累積和の構築は，漸化式 $S_i = S_{i-1} + A_i$ を用いて $O(N)$ で行うことができる．

3.1.3 クエリ処理

数列 A の区間 $[l, r]$ の和を求めるクエリ処理は，以下の定理 4 により $O(1)$ で行うことができる．

定理 5. 数列 A の区間 $[l, r]$ の和は，$-S_{l-1} + S_r$ で求めることができる．

証明.

$$\begin{aligned}
-S_{l-1} + S_r &= -\sum_{x=1}^{l-1} A_x + \sum_{x=1}^{r} A_x \\
&= -(A_1 + A_2 + \ldots + A_{l-1}) + (A_1 + A_2 + \ldots + A_{l-1} + \ldots + A_r) \\
&= -A_{l-1} - A_{l-2} \ldots - A_1 + A_1 + A_2 + \ldots + A_{l-1} + A_l + \ldots + A_r \\
&(\because \text{補題 2 より}) \\
&= A_l + A_{l+1} + \ldots + A_r
\end{aligned}$$

これはちょうど A の区間 $[l, r]$ の和である． □

3.2 二次元累積和

二次元累積和は，一次元累積和を二次元配列に拡張したものである．具体的には，$H \times W$ の二次元配列 A に対してクエリ (l_1, l_2, r_1, r_2) が与えられたとき

$$\sum_{x=l_1}^{r_1} \sum_{y=l_2}^{r_2} A_{i,j} \tag{3.3}$$

の値を高速に求めることができる．

3.2.1 定義

$(G, +)$ を可換群とする．一次元累積和と異なり，群に可換性を要請する理由については後述する．

$H \times W$ の二次元配列 $A = \{A_{x,y}\}$ が与えられたとき，A の二次元累積和 $S = \{S_{i,j}\}$ は以下のように定義される．

$$\begin{aligned} &S_{i,0} = 0, \\ &S_{0,j} = 0, \\ &S_{i,j} = \sum_{x=1}^{i} \sum_{y=1}^{j} A_{x,y} \quad (1 \leq i \leq H, 1 \leq j \leq W) \end{aligned} \tag{3.4}$$

3.2.2 構築

二次元累積和の構築は，以下のアルゴリズムを用いて $O(HW)$ で行うことができる．

Algorithm 1 二次元累積和の構築

```
for i = 0 to H do
    S_{i,0} ← 0
end for
for j = 0 to W do
    S_{0,j} ← 0
end for
for i = 1 to H do
    for j = 1 to W do
        S_{i,j} ← S_{i,j-1} + A_{i,j}
    end for
end for
for j = 1 to W do
    for i = 1 to H do
        S_{i,j} ← S_{i-1,j} + S_{i,j}
    end for
end for
```

3.2.3 クエリ処理

二次元累積和のクエリ処理は，以下の定理 5 により $O(1)$ で行うことができる．

定理 6. 二次元配列 A の区間 $[l_1, r_1] \times [l_2, r_2]$ の和は，$S_{r_1,r_2} - S_{l_1-1,r_2} - S_{r_1,l_2-1} + S_{l_1-1,l_2-1}$ で求めることができる．

証明. 累積和 $S_{i,j}$ は，A の左上 $(0,0)$ から (i,j) までの部分和である．すなわち，

$$S_{i,j} = \sum_{x=1}^{i} \sum_{y=1}^{j} A_{x,y} \tag{3.5}$$

したがって，累積和の差分を計算すると，

$$\begin{aligned} S_{r_1,r_2} - S_{l_1-1,r_2} &= \sum_{x=1}^{r_1} \sum_{y=1}^{r_2} A_{x,y} - \sum_{x=1}^{l_1-1} \sum_{y=1}^{r_2} A_{x,y} \\ &= \sum_{x=l_1}^{r_1} \sum_{y=1}^{r_2} A_{x,y} \end{aligned} \tag{3.6}$$

同様に，

$$\begin{aligned} S_{r_1,l_2-1} - S_{l_1-1,l_2-1} &= \sum_{x=1}^{r_1} \sum_{y=1}^{l_2-1} A_{x,y} - \sum_{x=1}^{l_1-1} \sum_{y=1}^{l_2-1} A_{x,y} \\ &= \sum_{x=l_1}^{r_1} \sum_{y=1}^{l_2-1} A_{x,y} \end{aligned} \tag{3.7}$$

以上より，

$$\begin{aligned} (S_{r_1,r_2} - S_{l_1-1,r_2}) - (S_{r_1,l_2-1} - S_{l_1-1,l_2-1}) &= \sum_{x=l_1}^{r_1} \sum_{y=1}^{r_2} A_{x,y} - \sum_{x=l_1}^{r_1} \sum_{y=1}^{l_2-1} A_{x,y} \\ &= \sum_{x=l_1}^{r_1} \sum_{y=l_2}^{r_2} A_{x,y} \end{aligned} \tag{3.8}$$

これはちょうど A の区間 $[l_1, r_1] \times [l_2, r_2]$ の和である． □

3.2.4 重み付き包除原理による考察

二次元累積和のクエリ処理において，重み付き包除原理を用いて定理 6. を証明することができる．

証明. 二次元配列 A を $A_{i,j}$ が重みの重み付き集合であるとみなすと，さらに A の重み付き部分集合 $W_{i,j}$ を以下のように定義する：

$$W_{i,j} = \{A_{x,y} | 1 \leq x \leq i, 1 \leq y \leq j\} \tag{3.9}$$

ここで，

$$
\begin{aligned}
\sum_{x=l_1}^{r_1}\sum_{y=l_2}^{r_2} A_{i,j} &= \sum_{i\in[l_1,r_1]\times[l_2,r_2]} A_{i,j} \\
&= \sum_{w\in W_{r_1,r_2}} w - \sum_{w\in|W_{l_1-1,r_2}\cup W_{r_1,l_2-1}|} w \\
&= \sum_{w\in W_{r_1,r_2}} w - \left(\sum_{w\in W_{l_1-1,r_2}} w + \sum_{w\in W_{r_1,l_2-1}} w - \sum_{w\in|W_{l_1-1,r_2}\cap W_{r_1,l_2-1}|} w \right) \\
&\quad (\because \text{重み付き包除原理より})
\end{aligned}
\tag{3.10}
$$

と表すことができ，特に $W_{i,j}$ の要素の重み総和が $S_{i,j}$ が等しいこと，および $|W_{l_1-1,r_2} \cap W_{r_1,l_2-1}| = W_{l_1-1,l_2-1}$ であることから，式 (3.10) は

$$
\sum_{x=l_1}^{r_1}\sum_{y=l_2}^{r_2} A_{i,j} = S_{r_1,r_2} - S_{l_1-1,r_2} - S_{r_1,l_2-1} + S_{l_1-1,l_2-1} \tag{3.11}
$$

であることが示された. □

この式は次のように解釈できる.

- S_{r_1,r_2} は $[1,r_1]\times[1,r_2]$ にあるすべての重みの合計である。
- S_{l_1-1,r_2} を引くことで、$[1,l_1-1]\times[1,r_2]$ の範囲を除去する。
- S_{r_1,l_2-1} を引くことで、$[1,r_1]\times[1,l_2-1]$ の範囲を除去する。
- しかし、$[1,l_1-1]\times[1,l_2-1]$ の範囲が二重に引かれているため、S_{l_1-1,l_2-1} を加える。

第 4 章

性能評価

謝辞

本研究を進めるにあたり,xxxxx 氏に深く感謝いたします. ありがとうございました.

参考文献

[1] 水木敬明, ”カード組を用いた秘密計算”, 電子情報通信学会 基礎・境界ソサイエティ Fundamentals Review, 2016 年 9 巻 3 号 pp179-187, 2016.

[2] Raimu Isuzugawa, Kodai Toyoda, Yu Sasaki, Daiki Miyahara, Takaaki Mizuki, ”A Card-Minimal Three-Input AND Protocol Using Two Shuffles”, Computing and Combinatorics (COCOON 2021), 2021.

[3] 吉田拓叶, 千田栄幸, 水木敬明, ”説明動画再生時間に基づくカードベースプロトコルの評価”, 電子情報通信学会 2023 年暗号と情報セキュリティシンポジウム予稿集, 4F2-1, 2023

[4] NRI 野村総合研究所, ”秘密計算—用語解説—野村総合研究所”, https://www.nri.com/jp/knowledge/glossary/lst/ha/secure_computation, （参照 2023-02-10）