秘密計算を用いた非負値行列因子分解の効率的な実現に関する一考察

天田 拓磨[1]　奈良 成泰[1]　西出 隆志[2]　吉浦 裕[1]

概要：近年の ICT の発展によりデータを集計して解析し，活用する重要性が高まっている．データが行列形式で表現されるとき，解析のためのツールの一つに行列分解がある．しかし，行列データから個人情報や機密情報が漏洩する可能性があるため，解析の際にデータを秘匿することが重要な課題となってくる．本稿では，非負値行列因子分解を取り上げ，秘密分散法により行列要素を秘匿し，マルチパーティ計算により行列分解を実行する手法を提案する．行列分解は平文でも処理コストが大きい．そのためマルチパーティ計算により実行するにあたり除算回数の削減と乗算の計算結果の再利用による効率化を提案する．また，データの桁あふれに対して浮動小数演算によるコストの増大を防ぐために，固定小数演算において小数点の位置を変更する手法について検討する．

キーワード：マルチパーティ計算，秘密計算，非負値行列因子分解，プライバシー保護

1. はじめに

近年の ICT の発展により，個人や組織，自然現象に関するデータを収集して解析し，幅広い分野で活用するニーズが高まっている．例えば，Wi-Fi 基地局が携帯端末との交信情報を収集する事例がある．Wi-Fi 基地局では周囲にある携帯端末と交信しており，端末の物理アドレスや電波強度などの情報を収集している．そういった情報は集計し，解析を行うことで携帯端末を持つ人の移動履歴などを推定することができる．この解析結果は都市計画やマーケティングに活用することが期待される．

また，個人の Web の閲覧履歴や購買履歴を活用する事例もある．Web の閲覧履歴や購買履歴を解析することで，商品のレコメンデーションやターゲッティング広告に活用することが期待される．

分析対象となるデータは行列の形式で表現されることが多い．行列の形式で表現されたデータを活用するためには，解析をする必要があるが，重要な解析ツールの一つに行列分解が挙げられる．行列分解は巨大な行列を小さな行列の積にし，データの特徴的な値を抽出することを目的とする．

しかし，解析対象のデータに個人情報や機密情報が含まれる場合，解析をする際に，行列表現されたデータから個人のプライバシーや企業の内部情報などが漏洩する可能性がある．したがって，解析の際にデータを秘匿することが重要な課題となってくる．そこで，データを秘匿したままで，行列の解析に有効な行列分解を実行する手法を提案する．具体的には，秘密分散法で行列の値を秘匿し，マルチパーティ計算で行列分解の結果のシェアを計算する．

秘密分散法に基づくマルチパーティ計算は通信を必要とし，実行の処理コストが大きい．また，行列分解は平文で計算する場合でも，処理コストが大きい．したがって，マルチパーティ計算で行列分解を実行する場合，実行の効率化が必要となる．本稿では，実行における効率化の工夫についても考える．

2. 先行研究

2.1 秘密分散法とマルチパーティ計算

秘密分散法 [1] は秘匿したいデータを分散して管理する手法である．あるデータを複数のサーバに分散して秘匿したとき，サーバ間でマルチパーティ計算（MPC）を行うことで元のデータを復元することなく，平文と同等の計算を行うことができる．このとき，閾値以上のサーバが一度に侵入されるか，閾値以上のサーバが結託しない限り，元のデータおよび計算途中の情報は漏洩することはない．

MPC の基礎演算プロトコルに加算と乗算があり，乗算のプロトコルではサーバ間での通信が必要である [2]．ま

[1] 電気通信大学
University of Electro-Communications
[2] 筑波大学
University of Tsukuba

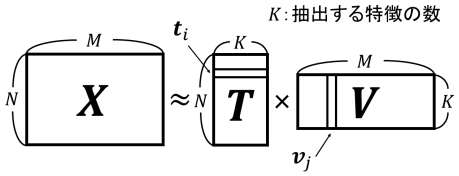

図 1 NMF の概略図

た，複数の加算と乗算を組み合わせて，除算，乱数生成，大小比較，等号判定などのプロトコル [3][4] が提案されており，これらのプロトコルの処理コストは乗算の回数で評価される．処理コストには通信量とラウンド数が用いられ，通信量は乗算プロトコルの回数を表し，ラウンド数は乗算プロトコルを並列実行した際の段数を表す．

2.2 行列分解

行列を複数の積の形式で表現することを行列分解という．行列分解の主な手法として，LU 分解，QR 分解，特異値分解（SVD），非負値行列因子分解（NMF）などが挙げられる．本稿では，非負値行列因子分解（NMF）を取り上げる．

2.2.1 NMF の概要

NMF[5][6][7] は値が 0 か正の値の行列（非負値行列）に対して実行する行列分解の一手法である．NMF は音声データや遺伝子データ，購買データなどの解析に広く用いられている．$N \times M$ サイズの非負値行列を $\boldsymbol{X}$ とする．この $\boldsymbol{X}$ を $\boldsymbol{T}$ と $\boldsymbol{V}$ という二つの行列の積に分解する．すなわち，$\boldsymbol{X} \approx \boldsymbol{TV}$ となるような行列の近似を考える．このとき $\boldsymbol{T}$ のサイズは $N \times K$ であり，$\boldsymbol{V}$ のサイズは $K \times M$ である．ここで，K は特徴の数であり，K の値は事前に設定することができる．NMF の概略図を図 1 に示す．

2.2.2 NMF のアルゴリズム

$N \times M$ 行列の $\boldsymbol{X}$ を $N \times K$ 行列 $\boldsymbol{T}$ と $K \times M$ 行列 $\boldsymbol{V}$ の積で近似するとき，その誤差の評価方法には Euclid 距離や Kullback-Leibler 距離などがよく用いられるが，ここでは Euclid 距離（二乗誤差）を用いた NMF について説明する．各行列の ij 要素をそれぞれ x_{ij}，t_{ij}，v_{ij} と表すと誤差評価の式は以下のように表せる．

$$d_{Eu} = \sqrt{\sum_{i,j} \left(x_{ij} - \sum_{k=1}^{K} t_{ik} v_{kj} \right)^2} \tag{1}$$

式（1）の d_{Eu} を最小化する行列 $\boldsymbol{T}$,$\boldsymbol{V}$ を EM アルゴリズムにより求める．すると，以下のような要素ごとの更新式が導出される．[8]

$$t_{ik}^{(\ell+1)} \leftarrow t_{ik}^{(\ell)} \frac{\sum_{0 \le j \le M} x_{ij} v_{kj}^{(\ell)}}{\sum_{0 \le j \le M} \left(\boldsymbol{t}_i^{(\ell)}, \boldsymbol{v}_j^{(\ell)} \right) v_{kj}^{(\ell)}} \tag{2}$$

$$v_{kj}^{(\ell+1)} \leftarrow v_{kj}^{(\ell)} \frac{\sum_{0 \le i \le N} x_{ij} t_{ik}^{(\ell+1)}}{\sum_{0 \le i \le N} \left(\boldsymbol{t}_i^{(\ell+1)}, \boldsymbol{v}_j^{(\ell)} \right) t_{ik}^{(\ell+1)}} \tag{3}$$

$$\left(\boldsymbol{t}_i^{(\ell)}, \boldsymbol{v}_j^{(\ell)} \right) = \sum_{k=0}^{K} t_{ik}^{(\ell)} v_{kj}^{(\ell)} \tag{4}$$

これは，乗法的更新ルールと呼ばれている．この更新を誤差が収束するまで繰り返し実行することで，$\boldsymbol{T}$ と $\boldsymbol{V}$ の値を求めることができる．ただし，$t_{ik}^{(\ell)}$ は第 ℓ ループ目の t_{ik} の要素を表しており，$(\boldsymbol{t}_i, \boldsymbol{v}_j)$ は行ベクトル $\boldsymbol{t}_i$ と列ベクトル $\boldsymbol{v}_j$ の内積を表している．

NMF は以下の手順で実行される．

1. $\boldsymbol{T}$ と $\boldsymbol{V}$ の各要素を乱数で初期化する．
2. 式（1）により，元の行列との誤差の計算をする．
3. 誤差が閾値以下になるか，一定回数以上繰り返したら，分解結果を出力して終了する．
4. 式（2）及び式（3）により行列要素を更新する．
5. step2 に戻る．

2.3 行列分解の秘匿計算

値を秘匿して行列分解を実行する先行研究はいくつかある．Yang ら [9] は紛失通信を用い，MPC で行列分解をセキュアに実行する手法を提案している．論文中では，行列分解として QR 分解などの秘密計算について述べられているが，2 パーティのセミオネストモデルを想定している．また，Li ら [10] はレコメンダシステムにおけるアルゴリズムの秘密計算を提案している．データを集計した行列に対して，NMF と協調フィルタリングを組み合わせて実行することで，個人のプライバシーを保護しつつデータ解析が可能であることを示した．また，Nikolaenko ら [11] はプライバシーを秘匿した行列分解の方法を提案している．レコメンダシステムの協調フィルタリングにおける秘密計算を前提にしており，準同型暗号と Yao の Garbled Circuit[12] を用いて行列分解の秘密計算を実装評価している．さらに行列のスパース性から実行の高速化手法も提案している．

3. 秘匿計算による行列分解の構成

3.1 記法と定義

本稿で用いる記号や記法についての定義を以下に示す．

- $\boldsymbol{X}$: 分解前の行列
- N: 行列 $\boldsymbol{X}$ の行数
- M: 行列 $\boldsymbol{X}$ の列数
- K: 抽出する特徴の数
- $\boldsymbol{T}$,$\boldsymbol{V}$: 分解結果の行列
  （$\boldsymbol{T}$ は $N \times K$ 行列，$\boldsymbol{V}$ は $K \times M$ 行列である．）
- L: NMF における更新のループ回数

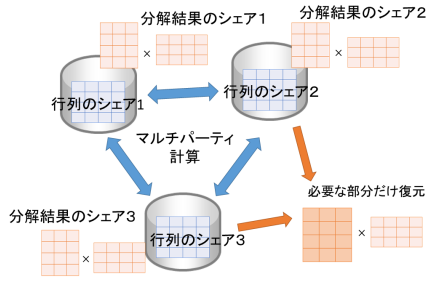

図 2 NMF を MPC で実行するときのモデル

- ℓ: ループの番号を表すパラメータ
- B_i: 値の整数部分のビット長
- B_d: 値の小数部分のビット長
- B: ビット長（$B = B_i + B_d$）
- $[a]$: 値 a のシェア
- $[\boldsymbol{A}]$: 要素がすべてシェアになっている行列 $\boldsymbol{A}$

3.2 システム構成

NMF を MPC により実行する方式について説明する．分解対象の行列 $\boldsymbol{X}$ のシェア $[\boldsymbol{X}]$ を各参加者に分散する．NMF の実行後は，各参加者が分解結果の行列 $\boldsymbol{T}$,$\boldsymbol{V}$ のシェア $[\boldsymbol{T}]$,$[\boldsymbol{V}]$ を持っているものとする．分解結果は秘匿されており，仮に $[\boldsymbol{T}]$,$[\boldsymbol{V}]$ の両方を復元すると，その行列積を計算することで，$[\boldsymbol{X}]$ の近似値を知ることができる．したがって，結果を得るときは必要な要素または基底のみを復元するものとする．その概略図を図 2 に示す．

3.3 基本方式による実行手順

2.2.2 で述べた，NMF のアルゴリズムを MPC で実行する．実行手順は以下のようになる．

1. 乱数生成プロトコルにより，$[t_{ik}], [v_{kj}]$ を初期化する．
2. 式 (1) の平方根の中のみ乗算，加算プロトコルを用いて計算する．
3. step2 で求めた値と閾値の二乗の大小を大小比較プロトコルにより計算し，結果を復元して誤差が閾値よりも小さい場合，実行を終了する．また，ループ回数が一定回数以上である場合も，実行を終了する．
4. 式 (2) 式 (3) により，加算，乗算，除算のプロトコルを用いて $[t_{ik}], [v_{kj}]$ を更新する．
5. step2 に戻る．

3.4 小数の扱い

一般に，$\boldsymbol{X}$,$\boldsymbol{T}$,$\boldsymbol{V}$ の各要素は小数である．そのため，値を小数で表現する必要があるので，固定小数のシェアを考えることとする．固定小数のシェアの表現方法とその演算については Catrina らのプロトコル [13] を基にする．固定小数は，有理数 $\mathbb{Q}$ から整数 $\mathbb{Z}$ への写像 $\mathbb{Q} \mapsto \mathbb{Z}$ により整数に変換され，仮想的な整数部分と小数部分に分かれている

表 1 基本方式の各ステップにおける演算回数

	乗算	除算	乱数生成	大小比較
step1	—	—	$K(N+M)$	—
step2	$2NM$	—	—	
step3	1	—	—	1
step4	$K(2NM + 3N + 3M)$	$K(N+M)$	—	

表 2 実行全体における各演算の回数

	実行回数
乗算	$L(2KNM + 3KN + 3KM + 2NM + 1)$
除算	$KL(N+M)$
大小比較	L
乱数生成	$K(N+M)$

連続したビット列で表現される．したがって，固定小数での四則演算は $\mathbb{Z}_p$ 上の演算と同様に実行できる．例として，整数部分 4 桁，小数部分 2 桁の合計 6 桁の 10 進数の固定小数で計算するシステムを想定する．実際のシステムでは，値は 2 進数で表現されているが，簡単のために 10 進数で例示する．数値の例として，$a = 0081.62$ と $b = 0004.91$ を考える．このとき，シェア $[a], [b]$ は小数から整数に変換され，整数として保持される．$[a] \times [b] = [ab]$ を計算することを考えると，$a \times b = 400.7542$ であるので $[ab] = [040075]$ のように整数部分は 4 桁，小数部分は 2 桁と桁をそろえて保持するものとする．このとき，下位に溢れた数は丸められる．丸める方法としては，$[ab]$ に乱数のシェア $[r]$ を加えて $ab + r$ を復元し，下位を丸めて再びシェアに戻す．この方法は Catrina らの論文 [13] で TruncPr というプロトコルが提案されている．ここでは，値の桁数を 6 と設定したが，実際のシステムではユーザが定義するものとする．

3.5 演算回数による処理コストの見積もり

基本方針での処理コストを概算する．処理コストは処理に用いる大小比較や除算などのプロトコルによって異なるので，各プロトコルの実行回数により評価するものとする．まず，必要な MPC のプロトコルについて述べる．3.3 で述べた手順に沿って概算すると，各 step におけるプロトコルの実行回数は表 1 のようになる．各プロトコルは固定小数演算のプロトコルである．step1 では，乱数生成プロトコルを $\boldsymbol{T}, \boldsymbol{V}$ の要素数だけ用いる．step2 の誤差計算では，$\boldsymbol{t}_i$ と $\boldsymbol{v}_j$ の内積を計算し，x_{ij} との差を二乗する．$\boldsymbol{X}$ のすべての要素に対して $\boldsymbol{t}_i$ と $\boldsymbol{v}_j$ との差を計算し，その総和をとる．step3 では誤差と収束判定の閾値との大小比較を実行する．step4 では，$\boldsymbol{T}$ と $\boldsymbol{V}$ を要素毎に更新する．ただし，$[a][b] + [c][d] = [ab + cd]$ のような 2 次以下の多項式計算は乗算 1 回に帰着できることを用いて，乗算の実行回数を算出した [14]．step2 から step4 までを L 回ループするとして，各プロトコルの合計実行回数を求めた結果を表 2 に示す．

表 3 固定小数での各演算の処理コスト（事前計算あり）

	通信量	ラウンド数
加算	0	0
乗算	2	2
除算	$4\theta + 1.5B\log B + 4B + 6$	$2\theta + 3\log B + 10$
大小比較	$2B - 2$	$\log B + 2$

Catrina らのプロトコル [4][13] を用いて，固定小数における MPC の各演算の処理コストを表 3 に示す．ただし，乗算は整数体上の乗算 1 回と桁を切る操作が含まれるので通信量，ラウンド数がともに 2 となっている．また，除算は乗算により近似するアルゴリズムを用いており，θ は近似における繰り返し回数である．乱数生成は事前計算可能であり，各プロトコルの処理コストの評価においても乱数生成の処理コストを含めていない．したがって，ここでは乱数生成は処理コストに含めないものとする．基本方式で実行する場合，乗算，除算，大小比較の実行回数がループ回数に比例して大きくなる．表 3 より，実行に必要なプロトコル中で最も処理コストの大きい演算は除算だとわかる．除算は更新 1 回につき $\boldsymbol{T}, \boldsymbol{V}$ の要素数だけ実行しなければならない．そこで，更新式を変形することで除算の実行回数を削減し，除算による処理コストを削減することを考える．

4. 改良方式

4.1 除算削減の方針

以下の命題を証明することを考える．

命題 4.1 $t_{ik}^{(\ell)}, v_{kj}^{(\ell)}$ の分子をそれぞれ $T_{TPik}^{(\ell)}, V_{TPkj}^{(\ell)}$，分母をそれぞれ $T_{BTik}^{(\ell)}, V_{BTkj}^{(\ell)}$ と表すとき，

$$t_{ik}^{(\ell)} = \frac{T_{TPik}^{(\ell)}}{T_{BTik}^{(\ell)}}, \quad v_{kj}^{(\ell)} = \frac{V_{TPkj}^{(\ell)}}{V_{BTkj}^{(\ell)}}$$

と書き表すことができ，$T_{TPik}^{(\ell)}$,$V_{TPkj}^{(\ell)}$,$T_{BTik}^{(\ell)}$,$V_{BTik}^{(\ell)}$ は x_{ij},$t_{ik}^{(0)}$,$v_{kj}^{(0)}$ の和と積で計算できる．

上記の命題に基づいて行列分解の結果である $t_{ik}^{(L)}, v_{kj}^{(L)}$ の分子分母を $x_{ij}, t_{ik}^{(0)}, v_{kj}^{(0)}$ の和と積のみで計算する．これにより除算の回数を L 回から 1 回に削減することができる．

4.2 命題の証明

説明の都合上，以下のような 2×2 の行列を 2 つの 2×2 の行列の積に分解する場合について具体的な証明を述べる．

$$\begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \approx \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix} \begin{pmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{pmatrix} \quad (5)$$

上記の命題 4.1 について，数学的帰納法により証明する．

証明 1 (i) $\ell = 1$ のとき

$$\begin{aligned} t_{ik}^{(1)} &\leftarrow \frac{t_{ik}^{(0)}(x_{i1}v_{k1}^{(0)} + x_{i2}v_{k2}^{(0)})}{(t_{i1}^{(0)}v_{11}^{(0)} + t_{i2}^{(0)}v_{21}^{(0)})v_{k1}^{(0)} + (t_{i1}^{(0)}v_{12}^{(0)} + t_{i2}^{(0)}v_{22}^{(0)})v_{k2}^{(0)}} \\ &= \frac{T_{TPik}^{(1)}}{T_{BTik}^{(1)}} \\ v_{kj}^{(1)} &\leftarrow \frac{v_{kj}^{(0)}(x_{1j}t_{1k}^{(0)} + x_{2j}t_{2k}^{(0)})}{(t_{11}^{(0)}v_{1j}^{(0)} + t_{12}^{(0)}v_{2j}^{(0)})t_{1k}^{(0)} + (t_{21}^{(0)}v_{1j}^{(0)} + t_{22}^{(0)}v_{2j}^{(0)})t_{2k}^{(0)}} \\ &= \frac{V_{TPkj}^{(1)}}{V_{BTkj}^{(1)}} \end{aligned}$$

より，$T_{TPik}^{(1)}$,$T_{BTik}^{(1)}$,$V_{TPkj}^{(1)}$,$V_{BTkj}^{(1)}$ は，x_{ij},$t_{ik}^{(0)}$,$v_{kj}^{(0)}$ の和と積のみで計算できる．

(ii) $\ell = \alpha$ のとき

命題 4.1 を仮定する．すなわち

$$t_{ik}^{(\alpha)} = \frac{T_{TPik}^{(\alpha)}}{T_{BTik}^{(\alpha)}}, \quad v_{kj}^{(\alpha)} = \frac{V_{TPkj}^{(\alpha)}}{V_{BTkj}^{(\alpha)}}$$

と書け，$T_{TPik}^{(\alpha)}$,$T_{BTik}^{(\alpha)}$,$V_{TPkj}^{(\alpha)}$,$V_{BTkj}^{(\alpha)}$ は，x_{ij},$t_{ik}^{(0)}$,$v_{kj}^{(0)}$ の和と積のみで計算できると仮定する．このとき，$\ell = \alpha + 1$ を考え，以下のように式を変形する．

$$\begin{aligned} & t_{ik}^{(\alpha+1)} \\ & \leftarrow \frac{t_{ik}^{(\alpha)}(x_{i1}v_{k1}^{(\alpha)} + x_{i2}v_{k2}^{(\alpha)})}{(t_{i1}^{(\alpha)}v_{11}^{(\alpha)} + t_{i2}^{(\alpha)}v_{21}^{(\alpha)})v_{k1}^{(\alpha)} + (t_{i1}^{(\alpha)}v_{12}^{(\alpha)} + t_{i2}^{(\alpha)}v_{22}^{(\alpha)})v_{k2}^{(\alpha)}} \\ & = \frac{\frac{T_{TPik}^{(\alpha)}}{T_{BTik}^{(\alpha)}}(x_{i1}\frac{V_{TPk1}^{(\alpha)}}{V_{BTk1}^{(\alpha)}} + x_{i2}\frac{V_{TPk2}^{(\alpha)}}{V_{BTk2}^{(\alpha)}})}{(\frac{T_{TPi1}^{(\alpha)}}{T_{BTi1}^{(\alpha)}}\frac{V_{TP11}^{(\alpha)}}{V_{BT11}^{(\alpha)}} + \frac{T_{TPi2}^{(\alpha)}}{T_{BTi2}^{(\alpha)}}\frac{V_{TP21}^{(\alpha)}}{V_{BT21}^{(\alpha)}})\frac{V_{TPk1}^{(\alpha)}}{V_{BTk1}^{(\alpha)}} + (\frac{T_{TPi1}^{(\alpha)}}{T_{BTi1}^{(\alpha)}}\frac{V_{TP12}^{(\alpha)}}{V_{BT12}^{(\alpha)}} + \frac{T_{TPi2}^{(\alpha)}}{T_{BTi2}^{(\alpha)}}\frac{V_{TP22}^{(\alpha)}}{V_{BT22}^{(\alpha)}})\frac{V_{TPk2}^{(\alpha)}}{V_{BTk2}^{(\alpha)}}} \\ & = \frac{T_{TPik}^{(\alpha)}}{T_{BTik}^{(\alpha)}} \\ & \times \frac{(x_{i1}V_{TPk1}^{(\alpha)}V_{BTk2}^{(\alpha)} + x_{i2}V_{TPk2}^{(\alpha)}V_{BTk1}^{(\alpha)})T_{TPik}^{(\alpha)}T_{BTi2}^{(\alpha)}V_{BT11}^{(\alpha)}V_{BT21}^{(\alpha)}V_{BT12}^{(\alpha)}V_{BT22}^{(\alpha)}}{\phi_1 V_{TPk1}^{(\alpha)}V_{BTk2}^{(\alpha)}V_{BT12}^{(\alpha)}V_{BT22}^{(\alpha)} + \phi_2 V_{TPk2}^{(\alpha)}V_{BTk1}^{(\alpha)}V_{BT11}^{(\alpha)}V_{BT21}^{(\alpha)}} \end{aligned} \quad (6)$$

$$\begin{aligned} & v_{kj}^{(\alpha+1)} \\ & \leftarrow \frac{v_{kj}^{(\alpha)}(x_{1j}t_{1k}^{(\alpha)} + x_{2j}t_{2k}^{(\alpha)})}{(t_{11}^{(\alpha)}v_{1j}^{(\alpha)} + t_{12}^{(\alpha)}v_{2j}^{(\alpha)})t_{1k}^{(\alpha)} + (t_{21}^{(\alpha)}v_{1j}^{(\alpha)} + t_{22}^{(\alpha)}v_{2j}^{(\alpha)})t_{2k}^{(\alpha)}} \\ & = \frac{\frac{V_{TPkj}^{(\alpha)}}{V_{BTkj}^{(\alpha)}}(x_{1j}\frac{T_{TP1k}^{(\alpha)}}{T_{BT1k}^{(\alpha)}} + x_{2j}\frac{T_{TP2k}^{(\alpha)}}{T_{BT2k}^{(\alpha)}})}{(\frac{T_{TP11}^{(\alpha)}}{T_{BT11}^{(\alpha)}}\frac{V_{TP1j}^{(\alpha)}}{V_{BT1j}^{(\alpha)}} + \frac{T_{TP12}^{(\alpha)}}{T_{BT12}^{(\alpha)}}\frac{V_{TP2j}^{(\alpha)}}{V_{BT2j}^{(\alpha)}})\frac{T_{TP1k}^{(\alpha)}}{T_{BT1k}^{(\alpha)}} + (\frac{T_{TP21}^{(\alpha)}}{T_{BT21}^{(\alpha)}}\frac{V_{TP1j}^{(\alpha)}}{V_{BT1j}^{(\alpha)}} + \frac{T_{TP22}^{(\alpha)}}{T_{BT22}^{(\alpha)}}\frac{V_{TP2j}^{(\alpha)}}{V_{BT2j}^{(\alpha)}})\frac{T_{TP2k}^{(\alpha)}}{T_{BT2k}^{(\alpha)}}} \\ & = \frac{V_{TPkj}^{(\alpha)}}{V_{BTkj}^{(\alpha)}} \\ & \times \frac{(x_{1j}T_{TP1k}^{(\alpha)}T_{BT2k}^{(\alpha)} + x_{2j}T_{TP2k}^{(\alpha)}T_{BT1k}^{(\alpha)})T_{BT11}^{(\alpha)}V_{BT1j}^{(\alpha)}T_{BT12}^{(\alpha)}V_{BT22}^{(\alpha)}T_{BT21}^{(\alpha)}T_{BT22}^{(\alpha)}}{\phi_3 T_{TP1k}^{(\alpha)}T_{BT2k}^{(\alpha)}T_{BT21}^{(\alpha)}T_{BT22}^{(\alpha)} + \phi_4 T_{TP2k}^{(\alpha)}T_{BT1k}^{(\alpha)}T_{BT11}^{(\alpha)}T_{BT12}^{(\alpha)}} \end{aligned} \quad (7)$$

ただし，

$$\phi_1 = T^{(\alpha)}_{TPi1}V^{(\alpha)}_{TP11}T^{(\alpha)}_{BTi2}V^{(\alpha)}_{BT21} + T^{(\alpha)}_{TPi2}V^{(\alpha)}_{TP21}T^{(\alpha)}_{BTi2}V^{(\alpha)}_{BTk2}$$
$$\phi_2 = T^{(\alpha)}_{TPi1}V^{(\alpha)}_{TP12}T^{(\alpha)}_{BTi2}V^{(\alpha)}_{BT22} + T^{(\alpha)}_{TPi2}V^{(\alpha)}_{TP22}T^{(\alpha)}_{BTi1}V^{(\alpha)}_{BTk2}$$
$$\phi_3 = T^{(\alpha)}_{TP11}V^{(\alpha)}_{TP1j}T^{(\alpha)}_{BT12}V^{(\alpha)}_{BT2j} + T^{(\alpha)}_{TP12}V^{(\alpha)}_{TP2j}T^{(\alpha)}_{BT11}V^{(\alpha)}_{BT1j}$$
$$\phi_4 = T^{(\alpha)}_{TP21}V^{(\alpha)}_{TP1j}T^{(\alpha)}_{BT22}V^{(\alpha)}_{BT2j} + T^{(\alpha)}_{TP22}V^{(\alpha)}_{TP2j}T^{(\alpha)}_{BT21}V^{(\alpha)}_{BT1j} \tag{8}$$

である．このとき，式 (6)，式 (7) の右辺の $T^{(\alpha)}_{TPik}$,$T^{(\alpha)}_{BTik}$,$V^{(\alpha)}_{TPkj}$,$V^{(\alpha)}_{BTkj}$ は，仮定から x_{ij},$t^{(0)}_{ik}$,$v^{(0)}_{kj}$ の和と積のみで計算されている．したがって，式 (6) も x_{ij},$t^{(0)}_{ik}$,$v^{(0)}_{kj}$ の和と積のみで計算されていることがわかる．ここで，式 (6) の右辺を $T^{(\alpha+1)}_{TPik}/T^{(\alpha+1)}_{BTik}$ とおき，式 (7) の右辺を $V^{(\alpha+1)}_{TPkj}/V^{(\alpha+1)}_{BTkj}$ とおくことで命題が成り立つ．したがって，(i) (ii) より数学的帰納法から命題 4.1 は成り立つ． □

命題 4.1 より，2×2 行列を 2 つの 2×2 行列の積に分解する NMF については，初期値から第 ℓ ループまでで $t^{(\ell)}_{ik}, v^{(\ell)}_{kj}$ を除算を実行しないで求めることができ，式 (6)，式 (7) により除算を実行しない更新式が導出される．

次に，より一般的な $N\times M$ 行列を $N\times K$ 行列と $K\times M$ 行列の積に分解する NMF に除算削減の工夫を適用することを考える．より一般的な更新式についても，命題 4.1 と同様に考えれば，数学的帰納法により除算を実行しない更新式が導出される．証明に関しては，紙面の都合上省略するが更新式は以下のようになる．

$$\frac{T^{(\ell+1)}_{TPik}}{T^{(\ell+1)}_{BTik}} \leftarrow \frac{T^{(\ell)}_{TPik}}{T^{(\ell)}_{BTik}} \frac{\sum_{1\le j\le M} x_{ij}\eta^{S_K}_{\boldsymbol{T},\phi}\eta^{S_M}_{\boldsymbol{T},j}\xi_{\boldsymbol{V},\phi}V^{(\ell)}_{TPkj}}{\sum_{1\le j\le M}\eta^{S_M}_{\boldsymbol{V},\boldsymbol{j}}V^{(\ell)}_{TPkj}\sum_{1\le m\le K}\left(\eta^{S_K}_{\boldsymbol{T},m}\xi_{\boldsymbol{V},(m,j)}T^{(\ell)}_{TPim}V^{(\ell)}_{TPmj}\right)} \tag{9}$$

$$\frac{V^{(\ell+1)}_{TPkj}}{V^{(\ell+1)}_{BTkj}} \leftarrow \frac{V^{(\ell)}_{TPkj}}{V^{(\ell)}_{BTkj}} \frac{\sum_{1\le i\le N} x_{ij}\psi^{S_K}_{\boldsymbol{V},\phi}\psi^{S_N}_{\boldsymbol{V},i}\xi_{\boldsymbol{T},\phi}T^{(\ell)}_{TPik}}{\sum_{1\le i\le N}\psi^{S_N}_{\boldsymbol{T},\boldsymbol{i}}T^{(\ell)}_{TPik}\sum_{1\le m\le K}\left(\psi^{S_K}_{\boldsymbol{V},m}\xi_{\boldsymbol{T},(i,m)}T^{(\ell)}_{TPim}V^{(\ell)}_{TPmj}\right)} \tag{10}$$

ただし，

$$\eta^{S_A}_{\boldsymbol{X},j} := \prod_{j'\in S_A\setminus\{j\}} X^{(\ell)}_{BTij'} \tag{11}$$
$$\psi^{S_A}_{\boldsymbol{X},i} := \prod_{i'\in S_A\setminus\{i\}} X^{(\ell)}_{BTi'j} \tag{12}$$
$$\xi_{\boldsymbol{X},(n,l)} := \prod_{(n,l)\in E_X\setminus\{(i,j)\}} X^{(\ell)}_{BTnl} \tag{13}$$
$$S_A := \{1,2,\cdots A\} \tag{14}$$
$$E_{\boldsymbol{X}} \ : \ \text{行列 } \boldsymbol{X} \text{ のすべての要素番号の組の集合}$$

$\eta^{S_A}_{\boldsymbol{X},j}$ は行列 $\boldsymbol{X}$ の j 列目の要素を除いた i 行目の $X_{BTij'}$ の総乗で，$\psi^{S_A}_{\boldsymbol{X},i}$ は行列 $\boldsymbol{X}$ の i 行目の要素を除いた j 列目の $X_{BTi'j}$ の総乗で，$\xi^{S_A}_{\boldsymbol{X},(n,l)}$ は行列 $\boldsymbol{X}$ の X_{BTnl} を除いた全ての X_{BTij} の総乗を表している．

表 4 改良方式の各ステップにおける演算回数

	乗算	除算	大小比較	乱数生成
step1	—	—	—	$K(N+M)$
step2	—	—	—	—
step3	$L\{K^3NM(N+M+1)+K^2NM(N+M)+2K(NM^2+N^2M-2NM+2N+2M)\}$	—	—	
step4	—	$K(N+M)$	—	
step5	$2NM$	—	—	
step6	—	—	1	

表 5 実行全体における各演算の回数

	回数
乗算	$L\{K^3NM(N+M+1)+K^2NM(N+M)+2K(NM^2+N^2M-2NM+2N+2M)\}+2NM$
除算	$K(N+M)$
大小比較	1
乱数生成	$(N+M)K$

4.3 改良方式による実行手順

改良方式の NMF では，式 (9) 及び式 (10) を更新式として用いる．以下に手順を示す．

1. 乱数生成プロトコルにより，$[t_{ik}],[v_{kj}]$ を初期化する．
2. $[T_{TPik}]\leftarrow[t_{ik}]$,$[V_{TPkj}]\leftarrow[v_{kj}]$,$[T_{BTik}]\leftarrow[1]$, $[V_{BTkj}]\leftarrow[1]$ とする．
3. 式 (9)，式 (10) により，加算プロトコルと乗算プロトコルを用いて，指定回数 L だけ値の更新を行う．
4. 除算プロトコルにより $[t_{ik}]\leftarrow[T_{TPik}]/[T_{BTik}],[v_{kj}]\leftarrow[V_{TPkj}]/[V_{BTkj}]$ を計算し，分解結果のシェアを得る．
5. 式 (1) の平方根の中のみ加算，乗算プロトコルを用いて計算する．
6. step5 で求めた値と閾値の二乗との大小を大小比較プロトコルにより計算し，結果を復元して誤差が閾値よりも小さい場合，実行を終了する．大きい場合，step2 に戻る．

基本方式では，誤差の計算と値の更新を交互に行い，値が収束したかどうか判定しながら実行する．改良方式ではループ毎に各要素の値は求めないので，ループ毎の収束判定は行わない．

4.4 演算回数による処理コストの見積もり

4.3 で述べた手順に沿ってプロトコルの実行回数を表 4 に示す．step1 では，乱数生成プロトコルを $\boldsymbol{T},\boldsymbol{V}$ の要素数

表 6 各プロトコルの実行回数の基本方式と改良方式での比較

	乗算	除算
基本方式	$O(KN^2L)$	$O(KNL)$
改良方式	$O(K^3N^3L)$	$O(KN)$

だけ用いる．step2 では，新しい更新式に適用するために，$\boldsymbol{T}, \boldsymbol{V}$ それぞれの分子に step1 で生成した乱数のシェアを代入し，分母に 1 のシェアを代入する．step3 では更新式を用いて $\boldsymbol{T}, \boldsymbol{V}$ を L 回更新する．step4 では，除算を実行して $[t_{ik}], [v_{kj}]$ を計算する．step5 では，加算と乗算を用いて誤差を計算する．step6 では，step5 で求めた誤差と閾値とを比較する．各プロトコルの合計実行回数を求める．実行における各演算の実行回数は表 5 のようになる．

式 (9)，式 (10) の更新式を用いて値の更新を行うことで，基本方式では除算回数が $(N+M)KL$ 回である一方，改良方式では除算回数が $(N+M)K$ 回になる．すなわち，実行全体における除算回数が $1/L$ になる．

このとき，各プロトコルの実行回数を表す式が繁雑であるため，回数をオーダーで表す．各プロトコルと回数のオーダーを表 6 に示す．表 6 からわかるように，除算削減の工夫を適用した更新式を用いることで，除算の実行回数は $O(KLN)$ から $O(KN)$ に削減できた．

4.5 安全性

本稿で述べている各方式では，データ秘匿の手法に秘密分散法を想定し，シェア同士の MPC により計算結果を求めている．したがって実行全体の安全性は秘密分散法と MPC の安全性に帰着する．

計算途中で値を復元するのは，誤差と設定した閾値との大小比較の結果のみである．$\boldsymbol{X}$ と $\boldsymbol{T}, \boldsymbol{V}$ は秘匿されており，秘匿されたもの同士の誤差から $\boldsymbol{X}, \boldsymbol{T}, \boldsymbol{V}$ が確率的に推定されることはない．ただし，誤差と閾値との大小から，収束の速さを観測することができる．収束の速さから $\boldsymbol{X}$ のスパース性などを経験的に知ることができる可能性がある．確実に収束する大きなループ回数を設定すればよいが，その場合は処理コストが大きくなる．したがって，この推定に関する安全性は処理コストのトレードオフになる．

また，参加者同士の結託に対する安全性は，システムで採用する秘密分散法と MPC のプロトコルに寄与する．例えば，Shamir(k, n) 閾値秘密分散法 [1] を用いた線形秘密分散の場合，参加者 n 人のうち $k-1$ 人までの結託には安全である．また，Damgård らの MPC の手法 (SPDZ)[15] を用いれば，参加者 n 人のうちの $n-1$ 人までの結託と悪意のある行動に対して安全である．ただし，この場合は 2 次以下の多項式を乗算 1 回と見なすことはできないので，注意が必要である．

5. 改良方式の問題点

更新式を変形したことにより，以下の 2 つの問題が発生する．

1. 乗算回数の増加
2. 桁数の増加

問題 1 については，表 6 により，乗算の回数が $O(KN^2L)$ から $O(K^3N^3L)$ に増加したことがわかる．除算の回数が $O(KNL)$ から $O(KN)$ に減少しても，乗算の回数が増加したことで全体の処理コストは増加する．

また，問題 2 については，除算を実行しないことにより発生する問題である．基本方式では値の更新において，除算を実行するので値の増加をある程度抑制することができる．しかし，改良方式では加算と乗算のみ実行し，除算を実行せずに更新していくので，桁数が急速に増加していく．そのため，非常に大きな桁を用意する必要がある．例として，初期値の整数部分を 32 ビットとすると，1 ループ終了後の T_{TPik} は約 35 万ビットの桁数となる．想定するシステムでは固定小数での演算を行うので，上位に桁あふれが発生し，正確な計算結果が得られない可能性がある．仮に設定するビット長を十分に長くとると，表 3 より，乗算と除算の処理コストが整数部分のビット長に比例して増大するので処理コストが膨大になる．

6. 乗算回数の抑制

2 つの問題のうち，乗算回数の増加に対する解決方法を示す．

6.1 計算結果の再利用

乗算の回数を削減するような工夫を考える．除算を削減するように変形した更新式では，要素毎の更新において共通の計算が含まれている．例えば，式 (9) の分母における $\xi_{\boldsymbol{X},\phi}$（$\boldsymbol{V}$ の全要素の分母の総乗）の計算は，更新する行列の要素 T_{TPik}/T_{BTik} の i, k によらず共通の計算である．要素毎に同じ計算をするのは効率が悪いので，ループ毎にあらかじめ計算することとする．そして計算結果をメモリに記憶しておき，適宜用いる．この操作により，要素ごとに計算していた式 (13) の計算をループ前に一回だけ計算すればよいことになる．$\boldsymbol{T}$ のループにおける式 (13) の計算を例にとると，式 (13) のループ 1 回における乗算回数は $KM-1$ 回である．$\boldsymbol{T}$ の要素すべてに対して式 (13) の計算を実行するので，式 (13) のための $\boldsymbol{T}$ のループ 1 回における乗算回数は，$(KM-1) \times KN = KN(KM-1)$ 回である．一方で，式 (13) をループ前に計算しておく方法であれば，式 (13) ための $\boldsymbol{T}$ のループ 1 回における乗算回数は，$(KM-1)$ 回である．すなわち，乗算回数は 1 ループにつき $1/KN$ に削減できる．

表 7 各プロトコルの実行回数の比較

	乗算	除算
基本方式	$O(KN^2L)$	$O(KNL)$
改良方式	$O(K^3N^3L)$	$O(KN)$
改良方式 (計算結果の再利用)	$O(KN^2L)$	$O(KN)$

さらに，$\xi_{\boldsymbol{X},\phi}$ を計算する過程においても，計算結果を最大限再利用し，効率化する．$\boldsymbol{T}$ のループ毎の計算手順について説明する．まず $\boldsymbol{T}$ について要素一つを抜いた各行 T_{BTij} の総乗を計算する．すなわち，$i = 1, 2, \cdots, N$ と $j = 1, 2, \cdots, K$ について $\eta^{S_K}_{\boldsymbol{T},j}$ を計算する．次に，$i = 1, 2, \cdots, N$ と $j = 1, 2, \cdots K$ について $\eta^{S_K}_{\boldsymbol{T},j}$ と T_{TPij} の積を計算し，最後に $\eta^{S_K}_{\boldsymbol{T},1}$ と T_{BTi1} との積を計算する．ここで，$\eta^{S_K}_{\boldsymbol{T},1}$ と T_{BTi1} との積は $\boldsymbol{T}$ の i 行目における分母の総乗，すなわち，$\eta^{S_K}_{\boldsymbol{T},\phi}$ に他ならない．$i = 1, 2, \cdots, N$ に対する $\eta^{S_K}_{\boldsymbol{T},j}$ と T_{TPij} の積は式 (9) の分母で用いることができる．また，$\eta^{S_K}_{\boldsymbol{T},\phi}$ は式 (9) の分子で用いることができる．

次に，$\boldsymbol{V}$ について，要素一つを抜いた各行 V_{BTij} の総乗を計算する．すなわち，$i = 1, 2, \cdots, K$ と $j = 1, 2, \cdots, M$ について $\eta^{S_M}_{\boldsymbol{V},j}$ を計算する．次に，$i = 1, 2, \cdots, K$ と $j = 1, 2, \cdots M$ について $\eta^{S_M}_{\boldsymbol{V},j}$ と V_{TPij} の積を計算し，最後に $i = 1, 2, \cdots, K$ について $\eta^{S_M}_{\boldsymbol{V},1}$ と V_{BTi1} との積を計算する．ここで，$\eta^{S_M}_{\boldsymbol{V},1}$ と V_{BTi1} との積は $\eta^{S_M}_{\boldsymbol{V},\phi}$ である．次に，列方向について，$\boldsymbol{V}$ の特定の 1 行を抜いたものの総乗を計算する．すなわち，$i = 1, 2, \cdots, K$ について*$\prod_{i' \in S_K \setminus \{i\}} \eta^{S_M}_{\boldsymbol{V},\phi}$ を計算する．さらに，$i = 1, 2, \cdots, K$ と $j = 1, 2, \cdots M$ について計算した $\eta^{S_M}_{\boldsymbol{V},j}$ と V_{TPij} の積と，*の積を計算する．これは，すなわち $\xi_{\boldsymbol{V},(i,j)}$ である．最後に，$\xi_{\boldsymbol{V},(i,j)}$ と V_{TPij} の積を計算する．これは，$\boldsymbol{V}$ の全要素の総乗である $\xi_{\boldsymbol{V},\phi}$ を計算することと等しい．以上の手順でループの前に計算を実行し，メモリに計算結果を保持しておくことで，式 (9) における η および ξ の値は各要素の更新で再利用できる．

同様に，$\boldsymbol{V}$ の更新についても更新前に ψ および ξ の値を計算をしておくことで，各要素ごとの更新における乗算回数を削減することができる．$\boldsymbol{V}$ の更新では，列ごとの総乗を求めていけばよい．大まかな手順は $\boldsymbol{T}$ の更新前の計算と同じである．

6.2 演算回数による処理コストの見積もり

計算結果を再利用した場合の処理コストについて述べる．$\boldsymbol{T}$ について，6.1 で述べた方法により，ループ前に η と ξ を計算すると，乗算回数は $KM^2+KN^2+K^2+KM-2K+1$ 回であり，ループのための乗算回数が $3KNM+4KN$ 回である．また，$\boldsymbol{V}$ について，更新前に ψ と ξ を計算すると，乗算回数が $KN^2+KM^2+K^2+KN-2N+1$ 回であり，ループのための乗算回数が $3KNM+4KM$ 回である．これを L ループすると考えると，4.1 の step3 における乗算回数は $L\{2K^2+K(N^2+M^2+6NM+5M+5N-4)+2\}$ である．step3 以外の各 step における各プロトコルの実行回数は 4.4 で述べた回数と同じである．最後に，実行全体の処理コストを各演算の回数のオーダーにより，表 7 に示す．オーダーのレベルで見ると，計算結果の再利用をして実行した改良方式と基本方式とで乗算の回数は変わらず，除算回数は $1/L$ になる．したがって，演算回数のオーダーで見ると基本方式から除算回数のみ削減されており，実行全体における改良になっているといえる．詳細は省略するが，基本方式に比べて係数は 2 倍になっている．

ただし，計算結果を再利用するために，各参加者が 1 回の更新の終了まで保持しておかなければならない値があるため，実行におけるメモリ使用量が増加する．

7. 値の桁数増加の抑制に関する見通し

問題点 2 の桁数の増加に対する解決策について検討する．

7.1 解決案 1

計算の途中で桁数が増加しても，設定した桁数からオーバーフローしないようにする工夫の一つに，浮動小数で値を表現するという方法がある．浮動小数は値を仮数部と指数部に分けて保持する方法である．例えば，10 進数の小数で $a = 3521.95$ という値があるとき，$a = 3.52195 \times 10^3$ のように表現する．このとき，仮数部が 352195 で，指数部が 3 となる．計算に際し，値を浮動小数で保持し，下位の桁を捨てていくことで最も価値のある上位の桁は常に保持される．

しかし，浮動小数で計算する方法にも問題がある．浮動小数における MPC の四則演算プロトコルは Aliasgari ら [16] が提案しているが，これによると，浮動小数での加算の処理コストが，通信量が $14B_i + 9B_d + (\log B_i)\log\log B_i + (B_i + 9)\log B_i + 4\log B_d + 37$ で，ラウンド数が $\log B_i + \log\log B_i + 27$ となっている．固定小数の MPC の場合，加算は各参加者がローカルに実行でき，処理コスト 0 であったのに対し，浮動小数の MPC では参加者間での通信があり，処理コストがかかる．その理由は，浮動小数の加算は 2 つのオペランドの桁をそろえて，加算を実行し，桁を戻すという操作が含まれているためである．

また，固定小数における四則演算のうち，加算と乗算に比べて除算の処理コストが特出して大きかった．しかし，浮動小数における四則演算では，各プロトコルの処理コストが全体的に大きくなる．したがって，改良方式に対して浮動小数演算を用いると，除算を削減する意味がなくなるので，不適当であると考えられる．

7.2 解決案 2

浮動小数でのプロトコルを用いると加算と乗算の処理コストが大きくなり，非効率であるので，固定小数での計算

方法で工夫をすることを考える．具体的には，改良方式のstep3 において上位に桁あふれが生じるタイミングで新たな固定小数の桁を設定する．

はじめに, 計算の精度すなわち表現できる最大の桁数 g を決める．g の大きさは固定とする．整数部分の桁数 e と小数部分の桁数 f の初期値を設定し, $e+f$ が最大の桁数以下になるようにする．e と f の値は $e+f=g$ を維持しながら計算の途中で変更してもよい．例として, $g=4$ として, 6.3×8.1 という乗算を考える．はじめ, $e=2, f=2$ に設定する．計算結果は $06.30\times 08.10=51.03$ となる．次に，この数と別の数との積を考える．例えば，$51.03\times 82.49=4209.4647$ となる乗算を考える．$g=4$ を維持するために上位の桁のみを残して下位の桁を切り，$e=4, f=0$ とする．積の値は 4209 となる．このとき，e は $2\rightarrow 4$ と変化したのに対し，f は $2\rightarrow 0$ と変化した．このように，乗算の実行において整数部分の桁数 e と小数部分の桁数 f を変化させて計算していくこととする．2 つのオペランドの乗算結果の桁数は高々，それぞれの桁数の和になり，2 つのオペランドの加算結果の桁数は高々 1 桁増える．したがって，複数の乗算と加算を実行するとき，行列 $\boldsymbol{X}, \boldsymbol{T}, \boldsymbol{V}$ のサイズがわかれば，乗算と加算の回数がわかり，計算途中で保持しておけばよい桁数と小数点の位置をずらすタイミングを予測することができる．

整数部分と小数部分の桁数を変化させながら計算する方法は，Catrina らのプロトコル [13] に整数と小数の桁を変化させることを組み合わせればよい．下位の桁を丸める方法として 3.4 で述べた TruncPr を用いれば実現可能である．したがって，この方法を本システムで適用できると考えられる．

8. 結論と今後の課題

非負値行列の解析に有用な NMF を，秘匿計算により実現する手法を提案した．また，実行における更新式の変形を行い，除算の実行回数を削減した効率的な方式についても提案した．更新式を変形して除算回数を削減する方式について，本稿では Euclid 距離に基づく更新式に適用させたが，Kullback-Leibler 距離に基づく更新式に対しても，適用可能である．

また，除算を削減した方式により発生する 2 つの問題について，途中の計算結果を再利用する方法と，固定小数の整数部分と小数部分の桁数を変化させる方法により解決案を示した．

今後の課題として，桁数を変化させていく方法を厳密に検討すること，また実装して実際のコストと安全性を評価することなどが挙げられる．

謝辞 本研究は，一般財団法人テレコム先端技術研究支援センター「SCAT 研究費助成」ならびに公益財団法人電気通信普及財団「情報通信技術に関する研究調査」の支援により推進しました．ここに深く御礼申し上げます．

参考文献

[1] Shamir, A. How to Share a Secret.: Communications of the ACM, Vol.22, No.11, pp.612-613, 1979.

[2] Gennaro, R. Rabin, M. and Rabin, T.: Simplified VSS and Fast-track Multiparty Computations with Applicants to Threshold Cryptography. In: 17th Annual ACM Symposium on Principles of Distributed Computing, pp.101-111, 1998.

[3] Nishide, T. and Ohta, K.: Multiparty Computation for Interval, Equality, and Comparison without Bit-Decomposition Protocol. proc. PKC 2007, LNCS, Vol.4450, pp.343-360, 2007.

[4] Catrina, O. and Hoogh, S.: Improved Primitives for Secure Multiparty Integer Computation. Proc. SCN 2010, LNCS, pp.182-199, 2010.

[5] Lee, D. and Seung, H.: Algorithms for Non-negative Matrix Factorization.: Advances in neural information processing systems 2001, pp.556-562, 2001.

[6] Lee, D. and Seung, H.: Learning the parts of objects by non-negative matrix factorization.: Nature, Vol. 401, No.6755, pp.788-791, 1999.

[7] 澤田宏 (2012).:" 非負値行列因子分解 NMF の基礎とデータ／信号解析への応用", 電子情報通信学会誌,Vol. 95, No.9, pp.829-833, 2012.

[8] 石井健一郎, 上田修功.:" 続・わかりやすいパターン認識 教師なし学習入門". オーム社, pp.113-120, 2014.

[9] Yang, X. and Liu, Z.: The Security Matrix Factorization Agreement with Two Parties.: Computational Intelligence and Security 2009, pp.218-221,2009.

[10] Li, T. Gao, C. and Du, J.: A NMF-Based Privacy-Preserving Recommendation Algorithm. 2009 First International Conference on Information Science and Engineering, pp.754-757, 2009.

[11] Nikolaenko, V. Ioannidis, S. Weinsberg, U. Joye, M. Taft, N. and Boneh, D.: Privacy-Preserving Matrix Factorization. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp.801-812, 2013.

[12] Yao, A. : Protocol for secure computations.: Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, pp.160-164, 1982 .

[13] Catrina, O. and Saxena, A.:Secure Computation With Fixed-Point Numbers.:Sion, R,(ed) FC2010, LSCS,Vol.6052, pp.35-50, 2010.

[14] SecureSCM. Security analysis technical report D9.2. SecureSCM.`https://www1.cs.fau.de/filepool/publications/octavian_securescm/SecureSCM-D.9.2.pdf`

[15] Damgård, I. Pastro, V. Smart, N. and Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption. Advances in Cryptology-CRYPTO 2012, pp.643-662, 2012.

[16] Aliasgari, M. Blanton, M. and Zhang, Y. and Steele, A.: Secure Computation on Floating Point Numbers. NDSS, 2013.