計算工学講演会論文集 Vol. 30 (2025 年 6 月)　　日本計算工学会

テンソルトレイン分解を用いた 大規模疎行列の反復解法の効率化の研究

Solving Large-Scale Sparse Liner Systems Using Tensor Train Decomposition

鳥井公平 [1] 小野謙二 [2]
Kohei Torii and Kenji Ono

[1]九州大学 大学院システム情報科学研究院 情報理工学専攻 (E-mail: torii.kohei.502@s.kyushu-u.ac.jp)
[2]九州大学 情報基盤研究開発センター （E-mail: keno@cc.kyushu-u.ac.jp）

We investigate the efficiency of iterative methods for large-scale sparse linear systems whose coefficient matrices originate from partial differential equations (PDEs). In this study, we incorporate a tensor train decomposition (TTD) into the Successive Over-Relaxation (SOR) method. We then evaluate the proposed approach in terms of convergence, computational time, and accuracy across various problem settings, each featuring different solution distributions. The results demonstrate the potential advantages of combining TTD with the SOR method for large-scale PDE-based simulations.

Key Words : *Tensor Train Decomposition,*

1. はじめに

近年，半導体の微細化に限界が見え始めたことでポストムーア時代へと突入し，計算資源をいかに効率的に活用するかが大きな課題となっている [1]．特に，大規模データを扱う数値シミュレーションや HPC（High-Performance Computing）の分野では，従来のクロック周波数向上に依存した性能向上が難しくなり，メモリ帯域の制約が顕著化している．こうしたオンメモリ計算のボトルネックを緩和するためには，大容量データの圧縮や，SIMD 機構・マルチコアによる並列化など，ハードウェア特性を最大限に活かす手法の開発が重要なテーマとなっている．数値シミュレーションでは，問題の高次元化や空間分割数の増大に伴い，大規模な疎行列を扱う需要が増え，メモリ帯域の制約が計算性能の深刻なボトルネックとなりやすい．こうした問題を背景として，テンソルトレイン分解は [2]，Oseledets によって提唱された高次元テンソルの低ランク近似手法であり，線形代数の数値解法や PDE 解法への応用 [3]，さらには並列計算環境における実装最適化 [4] について研究されている．そこで本研究では，テンソルトレイン分解によるデータ圧縮と，圧縮形式のまま演算を行うアプローチを組み合わせることで，反復法の収束過程に与える影響を調べた．具体的には，ポアソン方程式 $\nabla^2\varphi = b$ を離散化して得られる大規模疎行列を係数行列とする連立一次方程式を対象とし，従来の SOR 法にテンソルトレイン分解を組み込んだ TT-SOR 法を導入した．この手法では，逐次解 φ や右辺項 b を TT 形式に圧縮したうえで更新演算を進めるため，従来法と比較して演算量とメモリアクセス量の比（Byte/Flop）の削減が期待できる．右辺項 b のランク数が異なる三つの問題を用意し，TT-SOR 法と従来の SOR 法について，圧縮効率，残差の収束特性，計算時間，数値解の様子などを比較検証した．

2. テンソルトレイン分解の概要と利点

(1) テンソルトレイン分解の概要

テンソル分解とは，高次元テンソルをより少ないパラメータで表現するための手法である．代表的なものとして，CP 分解（Canonical Polyadic 分解）[5] や Tucker 分解 [6] などがある．

CP 分解とは，N 階テンソル $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ をランク 1 テンソルの和として近似する方法であり，具体的には，式 (1) のように表せる：

$$\mathcal{X} \approx \sum_{r=1}^{R} a_r^{(1)} \circ a_r^{(2)} \circ \cdots \circ a_r^{(N)} \tag{1}$$

ここで，$\circ$ は外積を示し，$a_r^{(k)} \in \mathbb{R}^{I_k}$ はテンソルの k 番目モードに対応する要素の列ベクトルである．R は CP ランクと呼ばれ，テンソルをいくつのランク 1 テンソルで近似するかを示すパラメータである．CP 分解は概念的にシンプルで分かりやすいが，アルゴリズムが局所解に陥りやすく，収束が不安定になりやすいことや，ランク R の選択が NP 困難であるといった問題点がある．

Tucker 分解は，式 (2) のように，コアテンソルと各モードごとの因子行列との積によってテンソルを表す手法である．

$$\mathcal{X} \approx \mathcal{G} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 \cdots \times_N U^{(N)} \tag{2}$$

ここで，$\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ はコアテンソル，$U^{(k)} \in \mathbb{R}^{I_k \times R_k}$ はモード k に対応する因子行列を表し，$\times_j$ は j 番目モード方向でのテンソルと行列の積（n-mode 積）を意味する．

Tucker 分解は，コアテンソルが各モード方向に縮約された形でデータの本質を捉えており，CP 分解よりも安定して最適化しやすい場合があるが，コアテンソル $\mathcal{G}$ が N 次元なので，N が増えると格納が困難になること

や，ランクの組 $(R_1, \ldots, R_N)$ を決める必要があるといった問題がある．

これらに対し，テンソルトレイン分解は高次元化に強い圧縮性能を持ち，近年注目されているテンソル分解の手法である．テンソルトレイン分解では，N 次元テンソルを複数の 3 次元コアテンソルに鎖状（トレイン状）に連結して表し，式 (3) のように書ける．

$$\mathcal{X}_{\mathrm{TTD}} = \left[G^{(1)} G^{(2)} \ \cdots \ G^{(N)} \right] \tag{3}$$

ここで，$G^{(k)} \in \mathbb{R}^{r_{k-1} \times I_k \times r_k}$ はコアテンソルであり，一般に $r_1 = r_n = 1$ である．テンソルトレイン分解は，CP 分解や Tucker 分解に比べ，高次元でもメモリ使用量の急増を抑えやすい．ランク構造が階層的で扱いやすく，数値的にも安定しやすいという利点があるほか，後述するように TT-format 上での計算が可能である．問題の細分化，高次元化が求められる大規模シミュレーションにおいて，こうした特長を活かして大規模データの圧縮や数値解法への応用を図るため，テンソルトレイン分解を用いる．

(2) TT-format

d 階テンソルをテンソルトレイン分解することで，d 個のコアテンソルとして保持することを”TT-format”と呼び，展開することなくテンソル演算を行うことができる．d 階テンソル $\mathcal{X}$ の $(i_1, i_2, \ldots i_d)$ の要素を取り出す場合は式 (4) のように表す．

$$x_{i_1, i_2, \ldots, i_d} = G^{(1)}[:, i_1, :] G^{(2)}[:, i_2, :] \cdots G^{(d)}[:, i_d, :] \tag{4}$$

各 $G^{(k)}$ は $\mathbb{R}^{r_{k-1} \times n_k \times r_k}$ のサイズを持ち，$G^{(k)}[:, i_k, :]$ は「第 i_k 番目のスライスを表す．式 (4) によって，大規模テンソルを完全に展開することなく，任意の要素へのアクセスが可能となる．

(3) TT-SVD

テンソルトレイン分解を求めるアルゴリズムはいくつか存在するが，ここではその代表的なものとして，TT-SVD（Tensor Train SVD）を使用する．TT-SVD は truncated-SVD を用いてテンソルを段階的に近似する手法であり，その大まかな流れを Algorithm-1 に示す．TT-SVD では特異値を切り捨てる際に，あらかじめ閾値を指定して条件を満たすように小さい特異値を切り捨てる方法と，TT-rank を直接指定してそれを超える特異値を切り捨てる方法がある．閾値を指定する場合，TT-SVD の精度を保ちながら分解できる利点がある．一方，TT-rank を指定する場合は，メモリや計算量を事前に見積もりやすいという利点がある．本研究では，閾値 ε を式 (5) と設定する．

$$\frac{\sum_{j=1}^{r} \sigma_j^2}{\sum_{j=1}^{n} \sigma_j^2} < 1 - \varepsilon \tag{5}$$

TT-SVD では，各ステップでテンソルを unfold(行列化) して truncated-SVD を行い $X = U \Sigma V^T$ とする．**Algorithm 1** 中の R_k は，truncated-SVD で得られた非ゼロ特異値の数，すなわち TT-rank を示す．TT-rank が小さいほど、元のテンソルがより少ない情報で表現されていることを意味する．得られた $U^{(k)}$ や $\Sigma^{(k)}$，$V^{(k)}$ を適切な reshape を行い，テンソルとして再表現することで，各コアテンソル $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ を作り，左から右へと段階的に処理する．最終的に $\mathcal{G}^{(k)}$ を連鎖させれば元のテンソルを近似するテンソルトレイン分解が完成する．このように TT-SVD は，一度に大きなテンソル全体を取り扱わず，各モードごとに trunated-SVD を実行することで，高次元データに対しても比較的安定した精度と計算量で TT-format を獲得できる利点を持つ．

Algorithm 1 TT-SVD Algorithm

Input: $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, $\varepsilon > 0$
Output: $\{\mathcal{G}^{(n)}\}_{n=1}^{N}$
$R_0 \leftarrow 1, \quad R_N \leftarrow 1$
$\mathcal{B} \leftarrow \mathcal{A}$
for $k = 1$ to $N - 1$ **do**
  $(U, \Sigma, V) \leftarrow$ truncated-SVD$(\mathbf{B}_{(2)}, \varepsilon)$
  $R_k \leftarrow$ (number of non-zero singular values of Σ)
  $\mathcal{G}_k \leftarrow \mathrm{fold}_{(R_{k-1}, I_k, R_k)}(U_{:, 1:R_k})$
  $\mathcal{B} \leftarrow \mathrm{fold}_{(R_k, I_{k+1}, \ldots, 1)}(\Sigma_{1:R_k, 1:R_k} V^T_{:, 1:R_k})$
end for
$\mathcal{G}_N \leftarrow \mathcal{B}$

(4) テンソルトレイン分解によるメモリ圧縮

d 階テンソル $\mathcal{X} \in \mathbb{R}^{n \times n \times \cdots \times n}$ の要素をすべて直接格納する場合，サイズが n^d 個の要素を必要とする．一方、テンソルトレイン分解では，式 (6) のように d 個のコアテンソル $\mathcal{G}^{(k)}$ を用いて

$$\mathcal{X} \approx \mathcal{G}^{(1)} \times \mathcal{G}^{(2)} \times \cdots \times \mathcal{G}^{(d)}, \tag{6}$$

と分解する．ここで各コア $\mathcal{G}^{(k)}$ は $\mathcal{G}^{(k)} \in \mathbb{R}^{r_{k-1} \times n \times r_k}$, $r_0 = r_d = 1$ という形状をもつので，そのサイズは式 (7) と表せる．

$$\sum_{k=1}^{d} (r_{k-1} \times n \times r_k) \tag{7}$$

特に，ランクを一様に r と仮定すると，各コアのパラメータ数はおよそ $r \cdot n \cdot r = nr^2$ 個であり，これが d 個あるため，

$$\sum_{k=1}^{d} (r_{k-1} \times n \times r_k) \ \approx \ d \cdot n \cdot r^2 \tag{8}$$

したがって，元の n^d 個の要素をもつテンソルを，約 $\mathrm{O}(d\, n\, r^2)$ 個の要素に圧縮できる．

3. TT-SOR アルゴリズム

(1) 離散化

本章では，ラプラス方程式を離散化して連立方程式を導出する過程と，本実験で使用したテンソルトレイン分解を用いた SOR 法について説明する．三次元ラプラス方程式 (9) において φ は領域内で求めたい未知のスカラー関数にあたる．

$$\nabla^2 \varphi \ = \ \frac{\partial^2 \varphi}{\partial x^2} \ + \ \frac{\partial^2 \varphi}{\partial y^2} \ + \ \frac{\partial^2 \varphi}{\partial z^2} \ = \ 0 \tag{9}$$

格子点 (x_i, y_j, z_k) に対して，式 (9) 中の二次偏微分の項は，中心差分法により近似すると式 (10) が導出される．(※式中の添え字は省略形である)

$$\varphi_{i+1} + \varphi_{i-1} + \varphi_{j+1} + \varphi_{j-1} + \varphi_{k+1} + \varphi_{k-1} - 6\varphi_{i,j,k} = 0 \quad (10)$$

分割数を (n_x, n_y, n_z) とすると，$1 \le i \le n_x$, $1 \le j \le n_y$, $1 \le k \le n_z$ であり，$n_x n_y n_z$ 個の方程式が得られる．式 (10) から分かるように，各格子点とその近傍の点のみ係数が非ゼロになるので，解くべき連立方程式の係数行列は，大規模疎行列である．この連立方程式を解くために，本実験では反復解法である SOR 法を用いる．

(2) TT-SOR 法

Algorithm 2 は，SOR 法にテンソルトレイン分解を加えた TT-SOR 法のアルゴリズムである．TT-SOR 法は，まず初期解 $\varphi^{(0)}$ と右辺項 b をテンソルトレイン分解を施す．そして，各反復で求まる逐次解 $\varphi^{(n+1)}$ にテンソルトレイン分解を行い，圧縮された TT-format を用いて残差計算を計算する点が特徴である．TT-rank が比較的小さい範囲で保たれる場合には大規模化してもメモリ使用量を抑えられる可能性がある．ここでは，収束性や計算時間，誤差などを通常の SOR 法を比較し，truncated-SVD の閾値を大きくして逐次解の TT-rank を削減した場合の収束可否，誤差と TT-rank のトレードオフについて検証する．

Algorithm 2 TT-SOR Algorithm

initial solution: $\varphi^{(0)} \leftarrow 0$
TTD($\varphi^{(0)}$), TTD(b) ▷ Apply TTD to $\varphi^{(0)}$ and b
for $n = 1$ to max_iteration **do**
  for each (i, j, k) in the interior domain **do**

$$r_{i,j,k} = \varphi_{i+1} + \varphi_{i-1} + \varphi_{j+1} + \varphi_{j-1} + \varphi_{k+1} + \varphi_{k-1} - 6\varphi_{i,j,k} - b_{i,j,k}$$

$$\Delta\varphi^{(n+1)}_{i,j,k} = \frac{1}{6} r_{i,j,k} - \varphi^{(n)}_{i,j,k}$$

$$\varphi^{(n+1)}_{i,j,k} \leftarrow \varphi^{(n)}_{i,j,k} + \omega\, \Delta\varphi^{(n+1)}_{i,j,k}$$

  end for
  Compute the residual $r^{(n+1)}_{i,j,k}$
  TTD($\varphi^{(n+1)}$)
  if $r^{(n+1)}_{i,j,k}$ < tolerance **then** ▷ tolerance = 10^{-7}
    break ▷ Convergence check
  end if
end for

4. 問題設定

本研究では，TT-SOR 法と従来の SOR 法とを比較し数値実験を行う．三次元の偏微分方程式 $\nabla^2\varphi = b$ について，(1) $b = 0$ の場合，(2) b の TT-rank が 3 である (特異値を切り詰めない TTD を想定) 場合，(3) b の TT-rank が大きいと予想される場合の 3 種類の問題を扱う．以下では，それぞれの問題について詳しく説明する．

(1) $b = 0$ の場合

扱う偏微分方程式はラプラス方程式 $\nabla^2\varphi = 0$ である．この問題について式 (11) のディリクレ条件を与える．

$$\varphi(x, y, z) = \begin{cases} \sin(\pi x)\,\sin(\pi y) & \text{on } (x, y, 0) \\ \sin(\pi x)\,\sin(\pi y) & \text{on } (x, y, 1) \\ 0 & \text{on other boundaries} \end{cases} \quad (11)$$

このとき，厳密解 $\varphi(x, y, z)$ は式 (12) のように表せる．

$$\varphi(x, y, z) = \sin(\pi x)\sin(\pi y)\frac{\sinh(\sqrt{2}\pi z) + \sinh(\sqrt{2}\pi(z-1))}{\sinh(\sqrt{2}\pi)} \quad (12)$$

この厳密解は，$\sin(\pi x)\sin(\pi y)$ に z のみの関数が掛かった形であり，変数分離型で表すことができることから，ランク 1 テンソルであるといえる．

(2) b の TT-rank が 3 である場合

扱う偏微分方程式はポアソン方程式 $\nabla^2\varphi = b$ である．境界条件は式 (11) のディリクレ条件を与える．b の TT-rank が小さい場合として，b を式 (13) とした．

$$\begin{aligned} b = {} & \sin(2\pi x)\sin(2\pi y)\sin(2\pi z) \\ & + \sin(3\pi x)\sin(3\pi y)\sin(3\pi z) \\ & + \sin(4\pi x)\sin(4\pi y)\sin(4\pi z) \end{aligned} \quad (13)$$

ここで，b は 3 つの積形式で表されていることから，離散化式 $b[i, j, k]$ の TT-rank は 3 である．

(3) b の TT-rank が大きい場合

扱う偏微分方程式はポアソン方程式 $\nabla^2\varphi = b$ である．周期境界条件の下で，右辺項 b を乱流の速度ベクトル $\mathbf{V}$ の発散とすると，式 (14) のように書ける．なお，本研究で用いる速度ベクトル $\mathbf{V}$ は，別途実行した乱流シミュレーションから取得した結果を利用している．ここで $\mathbf{V}$ は，各座標 (x, y, z) における乱流の速度ベクトルを表す．本式は，非圧縮性流体に対するナビエ・ストークス方程式の圧力を求めるためのポアソン方程式に相当する．この場合は，ポアソン方程式の右辺項 b が複雑な構造をとるため，厳密解も単純な外積や線形結合で表現することが難しく，複雑な構造をとることが考えられる．

$$\nabla^2\varphi = \frac{1}{\Delta t}(\nabla \cdot \mathbf{V}) \quad (14)$$

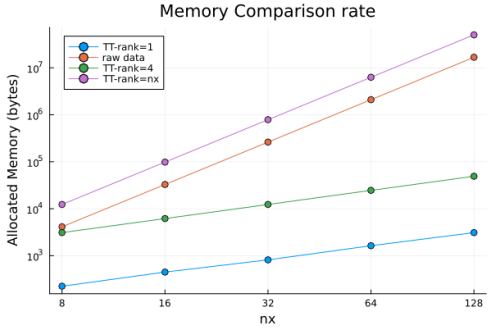
以上の 3 つの問題設定について，SOR 法と TT-SOR 法の振る舞いを比較することで，その性能を定量的に評価し，複雑なシミュレーションに対しての応用について考察する．

5. 実験結果

(1) TTD の圧縮性能

式 (7)(8) で説明したように，倍精度のデータを持つ厳密解にテンソルトレイン分解を行うと，空間量が $O(n^d)$ から $O(dnr^2)$ へと圧縮される．図-1 は，テンソルトレ

イン分解をしない場合と，TT-rank= $1, 4, nx$ のテンソルをテンソルトレイン分解した場合の分割数 nx とメモリ量の関係を表したグラフである．



図–1 Memory requirement for three problems

また，表-1 は，TT-rank = 1 のテンソルに対してテンソルトレイン分解を行った際の圧縮率を示している．圧縮率は式 (15) により算出される．テンソルの TT-rank が小さいほど，テンソルトレイン分解による圧縮性能が高くなる傾向がある．さらに，テンソルの次元数やサイズが大きくなるほど，圧縮によって削減できる情報量も増加し，圧縮率がより高くなることが確認された．

$$\text{Reduction ratio} = \frac{\text{size(TTD tensor)}}{\text{size(original tensor)}} \qquad (15)$$

表–1 Memory reduction for rank-1

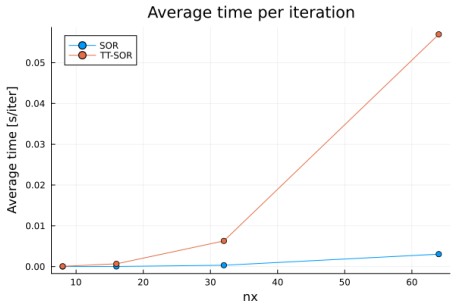
nx	8	16	32	64	128
Original size	4k	32k	262k	2,097k	12,777k
TT-format	0.336k	0.528k	0.912k	1.680k	3.216k
Reduction ratio	0.0809	0.0161	0.0035	0.0008	0.0002

(2) 計算時間比較

図-2 は b の TT-rank が 3 の問題における 1 反復中にかかる時間を表している．TT-SOR 法は分割数が大きくなるにつれて指数関数的に計算時間が増大してしまう問題がある．計算時間の詳細を表-2 に示す．TTD は逐次解のテンソルトレイン分解にかかった時間であり，SOR-update は反復中の残差計算などかかった時間を表している．

表–2 Computational Time Details(msec)

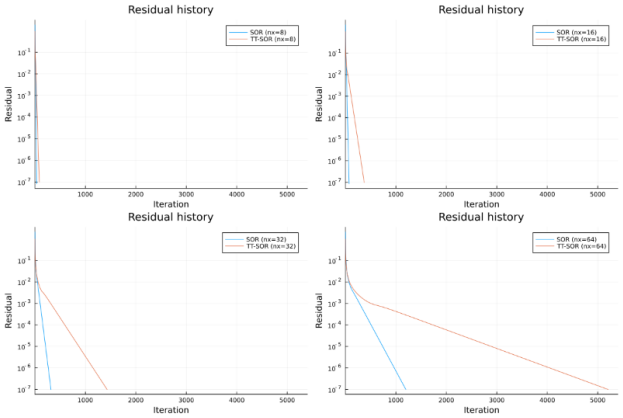
nx	8	16	32	64
average-time	2.21	1.84	7.4	63
SOR-update	0.040	0.59	4.95	41.4
TTD	0.038	0.33	1.33	20.3



図–2 SOR vs. TT-SOR: Time Comparison

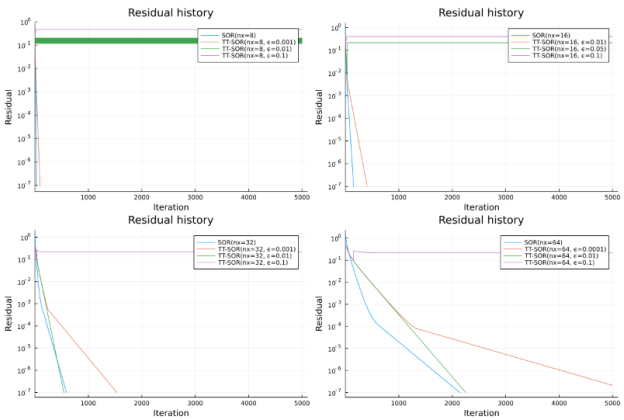
(3) 残差の収束特性

以下に各問題での残差の収束特性を示す．図-3 は $b = 0$ 問題の残差の収束特性を記録した．分割数を増やすと収束までの回数が増大し，その特徴は特に TT-SOR 法に顕著にみられた．



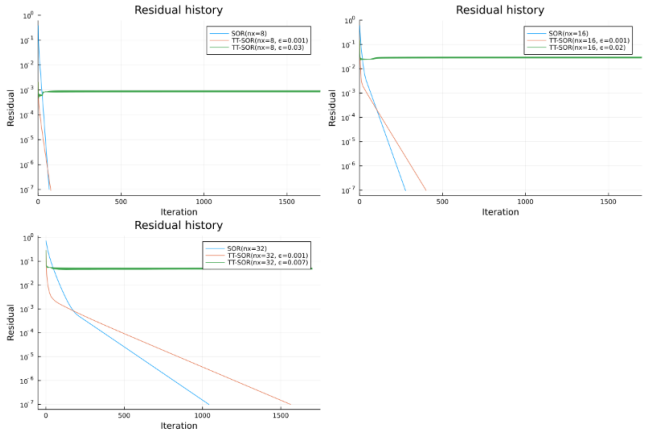
図–3 Residual Convergence (b = 0)

図-4 は，b の TT-rank が小さい問題の残差の収束特性を示す．TT-SVD の閾値を変えて TT-rank を削った場合に，収束するものとしないものが現れた．特異値を削りつつ収束が可能なときは，収束までの回数が大幅に減少した．



図–4 Residual Convergence (b TT-rank 3)

図-5 は，b の TT-rank が大きい問題の残差の収束特性を示す．閾値を大きくして TT-rank を一つでも削ると，TT-SOR 法の残差は発散した．したがって，厳密解が複雑な構造をとる場合においては，特異値を切り詰めつつ反復解法を実現させることは難しいことが分かった．



図–5 Residual Convergence (b high TT-rank)

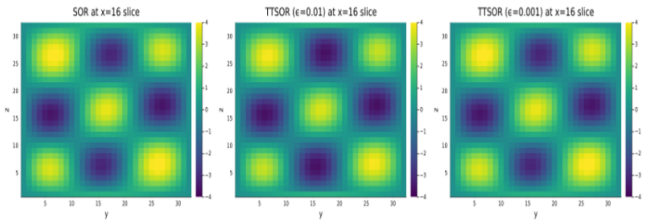
(4) TT-SOR 法の閾値と数値解

図-4 の TT-rank を削りつつ，残差が収束した場合における数値解の SOR 法との誤差を測定した．表-3 は分割数 $nx = 32$ とし，TT-SVD の閾値を変えた時の SOR 法と TT-SOR 法の数値解の差を示している．閾値を大きくしたところ，反復回数が大幅に小さくなることが分かった．しかし，特異値を切り詰めたことで，SOR 法の数値解との誤差は大きくなった．分割数当たりの誤差の大きさは，$\varepsilon = 0.01$ のとき 1.2×10^{-3}，$\varepsilon = 0.001$ のとき 3.6×10^{-9} となった．

表–3 Relationship between TT-SVD Threshold and Error

	$\varepsilon = 0.01$	$\varepsilon = 0.001$
Num. of iterations	542	1533
Sum of difference	3.978×10^{1}	1.184×10^{-4}
Diff. per cell	1.2×10^{-3}	3.6×10^{-9}
TT-rank	3	4

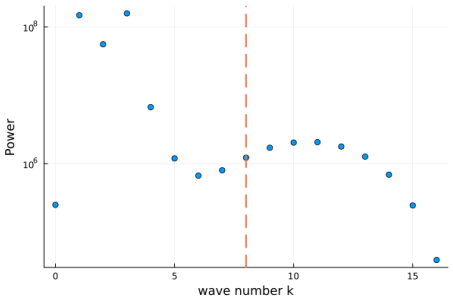
図-6 は数値解の分布を可視化したヒートマップである．解の分布は大まかに特徴をとらえていること分かる．したがって，厳密な精度を要求しないシミュレーションにおいては，TT-SOR 法を利用し，解の近似を行いつつ数値解を求める手法が有用であると考える．



図–6 Numerical Solution Distribution

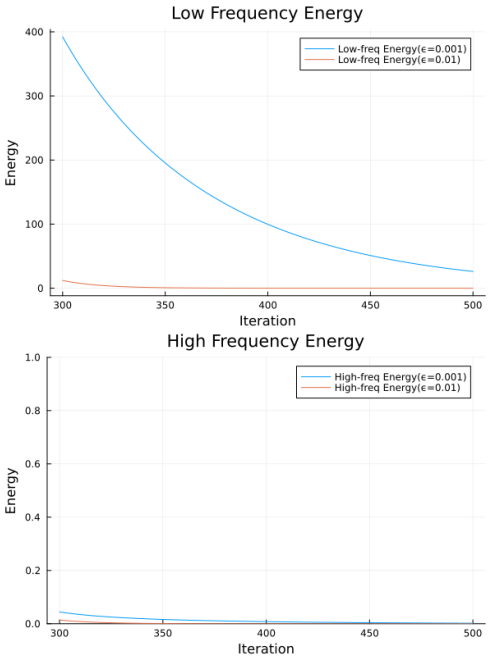
(5) TT-SOR 法の閾値が数値解の波数成分に与える影響

二種類の TT-SOR 法（$\varepsilon = 0.001$ および $\varepsilon = 0.01$）について，各反復における逐次解に対して三次元空間フーリエモード分析を行う．図-7 は，反復の一回目の逐次解 $\varphi^{(1)}$ に対して，各波数成分のフーリエ係数を計算し，低波数部分と高波数部分に分けた様子である．



図–7 wavenumber distribution of $\varphi^{(1)}$

図-8 は，TT-SOR 中の反復回数が 300〜500 回の範囲における高波数成分（左半分）および低波数成分（右半分）の時間変化を示している．



図–8 Evolution of Wavenumber Components

$\varepsilon = 0.001$ の場合，高波数成分は全般的に小さい値を保つ一方で，低波数成分は容易には減少しないことが確認された．これに対し，$\varepsilon = 0.01$ の場合は，高波数成分と低波数成分の双方とも $\varepsilon = 0.001$ に比べて，迅速に減少する傾向が観測された．閾値を大きく設定して特異値を切り詰めることで，低波数成分が素早く減少するようになり，TT-SOR 法の残差の収束性も改善したと考えられる．

6. 結論

本研究で提案した TT-SOR 法は，従来の SOR 法に比べて収束性が改善する場合があることが分かった．これは，TT-SVD により特異値を切り詰め，低波数成分を効率的に除去していることに起因すると考えられる．今後は，BiCGstab 法やナビエ・ストークス方程式の解法への適用，さらには，さまざまな問題設定における TT-SOR 法の振る舞いを多角的に検証し，効果を実証していく予定である．

参考文献

[1] Leiserson, C. E., Thompson, N. C., et al. :There ’ s Plenty of Room at the Top: What Will Drive Computer Performance After Moore ’ s Law. Science, 2020, 368(6495). https://doi.org/10.1126/science.aam9744

[2] I. V. Oseledets, :Tensor-train decomposition, SIAM Journal on Scientific Computing, vol. 33, no. 5, pp. 2295–2317, 2011. https://doi.org/10.1137/090752286

[3] Sergey V. Dolgov, Dmitry V. Savostyanov, :Alternating Minimal Energy Methods for Linear Systems in Higher Dimensions. SIAM Journal on Scientific Computing, 36(5), A2248–A2271. 2014

[4] S. Xie, A. Miura and K. Ono, :Error-bounded Scalable Parallel Tensor Train Decomposition,2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), St. Petersburg, FL, USA, pp. 345-353. 2013

[5] Hitchcock, Frank L, :The expression of a tensor or a polyadic as a sum of products. Journal of Mathematics and Physics 6.1-4: 164-189, 1927.

[6] L. R. Tucker, :Some mathematical notes on three-mode factor analysis, Psychometrika, vol. 31, no. 3, p. 279 – 311, 1966.