3 項テンソル和に対する最大／最小特異値計算に関する研究

大橋 あすか　　　指導教員：曽我部 知広

1　はじめに

本研究では，3 つの正方行列 $A \in \mathbb{R}^{\ell\times\ell}$, $B \in \mathbb{R}^{m\times m}$, $C \in \mathbb{R}^{n\times n}$ における 3 項テンソル和

$$T := I_n \otimes I_m \otimes A + I_n \otimes B \otimes I_\ell + C \otimes I_m \otimes I_\ell, \quad (1)$$

に対する最大・最小特異値計算法を考える．3 項テンソル和は，定数係数の 3 次元偏微分方程式 (以降，PDE) に対する離散化から得られる線形方程式の係数行列として現れ，その最大・最小特異値は離散化で得られた線形方程式の誤差解析に役立つ．

ここで，(1) における記号“$\otimes$”はテンソル積を表し，$\otimes : \mathbb{R}^{m\times n} \times \mathbb{R}^{p\times q} \to \mathbb{R}^{mp\times nq}$ となる演算で，$M = (m_{ij}) \in \mathbb{R}^{m\times n}, N \in \mathbb{R}^{p\times q}$ に対して

$$M \otimes N := \begin{pmatrix} m_{11}N & \cdots & m_{1n}N \\ \vdots & \ddots & \vdots \\ m_{m1}N & \cdots & m_{mn}N \end{pmatrix} \in \mathbb{R}^{mp\times nq}$$

と定義される．テンソル積の定義より，3 項テンソル和 T は ℓmn 次正方行列になり，その所要メモリは，$n = \ell = m$ に関して $O(n^4)$ である．

一般に特異値計算に用いられる特異値分解を用いて，3 項テンソル和の最大・最小特異値を求めると，所要メモリは $O(n^6)$ となり，$n = 100$ に関して約 8TB のメモリが必要となるため，通常の方法を用いることは難しい．

そこで本研究では，行列の最大・最小特異値を求める反復法の 1 つである Lanczos 2 重対角化法に着目し，3 項テンソル和に関して簡単な実装が可能になるよう改良を加えたアルゴリズムと，そのアルゴリズムの収束を高速化する初期値を提案する．

2　Lanczos 2 重対角化法

3 項テンソル和 T に対して通常の Lanczos 2 重対角化法 [1] を適用すると，Algorithm 1 が得られる．Lanczos 2 重対角化法の利点として，3 項テンソル和に関する演算がベクトルとの積のみであるという点が挙げられる．3 項テンソル和とベクトルとの積は，Algorithm 1 における波線部分であり，その演算に関しては，行列 T の非ゼロ成分を必要とするため，Algorithm 1 の所要メモリは $O(n^4)$ となる．つまり，$n = 100$ に関して約 2.4GB のメモリが必要となる．

Algorithm 1 をそのまま実行すると，行列 A, B, C から行列 T の非ゼロ成分を構成する必要があるため，実装が複雑になる．

そこで，実装の簡単化のために次に説明する 3 階のテンソルを導入する．

Algorithm 1 Lanczos 2 重対角化法

初期ベクトル $\boldsymbol{p}_1 \in \mathbb{R}^{\ell mn}(\|\boldsymbol{p}_1\|_2 = 1)$ を選択する．
$\boldsymbol{q}_1 := T\boldsymbol{p}_1$; $\alpha_1 := \|\boldsymbol{q}_1\|_2$;
$\boldsymbol{q}_1 := \boldsymbol{q}_1/\alpha_1$;
for until convergence **do**
　$\boldsymbol{r}_i := T^{\mathrm{T}}\boldsymbol{q}_i - \alpha_i\boldsymbol{p}_i$; $\beta_i := \|\boldsymbol{r}_i\|_2$;
　$\boldsymbol{p}_{i+1} := \boldsymbol{r}_i/\beta_i$;
　$\boldsymbol{q}_{i+1} := T\boldsymbol{p}_{i+1} - \beta_i\boldsymbol{q}_i$; $\alpha_{i+1} := \|\boldsymbol{q}_{i+1}\|_2$;
　$\boldsymbol{q}_{i+1} := \boldsymbol{q}_{i+1}/\alpha_{i+1}$;
end for

3　3 階のテンソル

3 階のテンソル（以降，テンソル）[2] は 3 次元配列を意味し，テンソル $\mathcal{X}$ は $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ と表される．テンソルと行列との積としてモード積“$\times_k$”$(k = 1, 2, 3)$ があり，テンソル $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ に関する各モード積の定義は，行列 $M \in \mathbb{R}^{P\times I}$ に対して，

$$(\mathcal{X} \times_1 M)_{pjk} := \sum_{i=1}^{I} x_{ijk}m_{pi}$$

であり，行列 $M \in \mathbb{R}^{P\times J}$ に対して，

$$(\mathcal{X} \times_2 M)_{ipk} := \sum_{j=1}^{J} x_{ijk}m_{pj}$$

であり，行列 $M \in \mathbb{R}^{P\times K}$ に対して，

$$(\mathcal{X} \times_3 M)_{ijp} := \sum_{k=1}^{K} x_{ijk}m_{pk}$$

である．ただし，$i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, J$, $k = 1, 2, \ldots, K$, $p = 1, 2, \ldots, P$ である．

また，テンソルをベクトルに変換する vec 作用素 $\mathrm{vec} : \mathbb{R}^{\ell\times m\times n} \to \mathbb{R}^{\ell mn}$ がある．反対に，ベクトルをテンソルに変換する逆 vec 作用素を vec^{-1} と表す．そして，テンソルに対するノルムは成分ごとの積“$*$”を用いて，$\|\mathcal{X}\| := \sqrt{\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}(\mathcal{X} * \mathcal{X})_{ijk}}$ と定義される．

4　本研究

まず Algorithm 1 における波線部分について考える．ベクトル $\boldsymbol{p}_i, \boldsymbol{q}_i$ に逆 vec 作用素を作用させると，$\mathcal{P}_i = \mathrm{vec}^{-1}(\boldsymbol{p}_i), \mathcal{Q}_i = \mathrm{vec}^{-1}(\boldsymbol{q}_i)$ となり，テンソル $\mathcal{P}_i \in \mathbb{R}^{\ell\times m\times n}, \mathcal{Q}_i \in \mathbb{R}^{\ell\times m\times n}$ が得られる．

ここで，テンソル積，vec 作用素，モード積の関係に着目すると，定義より

$$(I_n \otimes I_m \otimes A)\mathrm{vec}(\mathcal{P}_i) = \mathrm{vec}(\mathcal{P}_i \times_1 A),$$
$$(I_n \otimes B \otimes I_\ell)\mathrm{vec}(\mathcal{P}_i) = \mathrm{vec}(\mathcal{P}_i \times_2 B),$$
$$(C \otimes I_m \otimes I_\ell)\mathrm{vec}(\mathcal{P}_i) = \mathrm{vec}(\mathcal{P}_i \times_3 C)$$

と表されるため，波線部分は

$$T\mathrm{vec}(\mathcal{P}_i) = \mathrm{vec}(\mathcal{P}_i \times_1 A + \mathcal{P}_i \times_2 B + \mathcal{P}_i \times_3 C),$$
$$T^{\mathrm{T}}\mathrm{vec}(\mathcal{Q}_i) = \mathrm{vec}(\mathcal{Q}_i \times_1 A^{\mathrm{T}} + \mathcal{Q}_i \times_2 B^{\mathrm{T}} + \mathcal{Q}_i \times_3 C^{\mathrm{T}})$$

となる．これらの計算については行列 A, B, C の非ゼロ成分をそのまま利用すればよいため，Algorithm 1 における波線部分に比べると，実装は簡単化され，その所要メモリはテンソルの所要メモリと一致して，$O(n^3)$ となる．

Algorithm 2 には，Algorithm 1 に対して逆 vec 作用素を作用させて得られた “テンソル空間上の Lanczos 2 重対角化法”（提案手法）のアルゴリズムを示す．

Algorithm 2 テンソル空間上の Lanczos 2 重対角化法

初期テンソル $\mathcal{P}_1 \in \mathbb{R}^{\ell\times m\times n}(\|\mathcal{P}_1\| = 1)$ を選択する．
$\mathcal{Q}_1 := \mathcal{P}_1 \times_1 A + \mathcal{P}_1 \times_2 B + \mathcal{P}_1 \times_3 C$; $\alpha_1 := \|\mathcal{Q}_1\|$;
$\mathcal{Q}_1 := \mathcal{Q}_1/\alpha_1$;
for until convergence **do**
  $\mathcal{R}_i := \mathcal{Q}_i \times_1 A^{\mathrm{T}} + \mathcal{Q}_i \times_2 B^{\mathrm{T}} + \mathcal{Q}_i \times_3 C^{\mathrm{T}} - \alpha_i\mathcal{P}_i$;
  $\beta_i := \|\mathcal{R}_i\|$;
  $\mathcal{P}_{i+1} := \mathcal{R}_i/\beta_i$;
  $\mathcal{Q}_{i+1} := \mathcal{P}_{i+1} \times_1 A + \mathcal{P}_{i+1} \times_2 B + \mathcal{P}_{i+1} \times_3 C - \beta_i\mathcal{Q}_i$;
  $\alpha_{i+1} := \|\mathcal{Q}_{i+1}\|$;
  $\mathcal{Q}_{i+1} := \mathcal{Q}_{i+1}/\alpha_{i+1}$;
end for

ここで，Algorithm 2 に対する所要メモリは $n = \ell = m = 100$ に関して，約 24MB となる．

次に，Algorithm 2 に対する初期値として，3 項テンソル和の固有ベクトルを利用する方法を考える．行列 A, B, C に対する固有値・固有ベクトルを $\{\lambda_i^{(A)}, \boldsymbol{x}_i^{(A)}\}, \{\lambda_j^{(B)}, \boldsymbol{x}_j^{(B)}\}, \{\lambda_k^{(C)}, \boldsymbol{x}_k^{(C)}\}$ で表すとき，$s_1(\boldsymbol{x}_{i_{\mathrm{M}}}^{(A)} \circ \boldsymbol{x}_{j_{\mathrm{M}}}^{(B)} \circ \boldsymbol{x}_{k_{\mathrm{M}}}^{(C)}) + s_2(\boldsymbol{x}_{i_{\mathrm{m}}}^{(A)} \circ \boldsymbol{x}_{j_{\mathrm{m}}}^{(B)} \circ \boldsymbol{x}_{k_{\mathrm{m}}}^{(C)})$ で得られるテンソルを初期値 $\mathcal{P}_1$ として提案する．ここで，記号 “$\circ$” は外積を意味し，$s_1, s_2 \geq 0$ かつ $s_1 + s_2 = 1$ を満たすものとする．固有ベクトルの選び方を

$$i_{\mathrm{M}}, j_{\mathrm{M}}, k_{\mathrm{M}} = \arg\max_{i,j,k}\left\{\left|\lambda_i^{(A)} + \lambda_j^{(B)} + \lambda_k^{(C)}\right|\right\},$$
$$i_{\mathrm{m}}, j_{\mathrm{m}}, k_{\mathrm{m}} = \arg\min_{i,j,k}\left\{\left|\lambda_i^{(A)} + \lambda_j^{(B)} + \lambda_k^{(C)}\right|\right\},$$
$$(\text{ただし}, i = 1, 2, \ldots, \ell, j = 1, 2, \ldots, m, k = 1, 2, \ldots, n.)$$

とすることで，3 項テンソル和の絶対値最大・最小固有値に対応する固有ベクトルを利用している．

5 数値実験

定数係数の PDE に対して $(n+1)\times(n+1)\times(n+1)$ グリッドで離散化すると，n^3 次正方行列の 3 項テンソル和が得られ，行列 $A, B, C \in \mathbb{R}^{n\times n}$ となる．このとき，Algorithm 2 で扱うテンソルは $n\times n\times n$ である．この 3 項テンソル和に対する数値実験の結果を示す．

図 1 に，$21\times 21\times 21$ グリッドで離散化して得られた行列に対して，提案した初期値（ただし，$s_1 = s_2 = 0.5$ とした）を用いた Algorithm 2 の収束履歴を示す．図 1 の縦軸は残差ノルムの常用対数，横軸は反復回数である．

図 2 に，Algorithm 2 に対する初期値として，提案した初期値（ただし，$s_1 = s_2 = 0.5$ とした）を用いた場合と 2 種類の乱数を用いた場合に関して，反復回数の違いを示す．図 2 の縦軸は反復回数，横軸は n の値を表し，このときのテンソルサイズは $n\times n\times n$ である．

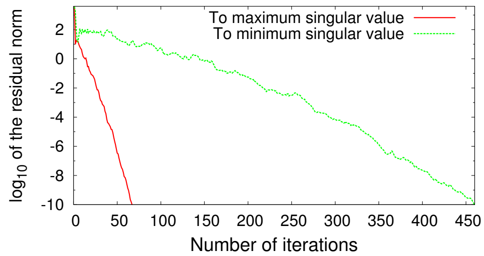

図 1: Algorithm 2 による収束履歴

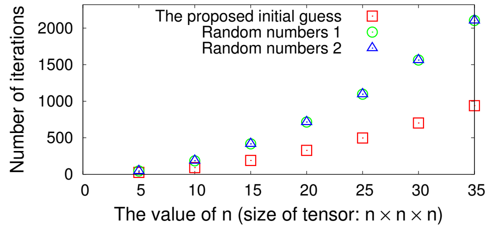

図 2: 初期値の違いによる反復回数の変化

図 1 より，Algorithm 2 を用いて近似最大・最小特異値が得られることが確認された．

図 2 より，提案した初期値は乱数による初期値と比較して，Algorithm 2 の収束を高速化する例が確認された．

6 おわりに

数値実験より，提案手法を用いて 3 項テンソル和に対する最大・最小特異値が求められ，さらに提案した初期値による収束の高速化が確認された．

所要メモリの観点では，直接法と Algorithm 1 に対してそれぞれ $O(n^6)$，$O(n^4)$ だったが，提案手法を用いることで $O(n^3)$ に削減された．

今後の課題として，最小特異値に関する Algorithm 2 の反復回数の削減が挙げられる．

参考文献

[1] G. Golub and W. Kahan, SIAM. J. Numer. Anal. Ser. B, 2(1964), pp. 205-224.

[2] T. G. Kolda and B. W. Bader, SIAM. Rev., 51(2009), pp. 455-500.