データサイエンスにおける固有値計算

二村　保徳

機械学習の手法では，行列の固有値計算や特異値計算が必要となる場合があり，特に主成分分析など，次元削減に関係する機械学習手法に多くみられる．しかしながら，機械学習手法や扱うデータによって解くべき固有値問題の性質が大きく異なる場合があるため，固有値問題の性質に合ったアルゴリズムを選択する必要がある．本稿では固有値計算における適切なアルゴリズム選択の重要性について，主に計算量やメモリ要求量の観点から示す．

キーワード：固有値問題，特異値分解，機械学習

1. はじめに

機械学習やパターン認識，多変量解析の手法では，行列の固有値計算や特異値計算が必要となる場合がある．特に主成分分析などの次元削減に関係する機械学習手法に多くみられる．しかしながら，機械学習手法や扱うデータによって固有値問題の性質が大きく異なる場合があり，その性質に合った固有値計算法を選択する必要がある．特に与えられた行列の要素にゼロが多い疎行列か，そうでない密行列であるかで，適切な固有値計算法が大きく異なる．

Python 言語の scikit-learn [1] などの機械学習手法のパッケージでは，ある程度適切な固有値計算法が用いられているが，自分が使いたいデータ構造や設定にパッケージが対応していない場合や，大量のデータ処理のために分散並列計算を行う場合などでは，やはり適切な固有値計算法を選択，実装することが必要となる．

機械学習手法の計算量の議論において『固有値計算を伴う手法は計算量が行列の次元 n に対して $O(n^3)$ であるため計算量が大きい』とされることがあるが，特に疎行列では必ずしもそうとはならない．固有値計算をはじめとする線形計算の既存の効率的なアルゴリズムを活かすことで，より多様な機械学習手法が現場で活用されることが望ましいと考えられる．

本稿では固有値計算において適切なアルゴリズムの選択の重要性を主に計算量やメモリ要求量の観点から示していく．本稿で述べる事項がデータ解析の現場にどのように関わってくるかを示すため，関連する Python 言語のパッケージについて適宜触れながら議論していく[1]．

ふたむら　やすのり
筑波大学システム情報系
〒 305–8573 茨城県つくば市天王台 1–1–1
futamura@cs.tsukuba.ac.jp

2. 固有値問題と特異値分解

本節では次節以降への準備のため，線形代数の教科書で説明されている基本的事項について確認する．本稿で扱う行列やベクトルの要素はすべて実数であるとする．

2.1 固有値問題

n 次の正方行列 A に関する固有値問題とは

$$A\boldsymbol{v} = \lambda\boldsymbol{v} \tag{1}$$

を満たす固有値 λ（スカラー）および固有ベクトル $\boldsymbol{v}$（非ゼロベクトル）を求める問題である．この形式の固有値問題は後述する一般化固有値問題に対して，標準固有値問題と呼ばれる．機械学習手法で扱う固有値問題では A が実対称行列であることが多いため，A は実対称行列であると仮定する．このとき固有値 λ は実数で n 個存在し，すべての固有値を対角成分にもつ n 次正方対角行列を Λ, 各固有ベクトルを対応する列にもつ n 次正方行列を V とすると，行列 A は $A = V\Lambda V^\top$ と行列の積の形で表すことができる．これは固有値分解と呼ばれる．また，V は直交行列になる（$V^{-1} = V^\top$ を満たす）．

機械学習手法では式 (1) の右辺に正方行列 $B \in \mathbb{R}^{n\times n}$ が入った問題

$$A\boldsymbol{v} = \lambda B\boldsymbol{v} \tag{2}$$

を解く必要が生じる場合がある．これは一般化固有値問題と呼ばれる．

なお本稿に限らず，『固有値』計算とはいうものの，ふつう固有値だけでなく固有ベクトルも計算する．

[1] 各パッケージの仕様は本稿執筆時点での最新バージョンにおけるものとする

2.2 特異値分解

一般の長方行列 $A \in \mathbb{R}^{n\times m}$ に関して，

$$A = U\Sigma V^\top \tag{3}$$

を満たす直交行列 $U \in \mathbb{R}^{n\times n}$，非負要素からなる対角行列 $\Sigma \in \mathbb{R}^{n\times m}$，および直交行列 $V \in \mathbb{R}^{m\times m}$ による行列分解を特異値分解 (Singular Value Decomposition: SVD) と呼ぶ．Σ の対角要素は特異値，U および V の列ベクトルはそれぞれ左特異ベクトル，右特異ベクトルと呼ばれる．ここで特異値は Σ の対角上で降順に並んでいるものとする．対称な正方行列 $A^\top A$ の固有値分解は $A^\top A = V\Sigma^2 V^\top$ となり，同じく対称な正方行列 $AA^\top$ の固有値分解は $AA^\top = U\Sigma^2 U^\top$ となる．このように特異値分解と固有値分解には密接な関係がある．

$n > m$ のとき，U_0 を U の先頭の m 列からなる行列，行列 Σ_0 を Σ の左上端の m 次正方対角行列とすると，$A = U_0\Sigma_0 V^\top$ とも分解できる．これは Thin SVD と呼ばれるが [2]，この分解が単に特異値分解と呼ばれることもある．Python の数値計算パッケージ NumPy [3] の特異値分解の関数 `numpy.linalg.svd` では第 2 引数を `False` とすると Thin SVD の計算になる．こちらの方が効率的な計算となる．

3. 疎行列計算

通常，計算機で行列に関する数値計算をする際，行列は 2 次元配列で表すが，n 次の正方行列の場合 $O(n^2)$ のメモリ容量が必要となる．しかしながら，たとえば単語文書行列や頂点次数の小さいグラフの隣接行列など，ゼロでない要素（非ゼロ要素）が少ない行列を扱う場合がある．そのような行列は疎行列 (Sparse matrix) と呼ばれる．どれほど非ゼロ要素が少なければ疎行列と呼ぶかについての明確な基準はない．

行列のゼロ要素を記憶しないよう工夫された疎行列を表すためのデータ構造が提案されており，その代表的なものとして Compressed Sparse Row 形式（Compress Row Storage とも呼ばれる）が挙げられる．これ以外にも特定の非ゼロ要素配置パターンをもった行列に対して，メモリアクセス効率の高いデータ構造が各種提案されている．

疎行列を表すデータ構造はプログラミング言語自身またはそのパッケージで提供されている場合がある．たとえば Python 言語の科学技術計算パッケージの SciPy [4] では疎行列向けデータ構造が複数提供されている．

疎行列向けのデータ構造を用いることで十分に疎な行列であれば，数百万次元や数千万次元以上の行列をメモリ上に置くことができる．ただし，疎行列向けのデータ構造を使う場合には行列が疎であることを活用できるアルゴリズムを選択することが望ましい．

4. 固有値問題／特異値分解に関係する機械学習手法

データ行列を $X \in \mathbb{R}^{n\times m}$ とする．n はサンプル数，m は特徴量の数とする．X の第 i 行の行ベクトルの転置を $\boldsymbol{x}_i \in \mathbb{R}^m$ とおき，これを i 番目のデータベクトルと呼ぶ．$X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^\top$ と表せる．

4.1 主成分分析

教師無し次元削減手法として最も良く知られている主成分分析 [5] について述べる．データ行列 X が中心化され，$\sum_{i=1}^n \boldsymbol{x}_i = \boldsymbol{0}$ をすでに満たしているとすると，X に対する共分散行列 C は $C = \frac{1}{n}X^\top X$ として与えられる．$\boldsymbol{x}_i^\top \boldsymbol{v}$ の分散が最大となるノルムが 1 のベクトル $\boldsymbol{v} \in \mathbb{R}^m$ は共分散行列の標準固有値問題 $C\boldsymbol{v} = \lambda\boldsymbol{v}$ の最大の固有値に対応する固有ベクトル $\boldsymbol{v}_1$ によって与えられる．$\boldsymbol{x}_i$ の $\boldsymbol{v}_1$ 成分 $\boldsymbol{x}_i^\top \boldsymbol{v}_1$ を（第 i データベクトルの）第 1 主成分と呼ぶ．さらに最大から j 番目の固有値の固有ベクトルを $\boldsymbol{v}_j$ としたとき $\tilde{x}_i^{(j)} := \boldsymbol{x}_i^\top \boldsymbol{v}_j$ は第 j 主成分と呼ばれる．

$n > m$ のとき X の Thin SVD を $X = U_0\Sigma_0 V^\top$ とすると $XV = U_0\Sigma_0$ であり，$\tilde{x}_i^{(j)}$ は XV の (i,j) 要素なので，$\boldsymbol{u}_j$ を U の第 j 列ベクトル（左特異ベクトル），σ_j を Σ_0 の第 j 対角要素（特異値）としたとき $\tilde{x}_i^{(j)}$ はベクトル $\sigma_j\boldsymbol{u}_j$ の第 i 要素となる．このため第 j 主成分は最大から j 番目の特異値と対応する左特異ベクトルで表すことができる．$k < m$ 個分の主成分が必要な場合は，A の (Thin) SVD そのものは必要でなく，最大から k 個の特異値と対応する左特異ベクトルさえ計算できれば良い．

主成分分析の計算は画像認識分野の部分空間法に用いられる．また，線形判別分析（フィッシャーの判別分析）や正準相関分析は一般化固有値問題に帰着される．

主成分分析／特異値分解は自然言語処理分野の潜在意味解析 [6] と関連が深い．潜在意味解析では単語文書行列 A の特異値・特異ベクトルの計算を行う．単語文書行列は通常疎行列になるが，A が疎である場合の特異値・特異ベクトルの計算については 6.2 節で触れる．

4.2 スペクトラルクラスタリング

代表的なクラスタリング手法の一つとしてスペクトラルクラスタリング [7, 8] がある（scikit-learn では `sklearn.cluster.SpectralClustering`）．デー

タ行列 $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^\top$ が与えられたとき，m 次元空間における類似度を導入して，無向グラフを定義する．たとえば i 番目と j 番目のデータの類似度を

$$a_{ij} = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\delta^2}\right) \tag{4}$$

として k 近傍グラフ (k-nearest neighbor graph)[8] を用いる（$\delta > 0$ はユーザーが与える実数）．k 近傍グラフの隣接行列 A を考えると，k が小さい場合，A は疎行列となる．$d_i := \sum_j a_{i,j}$ を対角に並べた対角行列を D としたとき，行列 $L = D - A$ はグラフラプラシアン行列と呼ばれる．スペクトラルクラスタリングではこれらの行列に関する一般化固有値問題

$$L\boldsymbol{v} = \lambda D\boldsymbol{v} \tag{5}$$

を考える．最小から s 個の固有値に対応する固有ベクトルを求めて，$V_s := [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_s]$ とおく．この V_s の（n 本の）行ベクトルに対する k-means 法を適用してクラスタリング結果を得る手法がスペクトラルクラスタリングである（k-means 法の k は前述の k 近傍グラフの k とは無関係）．V_s の各行ベクトルは対応するデータベクトルの低次元表現になっており，次元削減手法として V_s を得る計算は Laplacian Eigenmap [7, 9] と呼ばれている（scikit-learn では `sklearn.manifold.SpectralEmbedding`）．また一般化固有値問題 (5) はグラフ分割の目的関数の一つである Normalized Cut 関数の最小化の連続緩和問題であることが知られている．Normalized Cut 関数は画像領域分割などに応用されている．また，グラフラプラシアンやその固有値・固有ベクトルの概念はグラフ信号処理 [10] の分野や，グラフに対する畳込みニューラルネットワークで用いられている [11].

4.3 ニューラルネットワークの学習状況の解析

ニューラルネットワークの学習状況の解析では，ニューラルネットワークの損失関数から与えられるヘッセ行列の固有値・固有ベクトルが有用な情報となる．

ヘッセ行列の次元数 n はニューラルネットワークのパラメータ数であるため，パラメータ数が多いいわゆる深層学習モデル（たとえば有名な ResNet モデルでは 10^7 のオーダー）ともなると，ヘッセ行列を 2 次元配列で表現しようとすると $O(n^2)$ のメモリが必要となり現実的でないが，ヘッセ行列とベクトルの積の結果のベクトルを計算する処理は相対的に少ない計算量で実行できることが知られている．この性質を利用することで，ヘッセ行列そのものは疎行列ではないものの，5.2 節で述べる疎行列計算向けの固有値計算法を用いることで，現実的な計算時間で固有値計算を行うことができる．

ニューラルネットワークの分野では疎行列計算向け固有値計算法を活用して損失関数の形状に関する議論 [12] やヘッセ行列の固有値分布の評価 [13] が行われている．

5. 固有値計算法

本節では標準的な固有値計算法を紹介する．密行列の場合と疎行列の場合に分け，計算量やメモリ要求量の観点から各手法の違いについて述べる．

5.1 密行列の固有値計算

まず密行列の標準固有値問題 (1) の解法について述べる．標準的に用いられている解法は直交変換による三重対角化に基づく解法である [2, 14]．対称行列 A はある直交行列 Q によって三重対角行列

$$T = Q^\top AQ \tag{6}$$

に変換できる．このとき T は A と同じ固有値をもち，T の固有ベクトルを $\boldsymbol{w}$ とおくと，A の固有ベクトル $\boldsymbol{v}$ は $\boldsymbol{v} = Q\boldsymbol{w}$ となる．三重対角化に基づく解法では，Householder 変換という直交変換を用いて Q の作用を表現して A を三重対角行列 T に変換する．T を計算したのち，三重対角行列の性質を利用した QR 法や分割統治法などの反復アルゴリズムを用いて T の固有値・固有ベクトルを求める．

A の次元数を n としたとき，三重対角化に要する計算量は求める固有値・固有ベクトルの数によらず $O(n^3)$ である．また，三重対角行列の固有値計算の計算量は，求める固有値・固有ベクトルの数によるが最大でも $O(n^3)$ である．したがって，三重対角化があるために固有値計算全体の計算量は求める固有値・固有ベクトルの数によらず $O(n^3)$ となる．

密行列の一般化固有値問題 (2) で，A が対称で B が対称正定値の場合は，B の Cholesky 分解を利用して同じ固有値をもつ対称行列の標準固有値問題に帰着し，その後，前述の標準固有値問題に対するアルゴリズムを用いる．Cholesky 分解の計算量は三重対角化と同じ $O(n^3)$ であり，固有値計算全体の計算量は $O(n^3)$ となる．

このように一般の密行列向けのアルゴリズムの計算量が $O(n^3)$ であることが『固有値計算の計算量は $O(n^3)$』といわれるゆえんである．ただし，行列 A とベクトルの積の計算量が小さく（疎行列がその典型例である），求める固有値・固有ベクトルが少数の場合は次の 5.2 節

で述べるアルゴリズムを利用すれば必ずしも $O(n^3)$ とはならない．

5.2 疎行列の固有値計算

標準固有値問題 (1) の場合について述べる．行列 A が疎行列の場合，行列の疎性を活かしたデータ構造を用いることが望ましいが，5.1 節で述べた直交変換による三重対角化を行うには $O(n^2)$ のメモリ容量が必要であり，大規模な疎行列に対して直交変換による三重対角化に基づくアルゴリズムを実行することは計算量以前に要求メモリ量の観点からすでに困難である．

疎行列の場合かつ少数の固有値・固有ベクトルが必要である場合は，射影法と呼ばれるクラスの解法 [15–17] を用いる．射影法のなかでも直交射影法と呼ばれるクラスの解法では，ある k 次元の部分空間 $\mathcal{W}$ の正規直交基底を $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_k\}$, 行列 $W := [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_k]$ としたとき，k 次元の固有値問題

$$W^\top A W \boldsymbol{y} = \tilde{\lambda} \boldsymbol{y} \tag{7}$$

を解くことで元の固有値問題 (1) の固有値を $\lambda \approx \tilde{\lambda}$, 固有ベクトルを $\boldsymbol{v} \approx W\boldsymbol{y}$ として近似する．このとき k は求める固有値の数以上とする必要があり，通常，固有値の数と同じオーダーの値とする．k 次元の小さい固有値問題 (7) は 5.1 節の密行列向け固有値計算法で解かれる．

部分空間 $\mathcal{W}$ として, 必要な固有ベクトルに対応する不変部分空間を良く近似した部分空間であることが望ましいため, そのような部分空間の基底ベクトルをどのように計算するかが重要となる．その代表的な方法として，Lanczos 法がある．Lanczos 法ではベクトル $\boldsymbol{w}_0$ に対する Krylov 部分空間 $\mathrm{Span}(\{\boldsymbol{w}_0, A\boldsymbol{w}_0, \ldots, A^{k-1}\boldsymbol{w}_0\})$ の正規直交基底を計算し $\mathcal{W}$ の基底として用いる．Lanczos 法は Krylov 部分空間に基づくため，アルゴリズム中で行列 A は A とベクトルの積の形でのみ現れ，この性質により行列 A の疎性を活かすことができる．

部分空間の次元数 k はユーザーが与えるが，基底ベクトルを保持するために $O(nk)$ のメモリ容量が必要となってしまうため，あまり大きい値を与えるわけにはいかない．与えた k で十分な精度の固有値・固有ベクトルが得られない場合は，その時点での近似固有値・固有ベクトルを使い新たな初期ベクトル $\boldsymbol{w}_0$ を与えて部分空間を作りなおすなどする．これはリスタートと呼ばれる．リスタートには，複数の手法が提案されている．

Lanczos 法に限らず多くの疎行列向け計算法ではアルゴリズム中で行列 A がベクトル積の形でのみ現れる．たとえば 1 行あたりの非ゼロ要素数が $O(1)$ である場合，3 節で述べた疎行列向けデータ構造を用いれば，疎行列ベクトル積に要する計算量は $O(n)$ となる．十分な精度で近似固有値を得るために適切な部分空間の次元 k やリスタート回数が精度の要求レベルや行列 A の数値的な性質に依存することから，疎行列ベクトル積が何回必要となるか一概にはいえない．そのため，Lanczos 法の全体の計算量についての明確な提示は難しいが，A が十分に疎な行列で，かつ求める固有値数が少ない場合，5.1 節の密行列向け解法と比べ相当に高速となる．一例として性能比較を 6.3 節で示す．

通常の Lanczos 法で得られる近似固有値は最大や最小の固有値に収束する性質があることが知られており，特定の値に近い固有値を求める場合などには必ずしも適していない．Lanczos 法以外にもさまざまな解法が提案されている．詳しくは文献 [15–17] を参考とされたい．

6. 固有値計算に関係するソフトウェア

固有値計算を含む数値線形計算分野では古くから汎用的なソフトウェアパッケージが開発・公開されている．本節ではより身近な逐次計算または共有メモリ型並列計算向けのソフトウェアを紹介する．分散並列計算に関しては発展的な内容になるので本稿では触れない．分散並列計算向けのソフトウェアに関しては文献 [17]（筆者の担当箇所あり）などを参考とされたい．本節でも密行列の場合と疎行列の場合に分けて紹介し，最後に性能比較を示す．

6.1 密行列計算向けソフトウェア

密行列計算向けとしては BLAS と LAPACK という古くから FORTRAN 言語で開発されてきた行列計算のパッケージがある．BLAS は行列積などの行列やベクトル演算の関数群，LAPACK は固有値問題や特異値分解，連立一次方程式などの線形計算の各種問題を解く関数群を提供している．それぞれ一般の密行列および三重対角行列などの特殊な行列に対応する関数群がある．BLAS と LAPACK は一緒に提供されており，ここでは BLAS/LAPACK と呼称する．Python 言語の NumPy や R 言語，Matlab 言語などの行列計算では，裏で BLAS/LAPACK の関数が実行されている．

BLAS/LAPACK は本家の Netlib [18] 以外にもさまざまな実装がある．代表的な実装としてオープンソースの OpenBLAS [19] や Intel 社の Intel Math Kernel Library 内の実装などプロセッサベンダーが開発する BLAS/LAPACK 実装が存在する．これらの

BLAS/LAPACK 実装は FORTRAN 言語や C 言語やアセンブリ言語で実装され，各種プロセッサ向けにチューニングされている．また共有メモリ型並列（スレッド並列）での実行が可能である．なおパッケージ管理ツール `pip` でインストールされた NumPy では OpenBLAS がバックエンドで働く．NumPy のバックエンドで使う BLAS/LAPACK 実装を変更すると何もしなくても計算が高速化される可能性がある．

BLAS/LAPACK の日本語の書籍としては文献 [20] が詳しい．

6.2 疎行列計算向けソフトウェア

疎行列向けの固有値計算分野の歴史も長く，FORTRAN 言語や C/C++ 言語で書かれたさまざまな解法のソフトウェアが多数存在する．ここでは以下，機械学習ユーザーにとって身近な Python 言語に関わるソフトウェアについて述べる．

SciPy パッケージの対称な疎行列向け固有値計算関数である `eigsh` 関数では Implicit restart というリスタート法を用いる Lanczos 法を実装した ARPACK という FORTRAN 言語のパッケージの関数を裏で呼んでいる．最大や最小などから何個固有値・固有ベクトルを求めるかを指定できる．

さらに SciPy の疎行列向け特異値計算関数である `svds` 関数（のデフォルト動作）では，特異値分解と固有値分解の関係を利用し，$n \geq m$ の場合は $AA^\top$ に対して，$n < m$ の場合は $A^\top A$ に対して `eigsh` 関数を適用している．以下 $AA^\top$ の場合で説明する．$AA^\top$ の固有値問題に対して Lanczos 法を適用することを考えると，あるベクトル $\boldsymbol{x}$ と $AA^\top$ との積 $\boldsymbol{y} = AA^\top\boldsymbol{x}$ の形でのみ A が現れる．$AA^\top$ を作ってしまうと n 次正方の密行列になってしまうが，$\boldsymbol{y}$ の計算に対して $\boldsymbol{z} = A^\top\boldsymbol{x}$ を計算してから，$\boldsymbol{y} = A\boldsymbol{z}$ を計算するようにすれば行列 A の疎性を活かすことができる．`svds` 関数ではこのような形で Lanczos 法を利用して一部の特異値・特異ベクトルを計算している．

6.3 性能比較

スペクトラルクラスタリングで扱うグラフラプラシアン行列（疎行列）に対して，NumPy の密行列向け固有値計算関数 `eigh` と SciPy の疎行列向け固有値計算関数 `eigsh` を適用して計算時間を確認する．データ行列としては手書き数字画像データセット MNIST [21] からランダムに n 個取り出したデータを用いた．グラフラプラシアン行列を生成する k 近傍グラフの k は 7 とした（この k が小さいほど行列が疎になる）．行列サイズは $n = 1000, 2000, 3000, 4000, 5000, 6000$ の 6 ケースで確認した．それぞれの非ゼロ要素数は 11160, 22392, 33616, 44958, 56496, 67614 である．求める固有値・固有ベクトルの数はスペクトラルクラスタリングでのユースケースに対応して最小から 10 個とした．計算環境は MacBook Pro でプロセッサは Intel Core i5 2.9 GHz 2 cores，メモリは 8GB 2133 MHz LPDDR3 である．Python 3.8.1, NumPy 1.18.1, SciPy 1.4.1 を用いた．

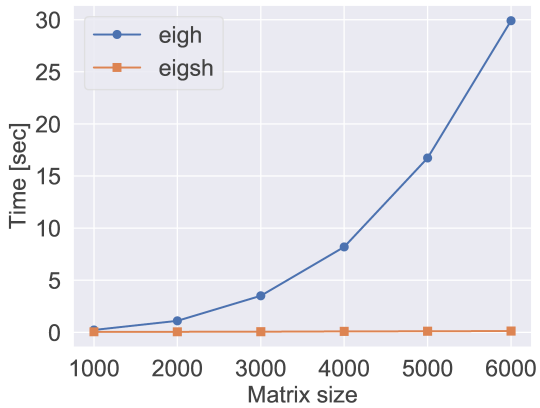

図 1 疎行列に対する疎行列向け関数 `eigsh` および密行列向け関数 `eigh` での固有値計算の計算時間の比較

計算時間の計測結果を図 1 に示す．かなり疎な行列に対して少数の固有値・固有ベクトルを計算する場合，疎行列向けの関数が密行列向けの関数より速いという当たり前の結果だが，特に $n = 6000$ では比が約 300 倍（29.9 秒と 0.12 秒）となっており，適切な固有値計算法の選択が重要であることがわかる．

7. おわりに

本稿では各種機械学習手法で解く必要が生じる固有値問題の解法について，行列が密行列である場合と疎行列である場合に分け，それぞれの標準的なアルゴリズムについて紹介した．また近年機械学習に使われている Python 言語関連を中心に固有値計算のソフトウェアについて解説した．通常，機械学習手法を使用する際は既存のパッケージを用いることと思うが，パッケージが対応していないテクニックを用いる場合や，データが大規模で分散並列計算が必要になった場合など，固有値計算法を選択，または自前で実装する必要が生じることもあるかもしれない．そのような際に本稿が適切なアルゴリズム選択の一助となれば幸いである．

参考文献

[1] scikit-learn, https://scikit-learn.org/（2020 年 3 月 5 日閲覧）.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations, Fourth Edition*, Johns Hopkins University Press, 2012.

[3] NumPy, https://numpy.org/（2020 年 3 月 5 日閲覧）.

[4] SciPy, https://www.scipy.org/（2020 年 3 月 5 日閲覧）.

[5] 平井有三，『はじめてのパターン認識』，森北出版，2012.

[6] 岩田具治，『トピックモデル』，講談社，2015.

[7] 赤穂昭太郎，『カーネル多変量解析』，岩波書店，2008.

[8] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, **17**, pp. 395–416, 2007.

[9] M. Belkin and P. Niyogi, "Laplacian Eigenmaps for dimensionality reduction and data representation," *Neural Computation*, **15**, pp. 1373–1396, 2003.

[10] 田中雄一，"グラフ信号処理のすゝめ"．電子情報通信学会 基礎・境界ソサイエティ Fundamentals Review, **8**, pp. 15–29, 2014.

[11] J. Bruna, W. Zaremba, A. Szlam and Y. Lecun, "Spectral networks and locally connected networks on graphs," In *International Conference on Learning Representations (ICLR2014)*, 2014.

[12] Z. Yao, A. Gholami, Q. Lei, K. Keutzer and M. W. Mahoney, "Hessian-based analysis of large batch training and robustness to adversaries," In *NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 4954–4964, 2018.

[13] B. Ghorbani, S. Krishnan and Y. Xiao, "An investigation into neural net optimization via Hessian eigenvalue density," In *Proceedings of the 36th International Conference on Machine Learning*, Long Beach, California, PMLR 97, 2019.

[14] J. W. Demmel, *Applied Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, 1997.

[15] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst (eds.), *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Society for Industrial and Applied Mathematics, 2000.

[16] Y. Saad, *Numerical Methods for Large Eigenvalue Problems: Revised Edition*, Society for Industrial and Applied Mathematics, 2011.

[17] 日本応用数理学会監修，櫻井鉄也，松尾宇泰，片桐孝洋編，『数値線形代数の数理と HPC』共立出版，2018.

[18] The Netlib, http://www.netlib.org/（2020 年 3 月 5 日閲覧）.

[19] OpenBLAS, https://www.openblas.net/（2020 年 3 月 5 日閲覧）.

[20] 幸谷智紀，『LAPACK/BLAS 入門』，森北出版，2016.

[21] THE MNIST DATABASE of handwritten digits, http://yann.lecun.com/exdb/mnist/（2020 年 3 月 5 日閲覧）.