I-SVD Library Manual version 0.5.0

内容

1. 概要

I-SVD ライブラリは mdLVs 法と dLV 型ツイスト分解を組み合わせた I-SVD 法を用いて，double 型の密正方行列・上二重対角行列の特異値計算や特異値分解を行うためのライブラリです．このライブラリは並列化されており，複数のコア・プロセッサが搭載されているマシンでは，特異値分解において一定の速度向上が見られます．速度が向上する割合は，行列の特異値分布によって変化します．

著作権

本ライブラリの並列 I-SVD 法の部分は，京都大学大学院 情報学研究科 数理工学専攻の中村佳正教授のチームが作成しました．また，神戸大学大学院 工学研究科の山本有作教授が作成した，Bischof の手法と村田法によって密行列を二重対角行列に変換・逆変換するコードが含まれています．

2. インストール

ライブラリの実行に必要な要件や，インストール手順，ライブラリを使用する際のコンパイル方法などを説明します．

システム要件

- x86 アーキテクチャのプロセッサ（並列化するには複数のコアまたはプロセッサが必要）
- OS は Linux
- 行列サイズに応じたメモリ
- LAPACK と BLAS
- C++と Fortran をサポートするコンパイラ（g++と gfortran など）
- OpenMP

※ライブラリの並列実行には内部で pthread と OpenMP を使用しています．そのため，MPI は不要です．

インストール手順

圧縮ファイルを展開してください．圧縮ファイルを展開すると，次のようなファイルが生成されます．

- isvd.h ..... I-SVD ライブラリを呼び出す際に必要なヘッダファイル
- libisvd.a ..... I-SVD ライブラリのファイル
- libisvd_goto2.a ..... I-SVD ライブラリのファイル（GotoBLAS2 をリンクする場合）
- sample_dbdslv.cpp ..... 4 章に掲載している DBDSLV のサンプルプログラム
- sample_dgeslv.cpp ..... 4 章に掲載している DGESLV のサンプルプログラム
- atconfig ..... DGESLVAT を呼び出す前に実行する必要のあるプログラム
- I-SVD_Library_Manual_0.5.0.pdf..... このマニュアル

インストールに際して，その他の操作は必要ありません．

I-SVD ライブラリを使用するプログラムのコンパイル

以下では GNU C++ Compiler (g++) 及び gfortran のバージョン 4.1.2 を用いることを前提に説明します．異なるコンパイラをご使用の場合は，以下のコマンドを適宜お使いのコンパイラに合わせて変更してください．
I-SVD ライブラリでは，pthread・OpenMP・LAPACK・BLAS を使用しています．そのため，-lpthread, -fopenmp, -llapack, -lblas オプションが必要です．また，-lgfortran オプションも必要です．
プログラムをコンパイルするには，以下のように行ってください．

```
g++ 〈プログラムのソースコードのファイル名〉-lpthread -fopenmp -L. -lisvd -lgfortran -llapack -lblas
```

ソースコードと libisvd.a が別のディレクトリにある場合は，-L オプションで libisvd.a の場所を指定してください．

その他の BLAS の使用

BLAS として netlib で入手可能な BLAS 以外に，GotoBLAS2 や Intel Math Kernel Library などを用いることも可能です．その際は，本ライブラリを用いるプログラムのコンパイル時に，目的の BLAS を用いるようにパラメータを変更してください．なお，GotoBLAS2 を用いる場合は libisvd.a ではなく libisvd_goto2.a を用いてください．

＜GotoBLAS2 を用いる場合のコンパイル例＞

g++ *〈プログラムのソースコードのファイル名〉* -lpthread -fopenmp -L. -lisvd_goto2 -lgfortran -L*〈GotoBLAS2 のパス〉* -lgoto2

GotoBLAS2 をインストールする際には，**USE_OPENMP=1 オプションを指定してください**．詳しくは GotoBLAS2 の 02QuickInstall.txt または Makefile.rule をご覧ください．

DGESLV, DGESLV1 および DGESLV2 の実行中は，複数のスレッドによるメモリアクセスの集中のため，計算機環境によってはパフォーマンスが悪化する場合があります．その場合は以下の方法を行い，**スレッドを各コアに均等に配置**するようにしてください．

- 環境変数 GOMP_CPU_AFFINITY="0-7:1"に設定（8 コアのコンピュータの場合）
- （Intel コンパイラを用いる場合のみ）環境変数 KMP_AFFINITY=scatter に設定

オートチューニング機能の前準備

関数 DGESLVAT は，DGESLV, DGESLV1 のブロックサイズ lb と高速化オプション ppmode の適切な値を，自動で設定して呼び出すものです．DGESLV, DGESLV1 を呼び出す代わりに DGESLVAT を呼び出すことで，パラメータ lb, ppmode を設定せずに特異値分解を行うことができます．

この関数を使用する前に，マシンの特性を必要があります．DGESLVAT を呼び出す前に，ファイルに添付されている atconfig ユーティリティを実行してください．

./atconfig *〈マシンのコア数〉*

引数として，マシンに搭載されているコア数（DGESLVAT を呼び出す際にパラメータ threads に渡す値）を指定してください．なお，atconfig ユーティリティの処理が完了するには 15 分～1 時間程度かかります．

atconfig ユーティリティを実行すると，time_bischof.txt, time_dongarra.txt, time_blas.txt, ISVDConfig.txt というファイルが作成されます．これらのファイルを libisvd.a や libisvd_goto2.a と同じディレクトリに入れておくと，これらのファイルを参照して最適なパラメータ lb, ppmode を設定します．これらのファイルが見つからないとエラーになります．

atconfig ユーティリティは，I-SVD ライブラリのインストール時などに 1 回のみ実行すれば，再び実行する必要はありません．ただし以下のように，ライブラリの実行時間に変化が生じる可能性がある場合は，再度 atconfig を実行して time_bischof.txt などを更新してください．

- I-SVD ライブラリのバージョンが変わったとき
- BLAS の種類やバージョンが変わったとき
- CPU の変更などコンピュータの性能が変わったとき

3. ライブラリの使用法

本ライブラリには特異値計算及び特異値分解を行う関数が複数含まれています．これらは与える行列が密行列か上二重対角行列かどうかで使い分けてください．

これらの関数の宣言は isvd.h に書かれています．このライブラリを使用する際には，isvd.h をインクルードしてくださ

い.

関数 DBDSLV

```
int DBDSLV(int size, double* diag, double* subdiag, double* Uvecs, double* Vvecs, int* UvecsOrder,
        int* VvecsOrder, double* work, int* iwork, int threads, int valueonly)
```

目的

渡された上二重対角行列 B の特異値分解を I-SVD 法によって実行します. マルチコアマシン上では pthread により並列実行できます. mdLVs 法による特異値計算のみを行うことも可能です.
これは I-SVD ライブラリバージョン 0.1.0 の関数 ISVD に相当します.

引数

- size ..... (入力 int 型) 行列のサイズ
- diag ..... (入力/出力 double*型, size 次元) 行列の対角成分. 処理が完了すると, 特異値が格納されます.
- subdiag ..... (入力 double*型, size 次元) 行列の副対角成分. 処理が完了すると, この配列の中身は破壊されます.
- U ..... (出力 double*型, (size*size)次元) 左特異ベクトルを格納する行列 U
- V..... (出力 double*型, (size*size)次元) 右特異ベクトルを格納する行列 V
- UvecsOrder ..... (出力 int*型, size 次元) Uvecs に格納されているベクトルの順番(※1)
- VvecsOrder..... (出力 int*型, size 次元) Vvecs に格納されているベクトルの順番(※1)
- work..... (ワークスペース double*型, (16*size + 20*size*threads)次元) double 型のワークスペース. この値は GetWorkSize(size, 0, threads, 0)で取得できます.
- iwork..... (ワークスペース int*型, (8*size*threads)次元) int 型のワークスペース. この値は GetWorkSize(size, 0, threads, 10)で取得できます.
- threads ..... (入力 int 型) 並列実行する際に起動するスレッドの数
- valueonly..... (入力 int 型) 特異値分解を行う場合は 0, 特異値計算のみ行い特異ベクトルを計算しない場合は 1 を渡してください.

戻り値

(int 型) 正常終了なら 0, エラーが発生した場合は 0 以外が返されます.

- 0 ..... 正常終了
- −*n* ..... *n* 番目のパラメータが不正(*n* は 1〜11)

※1: i 番目の特異値(i = 0, 1, …, size - 1)に対応する特異ベクトルは, Uvecs と Vvecs の(UvecsOrder(i) * size)番目から(UvecsOrder(i) * size + (size – 1))番目に格納されています. 特異ベクトルにアクセスする際には, 以下のようにしてください.

```
// C++言語の場合
// 0 <= i, j <= size - 1 ：インデックスの範囲に注意！
U[i][j] = Uvecs[UvecsOrder[j] * size + i];
V[i][j] = Vvecs[VvecsOrder[j] * size + i];
```

関数 DGESLVAT

```
int DGESLVAT(int size, double* A, double* values, double* U, double* V, int* iwork, int threads, int
        valueonly)
```

目的

渡された密正方行列 A の特異値分解を行います．内部では後述の DGESLV または DGESLV1 のうち，高速に処理できる方が自動的に選択されて呼び出されます．また，この関数を用いた場合は DGESLV や DGESLV1 のパラメータ lb, ppmode も自動的に適切な値が設定されます．ワークスペース work は，内部で自動的に確保されるため，I-SVD ライブラリを呼び出すプログラム側で確保する必要はありません．

なお，この関数はオートチューニング機能を用いて適切なパラメータを設定します．atconfig ユーティリティを用いて time_bischof.txt, time_dongarra.txt, time_blas.txt, ISVDConfig.txt を作成してから DGESLVAT を呼び出してください．

マルチコアマシン上では pthread と OpenMP により並列実行できます． mdLVs 法による特異値計算のみを行うことも可能です．A が上二重対角行列の場合は，DBDSLV を呼び出した方が高速に処理されます．

引数

- size ..... （入力 int 型）行列のサイズ
- A..... （入力 double*型，(size*size)次元）特異値計算・特異値分解する密正方行列 A．処理が完了すると，この配列の中身は破壊されます．
- values ..... （出力 double*型，size 次元）計算された特異値が格納されます
- U ..... （出力 double*型，(size*size)次元）左特異ベクトルを格納する行列 U
- V..... （出力 double*型，(size*size)次元）右特異ベクトルを格納する行列 V
- threads ..... （入力 int 型）並列実行する際に起動するスレッドの数
- valueonly..... （入力 int 型）特異値分解を行う場合は 0，特異値計算のみ行い特異ベクトルを計算しない場合は 1 を渡してください．

戻り値

(int 型) 正常終了なら 0，エラーが発生した場合は 0 以外が返されます．

- 0 ..... 正常終了
- －n ..... n 番目のパラメータが不正（n は 1～9）

関数 DGESLV

```
int DGESLV(int size, int lb, double* A, double* values, double* U, double* V, double* work, int* iwork,
        int threads, int valueonly)
```

目的

渡された密正方行列 A の特異値分解を行います. 前処理として Bischof の手法と村田法を用いて密正方行列を二重対角行列に変換し, 内部で DBDSLV を呼び出します. マルチコアマシン上では pthread と OpenMP により並列実行できます. mdLVs 法による特異値計算のみを行うことも可能です. A が上二重対角行列の場合は, DBDSLV を呼び出した方が高速に処理されます.

引数

- size ..... (入力 int 型) 行列のサイズ
- lb ..... (入力 int 型) 内部で行う行列処理での帯幅を表します. このパラメータは 2 以上 size 以下でなければなりません. 通常は 100～200 程度の整数を渡して下さい.
- A..... (入力 double*型, (size*size)次元) 特異値計算・特異値分解する密正方行列 A. 処理が完了すると, この配列の中身は破壊されます.
- values ..... (出力 double*型, size 次元) 計算された特異値が格納されます
- U ..... (出力 double*型, (size*size)次元) 左特異ベクトルを格納する行列 U
- V..... (出力 double*型, (size*size)次元) 右特異ベクトルを格納する行列 V
- work..... (ワークスペース double*型, $(10*size^2 + 16*lb^2 + 20*size*lb + 31*size + 21*size*threads)$次元) double 型のワークスペース. この値は GetWorkSize(N, lb, threads, 1)で取得できます.
- threads ..... (入力 int 型) 並列実行する際に起動するスレッドの数
- valueonly..... (入力 int 型) 特異値分解を行う場合は 0, 特異値計算のみ行い特異ベクトルを計算しない場合は 1 を渡してください.

戻り値

(int 型) 正常終了なら 0, エラーが発生した場合は 0 以外が返されます.

- 0 ..... 正常終了
- $-n$ ..... n 番目のパラメータが不正(n は 1～9)

関数 DGESLV1

```
int DGESLV1(int size, int lb, double* A, double* values, double* U, double* V, double* work, int* iwork,
        int threads, int valueonly, int ppmode)
```

目的

渡された密正方行列 A の特異値分解を行います. 前処理として Bischof の手法と村田法を用いて密正方行列を二重

対角行列に変換し，内部で DBDSLV を呼び出します．マルチコアマシン上では pthread と OpenMP により並列実行できます． mdLVs 法による特異値計算のみを行うことも可能です．A が上二重対角行列の場合は，DBDSLV を呼び出した方が高速に処理されます．
DGESLV と異なり，村田法と並列 I-SVD 法を同時に行うことで特異値分解全体を高速化するオプションが追加されています．それ以外は DGESLV と同じです．

引数

- ppmode ……（入力　int 型）特異値分解全体を高速化する場合は 1，そうでない場合は 0 を渡してください．このパラメータを 0 にした場合の動作は DGESLV と同じです．なお，このパラメータを 1 にした場合は，**計算機環境によっては逆に計算速度が低下する場合があります**（アーキテクチャや使用する BLAS ライブラリに依ります）．特に，**GotoBLAS2 を用いる場合は libisvd_goto2.a を使用してください**．速度が低下する場合はこのパラメータに 0 を渡すか，DGESLV を呼び出してください．

※ その他のパラメータは DGESLV と同じです．

戻り値

(int 型) 正常終了なら 0，エラーが発生した場合は 0 以外が返されます．

- 0 …… 正常終了
- －n …… n 番目のパラメータが不正（n は 1～9）

関数 GetWorkSize

```
int GetWorkSize(int size, int lb, int threads, int alg)
```

目的

DBDSLV, DGESLV, DGESLV1, DGESLV2 で使用するワークスペースの次元数を返します．ワークスペースの変数を確保する場合は，この関数の戻り値にデータ型のサイズ（sizeof(double)）を乗じたサイズのメモリを確保してください．

引数

- size ……（入力　int 型）行列のサイズ
- lb ……（入力　int 型）DGESLV, DGESLV1 の内部で行う行列処理での帯幅を渡します．DBDSLV ではこの値は無視されます（0 などを渡してください）．
- threads ……（入力　int 型）並列実行する際に起動するスレッドの数
- alg ……（入力　int 型）次元を調べるワークスペース変数．以下のいずれかの値を渡してください．
  - 0 …… DBDSLV の work 変数
  - 1 …… DGESLV, DGESLV1 の work 変数
  - 10 …… DBDSLV の iwork 変数

戻り値

(int 型) alg で指定した変数の必要な次元数を返します. malloc 関数などでメモリを確保する場合は, この次元数にデータ型のサイズ(sizeof(double)または sizeof(int))を乗じた値のサイズを確保してください.

4. サンプルプログラム

関数 DBDSLV のサンプル

以下のサンプルプログラムはsample_dbdslv.cppのファイル名で添付されています. このサンプルプログラムを実行するには,

```
g++ sample_dbdslv.cpp -lpthread -fopenmp -L. -lisvd -lgfortran -llapack -lblas
```

と入力してください(libisvd.aがソースコードと異なる場所にある場合は, -Lオプションで場所を指定してください).

```
#include <cstdio>
#include <cstdlib>
#include "isvd.h"

int main(){
    const int N = 500; // 行列サイズ
    const int P = 4; // 起動するスレッド数
    const int count_to_show = 5;
    double *diag, *subdiag, *U, *V, *work;
    int *UvecsOrder, *VvecsOrder, *iwork;
    int i, j;

    diag = (double*)malloc(sizeof(double) * N);
    subdiag = (double*)malloc(sizeof(double) * N);
    U = (double*)malloc(sizeof(double) * N * N);
    V = (double*)malloc(sizeof(double) * N * N);
    work = (double*)malloc(sizeof(double) * GetWorkSize(N, 0, P, 0));
    UvecsOrder = (int*)malloc(sizeof(int) * N);
    VvecsOrder = (int*)malloc(sizeof(int) * N);
    iwork = (int*)malloc(sizeof(int) * GetWorkSize(N, 0, P, 10));

    // 行列の初期化
    srand(0);
    for(i = 0; i < N; i++){
        diag[i] = (double)rand() / (double)RAND_MAX - 0.5; // 対角成分
```

```
    }
    for(i = 0; i < N - 1; i++){
        subdiag[i] = (double)rand() / (double)RAND_MAX - 0.5; // 副対角成分
    }

    // I-SVD法で特異値分解
    DBDSLV(N, diag, subdiag, U, V, UvecsOrder, VvecsOrder, work, iwork, P, 0);

    // 計算結果の一部を表示
    printf("Singular values = ¥n");
    for(i = 0; i < count_to_show; i++){
        printf("%le¥n", diag[i]);
    }
    printf("Matrix U = ¥n");
    for(i = 0; i < count_to_show; i++){
        for(j = 0; j < count_to_show; j++){
            printf("%le, ", U[UvecsOrder[j] * N + i]);
        }
        printf("¥n");
    }
    printf("Matrix V = ¥n");
    for(i = 0; i < count_to_show; i++){
        for(j = 0; j < count_to_show; j++){
            printf("%le, ", V[VvecsOrder[j] * N + i]);
        }
        printf("¥n");
    }

    free(diag); free(subdiag); free(U); free(V); free(work);
    free(UvecsOrder); free(VvecsOrder); free(iwork);
}
```

関数 DGESLV のサンプル

以下のサンプルプログラムはsample_dgeslv.cppのファイル名で添付されています．このサンプルプログラムを実行するには

```
g++ sample_dgeslv.cpp -lpthread -fopenmp -L. -lisvd -lgfortran -llapack -lblas
```

と入力してください(libisvd.aがソースコードと異なる場所にある場合は，-Lオプションで場所を指定してください)．

```
#include <cstdio>
#include <cstdlib>
#include "isvd.h"

int main(){
    const int N = 1000; // 行列サイズ
    const int lb = 100; // 帯幅
    const int P = 4; // 起動するスレッド数
    const int count_to_show = 5;
    double *A, *values, *U, *V, *work;
    int i, j;

    A = (double*)malloc(sizeof(double) * N * N);
    values = (double*)malloc(sizeof(double) * N);
    U = (double*)malloc(sizeof(double) * N * N);
    V = (double*)malloc(sizeof(double) * N * N);
    work = (double*)malloc(sizeof(double) * GetWorkSize(N, lb, P, 1));

    // 行列の初期化
    srand(0);
    for(i = 0; i < N * N; i++){
        A[i] = (double)rand() / (double)RAND_MAX - 0.5;
    }

    // I-SVD法で特異値分解
    DGESLV(N, lb, A, values, U, V, work, P, 0);

    // 計算結果の一部を表示
    printf("Singular values = ¥n");
    for(i = 0; i < count_to_show; i++){
        printf("%le¥n", values[i]);
    }
    printf("Matrix U = ¥n");
    for(i = 0; i < count_to_show; i++){
        for(j = 0; j < count_to_show; j++){
            printf("%le, ", U[j * N + i]);
        }
```

```
        printf("¥n");
    }
    printf("Matrix V = ¥n");
    for(i = 0; i < count_to_show; i++){
        for(j = 0; j < count_to_show; j++){
            printf("%le, ", V[j * N + i]);
        }
        printf("¥n");
    }

    free(A); free(values); free(U); free(V); free(work);
}
```