AA&S 疎行列線形ソルバ

説明書

２０１１年０９月　Ver.1.1

Advanced Algorithm & Systems

目次

１．概要

線形ソルバは以下の行列式の変数ベクトル x を解くものであり、偏微分方程式を陰解法で解く場合などに使用される。行列サイズ（行数と列数）が 10,000 を超えるようになると、通常の密行列用のソルバでなく、0 データを省略した疎行列用のソルバが必要になる。

Ax=b
A：係数行列
x：変数ベクトル
b：定数ベクトル

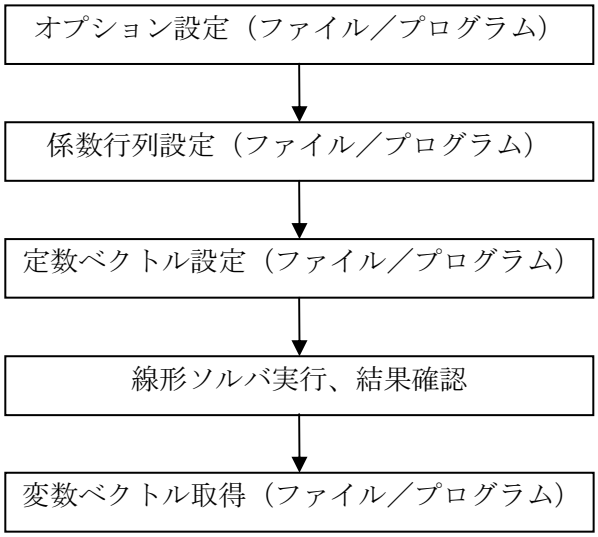
AA&S 疎行列線形ソルバは速度に優れた反復法と、安定性に優れた直接法の両方を実装している。また入出力機能やオプション機能が充実している。マルチスレッド処理や複素数対応も実装しており、汎用的で使いやすい線形ソルバである。

本線形ソルバでは以下の３つの解法を利用できる。

解法	反復法		直接法
	ILU0 前処理	ILU フィルイン前処理	PARDISO
	BiCGStab		
オプション調整	不要	必要	不要
速度	高速	場合により最高速	やや低速
安定性	中間	不安定	安定
メモリ消費	少ない	中間	多い

反復法は行列ごとの速度差が大きいため、直接法の方が早くなる場合もある。
安定性の高い直接法でもすべての行列が解ける訳ではない（困難な行列参照）。
ILU フィルインは最高速になる場合があるが、オプションの調整が必要なため簡単には利用できない。ほぼ同型の行列を何度も計算する場合のみ効果がある。

処理の流れは以下のようになる。ほとんどの機能でファイルからの入出力とプログラム内からの入出力の両方が利用できる。



本ソルバは Intel CPU を使った Linux マシンで共有ライブラリとして動作する。
呼び出しは C 言語から行う。

２．利用方法

２．１．利用手順

(1) インストール

配布されたファイルを指定された方法でインストールする。以下のディレクトリが作成される。

include　…　線形ソルバライブラリ用のインクルードファイル
lib　　…　線形ソルバライブラリ
sample　…　サンプルプログラム

(2) サンプル実行

サンプルディレクトリに移動後、以下のコマンドで環境変数のパス設定を行う

> cd <インストールパス>/sample
> source path_alsolv_bash　　（bash の場合）
> source path_alsolv_csh　　（csh の場合）

次の環境変数を自動的に設定する。

AASYS_ALSOLV_PATH：線形ソルバのパス、サンプルの make 時に使用
LD_LIBRARY_PATH　：ライブラリのパス、ライブラリの make と実行時に使用

サンプルプログラムを別の場所にコピーする。

> cd ..
> cp –r sample <コピー先パス>

コピーしたサンプルディレクトリに移動後、サンプルプログラムをコンパイルする。

> cd <コピー先パス>/sample
> make

サンプルプログラムを実行する。

> ./sample

サンプルプログラムからファイルを読み込んで実行する事もできる。固定ベクトルファイルがないと全部 1 になる。変数ベクトルは alsvSampleVecVar.mtx に保存される。

> ./sample <係数行列ファイル> [<固定ベクトルファイル>]

(3) 既存プログラム修正

サンプルプログラムを参考にして、プログラムを作成／修正して AA&S 線形ソルバを呼び出す。

行列計算の知識がなくても利用できる API になっているため、修正は比較的容易である。

alsolv.h をインクルードし、alsolv と pthread（マルチスレッド用ライブラリ）を動的リンク指定する。

make および実行時には環境変数 LD_LIBRARY_PATH に AA&S 線形ソルバライブラリのディレクトリを指定しておく必要がある。サンプルの path_alsolv_bash/csh 参照。

(4) オプションファイル修正

オプションファイルを修正して線形ソルバの動作を調整する。プログラムの修正なしにオプションを変更できる。反復法と直接法の切り替えもできる。

オプションは調整済みなのでそのままでも動作するが、解法とスレッド数は指定した方がいい。

詳細はオプション参照。

(5) 複数解法切り替えプログラム作成

処理中に特定の条件下で解法を切り替えたい場合は、オプションを変更する処理を入れる。

解法はプログラム内で直接指定することもできるし、複数のオプションファイルを作って読み込むようにする事もできる。

解法の種類により速度が変化するが、同時に複数の解法を実行しても遅くなるだけで意味がない。同様な行列計算を繰り返し行う場合は、最初に高速の解法で処理し、失敗したら低速の解法に切り替えると安定的かつ高速に処理できる。

(6) トラブル対応

トレースやエラーメッセージを参照すれば、利用者がある程度自力でトラブル対応できる。

サポートを受けたい場合は必要な情報をメール等で送る。

詳細はトラブル対応参照。

２．２．ライセンス表示

ライブラリの最初（alsvInit 関数）でライセンス表示が行われる。

ALSVWRN: License to <ユーザ名> until <有効期限>

有効期限が１ヶ月以内である場合は以下のような警告が表示される。

ALSVWRN: *** CAUTION: LICENSE WITHIN ONE MONTH ***

有効期限が切れると以下のエラーメッセージを表示して終了する。

ALSVERR: License load error (expiered date) in init alsv

２．３．データ形式と用語

行列は以下の疎行列形式で設定する。ベクトルは通常の固定サイズ形式である。

密行列

11	12	0
21	22	23
0	0	33

疎行列（0 データ省略）

データ番号	0	1	2	3	4	5
行番号	0	0	1	1	1	2
列番号	0	1	2	0	1	2
データ	11	12	23	21	22	33

行列は正方行列（行数＝列数）のみ可能である。データの大きさは以下の用語で呼ぶ。
サイズ　：行数＝列数＝ベクトルのサイズ
データ数：疎行列に入れるデータの数

疎行列データの順番に制約がないので入力しやすい。
ソルバ実行時に行番号＋列番号でソートした上、計算しやすい形式に変換される。そのためソルバの前後でデータの順番などが変わるので注意が必要である。

ファイル形式は Matrix Market という一般的なテキストファイル行列形式である。
詳細は以下の HP を参照。
http://math.nist.gov/MatrixMarket/
また、以下のフロリダ大 HP に大規模疎行列データサンプルがあり、動作確認で利用している。
http://www.cise.ufl.edu/research/sparse/matrices/
関数の行列番号は 0 ベース（C 型）だが、Matrix Market ファイルの行列番号は 1 ベース（FORTRAN 型）なので注意が必要である。ファイルの行列番号エラーの場合も 1 ベースで表示される。

精度を示す値には相対誤差ノルムと絶対誤差ノルムがある。
x の計算結果を x’とすると以下の値になる。
相対誤差ノルム：‖b - Ax’‖ / ‖b‖
絶対誤差ノルム：‖b - Ax’‖
相対誤差ノルムの方が汎用的なので、指定と結果取得ではこの値を使用する。
絶対誤差ノルムの方が計算は早いので、処理中はこの値を使用する。そのためエラーやトレースメッセージは主に絶対誤差ノルムで表示される。
変換比率はトレースで確認できる。

３．関数

３．１．全体管理

概要	初期処理
書式	#include " alsolv.h " ALSV_HANDLE alsvInit(int ver, const char* optFile)
引数と戻り値	int ver：(I) バージョン番号、ALSV_VER を設定 const char* optFile：(I) オプションファイル名、NULL なら既定値（alsv.ini） 戻り値：線形ソルバハンドラ、NULL なら失敗
説明	線形ソルバの初期処理を行う。

概要	終了処理
書式	#include " alsolv.h " void alsvTerm(ALSV_HANDLE alsvh)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ 戻り値：1 なら成功、0 なら失敗
説明	線形ソルバの終了処理を行う。

概要	オプションファイル読み込み
書式	#include " alsolv.h " int alsvReadOptFile(ALSV_HANDLE alsvh, const char* optFile)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ const char* optFile：(I) オプションファイル名 戻り値：1 なら成功、0 なら失敗
説明	オプションファイルを読み込む。

概要	オプション設定
書式	#include " alsolv.h " int alsvSetOpt(ALSV_HANDLE alsvh, const char* key, int intFlag, int valInt, double valDouble)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ const char* key：(I) オプションキー int intFlag：(I) int／double フラグ、1 なら int、0 なら double int valInt：(I) int 値、intFlag が 1 の場合のみ有効 double valDouble：(I) double 値、intFlag が 0 の場合のみ有効 戻り値：1 なら成功、0 なら失敗
説明	オプションを設定する。 valInt／valDouble のうち有効でない側は無視される。

概要	オプション取得
書式	#include " alsolv.h " int alsvGetOpt(ALSV_HANDLE alsvh, const char* key, int intFlag, int* valInt, double* valDouble)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ const char* key：(I) オプションキー int intFlag：(I) int／double フラグ、1 なら int、0 なら double int* valInt：(O) int 値のアドレス、intFlag が 1 の場合のみ有効 double* valDouble：(O) double 値のアドレス、intFlag が 0 の場合のみ有効 戻り値：1 なら成功、0 なら失敗
説明	オプションを取得する。 valInt／valDouble のうち有効でない側は無視される。

３．２．係数行列

係数行列はベクトルは Ax=b の A である。
疎行列形式（データ形式と用語参照）で係数行列を設定／取得する。
データはどのような順番でもよく、データに 0 があってもいい。
ソルバや結果確認で係数行列の整形と型変換を行う。行列データが変わるので注意が必要である。
関数内の番号はすべて 0 ベース（C 型）だが、ファイル内の行列番号は 1 ベース（FORTRAN 型）なので注意が必要である。

概要	係数行列ファイル読み込み
書式	#include " alsolv.h " int alsvMatCoefFileRead(ALSV_HANDLE alsvh, const char* file)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ const char* file：(I) 係数行列ファイル名 戻り値：1 なら成功、0 なら失敗
説明	Matrix Market 形式の係数行列ファイルを読み込む。既存データは破棄される。 疎行列形式で読み込む。 複素数形式なら複素数として読み込み、それ以外（実数、整数、パターン）は実数として読み込む。

概要	係数行列ファイル書き込み
書式	#include " alsolv.h " int alsvMatCoefFileWrite(ALSV_HANDLE alsvh, const char* file)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ const char* file：(I) 係数行列ファイル名 戻り値：1 なら成功、0 なら失敗
説明	Matrix Market 形式の係数行列ファイルを書き込む。 疎行列の実数または複素数形式で書き込む。

概要	係数行列作成
書式	#include " alsolv.h " int alsvMatCoefCreate(ALSV_HANDLE alsvh, int comp, int size, int dataNum)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int comp：(I) 複素数フラグ、1 なら複素数、0 なら実数 int size：(I) 行列のサイズ int dataNum：(I) 疎行列のデータ数 戻り値：1 なら成功、0 なら失敗
説明	係数行列を作成し、必要な領域を確保する。既存データは破棄される。 作成しても設定されないデータ番号があるとソルバや結果確認でエラーなる。

概要	係数行列情報取得
書式	#include " alsolv.h " int alsvMatCoefInfo(ALSV_HANDLE alsvh, int* comp, int* size, int* dataNum)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int* comp：(I) 複素数フラグ、1 なら複素数、0 なら実数 int* size：(O) 行列のサイズ int* dataNum：(O) 疎行列のデータ数 戻り値：1 なら成功、0 なら失敗
説明	係数行列の情報を取得する。 ソルバや結果確認で行列データが変わるので注意が必要である。

概要	係数行列設定
書式	#include " alsolv.h " int alsvMatCoefSet(ALSV_HANDLE alsvh, int row, int col, int pos, double real, double imag, int restoreType);
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int row：(I) 行列の行番号 int col：(I) 行列の列番号 int pos：(I) 疎行列のデータ位置番号 double real：(I) 実数値 double imag：(I) 虚数値、複素数ベクトルの場合のみ有効 int restoreType：(I) 型戻しフラグ、1 なら型戻しあり、0 ならなし（推奨） 戻り値：1 なら成功、0 なら失敗
説明	係数行列を設定する。 ソルバや結果確認で行列データが変わるので注意が必要である。 データ番号を直接指定できるが、ソルバなどの前後で同じ番号でも別データになる。 restoreType を 0 にしてソルバや結果確認後に呼び出すとエラーになる。安全のため 0 にしておくのが望ましい。 型変換後のデータはそのままでは設定できない。restoreType を 1 にすると変換前の型に戻す。ただしデータ整形が行われているため完全に戻るわけではない。0 データ削除とソートが行われた状態になる。

概要	係数行列順次取得
書式	#include " alsolv.h " int alsvMatCoefGetNext(ALSV_HANDLE alsvh, int* row, int* col, int* pos, double* real, double* imag, int* end, int first)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int* row：(O) 行列の行番号 int* col：(O) 行列の列番号 int* pos：(O) 疎行列のデータ位置番号 double* real：(O) 実数値 double* imag：(O) 虚数値、複素数ベクトルの場合のみ有効 int* end：(O) 終端フラグ、1 なら終端、0 ならそれ以外 int first：(I) 先頭フラグ、1 なら先頭、0 ならそれ以外 戻り値：1 なら成功、0 なら失敗
説明	係数行列を順番に取得する。 終端の場合は終端フラグ以外のデータを返さない。 順番に取得する途中で行列を変更したりソルバや結果確認を呼んではならない。 ソルバや結果確認で行列データが変わるので注意が必要である。 データ番号を直接指定できないので係数行列設定より安全である。 型変換後のデータでもそのまま取得できる。ただし 0 データ削除とソートが行われた状態になる。

３．３．定数ベクトル

定数ベクトルは Ax=b の b である。
通常の固定サイズ形式で定数ベクトルを設定／取得する。
位置番号は通常の配列番号（0 ベース）である。
係数行列が複素数で定数ベクトルが実数の場合、ソルバや結果確認で複素数に変換する。

概要	定数ベクトルファイル読み込み
書式	#include " alsolv.h " int alsvVecFixFileRead(ALSV_HANDLE alsvh, const char* file)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ const char* file：(I) 定数ベクトルファイル名 戻り値：1 なら成功、0 なら失敗
説明	Matrix Market 形式の定数ベクトルファイルを読み込む。既存データは破棄される。 密行列の実数、整数、複素数形式と、疎行列のパターン形式を読み込める。 複素数形式なら複素数として読み込み、それ以外（実数、整数、パターン）は実数として読み込む。 行列の 1 列目のみ使用する。それ以上の列がある場合は警告を出して無視する。

概要	定数ベクトルファイル書き込み
書式	#include " alsolv.h " int alsvVecFixFileWrite(ALSV_HANDLE alsvh, const char* file)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ const char* file：(I) 定数ベクトルファイル名 戻り値：1 なら成功、0 なら失敗
説明	Matrix Market 形式の定数ベクトルファイルを書き込む。 密行列の実数または複素数形式で書き込む。

概要	定数ベクトル作成
書式	#include " alsolv.h " int alsvVecFixCreate(ALSV_HANDLE alsvh, int comp, int size, double real, double imag)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int comp：(I) 複素数フラグ、1 なら複素数、0 なら実数 int size：(I) サイズ double real：(I) 実数初期値 double imag：(I) 虚数初期値、comp が 1 の場合のみ有効 戻り値：1 なら成功、0 なら失敗
説明	定数ベクトルを作成する。既存データは破棄される。 必要な領域を確保し、すべてのデータに初期値を入れる。

概要	定数ベクトル情報取得
書式	#include " alsolv.h " int alsvVecFixInfo(ALSV_HANDLE alsvh, int* comp, int* size)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int* comp：(O) 複素数フラグ、1 なら複素数、0 なら実数 int* size：(O) サイズ 戻り値：1 なら成功、0 なら失敗
説明	定数ベクトルの情報を取得する。

概要	定数ベクトル設定
書式	#include " alsolv.h " int alsvVecFixSet(ALSV_HANDLE alsvh, int pos, double real, double imag)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int pos：(I) ベクトルの位置番号 double real：(I) 実数値 double imag：(I) 虚数値、複素数ベクトルの場合のみ有効 戻り値：1 なら成功、0 なら失敗
説明	定数ベクトルを設定する。

概要	定数ベクトル取得
書式	#include " alsolv.h " int alsvVecFixGet(ALSV_HANDLE alsvh, int pos, double* real, double* imag)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int pos：(I) ベクトル内の位置番号 double* real：(O) 実数値 double* imag：(O) 虚数値、複素数ベクトルの場合のみ有効 戻り値：1 なら成功、0 なら失敗
説明	定数ベクトルを取得する。

３．４．変数ベクトル

変数ベクトルは Ax=b の x である。
通常の固定サイズ形式で定数ベクトルを設定／取得する。
位置番号は通常の配列番号（0 ベース）である。
ソルバは自動的に変数ベクトルを作成する。既にあるデータは削除される。
結果確認を独自に行うなどの場合のみ作成する。
係数行列が複素数で変数ベクトルが実数の場合、結果確認で複素数に変換する。

概要	変数ベクトルファイル読み込み
書式	#include " alsolv.h " int alsvVecVarFileRead(ALSV_HANDLE alsvh, const char* file)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ const char* file：(I) 変数ベクトルファイル名 戻り値：1 なら成功、0 なら失敗
説明	Matrix Market 形式の変数ベクトルファイルを読み込む。既存データは破棄される。 密行列の実数、整数、複素数形式と、疎行列のパターン形式を読み込める。 複素数形式なら複素数として読み込み、それ以外（実数、整数、パターン）は実数として読み込む。 行列の 1 列目のみ使用する。それ以上の列がある場合は警告を出して無視する。

概要	変数ベクトルファイル書き込み
書式	#include " alsolv.h " int alsvVecVarFileWrite(ALSV_HANDLE alsvh, const char* file)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ const char* file：(I) 変数ベクトルファイル名 戻り値：1 なら成功、0 なら失敗
説明	Matrix Market 形式の変数ベクトルファイルを書き込む。 密行列の実数または複素数形式で書き込む。

概要	変数ベクトル作成
書式	#include " alsolv.h " int alsvVecVarCreate(ALSV_HANDLE alsvh, int comp, int size, double real, double imag)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int comp：(I) 複素数フラグ、1 なら複素数、0 なら実数 int size：(I) サイズ double real：(I) 実数初期値 double imag：(I) 虚数初期値、comp が 1 の場合のみ有効 戻り値：1 なら成功、0 なら失敗
説明	変数ベクトルを作成する。既存データは破棄される。 必要な領域を確保し、すべてのデータに初期値を入れる。

概要	変数ベクトル情報取得
書式	#include " alsolv.h " int alsvVecVarInfo(ALSV_HANDLE alsvh, int* comp, int* size)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int* comp：(O) 複素数フラグ、1 なら複素数、0 なら実数 int* size：(O) サイズ 戻り値：1 なら成功、0 なら失敗
説明	変数ベクトルの情報を取得する。

概要	変数ベクトル設定
書式	#include " alsolv.h " int alsvVecVarSet(ALSV_HANDLE alsvh, int pos, double real, double imag)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int pos：(I) ベクトル内の位置番号 double real：(I) 実数値 double imag：(I) 虚数値、複素数ベクトルの場合のみ有効 戻り値：1 なら成功、0 なら失敗
説明	変数ベクトルを設定する。

概要	変数ベクトル取得
書式	#include " alsolv.h " int alsvVecVarGet(ALSV_HANDLE alsvh, int pos, double* real, double* imag)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ int pos：(I) ベクトル内の位置番号 double* real：(O) 実数値 double* imag：(O) 虚数値、複素数ベクトルの場合のみ有効 戻り値：1 なら成功、0 なら失敗
説明	変数ベクトルを取得する。

３．５．ソルバ

ソルバの実行と結果確認を行う。処理後に係数行列の形式が変わるので注意する必要がある。

概要	ソルバ
書式	#include " alsolv.h " int alsvSolv(ALSV_HANDLE alsvh)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ 戻り値：1 なら成功、0 なら失敗
説明	線形ソルバを実行する。 ソルバ実行前に係数行列の 0 データ削除、ソート、型変換を行う。このデータは元に戻らないので注意が必要である。 作成しても設定されない係数行列データ番号があるとソルバでエラーなる。 係数行列が複素数で定数ベクトルが実数の場合、定数ベクトルを複素数に変換する。それ以外のデータ型不一致はエラーになる。 自動的に変数ベクトルを作成する。既にあるデータは削除される。 ソルバが正常終了しても結果は保証されない。反復法で指定した精度も保障されない。必ず結果確認を行う必要がある。

概要	結果確認
書式	#include " alsolv.h " int alsvCheckResult(ALSV_HANDLE alsvh, double* normRel, double* normAbs)
引数と戻り値	HANDLE alsvh：(I) 線形ソルバハンドラ double* normRel：(O) 相対誤差ノルム double* normAbs：(O) 絶対誤差ノルム 戻り値：1 なら成功、0 なら失敗
説明	線形ソルバの結果を確認する。 相対／絶対誤差ノルムについてはデータ形式と用語参照。相対の方が汎用的である。 相対誤差ノルムがオプションの CHECK_RESULT_LIM 以上ならエラーになる。 ソルバを行わずに結果確認のみでも実行できる。別途変数ベクトルを作成しておく。この場合は結果確認で係数行列の 0 データ削除、ソート、型変換が行われる。 係数行列が複素数で定数／変数ベクトルが実数の場合、定数／変数ベクトルを複素数に変換する。 それ以外のデータ型不一致はエラーになる。

４．オプション

４．１．概要

オプションはオプションファイルとプログラムの両方から設定／取得が可能である。
各キーごとに int 値か double 値を設定できる。ここで記述した以外のキーは無効である。

オプションファイルの規定値は alsv.ini である。形式は以下のようになる。
キーと値を空白またはタブで分けて指定する。double 値は”1.0e-2”または”0.01”の形式で指定する。
先頭が”#”の行と、すべて空白／タブの行は無視される。

```
# Comment
FILE_VER              1
SOLVER                0
THREADS               2
CHECK_RESULT_LIM      1.0e-2
```

基本的なオプションは以下の通りである。ファイルバージョンは必ず指定する。
オプションは調整済みなのでそのままでも動作するが、解法とスレッド数は指定した方がいい。

(1) ファイルバージョン
FILE_VER に 1 を指定する。
(2) 解法
SOLVER に直接法／反復法を指定する。
ILU フィルイン前処理を使用する場合は ILU_FILLIN と ILU_FILLINMIN も指定する。
ILU_FILLINMIN は調整が必要であり、簡単には使えない。
(3) スレッド数
THREADS にスレッド数を指定する。
(4) 精度
CHECK_RESULT_LIM に結果確認の精度を指定する。
ITER_RESOLVE と ITER_GIVEUP_CHECK に反復法の目標精度を指定する。
相対誤差ノルムで指定する（データ形式と用語参照）。
(5) 反復回数
ITER_MAXSTEP と ITER_MAXSTEP_SIZE に反復法の最大ステップ数を指定する。
(6) トレース
トレースを出す事で線形ソルバの動作を確認できる。トレースは標準出力に表示される。
TRACE、ITER_STEPTRACE、PARDISO_TRACE を指定する。必要なら ILU_TRACE も指定する。
トレースの多くは絶対誤差ノルムで表示される（データ形式と用語参照）。

４．２．全体

キー	種別	規定値	説明
FILE_VER	int	-1	オプションファイルのバージョン 常に 1 を設定、プログラムからの設定は不要
TRACE	int	0	トレースフラグ、1 ならあり、0 ならなし
SOLVER	int	0	ソルバ、0 なら反復法、10 なら直接法
THREADS	int	1	スレッド数 0 なら CPU スレッド数、1 以上なら指定値 CPU スレッド数は CPU ごとに決まる並列処理可能な数、コア数より多い場合はコア数までのほうが安全
CHECK_RESULT_LIM	double	1.0e-3	結果確認の誤差ノルム限界値 相対誤差ノルムがこれ以上ならエラー 0 ならチェックなし
SAVE_ALL	int	0	全体保存フラグ、1 ならあり、0 ならなし ソルバ終了時に全体情報をファイル保存 alsvSave0001～9999：保存フォルダ alsv.ini：オプション matCoef.mtx：係数行列 vecFix.mtx：固定ベクトル vecVar.mtx：変数ベクトル vecDiff.mtx：残差ベクトル、Ax-b result.txt：結果、相対／絶対誤差ノルム
REV_DIAGZERO	int	1	対角 0 データ補正フラグ 0 ならなし、1 なら自動、2 なら強制 補正する場合は、ソルバ処理中のみ対角成分の 0 データを微小値に変換 自動の場合、対角 0 ありかつ反復なら補正 対角 0 ありで補正なしだと ILU 前処理失敗
REV_DIAGZERO_VAL	double	1.0e-12	対角 0 補正時に設定する微小値
DIAG_SCALE	int	0	内部用の値、使用しない事
USER_I1	int	0	利用者が任意に使える値
USER_I2	int	0	利用者が任意に使える値
USER_I3	int	0	利用者が任意に使える値
USER_I4	int	0	利用者が任意に使える値
USER_D1	double	0.0	利用者が任意に使える値
USER_D2	double	0.0	利用者が任意に使える値
USER_D3	double	0.0	利用者が任意に使える値
USER_D4	double	0.0	利用者が任意に使える値

４．３．反復法

キー	種別	規定値	説明
ILU_TRACE	int	0	ILU トレース行間隔、0 ならなし 指定間隔ごとにトレース表示 小さいと遅くなるので 100 以上推奨 全体の TRACE が 0 なら表示されない
ILU_FILLIN	int	0	ILU フィルインフラグ 1 なら ILU フィルイン、0 なら ILU0 ILU_FILLINMIN の調整が必要 正しく調整すると最高速になる事がある 他の解法で解けないデータが解けるようになる事は少ない 複素数だとあまり早くならない
ILU_FILLINMIN	double	1.0e-3	ILU フィルイン最小値 ILU 処理中に行列式を変更する値がこれ以上なら 0 データ箇所にデータを追加 値が小さいほどフィルインが増える 行列に対する以下の相対値と比較 　行列データ絶対値の最大値 * FILLINMIN 最高速を出すには各データごとに調整が必要 1.0e-1～-8 程度で調整、-3～-5 程度が多い トレースの“fill-in ... increase ...”で増加データ数を確認可能、0 なら効果なし、多すぎるとメモリ量が増え遅い
ILU_REV_DIAGZERO_VAL	double	1.0e-12	ILU 処理中に対角 0 が発生した場合、そこに設定する微小値 REV_DIAGZERO_VAL と違い ILU 処理内部でのみ使用
ILU_ACCEL	double	1.0	ILU 加速係数 1～1.3 程度で調整、まれに多少高速化
ITER_PREILU	int	1	反復 ILU 前処理フラグ 1 ならあり、0 ならなし なしでも収束する場合はあるが非常に低性能 なしで ITER_R_TYPE が 0 だと失敗する可能性が大きい
ITER_RESOLVE	double	1.0e-10	目標誤差 相対誤差ノルムがこれ以下なら成功 成功してもこの精度は保障されず、結果確認関数のみ保障される 途中でギブアップする可能性あり
ITER_STEPTRACE	int	0	反復ステップトレース間隔、0 ならなし 指定ステップ間隔ごとにトレース表示 小さいと遅くなるので 100 以上推奨 全体の TRACE が 0 なら表示されない

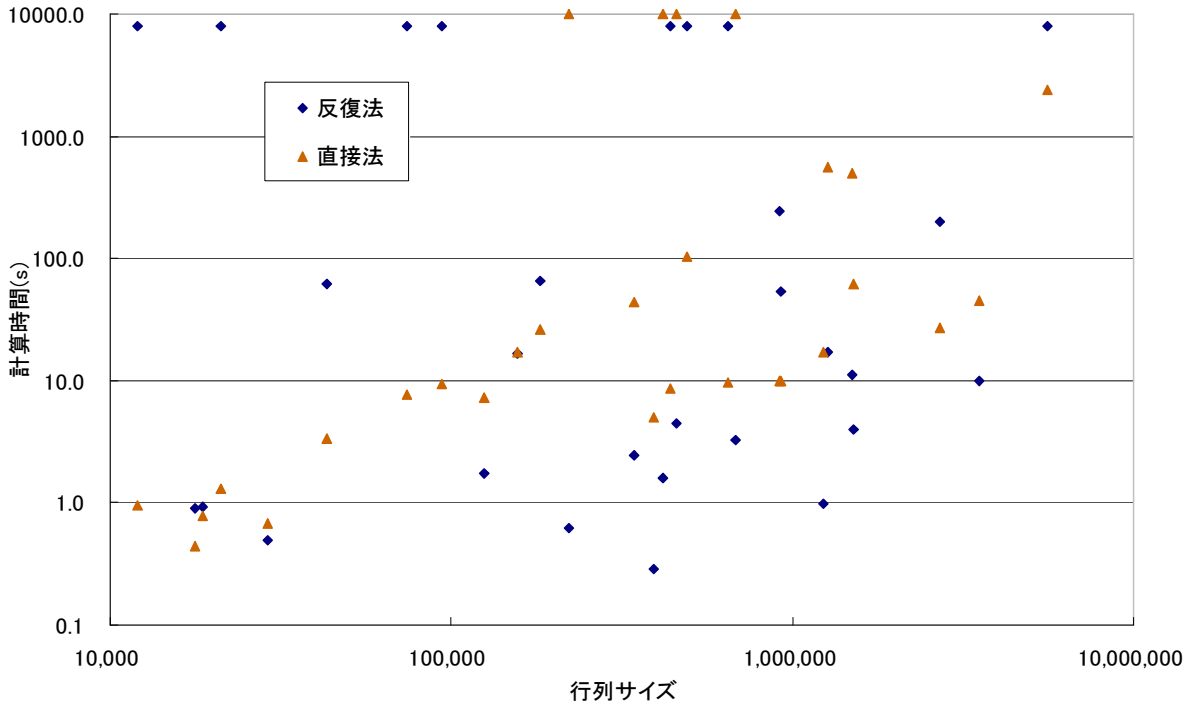
キー	種別	規定値	説明
ITER_MAXSTEP	int	5000	反復最大ステップ数
ITER_MAXSTEP_SIZE	double	0.0	行列サイズごとの反復最大ステップ数加算値 以下の値を最大ステップ数に加算 　行列サイズ * MAXSTEP_SIZE
ITER_GIVEUP	int	200	ギブアップステップ間隔、0 ならなし ある程度誤差が減り、それ以上収束しなくなった場合に処理を正常終了 相対誤差ノルムがこれ以下なら次回チェック 　RESOLVE * GIVEUP_CHECK 次回チェック時にノルムがこれ以上なら終了 　前回ノルム * GIVEUP_REDUCT
ITER_GIVEUP_CHECK	double	1.0e5	ギブアップチェック値 誤差ノルムをチェックする値で、RESOLVE に掛ける比率
ITER_GIVEUP_REDUCT	double	0.5	ギブアップ減少値 ギブアップステップ間隔後にここまで減らなければギブアップ
ITER_CORRECT_R	int	100	残差ベクトル補正間隔、0 ならなし 残差は固定ベクトル b までの差 指定ステップ間隔ごとに残差ベクトルを補正 誤差の拡大を抑える効果あり
ITER_R_TYPE	int	0	初期シャドウ残差ベクトル種別 0 なら r0（=b-Ax)、1 なら乱数 r0 の方が多くの場合高速 r0 で ITER_PREILU が 0 だと失敗する可能性が大きい
SAFE_CR	int	1	内部用の値、使用しない事
SAFE_CORRECT_Y	int	1	内部用の値、使用しない事
STABL_L	int	1	内部用の値、使用しない事

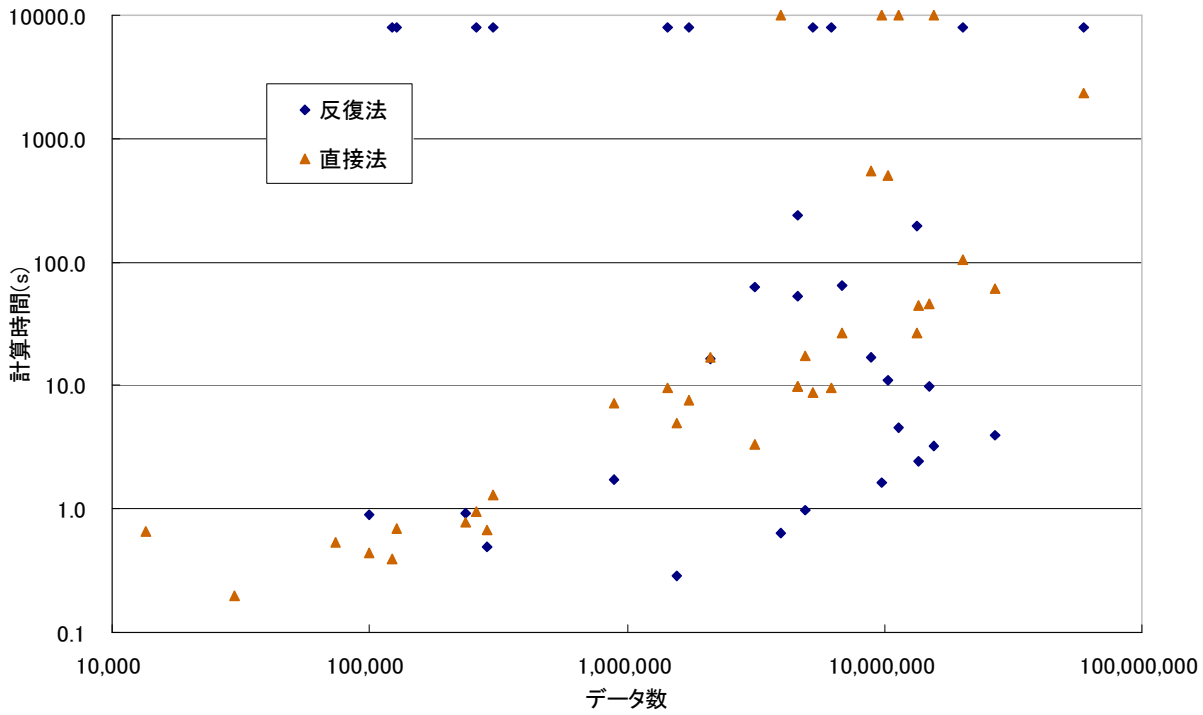
４．４．直接法

キー	種別	規定値	説明
PARDISO_TRACE	int	0	PARDISO トレースフラグ 1 ならあり、0 ならなし 全体の TRACE が 0 なら表示されない
PARDISO_MEM_LIM	int	0	PARDISO メモリ制限、0 ならなし 指定値 kb より予想使用メモリが大きければエラー
PARDISO_STAT	int	0	PARDISO 独自の詳細な状況説明表示 1 ならあり、0 ならなし
PARDISO_ITER	int	2	直接法終了後に簡単な反復法で精度を改良する最大回数 大きくするとエラーになる場合あり PARADISO エラーでメッセージに以下の表記がある場合、この値を小さくすると回避できる可能性あり “iterative refinement problem”
PARDISO_PIVOT	int	13	軸要素入替定数、1.0 E-(指定値) 行列の軸要素（ピボット）の値が小さい場合に軸を入れ替える目安 指定値が大きい（入替定数が小さい）と入替回数は多くなるが、あまり精度向上の効果なし

５．性能

以下に行列サイズ、データ数、計算時間のグラフを示す。反復法は ILU0 前処理である。
解法以外のオプションは規定値である。調整済みなので基本的に変える必要はない。
精度も規定値である。計算結果の相対誤差ノルムはデータにより幅があり、1.0e-4～-17 である。多くは 1.0e-11～-13 程度である。可能なら高精度で、無理なら低精度で解くようになっている。
最上部のデータは一方のソルバのみ解けたものである。





ILU フィルインを使うと最高で 2～3 倍程度速くなる。速くならない場合も多い。
ILU 前処理は処理時間が短いのでシングルスレッドで動作する。

メモリ消費量の目安は以下のようになる。データごとに差異があるため、おおよその値である。
ILU フィルインは増加する行列要素の数に依存する。
直接法は行列の種類ごとに大きく異なり明確でない。

解法	反復法（ILU0）	反復法（ILU フィルイン）	直接法
メモリ消費量 （メモリバイト数／データ数）	50 程度	100～150 程度	数百～数千

６．トラブル対応

６．１．エラーと対応策

主なエラーと対応策を以下に示す。行列変更については困難な行列参照。
エラーメッセージは”ALSVERR:...”と表示される。
まれに警告メッセージ” ALSVWRN:...”が表示される。動作可能であるが表示を確認した方がいい。

概要	計算中に 0 割などが発生して計算失敗
起こりやすい状況	困難な行列、反復法
判別方法	ソルバの戻り値がエラー、エラーメッセージに”nan error”
対処	解法を変える、可能なら行列変更

概要	メモリ不足
起こりやすい状況	行列サイズが大きい、困難な行列、直接法
判別方法	ソルバの戻り値がエラー、エラーメッセージに以下の表示 ”Alloc error”、”too large memory”、”not enough memory” top コマンドなどでメモリを確認すると 100%近い ファイルアクセスが頻発して遅い（メモリスワップ発生）
対処	解法を変える、行列サイズを減らす、可能なら行列変更

概要	処理が非常に長時間
起こりやすい状況	行列サイズが大きい、困難な行列、反復法
判別方法	エラーは発生しないが処理が終わらない 反復法トレースを見ると収束しない 直接法トレースを見ると予想使用メモリが大きい
対処	kill コマンド等でプロセス停止 解法を変える、行列サイズを減らす、可能なら行列変更

概要	反復法最大ステップ数超過
起こりやすい状況	困難な行列、反復法
判別方法	ソルバの戻り値がエラー、エラーメッセージに” Step over”
対処	解法を変える、可能なら行列変更 最大ステップ数を増やしても効果がない事が多い（収束不能）

概要	精度不足
起こりやすい状況	困難な行列
判別方法	結果確認の戻り値がエラー、エラーメッセージに” Limit ... over” 結果確認した誤差ノルムが大きすぎる
対処	解法を変える、可能なら行列変更

概要	直接法後の反復改良失敗
起こりやすい状況	直接法 PARDISO_ITER オプションが大きい 規定値ならほとんど起こらない
判別方法	ソルバの戻り値がエラー エラーメッセージに“iterative refinement problem”
対処	PARDISO_ITER を小さくするする

６．２．困難な行列

解く事が困難な行列の種類を以下に示す。可能ならば避ける事が望ましい。
これらはトレースの分析結果で確認できる。

(1) 複素数
複素数は実数より解ける確率が低い。反復法／直接法のうち一方だけが解ける場合も多くなる。
(2) 巨大データ
サイズやデータ数が非常に大きいとメモリ不足で解けなくなる。
(3) 対角 0 データ
対角成分に 0 データがあると反復法で解くのは難しい。直接法でも 0 データが多いと解けない場合がある。
(4) 行データ数最小 0
まるごとデータがない行があると解くのが非常に難しい。
(5) 行データ数最大巨大
サイズの 1 割を超えるような行データ数があると反復法ではほとんど解けない。普通はすぐ終わる ILU が非常に遅くなる場合もある。
直接法でも解くのが難しくなり、解けても長時間かかったりする。

６．３．トレースによる分析

トレースにより処理の状況を確認できる。”ALSVTRC:...”と表示される。
オプションの TRACE を 1、ITER_STEPTRACE を 100 にした場合の実行例を以下に示す。

```
ALSVTRC: Init ver 1 option file 'alsv.ini' ver 1
ALSVTRC: Open matCode='matrix coordinate real general'
ALSVTRC: Get matrix size 155924 - 5416358
ALSVTRC: Open matCode='matrix array real general'
ALSVTRC: Get vector size 155924
ALSVTRC: Option: Solver
ALSVTRC: -  USER_I1 0, USER_I2 0, USER_I3 0,
ALSVTRC: -  USER_I4 0, FILE_VER 1, TRACE 1,
ALSVTRC: -  SOLVER 0, THREADS 4, REV_DIAGZERO 1,
ALSVTRC: -  SAVE_ALL 0, DIAG_SCALE 0, ILU_TRACE 0,
ALSVTRC: -  ILU_FILLIN 0, ITER_PREILU 1, ITER_CORRECT_R 100,
ALSVTRC: -  ITER_STEPTRACE 100, ITER_MAXSTEP 5000, ITER_R_TYPE 0,
ALSVTRC: -  ITER_GIVEUP 200, SAFE_CR 1, SAFE_CORRECT_Y 1,
ALSVTRC: -  STABL_L 1, PARDISO_TRACE 1, PARDISO_MEM_LIM 0,
ALSVTRC: -  PARDISO_STAT 0, PARDISO_ITER 2, PARDISO_PIVOT 13,
ALSVTRC: -  USER_D1 0.0000e+00, USER_D2 0.0000e+00,
ALSVTRC: -  USER_D3 0.0000e+00, USER_D4 0.0000e+00,
ALSVTRC: -  REV_DIAGZERO_VAL 1.0000e-12, CHECK_RESULT_LIM 1.0000e-03,
ALSVTRC: -  ILU_REV_DIAGZERO_VAL 1.0000e-12, ILU_ACCEL 1.0000e+00,
ALSVTRC: -  ILU_FILLINMIN 1.0000e-03, ITER_MAXSTEP_SIZE 0.0000e+00,
ALSVTRC: -  ITER_RESOLVE 1.0000e-10, ITER_GIVEUP_CHECK 1.0000e+05,
ALSVTRC: -  ITER_GIVEUP_REDUCT 5.0000e-01,
ALSVTRC: Analyse in solver data type real
ALSVTRC:  - abs / rel norm 1.5291e+12
ALSVTRC:  - mat size 155924 data num 2094873
ALSVTRC:  - abs diag min 3.2174e-05 diag max 6.0169e+05 max 6.5632e+11
ALSVTRC:  - diag zero 0 -> 0 row data num min 1 max 6931
ALSVTRC: ILU 0 accel 1.0000e+00
ALSVTRC: ILU 0 L 1293803 U 645146 All 2094873
ALSVTRC: Iter stab pre 1 step trace 100 norm-0 1.5291e+12
ALSVTRC:  - resolve 1.0000e-10 max k 5000 correct r 100 rtype 0
ALSVTRC:  - giveup 200 check 1.0000e-05 reduct 5.0000e-01
ALSVTRC: Iter stab loop step 100 abs norm 1.7705e+07 resolve 1.5291e+02
ALSVTRC: Iter stab loop step 200 abs norm 4.7095e+05 resolve 1.5291e+02
ALSVTRC: Iter stab loop step 300 abs norm 2.0221e+05 resolve 1.5291e+02
ALSVTRC: Iter stab loop step 400 abs norm 9.6288e+03 resolve 1.5291e+02
ALSVTRC: Iter stab loop step 500 abs norm 6.9204e+02 resolve 1.5291e+02
ALSVTRC: Iter stab end 1 step 576 abs norm 1.0197e+02 resolve 1.5291e+02
ALSVTRC: Time Solver:
ALSVTRC:  - times 16.660s clock 64.990s omtime 16.659s
ALSVTRC: Check solver result norm rel 6.6689e-11 abs 1.0197e+02
```

主なトレースを以下に示す。

キーワード	概要
Option	オプションの内容
Analyse 　data type 　abs / rel norm 　mat size 　data num 　abs diag min 　diag max 　max 　diag zero 0 -> 0 　row data num	データの分析結果、困難な行列の判別に有効 データ種別（実数／複素数） 絶対誤差ノルムと相対誤差ノルムの変換比率（‖b‖） 行列サイズ 行列データ数 行列対角成分の絶対値最小値 行列対角成分の絶対値最大値 行列全データの絶対値最大値 行列対角 0 データ数、”->”以降は補正値 行列の 1 行データ数最小／最大値
Iter stab loop step	反復ステップの誤差ノルムと目標精度、収束状況を確認 ステップ番号は 0 ベースで表示される
PARDISO symbolic factorization end memory	直接法の予想使用メモリ
Time	計算時間、times と omtime は通常の計算時間 clock はスレッドごとの計算時間の総和

６．４．サポート

サポートを受ける場合は以下の情報をメール等で送る。多い方がサポートを受けやすい。

- 実行環境
- 作成プログラム内容
- エラーメッセージ
- トレース
- 全体情報ファイル保存データ（オプションの SAVE_ALL）