ユーザズ・ガイド
KMATHLIB API

Version 1.0

大規模並列数値計算技術研究チーム
理化学研究所
計算科学研究機構

2016 年 4 月 11 日

目 次

第1章　はじめに

KMATHLIB API は，多様な数値計算ライブラリを共通のインターフェースを介して使用するための API である．本 API は，高性能ライブラリを積極的にソフトウェアに組み込むことを支援し，計算科学ソフトウェア開発者の負担を軽減することを目的としている．

計算科学ソフトウェアの開発においては高性能数値計算ライブラリの利用が必須であり，しばしば複数の数値計算ライブラリが同時に使用される．しかしながら，一般にライブラリのデータ構造や使用方法はそれぞれにまったく異なっており，開発者はこれらの差異を理解し，また，必要に応じてデータ構造の変換機能を実装しなければならない．このために，開発されるプログラムは煩雑なものになり，計算科学ソフトウェアの開発と保守に必要な労力は増大し，また，計算科学ソフトウェア開発者に新たなライブラリの採用を躊躇させる要因にもなりえる．

KMATHLIB API を利用することにより，開発者は多様な数値計算ライブラリ（PETSc[5, 6, 7], ScaLAPACK[4]，FFTW[9]，EigenExa[10] 等）を共通のインターフェースを介してライブラリ間の差異を意識せずに統一的な手順で利用することが可能になり，前述の数値計算ライブラリ使用時の負担が軽減される．

KMATHLIB API は特定の計算機アーキテクチャに依存しないように設計されており，現在までに，京コンピュータ，富士通 PRIMEHPC FX10，同 FX100，Intel x86 系プロセッサを搭載するクラスタシステムなど，多くの高性能計算プラットフォームで動作することが確認されている．なお，KMATHLIB API を利用可能なプログラミング言語は，現在は Fortran 90/95 のみである．

本ドキュメントは KMATHLIB API 1.0 のユーザズ・マニュアルである．次章以降では，KMATHLIB API の基本的仕組み，KMATHLIB API パッケージのビルド手順，KMATHLIB API を使用したソフトウェア開発手法および KMATHLIB API の仕様について述べる．

利用許諾/Copyright

LICENCE.txt

第2章　KMATHLIB APIとその機能

KMATHLIB API は様々な数値計算ソフトウェア（ソルバ）を使用するための API である．API と共にいくつかの補助機能が提供される．

KMATHLIB API とその補助機能群（KMATHLIB API 機能）を使用したソフトウェア開発では，各種のソルバを使用する際，個別のソルバインターフェースを直接に使用するのではなく，KMATHLIB API を呼び出すことにより間接的にソルバの機能を利用する．KMATHLIB API は各種のソルバを統一的手段により実行することが可能になるように設計されており，ソルバごとのデータ形式や使用手順の違いを意識せずにソルバを利用したソフトウェアの開発が可能になる．

本章では，KMATHLIB API と補助機能群，およびそれらの注意点について説明する．

2.1　階層構造

KMATHLIB API 機能は，利用者が直接使用する KMATHLIB API，プラグイン管理機能，個別のソルバを扱うためのプラグインなどに分けられ，図 2.1 に示す階層構造をもつ．KMATHLIB API および標準形式のデータ（第 2.2 節にて説明）を扱うための一部の PETSc の機能のみが，KMATHLIB API 機能を利用するソフトウェア（アプリケーション）から直接呼び出される．プラグイン管理機能やプラグインの呼び出しは KMATHLIB API から内部的に行われ，アプリケーションがこれらを直接呼び出す必要はない．プラグイン管理機能は，個別のプラグインを使用する際の初期化，終了，データ変換の必要性の判定，パラメータ設定などを行うための機能がまとめられている．プラグインは個別のソルバを扱うための機能がまとめられており，データ形式の変換や，KMATHLIB API を通じてユーザの設定したパラメータに応じたソルバ・ルーチンの選択，実際のソルバ・ルーチンの呼び出しが行われる．

2.2　データ変換機能

KMATHLIB API では，データ形式の異なるソルバの統一的な方法による利用を実現するため，KMATHLIB API の入出力変数として使用するデータ形式（標準形式）を定めている．

プラグインの機能を通じてソルバを呼び出す際には，ソルバで使用されるデータ形式（内部形式）との間で自動的に入出力変数のデータ変換を行う．なお，KMATHLIB API では入出力変数のデータ変換は必ず行われるわけではなく，プラグイン管理機能に組み込まれたデータ変換の必要性の判定機能を通じて不要な変換を抑制することがある．

KMATHLIB API 機能の標準形式として PETSc で用いられるデータ形式が採用されており，そのために，KMATHLIB API 機能の一部では PETSc の機能が使用されている．また，データ形式

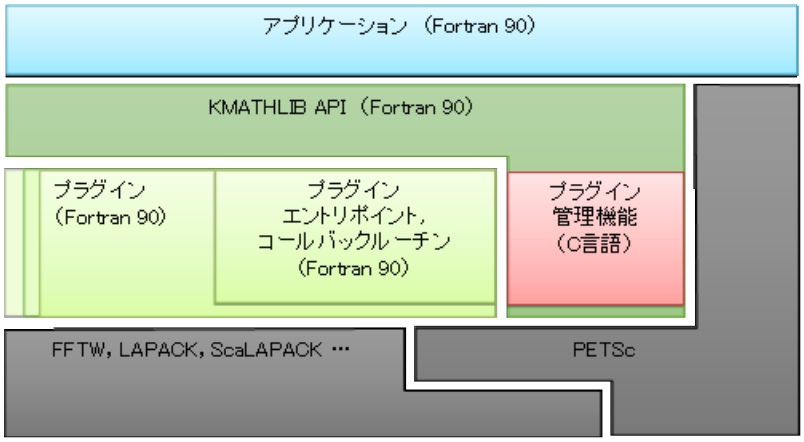

図 2.1: KMATHLIB API 機能のモジュール構成.

として PETSc のデータ型を用いていることに由来して，KMATHLIB API をビルドする際に参照した PETSc のスカラー型（倍精度実数や倍精度複素数等）により，標準形式のスカラー型が影響を受ける点については注意が必要である.

2.3　プラグイン

第 3 章で示される通り，配布される KMATHLIB API パッケージにはいくつかプラグイン（標準プラグイン）が付属する．しかしながら，プラグインは自由に追加可能であり，標準プラグインとユーザが独自に追加するプラグインを区別する仕組みは存在せず，これらはまったく同じように扱われる．また，ユーザが組み込むプラグインの解法の種別（連立一次方程式解法，固有値解法等）については制限はなく，内部形式と標準形式の変換をプラグインに記述することができれば，どのような解法であってもプラグインとして組み込むことができる.

第3章　KMATHLIB APIのビルドとインストール

本章ではKMATHLIB APIのパッケージの概要および，そのビルドとインストール手順について述べる．

3.1　KMATHLIB APIパッケージの概要

KMATHLIB APIパッケージには，プラグイン管理機能を含むKMATHLIB API（コア部）のソースコード，標準プラグインおよびテスト・プログラムのソースコードおよびそれらをビルドするための`Makefile`等が含まれている．パッケージのディレクトリ構造は図3.1に示す通りである．

必要要件

KMATHLIBのコア部のビルドには，

- C/C++コンパイラ
- Fortran 90/95 コンパイラ
- MPI
- PETSc

が必要である．現在までに，表3.1に示すソフトウェアを使用してビルドできることが確認されている．

表 3.1: 使用可能であることが確認されているソフトウェア（コア部）．

コンパイラ	富士通コンパイラ mpifrtpx，mpifccpx，mpiFCCpx （京コンピュータ, FX10，FX100用のクロスコンパイラ）， Intel コンパイラ（ifort，icc 13.0.1）
MPI	富士通 MPI, OpenMPI 1.6.5
PETSc	PETSc 3.6.3

表 3.2: プラグインが追加で必要とするソフトウェアおよびプラグインのスカラー型依存性.

プラグイン	追加で必要なソフトウェア	スカラー型依存性
template	なし	なし
dls_ssl2	SSL II[1, 2]	実数型である場合のみ動作
dls_lapack	LAPACK[3]	なし
dls_scalapack	ScaLAPACK	なし
dlsm_scalapack	ScaLAPACK	なし
deig_ssl2	SSL II	実数型である場合のみ動作
deig_lapack	LAPACK	なし
deig_scalapack	ScaLAPACK	なし
deig_eigenea	EigenExa	実数型である場合のみ動作
dsvd_scalapack	ScaLAPACK	なし
sls_petsc	なし	なし
fft_fftw	FFTW	複素数型である場合のみ動作
fft_kmath_fft3d	FFTE[9]（ソースコード）	複素数型である場合のみ動作
rnd_kmath_random	KMATH RANDOM[11]	なし

プラグインのビルドには追加のソフトウェアが必要である．一部のプラグインは PETSc のスカラー型を実数とした場合，または複素数とした場合のいずれかの場合のみに動作可能である．各プラグインの追加で必要となるソフトウェアおよびスカラー型依存性について表 3.2 に示す.

プラグインは個別にビルド，インストールすることが可能であり，ユーザは使用したいプラグインで必要となるソフトウェアのみを用意して使用することができる．現在までに，表 3.3 に示すソフトウェアを使用してビルドできることが確認されている.

3.2 ビルド手順

3.2.1 環境変数の設定

KMATHLIB API のコア部およびプラグインのビルド時に必要となる環境変数について述べる．なお，本副節では実行シェルが bash，zsh であることを想定して `export` コマンドによる設定例を示す．csh，tcsh 使用時は適宜コマンドを置換して実行されたい.

`PETSC_INC`，`PETSC_LIB`

PETSc のライブラリ・パスおよびインクルード・パスを `PETSC_INC`，`PETSC_LIB` として以下のように設定する.

表 3.3: 使用可能であることが確認されているソフトウェア（標準プラグイン）.

SSL II	富士通 SSL II（京コンピュータ，FX10，FX100 で提供されるもの）
LAPACK	富士通 SSL II（京コンピュータ，FX10，FX100 で提供されるもの）， Intel Math Kernel Library 11.0.1.117
ScaLAPACK	ScaLAPACK（京コンピュータ，FX10，FX100 で提供されるもの）， Intel Math Kernel Library 11.0.1.117
EigenExa	EigenExa 2.3c
FFTW	FFTW 3.3.3
FFTE	FFTE 6.0.0
KMATH RANDOM	KMATH RANDOM 1.1

PETSc 環境変数の設定例

```
$ export PETSC_INC\
  ="$HOME"/opt/petsc-3.6.3_complex/arch-fujitsu-sparc64fx-opt-z/include
$ export PETSC_LIB\
  ="$HOME"/opt/petsc-3.6.3_complex/arch-fujitsu-sparc64fx-opt-z/lib
```

環境変数 PETSC_INC はコア部のビルドおよび template を除くすべてのプラグインのビルド時に参照される．また，環境変数 PETSC_LIB は KMATHLIB API を用いたテスト・プログラムのビルド時に参照される．本環境変数は使用するスカラー型に応じた PETSc パスを設定しなければならない.

EIGENEXA_INC，EIGENEXA_LIB

EigenExa のライブラリ・パスおよびインクルード・パスを EIGENEXA_INC，EIGENEXA_LIB として以下のように設定する.

EigenExa 環境変数の設定例

```
$ export EIGENEXA_INC="$HOME"/opt/EigenExa-2.3c/include
$ export EIGENEXA_LIB="$HOME"/opt/EigenExa-2.3c/lib
```

環境変数 EIGENEXA_INC は deig_eigenexa プラグインのビルド時に参照される．また，環境変数 EIGENEXA_LIB は deig_eigenexa プラグインのテスト・プログラム（test/04）のビルド時に参照される.

FFTW_INC，FFTW_LIB

FFTW のライブラリ・パスおよびインクルード・パスを FFTW_INC，FFTW_LIB として以下のように設定する.

FFTW 環境変数の設定例

```
$ export FFTW_INC=/home/apps/fftw/3.3.3/include
$ export FFTW_LIB=/home/apps/fftw/3.3.3/lib64
```

環境変数 FFTW_INC は fft_fftw プラグインのビルド時に参照される．また，環境変数 FFTW_LIB は，fft_fftw プラグインのテスト･プログラム（test/09，test/10）および，fft_fftw と fft_kmath_fft3d プラグインのテスト・プログラム（test/11）のビルド時に参照される．

FFTE_SRC

FFTE のソースコードが置かれたディレクトリ・パスを FFTE_SRC として以下のように設定する．

FFTE 環境変数の設定例

```
$ export FFTE_SRC="$HOME"/opt/ffte-6.0
```

本環境変数は，fft_kmath_fft3d プラグインのビルド時に使用される．

KMATH_RANDOM_INC，KMATH_RANDOM_LIB

KMATH RANDOM のライブラリ･パスおよびインクルード･パスを KMATH_RANDOM_INC，KMATH_RANDOM_LIB として以下のように設定する．

KMATH RANDOM 環境変数の設定例

```
$ export KMATH_RANDOM_INC="$HOME"/opt/KMATH_RANDOM-1.1/random/f90
$ export KMATH_RANDOM_LIB="$HOME"/opt/KMATH_RANDOM-1.1/random/f90
```

環境変数 KMATH_RANDOM_INC は rnd_kmath_random プラグインのビルド時に参照される．また，環境変数 KMATH_RANDOM_LIB は rnd_kmath_random プラグインのテスト・プログラム（test/12）のビルド時に参照される．

3.2.2 KMATHLIB API のビルドとインストール

本副節では，京コンピュータ，富士通 PRIMEHPC FX10 および x86 クラスタにおける KMATHLIB API のビルドとインストール手順について説明する．なお，これら以外の計算機環境であっても若干の修正を行うことで同様のビルドが可能である．

KMATHLIB API パッケージのあるディレクトリに移動し，次のコマンドによりパッケージを展開し，展開されたディレクトリに移動する．

KMATHLIB API パッケージの展開

```
$ tar zxf KMATHLIB_API-1.0.tgz
$ cd KMATHLIB_API-1.0
```

x86 クラスタでビルドする場合，以下のコマンドにより計算機情報の設定ファイルをリンク，編集する．なお，京コンピュータおよび富士通 PRIMEHPC FX10 では以下の設定は不要である．

計算機情報ファイルの設定

```
$ cd src/
$ rm Makefile.machine
$ ln -s arch/Makefile.machine.Intel Makefile.machine
$ vim Makefile.machine      （必要に応じて適宜設定を変更する）
$ cd ../
```

コア部のビルドおよびインストールを以下のコマンドにより行う．

コア部のビルドおよびインストール

```
$ cd src/core
$ make install
$ cd ../..
```

本コマンドにより自動的にコア部のモジュール中間ファイル（*.mod ファイル）およびライブラリファイル（*.a ファイル）が，それぞれ include/，lib/ディレクトリにインストールされる．

続いて，標準プラグインのビルド・インストール方法について述べる．インストールしたいプラグインが仮に dls_scalapack である場合，以下のコマンドによりインストールを行う．

プラグインのビルドおよびインストール

```
$ cd src/plugins/dls_scalapack
$ make install
$ cd ../../..
```

本コマンドにより dls_scalapack プラグインのモジュール中間ファイルおよびライブラリファイルが，それぞれ include/，lib/ディレクトリにインストールされる．この手順をインストールしたい標準プラグインについて繰り返し行う．なお，すべてのプラグインをインストールする場合は，以下のコマンドにより自動的に全標準プラグインのビルドが可能である．

全プラグインのビルドおよびインストール

```
$ cd src/plugins
$ make install
$ cd ../..
```

3.2.3 テストプログラムのビルドと実行

本パッケージに付属するテストプログラムのビルドおよび実行方法について述べる．dls_scalapack プラグインのテスト・プログラム（test/06）の場合，以下の手順により，プログラムのビルドおよび実行が可能である．

テストプログラム test/06 のビルド例

```
$ cd test/06
$ make
（実行ファイル test が作成される）
$ vim run.sh
（ジョブスクリプト・ファイルを作成する）
$ pjsub run.sh
（ジョブが実行され run.sh.o[ジョブ番号] という結果が出力される）
$ cd ../..
```

参考までに，京コンピュータ向けのジョブ・スクリプトファイルの作成例を次に示す.

ジョブ・スクリプトファイル（run.sh）の作成例（京コンピュータ）

```
#!/bin/bash -x
#
#PJM --rsc-list "node=6"
#PJM --rsc-list "elapse=00:05:00"
#PJM --stg-transfiles all
#PJM --stgin "./test ./"
#PJM -s
#
. /work/system/Env_base

mpiexec -n 6 ./test
```

他のテストプログラムについても同様の手段でビルドおよび実行が可能である.

```
KMATHLIB_API_v1.0
|-- include                    モジュール中間ファイル（*.mod）がインストールされる.
|-- lib                        ライブラリ（*.a）がインストールされる.
|-- src
|   |-- arch                   ビルド時に必要な計算機依存情報ファイルが格納されている.
|   |-- core                   コア部が格納されている.
|   `-- plugins                標準プラグインが格納されている.
|       |-- deig_eigenexa          実対称標準固有値問題を EigenExa により解くプラグイン
|       |-- deig_lapack            標準固有値問題を LAPACK により解くプラグイン
|       |-- deig_scalapack         標準固有値問題を ScaLAPACK により解くプラグイン
|       |-- deig_ssl2              標準固有値問題を SSL II により解くプラグイン
|       |-- dsvd_scalapack         特異値分解を ScaLAPACK により行うプラグイン
|       |-- dls_lapack             連立一次方程式を LAPACK により解くプラグイン
|       |-- dls_scalapack          連立一次方程式を ScaLAPACK により解くプラグイン
|       |-- dls_ssl2               連立一次方程式を SSL II により解くプラグイン
|       |-- dlsm_scalapack         複数右辺項連立一次方程式を ScaLAPACK により解く
|       |                          プラグイン
|       |-- fft_fftw               FFT を FFTW により実行するプラグイン
|       |-- fft_kmath_fft3d        三次元 FFT を KMATH FFT3D により実行するプラグイン
|       |-- rnd_kmath_random       乱数生成を KMATH RANDOM により実行するプラグイン
|       |-- sls_petsc              疎行列連立一次方程式を PETSc により解くプラグイン
|       `-- template              プラグインテンプレート
`-- test                       標準プラグインのテストプログラムが格納されている.
    |-- 01                         template のテストプログラム
    |-- 02                         template のテストプログラム
    |-- 03                         sls_petsc のテストプログラム
    |-- 04                         deig_scalapack, deig_eigenexa のテストプログラム
    |-- 05                         deig_ssl2, deig_lapack のテストプログラム
    |-- 06                         dls_scalapack のテストプログラム
    |-- 07                         dls_lapack のテストプログラム
    |-- 08                         dls_ssl2 のテストプログラム
    |-- 09                         fft_fftw のテストプログラム
    |-- 10                         fft_fftw, fft_kmath_fft3d のテストプログラム
    |-- 11                         fft_fftw, fft_kmath_fft3d のテストプログラム
    |-- 12                         rnd_kmath_random のテストプログラム
    |-- 13                         dsvd_scalapack のテストプログラム
    `-- 14                         dlsm_scalapack のテストプログラム
```

図 3.1: KMATHLIB API パッケージのディレクトリ構造.

第4章　KMATHLIB APIを使用したソフトウェア開発

ScaLAPACK ルーチンを使用して連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解くテストプログラム（図 4.1）を例に，Fortran 90/95 環境における KMATHLIB API の使用方法について説明する．

モジュールの定義およびヘッダインクルード

KMATHLIB API を使用するソフトウェアでは最初に，use 文により KMATHLIB API のコア部の機能を使用するための kmath_lib_mod モジュール，プログラムで呼び出されるプラグインを使用するためのモジュール（本例では kmath_plugin_dls_scalapack_mod モジュール）を定義する．PETSc の機能を使用するために必要なヘッダファイルのインクルードを行う．本例では，行列およびベクトルの生成を行うため，finclude/petscsys.h，finclude/petscvec.h，finclude/petscmat.h をインクルードしている．

変数の宣言および変数への値の設定

変数の宣言部では，標準形式（すなわち PETSc のデータ型）を使用してソルバに渡すデータを格納する変数を宣言する．本例では，係数行列 A を Mat :: A，右辺ベクトル $\boldsymbol{b}$ および解ベクトル $\boldsymbol{x}$ をそれぞれ Vec :: B，Vec :: X として宣言している．また，KMATHLIB API のコンテキストハンドルを格納する変数（type(s_context) :: h）および，各ルーチンが返すエラー情報を受け取るための変数（本例では PetscInt :: ierr）の宣言が必要である．

プログラム実行部では，宣言された変数に対する値の設定や，KMATHLIB API を通じたソルバの呼び出しなどを行う．変数に対する値の設定では，必要に応じて PETSc の機能を用いるなどしてソルバに渡すデータに値を設定する．なお，PETSc の機能を用いる場合には PETSc の初期化が必要である．

KMATHLIB APIの使用

KMATHLIB API によるソルバの呼び出しの手順について述べる．まず，KMATH_Create によって MPI コミュニケータを指定して，KMATHLIB API を初期化し，コンテキストハンドラを作成する．状態（パラメータ等の設定）はコンテキストハンドラによって管理されており，KMATH_Destroy によって破棄されるまで状態は保持される．続いて，KMATH_Set_Parameter(...) により，標準形式の変数をソルバの変数として登録する．本例では，まず係数行列の変数 Mat :: A を，ソルバ

の変数'A' として登録し，続けて右辺ベクトル，解ベクトルの変数についても同様に登録している．その後，KMATH_Solve(h, ierr) を呼び出すことにより連立一次方程式を解くソルバルーチンが自動的に呼び出される．なお，KMATHLIB API を使用するには，事前に MPI の初期化が必要である．ただし，PETSc の初期化は MPI の初期化を兼ねているため，PETSc の初期化が行われている場合には，MPI の追加の初期化は不要である．

標準形式と内部形式の変換は自動的に行われており，KMATHLIB API 機能の利用者は直接に記述を行う必要はない．しかしながら，KMATH_Solve(h, ierr) を呼び出した後も，KMATH_Destroy(h, ierr) または KMATH_Flush(h, ierr) が呼び出されるまでは，計算結果は登録された変数（本例では X）には反映されていないので注意が必要である．出力結果を再利用して再び同じソルバを呼び出す場合（例えば追加で $A\boldsymbol{y} = \boldsymbol{x}$ を解く場合）には，内部形式で記憶された結果データの標準形式への変換は不要であるため，KMATHLIB API では明示的な指示がない限り登録された変数への結果の反映は行わない仕組みとなっている．

```
program test
  use kmath_lib_mod
  use kmath_plugin_dls_scalapack_mod
  implicit none
#include <finclude/petscsys.h>
#include <finclude/petscvec.h>
#include <finclude/petscmat.h>

  Mat :: A
  Vec :: B
  Vec :: X
  ...
  PetscInt             :: ierr
  ...
  type(s_context)      :: h

  call PetscInitialize(PETSC_NULL_CHARACTER, ierr)

  ...（PETSc 機能を用いた行列，右辺ベクトルの生成など）...

  ! Solve the linear system A X = B
  call KMATH_Create(h, PETSC_COMM_WORLD, ierr)

  call KMATH_Set_Plugin(h, KMATH_Plugin_Setup_DLS_ScaLAPACK, ierr)
  call KMATH_Set_Parameter(h, 'A', A, ierr)
  call KMATH_Set_Parameter(h, 'B', B, ierr)
  call KMATH_Set_Parameter(h, 'X', X, ierr)
  call KMATH_Solve(h, ierr)

  call KMATH_Destroy(h, ierr)

  ...（計算結果を参照した処理など）...

  call PetscFinalize(ierr)
end program test
```

図 4.1: `test/06/test.fpp` の主要部分.

第5章　インターフェースおよび標準プラグイン仕様解説

5.1　KMATHLIB API

KMATHLIB API の各インターフェースについて記載する．なお，いずれも集団処理（collective operation）であり，ハンドルに登録されている MPI コミュニケータに属するすべての MPI ランクで同時に呼び出さなければならない．

5.1.1　インターフェース KMATH_Create

MPI コミュニケータを指定して KMATHLIB API を初期化する．初期化に必要な処理を行い，そのコンテキストハンドルを返す．

```
use kmath_lib_mod
subroutine KMATH_Create(handle, comm, ierr)
```

引数	型	入出力	説明
handle	type(s_context)	入力	コンテキストハンドル．
comm	integer	入力	MPI コミュニケータ．
ierr	integer	出力	エラー値．ゼロであれば正常終了．

5.1.2　インターフェース KMATH_Destroy

KMATHLIB API を終了する．ハンドルに紐付けられたコンテキストが解放される．計算結果が内部形式で保留されている場合，この呼び出しによりユーザ側の標準形式の出力変数に内容が書き出される．

```
use kmath_lib_mod
subroutine KMATH_Destroy(handle, ierr)
```

引数	型	入出力	説明
handle	type(s_context)	入力	コンテキストハンドル．
ierr	integer	出力	エラー値．ゼロであれば正常終了．

5.1.3　インターフェース KMATH_Set_Plugin

プラグインエントリポイントを指定して，プラグイン登録を行う．標準プラグインのプラグインエントリポイント名は第 5.2 節を参照のこと．既にプラグインが設定されており，かつ計算結果が内部形式で保留されている場合，この呼び出しによりユーザ側の標準形式の出力変数に内容が書き出される．

```
use kmath_lib_mod
subroutine KMATH_Set_Plugin(handle, comm, ierr)
```

引数	型	入出力	説明
handle	type(s_context)	入力	コンテキストハンドル.
plugin_setup	interface	入力	プラグインエントリポイント.
ierr	integer	出力	エラー値．ゼロであれば正常終了.

5.1.4　インターフェース KMATH_Solve

計算を実行する．必要に応じてユーザ側の標準形式の入力変数が内部形式に変換され，プラグインによる実際の計算が実行される．この呼び出し終了時点では，計算結果は内部形式のまま保留された状態となる．解が求まらなかった場合にはエラー値として -1（E_SolveFail）を返す.

```
use kmath_lib_mod
subroutine KMATH_Solve(handle, ierr)
```

引数	型	入出力	説明
handle	type(s_context)	入力	コンテキストハンドル.
ierr	integer	出力	エラー値．ゼロであれば正常終了.

5.1.5　インターフェース KMATH_Flush

計算結果が内部形式で保留されている場合，この呼び出しによりユーザ側の標準形式の出力変数に内容が書き出される．

```
use kmath_lib_mod
subroutine KMATH_Flush(handle, ierr)
```

引数	型	入出力	説明
handle	type(s_context)	入力	コンテキストハンドル.
ierr	integer	出力	エラー値．ゼロであれば正常終了.

5.1.6 インターフェース KMATH_Set_Parameter

指定したパラメータの値を設定する．入出力データの変数もこのインターフェースを介して KMATHLIB API に対して与える．

```
use kmath_lib_mod
subroutine KMATH_Set_Parameter(handle, key, val, ierr)
```

引数	型	入出力	説明
handle	type(s_context)	入力	コンテキストハンドル．
key	character(*)	入力	パラメータ名．
val	integer	入力	設定値．
	character(*)		
	double precision		
	PetscFortranAddr		
ierr	integer	出力	エラー値．ゼロであれば正常終了．

5.1.7 インターフェース KMATH_Set_Parameters

指定したパラメータの値を設定する．入出力データの変数もこのインターフェースを介して KMATHLIB API に対して与える．

```
use kmath_lib_mod
subroutine KMATH_Set_Parameters(handle, key, val, ierr)
```

引数	型	入出力	説明
handle	type(s_context)	入力	コンテキストハンドル．
key	character(*)(:)	入力	パラメータ名．
val	integer(:)	入力	設定値．
	character(*)(:)		
	double precision(:)		
	PetscFortranAddr(:)		
ierr	integer	出力	エラー値．ゼロであれば正常終了．

5.1.8 構造体 s_context

KMATHLIB API コンテキストの構造体である．

```
use kmath_lib_mod
type(s_context) :: context
```

メンバ	型	説明
c	integer(8)	コンテキストの実体．C 言語によりアロケートされたメモリ領域へのポインタ値が格納される．

5.2　標準プラグイン

標準プラグインの仕様について記述する.

5.2.1　プラグイン dls_ssl2

密行列連立一次方程式を SSL II により解く.

プラグインモジュール名　`use kmath_plugin_dls_ssl2_mod`
エントリポイント名　`KMATH_Plugin_Setup_DLS_SSL2`

パラメータ	型	デフォルト値	入出力	説明
A	Mat	なし	入力	係数行列.
B	Vec	なし	入力	右辺ベクトル.
X	Vec	なし	出力	解ベクトル.
matrix_type	character(*)	"general"	入力	係数行列の行列タイプ. 一般行列の場合 "general", 実対称行列の場合 "symmetric" を指定.

5.2.2　プラグイン dls_lapack

密行列連立一次方程式を LAPACK により解く.

プラグインモジュール名　`use kmath_plugin_dls_lapack_mod`
エントリポイント名　`KMATH_Plugin_Setup_DLS_LAPACK`

パラメータ	型	デフォルト値	入出力	説明
A	Mat	なし	入力	係数行列.
B	Vec	なし	入力	右辺ベクトル.
X	Vec	なし	出力	解ベクトル.
matrix_type	character(*)	"general"	入力	係数行列の行列タイプ. 一般行列の場合 "general", 実対称行列の場合 "symmetric" を指定.

5.2.3　プラグイン dls_scalapack

密行列連立一次方程式を ScaLAPACK により解く.

プラグインモジュール名　`use kmath_plugin_dls_scalapack_mod`
エントリポイント名　`KMATH_Plugin_Setup_DLS_ScaLAPACK`

パラメータ	型	デフォルト値	入出力	説明
A	Mat	なし	入力	係数行列.
B	Vec	なし	入力	右辺ベクトル.
X	Vec	なし	出力	解ベクトル.
matrix_type	character(*)	"general"	入力	係数行列の行列タイプ. 一般行列の場合 "general", 実対称行列の場合 "symmetric" を指定.

5.2.4 プラグイン dlsm_scalapack

複数右辺項密行列連立一次方程式を ScaLAPACK により解く.

プラグインモジュール名 use kmath_plugin_dlsm_scalapack_mod
エントリポイント名 KMATH_Plugin_Setup_DLSM_ScaLAPACK

パラメータ	型	デフォルト値	入出力	説明
A	Mat	なし	入力	係数行列.
B	Vec	なし	入力	各列に右辺ベクトルを並べた行列.
X	Vec	なし	出力	各列に解ベクトルを並べた行列.
matrix_type	character(*)	"general"	入力	係数行列の行列タイプ. 一般行列の場合 "general", 実対称行列の場合 "symmetric" を指定.

5.2.5 プラグイン deig_ssl2

実対称密行列の固有値問題を SSL II により解く. スカラー型が倍精度実数の場合のみ動作する.

プラグインモジュール名 use kmath_plugin_deig_ssl2_mod
エントリポイント名 KMATH_Plugin_Setup_DEIG_SSL2

パラメータ	型	デフォルト値	入出力	説明
A	Mat	なし	入力	固有対の計算を行う行列.
X	Vec	なし	出力	固有値を並べたベクトル.
Z	Mat	なし	出力	各列に固有ベクトルを並べた行列.

5.2.6　プラグイン deig_lapack

行列の固有値問題を LAPACK により解く．スカラー型を倍精度実数としてビルドした場合は実対称行列，倍精度複素数としてビルドした場合はエルミート行列として扱われる．

プラグインモジュール名　`use kmath_plugin_deig_lapack_mod`
エントリポイント名　`KMATH_Plugin_Setup_DEIG_LAPACK`

パラメータ	型	デフォルト値	入出力	説明
A	Mat	なし	入力	固有対の計算を行う行列.
X	Vec	なし	出力	固有値を並べたベクトル.
Z	Mat	なし	出力	各列に固有ベクトルを並べた行列.

5.2.7　プラグイン deig_scalapack

行列の固有値問題を ScaLAPACK により解く．スカラー型を倍精度実数としてビルドした場合は実対称行列，倍精度複素数としてビルドした場合はエルミート行列として扱われる．

プラグインモジュール名　`use kmath_plugin_deig_scalapack_mod`
エントリポイント名　`KMATH_Plugin_Setup_DEIG_ScaLAPACK`

パラメータ	型	デフォルト値	入出力	説明
A	Mat	なし	入力	固有対の計算を行う行列.
X	Vec	なし	出力	固有値を並べたベクトル.
Z	Mat	なし	出力	各列に固有ベクトルを並べた行列.

5.2.8　プラグイン deig_eigenexa

実対称密行列の固有値問題を EigenExa により解く．スカラー型が倍精度実数の場合のみ動作する．

プラグインモジュール名　`use kmath_plugin_deig_eigenexa_mod`
エントリポイント名　`KMATH_Plugin_Setup_DEIG_EigenExa`

パラメータ	型	デフォルト値	入出力	説明
A	Mat	なし	入力	固有対の計算を行う行列.
X	Vec	なし	出力	固有値を並べたベクトル.
Z	Mat	なし	出力	各列に固有ベクトルを並べた行列.

5.2.9 プラグイン dsvd_scalapack

密行列の特異値分解を ScaLAPACK により求める.

プラグインモジュール名 `use kmath_plugin_dsvd_scalapack_mod`
エントリポイント名 `KMATH_Plugin_Setup_DSVD_ScaLAPACK`

パラメータ	型	デフォルト値	入出力	説明
A	Mat	なし	入力	特異値分解を求める行列.
S	Vec	なし	出力	特異値を並べたベクトル.
U	Mat	なし	出力	各列に左特異ベクトルを並べた行列.
VT	Mat	なし	出力	各列に右特異ベクトルを並べた行列の転置行列.

5.2.10 プラグイン sls_petsc

疎行列連立一次方程式を PETSc により解く.

プラグインモジュール名 `use kmath_plugin_sls_petsc_mod`
エントリポイント名 `KMATH_Plugin_Setup_SLS_PETSc`

パラメータ	型	デフォルト値	入出力	説明
A	Mat	なし	入力	係数行列.
B	Vec	なし	入力	右辺ベクトル.
X	Vec	なし	出力	解ベクトル.
matrix_type	character(*)	"general"	入力	係数行列の行列タイプ. 一般行列の場合 "general", 実対称行列の場合 "symmetric" を指定.

5.2.11 プラグイン fft_fftw

離散フーリエ変換を FFTW により行う.

プラグインモジュール名 `use kmath_plugin_fft_fftw_mod`
エントリポイント名 `KMATH_Plugin_Setup_FFT_FFTW`

パラメータ	型	デフォルト値	入出力	説明
dimension	integer	1	入力	（後述）
direction	charactr(*)	"forward"	入力	フーリエ変換の方向. 順変換の場合 "forward", 逆変換の場合 "backward" を指定する.
size_x	integer	0	入力	（後述）
size_y	integer	0	入力	（後述）
size_z	integer	0	入力	（後述）
A	Mat	なし	入力	入力データとなる行列.
X	Mat	なし	出力	出力データとなる行列.

パラメータ dimension

フーリエ変換対象データの次元数を 1（1 次元），2（2 次元），3（3 次元）の範囲で指定する．入出力データには Mat 型が利用されるが，指定する次元数と行列への値の格納方法の関係は以下の通りである．

1 次元 (n)	n 行 1 列の行列を作成.
2 次元 (n, m)	n 行 m 列の行列を作成.
3 次元 (n, m, o)	$n \times m$ 行 o 列の行列を作成.

パラメータ size_x，size_y，size_z

フーリエ変換対象のデータのサイズを指定する．1 次元の場合には size_x のみ，2 次元の場合には size_x および size_y，3 次元の場合にはすべてを指定する必要がある．

5.2.12　プラグイン fft_kmath_fft3d

3 次元離散フーリエ変換を KMATH FFT3D により行う．

プラグインモジュール名	use kmath_plugin_fft_kmath_fft3d_mod
エントリポイント名	KMATH_Plugin_Setup_FFT_KMATH_FFT3D

パラメータ	型	デフォルト値	入出力	説明
direction	charactr(*)	"forward"	入力	フーリエ変換の方向. 順変換の場合 "forward", 逆変換の場合 "backward" を指定する.
size_x	integer	0	入力	プラグイン fft_fftw を参照のこと.
size_y	integer	0	入力	プラグイン fft_fftw を参照のこと.
size_z	integer	0	入力	プラグイン fft_fftw を参照のこと.
A	Mat	なし	入力	入力データとなる行列.
X	Mat	なし	出力	出力データとなる行列.

5.2.13 プラグイン rnd_kmath_random

[1, 2) に一様分布する擬似乱数生成を KAMTH Random により行う.

プラグインモジュール名 use kmath_plugin_fft_rnd_kmath_random_mod
エントリポイント名 KMATH_Plugin_Setup_Rnd_KMATH_Random

パラメータ	型	デフォルト値	入出力	説明
X	Vec	なし	出力	生成された擬似乱数列.

第6章　KMATHLIB APIの現在と今後

現在のKMATHLIB APIには，以下に示す制約や困難，あるいは追加の検討される機能がある．今後のバージョンアップでは，特にこれらの点について改善，機能追加がなされる可能性がある．

1. 今日の計算科学ソフトウェアでは非常に多くの数値計算ライブラリ（ソルバ）が使用されているが，現在のKMATHLIB APIでは一部の機能以外は標準プラグインとして実装されていない．KMATHLIB APIの使用者に更なる利便を与えるべく，標準プラグインに新たな機能を追加し，より多くのソルバをカバーする必要がある．

2. 現在のKMATHLIB APIでは，同じプラグインが連続して使用され，かつ直前の計算の出力変数を次の計算の入力変数として使用する場合にのみ，内部形式に変換された変数の再利用が行われる．しかしながら，入力変数が非破壊のソルバにより計算を行う場合や，内部形式に変換された入力変数のコピーを別に保持する場合，入力変数についても，後続の計算で内部形式データの再利用可能性がある．また，異なるプラグインに切り替える場合であっても同じ内部形式が利用される場合には内部形式データの再利用可能性があるが，現在のKMATHLIB API機能では一旦標準形式に書き戻されてしまう．これらの再利用可能性を判断する機構が作成されれば，データ形式の変換に伴うオーバヘッドが削減され，KMATHLIB APIを使用したソフトウェアの性能向上につながると考えられる．

3. KMATHLIB APIを使用したソルバの呼び出しでは，プラグイン内で内部形式データの記憶領域が確保される仕組みとなっている．しかしながら，現在のKMATHLIB API機能群には確保される記憶領域のサイズを知る手段がなく，また，プラグイン内部で必要メモリが確保できなかった場合のエラー処理も不完全である．確保される記憶領域サイズの問い合わせ機能や，適切なエラー処理の実装は急務である．

4. 現在の標準プラグインは，プラグインごとに使用するライブラリが固定されている．このため，KMATHLIB API利用者自身がどのライブラリを使用するか判断せねばならず，数値計算ライブラリの詳しい知識をもたない利用者には困難が生じる可能性がある．このような問題の解決手段として，問題クラス（密行列連立一次方程式，密行列固有値問題等）ごとに，設定されたパラメータ（行列の対称性や次数など）や計算機環境に応じて適切なライブラリ（SSL II，LAPACK，ScaLAPACK等）を自動的に選択するプラグインを作成することが考えられる．

謝辞

KMATHLIB API の開発は理化学研究所のスーパーコンピュータ「京」を利用して行われている（「京」調整高度化枠利用：課題場号 ra000005（平成 25 年度–））．理化学研究所計算科学研究機構各位の支援に対して感謝する．

富士通 PRIMEHPC FX10 における KMATHLIB API の動作確認のための環境構築に際して，東京大学 Oakleaf-FX 利用支援ポータル掲載の「PETSc 構築手順」を参考にした．本手順書作成に携わった関係者に感謝の意を表す．

関連図書

[1] 富士通 SSL II 使用手引書，2014.

[2] Fujitsu SSL II/MPI 使用手引書，2014.

[3] LAPACK — Linear Algebra PACKage，http://www.netlib.org/lapack/（2016-02-15 確認）.

[4] ScaLAPACK — Scalable Linear Algebra PACKage，http://www.netlib.org/scalapack/（2016-02-15 確認）.

[5] S. Balay et al.，PETSc Web page，http://www.mcs.anl.gov/petsc（2016-02-15 確認）.

[6] S. Balay et al., “PETSc Users Manual”, 2015.

[7] S. Balay et al., “Efficient Management of Parallelism in Object Oriented Numerical Software Libraries”，Modern Software Tools in Scientific Computing, pp. 163–202, Birkhäuser Press, 1997.

[8] F. Matteo and J. G. Steven, “The Design and Implementation of FFTW3”, Proceedings of the IEEE, Vol. 93, No. 2, pp. 216–331, 2005.

[9] D. Takahashi，FFTE: A Fast Fourier Transform Package，http://www.ffte.jp/（2016-02-15 確認）.

[10] 高性能固有値ソルバ EigenExa，http://www.aics.riken.jp/labs/lpnctrt/EigenExa.html（2016-02-15 確認）.

[11] 高性能擬似乱数生成ルーチン KMATH_RANDOM，http://www.aics.riken.jp/labs/lpnctrt/KMATH_RANDOM.html（2016-02-15 確認）.

付 録A　プラグイン開発チュートリアル

第 2.3 節で述べた通り，KMATHLIB API ではユーザが自由にプラグインを追加することができる．

プラグイン管理機能は，計算処理や標準形式のデータを内部形式に変換する処理，内部形式を標準形式に戻す処理などを担うコールバックルーチン（図 A）を適切なタイミングで呼び出す．プラグインの開発者は，これらのコールバックルーチンを適切に実装し，コールバックルーチンを KMATHLIB API 側に登録するためのプラグインエントリポイントを作成しなければならない．

A.1　プラグインエントリポイントとコールバックルーチン

付属プラグイン `template` を例に，プラグイン開発の方法を解説する．

プラグインエントリポイント

プラグイン開発者はプラグインのエントリポイントとなるサブルーチンを以下の仕様で開発する必要がある．ユーザコードでは，このサブルーチンを KMATHLIB API のプラグイン登録インターフェース `KMATH_Set_plugin()` に渡すことで，そのプラグインが利用可能な状態になる．なお，エントリポイント名は任意であるが，プログラム内でユニークでなければならない．同一のエントリポイント名が存在する場合，リンク時にシンボル重複エラーが発生する．

プラグイン `template` では，図 A.2 に示されるように，エントリポイント名を `KMATH_Plugin_Setup_Template` としている．

`subroutine` *plugin_entry_point_name*`(context)`
インターフェース `KMATH_Set_plugin()` 呼び出し時に，プラグイン管理機能から呼び出される．その際 KMATHLIB API 初期化時に作成されたコンテキストデータ（構造体 `s_context` のメンバ `c` と同等，第 5.1.8 節を参照）が渡される．

引数	型	入出力	説明
`context`	`integer(8)`	入力	コンテキストデータ.

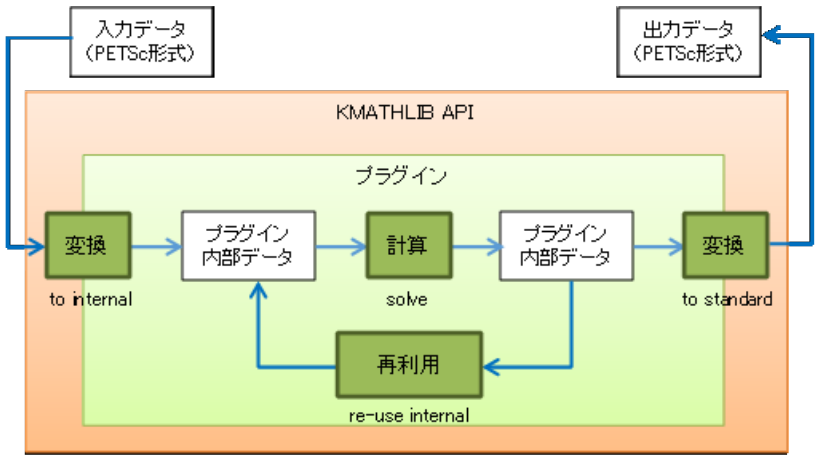

図 A.1: データ変換の流れとプラグイン内のコールバックルーチン.

プラグイン登録ルーチン kml_regist_plugin_routines

プラグインエントリポイントのサブルーチン内で，本ルーチンを一度だけ呼び出し，プラグインコールバックルーチンの登録を行う．プラグイン template における使用例は図 A.2 を参照されたい．

宣言 src/core/kml_pluin_if.h

```
subroutine kml_regist_plugin_routines(context,
                                      pcontext,
                                      in_parameters,
                                      out_parameters,
                                      mem_parameters,
                                      func_initialize,
                                      func_shutdown,
                                      func_realloc,
                                      func_solve,
                                      func_to_internal,
                                      func_to_standard,
                                      func_reuse_internal)
```

```
...
subroutine KMATH_Plugin_Setup_Template(context)

  ! formal arguments
  integer(8),              intent(in)    :: context

  ! local variables
  type(s_pcontext), pointer :: pcontext


  write(6,*) "KMATH_Plugin_Setup_Template> called."

  allocate(pcontext)
  pcontext%self => pcontext

  call kml_regist_plugin_routines(context,           &
                                  pcontext,          &
                                  "A,B",             &
                                  "X",               &
                                  "P1,P2",           &
                                  kmp_initialize,    &
                                  kmp_shutdown,      &
                                  kmp_realloc,       &
                                  kmp_solve,         &
                                  kmp_to_internal,   &
                                  kmp_to_standard,   &
                                  kmp_reuse_internal)

  return

end subroutine KMATH_Plugin_Setup_Template
...
```

図 A.2: src/plugins/template/kmath_plugin_template.fpp のプラグインエントリポイント.

引数	型	入出力	説明
context	integer(8)	入力	コンテキストデータ.
pcontext	integer(8)	入力	プラグインコンテキストデータ.
in_parameters	character(*)	入力	計算の入力パラメータを指定. 複数存在する場合は, 名前をカンマで区切る. このパラメータの順序がそのまま, ”0 番目の入力パラメータ”, ”1 番目の入力パラメータ”，... となる.
out_parameters	character(*)	入力	計算の出力パラメータを指定. 複数存在する場合は, 名前をカンマで区切る. このパラメータの順序がそのまま, ”0 番目の出力パラメータ”, ”1 番目の出力パラメータ”，... となる.
mem_parameters	character(*)	入力	内部形式を記憶するメモリを アロケート/リアロケートする きっかけとするパラメータ名を 指定する．複数存在する場合は, 名前をカンマで区切る.
func_initialize	subroutine	入力	初期化コールバックルーチンを指定.
func_shutdown	subroutine	入力	シャットダウンコールバック ルーチンを指定.
func_realloc	subroutine	入力	リアロケーションコールバック ルーチンを指定.
func_solve	subroutine	入力	計算コールバックルーチンを指定.
func_to_internal	subroutine	入力	内部形式への変換コールバック ルーチンを指定.
func_to_standard	subroutine	入力	標準形式への変換コールバック ルーチンを指定.
func_reuse_internal	subroutine	入力	内部形式データ再利用コールバック ルーチンを指定.

プラグインコールバックルーチン

プラグインコールバックルーチンは KMATHLIB API の一連の処理の中でプラグイン管理機能から必要に応じて呼び出される．プラグイン開発者が実装する必要があるルーチンは以下の通りで

ある．ルーチン名は任意である．返すべきエラーについては src/core/kml_error_code.h に定義されている．

subroutine ***callback_initialize_name***(context, pctxt, error)

初期化コールバックルーチンである．プラグインの初期化処理を実装する．プラグイン template における実装例は図 A.3 を参照のこと．

引数	型	入出力	説明
context	integer(8)	入力	コンテキストデータ.
pctxt	<any type>	入出力	プラグインコンテキストデータ.
error	integer	出力	エラー情報.
			正常終了 EP_None
			異常終了 EP_FatalInit

subroutine ***callback_shutdown_name***(context, pctxt, error)

シャットダウンコールバックルーチンである．プラグインのシャットダウン処理を実装する．プラグイン template における実装例は図 A.4 を参照のこと．

引数	型	入出力	説明
context	integer(8)	入力	コンテキストデータ.
pctxt	<any type>	入出力	プラグインコンテキストデータ.
error	integer	出力	エラー情報.
			正常終了 EP_None
			異常終了 EP_FatalShutdown

subroutine ***callback_realloc_name***(context, pctxt, error)

リアロケートコールバックルーチンである．内部形式でデータを格納するメモリのリアロケート処理を実装する．プラグイン template における実装例は図 A.5 を参照のこと．

引数	型	入出力	説明
context	integer(8)	入力	コンテキストデータ.
pctxt	<any type>	入出力	プラグインコンテキストデータ.
error	integer	出力	エラー情報.
			正常終了 EP_None
			異常終了 EP_FatalRealloc

```
...
subroutine kmp_initialize(context, pcontext, error)

  ! formal arguments
  integer(8),              intent(in)    :: context
  type(s_pcontext),        intent(inout) :: pcontext
  integer,                 intent(out)   :: error

  ! local variables
  integer                  :: comm, rank, nproc, ierr


  call kml_get_mpi_communicator(context, comm)

  call MPI_Comm_rank(comm, rank, ierr)
  call MPI_Comm_size(comm, nproc, ierr)

  write(6,*) "Kmp_Initialize> called. Rank:", rank, " / ", nproc

  call kml_set_error(error, EP_None)
  return

end subroutine kmp_initialize
...
```

図 A.3: src/plugins/template/kmath_plugin_template.fpp の初期化コールバックルーチン.

```
...
subroutine kmp_shutdown(context, pcontext, error)

  ! formal arguments
  integer(8),              intent(in)    :: context
  type(s_pcontext),        intent(inout) :: pcontext
  integer,                 intent(out)   :: error


  write(6,*) "Kmp_Shutdown> called."

  deallocate(pcontext%self)

  call kml_set_error(error, EP_None)
  return

end subroutine kmp_shutdown
...
```

図 A.4: src/plugins/template/kmath_plugin_template.fpp のシャットダウンコールバックルーチン.

```
...
subroutine kmp_realloc(context, pcontext, error)

  ! formal arguments
  integer(8),              intent(in)    :: context
  type(s_pcontext),        intent(inout) :: pcontext
  integer,                 intent(out)   :: error


  write(6,*) "Kmp_Realloc> called. "

  call kml_set_error(error, EP_None)
  return

end subroutine kmp_realloc
...
```

図 A.5: src/plugins/template/kmath_plugin_template.fpp のリアロケートコールバックルーチン.

subroutine ***callback_solve_name***(context, pctxt, error)

計算コールバックルーチンである．計算処理を実装する．プラグイン template における実装例は図 A.6 を参照のこと.

引数	型	入出力	説明
context	integer(8)	入力	コンテキストデータ.
pctxt	<any type>	入出力	プラグインコンテキストデータ.
error	integer	出力	エラー情報.
			正常終了 EP_None
			異常終了 EP_FatalRealloc

```
...
subroutine kmp_solve(context, pcontext, error)

  ! formal arguments
  integer(8),              intent(in)    :: context
  type(s_pcontext),        intent(inout) :: pcontext
  integer,                 intent(out)   :: error


  write(6,*) "Kmp_Solve> called."

  call kml_set_error(error, EP_None)
  return

end subroutine kmp_solve
...
```

図 A.6: src/plugins/template/kmath_plugin_template.fpp の計算コールバックルーチン.

subroutine ***callback_to_internal_name***(context, pctxt, error, in_param, in_petsc)

データ形式を内部形式に変換するコールバックルーチンである．KMATHLIB API の入力データについて，標準形式から内部形式に変換する処理を実装する．プラグイン template における実装例は図 A.7 を参照のこと．

引数	型	入出力	説明
context	integer(8)	入力	コンテキストデータ.
pctxt	<any type>	入出力	プラグインコンテキストデータ.
error	integer	出力	エラー情報.
			正常終了 EP_None
			異常終了 EP_FatalToInternal
in_param	integer	入力	何番目の入力パラメータが
			内部形式に変換されるのかを表す
			ゼロ以上の番号.
in_petsc	PetscFortranAddr	入力	変換対象となる実際の標準形式データ.

subroutine ***callback_to_standard_name***(context, pctxt, error, out_param, out_petsc)

データ形式を標準形式に変換するコールバックルーチンである．KMATHLIB API の出力データについて，内部形式から標準形式に変換する処理を実装する．プラグイン template における実装例は図 A.8 を参照のこと．

引数	型	入出力	説明
context	integer(8)	入力	コンテキストデータ.
pctxt	<any type>	入出力	プラグインコンテキストデータ.
error	integer	出力	エラー情報.
			正常終了 EP_None
			異常終了 EP_FatalToStandard
out_param	integer	入力	何番目の出力パラメータが
			標準形式に変換されるのかを表す
			ゼロ以上の番号.
out_petsc	PetscFortranAddr	入力	変換対象となる実際の内部形式データ.

subroutine ***callback_to_reuse_name***(context, pctxt, error, out_param, in_param)

内部形式データの再利用コールバックルーチンである．内部形式データを再利用する処理を実装する．このルーチンは，次に示すようなユーザコードがあった場合に呼び出される．

```
...
subroutine kmp_to_internal(context, pcontext, error, in_param, in_petsc)

  ! formal arguments
  integer(8),              intent(in)    :: context
  type(s_pcontext),        intent(inout) :: pcontext
  integer,                 intent(out)   :: error
  integer,                 intent(in)    :: in_param
  PetscFortranAddr,        intent(in)    :: in_petsc


  write(6,*) "Kmp_To_Internal> called.  IN:", in_param, &
             " (", in_petsc, ")"

  call kml_set_error(error, EP_None)
  return

end subroutine kmp_to_internal
...
```

図 A.7: src/plugins/template/kmath_plugin_template.fpp のデータを内部形式に変換するコールバックルーチン.

```
...
subroutine kmp_to_standard(context, pcontext, error, out_param, out_petsc)

  ! formal arguments
  integer(8),              intent(in)    :: context
  type(s_pcontext),        intent(inout) :: pcontext
  integer,                 intent(out)   :: error
  integer,                 intent(in)    :: out_param
  PetscFortranAddr,        intent(in)    :: out_petsc


  write(6,*) "Kmp_To_Standard> called. OUT:", out_param, &
             " (", out_petsc, ")"

  call kml_set_error(error, EP_None)
  return

end subroutine kmp_to_standard
...
```

図 A.8: src/plugins/template/kmath_plugin_template.fpp のデータを標準形式に変換するコールバックルーチン.

```
call KMATH_Set_Parameter(h, 'InputParam', A, ierr)
call KMATH_Set_Parameter(h, 'OutputParam', X, ierr)
call KMATH_Solve(h, ierr)

call KMATH_Set_Parameter(h, 'InputParam', X, ierr)
call KMATH_Set_Parameter(h, 'OutputParam', Y, ierr)
call KMATH_Solve(h, ierr)
```

1 回目の call KMATH_Solve が完了した段階では，変数 X には'OutputParam' 内部形式からのデータ変換はなされていない．このコールバックルーチンによって，'OutputParam' の内部形式データから，'InputParam' の内部形式データへデータがコピーされる（開発者によるコピー処理の実装が必要である）．プラグイン template における実装例は図 A.9 を参照のこと．

引数	型	入出力	説明
context	integer(8)	入力	コンテキストデータ.
pctxt	<any type>	入出力	プラグインコンテキストデータ.
error	integer	出力	エラー情報.
			正常終了 EP_None
			異常終了 EP_FatalToReuse
out_param	integer	入力	何番目の出力パラメータが
			再利用元となるかを表すゼロ以上の番号.
in_param	integer	入力	何番目の入力パラメータを
			再利用先とするかを表すゼロ以上の番号.

```
...
subroutine kmp_reuse_internal &
           (context, pcontext, error, out_param, in_param)

  ! formal arguments
  integer(8),              intent(in)    :: context
  type(s_pcontext),        intent(inout) :: pcontext
  integer,                 intent(out)   :: error
  integer,                 intent(in)    :: out_param
  integer,                 intent(in)    :: in_param


  write(6,*) "Kmp_Reuse_Internal> called. OUT:", out_param, &
             " => IN:", in_param

  call kml_set_error(error, EP_None)
  return

end subroutine kmp_reuse_internal
...
```

図 A.9: src/plugins/template/kmath_plugin_template.fpp の内部形式データの再利用コールバックルーチン.

A.2 プラグインから利用可能なサブルーチン群解説

プラグインから利用可能なサブルーチンについて解説する.

サブルーチン kml_get_mpi_communicator

指定したコンテキストに紐付けられた MPI コミュニケータを返す.

宣言 src/core/kml_pluin_if.h

```
subroutine kml_get_mpi_communicator(handle, comm)
```

引数	型	入出力	説明
handle	integer(8)	入力	コンテキストハンドル.
comm	integer	出力	MPI コミュニケータ.

サブルーチン kml_get_parameter_int

指定した名前のパラメータ値を整数型として取得する.

宣言 src/core/kml_pluin_if.h

```
subroutine kml_get_parameter_int(handle, key, val, err)
```

引数	型	入出力	説明
handle	integer(8)	入力	コンテキストハンドル.
key	character(*)	入力	パラメータ名.
val	integer	出力	取得された整数型のパラメータ値.
err	integer	出力	エラー値.
			EI_None(= 0)
			正常終了.
			EI_ParamNotFound(= 200)
			指定した名前のパラメータが
			存在しない.
			EI_FatalGetParam(= 201)
			設定されている値の形と不一致.

サブルーチン kml_get_parameter_real

指定した名前のパラメータ値を実数型として取得する.

宣言 src/core/kml_pluin_if.h

```
subroutine kml_get_parameter_real(handle, key, val, err)
```

引数	型	入出力	説明
handle	integer(8)	入力	コンテキストハンドル.
key	character(*)	入力	パラメータ名.
val	double precision	出力	取得された実数型のパラメータ値.
err	integer	出力	エラー値.
			EI_None(= 0)
			正常終了.
			EI_ParamNotFound(= 200)
			指定した名前のパラメータが
			存在しない.
			EI_FatalGetParam(= 201)
			設定されている値の形と不一致.

サブルーチン kml_get_parameter_string

指定した名前のパラメータ値を文字列型として取得する.

宣言　src/core/kml_pluin_if.h

```
subroutine kml_get_parameter_string(handle, key, val, err)
```

引数	型	入出力	説明
handle	integer(8)	入力	コンテキストハンドル.
key	character(*)	入力	パラメータ名.
val	character(*)	出力	取得された文字列型のパラメータ値.
err	integer	出力	エラー値.
			EI_None(= 0)
			正常終了.
			EI_ParamNotFound(= 200)
			指定した名前のパラメータが
			存在しない.
			EI_FatalGetParam(= 201)
			設定されている値の形と不一致.

サブルーチン kml_get_parameter_petsc

指定した名前のパラメータ値を PetscFortranAddr 型として取得する.

宣言　src/core/kml_pluin_if.h

```
subroutine kml_get_parameter_petsc(handle, key, val, err)
```

引数	型	入出力	説明
handle	integer(8)	入力	コンテキストハンドル.
key	character(*)	入力	パラメータ名.
val	PetscFortranAddr	出力	取得された PetscFortranAddr 型のパラメータ値.
err	integer	出力	エラー値. EI_None(= 0) 正常終了. EI_ParamNotFound(= 200) 指定した名前のパラメータが 存在しない. EI_FatalGetParam(= 201) 設定されている値の形と不一致.

サブルーチン kml_get_row_col

指定したプロセス数から，プロセス行数およびプロセス列数を求める.

宣言　src/core/kml_pluin_if.h

```
subroutine kml_get_row_col(mprocs, nrow, ncol)
```

引数	型	入出力	説明
nprocs	integer	入力	プロセス数.
nrow	integer	出力	求められたプロセス行数.
ncol	integer	出力	求められたプロセス列数.

サブルーチン kml_get_prime_factor

指定された値を因数分解する.

宣言　src/core/kml_pluin_if.h

```
subroutine kml_get_prime_factor(val, factors)
```

引数	型	入出力	説明
val	integer	入力	素因数分解対象の値.
factors	integer(:)	出力	素因数の配列.

サブルーチン kml_tolower_case

指定した文字列を小文字に変換する.

宣言　src/core/kml_pluin_if.h

```
subroutine kml_tolower_case(str)
```

引数	型	入出力	説明
str	character(*)	入出力	変換対象の文字列.

サブルーチン kml_set_error

エラーコードを設定する．プリプロセッサマクロ DEBUG が有効な場合，エラーコードに対応するエラー文字列を標準入力に対して出力する.

宣言　src/core/kml_pluin_if.h

```
subroutine kml_set_error(error, code)
```

引数	型	入出力	説明
error	integer	入出力	エラーコード設定対象の変数.
code	integer	入力	エラーコード.

付 録B　インターフェース呼び出しの際に行われるデータ変換，再利用処理

インターフェース KMATH_Solve() が実行される際，出力パラメータに与えられた変数に計算結果が格納されたことを表すハッシュテーブルに登録される．次の KMATH_Solve() が実行される際，入力パラメータの変数をこのハッシュテーブルから検索し，見つかった場合には前回の出力結果の内部形式データをそのまま利用可能であるとみなし，再利用のためのコールバックルーチンが呼び出される．見つからなかった場合は，標準形式から内部形式への変換のコールバックルーチンが呼び出される．

各インターフェースを呼び出したときに発生する処理について次の表に示す．

インターフェース	処理順序	説明
KMATH_Create		なし．
KMATH_Destroy	1	内部形式から標準形式へ変換が発生．
	2	シャットダウン処理の実行．
KMATH_Set_Plugin	1	内部形式から標準形式へ変換が発生．
	2	プラグイン切り替え（新規登録）と初期化．
KMATH_Solve	1	標準形式から内部形式へ変換， または，再利用が発生．
	2	（前回の計算結果が再利用されない場合） 内部形式から標準形式への変換が発生．
	3	計算実行． 新たな出力結果（内部形式）が求まり， ハッシュテーブルに格納される．
KMATH_Flush	1	内部形式から標準形式への変換が発生．
KMATH_Set_Parameter	1	内部形式から標準形式への変換が発生．
	2	リアロケートが発生．