NAG Library, Mark 27.1
NLL6I271BL - Licence Managed
Linux, 64-bit, Intel C/C++ or Intel Fortran

ユーザーノート

内容

1. イントロダクション

本ユーザーノートは，NAG Library，Mark 27.1 - NLL6I271BL（ライブラリ）のご利用方法（リンク方法）を説明します.

本ユーザーノートには，NAG Library Manual，Mark 27.1（ライブラリマニュアル）には含まれない製品毎の情報が含まれています．ライブラリマニュアルに「ユーザーノート参照」などと書かれている場合は，本ユーザーノートをご参照ください.

ライブラリルーチンのご利用に際しては，ライブラリマニュアル（「5. ドキュメント」参照）の以下のドキュメントをお読みください.

(a) How to Use the NAG Library
(b) Chapter Introduction
(c) Routine Document

2. 追加情報

本ライブラリの動作環境やご利用方法についての最新の情報は，以下のウェブページをご確認ください.
https://www.nag.com/doc/inun/nl27/l6i1bl/supplementary.html

3. 一般情報

本製品では，Intel ® Math Kernel Library for Linux（MKL）が提供する BLAS／LAPACK ルーチンを利用するスタティックライブラリ libnag_mkl.a および共有ライブラリ libnag_mkl.so と，NAG が提供する BLAS／LAPACK ルーチンを利用するスタティックライブラリ libnag_nag.a および共有ライブラリ libnag_nag.so が提供されます．本ライブラリは，MKL version 2020.0.2 を用いてテストされています．MKL version 2019.0.3 は本製品の一部として提供されます．MKL の詳細につきましては，Intel 社のウェブサイト https://software.intel.com/content/www/us/en/develop/tools/math-kernel-library.html をご参照ください．パフォーマンスの面からは，MKL を利用するバージョンの NAG ライブラリ libnag_mkl.a または libnag_mkl.so のご利用を推奨します．本製品では，32-bit 整数（lp64 と表記）と 64-bit 整数（ilp64 と表記）の両バージョンのライブラリ（および関連ファイル）が提供されます．

本製品には，NAG AD ライブラリ libnag_nag_ad.a，libnag_nag_ad.so，libnag_mkl_ad.a，libnag_mkl_ad.so が含まれています．NAG AD ライブラリのルーチンを利用する場合は，dcof インターフェースレイヤーライブラリ libnag_dcof.a と共に，これらのライブラリを使う必要があります．

dcof インターフェースレイヤーライブラリ libnag_dcof.a は，２つのテープメモリモデル（blob と chunk）用に提供されます．メモリモデルの詳細については，NAG dco/c++ のユーザーガイドをご参照ください．

NAG ライブラリはメモリリークが起きないように設計されています．メモリの解放は NAG ライブラリ自身によってか，もしくは，C ルーチンに対しては，ユーザーが NAG_FREE() を呼び出すことによって行われます．しかしながら，NAG ライブラリが依存している他のライブラリ（コンパイラのランタイムライブラリなど）がメモリリークを起こすかもしれません．このため，NAG ライブラリをリンクしているプログラムに対して何らかのメモリトレースツールを使った際に，場合によってはメモリリークが検出されるかもしれません．リークするメモリの量はアプリケーションによって異なると思われますが，NAG ライブラリの呼び出し回数に比例して際限なく増加するものではありません．

NAG ライブラリをマルチスレッドアプリケーションで利用する場合の詳細は，ライブラリマニュアルの "CL Interface Multithreading" または "FL Interface Multithreading" ドキュメントをご参照ください．本製品で提供される Intel MKL ライブラリをマルチスレッドアプリケーションで利用する場合の詳細は，以下の Intel 社のウェブサイトをご参照ください．
https://software.intel.com/en-us/articles/intel-math-kernel-library-intel-mkl-using-intel-mkl-with-threaded-applications

NAG AD ライブラリはスレッドセーフではありませんので，NAG AD ライブラリのルーチンの呼び出しを並列で行うことはできません．

本製品で提供されているライブラリは並列化されていません．ただし，MKL ライブラリは OpenMP で並列化されています．スレッド数の設定につきましては「3.1.1 使用するスレッド数の設定」をご参照ください．

また，MKL には，条件付きビット単位の再現性（Bit-wise Reproducibility（BWR））オプションがあります．ユーザーコードが一定の条件を満たしていれば，（https://software.intel.com/en-us/mkl-linux-developer-guide-reproducibility-conditions 参照）環境変数 MKL_CBWR を設定することにより BWR が有効になります．詳細は MKL のドキュメントをご参照ください．ただし，多くの NAG ルーチンはこれらの条件を満たしていません．従って，MKL を利用するバージョンの NAG ライブラリの全ルーチンに対して，異なる CPU アーキテクチャに渡り MKL_CBWR による BWR を保証することはできません．BWR に関するより一般的な情報は，ライブラリマニュアルの "How to Use the NAG Library" ドキュメントの「8.1 Bit-wise Reproducibility (BWR)」をご参照ください．

本製品は，MKL 10.3 よりも古いバージョンの MKL とは互換性がありません．

3.1. ライブラリのリンク方法

本セクションでは，本製品が［INSTALL_DIR］にインストールされていることが前提となります．デフォルトの［INSTALL_DIR］は $HOME/NAG/nll6i271bl となります．また，インストール時に［INSTALL_DIR］を指定することもできます．

NAG ライブラリは，NAG C ライブラリと NAG Fortran ライブラリ（Fortran ライブラリインタフェースの C ラッパーを含む）の両方を含んでいます．また，本製品に含まれる様々なライブラリの呼び出しを支援するためのスクリプト nagvars.sh および nagvars.csh が提供されます．これらのスクリプトは，NAG ルーチンを呼び出すアプリケーションのコンパイル・リンクのための NAG 固有の環境変数を設定します．また，NAG の実行プログラムとライブラリがコンパイル・リンク・実行時に見つかるように，標準の環境変数 PATH と LD_LIBRARY_PATH も設定します．

nagvars スクリプトの使用法は以下のようになります．

```
. [INSTALL_DIR]/scripts/nagvars.sh [-help] [-unset] [-quiet] [-gnu] \
  {int32,int64} {vendor,nag} {static,dynamic}
```

または，

```
source [INSTALL_DIR]/scripts/nagvars.csh [-help] [-unset] [-quiet] [-gnu] \
       {int32,int64} {vendor,nag} {static,dynamic}
```

ここで，

- nagvars.sh は，Bourne，Bash または同等のシェル（注：Ubuntu などの一部の Debian 系の Linux ディストリビューションで利用可能な Dash ではなく）で使用する必要があります．また，nagvars.csh は，Csh，Tcsh または同等のシェルで使用する必要があります．
- {int32，int64} は，NAG ルーチンの整数の引数と変数のデフォルトサイズを指定します．
- {vendor，nag} は，本製品で提供される MKL ライブラリに依存する NAG ライブラリを使用するか（オプション vendor），もしくは，MKL ライブラリに依存しない NAG ライブラリを使用するか(オプション nag)を指定します．パフォーマンスの面からは，オプション vendor を推奨します．

- {static, dynamic} は，NAG ライブラリの静的バージョンと動的（共有）バージョンのどちらにリンクするかを指定します.

デフォルト値は設定されていないため，３つのオプション全てを指定する必要があります．どの順序で指定しても構いません.

任意で指定できるオプションは以下のとおりです.

- -help は，スクリプトについての情報を表示します.
- -unset は，NAG 固有の環境変数を削除し，標準の環境変数 PATH と LD_LIBRARY_PATH から本製品に関する全ての参照を削除します.
- -quiet は，標準出力への出力を抑止します.
- -gnu は，本セクションの最後で説明されます.

以下に，Bash での nagvars スクリプトのコマンド例を示します.

```
source [INSTALL_DIR]/scripts/nagvars.sh int64 vendor dynamic
```

NAG 固有の環境変数は以下のとおりです.

- NAGLIB_CC - NAG ライブラリの作成に使用された C コンパイラ
- NAGLIB_CXX - NAG ライブラリの作成に使用された C++ コンパイラ
- NAGLIB_F77 - NAG ライブラリの作成に使用された Fortran コンパイラ
- NAGLIB_CFLAGS - 必要または推奨される C コンパイラオプション
- NAGLIB_CXXFLAGS - 必要または推奨される C++ コンパイラオプション
- NAGLIB_FFLAGS - 必要または推奨される Fortran コンパイラオプション
- NAGLIB_INCLUDE - NAG C ヘッダーおよび Fortran モジュールファイルへのパス
- NAGLIB_CLINK - NAG（および，オプションでベンダーの BLAS と LAPACK）ルーチンのリンクに必要なライブラリ，C コンパイラ使用時
- NAGLIB_CXXLINK - NAG（および，オプションでベンダーの BLAS と LAPACK）ルーチンのリンクに必要なライブラリ，C++ コンパイラ使用時
- NAGLIB_FLINK - NAG（および，オプションでベンダーの BLAS と LAPACK）ルーチンのリンクに必要なライブラリ，Fortran コンパイラ使用時
- NAGLIB_AD_LINK - NAG AD ルーチンに必要な追加のライブラリ（NAGLIB_CLINK, NAGLIB_CXXLINK, NAGLIB_FLINK の前にリンクする必要があります.）

NAG ライブラリおよび（必要に応じて）本製品で提供される MKL ライブラリを利用する場合は，以下のようにコンパイル・リンクを行ってください.

C プログラム，NAG AD ライブラリを利用する場合：

```
${NAGLIB_CC} ${NAGLIB_CFLAGS} ${NAGLIB_INCLUDE} program.c ${NAGLIB_AD_LINK} \
${NAGLIB_CLINK}
```

C プログラム，NAG AD ライブラリを利用しない場合：

```
${NAGLIB_CC} ${NAGLIB_CFLAGS} ${NAGLIB_INCLUDE} program.c ${NAGLIB_CLINK}
```

C++ プログラム，NAG AD ライブラリを利用する場合：

```
${NAGLIB_CXX} ${NAGLIB_CXXFLAGS} ${NAGLIB_INCLUDE} program.cpp ${NAGLIB_AD_LINK} \
${NAGLIB_CXXLINK}
```

C++ プログラム，NAG AD ライブラリを利用しない場合：

```
${NAGLIB_CXX} ${NAGLIB_CXXFLAGS} ${NAGLIB_INCLUDE} program.cpp ${NAGLIB_CXXLINK}
```

Fortran プログラム，NAG AD ライブラリを利用する場合：

```
${NAGLIB_F77} ${NAGLIB_FFLAGS} ${NAGLIB_INCLUDE} program.f90 ${NAGLIB_AD_LINK} \
${NAGLIB_FLINK}
```

Fortran プログラム，NAG AD ライブラリを利用しない場合：

```
${NAGLIB_F77} ${NAGLIB_FFLAGS} ${NAGLIB_INCLUDE} program.f90 ${NAGLIB_FLINK}
```

場合に依っては（例えば，より新しいバージョンのコンパイラを使用する場合など），その他のパスを（例えば，コンパイラのランタイムライブラリのパスを）LD_LIBRARY_PATH に設定する必要があるかもしれません.

異なるコンパイラもしくは異なるバージョンの Intel コンパイラを使用している場合は，[INSTALL_DIR]/rtl/lib/intel64 ディレクトリに提供される Intel コンパイラのランタイムライブラリをリンクする必要があります. この場合，

[INSTALL_DIR]/rtl/lib/intel64

を LD_LIBRARY_PATH に追加してください.

NAG AD ルーチンをご利用になる場合，nagvars スクリプトは，dcof インターフェースレイヤーライブラリの blob バージョンに対してリンクを構成します. chunk バージョンへのアクセスに切り替えるには，nagvars スクリプト（または，NAGLIB_AD_LINK に設定した値）を編集し，blob/libnag_dcof を chunk/libnag_dcof に置き換えてください.

本 NAG ライブラリは（少なくとも「2. 追加情報」に記載されているバージョンの）GNU gcc および g++ コンパイラから呼び出すことができます. これを行う最も簡単な方法は，nagvars.sh または nagvars.csh スクリプトに -gnu オプションを使用することです. このオプションを使用すると，NAG 固有の環境変数が，GNU gcc および（必要に応じて）g++ を使用するように変更されます.

GNU gfortran はサポートされていないため，Fortran 関連の NAG 環境変数は，引き続き Intel ifort コンパイラを参照することに注意してください. また，MKL とのリンクでは NAG ライブラリに存在する OpenMP ルーチンとの一貫性を保つために，引き続き Intel コンパイラ OpenMP ランタイムが使用されることに注意してください.

3.1.1. 使用するスレッド数の設定

MKL は OpenMP を用いて並列化されています. 実行時に使用するスレッド数を環境変数 OMP_NUM_THREADS に設定してください.

C シェルの場合 :
setenv OMP_NUM_THREADS N

Bourne シェルの場合 :
OMP_NUM_THREADS=N
export OMP_NUM_THREADS

N はご利用のスレッド数です. 環境変数 OMP_NUM_THREADS はプログラムの実行毎に再設定することができます.

MKL のいくつかのルーチンは複数レベルの OpenMP 並列処理を持ちます. これらのルーチンはユーザーアプリケーションの OpenMP 並列領域内から呼び出すこともできます. デフォルトでは OpenMP ネスト並列処理は無効になっており, 最も外側の並列領域だけが N スレッドで実行されます. 内部レベルはアクティブにならず, 1 スレッドで実行されます. OpenMP 環境変数 OMP_NESTED の値を確認・設定することにより, OpenMP ネスト並列処理の有効／無効の確認・設定を行うことができます. OpenMP ネスト並列処理が有効になっている場合,上位レベルの各スレッドの各並列領域に N 個のスレッドが作成されるため, 例えば, 2 つのレベルの OpenMP 並列処理がある場合, 合計 N * N スレッドになります. ネスト並列処理では, 各レベルで必要なスレッド数を, 環境変数 OMP_NUM_THREADS にカンマ区切りで指定することができます.

C シェルの場合 :
setenv OMP_NUM_THREADS N, P

Bourne シェルの場合 :
OMP_NUM_THREADS=N, P
export OMP_NUM_THREADS

この設定例では, 第 1 レベルの並列処理に対して N 個のスレッドが生成され, 内部レベルの並列処理に対して P 個のスレッドが生成されます.

注意：環境変数 OMP_NUM_THREADS が設定されていない場合，デフォルト値はコンパイラ毎，またベンダーライブラリ毎に異なります．通常は，1 もしくはシステムで使用可能な最大コア数に等しくなります．特に後者では，システムを他のユーザーと共有している場合や，自分のアプリケーション内で複数レベルの並列処理を実行している場合に問題となる可能性があります．従って，OMP_NUM_THREADS は明示的に設定することをお勧めします．

一般的に，推奨されるスレッドの最大数は，ご利用の共有メモリシステムの物理コア数です．ただし，殆どの Intel プロセッサはハイパースレッディングと呼ばれる機能をサポートしています．この機能は１つの物理コアが同時に２つのスレッドをサポートすることを可能にします（従って，オペレーティングシステムには２つの論理コアとして認識されます）．この機能が有益かどうかは，使用するアルゴリズムや問題のサイズに依存します．従って，自身のアプリケーションにとってこの機能が有益かどうかは，追加の論理コアを使用する場合と使用しない場合でベンチマークを取り判断することをお勧めします．これは，使用するスレッド数を OMP_NUM_THREADS に設定するだけで簡単に実現できます．ハイパースレッディングの完全な無効化は，通常，起動時にシステムの BIOS 設定で行うことができます．

本製品で提供される Intel MKL ライブラリには（OMP_NUM_THREADS 以外にも）MKL 内のスレッドをより細かく制御するための幾つもの環境変数があります．これらの環境変数の詳細につきましては，以下の Intel 社のウェブサイトをご参照ください．
https://software.intel.com/en-us/articles/intel-math-kernel-library-intel-mkl-intel-mkl-100-threading
多くの NAG ルーチンは MKL ルーチンを利用しています．従って，MKL 環境変数は間接的に NAG ライブラリの動作にも影響を与えます．基本的には，MKL 環境変数のデフォルト設定が NAG ライブラリには適しています．従って，これらの MKL 環境変数を明示的に設定しないことをお勧めします．

3.2. Fortran インターフェースブロック

NAG ライブラリのインターフェースブロック(引用仕様宣言)は NAG ライブラリの Fortran ルーチンの型と引数を定義します. Fortran プログラムから NAG ライブラリを呼び出す際に必ず必要という性質のものではありませんが, その利用が推奨されます（ただし, 本製品で提供される Example を利用する際には必ず必要です). これを用いることで NAG ライブラリルーチンが正しく呼び出されているかどうかのチェックを Fortran コンパイラに任せる事ができます. 具体的にはコンパイラが以下のチェックを行うことを可能とします.

(a) サブルーチン呼び出しの整合性
(b) 関数宣言の型
(c) 引数の数
(d) 引数の型

NAG ライブラリのインターフェースブロックファイルはチャプター毎のモジュールとして提供されますが, これらをまとめて一つにしたモジュールが提供されます.

nag_library

これらのモジュールは, Intel Fortran コンパイラ用にコンパイルされた形式（*.mod ファイル）で提供されます. コンパイル時に -I"pathname" オプションを用いて, モジュールファイルが置かれているディレクトリのパス,
[INSTALL_DIR]/lp64/nag_interface_blocks
または,
[INSTALL_DIR]/ilp64/nag_interface_blocks
を（必要な整数サイズに応じて）指定してください.

提供されるモジュールファイル（.mod ファイル）は, インストールノートの「2.2. 開発環境」にある Fortran コンパイラを用いて生成されています. モジュールファイルはコンパイラ依存のファイルであるため, ご利用のコンパイラとの間に互換性がない場合は, ご利用のコンパイラでモジュールファイルを生成する必要があります.（自身のプログラムでインターフェースブロックをご利用にならないのであれば, この限りではありません. ただし, Example プログラムはインターフェースブロックを利用しますので, Example プログラムをご利用になる場合は必要です.）

提供されるスクリプト nag_recompile_mods を用いて，モジュールファイルのセットを指定のディレクトリ（例えば，nag_interface_blocks_alt）に生成することができます.

[INSTALL_DIR]/scripts/nag_recompile_mods {int32,int64} nag_interface_blocks_alt

このスクリプトは PATH 環境変数に設定された Intel Fortran コンパイラを使用します. [INSTALL_DIR]/scripts/nag_recompile_mods を実行する前に，ご利用の Intel Fortran コンパイラの環境設定スクリプトを実行しておくと安心です.

新しいモジュールファイルのセットをデフォルトのセットとして使用するには，以下のコマンドで（ [INSTALL_DIR] を適切な実際のディレクトリパスに置き換えて），デフォルトのセットを変更します.

int32（lp64）の場合：

```
mv [INSTALL_DIR]/lp64/nag_interface_blocks \
   [INSTALL_DIR]/lp64/nag_interface_blocks_original
mv [INSTALL_DIR]/lp64/nag_interface_blocks_alt \
   [INSTALL_DIR]/lp64/nag_interface_blocks
```

int64（ilp64）の場合：

```
mv [INSTALL_DIR]/ilp64/nag_interface_blocks \
   [INSTALL_DIR]/ilp64/nag_interface_blocks_original
mv [INSTALL_DIR]/ilp64/nag_interface_blocks_alt \
   [INSTALL_DIR]/ilp64/nag_interface_blocks
```

これで，新しくコンパイルされたモジュールファイルを通常の方法で使用できるようになります.

NAG AD ルーチンのインターフェースブロックについては，ユーザーが再コンパイルすることはできません. もし必要な場合は，NAG にお問い合わせください（「7. コンタクト情報」参照）.

3.3. Example プログラム

提供される Example 結果は，インストールノートの「2.2. 開発環境」に記載されている環境で生成されています．Example プログラムの実行結果は，異なる環境下（例えば，異なる C または Fortran コンパイラ，異なるコンパイラランタイムライブラリ，異なる BLAS または LAPACK ルーチンなど）で若干異なる場合があります．そのような違いが顕著な計算結果としては，固有ベクトル（スカラー（多くの場合 -1）倍の違い），反復回数や関数評価，残差（その他マシン精度と同じくらい小さい量）などがあげられます．

提供される Example 結果は，32-bit 整数のスタティックライブラリ libnag_mkl.a（MKL 提供の BLAS／LAPACK ルーチンを使用）を用いて得られたものです．NAG 提供の BLAS／LAPACK ルーチンを使用した場合は，結果が僅かに異なるかもしれません．

提供される NAG AD Example 結果は，32-bit 整数のスタティックライブラリ libnag_mkl.a と libnag_mkl_ad.a を用いて得られたものです．

Example プログラムは本ライブラリが想定する動作環境に適した状態で提供されます．そのため，ライブラリマニュアルに記載されている Example プログラムに比べて，その内容が若干異なる場合があります．

スクリプト nag_example を用いて Example プログラムを簡単に利用する事ができます．このスクリプトは，[INSTALL_DIR]/scripts ディレクトリに提供されます．

このスクリプトは，Example プログラムのソースファイル（必要に応じて，データファイル，オプションファイルその他）をカレントディレクトリにコピーして，コンパイル・リンク・実行を行います．デフォルトでは，nag_example は，32-bit 整数，スタティック，MKL に依存しないライブラリ libnag_nag.a をリンクします．これらのデフォルト値はそれぞれ，-int64，-shared，-vendor オプションで変更することができます（これにより，リンクするライブラリの種類を選択することができます）．-quiet オプションを指定すると，コマンドとコメントの両方の出力を最小限にします．以下のコマンドで，利用可能なオプションのリストを見ることができます．

```
nag_example -help
```

nag_example スクリプトは，「3.1. ライブラリのリンク方法」で説明した nagvars スクリプトを利用しますが，呼び出しシェルの環境を変更しないことに注意してください．

ご利用の NAG ライブラリルーチンの名前をスクリプトの引数に指定してください.
例えば, NAG C ルーチン e04ucc の場合,

```
nag_example e04ucc
```

この例では, e04ucce.c（ソースファイル), e04ucce.d（データファイル), e04ucce.opt（オプションファイル）をカレントディレクトリにコピーして, コンパイル・リンク・実行を行い, e04ucce.r（結果ファイル）を生成します.

同様に, NAG Fortran ルーチン e04nrf の場合,

```
nag_example e04nrf
```

この例では, e04nrfe.f90（ソースファイル), e04nrfe.d（データファイル), e04nrfe.opt（オプションファイル）をカレントディレクトリにコピーして, コンパイル・リンク・実行を行い, e04nrfe.r（結果ファイル）を生成します.

NAG AD ルーチンの Example を実行するには, -ad オプションを追加する必要があります.

```
nag_example -ad s01ba_a1w_hcpp
```

AD の Example のいくつかは, 本製品に付属していない dco.hpp ファイルに依存していることに注意してください. これらの Example の使用に興味がある場合は, NAG にお問い合わせください（「7. コンタクト情報」参照).

dco.hpp ファイルおよび他の dco/c++ ファイルが既にシステムにある場合は, 環境変数 DCODIR にそれらの格納場所が設定されているかもしれません. nagvars スクリプトの実行時にこれが行われた場合は, コンパイルコマンドで使用されるインクルードパスに DCODIR ディレクトリが追加されます.

3.4. メンテナンスレベル

ライブラリのメンテナンスレベルは，ライブラリルーチン a00aaf または a00aac の Example プログラムをコンパイル･リンク･実行することにより確認することができます. この時，スクリプト nag_example を引数 a00aaf または a00aac と共に用いれば，Example プログラムのコンパイル・リンク・実行を容易に行うことができます（「3.3. Example プログラム」参照）. ライブラリルーチン a00aaf または a00aac は，ライブラリの詳細（タイトル，製品コード，使用されるコンパイラおよび精度，Mark など）を出力します.

3.5. C データ型

本製品には，32-bit 整数と 64-bit 整数の両方のライブラリが含まれています.

32-bit 整数ライブラリ（ [INSTALL_DIR]/lp64/lib ディレクトリにある）の場合，NAG C データ型の Integer と Pointer は，次のように定義されています.

NAG 型	C 型	サイズ（バイト）
Integer	int	4
Pointer	void *	8

64-bit 整数ライブラリ（ [INSTALL_DIR]/ilp64/lib ディレクトリにある）の場合，NAG C データ型の Integer と Pointer は，次のように定義されています.

NAG 型	C 型	サイズ（バイト）
Integer	long	8
Pointer	void *	8

sizeof(Integer) と sizeof(Pointer) の値は a00aac の Example プログラムから得ることもできます. その他の NAG データ型の情報は，ライブラリマニュアルの
"NAG CL Interface Introduction" ドキュメントの「3.1.1 NAG data types」をご参照ください.

3.6. Fortran データ型

本製品には, 32-bit 整数ライブラリ（ [INSTALL_DIR]/lp64/lib ディレクトリにある）と 64-bit 整数ライブラリ（ [INSTALL_DIR]/ilp64/lib ディレクトリにある）の両方が含まれています.

NAG ライブラリとライブラリマニュアルでは, 実数の変数を以下のようにパラメーター化された型を用いて記述しています.

REAL(KIND=nag_wp)

ここで nag_wp は Fortran の種別パラメーターを表しています.
nag_wp の値は製品毎に異なり, その値は nag_library モジュールに定義されています.

これに加え, いくつかのルーチンで以下の型が使用されます.

REAL(KIND=nag_rp)

これらの型の使用例については, 各 Example プログラムをご参照ください.

本製品では, これらの型は次のような意味を持っています.

REAL(KIND=nag_rp)　　- REAL（単精度実数）
REAL(KIND=nag_wp)　　- DOUBLE PRECISION（倍精度実数）
COMPLEX(KIND=nag_rp) - COMPLEX（単精度複素数）
COMPLEX(KIND=nag_wp) - 倍精度複素数（e.g. COMPLEX*16）

これらに加え, ライブラリマニュアルの FL Interface セクションでは, 強調斜体文字を用いて, いくつかの用語を表現しています. 詳細につきましては,
ライブラリマニュアルの "NAG FL Interface Introduction" ドキュメントの「2.5 Implementation-dependent Information」をご参照ください.

3.7. C/C++ からの NAG Fortran ルーチンの呼び出し

NAG ライブラリの Fortran ルーチンは C, C++（または互換性のある環境）からもご利用いただけます. ユーザーが Fortran 型と C 型の間のマッピングを行うのを支援するために, C の観点からの Fortran インターフェースの説明（C ヘッダーインターフェース）が各 Fortran ルーチンのドキュメントに含まれています. また, C/C++ ヘッダーファイル（32-bit 整数の場合は [INSTALL_DIR]/lp64/include/nag.h, 64-bit 整数の場合は [INSTALL_DIR] /ilp64/include/nag.h）が提供されます.

C または C++ から NAG ライブラリの Fortran ルーチンを呼び出す際のアドバイスは, "alt_c_interfaces.html" ドキュメントをご参照ください.（なお, NAG ライブラリの以前の Mark では, このドキュメントは "techdoc.html" と呼ばれていました.）

3.8. LAPACK, BLAS などの C 宣言

NAG C/C++ ヘッダーファイルには, NAG ライブラリに含まれている LAPACK, BLAS および BLAS Technical Forum（BLAST）ルーチンの宣言が含まれています. ユーザーは, 他のライブラリ（例えば, 付属の Intel MKL など）の C インクルードファイルから, これらのルーチンの定義を取得したいと思うかもしれません. このような場合, 異なる C ヘッダー宣言の衝突を避けるため,「3.1. ライブラリのリンク方法」の C または C++ のコンパイルコマンドに以下のコンパイルオプションを追加することで, これらのルーチンの NAG 宣言を無効にすることができます.

```
-DNAG_OMIT_LAPACK_DECLARATION -DNAG_OMIT_BLAS_DECLARATION \
-DNAG_OMIT_BLAST_DECLARATION
```

ただし, F01, F06, F07, F08 の NAG ルーチン名の宣言は残ります.

4. ルーチン固有の情報

本製品のライブラリルーチン固有の情報を（チャプター毎に）以下に示します.

a. F06, F07, F08, F16

チャプター F06, F07, F08, F16 では, BLAS／LAPACK 由来のルーチンに対して NAG スタイルのルーチン名が与えられています. ルーチン名の詳細については, 各チャプターイントロダクションをご参照ください. パフォーマンスの面からは, NAG スタイルの名前よりも BLAS／LAPACK スタイルの名前でルーチンを使用してください.

多くの LAPACK ルーチンには, 呼び出し側にどれだけのワークスペースが必要であるかをルーチンに問い合わせる "workspace query" メカニズムがあります. NAG が提供する LAPACK と MKL が提供する LAPACK では, このワークスペースのサイズが異なる場合があるので注意してください.

MKL に依存するバージョンの NAG ライブラリでは, BLAS／LAPACK ルーチンの呼び出しに関して, MKL が提供する BLAS／LAPACK ルーチンが使われます. ただし, 以下の BLAS／LAPACK ルーチンの呼び出しは, NAG が提供する BLAS／LAPACK ルーチンが使われます.

```
blas_damax_val  blas_damin_val  blas_daxpby     blas_ddot       blas_dmax_val
blas_dmin_val   blas_dsum       blas_dwaxpby    blas_zamax_val  blas_zamin_val
blas_zaxpby     blas_zsum       blas_zwaxpby
dbdsvdx dgesvdx dgesvj  dsbgvd  zgejsv  zgesvdx zgesvj  zhbgvd
```

b. S07 - S21

これらのチャプターの関数の動作は製品毎に異なります.

一般的な詳細はライブラリマニュアルをご参照ください.
本製品に固有の値を以下に示します.

```
s07aa[f] (nag[f]_specfun_tan)
    F_1 = 1.0e+13
    F_2 = 1.0e-14
```

```
s10aa[fc] (nag[f]_specfun_tanh)
    E_1 = 1.8715e+1
s10ab[fc] (nag[f]_specfun_sinh)
    E_1 = 7.080e+2
s10ac[fc] (nag[f]_specfun_cosh)
    E_1 = 7.080e+2

s13aa[fc] (nag[f]_specfun_integral_exp)
    x_hi = 7.083e+2
s13ac[fc] (nag[f]_specfun_integral_cos)
    x_hi = 1.0e+16
s13ad[fc] (nag[f]_specfun_integral_sin)
    x_hi = 1.0e+17

s14aa[fc] (nag[f]_specfun_gamma)
    ifail = 1 (NE_REAL_ARG_GT) if x > 1.70e+2
    ifail = 2 (NE_REAL_ARG_LT) if x < -1.70e+2
    ifail = 3 (NE_REAL_ARG_TOO_SMALL) if abs(x) < 2.23e-308
s14ab[fc] (nag[f]_specfun_gamma_log_real)
    ifail = 2 (NE_REAL_ARG_GT) if x > x_big = 2.55e+305

s15ad[fc] (nag[f]_specfun_erfc_real)
    x_hi = 2.65e+1
s15ae[fc] (nag[f]_specfun_erf_real)
    x_hi = 2.65e+1
s15ag[fc] (nag[f]_specfun_erfcx_real)
    ifail = 1 (NW_HI) if x >= 2.53e+307
    ifail = 2 (NW_REAL) if 4.74e+7 <= x < 2.53e+307
    ifail = 3 (NW_NEG) if x < -2.66e+1

s17ac[fc] (nag[f]_specfun_bessel_y0_real)
    ifail = 1 (NE_REAL_ARG_GT) if x > 1.0e+16
s17ad[fc] (nag[f]_specfun_bessel_y1_real)
    ifail = 1 (NE_REAL_ARG_GT) if x > 1.0e+16
    ifail = 3 (NE_REAL_ARG_TOO_SMALL) if 0 < x <= 2.23e-308
```

s17ae[fc] (nag[f]_specfun_bessel_j0_real)

ifail = 1 (NE_REAL_ARG_GT) if abs(x) > 1.0e+16

s17af[fc] (nag[f]_specfun_bessel_j1_real)

ifail = 1 (NE_REAL_ARG_GT) if abs(x) > 1.0e+16

s17ag[fc] (nag[f]_specfun_airy_ai_real)

ifail = 1 (NE_REAL_ARG_GT) if x > 1.038e+2

ifail = 2 (NE_REAL_ARG_LT) if x < -5.7e+10

s17ah[fc] (nag[f]_specfun_airy_bi_real)

ifail = 1 (NE_REAL_ARG_GT) if x > 1.041e+2

ifail = 2 (NE_REAL_ARG_LT) if x < -5.7e+10

s17aj[fc] (nag[f]_specfun_airy_ai_deriv)

ifail = 1 (NE_REAL_ARG_GT) if x > 1.041e+2

ifail = 2 (NE_REAL_ARG_LT) if x < -1.9e+9

s17ak[fc] (nag[f]_specfun_airy_bi_deriv)

ifail = 1 (NE_REAL_ARG_GT) if x > 1.041e+2

ifail = 2 (NE_REAL_ARG_LT) if x < -1.9e+9

s17dc[fc] (nag[f]_specfun_bessel_y_complex)

ifail = 2 (NE_OVERFLOW_LIKELY) if abs(z) < 3.92223e-305

ifail = 4 (NW_SOME_PRECISION_LOSS) if abs(z) or fnu+n-1 > 3.27679e+4

ifail = 5 (NE_TOTAL_PRECISION_LOSS) if abs(z) or fnu+n-1 > 1.07374e+9

s17de[fc] (nag[f]_specfun_bessel_j_complex)

ifail = 2 (NE_OVERFLOW_LIKELY) if AIMAG(z) > 7.00921e+2

ifail = 3 (NW_SOME_PRECISION_LOSS) if abs(z) or fnu+n-1 > 3.27679e+4

ifail = 4 (NE_TOTAL_PRECISION_LOSS) if abs(z) or fnu+n-1 > 1.07374e+9

s17dg[fc] (nag[f]_specfun_airy_ai_complex)

ifail = 3 (NW_SOME_PRECISION_LOSS) if abs(z) > 1.02399e+3

ifail = 4 (NE_TOTAL_PRECISION_LOSS) if abs(z) > 1.04857e+6

s17dh[fc] (nag[f]_specfun_airy_bi_complex)

ifail = 3 (NW_SOME_PRECISION_LOSS) if abs(z) > 1.02399e+3

ifail = 4 (NE_TOTAL_PRECISION_LOSS) if abs(z) > 1.04857e+6

s17dl[fc] (nag[f]_specfun_hankel_complex)

ifail = 2 (NE_OVERFLOW_LIKELY) if abs(z) < 3.92223e-305

ifail = 4 (NW_SOME_PRECISION_LOSS) if abs(z) or fnu+n-1 > 3.27679e+4

ifail = 5 (NE_TOTAL_PRECISION_LOSS) if abs(z) or fnu+n-1 > 1.07374e+9

s18ad[fc] (nag[f]_specfun_bessel_k1_real)
    ifail = 2 (NE_REAL_ARG_TOO_SMALL) if 0 < x <= 2.23e-308
s18ae[fc] (nag[f]_specfun_bessel_i0_real)
    ifail = 1 (NE_REAL_ARG_GT) if abs(x) > 7.116e+2
s18af[fc] (nag[f]_specfun_bessel_i1_real)
    ifail = 1 (NE_REAL_ARG_GT) if abs(x) > 7.116e+2
s18dc[fc] (nag[f]_specfun_bessel_k_complex)
    ifail = 2 (NE_OVERFLOW_LIKELY) if abs(z) < 3.92223e-305
    ifail = 4 (NW_SOME_PRECISION_LOSS) if abs(z) or fnu+n-1 > 3.27679e+4
    ifail = 5 (NE_TOTAL_PRECISION_LOSS) if abs(z) or fnu+n-1 > 1.07374e+9
s18de[fc] (nag[f]_specfun_bessel_i_complex)
    ifail = 2 (NE_OVERFLOW_LIKELY) if REAL(z) > 7.00921e+2
    ifail = 3 (NW_SOME_PRECISION_LOSS) if abs(z) or fnu+n-1 > 3.27679e+4
    ifail = 4 (NE_TOTAL_PRECISION_LOSS) if abs(z) or fnu+n-1 > 1.07374e+9

s19aa[fc] (nag[f]_specfun_kelvin_ber)
    ifail = 1 (NE_REAL_ARG_GT) if abs(x) >= 5.04818e+1
s19ab[fc] (nag[f]_specfun_kelvin_bei)
    ifail = 1 (NE_REAL_ARG_GT) if abs(x) >= 5.04818e+1
s19ac[fc] (nag[f]_specfun_kelvin_ker)
    ifail = 1 (NE_REAL_ARG_GT) if x > 9.9726e+2
s19ad[fc] (nag[f]_specfun_kelvin_kei)
    ifail = 1 (NE_REAL_ARG_GT) if x > 9.9726e+2

s21bc[fc] (nag[f]_specfun_ellipint_symm_2)
    ifail = 3 (NE_REAL_ARG_LT) if an argument < 1.583e-205
    ifail = 4 (NE_REAL_ARG_GE) if an argument >= 3.765e+202
s21bd[fc] (nag[f]_specfun_ellipint_symm_3)
    ifail = 3 (NE_REAL_ARG_LT) if an argument < 2.813e-103
    ifail = 4 (NE_REAL_ARG_GT) if an argument >= 1.407e+102

c. X01

数学定数を以下に示します.

```
x01aa[fc] (nag[f]_math_pi)
     = 3.1415926535897932
x01ab[fc] (nag[f]_math_euler)
     = 0.5772156649015328
```

d. x02

マシン定数を以下に示します.

浮動小数点演算の基本的なパラメーター:

```
x02bh[fc] (nag[f]_machine_model_base)
     = 2
x02bj[fc] (nag[f]_machine_model_digits)
     = 53
x02bk[fc] (nag[f]_machine_model_minexp)
     = -1021
x02bl[fc] (nag[f]_machine_model_maxexp)
     = 1024
```

浮動小数点演算の派生的なパラメーター:

```
x02aj[fc] (nag[f]_machine_precision)
     = 1.11022302462516e-16
x02ak[fc] (nag[f]_machine_real_smallest)
     = 2.22507385850721e-308
x02al[fc] (nag[f]_machine_real_largest)
     = 1.79769313486231e+308
x02am[fc] (nag[f]_machine_real_safe)
     = 2.22507385850721e-308
x02an[fc] (nag[f]_machine_complex_safe)
     = 2.22507385850721e-308
```

コンピューター環境のその他のパラメーター：

```
x02ah[fc] (nag[f]_machine_sinarg_max)
     = 1.42724769270596e+45
x02bb[fc] (nag[f]_machine_integer_max)
     =          2147483647 (for 32-bit integer libraries)
     = 9223372036854775807 (for 64-bit integer libraries)
x02be[fc] (nag[f]_machine_decimal_digits)
     = 15
```

e. X04

Fortran ルーチン：エラーメッセージおよびアドバイスメッセージのデフォルトの出力先装置番号は 6 番となります.

f. X06

本製品では, X06 ルーチンは MKL のスレッドの振る舞いに影響を与えません.

5. ドキュメント

ライブラリマニュアルは本製品の一部として提供されます.
また, NAG のウェブサイトからダウンロードすることもできます.
ライブラリマニュアルの最新版は, 以下のウェブサイトをご参照ください.
https://www.nag.com/numeric/nl/nagdoc_27.1/

ライブラリマニュアルは HTML5（HTML／MathML マニュアル）で提供されます.

これらのドキュメントは Web ブラウザでご利用いただけます.

以下のマスター目次ファイルが提供されます.

nagdoc_27.1/index.html

ドキュメントの閲覧方法および操作方法については, 以下のドキュメントをご参照ください.
https://www.nag.com/numeric/nl/nagdoc_27.1/nlhtml/genint/naglibdoc.html

加えて, 以下のドキュメントが提供されます.

- in.html - インストールノート（英語版）
- un.html - ユーザーノート（本ドキュメントの英語版）
- alt_c_interfaces.html - C および C++ から NAG ライブラリの Fortran ルーチンを呼び出すためのアドバイス

6. サポート

製品のご利用に関してご質問等がございましたら，電子メールにて「日本 NAG ヘルプデスク」までお問い合わせください．その際，ご利用の製品の製品コード（NLL6I271BL）並びに，お客様の User ID をご明記いただきますようお願い致します．
ご返答は平日 9：30～12:00，13:00～17:30 に行わせていただきます．

日本 NAG ヘルプデスク

Email: naghelp@nag-j.co.jp

7. コンタクト情報

日本ニューメリカルアルゴリズムズグループ株式会社（日本 NAG）

〒104-0032
東京都中央区八丁堀 4-9-9 八丁堀フロンティアビル 2F

Email: sales@nag-j.co.jp
Tel:　03-5542-6311
Fax:　03-5542-6312

NAG のウェブサイトでは製品およびサービスに関する情報を定期的に更新しています．

https://www.nag-j.co.jp/　（日本）
https://www.nag.com/　　（英国本社）