NAG Library, Mark 30

NLL6I30DBL - Licence Managed

Linux, 64-bit, Intel Classic C/C++ or Intel Classic Fortran

# ユーザーノート

目次

## 1 イントロダクション

本ドキュメントは、NAG ライブラリの実装「NLL6I30DBL」を使用する方ための必読ドキュメントです。

このドキュメントは、NAG Mark 30 Library Manual（以下、Library Manual と呼びます）に記載されている情報を補足する実装固有の詳細を提供します。ライブラリマニュアルで「お使いの実装のユーザーズノート」と書かれている場所では、必ずこのノートを参照してください。

さらに、NAG では、ライブラリルーチンを呼び出す前に、ライブラリマニュアルから以下の参考資料を読むことをお勧めします（セクション 5 を参照）。

1. (a)NAG ライブラリの使用方法
2. (b)章のイントロダクション
3. (c)ルーチンドキュメント

## 2 追加情報

このライブラリの適用性や使用法に関する新しい情報の詳細については、以下の URL をご確認ください。

https://support.nag.com/doc/inun/nl30/l6idbl/supplementary.html

## 3 一般情報

この NAG ライブラリの実装では、Basic Linear Algebra Subprograms（BLAS）および Linear Algebra PACKage（LAPACK）ルーチン（セクション 4 にリストされているルーチンを除く）を提供するために、サードパーティベンダーのパフォーマンスライブラリである Linux 用の Intel ® Math Kernel Library（MKL）を使用する静的ライブラリと共有ライブラリを提供しています。また、これらのルーチンの NAG 参照バージョンを自己完結型で使用する静的ライブラリと共有ライブラリ（以下、自己完結型ライブラリと呼びます）も提供しています。この実装では、本製品に付属する MKL のバージョン 2021.0.4 でテストされています。MKL の詳細については、Intel のウェブサイト（https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html）を参照してください。パフォーマンスを最大限に発揮するには、自己完結型の NAG ライブラリ libnag_nag.a または libnag_nag.so を使用するよりも、提供されている MKL ベンダーライブラリに基づく NAG ライブラリのバリアント、つまり libnag_mkl.a または libnag_mkl.so を使用することをお勧めします。

この実装には、32 ビット整数（lp64 で表す）と 64 ビット整数（ilp64 で表す）の両方で使用するためのライブラリ（および関連ファイル）が含まれています。

NAG AD Library も、バリアントアドオンライブラリとして libnag_nag_ad.a、libnag_nag_ad.so、libnag_mkl_ad.a、および libnag_mkl_ad.so として含まれています。nag_mkl_ad バリアントでは、いくつかのコアレベル 3 BLAS ルーチンの導関数をより効率的に計算するために、シンボリック行列計算が使用されます。Fortran ユーザーのためのみの NAG AD ライブラリを使用する場合、上記のライブラリに加えて libnag_dcof.a をリンクする必要があります。

NAG ライブラリは、使用されるメモリを回収できるように慎重に設計されていることに注意してください。ライブラリ自体による回収や、C ルーチンでは NAG_FREE()呼び出しによるユーザー呼び出しによる回収が可能です。しかし、ライブラリ自体は、コンパイラの実行時間やその他のライブラリの使用に依存

しており、それらがメモリをリークすることがあります。NAG ライブラリにリンクされたプログラムでメモリトレースツールを使用すると、このことが報告される場合があります。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りますが、過剰になることはなく、NAG ライブラリの呼び出しを増やしても際限なく増加することはありません。

マルチスレッドアプリケーション内で NAG ライブラリを使用する場合は、ドキュメント *CL インターフェイスマルチスレッド*または *FL インターフェイスマルチスレッド*(適切な方)を参照してください。

提供された Intel MKL ライブラリをスレッド化されたアプリケーションで使用する方法の詳細については、https://software.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top/managing-performance-and-memory/improving-performance-with-threading.html を参照してください。

NAG AD ライブラリはスレッドセーフです。ただし、コードをプリプロセッサマクロ DCO_NO_THREADLOCAL を定義してコンパイルすると、スレッドセーフ性が保持されなくなる可能性があることに注意してください。

この実装に付属のライブラリには、OpenMP やその他のスレッドメカニズムは含まれていません。ただし、MKL ベンダーライブラリは OpenMP スレッド化されています。このスレッド化を制御する方法の詳細については、セクション 3.1.1 を参照してください。

Intel は MKL に条件付きビット単位再現性(BWR)オプションを導入しました。ユーザーのコードが特定の条件(https://software.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top/obtaining-numerically-reproducible-results/reproducibility-conditions.html を参照)に準拠している場合、環境変数 MKL_CBWR を設定することで BWR を強制することができます。詳細は MKL のドキュメントを参照してください。ただし、多くの NAG ルーチンはこれらの条件に準拠していないことに注意してください。つまり、MKL に基づいて構築された NAG ライブラリでは、MKL_CBWR を設定しても、異なる CPU アーキテクチャ間ですべての NAG ルーチンの BWR を保証できない可能性があります。ビット単位再現性に関する一般的な情報については、*NAG ライブラリの使用方法*のセクション 8.1 を参照してください。

### 3.1 ライブラリへのアクセス

このセクションでは、ライブラリがディレクトリ[INSTALL_DIR]にインストールされていることを前提としています。デフォルトでは、[INSTALL_DIR](インストーラーズノート(in.html)を参照)は $HOME/NAG/nll6i30dbl ですが、インストールを行った人によって変更されている可能性があります。その場合は、その人に確認する必要があります。リンク時と実行時に、環境変数 LD_LIBRARY_PATH を適切なライブラリの場所([INSTALL_DIR]の下)を指すように正しく設定する必要があることに注意してください。設定方法は以下を参照してください。

NAG ライブラリは、NAG C ライブラリと NAG Fortran ライブラリ(Fortran ライブラリインターフェイスへの C ラッパーを含む)の両方のユーザーのための統合された代替ライブラリです。NAG ルーチンを呼び出すアプリケーションのコンパイルとリンクを支援するために、スクリプト nagvars.sh と nagvars.csh が含まれており、NAG 固有の環境変数を設定します。また、標準の環境変数 PATH と LD_LIBRARY_PATH を修正し、NAG の実行可能プログラムとライブラリがコンパイル時、リンク時、実行時に見つかるようにします。

nagvars スクリプトは以下のように使用するように設計されています。

```
. [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、

- ·Bourne、Bash、または同等のシェルには nagvars.sh を使用します（注：Ubuntu などの Debian 系 Linux ディストリビューションで利用可能な Dash は使用できません）。Csh、Tcsh、または同等のシェルには nagvars.csh を使用します。
- ·{int32,int64}は、NAG ルーチン内の整数引数と変数のデフォルトサイズを指定します。
- ·{vendor,nag}は、BLAS と LAPACK に付属の MKL ライブラリに依存する NAG ライブラリのセット（オプション vendor）を使用するか、自己完結型の NAG ライブラリ（オプション nag）を使用するかを指定します。最高のパフォーマンスを発揮するには、オプション vendor をお勧めします。
- ·{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。

デフォルト値は使用されないため、3 つのオプションすべてを設定する必要があります。指定する順序は重要ではありません。nagvars スクリプトのオプション引数は次のとおりです。

- ·–help はスクリプトに関する情報を表示します。
- ·–unset は、NAG 固有の環境変数をクリアし、標準の環境変数 PATH と LD_LIBRARY_PATH から、**この NAG 実装からのみの**マテリアルへのすべての参照を削除しようとします。
- ·–quiet は標準出力への出力を抑制します。
- ·–gnu はこのセクションの最後で説明します。

したがって、Bash で環境を設定するために nagvars スクリプトを使用する例は次のようになります。

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

設定される NAG 固有の環境変数は次のとおりです。

- ·NAGLIB_CC – この NAG ライブラリの作成に使用された C コンパイラ。
- ·NAGLIB_CXX – この NAG ライブラリの作成に使用された C++コンパイラ。
- ·NAGLIB_F77 – この 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 コンパイラフラグ。
- ·NAGLIB_CXXFLAGS – 必要または推奨される C++コンパイラフラグ。
- ·NAGLIB_FFLAGS – 必要または推奨される Fortran コンパイラフラグ。

- ·NAGLIB_INCLUDE – NAG C ヘッダーおよび Fortran モジュールファイルへのインクルードパス。
- ·NAGLIB_CLINK – C コンパイラを使用して通常の NAG（およびオプションでベンダーBLAS と LAPACK）ルーチンにリンクするために必要なライブラリ。
- ·NAGLIB_CXXLINK – C++コンパイラを使用して通常の NAG（およびオプションでベンダー BLAS と LAPACK）ルーチンにリンクするために必要なライブラリ。
- ·NAGLIB_FLINK – Fortran コンパイラを使用して通常の NAG（およびオプションでベンダー BLAS と LAPACK）ルーチンにリンクするために必要なライブラリ。
- ·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、または NAGLIB_FLINK の前にリンクする必要があります。

NAG ライブラリと提供された Intel MKL ライブラリ（必要な場合）を使用するには、以下のようにリンクできます。

- C プログラムの場合:
  - ${NAGLIB_CC} ${NAGLIB_CFLAGS} ${NAGLIB_INCLUDE} program.c ${NAGLIB_AD_LINK} ¥
    ${NAGLIB_CLINK}

    NAG AD ルーチンが必要な場合。それ以外の場合は、

    ${NAGLIB_CC} ${NAGLIB_CFLAGS} ${NAGLIB_INCLUDE} program.c ${NAGLIB_CLINK}
- C++プログラムの場合:
  - ${NAGLIB_CXX} ${NAGLIB_CXXFLAGS} ${NAGLIB_INCLUDE} program.cpp ${NAGLIB_AD_LINK} ¥
    ${NAGLIB_CXXLINK}

    NAG AD ルーチンが必要な場合。それ以外の場合は、

    ${NAGLIB_CXX} ${NAGLIB_CXXFLAGS} ${NAGLIB_INCLUDE} program.cpp ${NAGLIB_CXXLINK}
- Fortran プログラムの場合:
  - ${NAGLIB_F77} ${NAGLIB_FFLAGS} ${NAGLIB_INCLUDE} program.f90 ${NAGLIB_AD_LINK} ¥
    ${NAGLIB_FLINK}

    NAG AD ルーチンが必要な場合。それ以外の場合は、

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

コンパイラのランタイムライブラリなど、他のものを指すように LD_LIBRARY_PATH を設定する必要がある場合もあることに注意してください。例えば、コンパイラの新しいバージョンを使用している場合などです。

異なるコンパイラや Intel コンパイラの異なるバージョンを使用している場合は、[INSTALL_DIR]/rtl/lib/intel64 にある Intel コンパイラのランタイムライブラリにリンクする必要がある場合があります。これは、

```
[INSTALL_DIR]/rtl/lib/intel64
```

を LD_LIBRARY_PATH に追加することで実現できます。

このバージョンの NAG ライブラリは、gcc と g++がサポートされているバージョン以上を使用して、GNU コンパイラの gcc と g++から呼び出すことができます。サポートされているバージョンについては、追加情報のウェブページを参照してください。これを行う最も簡単な方法は、nagvars.sh および nagvars.csh スクリプトで-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 固有の環境変数が変更され、GNU gcc と g++（適切な場合）が使用されるようになります。

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 コンパイラの OpenMP ランタイムが使用されます。

### *3.1.1 使用するスレッド数の設定*

MKL は、一部のライブラリルーチンでスレッド化を実装するために OpenMP を利用しています。実行時に使用されるスレッド数は、環境変数 OMP_NUM_THREADS を適切な値に設定することで制御できます。

C シェルでは、次のように入力します。

```
setenv OMP_NUM_THREADS N
```

Bourne シェルでは、次のように入力します。

```
OMP_NUM_THREADS=N
export OMP_NUM_THREADS
```

ここで、N は必要なスレッド数です。環境変数 OMP_NUM_THREADS は、プログラムの実行ごとに必要に応じて再設定できます。

一部の MKL ルーチンでは、複数レベルの OpenMP 並列処理が存在する可能性があります。また、独自のアプリケーションの OpenMP 並列領域内からこれらのマルチスレッドルーチンを呼び出すこともできます。デフォルトでは、OpenMP のネスト並列処理は無効になっているため、最も外側の並列領域のみが実際にアクティブになり、上記の例では N スレッドを使用します。内側のレベルはアクティブではなく、つまり 1 つのスレッドで実行されます。OpenMP のネスト並列処理が有効になっているかどうかを確認し、有効/無効を選択するには、OMP_NESTED OpenMP 環境変数を照会して設定します。OpenMP のネスト並列処理が有効になっている場合、上記の例では、上位レベルの各スレッドに対して各並列領域に N スレッドが作成されるため、OpenMP 並列処理が 2 レベルある場合は合計 N*N スレッドになります。ネスト並列処理をより詳細に制御するために、環境変数 OMP_NUM_THREADS をコンマ区切りのリストに設定して、各レベルで必要なスレッド数を指定できます。

C シェルでは、次のように入力します。

```
setenv OMP_NUM_THREADS N,P
```

Bourne シェルでは、次のように入力します。

```
OMP_NUM_THREADS=N,P
export OMP_NUM_THREADS
```

これにより、並列処理の最初のレベルに N スレッドが作成され、内側のレベルの並列処理が検出されたときに、各外側のレベルのスレッドに対して P スレッドが作成されます。

注：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ごと、および異なるベンダーライブラリごとに異なる可能性があります。通常、1 か、システムで利用可能な最大コア数のいずれかになります。後者は、システムを他のユーザーと共有している場合や、独自のアプリケーション内で高いレベルの並列処理を実行している場合に問題になる可能性があります。したがって、OMP_NUM_THREADS を必要な値に明示的に設定することをお勧めします。

一般に、使用することをお勧めする最大スレッド数は、共有メモリシステムの物理コア数です。 ただし、ほとんどの 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おり、各物理コアが同時に最大 2 つのスレッドをサポートし、オペレーティングシステムに 2 つの論理コアとして表示されます。この機能を利用すると有益な場合がありますが、この選択は、使用する特定のアルゴリズムと問題サイズに依存します。追加の論理コアを利用する場合と利用しない場合の両方でパフォーマンスが重要なアプリケーションのベンチマークを取り、自分に最適な選択を決定することをお勧めします。これは通常、OMP_NUM_THREADS を介して使用するスレッド数を適切に選択するだけで実現できます。ハイパースレッディングを完全に無効にするには、通常、起動時にシステムの BIOS で必要な選択肢を設定する必要があります。

提供された Intel MKL ライブラリには、MKL 内のスレッド化をより細かく制御できる追加の環境変数が含まれています。これらについては、
https://www.intel.com/content/www/us/en/docs/onemkl/developer-guide-linux/2023-0/onemkl-specific-env-vars-for-openmp-thread-ctrl.html で説明されています。多くの NAG ルーチンは MKL 内のルーチンを呼び出しているため、MKL 環境変数は NAG ライブラリの動作にも間接的に影響を与える可能性があります。 MKL 環境変数のデフォルト設定はほとんどの目的に適しているはずなので、これらの変数を明示的に設定しないことをお勧めします。必要に応じて、NAG にお問い合わせください。

### 3.2 Fortran 引用仕様宣言

NAG ライブラリの引用仕様ブロックは、ユーザーが呼び出し可能な各 NAG ライブラリ 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ために不可欠ではありませんが、その使用を強くお勧めします。 また、提供される Example を使用する場合は不可欠です。

その目的は、Fortran コンパイラが NAG ライブラリルーチンが正しく呼び出されていることをチェックできるようにすることです。引用仕様ブロックにより、コンパイラは以下のことをチェックできます。

1. (a)サブルーチンがサブルーチンとして呼び出されている。
2. (b)関数が正しい型で宣言されている。

3. (c)正しい数の引数が渡されている。
4. (d)すべての引数が型と構造で一致している。

NAG ライブラリの引用仕様ブロックファイルは、ライブラリの章ごとに整理されています。それらは 1 つのモジュールにまとめられています。

```
nag_library
```

モジュールは、Intel Classic Fortran コンパイラ用のコンパイル済み形式（.mod ファイル）で提供されています。各コンパイラ呼び出しで-Ipathname オプションを指定することでアクセスできます。ここで、pathname（[INSTALL_DIR]/lp64/nag_interface_blocks または [INSTALL_DIR]/ilp64/nag_interface_blocks）は、必要な整数サイズのコンパイル済み引用仕様ブロックを含むディレクトリのパスです。

[.mod]{mono}モジュールファイルは、**インストーラーズノート**のセクション 2.2 に示されている Fortran コンパイラでコンパイルされました。このようなモジュールファイルはコンパイラに依存するため、NAG の Example プログラムを使用する場合や、独自のプログラムで引用仕様ブロックを使用する場合に、これらのモジュールと互換性のないコンパイラを使用する場合は、まず引用仕様ブロックを自分のコンパイラバージョンで再コンパイルする必要があります。再コンパイルされた引用仕様ブロックのセットは、スクリプトコマンド

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

を[INSTALL_DIR]ディレクトリから使用して、別のディレクトリ（例：nag_interface_blocks_alt）に作成できます。このスクリプトは、PATH 環境にある Intel Classic Fortran コンパイラのバージョンを使用します。別のバージョンを指定するには、[INSTALL_DIR]/scripts/nag_recompile_mods を実行する前に、そのバージョンの Intel Classic Fortran コンパイラ環境スクリプトを実行するのが最も安全です。

新しいコンパイル済みモジュールのセットをデフォルトのセットにするには、まず [INSTALL_DIR]/scripts/nag_recompile_mods への別々の呼び出しで両方のオプション{int32,int64}を使用して、両方の整数サイズの新しいモジュールセットを生成することをお勧めします。次に、以下のコマンド（[INSTALL_DIR]は実際のディレクトリパスに適宜置き換えてください）を使用して、両方のデフォルトセットを変更できます。

```
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
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はずです。

現在、ユーザーが AD ルーチンの引用仕様ブロックを再コンパイルすることはできません。これが必要な場合は、NAG にサポートをお問い合わせください（コンタクト情報の詳細はセクション 7 を参照）。

### 3.3 Example プログラム

配布されている Example 結果は、**インストーラーズノート**のセクション 2.2 で説明されているソフトウェアを使用して、Mark 30 で生成されました。これらの Example 結果は、Example プログラムを少し異な

る環境（たとえば、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、または異なる BLAS または LAPACK ルーチンのセット）で実行すると、正確に再現できない場合があります。このような違いに最も敏感な結果は次のとおりです。固有ベクトル（スカラー倍、多くの場合−1、場合によっては複素数で異なる可能性がある）、反復回数と関数評価の回数、残差、および機械精度と同じオーダーの他の「小さな」量。

配布されている Example 結果は、32 ビット整数の静的ライブラリ libnag_mkl.a（つまり、MKL BLAS および LAPACK ルーチンを使用）で取得された結果です。NAG BLAS または LAPACK で Example を実行すると、結果が少し異なる場合があります。

配布されている NAG AD の Example 結果は、32 ビット整数の静的ライブラリ libnag_mkl.a と libnag_mkl_ad.a で取得された結果です。

Example 資料は、必要に応じて、ライブラリマニュアルで公開されているものから適合され、この実装でそれ以上の変更を加えずに実行できるようになっています。可能な限り、ライブラリマニュアルのバージョンではなく、配布されている Example プログラムを使用してください。Example プログラムは、ディレクトリ[INSTALL_DIR]/scripts にあるスクリプト nag_example を使用すると最も簡単にアクセスできます。

このスクリプトは、Example プログラム（およびそのデータファイルとオプションファイル（存在する場合））のコピーを提供し、プログラムをコンパイルして適切なライブラリにリンクします。最後に、実行可能プログラムが実行され（必要に応じてデータ、オプション、結果ファイルを指定する適切な引数を使用）、結果がファイルとコマンドウィンドウに送信されます。デフォルトでは、nag_example は 32 ビット整数を使用し、自己完結型の libnag_nag.a ライブラリへの静的リンクを選択します。これらの選択は、オプションのスイッチ−int64、−shared、−vendor でそれぞれ変更できます。−quiet オプションを指定すると、上記の各ステージで実行されているコマンドと、コメントの両方の出力を最小限に抑えることができます。

```
nag_example −help
```

を実行すると、利用可能なオプションのリストが表示されます。

nag_example スクリプトは、セクション 3.1 で説明した nagvars スクリプトの使用法を示していますが、呼び出し元のシェルの環境は変更しないことに注意してください。

対象の Example プログラムは、コマンドの引数で指定します。例えば、NAG C ルーチンの場合、

```
nag_example e04ucc
```

とすると、Example プログラムとそのデータファイルおよびオプションファイル（e04ucce.c、e04ucce.d、e04ucce.opt）が現在のディレクトリにコピーされ、プログラムがコンパイルおよびリンクされて実行され、Example プログラムの結果が e04ucce.r というファイルに生成されます。

同様に、NAG Fortran ルーチンの場合、

```
nag_example e04nrf
```

とすると、Example プログラムとそのデータファイルおよびオプションファイル（e04nrfe.f90、e04nrfe.d、e04nrfe.opt）が現在のディレクトリにコピーされ、プログラムがコンパイルおよびリンクされて実行され、Example プログラムの結果が e04nrfe.r というファイルに生成されます。

NAG AD ルーチンの Example を実行するには、-ad フラグを追加する必要があります。例えば、

```
nag_example -ad s01ba_a1w_hcpp
```

のようにします。

AD の Example のいくつかは、このライブラリに同梱されていない 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 を使用することに興味がある場合は、NAG にお問い合わせください（コンタクト情報の詳細はセクション 7 を参照）。

システムに dco.hpp ファイルとその他の dco/c++ファイルがすでにある場合は、環境変数 DCODIR を設定して、それらの場所を指すようにすることができます。nagvars スクリプトの実行時にこれが行われていた場合、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、a00aaf または a00aac を呼び出す Example をコンパイルして実行するか、nag_example スクリプトに引数 a00aaf または a00aac を指定して呼び出すことで確認できます。セクション 3.3 を参照してください。この Example では、使用されたタイトルと製品コード、コンパイラと精度、マークとメンテナンスレベルなど、実装の詳細が出力されます。

### 3.5 C データ型

この実装では、32 ビットと 64 ビットの整数用のライブラリが両方含まれています。

32 ビット整数ライブラリ（ディレクトリ[INSTALL_DIR]/lp64/lib にあります）では、NAG C の型 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 ビット整数ライブラリ（ディレクトリ[INSTALL_DIR]/ilp64/lib にあります）では、NAG C の型 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 Example プログラムでも提供されています。他の NAG データ型に関する情報は、ライブラリマニュアルの *NAG CL インターフェイスイントロダクション*のセクション 3.1.1（セクション 5 を参照）で入手できます。

### 3.6 Fortran データ型と太字斜体の用語の解釈

この NAG ライブラリの実装には、32 ビット整数（ディレクトリ[INSTALL_DIR]/lp64/lib にあります）と 64 ビット整数（ディレクトリ[INSTALL_DIR]/ilp64/lib にあります）の両方のライブラリが含まれています。

NAG ライブラリとドキュメントでは、浮動小数点変数にパラメータ化された型を使用しています。したがって、

```
REAL(KIND=nag_wp)
```

という型は、NAG ライブラリルーチンのすべてのドキュメントに登場します。ここで、nag_wp は Fortran の KIND パラメータです。nag_wp の値は実装によって異なり、その値は nag_library モジュールを使用して取得できます。ライブラリで使用される浮動小数点引数と内部変数のほとんどがこの型であるため、nag_wp 型を NAG ライブラリの「作業精度」型と呼んでいます。

さらに、少数のルーチンでは、

```
REAL(KIND=nag_rp)
```

という型を使用します。ここで、nag_rp は「縮小精度」型を表します。ライブラリでは現在使用されていない別の型として、

```
REAL(KIND=nag_hp)
```

があり、「高精度」型または「追加精度」型を表します。

これらの型を正しく使用するには、ほとんどのライブラリに付属の Example プログラムを参照してください。

この実装では、これらの型は次のような意味を持ちます。

```
REAL (kind=nag_rp)      means REAL (i.e. single precision)
REAL (kind=nag_wp)      means DOUBLE PRECISION
COMPLEX (kind=nag_rp)   means COMPLEX (i.e. single precision complex)
COMPLEX (kind=nag_wp)   means double precision complex (e.g. COMPLEX*16)
```

さらに、マニュアルの FL インターフェイスのセクションでは、いくつかの用語を区別するために***太字斜体***を使用する規則を採用しています。詳細については、*NAG FL インターフェイスイントロダクション*のセクション 2.5 を参照してください。

### 3.7 C/C++から NAG Fortran ルーチンを呼び出す

注意すれば、NAG ライブラリの Fortran ルーチンを C、C++、または互換性のある環境内から使用できます。Fortran ルーチンをこのように使用するのは、C ルーチン相当が利用できないレガシー Fortran ルーチンにアクセスするため、または基本的な C データ型のみを使用する低レベルの C インターフェイスを持つ方が、他の言語から使用するのに便利な場合があるためです。Fortran と C の型のマッピングをユーザーが行えるように、各 Fortran ルーチンのドキュメントには、C の観点からみた Fortran インターフェイス（C ヘッダーインターフェイス C ヘッダーインターフェイス）の説明が含まれています。C/C++ヘッダーファイル（32 ビット整数用の[INSTALL_DIR]/lp64/include/nag.h と 64 ビット整

数用の[INSTALL_DIR]/ilp64/include/nag.h)も提供されています。NAG の Fortran ルーチンをこのように使用したい場合は、アプリケーションの適切なヘッダーファイルに#include することをお勧めします。

NAG ライブラリの Fortran ルーチンを C および C++から呼び出す方法について助言するドキュメント alt_c_interfaces.html も提供されています。(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 等の 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、これらの定義を他のライブラリ(例えば提供されている Intel MKL)に関連する C インクルードファイルから取得することを好む場合があります。このような状況では、異なる C ヘッダー宣言間の衝突を避けるために、コンパイルフラグ

-DNAG_OMIT_LAPACK_DECLARATION -DNAG_OMIT_BLAS_DECLARATION -DNAG_OMIT_BLAST_DECLARATION

をセクション 3.1 で説明した C または C++コンパイル文に追加することで、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。代替の NAG F01、F06、F07、F08 ルーチン名の宣言は残ります。

## 4 ルーチン固有の情報

この実装のルーチンの 1 つ以上に適用されるその他の情報は、以下に章ごとにリストされています。

1. (a)F06、F07、F08、F16

   第 F06 章、F07 章、F08 章、および F16 章では、BLAS および LAPACK 派生ルーチンに対して代替ルーチン名が利用可能です。代替ルーチン名の詳細については、関連する章のイントロダクションを参照してください。最適なパフォーマンスを得るには、NAG スタイルの名前ではなく、BLAS/LAPACK 名でルーチンを参照する必要があることに注意してください。

   多くの LAPACK ルーチンには、呼び出し側がルーチンに問い合わせて、供給する必要のあるワークスペースの量を判断できる「ワークスペースクエリ」メカニズムがあります。MKL ライブラリの LAPACK ルーチンでは、これらのルーチンの同等の NAG 参照バージョンとは異なる量のワークスペースが必要になる場合があることに注意してください。ワークスペースクエリメカニズムを使用する際は注意が必要です。

   この実装では、以下のルーチンを除いて、非自己完結型 NAG ライブラリの BLAS および LAPACK ルーチンへの呼び出しは、MKL への呼び出しで実装されています。

   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

2. (b)S07 - S21

これらの章の関数の動作は、実装固有の値に依存する場合があります。

一般的な詳細はライブラリマニュアルに記載されていますが、この実装で使用される具体的な値は以下のとおりです。

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

3. (c)X01

数学定数の値は次のとおりです。

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

4. (d)X02

マシン定数の値は次のとおりです。

モデルの基本パラメータ

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パラメータ

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

コンピューティング環境の他の側面のパラメータ

x02ah[fc] (nag[f]_machine_sinarg_max)
= 1.42724769270596e+45
x02bb[fc] (nag[f]_machine_integer_max)
= 2147483647 (32 ビット整数ライブラリの場合)
= 9223372036854775807 (64 ビット整数ライブラリの場合)
x02be[fc] (nag[f]_machine_decimal_digits)
= 15

5. (e)X04

Fortran ルーチン：明示的な出力を生成できるルーチンのエラーメッセージと勧告メッセージのデフォルトの出力ユニットは両方とも Fortran ユニット 6 です。

6. (f)X06

第 X06 章のルーチンは、このライブラリの実装では MKL スレッド化の動作を変更しません。

## 5 ドキュメント

ライブラリマニュアルは，以下のウェブサイトをご参照ください．NAG ライブラリマニュアル、Mark 30

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ウェブブラウザでご利用いただけます．

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．NAG ライブラリドキュメントガイド

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（NLL6I30DBL）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．

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://nag.com/（英国本社）