

NAG Library, Mark 30
NLL6I30DNL - Licence Managed
Linux, 64-bit, C++98 ABI, Intel Classic C/C++ or Intel Classic Fortran

ユーザーノート

内容

1	イントロダクション	2
2	追加情報	2
3	一般情報	2
3.1	ライブラリへのアクセス	3
3.2	Fortran 引用仕様宣言	7
3.3	Example プログラム	9
3.4	メンテナンスレベル	10
3.5	C データ型	10
3.6	Fortran データ型と太字斜体の用語の解釈	11
3.7	C/C++から NAG Fortran ルーチン呼び出す	11
3.8	LAPACK、BLAS 等の C 宣言	12
4	ルーチン固有の情報	12
5	ドキュメント	16
6	サポート	17
7	連絡先	17

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「NLL6I30DNL」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。Library Manual でお使いの実装のユーザーノートを参照している箇所では、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呼び出す前に、ライブラリマニュアルの以下のリファレンス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

この実装の適用性または使用法に関連する新しい情報の詳細については、以下の URL を確認してください。

<https://support.nag.com/doc/inun/nl30/l6idnl/supplementary.html>

3 一般情報

この NAG ライブラリの実装では、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 Intel® Math Kernel Library for Linux (MKL)を使用して、基本線形代数サブルーチン(BLAS)と線形代数パッケージ(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 リファレンス版を含む自己完結型の静的ライブラリと共有ライブラリ(自己完結型ライブラリと呼ばれる)も提供しています。この実装は、このプロダクトの一部として提供される 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。最高のパフォーマンスを得るためには、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、提供された MKL ベンダーライブラリに基づく NAG ライブラリのバリエーション(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 で示される)と 64 ビット整数(ilp64 で示される)の両方で使用するライブラリ(および関連ファイル)が含まれています。

NAG AD Library も、バリエーションアドオンライブラリとして含まれています。libnag_nag_ad.a、libnag_nag_ad.so、libnag_mkl_ad.a、libnag_mkl_ad.so です。nag_mkl_ad バリエーションでは、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せん。Fortran ユーザー向けの注意事項: 上記のライブラリを Fortran から使用する場合は、libnag_dcof.a も追加で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、ライブラリ自体または C ルーチンの場合はユーザーが NAG_FREE()を呼び出すことで再利用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。ただし、ライブラリ自体は、コンパイラのランタイムライブラリやその他のライブラリの使用に依存しており、

これらのライブラリ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ことがあります。メモリリークの量は、アプリケーションによって異なりますが、過度なものではなく、NAG ライブラリの呼び出しを増やしても無限に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予定の場合は、ドキュメント *CL Interface Multithreading* または *FL Interface Multithreading* (適宜) を参照してください。

スレッド化されたアプリケーションで提供された Intel MKL ライブラリを使用する方法の詳細は、<https://software.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top/managing-performance-and-memory/improving-performance-with-threading.html> で入手できます。

NAG AD Library はスレッドセーフです。ただし、プリプロセッサマクロ `DCO_NO_THREADLOCAL` を定義してコードをコンパイルすると、この保証が失われる可能性があることに注意してください。

この実装で提供されるライブラリには、OpenMP やその他のスレッド化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 で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に従っている

(<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) 場合、MKL_CBWR 環境変数を設定することで BWR を強制できます。詳細については、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従っていないことに注意してください。つまり、MKL をベースにした 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) を参照) は `$HOME/NAG/nll6i30dnl` 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、その人に相談してください。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。設定方法について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) の両方のユーザー向けの統合された置き換えです。このライブラリに含まれるさまざまなライブラリを呼び出すのを助けるために、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て、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します。また、標準の環境変数 `PATH` と `LD_LIBRARY_PATH` を修正して、コンパイル時、リンク時、実行時に NAG の実行可能プログラムとライブラリを見つけられるようにします。

`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(注: Ubuntu などの一部の Debian 派生の Linux ディストリビューションで利用可能な `Dash` は使用しないでください)、`· 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のサイズを指定します。
- ・ `· {vendor,nag}` は、BLAS と LAPACK に提供された MKL ライブラリに依存する NAG ライブラリのセットを使用するか(オプション `· vendor`)、自己完結型の NAG ライブラリを使用するか(オプション `· nag`)を指定します。最高のパフォーマンスを得るには、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。オプションの `nagvars` スクリプト引数は以下の通りです。

- ・ `· --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 実装のみ**の資料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` について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の `nagvars` スクリプトの使用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通りです。

- ・ `· 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... paramtext 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/または 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 ルーチン(およびオプションでベンダー - BLAS と LAPACK)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 ルーチン(およびオプションでベンダー BLAS と LAPACK)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 ルーチン(およびオプションでベンダー BLAS と LAPACK)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、

```
... verbatim
      ${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、

```
... verbatim
      ${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、

```
... verbatim
      ${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、他のものを指すために 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、新しいバージョンのコンパイラを使用している場合など)。

異なるコンパイラ、あるい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 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かもしれません。これは、次のように LD_LIBRARY_PATH に

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

を追加することで簡単にできます。

このバージョンの NAG ライブラリは、GNU gcc および g++ コンパイラから呼び出すことができます。ただし、少なくとも私たちのウェブサイトの追加情報ページに記載されているバージョンの gcc を使用する必要があります。これを行う最も簡単な方法は、nagvars.sh および/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 固有の環境変数が変更され、GNU gcc および g++(適宜)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 へのリンクでは、NAG ライブラリに存在する OpenMP ルーチンとの整合性を保つために、引き続き Intel コンパイラの OpenMP ランタイムを使用することにも注意してください。

3.1.1 使用するスレッド数の設定

MKL は OpenMP を利用して、ライブラリルーチンの一部にスレッド化を実装しています。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、OpenMP の並列性に複数のレベルが存在する可能性があり、また、アプリケーション内の OpenMP の並列領域から、これらのマルチスレッドルーチン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 個のスレッドを使用します。内側のレベルはアクティブにはならず、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、OMP_NESTED の OpenMP 環境変数を照会および設定することで、有効/無効を選択できます。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、OpenMP の並列性が 2 レベルある場合は合計 N*N 個のスレッドが作成されます。ネスト並列処理をより詳細に制御するには、

環境変数 OMP_NUM_THREADS をカ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、最初の並列レベルに N 個のスレッドが作成され、内側の並列レベルが検出されたときに、外側のレベルの各スレッドに対して P 個の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。通常は 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、アプリケーション内で上位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 を常に明示的に目的の値に設定することをお勧めします。

一般に、使用が推奨される最大スレッド数は、共有メモリシステム上の物理コアの数です。ただし、ほとんどの Intel プロセッサは、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見えるようにする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。この機能を利用するのは有益かもしれませんが、この選択は、使用する特定のアルゴリズムと問題のサイズに依存します。追加の論理コアを利用する場合とそうで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をベンチマークし、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、システムの起動時に BIOS で目的の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 内のルーチン呼び出すため、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 にアドバイスを求めてください。

3.2 Fortran 引用仕様宣言

NAG ライブラリの引用仕様宣言は、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。これらは、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、提供されている Example プログラムを使用する場合は必須です。

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確認できるようにすることです。引用仕様宣言により、コンパイラは以下のことを確認できます。

1. (a) サブルーチンがそのように呼び出されていること。
2. (b) 関数が正しい型で宣言されていること。
3. (c) 正しい数の引数が渡されていること。
4. (d) すべての引数が型と構造において一致していること。

NAG ライブラリの引用仕様宣言ファイルは、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) で提供されます。これらは、各コンパイラ呼び出しで `-lpathname` 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宣言を含むディレクトリのパスです。

.mod モジュールファイルは、**インストーラズノート**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宣言を使用する場合に、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、最初に独自のコンパイラバージョンで引用仕様宣言を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宣言のセットは、提供されたスクリプトコマンド

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

を使用して、別のディレクトリ (例えば `nag_interface_blocks_alt`) で作成できます。このスクリプトは、PATH 環境からの 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 `{int32,int64}` の両方のオプションを使用して、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、以下のコマンドを使用して (実際のディレクトリパスで `[INSTALL_DIR]` を適宜置き換えて)、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ようになります。

現在、ユーザーが AD ルーチンの引用仕様宣言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 にサポートをご依頼ください (連絡先の詳細についてはセクション 7 を参照)。

3.3 Example プログラム

配布されている Example プログラムの実行結果は、Mark 30 でインストーラーズノートのセクション 2.2 に記載されているソフトウェアを使用して生成されました。Example プログラムを少し異なる環境（例えば、異なる C または Fortran コンパイラ、異なるコンパイラの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、これらの Example 実行結果が正確に再現され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、固有ベクトル（スカラー倍、多くの場合-1、時に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、および機械精度と同じ程度の”小さな”量の残差などです。

配布されている Example 実行結果は、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）で得られた結果です。NAG BLAS または LAPACK で Example プログラムを実行すると、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 実行結果は、32 ビット整数の静的ライブラリ libnag_mkl.a と libnag_mkl_ad.a で得られた結果です。

Example プログラムの資料は、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応されていることに注意してください。これにより、プログラムをこの実装で実行できるようになり、それ以上の変更は必要ありません。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 プログラムを可能な限り使用してください。Example プログラムは、[INSTALL_DIR]/scripts ディレクトリにあるスクリプト 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、および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、および-vendor によって変更できます。各段階で実行されているコメントとコマンドの両方の出力を最小限に抑えるには、-quiet オプションを指定できます。使用可能なオプションのリストを表示するには、

```
nag_example -help
```

を実行してください。

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を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
```

と入力すると、現在のディレクトリに 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）がコピーされ、プログラムがコンパイル・リンクされて実行され、Example プログラムの結果がファイル e04ucce.r に出力されます。

同様に、NAG Fortran ルーチンの場合、

```
nag_example e04nrf
```

と入力すると、現在のディレクトリに 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)がコピーされ、プログラムがコンパイル・リンクされて実行され、Example プログラムの結果がファイル e04nrfe.r に出力されます。

NAG AD ルーチンの Example プログラムを実行するには、`-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
```

となります。

AD の Example プログラムのいくつかは、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 プログラム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(連絡先の詳細についてはセクション 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、それらの場所を指定できます。`nagvars` スクリプトの実行時に `DCODIR` が設定されている場合、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 プログラムをコンパイルして実行するか、引数 `a00aaf` または `a00aac` を指定して `nag_example` スクリプトを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 プログラムは、タイトルと製品コード、使用されたコンパイラと精度、マークとメンテナンスレベルを含む実装の詳細を出力します。

3.5 C データ型

この実装では、32 ビット整数と 64 ビット整数の両方のライブラリが含まれています。

32 ビット整数ライブラリ(ディレクトリ`[INSTALL_DIR]/lp64/lib`にある)では、NAG の C データ型 `Integer` と `Pointer` は以下のように定義されています。

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

64 ビット整数ライブラリ(ディレクトリ`[INSTALL_DIR]/ilp64/lib`にある)では、NAG の C データ型 `Integer` と `Pointer` は以下のように定義されています。

NAG 型	C 型	サイズ(バイト)
<code>Integer</code>	<code>long</code>	8
<code>Pointer</code>	<code>void *</code>	8

sizeof(Integer)と sizeof(Pointer)の値は、a00aac の Example プログラムでも示されています。その他の 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)にあります。

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 にある)と 64 ビット整数(ディレクトリ[INSTALL_DIR]/ilp64/lib にある)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、型

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

は、nag_wp が Fortran の KIND パラメータである 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。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せん。nag_wp 型を NAG ライブラリの「作業精度」型と呼んでいます。なぜなら、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だからです。

さらに、少数のルーチンでは、型

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

が使用されます。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は、

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

で、「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、ライブラリに付属の Example プログラムのほとんどを参照してください。

この実装では、これらの型は以下の意味を持ちます。

```
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 ルーチン呼び出す

注意して使用すれば、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ことは、C ルーチンの同等物が利用できないレガシーの Fortran ルーチンにアクセスする場合や、他の言語から使用するのに便利な、初等的な 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場合に望ましいかもしれませんが、ユーザーが Fortran と C の型のマッピングを行うのを支援するために、C の観点から Fortran インターフェイスの説明 (C ヘッダーインターフェイス) が各 Fortran ルーチンのドキュメントに含まれています。C/C++ヘッダーファイル (32 ビット整数の場合は [INSTALL_DIR]/lp64/include/nag.h、64 ビット整数の場合は [INSTALL_DIR]/ilp64/include/nag.h) も提供されています。この方法で NAG Fortran ルーチンを使用したい場合は、適切なヘッダーファイルを #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++ から呼び出す際のアドバイスを記載したドキュメント `alt_c_interfaces.html` も利用できます。(NAG ライブラリの以前のマークでは、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 の 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、BLAS Technical Forum (BLAST) ルーチンの宣言が含まれています。ユーザーは、提供された Intel MKL などの他のライブラリに関連する C インクルードファイルからこれらの定義を取得することを好むかもしれませんが、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++ のコンパイル文に追加することで、NAG によるこれらのルーチンの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 1 つ以上のルーチンに適用されるその他の情報は、以下に章ごとに示します。

1. (a) F06、F07、F08、F16

F06、F07、F08、F16 の各章では、BLAS および LAPACK 由来の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、アプリケーションでは、NAG スタイルの名前ではなく、BLAS/LAPACK 名を使用してルーチンを参照する必要があります。

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、供給す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 リファレンス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、非自己完結型の NAG ライブラリにおける BLAS および LAPACK ルーチンの呼び出しは、以下のルーチンを除いて、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通りです。

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 $\text{abs}(x) < 2.23e-308$

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 $x \geq 2.53e+307$

ifail = 2 (NW_REAL) if $4.74e+7 \leq x < 2.53e+307$

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 $0 < x \leq 2.23e-308$
 s17ae[fc] (nag[f].specfun_bessel_j0_real)
 ifail = 1 (NE_REAL_ARG_GT) if $\text{abs}(x) > 1.0e+16$
 s17af[fc] (nag[f].specfun_bessel_j1_real)
 ifail = 1 (NE_REAL_ARG_GT) if $\text{abs}(x) > 1.0e+16$
 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 $\text{abs}(z) < 3.92223e-305$
 ifail = 4 (NW_SOME_PRECISION_LOSS) if $\text{abs}(z)$ or $\text{fnu}+n-1 > 3.27679e+4$
 ifail = 5 (NE_TOTAL_PRECISION_LOSS) if $\text{abs}(z)$ or $\text{fnu}+n-1 > 1.07374e+9$
 s17de[fc] (nag[f].specfun_bessel_j_complex)
 ifail = 2 (NE_OVERFLOW_LIKELY) if $\text{AIMAG}(z) > 7.00921e+2$
 ifail = 3 (NW_SOME_PRECISION_LOSS) if $\text{abs}(z)$ or $\text{fnu}+n-1 > 3.27679e+4$
 ifail = 4 (NE_TOTAL_PRECISION_LOSS) if $\text{abs}(z)$ or $\text{fnu}+n-1 > 1.07374e+9$
 s17dg[fc] (nag[f].specfun_airy_ai_complex)
 ifail = 3 (NW_SOME_PRECISION_LOSS) if $\text{abs}(z) > 1.02399e+3$
 ifail = 4 (NE_TOTAL_PRECISION_LOSS) if $\text{abs}(z) > 1.04857e+6$
 s17dh[fc] (nag[f].specfun_airy_bi_complex)
 ifail = 3 (NW_SOME_PRECISION_LOSS) if $\text{abs}(z) > 1.02399e+3$
 ifail = 4 (NE_TOTAL_PRECISION_LOSS) if $\text{abs}(z) > 1.04857e+6$
 s17dl[fc] (nag[f].specfun_hankel_complex)
 ifail = 2 (NE_OVERFLOW_LIKELY) if $\text{abs}(z) < 3.92223e-305$
 ifail = 4 (NW_SOME_PRECISION_LOSS) if $\text{abs}(z)$ or $\text{fnu}+n-1 > 3.27679e+4$
 ifail = 5 (NE_TOTAL_PRECISION_LOSS) if $\text{abs}(z)$ or $\text{fnu}+n-1 > 1.07374e+9$

s18ad[fc] (nag[f].specfun_bessel_k1_real)
 ifail = 2 (NE_REAL_ARG_TOO_SMALL) if $0 < x \leq 2.23e-308$
 s18ae[fc] (nag[f].specfun_bessel_i0_real)
 ifail = 1 (NE_REAL_ARG_GT) if $\text{abs}(x) > 7.116e+2$
 s18af[fc] (nag[f].specfun_bessel_i1_real)
 ifail = 1 (NE_REAL_ARG_GT) if $\text{abs}(x) > 7.116e+2$
 s18dc[fc] (nag[f].specfun_bessel_k_complex)
 ifail = 2 (NE_OVERFLOW_LIKELY) if $\text{abs}(z) < 3.92223e-305$
 ifail = 4 (NW_SOME_PRECISION_LOSS) if $\text{abs}(z)$ or $\text{fnu}+n-1 > 3.27679e+4$

ifail = 5 (NE_TOTAL_PRECISION_LOSS) if $\text{abs}(z)$ or $\text{fnu}+n-1 > 1.07374\text{e}+9$
s18de[fc] (nag[f].specfun_bessel_i_complex)
ifail = 2 (NE_OVERFLOW_LIKELY) if $\text{REAL}(z) > 7.00921\text{e}+2$
ifail = 3 (NW_SOME_PRECISION_LOSS) if $\text{abs}(z)$ or $\text{fnu}+n-1 > 3.27679\text{e}+4$
ifail = 4 (NE_TOTAL_PRECISION_LOSS) if $\text{abs}(z)$ or $\text{fnu}+n-1 > 1.07374\text{e}+9$

s19aa[fc] (nag[f].specfun_kelvin_ber)
ifail = 1 (NE_REAL_ARG_GT) if $\text{abs}(x) \geq 5.04818\text{e}+1$
s19ab[fc] (nag[f].specfun_kelvin_bei)
ifail = 1 (NE_REAL_ARG_GT) if $\text{abs}(x) \geq 5.04818\text{e}+1$
s19ac[fc] (nag[f].specfun_kelvin_ker)
ifail = 1 (NE_REAL_ARG_GT) if $x > 9.9726\text{e}+2$
s19ad[fc] (nag[f].specfun_kelvin_kei)
ifail = 1 (NE_REAL_ARG_GT) if $x > 9.9726\text{e}+2$

s21bc[fc] (nag[f].specfun_ellipint_symm_2)
ifail = 3 (NE_REAL_ARG_LT) if an argument $< 1.583\text{e}-205$
ifail = 4 (NE_REAL_ARG_GE) if an argument $\geq 3.765\text{e}+202$
s21bd[fc] (nag[f].specfun_ellipint_symm_3)
ifail = 3 (NE_REAL_ARG_LT) if an argument $< 2.813\text{e}-103$
ifail = 4 (NE_REAL_ARG_GT) if an argument $\geq 1.407\text{e}+102$

3. (c)X01

数学定数の値は以下の通りです。

x01aa[fc] (nag[f].math_pi)
= 3.1415926535897932
x01ab[fc] (nag[f].math_euler)
= 0.5772156649015328

4. (d)X02

マシン定数の値は以下の通りです。

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場合、エラーメッセージと警告メッセージのデフォルトの出力ユニットはいずれも Fortran ユニット 6 です。

6. (f)X06

X06 の章のルーチンは、NAG ライブラリのこの実装において 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(NLL6I30DNL)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 連絡先

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)