NAG Library, Mark 30

NLW6I30DEL - Licence Managed

Microsoft Windows x64, 64-bit, Intel Classic C/C++ or Microsoft C/C++ or Intel Classic Fortran, 32-bit integers, VS2019

ユーザーノート

内容

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「NLW6I30DEL」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「実装毎のユーザーノート」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 は、ライブラリルーチンを呼び出す前に、Library Manual 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

以下の URL をご確認ください。

https://support.nag.com/doc/inun/nl30/w6idel/supplementary.html

この実装の適用性や使用法に関連する新しい情報の詳細については、こちらをご覧ください。

3 一般情報

この NAG ライブラリの実装では、Basic Linear Algebra Subprograms (BLAS)と 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 Windows 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も提供します （自己完結型ライブラリと呼びます）。この実装は、本製品に付属している 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）をご覧ください。

最高のパフォーマンスを得るためには、nag_nag_MT.lib、nag_nag_MD.lib、 NLW6I30DE_nag.lib、NLW6I30DE_nag.dll などの自己完結型の NAG ライブラリを 使用するよりも、nag_mkl_MT.lib、nag_mkl_MD.lib、 NLW6I30DE_mkl.lib、NLW6I30DE_mkl.dll など、提供されている MKL ベンダー ライブラリに基づく NAG ライブラリのバージョンを使用することをお勧めします。

どの静的バージョンの NAG ライブラリを使用すべきかは、Microsoft ランタイムライブラリへのリンク方法にも依存します。例えば、マルチスレッド静的ランタイムライブラリとリンクする場合は、nag_mkl_MT.lib または nag_nag_MT.lib を使用し、マルチスレッド DLL ランタイム ライブラリとリンクする場合は、nag_mkl_MD.lib または nag_nag_MD.lib を使用 する必要があります。あるいは、NAG ライブラリの DLL バージョンを呼び出す場合は、インポート ライブラリの NLW6I30DE_mkl.lib または NLW6I30DE_nag.lib とリンクする必要が あります（実行時には、対応する DLL である NLW6I30DE_mkl.dll または NLW6I30DE_nag.dll がパスに存在することを確認してください）。詳細については、 セクション 3.1.1 を参照してください。

NAG AD ライブラリも、nag_nag_ad_MT.lib、nag_nag_ad_MD.lib、 nag_mkl_ad_MT.lib、nag_mkl_ad_MD.lib などのバージョンのアドオンライブラリ として含まれています。nag_mkl_ad バージョンでは、いくつかのコアレベル 3 BLAS ルーチンの微分を 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ています。

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ユーザーが NAG_FREE()を呼び出すことに よって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ランタイムやその他のライブラリの使用に依存しており、これらはメモリリークを起こす 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 ライブラリを呼び出すたびに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予定の場合は、 *CL Interface Multithreading* または *FL Interface Multithreading* (適切な方)のドキュメント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-windows-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、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、これが当てはまら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 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 3.1.0 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-windows-developer-guide/top/obtaining-numerically-reproducible-results/reproducibility-conditions.html を参照)に従っている場合、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 の上に構築された 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 保証することはできない可能性があります。ビット単位の再現性に関するより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、ライブラリがデフォルトのフォルダ、つまり

C:¥Program Files¥NAG¥NL30¥nlw6i30del

にインストールされていると想定しています。

実際の"Program Files"フォルダの名前は、ロケールによって異なる場合があります。上記のフォルダが 存在しない場合は、システム管理者(またはインストールを行った人)に相談してください。以下のサブセクションでは、このフォルダを *install_dir* と呼びます。

また、ライブラリコマンドプロンプトのショートカットが、スタートメニューまたはすべてのアプリの NAG Library (NLW6I30DEL)セクションにあることも想定しています。

NAG NLW6I30DEL Command Prompt

このショートカットが存在しない場合は、システム管理者(またはインストールを行った人)に相談してください。(ライブラリのインストール手順の一部として作成された他のショートカットも、この 場所にあると想定しています。)

ライブラリの DLL 形式を使用している場合(セクション 3.1.1 を参照)、NAG DLL (NLW6I30DE_mkl.dll または NLW6I30DE_nag.dll)を実行時にアクセスできる ようにする必要があります。そのため、*install_dir*¥bin フォルダをパスに含める必要が あります。また、*install_dir*¥rtl¥bin フォルダもパスに含める必要があります(適切な Intel ランタイムライブラリがパスに含まれていない場合)。MKL ベースのバージョンのライブラリを 使用する場合は、*install_dir*¥mkl¥bin フォルダもパスに含める必要がありますが、 *install_dir*¥bin フォルダよりも後にパスに含める必要があります。これは、ベンダー バージョンで問題が発生するのを避けるため、いくつかの BLAS/LAPACK ルーチンの NAG バージョンが NAG ライブラリに含まれている可能性があるためです(詳細はセクション 4 を 参照)。

NAG DLL のアクセス可能性を確認するには、スタートメニューまたはすべてのアプリの ショートカットから利用可能なプログラム NAG_Library_DLL_info.exe を実行してください。

Check NAG NLW6I30DEL DLL Accessibility

このユーティリティの詳細については、**インストーラーノート**のセクション 4.2.2 を参照してください。

3.1.0 使用するスレッド数の設定

この NAG ライブラリの実装と MKL では、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 使用しています。実行時に使用されるスレッドの数は、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。

コマンドウィンドウで環境変数を設定できます。

```
set OMP_NUM_THREADS=N
```

ここで、N は必要なスレッド数です。環境変数は、Windows のコントロールパネルを通じて 通常の方法で設定することもできます。環境変数 OMP_NUM_THREADS は、必要に応じて プログラムの実行ごとに再設定できます。

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に なりません。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、OMP_NESTED 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、上位レベルの各スレッドに対して各並列領域で N スレッドが作成されます。つまり、OpenMP 並列処理の レベルが 2 つある場合は合計 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。例えば、

```
set OMP_NUM_THREADS=N,P
```

これにより、並列処理の最初のレベルに対して 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、外側のレベルの各スレッドに対して P スレッドが作成 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や ベンダーライブラリによって異なりますが、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 は常に目的の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、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でき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と有益な場合もありますが、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。パフォーマンスが重要なアプリケーションでは、 追加の論理コアを使用する場合と使用しない場合の両方でベンチマークを行い、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するための追加の環境変数が 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-windows/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すため、MKL の環境変数も NAG ライブラリの動作に間接的に影響を与える可能性があります。MKL の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。

3.1.1 コマンドウィンドウから

コマンドウィンドウからこの実装にアクセスするには、いくつかの環境変数を設定する必要があります。

ショートカット：

NAG NLW6I30DEL Command Prompt

を使用して、ライブラリと提供された MKL の INCLUDE、LIB、PATH 環境 変数の正しい設定を使用してコマンドプロンプトウィンドウを起動できます。 nag_example_*.bat バッチファイルで必要とされる環境変数 NAG_NLW6I30DEL も 設定されます。

ショートカットを使用しない場合は、この実装用のバッチファイル envvars.bat を実行して 環境変数を設定できます。このファイルのデフォルトの場所は次のとおりです。

```
C:¥Program Files¥NAG¥NL30¥nlw6i30del¥batch¥envvars.bat
```

このファイルがデフォルトの場所にない場合は、nlw6i30del を含むファイル envvars.bat を検索して見つけることができます。

その後、次のいずれかのコマンドを使用して、コマンドラインで NAG ライブラリをコンパイルおよび リンクできます。

```
cl /MD driver.c NLW6I30DE_mkl.lib
ifort  /MD driver.f90 NLW6I30DE_mkl.lib

cl /MD driver.c NLW6I30DE_nag.lib
ifort  /MD driver.f90 NLW6I30DE_nag.lib

cl /MT driver.c nag_mkl_MT.lib mkl_intel_lp64.lib mkl_intel_thread.lib
   mkl_core.lib libiomp5md.lib user32.lib
   /link /nodefaultlib:ifconsol.lib /nodefaultlib:ifmodintr.lib /nodefaultlib:libifcoremt.lib
         /nodefaultlib:libifport.lib /nodefaultlib:ifwin.lib
ifort  /MT driver.f90 nag_mkl_MT.lib mkl_intel_lp64.lib mkl_intel_thread.lib
   mkl_core.lib libiomp5md.lib user32.lib

cl /MT driver.c nag_nag_MT.lib user32.lib
   /link /nodefaultlib:ifconsol.lib /nodefaultlib:ifmodintr.lib /nodefaultlib:libifcoremt.lib
         /nodefaultlib:libifport.lib /nodefaultlib:ifwin.lib
ifort  /MT driver.f90 nag_nag_MT.lib user32.lib

cl /MD driver.c nag_mkl_MD.lib mkl_intel_lp64.lib mkl_intel_thread.lib
   mkl_core.lib libiomp5md.lib user32.lib
   /link /nodefaultlib:ifconsol.lib /nodefaultlib:ifmodintr.lib /nodefaultlib:libifcoremd.lib
         /nodefaultlib:libifportmd.lib /nodefaultlib:ifwin.lib
ifort  /MD driver.f90 nag_mkl_MD.lib mkl_intel_lp64.lib mkl_intel_thread.lib
   mkl_core.lib libiomp5md.lib user32.lib

cl /MD driver.c nag_nag_MD.lib user32.lib
   /link /nodefaultlib:ifconsol.lib /nodefaultlib:ifmodintr.lib /nodefaultlib:libifcoremd.lib
         /nodefaultlib:libifportmd.lib /nodefaultlib:ifwin.lib
ifort  /MD driver.f90 nag_nag_MD.lib user32.lib
```

ここで、driver.c または driver.f90 はアプリケーションプログラムです。（注：上記では Microsoft C コンパイラ cl を使用していると想定しています。Intel C コンパイラ icl を使用することもできます。両方のコンパイラのオプションは同じです。）

上記の指示は、NAG AD ライブラリを使用せずに NAG ライブラリを使用する方法を示しています。NAG AD ライブラリを追加するには、名前付きの NAG ライブラリの後に、以下のライブラリとリンカー オプション セットのうちの 1 つだけを使用して、それらのライブラリの適切なバージョンを追加する必要があります。

- ·nag_nag_ad_MT.lib nag_nag_MT.lib user32.lib¥ /link /nodefaultlib:ifconsol.lib /nodefaultlib:ifmodintr.lib /nodefaultlib:libifcoremt.lib¥ /nodefaultlib:libifport.lib /nodefaultlib:ifwin.lib
- ·nag_nag_ad_MD.lib nag_nag_MD.lib user32.lib¥ /link /nodefaultlib:ifconsol.lib /nodefaultlib:ifmodintr.lib /nodefaultlib:libifcoremd.lib¥ /nodefaultlib:libifportmd.lib /nodefaultlib:ifwin.lib

Fortran ユーザーのみ：上記のライブラリを Fortran から使用する場合は、libnag_dcof_MT.lib または libnag_dcof_MD.lib（適切な方）も追加でリンクする 必要があります。

コンパイラ/リンカーオプション：

/MD
コンパイラランタイムライブラリのマルチスレッド DLL バージョン用のインポートライブラリと リンクするようにコードをコンパイルすることを意味します。

c_header の例をコンパイル/リンクする場合は、このオプションを指定する必要があることに注意 してください。

/MT
コンパイラランタイムライブラリの静的マルチスレッドバージョンとリンクするようにコードを コンパイルすることを意味します。

/nodefaultlib:
リンカーに、不足している（ここでは不要な）ランタイムライブラリについて警告しないように 指示します。

これらのオプションは、プロジェクト全体で一貫して使用する必要があります。

NLW6I30DE_mkl.lib は、BLAS/LAPACK ルーチンに MKL を使用する DLL インポートライブラリ です。NLW6I30DE_nag.lib は、NAG BLAS/LAPACK を含む DLL インポートライブラリです。 両方のライブラリは/MD オプションでコンパイルされています。このオプションは、 そのようなライブラリとリンクするアプリケーションをコンパイルする際に、正しいコンパイラ ランタイムライブラリとのリンクを確実にするために使用する必要があります。

nag_mkl_MT.lib は、BLAS/LAPACK を含まない静的ライブラリであり、MKL 静的ライブラリと リンクする必要があります。nag_nag_MT.lib は、NAG BLAS/LAPACK を含む静的ライブラリ です。両方のライブラリは/MT オプションでコンパイルされています。このオプションは、 そのようなライブラリとリンクするアプリケーションをコンパイルする際に、正しいコンパイラ ランタイムライブラリとのリンクを確実にするために使用する必要があります。

nag_mkl_MD.lib は、BLAS/LAPACK を含まない静的ライブラリであり、MKL 静的ライブラリとリンクする必要があります。nag_nag_MD.lib は、 NAG BLAS/LAPACK を含む静的ライブラリです。両方のライブラリは/MD オプションでコンパイル されています。このオプションは、そのようなライブラリとリンクするアプリケーションをコンパイル する際に、正しいコンパイラランタイムライブラリとのリンクを確実にするために使用する必要が あります。

3.1.2 Microsoft Visual Studio から

以下の指示は、Microsoft Visual Studio 2019 に適用されます。異なるバージョンの Visual Studio を使用している場合、手順が若干異なる場合があります。

Microsoft Visual Studio を使用して NAG ライブラリを使用するプログラムをビルドする予定の場合、各ユーザーは適切なオプションを設定する必要があります。

Visual Studio を起動し、通常の方法でプロジェクトを作成します。プロジェクトが NAG ライブラリを 使用することを想定しています。

ライブラリは完全に最適化されたモードで実行することを目的としているため、警告メッセージを 回避するために、アクティブな構成を **Release** に設定することを決定する場合があります。 Visual Studio を開いたら、ツールバーから、または**ビルド¥|構成マネージャー**メニューから これを行うことができます。**デバッグ**モードで作業する場合、競合するランタイムライブラリに 関する警告メッセージが表示される可能性があることに注意してください。

プラットフォームが **x64** に設定されていることを確認してください（この 64 ビット実装の NAG ライブラリとの互換性を確保するため）。これは、プロパティページの**構成マネージャー...** ボタンから変更できます。

以下の手順は、NAG ライブラリをプロジェクトに追加する方法を示しています。

1. 1.プロジェクトのプロパティページを開きます。これを行う方法はいくつかあります。
   - ·ソリューションエクスプローラーウィンドウが開いている場合は、グループ プロジェクト（最初の行）が選択されていないことを確認します。**プロジェクト**メニューから、**プロパティ**（または***プロジェクトプロパティ***）項目を選択します。
   - ·または、ソリューションエクスプローラーの特定の単一のプロジェクトを右クリック して、**プロパティ**を選択します。
   - ·**プロパティ**情報は、ツールバーからもアクセスできます。ソリューション エクスプローラーでプロジェクトを選択し、ツールバーの**プロパティウィンドウ**ボタンを 選択します。表示されたウィンドウで、右端の**プロパティページ**アイコンを選択します。
2. 2.さまざまなフォルダの場所を構成する必要があります。フォームから、**構成プロパティ** をクリック/展開します。

   プロジェクトが Microsoft または Intel C または C++プロジェクトの場合：

   - ·左側のパネルで **VC++ディレクトリ**をクリック/展開します。次に、
     - ◦**インクルードディレクトリ**を選択し、*install_dir*¥include フォルダを追加します。
     - ◦**ライブラリディレクトリ**を選択し、*install_dir*¥lib フォルダと *install_dir*¥rtl¥lib フォルダ（および必要に応じて *install_dir*¥mkl¥lib フォルダ）を追加します。（または、以下の Fortran プロジェクトの指示にあるように、これらを**追加のライブラリディレクトリ**設定で 指定することもできます。）

   プロジェクトが Intel Fortran プロジェクトの場合：

   - ·左側のパネルで **Fortran** をクリック/展開し、**全般**を選択します。次に、

- ◦**追加のインクルードディレクトリ**を選択し、 *install_dir*¥nag_interface_blocks フォルダのフルネームを追加します。

- ·左側のパネルで**リンカー**をクリック/展開し、**全般**を選択します。次に、
  - ◦**追加のライブラリディレクトリ**を選択し、*install_dir*¥lib フォルダと *install_dir*¥rtl¥lib フォルダ（および必要に応じて *install_dir*¥mkl¥lib フォルダ）を追加します。

デフォルトのフォルダは次のとおりです。

C/C++プロジェクトのインクルードディレクトリ
C:¥Program Files¥NAG¥NL30¥nlw6i30del¥include

Fortran プロジェクトの追加のインクルードディレクトリ
C:¥Program Files¥NAG¥NL30¥nlw6i30del¥nag_interface_blocks

C/C++または Fortran プロジェクトの[追加の]ライブラリディレクトリ
C:¥Program Files¥NAG¥NL30¥nlw6i30del¥lib
C:¥Program Files¥NAG¥NL30¥nlw6i30del¥rtl¥lib
C:¥Program Files¥NAG¥NL30¥nlw6i30del¥mkl¥lib

変更を受け入れるには**適用**ボタンをクリックするか、変更を受け入れてフォームを閉じるには **OK** ボタンをクリックします。

3. 3.NAG ライブラリと Intel ランタイムライブラリ（および MKL ライブラリ）をリンカー オプションで指定する必要があります。プロパティページフォームから、左側のパネルの **構成プロパティ** の下にある**リンカー**をクリック/展開し、**入力**を選択して、適切な ライブラリファイルを**追加の依存ファイル**リストに追加します。下の表を参照してください。

   変更を受け入れるには**適用**ボタンをクリックするか、変更を受け入れてフォームを閉じるには **OK** ボタンをクリックします。

4. 4.さらに、適切なランタイムライブラリオプションを設定する必要があります。これは、リンクする NAG ライブラリのバージョンと一致する必要があります。

   プロジェクトが Microsoft または Intel C または C++プロジェクトの場合：

   - ·最初に、NAG のサンプルプログラムなどのソースファイルをプロジェクトに追加 します。**プロジェクト**メニューの**既存項目の追加...**を使用します。（プロジェクトに C または C++ファイルがない場合、C++オプションが表示されない場合があります。）
   - ·プロパティページを再度開き（上記の詳細を参照）、**構成プロパティ**（必要な場合）をクリック/展開してから、**C/C++**をクリックし、左側のパネルで **コード生成**をクリックします。次に、右側のパネルで**ランタイムライブラリ**を選択し、適切なバージョンに変更します。例えば、プロジェクトで nag_nag_MT.lib または nag_mkl_MT.lib のいずれかのライブラリを使用する場合は**マルチスレッド(/MT)** です。プロジェクトで他の NAG ライブラリのいずれかを使用する場合は、**マルチスレッド DLL (/MD)**を選択する必要があります。

- ·正しいランタイムライブラリを選択したら、**適用**ボタンをクリックして変更を 受け入れるか、**OK** ボタンをクリックして変更を受け入れてフォームを閉じます。

プロジェクトが Intel Fortran プロジェクトの場合：

- ·プロパティフォームから、左側のパネルで **Fortran** をクリック/展開し、**ライブラリ**を選択します。右側のパネルに**ランタイムライブラリ**エントリが表示 されるので、プロジェクトで nag_nag_MT.lib または nag_mkl_MT.lib の いずれかのライブラリを使用する場合は**マルチスレッド**を選択する必要があります。 プロジェクトで他の NAG ライブラリのいずれかを使用する場合は、**マルチスレッド DLL** を 選択する必要があります。
- ·正しいランタイムライブラリを選択したら、**適用**ボタンをクリックして変更を 受け入れるか、**OK** ボタンをクリックして変更を受け入れてフォームを閉じます。

5. 5. *Microsoft C または C++プロジェクトの場合*、NAG ライブラリの静的バージョン、つまり nag_mkl_MT.lib または nag_mkl_MD.lib ではなく DLL にリンクする場合は、プロジェクト設定を変更して、リンカーにいくつかのランタイムライブラリを無視するように指示する 必要もあります。再度**構成プロパティ**で、左側のパネルの**リンカー**セクションをクリック/ 展開し、**入力**をクリックします。右側のパネルで、**特定のデフォルトライブラリを無視**を 選択し、**編集**を選択して、セクション 3.1.1 のコマンドに /nodefaultlib:として示されているライブラリのリストを追加します。/MD ビルドまたは/MT ビルドに適したセットを選択します（セットは似ていますが、同一ではありません）。ライブラリ名はセミコロンで区切ります。

   変更を受け入れてフォームを閉じるには、**OK** ボタンをクリックします。

これで、プロジェクトは**ビルド**メニューから適切な選択肢を使用してコンパイルおよびリンクできる はずです。

Microsoft 開発環境内からプログラムを実行するには、**デバッグ**メニュー（例えば、**デバッグなしで 開始**（Ctrl+F5）を選択）からプログラムを実行できます。PATH 環境変数を上記の セクション 3.1.1 で詳しく説明したように適切に設定する必要があることに注意して ください。

データファイルを標準入力に接続する必要がある場合、またはプログラムの出力を標準出力にリダイレクト する必要がある場合は、プロパティフォームの**デバッグ**セクションを選択し、**コマンド引数** フィールドに適切なコマンドを挿入することで実現できます。例えば、

< input_file > output_file

入力ファイルと出力ファイルがアプリケーションの作業ディレクトリにない場合は、完全パスまたは 相対パスを指定する必要がある場合があります。.opt ファイルを使用する NAG の例では、これを作業ディレクトリに配置する必要があります。このディレクトリは、プロパティフォームの **デバッグ**ページにある**作業ディレクトリ**フィールドで設定できます。

3.1.3 Fortran モジュールファイルについての Note

nag_interface_blocks フォルダにあるこの NAG ライブラリ実装に付属の Fortran .mod モジュールファイルは、Intel ifort コンパイラでコンパイルされています。このようなモジュール ファイルはコンパイラに依存しており、他の Fortran コンパイラでの使用には適していません。他のコンパイラ を使用する場合に NAG のサンプルプログラムを使用したい場合や、独自のプログラムでインターフェイス ブロック

を使用したい場合は、最初に独自のモジュールファイルを作成する必要があります。詳細についてはセクション 3.2 を参照してください。

3.1.4 NAG Fortran Builder から

検討すべき 3 つのケースがあります。

- ·Fortran Builder には、いくつかのバージョンの NAG ライブラリに関する組み込みの知識があります。所有している Fortran Builder のバージョンによっては、このライブラリ NLW6I30DEL について 認識している可能性があります。その場合、ライブラリがインストールされていることを自動的に検出し、 Fortran Builder IDE で新しい"NAG ライブラリプロジェクト"を作成すると、それを使用するはずです。 これは、NLW6I30DEL を使用するためにライブラリやインターフェイスブロックの場所を IDE に指示する 必要がないため、Fortran Builder から NLW6I30DEL を使用する最も簡単な方法です。NAG ライブラリ プロジェクトを作成する方法については、Fortran Builder のドキュメントを参照してください。

- ·Fortran Builder のバージョンがこのライブラリ NLW6I30DEL がリリースされる*前*に リリースされた場合、この組み込みの知識がない可能性があります。ただし、以下のような手順に従って、IDE 内から新しいライブラリを使用することは可能なはずです。

  - ◦新しいコンソールプロジェクトを作成します。
  - ◦プロジェクトメニューからプロジェクト設定に移動します。
  - ◦基本設定タブで、ビットモードが 64 ビットに設定されていることを確認します。
  - ◦ディレクトリタブをクリックし、インクルードタブをクリックします。
  - ◦インクルードディレクトリ *install_dir*¥nag_interface_blocks_nagfor を追加します（スペースを含む可能性があっても、ディレクトリ名を引用符で囲まないでください）。
  - ◦リンクタブをクリックします。
  - ◦リンクライブラリを追加します。例えば、 *install_dir*¥bin¥NLW6I30DE_nag.dll（関連するインポートライブラリではなく、 DLL 自体にリンクすることが重要です。また、静的ライブラリではなく、DLL にのみリンクできます）。
  - ◦OK をクリックして変更を受け入れます。
  - ◦通常の方法でプロジェクトをビルドし、プログラムを実行します。

  デバッグモード（デフォルト）でプロジェクトをビルドすると、プロジェクト設定の Fortran コンパイラ/ ランタイムチェックタブからアクセスできる*未定義変数*オプションを使用できないことに注意してください。 これは、NAG ライブラリがこのオプションでコンパイルされていない*ない*ためです。これを使用しようと すると、NAG インターフェイスブロックを使用するときに、"互換性のないオプション設定"を示す Fortran Builder のコンパイル時エラーが発生します。

- ·最後に、NAG Fortran Builder に付属の Fortran コンパイラ nagfor のコマンドライン バージョンを使用して、NAG ライブラリの DLL バージョンにリンクすることも可能です。Fortran Builder のバージョンで使用するインターフェイスブロックは、フォルダ *install_dir*¥nag_interface_blocks_nagfor に提供されています。NAG コンパイラの異なる バージョンがある場合は、セクション 3.2 で説明されているように、最初に モジュールファイルを再コンパイルする必要がある場合があります。

ここでも、関連するインポートライブラリではなく、DLL 自体にリンクする必要があることに注意することが 重要です。

Windows コマンドプロンプトから、まずセクション 3.1.1 で説明したように、 PATH 環境変数が正しく設定されていることを確認してください。

その後、以下のいずれかのコマンドを使用して、コマンドラインで NAG ライブラリをコンパイルおよび リンクできます。

```
nagfor -ieee=full -I"install_dir¥nag_interface_blocks_nagfor" driver.f90
      "install_dir¥bin¥NLW6I30DE_mkl.dll" -o driver.exe
nagfor -ieee=full -I"install_dir¥nag_interface_blocks_nagfor" driver.f90
      "install_dir¥bin¥NLW6I30DE_nag.dll" -o driver.exe
```

MKL サポートバージョンのライブラリにリンクするか、すべて NAG バージョンのライブラリにリンクするかに よって異なります。

NLW6I30DE_mkl または NLW6I30DE_nag ライブラリファイルの完全パス名を指定する 必要があり、スペースを含む場合は引用符で囲む必要があります。

3.1.5 その他の環境から

上記以外の環境から NAG ライブラリを呼び出す方法に関する情報は、以下の追加情報ページで入手できる 場合があります。

https://support.nag.com/doc/inun/nl30/w6idel/supplementary.html

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、提供されているサンプル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で一致していること。

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 Fortran コンパイラ ifort で使用するためにコンパイル済みの形式 （.mod ファイル）で提供されています。

ライブラリコマンドプロンプトのショートカットを使用するか、この実装用のバッチファイル envvars.bat を実行して環境変数を設定し（セクション 3.1.1 を参照）、Intel ifort コンパイラを使用する場合、セクション 3.1.1 で説明した コマンドのいずれかを使用してこれらのモジュールにアクセスできます。環境変数 INCLUDE が 設定されるためです。

.mod モジュールファイルは、**インストーラーノート**のセクション 2.2{.sec} 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、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に NAG のサンプルプログラムを 使用したい場合や、独自のプログラムで引用仕様宣言を使用したい場合は、ここで説明するように、最初に 独自のモジュールファイルを作成する必要があります。

任意の場所に nag_interface_blocks_original という名前のフォルダを作成し（正確なフォルダ名は 重要ではありません）、nag_interface_blocks の内容を nag_interface_blocks_original に コピーして、元の引用仕様宣言のセットを保存します。

次に、フォルダ nag_interface_blocks で、使用しているコンパイラを使用して、すべての .f90 ファイルをオブジェクトに再コンパイルします。引用仕様宣言にはいくつかの相互依存関係が 含まれているため、コンパイルの順序は重要ですが、FCOMP が Fortran コンパイラの名前である として、以下のコンパイル順序が機能するはずです。

```
FCOMP -c nag_precisions.f90
FCOMP -c nag_a_ib.f90
FCOMP -c nag_blast_ib.f90
FCOMP -c nag_blas_consts.f90
FCOMP -c nag_blas_ib.f90
FCOMP -c nag_c_ib.f90
FCOMP -c nag_d_ib.f90
FCOMP -c nag_e_ib.f90
FCOMP -c nag_f_ib.f90
FCOMP -c nag_g_ib.f90
FCOMP -c nag_h_ib.f90
FCOMP -c nag_lapack_ib.f90
FCOMP -c nag_m_ib.f90
FCOMP -c nag_s_ib.f90
FCOMP -c nag_x_ib.f90
FCOMP -c nag_long_names.f90
FCOMP -c nag_library.f90
```

コンパイルによって生成されたオブジェクトファイルは破棄できます。モジュールファイルのみが必要です。

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した。

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、サポートについて 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 を参照してください）。

3.3 Example プログラム

配布されたサンプル結果は、**インストーラーノート**のセクション 2.2{.sec}に 記載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。これらのサンプル結果は、サンプル プログラムをわずかに異なる環境（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。そのような違いに 最も敏感な結果は、固有ベクトル（スカラー倍数、多くの場合−1、時には複素数で異なる可能性が あります）、反復回数と関数評価の数、および機械精度と同じ程度の「小さな」量の残差などです。

配布されたサンプル結果は、静的ライブラリ nag_mkl_MD.lib（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 でサンプルを実行すると、結果が若干異なる 場合があります。

配布された NAG AD サンプル結果は、静的ライブラリ nag_mkl_MD.lib と nag_nag_ad_MD.lib で得られた結果です。

サンプル資料は、必要に応じて、Library Manual に掲載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、プログラムはこの実装で実行するのに適しており、これ以上の変更は必要ありません。 可能な限り、Library Manual のバージョンではなく、配布されたサンプルプログラムを使用してください。

サンプルプログラムは、*install_dir*¥batch フォルダにあるバッチファイル nag_example_DLL.bat、nag_example_static_MT.bat、 nag_example_static_MD.bat を使用すると最も簡単にアクセスできます。

これらのバッチファイルでは、C/C++または Fortran コンパイラと NAG ライブラリの環境変数が設定されている 必要があります。特に、環境変数 NAG_NLW6I30DEL を NAG ライブラリの場所に設定する必要が あります。詳細については、セクション 3.1.1 を参照してください。

上記の nag_example_*.bat バッチファイルのそれぞれは、サンプルプログラム（およびそのデータ 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とリンクし（独自バージョンのプログラムを再コンパイルできるようにコンパイルコマンドを表示）、最後に実行可能プログラムを実行します（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 使用して）。結果はファイルとコマンドウィンドウに送信されます。

C と Fortran の両方のサンプルプログラムが提供されています。

対象のサンプルプログラムは、コマンドの引数で指定します。例えば、

```
nag_example_DLL e04ucc
nag_example_DLL e04ucf
```

これにより、サンプルプログラムとそのデータおよびオプションファイル（C の場合は e04ucce.c、e04ucce.d、e04ucce.opt、Fortran の場合は e04ucfe.f90 と e04ucfe.d）が現在のフォルダにコピーされ、

プログラムが コンパイルおよびリンクされて実行され、サンプルプログラムの結果が C の場合は e04ucce.r ファイル、Fortran の場合は e04ucfe.r ファイルに生成されます。

nag_example_DLL.bat は、NAG BLAS/LAPACK を使用して NAG ライブラリの DLL バージョンにリンクします。

DLL の MKL バージョンにリンクするには、-mkl オプションを使用します。例えば、

```
nag_example_DLL -mkl e04ucc
nag_example_DLL -mkl e04ucf
```

nag_example_static_MD.bat バッチファイルも同じように使用され、/MD で コンパイルされた静的 NAG ライブラリにリンクします。

```
nag_example_static_MD e04ucc
nag_example_static_MD e04ucf
```

ここでも、-mkl オプションを使用して MKL BLAS/LAPACK をリンクできます。

```
nag_example_static_MD -mkl e04ucc
nag_example_static_MD -mkl e04ucf
```

nag_example_static_MT.bat バッチファイルは、/MT でコンパイルされた静的 ライブラリにリンクします。例えば、

```
nag_example_static_MT e04ucc
nag_example_static_MT e04ucf
nag_example_static_MT -mkl e04ucc
nag_example_static_MT -mkl e04ucf
```

上記のいずれかのバッチファイルで、-ad スイッチを追加すると、代わりに NAG AD ライブラリの サンプルプログラムが実行されます。例えば、

```
nag_example_static_MT -ad s01ba_a1w_hcpp
```

AD サンプルの中には、このライブラリには付属していない dco.hpp ファイルに依存するものが いくつかあることに注意してください。これらのサンプルの使用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 を参照してください）。

dco.hpp ファイルと他の dco/c++ファイルがシステムにすでにある場合は、それらの場所を INCLUDE 環境変数に追加する必要があります。または、ファイルを *install_dir*¥include フォルダにコピーすることもできます。

Microsoft C/C++コンパイラの代わりに Intel C/C++コンパイラを呼び出すには、-icl オプションを バッチファイルコマンドに追加します。

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 を呼び出すサンプルをコンパイルして実行するか、 nag_example_*.bat バッチファイルの 1 つを引数 a00aaf または a00aac で 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。このサンプルは、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。

または、診断プログラム NAG_Library_DLL_info.exe を実行します。これ自体が a00aac と a00aaf を 呼び出します（**インストーラーノート**のセクション 4.2.2{.sec}を参照）。

3.5 C データ型

この実装では、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 [i	nt]{.mono} [4]{	.mono}
Pointer [v	oid *]{.mono} [8]{	.mono}

sizeof(Integer)と sizeof(Pointer)の値は、a00aac サンプルプログラムでも 提供されます。他の NAG データ型に関する情報は、Library Manual 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 以下を参照）で入手できます。

3.6 Fortran データ型と太字の用語の解釈

この NAG ライブラリの実装には、32 ビット整数のみのライブラリが含まれています。ライブラリは *install_dir*¥lib にあります。

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、すべての NAG ライブラリルーチンのドキュメントに表示されます。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。nag_wp 型を、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 ライブラリの「作業精度」型 と呼びます。

さらに、少数のルーチンでは、型

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

を使用します。ここで、nag_rp は「縮小精度」型を表します。もう 1 つの型は、

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

で、「高精度」型または「追加の精度」型を表しますが、現在ライブラリでは使用されていません。

これらの型を正しく使用するには、ライブラリに付属のほぼすべてのサンプル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)　は REAL（つまり単精度）を意味します
REAL (kind=nag_wp)　は DOUBLE PRECISION を意味します
COMPLEX (kind=nag_rp)　は COMPLEX（つまり単精度複素数）を意味します
COMPLEX (kind=nag_wp)　は倍精度複素数（例えば COMPLEX*16）を意味します

さらに、マニュアルの FL インターフェイスセクションでは、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と、C ルーチンの同等のものが利用できない従来の Fortran ルーチンにアクセスする場合や、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 持つ場合に、他の言語から使用するのがより便利な場合があります。Fortran と C の型のマッピングを ユーザーが行えるようにするために、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（*install_dir*¥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、アプリケーションでこの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++から呼び出すためのアドバイスを提供するドキュメント https://support.nag.com/numeric/nl/nagdoc_30/clhtml/genint/alt_c_interfaces.html も利用可能です。（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、 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、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_DEC
LARATION
```

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、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 1 つ以上のルーチンに適用される追加情報は、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、F16 章では、BLAS および LAPACK の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ためには、アプリケーションで 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、供給するワークスペースの量を 決定できる「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場合は注意が必要です。

この実装では、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、以下の ルーチンを除いて、MKL への呼び出しによって実装されています。

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 Library Manual 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
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助言メッセージのデフォルトの出力ユニットはどちらも Fortran ユニット 6 です。

6. (f)X06

   第 X06 章のルーチンは、このライブラリ実装で 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（NLW6I30DEL）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）

: