NAG Library, Mark 31
NLW6I31DEL - 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

目次

1 はじめに

このドキュメントは、タイトルに記載されている NAG ライブラリ実装のすべてのユーザーにとって必読の資料です。NAG Mark 31 ライブラリマニュアル（以下、ライブラリマニュアルと呼びます）に記載されている情報を補完する実装固有の詳細情報を提供しています。ライブラリマニュアルで「お使いの実装のユーザーノート」という記述がある場合は、このノートを参照してください。

さらに、NAG は任意の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（セクション 5 参照）から以下の参考資料を読むことをお勧めします：

(a) NAG ライブラリの使用方法
(b) 章の概要
(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/nl31/w6idel/supplementary.html

この実装の適用性や使用方法に関する新しい情報の詳細が記載されています。

3 一般情報

この NAG ライブラリの実装では、Intel® Math Kernel Library for Windows（MKL）というサードパーティのベンダー性能ライブラリを使用して、Basic Linear Algebra Subprograms（BLAS）と Linear Algebra PACKage（LAPACK）ルーチン（セクション 4 に記載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が用意されています。また、これらのルーチンの NAG 参照版を使用した自己完結型の静的ライブラリと共有ライブラリも提供しています（自己完結型ライブラリと呼びます）。この実装は MKL のバージョン 2021.0.4 でテストされており、このバージョンは本製品の一部として提供されています。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には、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（`nag_mkl_MT.lib`、`nag_mkl_MD.lib`、または `NLW6I31DE_mkl.lib/NLW6I31DE_mkl.dll`）を使用することをお勧めします。自己完結型の NAG ライブラリ（`nag_nag_MT.lib`、`nag_nag_MD.lib`、または `NLW6I31DE_nag.lib/NLW6I31DE_nag.dll`）よりも優先して使用してください。

使用する静的バリアントの NAG ライブラリは、Microsoft ランタイムライブラリと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、マルチスレッド動的リンク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` を使用します。あるいは、NAG ライブラリのダイナミックリンクライブラリ（DLL）バリアントを呼び出したい場合は、インポートライブラリ `NLW6I31DE_mkl.lib` または `NLW6I31DE_nag.lib` とリンクし（実行時には対応する DLL、`NLW6I31DE_mkl.dll` または `NLW6I31DE_nag.dll` がパス上にあ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、以下のバリアントのアドオンライブラリとして含まれています：`nag_nag_ad_MT.lib`、`nag_nag_ad_MD.lib`、`nag_mkl_ad_MT.lib`、`nag_mkl_ad_MD.lib`。nag_mkl_ad バリア

ントでは、多くのコアレベル 3 BLAS ルーチンの導関数をより効率的に計算するために、記号行列微積分が使用されます。

NAG ライブラリは、使用されたメモリがライブラリ自体によって、またはユーザーが NAG_FREE()を呼び出すことで回収できるように注意深く設計されています。しかし、ライブラリ自体がコンパイラのランタイムやその他のライブラリに依存しており、これらが時々メモリリークを起こす可能性があります。NAG ライブラリにリンクされたプログラムにメモリトレースツールを使用すると、これ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 ライブラリへの呼び出しが増えても無制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場合は、以下のドキュメントを参照してください：

- CL インターフェースのマルチスレッド処理
- FL インターフェースのマルチスレッド処理

（適切な方）詳細情報については。

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

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、これが保持されなくなる場合があることに注意してください。

この実装で提供されているライブラリには、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 3.1.0 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を満たしていれば（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）、`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を満たしていないことに注意してください。つまり、MKL 上に構築された特定の NAG ライブラリに対して、`MKL_CBWR` を設定することで異なる CPU アーキテクチャ間ですべての NAG ルーチンの BWR を保証することは不可能かもしれません。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、ライブラリがデフォルトのフォルダにインストールされていることを前提としています：

```
C:\Program Files\NAG\NL31\nlw6i31del
```

「Program Files」フォルダの実際の名前は、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。以下のいくつか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 (NLW6I31DEL)セクションにある前提です：

```
NAG NLW6I31DEL 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されています。）

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 参照）、実行時に NAG DLL（NLW6I31DE_mkl.dll または NLW6I31DE_nag.dll）にアクセスできるようにする必要があります。したがって、*install_dir*。*install_dir*（適切な Intel ランタイムライブラリがパス上に既にない限り）。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*、NAG バージョンのいくつかの BLAS / LAPACK ルーチンがベンダーバージョンの問題を回避するために NAG ライブラリに含まれている可能性があるため、*install_dir*。（詳細はセクション 4 を参照してください。）

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 NAG_Library_DLL_info.exe プログラムを実行します：

```
Check NAG NLW6I31DEL DLL Accessibility
```

このユーティリティの詳細については、インストール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 は、OpenMP を使用してライブラリルーチンの一部でスレッド処理を実装しています。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。

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、通常の Windows コントロールパネルを介して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各実行の間で再設定できます。

MKL ルーチンには複数レベルの OpenMP 並列性が存在する可能性があり、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場合もあります。デフォルトでは、OpenMP のネストされた並列性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。内部レベルはアクティブにならず、1 つのスレッドで実行されます。OpenMP のネストされた並列性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、OpenMP 環境変数 OMP_NESTED を照会および設定します。OpenMP のネストされた並列性が有効になっている場合、上記の例では各並列領域で上位レベルの各スレッドに対して N 個のスレッドを作成するため、OpenMP 並列性が 2 レベルある場合は合計 N*N 個のスレッドとなります。ネストされた並列性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として設定し、各レベルで希望するスレッド数を指定できます。例：

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

これにより、最初のレベルの並列性で N 個のスレッドが作成され、内部レベルの並列性に遭遇したときに各外部レベルスレッドに対して P 個の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、通常は 1 か、システムで利用可能な最大コア数に等しくなります。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レベルの並列性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、常に OMP_NUM_THREADS を希望する値に明示的に設定することをお勧めします。

一般的に、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上の物理コア数です。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、これによ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は 2 つの論理コアとして認識されます。この機能を利用することが有益な場合もありますが、この選択は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については、追加の論理コアを利用する場合としない場合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達成できます。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 BIOS で起動時に希望の選択を設定する必要があります。

提供されている Intel MKL ライブラリには、MKL 内のスレッド処理をより詳細に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 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ため、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。さらなるアドバイスが必要な場合は、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 NLW6I31DEL Command Prompt
```

を使用して、ライブラリと提供されている MKL の INCLUDE、LIB、PATH 環境変数が正しく設定されたコマンドプロンプトウィンドウを起動できます。nag_example_*.bat バッチファイルに必要な環境変数 NAG_NLW6I31DEL 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以下の通りです：

```
C:\Program Files\NAG\NL31\nlw6i31del\batch\envvars.bat
```

このファイルがデフォルトの場所にない場合は、nlw6i31del を含む envvars.bat ファイルを検索して見つけることができます。

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：

```
cl /MD driver.c NLW6I31DE_mkl.lib
ifort  /MD driver.f90 NLW6I31DE_mkl.lib

cl /MD driver.c NLW6I31DE_nag.lib
ifort  /MD driver.f90 NLW6I31DE_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.l
ib \
         /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.l
ib \
         /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.l
ib \
         /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.l
ib \
         /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` はあなたのアプリケーションプログラムです。（注意 – 上記では Microsoft C コンパイラ `cl` の使用を想定しています。Intel C コンパイラ `icl` または `icx` 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。同様に、Intel `ifx` コンパイラを `ifort` の代わりに使用することもできます。）

上記の指示は、NAG AD ライブラリなしで 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、指定された NAG ライブラリの後に適切なバリアントのライブラリを追加し、以下のライブラリとリンカオプションセットのいずれか 1 つだけを使用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、追加で `libnag_dcof_MT.lib` または `libnag_dcof_MD.lib`（適切な方）を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コードをコンパイルすることを意味します。このオプションは、c_header の例をコンパイル/リンクする際に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:`：リンカーに不要な（ここでは不要な）ランタイムライブラリについて苦情を言わせないようにします。 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。

NLW6I31DE_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。NLW6I31DE_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、正しいコンパイラランタイムライブラリにリンクすることを確実にするために、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、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、正しいコンパイラランタイムライブラリにリンクすることを確実にするために、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、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、正しいコンパイラランタイムライブラリにリンクすることを確実にするために、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。

NAG ライブラリを使用するプログラムの構築に Microsoft Visual Studio を使用する予定がある場合、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。ここでは、プロジェクトが NAG ライブラリを使用する予定であると仮定します。

ライブラリは完全に最適化されたモードで実行されることを意図しているため、警告メッセージを避けるために、アクティブな構成を Release に設定することをお勧めします。Visual Studio を開いた後、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。Debug 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ついての警告メッセージを受け取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 NAG ライブラリの 64 ビット実装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...**ボタンを介して変更できます。

以下の手順は、プロジェクトに NAG ライブラリを追加する方法を示しています： 1. プロジェクトのプロパティページを開きます。これを行うにはいくつかの方法があります： - 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、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。 - あるいは、ソリューションエクスプローラーで特定の単一プロジェクトを右クリックし、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。結果のウィンドウで、右端のプロパティページアイコンを選択します。

2. さまざまなフォルダの場所を設定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。そして
  - インクルードディレクトリを選択し、*install_dir*。
  - ライブラリ ディレクトリを選択し、*install_dir*install_dir *(および必要に応じて* install_dir*）を追加します。（あるいは、これらは以下の Fortran プロジェクトの指示のように、追加のライブラリ ディレクトリ設定を介して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、次に全般を選択します。そして
  - 追加のインクルードディレクトリを選択し、*install_dir*_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をクリック/展開し、次に全般を選択します。そして
  - 追加のライブラリ ディレクトリを選択し、*install_dir*install_dir *(および必要に応じて* install_dir*）を追加します。* デフォルトのフォルダは以下の通りです：``` C/C++プロジェクトのインクルードディレクトリ C:Files6i31del

Fortran プロジェクトの追加のインクルードディレクトリ C:Files6i31del_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:Files6i31del C:Files6i31del C:Files6i31del ``` 変更を受け入れるには適用ボタンをクリックするか、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。

3. NAG ライブラリと Intel ランタイムライブラリ（および場合によっては MKL ライブラリ）をリンカ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、左側のパネルで（構成プロパティの下の）リンカをクリック/展開し、入力を選択して、追加の依存ファイルリストに適切なライブラリファイルを追加します。以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、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。

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：**

   - まず、プロジェクトメニューの既存項目の追加...を使用して、ソースファイル（例えば NAG の Example プログラム）をプロジェクトに追加します。（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、次に 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、右側のパネルからランタイム ライブラリを選択し、これを適切なバージョンに変更します。例えば、プロジェクトが `nag_nag_MT.lib` または `nag_mkl_MT.lib` の 2 つの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 (/MT)を選択します。プロジェクトが他の NAG ライブラリを使用する場合は、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後、変更を受け入れるには適用ボタンをクリックするか、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、次にライブラリを選択します。右側のパネルにランタイム ライブラリのエントリが表示されます。プロジェクトが `nag_nag_MT.lib` または `nag_mkl_MT.lib` の 2 つの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を選択する必要があります。プロジェクトが他の NAG ライブラリを使用する場合は、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後、変更を受け入れるには適用ボタンをクリックするか、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。

NAG ライブラリ	MKL およびその他のライブラリ	ランタイムライブラリ
NLW6I31DE_mkl.lib	user32.lib (AD ライブラリとリンクする場合のみ必要)	マルチスレッド DLL (/MD)
NLW6I31DE_nag.lib	user32.lib (AD ライブラリとリンクする場合のみ必要)	マルチスレッド DLL (/MD)
nag_mkl_MT.lib	mkl_intel_lp64.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib user32.lib	マルチスレッド (/MT)
nag_nag_MT.lib	user32.lib	マルチスレッド (/MT)
nag_mkl_MD.lib	mkl_intel_lp64.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib user32.lib	マルチスレッド DLL (/MD)
nag_nag_MD.lib	user32.lib	マルチスレッド DLL (/MD)

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 DLL ではなく `nag_mkl_MT.lib` または `nag_mkl_MD.lib`）にリンクする場合、いくつかのランタイムライブラリを無視するようにリンカに指示するためにプロジェクト設定を変更する必要があります。再び構成プロパティで、左側のパネルでリンカ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、特定の既定のライブラリの無視を選択し、編集を選択して、セクション 3.1.1 のコマンドで`/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` または`/MT` ビルドに適したセットを選択してください（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、ビルドメニューから適切な選択肢を選んでプロジェクトを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、デバッグメニューからプログラムを実行できます（例えば、デバッグなしで開始（Ctrl+F5）を選択することで）。セクション 3.1.1 で説明したように、`PATH` 環境変数を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や、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、これ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：

```
< 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、完全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 の例題については、このファイル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を介して設定できます。

3.1.3 Fortran モジュールファイルに関する注意

この NAG ライブラリ実装で提供されている `nag_interface_blocks` フォルダ内の 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、他の Fortran コンパイラでは使用できません。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インターフェースブロックを使用する場合で、別のコンパイラを使用する場合は、

まず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

考慮すべき 3 つのケースがあります： 1. Fortran Builder には、NAG ライブラリの一部のバージョンに関する組み込みの知識があります。お使いの Fortran Builder のバージョンによっては、このライブラリ NLW6I31DEL について知っている場合があります。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」を作成する際にそれを使用するはずです。これは NLW6I31DEL を Fortran Builder から使用する最も簡単な方法です。なぜなら、NAG ライブラリやインターフェースブロックの場所を IDE に伝える必要がないからです。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。 2. Fortran Builder のバージョンがこのライブラリ NLW6I31DEL のリリースよりも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。しかし、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、次に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*_interface_blocks_nagfor を追加します（ディレクトリ名にスペースが含まれていても、引用符で囲む必要はないことに注意してください） - 次に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*6I31DE_nag.dll（DLL 自体にリンクすることが重要で、関連するインポートライブラリにはリンクしないことに注意してください。また、静的ライブラリにリンクすることはできず、DLL のみ可能です） - 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場合、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でアクセス可能な*未定義変数*オプションを使用することは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ためです。これを使用しようとすると、Fortran Builder で NAG インターフェースブロックを使用する際に「互換性のないオプション設定」を示すコンパイル時エラーが発生します。 3. 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 で使用するためのインターフェースブロックは、フォルダ *install_dir*_interface_blocks_nagfor で提供されています。NAG コンパイラの異なるバージョンをお持ちの場合は、まずセクション 3.2 で説明されているよう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。DLL 自体にリンクする必要があり、関連するインポートライブラリにはリンクしないことが重要です。Windows のコマンドプロンプトから、まずセクション 3.1.1 で説明されているように `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： `bash nagfor -ieee=full -I"*install_dir*\nag_interface_blocks_nagfor" driver.f90 \ "*install_dir*\bin\NLW6I31DE_mkl.dll" -o driver.exe nagfor -ieee=full -I"*install_dir*\nag_interface_blocks_nagfor" driver.f90 \ "*install_dir*\bin\NLW6I31DE_nag.dll" -o driver.exe` MKL サポート版のライブラリにリンクするか、完全 NAG 版にリンクするかによって選択します。`NLW6I31DE_mkl` または `NLW6I31DE_nag` ライブラリファイルのフルパス名を指定する必要があり、スペースが含まれている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

上記で言及されていない環境から NAG ライブラリを呼び出す方法に関する情報は、補足情報ページで入手できる場合があります： https://support.nag.com/doc/inun/nl31/w6idel/supplementary.html

3.2 Fortran インターフェースブロック

NAG ライブラリインターフェースブロックは、ユーザーが呼び出し可能な各 NAG ライブラリ Fortran ルーチンのタイプ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必須ではありませんが、その使用を強く推奨します。また、提供されている例題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かどうかをチェックできるようにすることです。インターフェースブロックを使用することで、コンパイラは以下のことをチェックできます： - (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こと - (b) 関数が正しい型で宣言されていること - (c) 正しい数の引数が渡されていること - (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 ライブラリインターフェース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。これらは以下の名前の 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 参照）、Intel `ifort` コンパイラを使用する場合、環境変数 `INCLUDE` が設定されるため、セクション 3.1.1 で説明されている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。

`.mod` モジュールファイルは、インストール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、これらのモジュールと互換性のないコンパイラを使用して NAG の Example プログラムを使用したり、独自のプログラムでインターフェースブロックを使用したりする場合は、まず独自のモジュールファイルを作成する必要があります。ここでその方法を説明します。

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、元のインターフェースブロックのセットを保存します。

次に、`nag_interface_blocks` フォルダ内で、お使いの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コンパイルし直します。インターフェースブロックにはいくつかの相互依存関係があるため、コンパイルの順序が重要ですが、以下のコンパイル順序で問題ないはずです。ここで `FCOMP` はあなたの Fortran コンパイラの名前です：

```
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してかまいません - 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はずです。

現在、ユーザーが AD ルーチンのインターフェース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 サポートにお問い合わせください（連絡先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、インストール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 31 で生成されました。これらの Example 結果は、Example プログラムが若干異なる環境（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）で実行された場合、厳密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以下の通りです：固有ベクトル（多くの場合-1、時には複素数のスカラー倍で異なる場合があります）、反復回数と関数評価回数、および残差やマシン精度と同じオーダーの他の「小さな」量です。

配布されている Example 結果は、静的ライブラリ `nag_mkl_MD.lib`（つまり MKL BLAS と LAPACK ルーチンを使用）で得られたものです。NAG BLAS または LAPACK を使用して例題を実行すると、若干異なる結果が得られる場合があります。 配布されている NAG AD の例題結果は、静的ライブラリ `nag_mkl_MD.lib` および `nag_nag_ad_MD.lib` を用いて得られたものです。

例題資料は、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おり、これにより、プログラムはこの実装でそれ以上の変更なしで実行するのに適しています。配布されている Example プログラムは、可能な限りライブラリマニュアルのバージョンよりも優先して使用する必要があります。

Example プログラムには、*install_dir*`nag_example_DLL.bat`、`nag_example_static_MT.bat`、`nag_example_static_MD.bat` バッチファイル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ことを必要とします。特に、環境変数 `NAG_NLW6I31DEL` を NAG ライブラリの場所に設定する必要があります。これを行う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。

上記の各 `nag_example_*.bat` バッチファイルは、Example プログラム（およびそのデータとオプションファイル、もしあれば）のコピーを提供し、プログラムをコンパイルし、適切なライブラリとリンクします（プログラムの独自バージョン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られます。 C と Fortran の両方の Example プログラムが提供されています。

対象となる Example プログラムは、コマンドの引数で指定します。例：

```
nag_example_DLL e04ucc
nag_example_DLL e04ucf
```

これにより、Example 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、実行されて、Example プログラムの結果が `e04ucce.r`（C の場合）または `e04ucfe.r`（Fortran の場合）ファイルに出力されます。

`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。 MKL バージョンの DLL にリンクするには、`-mkl` オプションを使用します。例：

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

`nag_example_static_MD.bat` バッチファイルは同じ方法で使用され、`/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 にリンクすることができます：

```
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：

```
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 bat ファイルに`-ad` スイッチを追加すると、代わりに NAG AD ライブラリの Example プログラムが実行されます。例：

```
nag_example_static_MT -ad s01ba_a1w_hcpp
```

AD の例題のいくつかは、このライブラリには同梱されていないファイル `dco.hpp` に依存していることに注意してください。これらの例題の使用に興味がある場合は、NAG にお問い合わせください（連絡先の詳細はセクション 7 を参照）。

システムに `dco.hpp` ファイルや他の dco/c++ファイルが既にある場合は、それらの場所を `INCLUDE` 環境変数に追加する必要があります。あるいは、ファイルを *install_dir*。

Microsoft C/C++コンパイラの代わりに Intel Classic C/C++コンパイラ（`icl`）を使用するには、バッチファイルコマンドに`-icl` オプションを追加します。あるいは、Intel `icx` または `ifx` コンパイラを指定するには、それぞれ`-icx` または`-ifx` 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 を呼び出す例題をコンパイルおよび実行するか、`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。この例題は、タイトルと製品コード、使用されているコンパイラと精度、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。

あるいは、a00aac と a00aaf を呼び出す診断プログラム NAG_Library_DLL_info.exe を実行します（インストールノートのセクション 4.2.2 を参照）。

3.5 C データ型

この実装では、NAG C タイプ Integer と Pointer は以下のように定義されています：

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

sizeof(Integer)と sizeof(Pointer)の値は、a00aacExample 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（セクション 5 参照）の NAG CL インターフェース概要のセクション 3.1.1 で入手できます。

3.6 Fortran データ型と太字斜体語の解釈

この NAG ライブラリの実装には、32 ビット整数用のライブラリのみが含まれています。ライブラリは *install_dir*。

NAG ライブラリとドキュメントは、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、すべての NAG ライブラリルーチンのドキュメントには、以下の型が表示されます：

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

ここで、nag_wp は Fortran 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 モジュールの使用によって取得できます。我々は nag_wp 型を NAG ライブラリの「作業精度」型と呼びます。なぜなら、ライブラリで使用される多くの浮動小数点引数と内部変数がこの型だからです。

さらに、少数のルーチンは以下の型を使用します：

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

ここで、nag_rp は「精度低下」型を表します。現在ライブラリでは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。

これらの型の正しい使用については、ライブラリと一緒に配布されているほとんどの Example プログラムを参照してください。

この実装では、これらの型は以下の意味を持ちます：

```
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 インターフェースセクションでは、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 インターフェース概要のセクション 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

注意深く行えば、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から使用することができます。Fortran ルーチンをこの方法で使用することは、C 言語ルーチンの同等物が利用できないレガシーFortran ルーチンにアクセスする場合や、他の言語からの使用がより便利な可能性のある、基本的な C データ型のみを使用したより低レベルの C インターフェースを持つ場合に好ましい場合があります。

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、C 視点からの Fortran インターフェースの説明（C ヘッダインターフェース）が各 Fortran ルーチンドキュメントに含まれています。C/C++ヘッダファイル（*install_dir*

.h）も提供されています。NAG Fortran ルーチンをこの方法で使用したいユーザーは、アプリケーションでこのヘッダファイルを`#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++から呼び出す方法についてのアドバイスを提供するドキュメント `alt_c_interfaces.html` も利用可能です。（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 C/C++ヘッダファイルには、NAG ライブラリに含まれる LAPACK、BLAS、BLAS 技術フォーラム（BLAST）ルーチンの宣言が含まれています。ユーザーは、提供されている Intel MKL など、他のライブラリに関連する C include ファイルからこれらの定義を取得することを好む場合があります。このような状況で、異なる C ヘッダ宣言間の衝突を避けるために、これらのルーチンの NAG 宣言は、セクション 3.1 で説明されている C または C++コンパイル文に以下のコンパイルフラグを追加することで無効にすることができます：

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

代替 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。

(a) F06、F07、F08、F16

F06、F07、F08、F16 の章では、BLAS および LAPACK 派生ルーチンの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、関連する章の概要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提供するワークスペースの量を決定するためにルーチンを照会できる「ワークスペース照会」メカニズムがあります。MKL ライブラリの LAPACK ルーチンは、同等の NAG 参照バージョンのこれらのルーチンとは異なる量のワークスペースを必要とする場合があることに注意してください。ワークスペース照会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、自己完結型でない 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
```

(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 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
```

(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
x02be[fc] (nag[f]_machine_decimal_digits)
     = 15
```

(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およびアドバイザリメッセージのデフォルト出力ユニットは、どちらも Fortran ユニット 6 です。

(f) X06

X06 章のルーチンは、このライブラリ実装で MKL スレッド処理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、NAG ウェブサイトの NAG Library Manual, Mark 31 でアクセスできます。

ライブラリマニュアルは、HTML と MathML を使用した完全にリンクされたバージョンのマニュアルである HTML5 で提供されています。これらのドキュメントは、ウェブブラウザを使用してアクセスできます。

ドキュメントの表示とナビゲーションに関するアドバイスは、Guide to the NAG Library Documentation で見つけることができます。

さらに、以下が提供されています： `-in.html` – インストールノート `-un.html` – ユーザーノート（このドキュメント） `-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方法に関するアドバイス

ユーザーノートは、デフォルトでスタートメニューまたはすべてのアプリの NAG Library (NLW6I31DEL)セクションから

`NAG NLW6I31DEL Users' Note`

として利用できます。

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（NLW6I31DE）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://nag.com/ （英国本社）