NAG Library, Mark 30.1, Multithreaded

NSL6I301BL - Licence Managed

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

ユーザーノート

内容

1 はじめに 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 インターフェースブロック 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 ルーチンを呼び出す 12

3.8 LAPACK、BLAS 等の C 宣言 12

4 ルーチン固有の情報 12

5 ドキュメンテーション 17

6 サポート 18

7 コンタクト情報 18

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です。これは、NAG Mark 30.1 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ノート」を参照するように指示されている場合は、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、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推奨しています（セクション 5 を参照）：

(a) NAG ライブラリの使用方法
(b) 章の紹介
(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/ns30/l6i1bl/supplementary.html

この実装の適用性または使用に関する新しい情報の詳細については、上記 URL 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、Linux 用 Intel® Math Kernel Library（MKL）を使用した静的および共有ライブラリを提供しています。MKL は、Basic Linear Algebra Subprograms（BLAS）と Linear Algebra PACKage（LAPACK）ルーチンを提供するサードパーティのベンダーパフォーマンスライブラリです（セクション 4 に記載されているルーチンを除く）。また、これらのルーチンの NAG 参照バージョンを使用した自己完結型の静的および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、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（`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` で表記）と 64 ビット整数（`ilp64` で表記）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 ライブラリはこの実装には含まれていません。

NAG ライブラリは、使用されるメモリを確実に回収できるように慎重に設計されています。これはライブラリ自体によって、または C ルーチンの場合はユーザーが NAG_FREE()を呼び出すことで行われます。ただし、ライブラリ自体は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をリークする可能性があります。NAG ライブラリにリンクされたプログラムで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、過剰であってはならず、NAG ライブラリへの呼び出しが増えるにつれて無制限に増加することはありません。

NAG ライブラリをマルチスレッドアプリケーション内で使用する場合は、詳細情報について CL インターフェースマルチスレッディングまたは FL インターフェースマルチスレッディング（適宜）のドキュメント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際の詳細情報は、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。

この実装で提供されるライブラリは OpenMP でコンパイルされています。ただし、異なるコンパイラの OpenMP ランタイムライブラリは互換性がない場合があるため、インストーラーノートのセクション 2.2 に記載されているコンパイラと対応する OpenMP ランタイムを使用する場合にのみ、この実装を独自の OpenMP コード（セクション 4 に記載されているルーチンのユーザー提供関数で必要な OpenMP 文を含む）と併用することをお勧めします。

マルチスレッドアプリケーション内ですべての NAG ライブラリルーチンを実行するためには、システムのデフォルトのスレッドスタックサイズが十分でない場合があることに注意してください。OpenMP 環境変数 `OMP_STACKSIZE` を使用してこのスタックサイズを増やすことができます。

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す場合（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）、`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 ライブラリについて、`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 を保証することができない場合があります。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`[INSTALL_DIR]`ディレクトリにインストールされていると仮定します。デフォルトでは`[INSTALL_DIR]`（インストーラーノート（in.html）を参照）は `$HOME/NAG/nsl6i301bl` 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、その人に確認してください。リンク時と実行時に適切なライブラリの場所を指すように、環境変数 LD_LIBRARY_PATH を正しく設定する必要があることに注意してください。その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 ライブラリとナグフォートランライブラリ（フォートランライブラリインターフェースへの 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品です。この実装に含まれる異なるライブラリの呼び出しを支援するために、nagvars.sh と nagvars.csh スクリプトが含まれています。これらは、NAG ルーチンを呼び出すアプリケーションのコンパイルとリンクを支援するための NAG 固有の環境変数を設定します。また、コンパイル時、リンク時、実行時に NAG 実行可能プログラムとライブラリを見つけることができるように、標準環境変数 PATH と LD_LIBRARY_PATH も修正します。

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] [-ifx] [-gnu] \
  {int32,int64} {vendor,nag} {static,dynamic}
```

または

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

ここで：- nagvars.sh は Bourne、Bash、または同等のシェル用です（注：Ubuntu などの Debian 派生の Linux ディストリビューションで利用可能な Dash は使用しません）。nagvars.csh は Csh、Tcsh、または同等のシェル用です。 - {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、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または動的（共有）バージョン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、標準環境変数 PATH と LD_LIBRARY_PATH から**この NAG 実装のみ**の材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 nagvars スクリプトを使用して環境を設定する例は以下の通りです：

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

設定される NAG 固有の環境変数は以下の通りです： - 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 コンパイラで NAG（およびオプションでベンダーBLAS と LAPACK）ルーチンへのリンクに必要なライブラリ。 - NAGLIB_CXXLINK - C++コンパイラで NAG（およびオプションでベンダーBLAS と LAPACK）ルーチンへのリンクに必要なライブラリ。 - NAGLIB_FLINK - Fortran コンパイラで NAG（およびオプションでベンダーBLAS と LAPACK）ルーチンへのリンクに必要なライブラリ。

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、以下の方法でリンクできます：

C プログラムの場合：

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

C++プログラムの場合：

```
${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_FLINK}
```

コンパイラのランタイムライブラリなど、他の項目を指すために LD_LIBRARY_PATH を設定する必要がある場合もあります。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。

異なるコンパイラを使用している場合、あるいは Intel コンパイラの異なるバージョンを使用している場合、[INSTALL_DIR]/rtl/lib/intel64 に提供されている Intel コンパイラ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、以下を LD_LIBRARY_PATH に追加することで容易になります：

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

Intel icx または ifx コンパイラから呼び出すには、単に icc または ifort をそれぞれ icx または ifx に置き換えるだけです（ただし、ifx での静的リンクに関する以下の注意を参照してください）。これを最も簡単に行うには、nagvars.sh および/または nagvars.csh スクリプトの-ifx オプションを使用します。このオプションを使用すると、NAG 固有の環境変数が適切に Intel ifx、icx、または icpx コンパイラを使用するように修正されます。

より新しい Intel Fortran コンパイラ（ifort または ifx）での*静的*リンクが、多重定義によりリンクエラーを引き起こすことは既知の問題です。ライブラリの構築に使用された少し古いバージョンの ifort では、静的リンクを簡素化するために一部の Intel ランタイムライブラリがライブラリの静的バリアントに静的に含まれていました。これは、これらのランタイムが ifort によって自動的にリンクされなかったためです。より新しい Fortran コンパイラでは、これらのランタイムが自動的にリンクされるようになり、Linux では、これらのデフォルトライブラリをリンク中に個別に無効化することができません（Windows ではできます）。Linux での回

避策は、代わりに NAG ライブラリの共有バリアントの 1 つを使用することです。これはまた、パフォーマンスの大きな損失がなく、はるかに小さなサイズの実行可能ファイルが生成されるため、推奨されるリンク形式でもあります。

このバージョンの NAG ライブラリは、少なくともウェブサイトの補足情報ページに記載されているバージョンの gcc を使用して、GNU gcc および g++コンパイラから呼び出すことができます。これを最も簡単に行うには、`nagvars.sh` および/または `nagvars.csh` スクリプトの`-gnu` オプションを使用します。このオプションを使用すると、NAG 固有の環境変数が適切に GNU gcc および g++を使用するように修正されます。

GNU gfortran はサポートされていないため、Fortran 関連の NAG 環境変数は引き続き Intel ifort コンパイラを参照することに注意してください。また、MKL とのリンクは、NAG ライブラリに存在する可能性のある 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 ランタイムを使用することに注意してください。

3.1.1 使用するスレッド数の設定

NAG ライブラリのこの実装と MKL は、一部のライブラリルーチンでスレッド処理を実装するために 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、以下のように入力します：

```
setenv OMP_NUM_THREADS N
```

Bourne シェルでは、以下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、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。プログラムの実行中に異なる部分で使用するスレッド数を変更したい場合、NAG ライブラリの第 X06 章にこのプロセスを支援するルーチンが提供されています。

一部の NAG ライブラリおよび 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、これらのマルチスレッドルーチンを独自のアプリケーションの OpenMP 並列領域内から呼び出すこともできます。デフォルトでは、OpenMP 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、つまり 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` をクエリおよび設定するか、第 X06 章の適切なルーチンを使用します。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` 個のスレッドが作成されるため、2 レベルの OpenMP 並列性がある場合は合計 `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、以下のように入力します：

```
setenv OMP_NUM_THREADS N,P
```

Bourne シェルでは、以下のように入力します：

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

これにより、並列性の最初のレベルで N 個のスレッドが作成され、内部レベルの並列性が発生したときに外部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る場合があり、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、独自のアプリケーション内でより高い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 を明示的に希望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表示されます。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利用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。

提供された Intel MKL ライブラリには、MKL 内のスレッド処理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-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。代わりに、ユーザーは第 X06 章のルーチンを使用することをお勧めします。これらは、呼び出しプログラム、NAG ルーチン、MKL の OpenMP に等しく適用されます。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。

3.2 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

ラリを呼び出すの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、提供された例を使用する場合は必須です。

これらの目的は、Fortran コンパイラがリ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により、コンパイラは以下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で一致していること

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編成されています。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 Classic Fortran コンパイラ用にコンパイルされた形式（.mod ファイル）で提供されています。これらには、各コンパイラ呼び出し時に-I パス名オプションを指定することでアクセスできます。ここでパス名（[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インターフェースブロック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、インストーラー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 の例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ースブロックを使用したい場合で、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いる場合は、まず独自のコンパイラバージョンでインターフェースブロックを再コンパイルする必要があります。独立したディレクトリ（例：nag_interface_blocks_alt）に再コンパイルされたインターフェースブロックのセットを作成するには、提供されたスクリプトコマンドを使用します：

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

これは[INSTALL_DIR]ディレクトリから実行します。このスクリプトは、PATH 環境から 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最初に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、まず両方の整数サイズに対して{int32,int64}オプションを別々に使用して
[INSTALL_DIR]/scripts/nag_recompile_mods を呼び出し、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。その後、以下のコマンドを使用して両方のデフォルトセットを変更できます（[INSTALL_DIR]を適切なディレクトリパスに置き換えてください）：

```
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_blo
cks
mv [INSTALL_DIR]/ilp64/nag_interface_blocks [INSTALL_DIR]/ilp64/nag_interface_block
s_original
mv [INSTALL_DIR]/ilp64/nag_interface_blocks_alt [INSTALL_DIR]/ilp64/nag_interface_b
locks
```

これで、通常の方法で新しくコンパイルされたモジュールファイルを使用できるはずです。

3.3 Example プログラム

配布された例の結果は、インストーラーノートのセクション 2.2 に記述されたソフトウェアを使用して Mark 30.1 で生成されました。これらの例の結果は、例プログラムが少し異なる環境で実行された場合（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 ルーチンのセットを使用した場合）、厳密に再現できない場合があります。このような差異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場合がある）、反復回数と関数評価回数、および残差や機械精度と同じオーダーの他の「小さな」量です。

配布された例の結果は、32 ビット整数の静的ライブラリ `libnag_mkl.a`（つまり、MKL BLAS 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。

例の資料は、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おり、これにより、この実装でさらなる変更なし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よりも、配布された例プログラムを使用すべきです。例プログラムには、`[INSTALL_DIR]/scripts` ディレクトリにある `nag_example` スクリプト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、例プログラム（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て適切なライブラリとリンクします。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られます。デフォルトでは、`nag_example` は 32 ビット整数と自己完結型の `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、上記の各段階で実行されるコメントとコマンドの印刷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` スクリプトの使用を示していますが、呼び出しシェルの環境を変更しないことに注意してください。

対象の例プログラムと使用する OpenMP スレッド数は、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 4
```

は、例プログラムとそのデータおよびオプションファイル（e04ucce.c、e04ucce.d、e04ucce.opt）を現在のディレクトリにコピーし、プログラムをコンパイルしてリンクし、4 つの OpenMP スレッドを使用して実行し、例プログラムの結果を e04ucce.r ファイルに出力します。

同様に、NAG Fortran ルーチンの場合：

```
nag_example e04nrf 4
```

は、例プログラムとそのデータおよびオプションファイル（e04nrfe.f90、e04nrfe.d、e04nrfe.opt）を現在のディレクトリにコピーし、プログラムをコンパイルしてリンクし、4 つの OpenMP スレッドを使用して実行し、例プログラムの結果を e04nrfe.r ファイルに出力します。

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 を呼び出す例をコンパイルして実行することで確認できます。または、nag_example スクリプトを引数 a00aaf または a00aac で呼び出すこともできます。セクション 3.3 を参照してください。この例では、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（[INSTALL_DIR]/lp64/lib ディレクトリにある）では、NAG C 型の 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）
Integer	int	4
Pointer	void *	8

64 ビット整数ライブラリ（[INSTALL_DIR]/ilp64/lib ディレクトリにある）では、NAG C 型の 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）
Integer	long	8
Pointer	void *	8

sizeof(Integer)と sizeof(Pointer)の値は、a00aac 例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 を参照）の NAG CL インターフェース紹介コンポーネントのセクション 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（[INSTALL_DIR]/lp64/lib ディレクトリにある）と 64 ビット整数（[INSTALL_DIR]/ilp64/lib ディレクトリにある）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 モジュールを使用して取得できます。我々は nag_wp 型を NAG ライブラリの「作業精度」型と呼んでいます。なぜなら、ライブラリで使用されるほとんど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ています：

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

ここで、nag_rp は「減少精度」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として、以下のものがあります：

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

これは「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、ライブラリに付属のほとんどの例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：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、一部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 インターフェース紹介のセクション 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できます。このように Fortran ルーチンを使用することは、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より便利な、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に、C 視点からの Fortran インターフェースの説明（C ヘッダーインターフェース）が各 Fortran ルーチン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。このように NAG Fortran ルーチンを使用したいユーザーは、アプリケーションで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++から呼び出す方法についてのアドバイスを提供する `alt_c_interfaces.html` というドキュメントも利用可能です。（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、提供された Intel MKL など、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場合があります。このような状況で、異なる C ヘッダー宣言間の衝突を避けるために、以下のコンパイルフラグを追加することで、これらのルーチンの NAG 宣言を無効にすることができます：

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

これらのフラグは、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します。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さらなる情報は、以下に章ごとに記載されています。

(a) OpenMP 並列領域内でユーザー関数を呼び出すルーチン

この実装では、以下のルーチンが NAG ルーチン内の OpenMP 並列領域からユーザー関数を呼び出します。

C ルーチン：

```
e05ucc  e05usc  f01elc  f01emc  f01flc  f01fmc  f01jbc  f01jcc
f01kbc  f01kcc
```

Fortran ルーチン：

```
d03raf  d03rbf  e05saf  e05sbf  e05ucf  e05usf  f01elf  f01emf
f01flf  f01fmf  f01jbf  f01jcf  f01kbf  f01kcf
```

したがって、インストーラーノートのセクション 2.2 に記載されているものとは異なるコンパイラを使用している場合を除き、ユーザー関数で孤立した OpenMP ディレクティブを使用できます。また、ユーザーワークスペース配列 IUSER、RUSER、CPUSER をスレッドセーフな方法で使用する必要があります。これは、ユーザー関数に読み取り専用データを提供するためにのみこれらを使用することで最もよく達成されます。

(b) C06

この実装では、以下の NAG C ルーチンで可能な限り、提供された MKL ライブラリから Intel Discrete Fourier Transforms Interface（DFTI）ルーチンへの呼び出しが行われます：

```
c06pac  c06pcc  c06pfc  c06pjc  c06pkc  c06ppc  c06pqc  c06prc
c06psc  c06puc  c06pvc  c06pwc  c06pxc  c06pyc  c06pzc  c06rac
c06rbc  c06rcc  c06rdc
```

そして以下の NAG Fortran ルーチンで：

```
c06paf  c06pcf  c06pff  c06pjf  c06pkf  c06ppf  c06pqf  c06prf
c06psf  c06puf  c06pvf  c06pwf  c06pxf  c06pyf  c06pzf  c06raf
c06rbf  c06rcf  c06rdf
```

Intel DFTI ルーチンは内部的に独自のワークスペースを割り当てるため、上記の NAG Fortran ルーチンに渡されるワークスペース配列 WORK のサイズを、それぞれのライブラリドキュメントで指定されているものから変更する必要はありません。

(c) F06、F07、F08、F16

F06、F07、F08、F16 の章では、BLAS および LAPACK 由来のルーチンに代替の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、関連する章の紹介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必要な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、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
```

(d) **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
```

(e) **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
```

(f) **X02**

機械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
```

(g) **X04**

Fortran ルーチン：明示的な出力を生成できるこれらのルーチンのエラーおよびアドバイザリー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。

(h) **X06**

X06 章のルーチンは、このライブラリの実装で MKL のスレッド処理の動作も変更します。

5 ドキュメンテーション

ライブラリマニュアルは、NAG ウェブサイトの NAG ライブラリマニュアル、Mark 30.1 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。

ドキュメントの閲覧とナビゲーションに関するアドバイスは、NAG ライブラリドキュメンテーションガイドにあります。

さらに、以下が提供されています：

- `in.html` – インストーラーノート
- `un.html` – ユーザーノート（このドキュメント）
- `alt_c_interfaces.html` – 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

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（NSL6I301BL）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）