NAG C Library, Mark 26, Multithreaded
CSL6I26DDL - License Managed
Linux 64 (Intel 64 / AMD64), Intel C/C++, 64-bit integers

ユーザーノート

内容

1. イントロダクション

本ユーザーノートは，NAG C Library，Mark 26，Multithreaded - CSL6I26DDL（ライブラリ）のご利用方法（リンク方法）を説明します.

本ユーザーノートには，NAG Library Manual，Mark 26（ライブラリマニュアル）には含まれない製品毎の情報が含まれています．ライブラリマニュアルに「ユーザーノート参照」などと書かれている場合は，本ユーザーノートをご参照ください.

ライブラリルーチンのご利用に際しては，ライブラリマニュアル（「5. ドキュメント」参照）の以下のドキュメントをお読みください.

(a) How to Use the NAG Library and its Documentation
(b) Chapter Introduction
(c) Function Document

2. 追加情報

本ライブラリの動作環境やご利用方法についての最新の情報は，以下のウェブページをご確認ください.

http://www.nag.co.uk/doc/inun/cs26/l6iddl/supplementary.html

3. 一般情報

本製品では，Intel ® Math Kernel Library for Linux（MKL）が提供する BLAS／LAPACK ルーチンを利用するスタティックライブラリ libnagc_mkl.a および共有ライブラリ libnagc_mkl.so と，NAG が提供する BLAS／LAPACK ルーチンを利用するスタティックライブラリ libnagc_nag.a および共有ライブラリ libnagc_nag.so を提供します．本ライブラリは，MKL version 11.3.3 を用いてテストされています．MKL version 11.3.3 は本製品の一部として提供されます．MKL の詳細については Intel 社のウェブサイト https://software.intel.com/intel-mkl をご参照ください．パフォーマンスの面からは，MKL を利用するバージョンの NAG ライブラリ libnagc_mkl.a または libnagc_mkl.so のご利用を推奨します．

NAG ライブラリはメモリリークが起きないように設計されています．メモリの解放は NAG ライブラリ自身によってか，もしくはユーザーが NAG_FREE()を呼び出すことによって行われます．ただし，NAG ライブラリが依存している他のライブラリ（コンパイラのランタイムライブラリなど）がメモリリークを起こすかもしれません．このため，NAG ライブラリをリンクしているプログラムに対して何らかのメモリトレースツールを使った際に，場合によってはメモリリークが検出されるかもしれません．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ると思われますが，NAG ライブラリの呼び出し回数に比例して際限なく増加するものではありません．

NAG ライブラリをマルチスレッドアプリケーションから使用する場合の注意点については，"How to Use the NAG Library and its Documentation" ドキュメントの「2.10.1 スレッドセーフ」をご参照ください．また，本製品に付属の MKL をマルチスレッドアプリケーションから使用する場合の詳細については，以下の Intel 社のウェブサイトをご参照ください．

http://software.intel.com/en-us/articles/intel-math-kernel-library-intel-mkl-using-intel-mkl-with-threaded-applications

本ライブラリは OpenMP を用いてコンパイルされています．また，異なるコンパイラ間では OpenMP ランタイムライブラリの互換性は保証されません．従って，自身のコードが OpenMP を利用している場合は，インストールノートに記載されているコンパイラを使用することを推奨します．なお，システムのデフォルトのスレッドスタックサイズは，マルチスレッドアプリケーション内の全ての NAG ライブラリルーチンを実行するには十分ではないことに注意してください．スレッドスタックサイズは OpenMP 環境変数 OMP_STACKSIZE で増やすことができます．

MKL には，条件付きビット単位の再現性（Bit-wise Reproducibility（BWR））オプションがあります．一定の条件（https://software.intel.com/en-us/node/528579 参照）をユーザーコードが満たしていれば，環境変数 MKL_CBWR を設定することにより BWR が有効になります．詳細は MKL のドキュメントをご参照ください．しかしながら，多くの NAG ルーチンはこれらの条件を満たしていません．従って，MKL を利用するバージョンの NAG ライブラリの全ルーチンに対して，異なる CPU アーキテクチャに渡り MKL_CBWR による BWR を保証することはできません．BWR に関するより一般的な情報は，"How to Use the NAG Library and its Documentation" の「2.9.1 Bit-wise Reproducibility (BWR)」をご参照ください．

3.1. ライブラリのリンク方法

本セクションでは [INSTALL_DIR] に本ライブラリがインストールされていることが前提となります.

デフォルトの [INSTALL_DIR] は $HOME/NAG/csl6i26ddl となります. また, インストール時に [INSTALL_DIR] を指定することもできます.

MKL を利用するバージョンの NAG ライブラリを利用する場合は, 以下のようにコンパイル・リンクを行ってください.（ここで driver.c がユーザープログラムです.）

スタティックライブラリを利用する場合：

```
icc -qopenmp driver.c -I[INSTALL_DIR]/include \
    [INSTALL_DIR]/lib/libnagc_mkl.a \
    -Wl,--start-group \
    [INSTALL_DIR]/mkl_intel64_11.3.3/lib/libmkl_intel_ilp64.a \
    [INSTALL_DIR]/mkl_intel64_11.3.3/lib/libmkl_intel_thread.a \
    [INSTALL_DIR]/mkl_intel64_11.3.3/lib/libmkl_core.a \
    -Wl,--end-group \
    [INSTALL_DIR]/rtl/intel64/libiomp5.a -lpthread -lm -ldl \
    [INSTALL_DIR]/rtl/intel64/libifcoremt.a -lstdc++
```

共有ライブラリを利用する場合：

```
icc -qopenmp driver.c -I[INSTALL_DIR]/include \
    [INSTALL_DIR]/lib/libnagc_mkl.so \
    -L[INSTALL_DIR]/mkl_intel64_11.3.3/lib \
    -lmkl_intel_ilp64 -lmkl_intel_thread -lmkl_core \
    -L[INSTALL_DIR]/rtl/intel64 \
    -liomp5 -lpthread -lm -ldl -lifcoremt
```

MKL を利用しないバージョンの NAG ライブラリを利用する場合は，以下のようにコンパイル・リンクを行ってください．（ここで driver.c がユーザープログラムです．）

スタティックライブラリを利用する場合：

```
icc -qopenmp driver.c -I[INSTALL_DIR]/include \
    [INSTALL_DIR]/lib/libnagc_nag.a \
    [INSTALL_DIR]/rtl/intel64/libifcoremt.a -lpthread -lstdc++
```

共有ライブラリを利用する場合：

```
icc -qopenmp driver.c -I[INSTALL_DIR]/include \
    [INSTALL_DIR]/lib/libnagc_nag.so \
    -L[INSTALL_DIR]/rtl/intel64 -lifcoremt -lpthread
```

共有ライブラリを利用する場合には，環境変数 LD_LIBRARY_PATH を正しく設定し，実行時のリンクが行えるようにしてください.

C シェルの場合：

```
setenv LD_LIBRARY_PATH [INSTALL_DIR]/lib:[INSTALL_DIR]/mkl_intel64_11.3.3/lib
```

または，既存の設定がある場合には次のように拡張します.

```
setenv LD_LIBRARY_PATH \
  [INSTALL_DIR]/lib:[INSTALL_DIR]/mkl_intel64_11.3.3/lib:${LD_LIBRARY_PATH}
```

Bourne シェルの場合：

```
LD_LIBRARY_PATH=[INSTALL_DIR]/lib:[INSTALL_DIR]/mkl_intel64_11.3.3/lib
export LD_LIBRARY_PATH
```

または，既存の設定がある場合には次のように拡張します.

```
LD_LIBRARY_PATH=[INSTALL_DIR]/lib:\
  [INSTALL_DIR]/mkl_intel64_11.3.3/lib:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH
```

場合に依っては（例えば，より新しいバージョンのコンパイラを使用する場合など），その他のパス（例えば，コンパイラのランタイムライブラリなど）を LD_LIBRARY_PATH に設定する必要があるかもしれません.

また，Intel コンパイラの異なるバージョンを使用する場合は，[INSTALL_DIR]/rtl/intel64 ディレクトリに提供される Intel コンパイラのランタイムライブラリをリンクする必要があるかもしれません.

3.1.1. スレッド数の設定

本ライブラリと MKL は，OpenMP を用いてマルチスレッドを実装しています.
実行時に使用されるスレッド数を環境変数 OMP_NUM_THREADS に設定してください.

C シェルの場合：
setenv OMP_NUM_THREADS N

Bourne シェルの場合：
OMP_NUM_THREADS=N
export OMP_NUM_THREADS

N はご利用のスレッド数です. 環境変数 OMP_NUM_THREADS はプログラムの実行毎に再設定することができます. プログラムの異なる部分で，使用するスレッド数を変更したい場合は，NAG ライブラリのチャプター x06 のルーチンがご利用いただけます.

NAG ライブラリと MKL のいくつかのルーチンは，複数レベルの OpenMP 並列処理を持ちます. これらのルーチンは，ユーザー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 ネスト並列処理は無効になっており，最も外側の並列領域だけが N スレッドで実行されます. 内部レベルはアクティブにならず，1 スレッドで実行されます. OpenMP 環境変数 OMP_NESTED の値を確認・設定するか，もしくはチャプター x06 のルーチンを使用して，OpenMP ネスト並列処理の有効／無効の確認・設定を行うことができます. OpenMP ネスト並列処理が有効になっている場合，上位レベルの各スレッドの各並列領域に N 個のスレッドが作成されるため，例えば，2 つのレベルの OpenMP 並列処理がある場合，合計 N * N スレッドになります. ネスト並列処理では，各レベルで必要なスレッド数を，環境変数 OMP_NUM_THREADS にカンマ区切りで指定することができます.

C シェルの場合：
setenv OMP_NUM_THREADS N, P

Bourne シェルの場合：
OMP_NUM_THREADS=N, P
export OMP_NUM_THREADS

この設定例では，第 1 レベルの並列処理に対して N 個のスレッドが生成され，内部レベルの並列処理に対して P 個のスレッドが生成されます.

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毎，またベンダーライブラリ毎に異なります．通常は，１もしくはシステムで使用可能な最大コア数に等しくなります．特に後者では，システムを他のユーザーと共有している場合や，自分のアプリケーション内で複数レベルの並列処理を実行している場合に問題となる可能性があります．従って，OMP_NUM_THREADS は明示的に設定することをお勧めします.

一般的に，推奨されるスレッドの最大数は，ご利用の共有メモリシステムの物理コア数です．ただし，殆どの 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います．この機能は１つの物理コアが同時に２つのスレッドをサポートすることを可能にします（従って，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）．この機能が有益かどうかは，使用するアルゴリズムや問題のサイズに依存します．従って，自身のアプリケーションにとってこの機能が有益かどうかは，追加の論理コアを使用する場合と使用しない場合でベンチマークを取り判断することをお勧めします．これは，使用するスレッド数を OMP_NUM_THREADS に設定するだけで簡単に実現できます．ハイパースレッディングの完全な無効化は，通常，起動時にシステムの BIOS 設定で行うことができます.

3.2. Example プログラム

提供される Example 結果は，インストールノートの「2.2. 開発環境」に記載されている環境で生成されています．Example プログラムの実行結果は，異なる環境下（例えば，異なる C コンパイラ，異なるコンパイラライブラリ，異なる BLAS または LAPACK ルーチンなど）で若干異なる場合があります．そのような違いが顕著な計算結果としては，固有ベクトル（スカラー（多くの場合 -1）倍の違い），反復回数や関数評価，残差（その他マシン精度と同じくらい小さい量）などがあげられます．

提供される Example 結果は NAG スタティックライブラリ libnagc_mkl.a（MKL 提供の BLAS／LAPACK ルーチンを使用）を用いて算出されています．NAG 提供の BLAS／LAPACK ルーチンを使用した場合，結果が僅かに異なるかもしれません．

Example プログラムは本ライブラリが想定する動作環境に適した状態で提供されます．そのため，ライブラリマニュアルに記載されている Example プログラムに比べて，その内容が若干異なる場合があります．

以下のスクリプトを用いて Example プログラムを簡単に利用することができます．
（これらのスクリプトは，[INSTALL_DIR]/scripts ディレクトリに提供されます．）

- nagc_example_mkl
  NAG スタティックライブラリ libnagc_mkl.a および本製品で提供される MKL をリンクします．
- nagc_example_shar_mkl
  NAG 共有ライブラリ libnagc_mkl.so および本製品で提供される MKL をリンクします．
- nagc_example
  NAG スタティックライブラリ libnagc_nag.a をリンクします．
- nagc_example_shar
  NAG 共有ライブラリ libnagc_nag.so をリンクします．

これらのスクリプトは，Example プログラムのソースファイル（必要に応じて，データファイル，オプションファイルその他）をカレントディレクトリにコピーして，コンパイル・リンク・実行を行います．

ご利用の NAG ライブラリルーチンの名前と OpenMP スレッド数をスクリプトの引数に指定してください.
例）
nagc_example_mkl e04ucc 4

この例では, e04ucce.c（ソースファイル), e04ucce.d（データファイル), e04ucce.opt（オプションファイル）をカレントディレクトリにコピーして, コンパイル・リンク, および 4 OpenMP スレッドで実行を行い, e04ucce.r（結果ファイル）を生成します.

3.3. データ型

NAG データ型 Integer と Pointer は, 本ライブラリで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 の Example プログラムから得ることもできます. その他の NAG データ型の情報はライブラリマニュアル（「5. ドキュメント」参照）の "How to Use the NAG Library and its Documentation" ドキュメントをご参照ください.

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, ライブラリルーチン a00aac の Example プログラムをコンパイル・リンク・実行することにより確認することができます. この時, スクリプト nagc_example* を引数 a00aac と共に用いれば, Example プログラムのコンパイル・リンク・実行を容易に行うことができます（「3.2. Example プログラム」参照). ライブラリルーチン a00aac はライブラリの詳細（タイトル, 製品コード, 使用されるコンパイラおよび精度, バージョン（Mark）など）を出力します.

4. ルーチン固有の情報

本ライブラリルーチン固有の情報を以下に示します.

a. OpenMP 並列領域からユーザー関数を呼び出すルーチン

以下の NAG ルーチンはルーチン内の OpenMP 並列領域からユーザー関数を呼び出します.

e05ucc e05usc f01elc f01emc f01flc f01fmc f01jbc f01jcc f01kbc f01kcc

従って, 本ライブラリの製造に用いられたコンパイラと同じコンパイラを使用する限り, ユーザー関数で orphaned OpenMP 指示文を使うことができます. また, ユーザー用のワークスペース配列 IUSER と RUSER もスレッドセーフである必要があります. これらの配列は読み取り専用のデータをユーザー関数に与えるためにだけ使用するのがベストです.

b. c06

以下の NAG ルーチンは可能な限り本製品で提供される MKL の
Intel Discrete Fourier Transforms Interface (DFTI) ルーチンを利用します.

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

c. f06, f07, f08, f16

チャプター f06, f07, f08, f16 では, BLAS／LAPACK 由来のルーチンに対して別個のルーチン名が用意されています. これらのルーチン名については, 関係する Chapter Introduction をご参照ください. パフォーマンス面からは, NAG スタイルの名前よりも BLAS／LAPACK スタイルの名前でルーチンをご利用ください.

多くの LAPACK ルーチンは "workspace query" メカニズムを利用します. ルーチン呼び出し側にどれだけのワークスペースが必要であるかを問い合わせるメカニズムですが, NAG 提供の LAPACK と MKL 提供の LAPACK ではこのワークスペースのサイズが異なる場合があるので注意してください.

MKL を利用するバージョンの NAG ライブラリでは, BLAS／LAPACK ルーチンは MKL 提供のものが使われます. ただし, 以下のルーチンは NAG 提供のものが使われます.

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		

MKL を利用するバージョンの NAG ライブラリでは,
以下の NAG ルーチンは MKL から LAPACK ルーチンを呼び出すためのラッパーです.

nag_dgetrf/f07adc	nag_dgetrs/f07aec	nag_zgetrf/f07arc	nag_zgetrs/f07asc
nag_dgbtrs/f07bec	nag_zgbtrs/f07bsc	nag_dpotrf/f07fdc	nag_dpotrs/f07fec
nag_zpotrf/f07frc	nag_zpotrs/f07fsc	nag_dpptrs/f07gec	nag_zpptrs/f07gsc
nag_dpbtrs/f07hec	nag_zpbtrs/f07hsc	nag_dgeqrf/f08aec	nag_dormqr/f08agc
nag_zgeqrf/f08asc	nag_zunmqr/f08auc	nag_dsytrd/f08fec	nag_zhetrd/f08fsc
nag_dsptrd/f08gec	nag_dopgtr/f08gfc	nag_zhptrd/f08gsc	nag_zupgtr/f08gtc
nag_dsteqr/f08jec	nag_zsteqr/f08jsc	nag_dgebrd/f08kec	nag_zgebrd/f08ksc
nag_dbdsqr/f08mec	nag_zbdsqr/f08msc		

d. s10 - s21

これらのチャプターの関数の動作は, ライブラリ実装毎に異なります.

一般的な詳細はライブラリマニュアルをご参照ください.
本ライブラリ固有の値を以下に示します.

s10aac E_1 = 1.8715e+1
s10abc E_1 = 7.080e+2
s10acc E_1 = 7.080e+2

s13aac x_hi = 7.083e+2
s13acc x_hi = 1.0e+16
s13adc x_hi = 1.0e+17

s14aac fail.code = NE_REAL_ARG_GT if x > 1.70e+2
        fail.code = NE_REAL_ARG_LT if x < -1.70e+2
        fail.code = NE_REAL_ARG_TOO_SMALL if abs(x) < 2.23e-308
s14abc fail.code = NE_REAL_ARG_GT if x > x_big = 2.55e+305

s15adc x_hi = 2.65e+1
s15aec x_hi = 2.65e+1
s15agc fail.code = NW_HI if x >= 2.53e+307
        fail.code = NW_REAL if 4.74e+7 <= x < 2.53e+307
        fail.code = NW_NEG if x < -2.66e+1

s17acc fail.code = NE_REAL_ARG_GT if x > 1.0e+16
s17adc fail.code = NE_REAL_ARG_GT if x > 1.0e+16
        fail.code = NE_REAL_ARG_TOO_SMALL if 0 < x <= 2.23e-308
s17aec fail.code = NE_REAL_ARG_GT if abs(x) > 1.0e+16
s17afc fail.code = NE_REAL_ARG_GT if abs(x) > 1.0e+16
s17agc fail.code = NE_REAL_ARG_GT if x > 1.038e+2
        fail.code = NE_REAL_ARG_LT if x < -5.7e+10
s17ahc fail.code = NE_REAL_ARG_GT if x > 1.041e+2
        fail.code = NE_REAL_ARG_LT if x < -5.7e+10
s17ajc fail.code = NE_REAL_ARG_GT if x > 1.041e+2

```
        fail.code = NE_REAL_ARG_LT if x < -1.9e+9
s17akc  fail.code = NE_REAL_ARG_GT if x > 1.041e+2
        fail.code = NE_REAL_ARG_LT if x < -1.9e+9
s17dcc  fail.code = NE_OVERFLOW_LIKELY if abs(z) < 3.92223e-305
        fail.code = NW_SOME_PRECISION_LOSS if abs(z) or fnu+n-1 > 3.27679e+4
        fail.code = NE_TOTAL_PRECISION_LOSS if abs(z) or fnu+n-1 > 1.07374e+9
s17dec  fail.code = NE_OVERFLOW_LIKELY if AIMAG(z) > 7.00921e+2
        fail.code = NW_SOME_PRECISION_LOSS if abs(z) or fnu+n-1 > 3.27679e+4
        fail.code = NE_TOTAL_PRECISION_LOSS if abs(z) or fnu+n-1 > 1.07374e+9
s17dgc  fail.code = NW_SOME_PRECISION_LOSS if abs(z) > 1.02399e+3
        fail.code = NE_TOTAL_PRECISION_LOSS if abs(z) > 1.04857e+6
s17dhc  fail.code = NW_SOME_PRECISION_LOSS if abs(z) > 1.02399e+3
        fail.code = NE_TOTAL_PRECISION_LOSS if abs(z) > 1.04857e+6
s17dlc  fail.code = NE_OVERFLOW_LIKELY if abs(z) < 3.92223e-305
        fail.code = NW_SOME_PRECISION_LOSS if abs(z) or fnu+n-1 > 3.27679e+4
        fail.code = NE_TOTAL_PRECISION_LOSS if abs(z) or fnu+n-1 > 1.07374e+9

s18adc  fail.code = NE_REAL_ARG_TOO_SMALL if 0 < x <= 2.23e-308
s18aec  fail.code = NE_REAL_ARG_GT if abs(x) > 7.116e+2
s18afc  fail.code = NE_REAL_ARG_GT if abs(x) > 7.116e+2
s18dcc  fail.code = NE_OVERFLOW_LIKELY if abs(z) < 3.92223e-305
        fail.code = NW_SOME_PRECISION_LOSS if abs(z) or fnu+n-1 > 3.27679e+4
        fail.code = NE_TOTAL_PRECISION_LOSS if abs(z) or fnu+n-1 > 1.07374e+9
s18dec  fail.code = NE_OVERFLOW_LIKELY if REAL(z) > 7.00921e+2
        fail.code = NW_SOME_PRECISION_LOSS if abs(z) or fnu+n-1 > 3.27679e+4
        fail.code = NE_TOTAL_PRECISION_LOSS if abs(z) or fnu+n-1 > 1.07374e+9

s19aac  fail.code = NE_REAL_ARG_GT if abs(x) >= 5.04818e+1
s19abc  fail.code = NE_REAL_ARG_GT if abs(x) >= 5.04818e+1
s19acc  fail.code = NE_REAL_ARG_GT if x > 9.9726e+2
s19adc  fail.code = NE_REAL_ARG_GT if x > 9.9726e+2

s21bcc  fail.code = NE_REAL_ARG_LT if an argument < 1.583e-205
        fail.code = NE_REAL_ARG_GE if an argument >= 3.765e+202
s21bdc  fail.code = NE_REAL_ARG_LT if an argument < 2.813e-103
        fail.code = NE_REAL_ARG_GT if an argument >= 1.407e+102
```

e. x01

以下の数学定数がヘッダーファイル nagx01.h に提供されます.

x01aac (pi) = 3.1415926535897932
x01abc (gamma) = 0.5772156649015328

f. x02

以下のマシン定数がヘッダーファイル nagx02.h に提供されます.

浮動小数点演算の基本的なパラメーター:

x02bhc = 2
x02bjc = 53
x02bkc = -1021
x02blc = 1024

浮動小数点演算の派生的なパラメーター:

x02ajc = 1.11022302462516e-16
x02akc = 2.22507385850721e-308
x02alc = 1.79769313486231e+308
x02amc = 2.22507385850721e-308
x02anc = 2.22507385850721e-308

コンピューター環境のその他のパラメーター:

x02ahc = 1.42724769270596e+45
x02bbc = 9223372036854775807
x02bec = 15

g. x06

本チャプターのルーチンは, 本ライブラリの MKL スレッドの動作も変更します.

5. ドキュメント

ライブラリマニュアルは本製品の一部として提供されます.
また, NAG のウェブサイトからダウンロードすることもできます.
ライブラリマニュアルの最新版は以下のウェブサイトをご参照ください.

http://www.nag.co.uk/content/nag-c-library-manual

ライブラリマニュアルは以下の形式で提供されます.

- HTML5 - HTML／MathML マニュアル（各ドキュメントの PDF 版へのリンクを含む）
- PDF - PDF マニュアル（PDF のしおり, または HTML 目次ファイルから閲覧する）

これらの形式に対して, 以下の目次ファイルが提供されます.

nagdoc_cl26/html/frontmatter/manconts.html
nagdoc_cl26/pdf/frontmatter/manconts.pdf
nagdoc_cl26/pdf/frontmatter/manconts.html

また, これらの目次ファイルへのリンクをまとめたマスター目次ファイルが提供されます.

nagdoc_cl26/index.html

各形式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.

http://www.nag.co.uk/numeric/cl/nagdoc_cl26/html/genint/essint.html

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（英語版）

MKL の詳細については, 以下の Intel 社のウェブサイトをご参照ください.

https://software.intel.com/intel-mkl

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（CSL6I26DDL）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．

http://www.nag-j.co.jp/　（日本）
http://www.nag.co.uk/　（英国本社）
http://www.nag.com/　（米国）