NAG Fortran Library

ライブラリとドキュメントの使い方

目次

1　ライブラリの識別

定期的にライブラリの新たな **Mark**（バージョン）がリリースされます．具体的には新たなルーチンの追加，既存ルーチンの修正，改良，さらに改良版導入に伴うルーチンの削除といった内容が伴います．ただし，Mark 26.1 のようなポイント・リリースにはルーチンの削除はありません．

ユーザーはライブラリの**どの実装，どの演算精度，どの Mark 及びリビジョン**を使用しているかを知っていなくてはなりません．これらの情報を確認するには，ライブラリルーチン a00aaf を呼び出すプログラムを実行します．

次に示すのはプログラムの一例です．

```
Use nag_library, Only: a00aaf
Call a00aaf
End
```

これとは別に，実装と共に提供される `nag_example` スクリプトを使って a00aaf の Example プログラムを実行させることもできます．（詳細はユーザーノートをご参照ください．）

出力例は次のようになります．

```
*** Start of NAG Library implementation details ***

Implementation title: Linux, 64-bit, NAG Fortran (32-bit integers)
           Precision: FORTRAN double precision
        Product Code: FLL6A2619L
                Mark: 26.1 (self-contained)

 *** End of NAG Library implementation details ***
```

2　NAG ライブラリルーチンの探し方

NAG ライブラリの利用経験を問わず，以下の利用要領が推奨されます．

(a) この **How to Use the NAG Library and its Documentation**（ライブラリとドキュメントの使い方）を読む．
(b) **Keyword and GAMS Search** を用いて，適切な Chapter（チャプター）や Routine（ルーチン）を選択する．
(c) 関連する **Chapter Introduction**（チャプターイントロダクション）を読む．
(d) ルーチンを選択し **Routine Document**（ルーチンドキュメント）を読む．ルーチンがニーズに合致しなかった場合にはステップ (b) に戻る．
(e) ご利用のライブラリ製品の**ユーザーノート**（Users' Note）を読む．
（ユーザーノートには，ご利用のライブラリのリンク方法が記載されています．）
(f) 利用方法についてのローカルなドキュメント（サイトで用意されているもの等）があれば，それを読む．
(g) 該当ルーチンの Example プログラム（セクション 4.5 参照）を使用してみる．

この段階でユーザープログラムへのライブラリルーチンのコーディングは終わり，コンパイル，実行を試みる段階にきているはずです．もちろん問題が発生したり，結果に確信が持てないような場合には，再度関連するドキュメントを参照する必要があります．

ライブラリの利用経験に応じて (a) から (g) までのステップのいくつかはスキップできますが，内容変更の可能性のある以下のドキュメントは常に最新の状態に保つことが望まれます．

- **How to Use the NAG Library and its Documentation**
- **Chapter Introduction**（チャプターイントロダクション）
- **Routine Document**（ルーチンドキュメント）
- 実装固有の**ユーザーノート**（Users' Note）

3 ライブラリの使い方

3.1 ライブラリの構成

NAG ライブラリは，数値計算や統計解析の分野の問題を解くための様々な **Routine**（ルーチン）の集合体です．

ライブラリは **Chapter**（チャプター）に区分されており，その各々は数値計算や統計解析の個別の分野に対応しています．各チャプターには 3 文字からなる名称とタイトルが付けられています．

（例） D01 - Quadrature（数値積分）

チャプター H と S は例外で 1 文字の名称からなります．これらのチャプターの構成と名称は ACM 修正版 SHARE 分類インデックス（ACM (1960–1976) 参照）に基づいています．

ライブラリルーチンにはチャプターの名前から始まる 6 文字の名称が付けられています．

（例） c06pcf

2 番目と 3 番目の文字は英字ではなく**数字**である点に注意してください（英字の O ではなく数字の 0 です）．ルーチン名の末尾の文字はほとんどが 'f' となります．ただしチャプター D03 と E04 の中には末尾の文字が 'f' ではなく 'a' のルーチンが含まれています．'a' ルーチンは常に 'f' ルーチンとペアになっています．'a' ルーチンはマルチスレッド環境でも安全に使用できますが，その他の機能面では 'f' ルーチンと差はありません．

チャプター F06（線形代数サポートルーチン）には Basic Linear Algebra Subprograms, BLAS（Dongarra *et al.* (1988)，Dongarra *et al.* (1990)）が含まれています．その各々には NAG スタイルの名称の他に実際の BLAS 名も付けられています（例：f06paf（dgemv））．括弧内の名称は等価な倍精度 BLAS 名を意味しています．チャプター F16 には BLAS 技術フォーラム（The BLAS Technical Forum Standard (2001)，Blackford *et al.* (2002)）で指定されたルーチンのいくつかと標準にはない整数ベクトルのルーチンが含まれています．チャプター F16 のルーチンのいくつかは NAG スタイルの名称と BLAS 名の両方を持っています．チャプター F07（線形方程式（LAPACK））とチャプター F08（最小二乗／固有値問題（LAPACK））には LAPACK プロジェクトに由来するルーチンが含まれています（Anderson *et al.* (1999)）．また，チャプター F01（行列の演算（逆行列を含む））には LAPACK プロジェクトに由来する格納形式変換ルーチンが含まれています．BLAS と同様，これら

のルーチンには NAG スタイルの名称と共に LAPACK 名が付けられています（例：f07adf（dgetrf））．これら別名に関する詳細については該当するチャプターのイントロダクション部（Chapter Introduction）をご参照ください．

何社かのハードウェアベンダーから提供されているマシン固有の BLAS/LAPACK ルーチンを利用できるようにするためには，プログラム中で NAG スタイルの名称（例えば，f06paf や f07adf 等）ではなく BLAS 名や LAPACK 名（例えば，dgemv や dgetrf 等）をなるべく使用するようにしてください．

3.1.1 実験的なルーチン

ライブラリルーチンには，Experimental（実験的）というカテゴリに分類されるルーチンがあります．ルーチンが実験的である場合は，該当のルーチンドキュメントの一番上にその旨の Note（注意書き）があります．

次の場合に，ルーチンは実験的として分類されます．

(a) ルーチンの引数および／または機能が，ライブラリの Mark 間で変わる可能性がある場合．
これらのルーチンは，そのような変更を最小限に抑えるように設計されています．

(b) ルーチンの複雑さのため，またはルーチンスイートの一部分であるため，包括的なテストが難しい場合．
これらのルーチンは，通常のルーチンと同じテストおよびレビュープロセスを経ていますが，使用するときには注意が必要です．

実験的なルーチンは，後に実験的でなくなった時点で通常のルーチンと同様に引数の仕様が固定します．

3.1.2 ライブラリルーチンの長い名前

ライブラリルーチンは 6 文字から成る短い名前に加えて，**nagf_**で始まり複数の単語を下線でつないだ長い名前を持っています．長い名前は各ルーチンがその関連性に伴いグループ化されるように命名されています．

各ルーチンの長い名前は各チャプターの Chapter Contents に記載されています．長い名前の二番目の単語は各チャプターに対応しており，例えば，チャプター D01（Quadrature（数値積分））のルーチンはすべて**nagf_quad_**で始まる長い名前を持っています．

なお，長い名前の二番目の単語はチャプター毎に異なりますが，例外としてチャプター F07 と F08 は同じ単語（**lapack**）を共有しています．

BLAS/LAPACK ルーチンの長い名前（例えば，nagf_blas_dgemm）は，ハードウェアベンダー提供のマシン固有の BLAS/LAPACK には使えません．セクション 3.1 でも述べたように，パフォーマンス面からは本来の BLAS/LAPACK 名（例えば，dgemm）の使用が推奨されます．

ペアとなっている ‘a’ ルーチンと ‘f’ ルーチンの長い名前は同じです．ただし ‘f’ ルーチンの長い名前の末尾には **old**（‘f’ ルーチンが ‘a’ ルーチンよりも古いことを示す）が追加されています．

これらの長い名前は NAG ライブラリのインターフェースブロックモジュールの中で別名を付けること（エイリアシング）によって実装されています．従って，nag_library モジュールを Use することによって Fortran プログラムからこれらの長い名前が利用可能となります．

セクション 3.3.1 において，ルーチンの別名でのご利用やルーチンインターフェースの単純化についてのアドバイスを得ることができます．

3.2 一般的なアドバイス

NAG ライブラリルーチンは，与えられたデータとは無関係に常に有意義な結果を返すことを**保証するものではありません**．次の点に関する**注意**や**配慮**が必要です．

(a) 問題の定式化
(b) ライブラリルーチンを使用したプログラミング
(c) 結果の評価

マニュアルの Foreword ドキュメントに (a) と (c) に関する更なる記述があります．(b) と (c) については本ドキュメントの残りの部分で説明します．

3.3 プログラミングに関するアドバイス

ライブラリとそのドキュメントはユーザーがルーチンを呼び出すプログラムを Fortran で書けることを前提に作成されています．(その他の言語からの利用に関してはセクション 3.10 をご参照ください．)

ライブラリルーチンの呼び出しをプログラミングする際には，ルーチンドキュメント，特に **Arguments**（引数）部の記述を注意して読んでください．この記述部に，ルーチンの呼び出し（entry）時にどの引数に値がセットされていなくてはならないか，また復帰（exit）時にどの引数に有用な情報が含まれているかについて明確な説明があります．更なる詳細はセクション 4.3 をご参照ください．

ライブラリを使用する上で最も良く見られるプログラミングエラーのタイプは次の 2 つです．

- ライブラリルーチンを呼び出す際の引数不正
- 単精度プログラムからのライブラリの呼び出し

`nag_library` モジュールはこれらのエラーを検知して防ぐために役立ちます．これを用いれば（Use すれば），不正な引数型はコンパイル時に検知されるようになります．また，更に `KIND=nag_wp` を用いることで，実数と複素数の変数は精度の面でライブラリとの一貫性を保つことができます．

従って，ライブラリルーチンの呼び出しによってシステムからの（あるいはライブラリ内からの）予期せぬエラーメッセージが生じた場合には，次の点を**チェック**してください．

- **INTENT 属性が異なる複数の配列引数に同一の実配列が渡されていないか？**
- **配列引数の寸法は正しいか？**

ユーザー自身のプログラム単位や COMMON ブロックに対し NAG タイプの名称を使わないようにしてください．一般論として，3 文字からなる NAG のチャプター名を含む名称をそれらの中で使用しないでください．ライブラリで用いられている補助ルーチン名や COMMON ブロック名と競合する恐れがあります．

3.3.1 ルーチン名の代替

Fortran プログラムからライブラリルーチンを別の名前で利用することができます．これは，ライブラリルーチンを呼び出しているプログラムの頭の方で，'Use nag_library' 文を通して行うことができます．例えば，ラ

イブラリルーチン名 s17aef の代わりに ‘BesselJ0’ という名前を使いたい場合，以下の行を，

```
Use nag_library, Only: s17aef
```

次の行で置き換えます．

```
Use nag_library, Only: BesselJ0 => s17aef
```

これにより，‘BesselJ0’ という名前でライブラリルーチン s17aef を利用できるようになります．

ライブラリルーチンを他の環境から利用する場合にも，多くの環境がルーチン名に別名を付ける方法を提供しています．

複雑なインターフェースのライブラリルーチンをご利用の場合には，一部の引数だけを必要とし，その他の引数は値が常に一定である（または参照されない）ことが多いでしょう．そのような場合，ライブラリルーチンのラッパーを書くことで，より単純なインターフェース（および適当な代替名）でライブラリルーチンを利用することができます．例えば，求根と中間出力なしで硬い連立常微分方程式を解く場合，d02ejf の複雑なインターフェースに対して以下のようなラッパーを作成することができます．

```
Subroutine BDFsolve(xend,y)
Use nag_library, Only: nag_wp, d02ejf, d02ejw, d02ejx, d02ejy
Real(kind=nag_wp) :: xend, y(:)
Real(kind=nag_wp) :: tol, xstart
INTEGER           :: ifail, iw, n
Character         :: relabs
Real(kind=nag_wp), Allocatable :: w(:)

n = Size(y)
tol = 1.0e-3_nag_wp
relabs = 'M'
iw = (12+n)*n + 50
Allocate(w(iw))
ifail = 0
xstart = 0.0_nag_wp
Call d02ejf(xstart,xend,n,y,fcn,d02ejy,tol,relabs,d02ejx, &
            d02ejw,w,iw,ifail)
Return
End Subroutine BDFsolve
```

そして，次のように単純なインターフェースでルーチンを呼び出すことができます．

```
Call BDFsolve(xend,y)
```

3.3.2 NAG Fortran 環境

NAG ライブラリの環境は `nag_library` モジュールに定義されています．NAG 定義の定数（例えば，nag_wp）を利用するためには，このモジュールを Use する必要があります．また，ライブラリルーチンのイ

ンターフェースブロックを利用するためにも，このモジュールを Use することが推奨されます．

`nag_library.mod` の格納場所はインストールに依存しています．詳細はご利用の製品のユーザーノートをご参照ください．

3.3.3 ダイレクト／リバースコミュニケーションルーチン

ユーザー提供の関数を必要とするライブラリルーチンは，ダイレクトコミュニケーションとリバースコミュニケーションのどちらかに分類されます．

ダイレクトコミュニケーションルーチンでは，ユーザー提供のサブルーチンを実引数として渡します．ユーザー提供のサブルーチンは，該当のルーチンドキュメントに記載される引用仕様に従って書かなくてはなりません．大抵の場合，ダイレクトコミュニケーションの方が簡単かつ便利です．しかし時に，この用法がネックになる場合があります．

(i) 決められたサブルーチンの仕様ではライブラリルーチンと呼び出し元プログラムの間で必要な情報をやり取りできない場合
(ii) 他のコンピューター言語からダイレクトコミュニケーションルーチンを呼び出す際に，そのコンピューター言語がライブラリと完全に互換性のある形で手続引数をサポートしていない場合

これらの制限はリバースコミュニケーションルーチンを用いることで取り除かれます．リバースコミュニケーションルーチンは，1 回の呼び出しで解を得るのではなく，解法プロセスを 1 ステップだけ実行し適切なフラグ（irevcm）をセットして呼び出し元プログラムに戻ります．プロセスが終了したかどうか，もしくは新しい情報が必要かどうかは irevcm の値で判断します．新しい情報が必要な場合，リバースコミュニケーションルーチンを再度呼び出す前に必要な情報を計算しなければなりません．要するに，解法プロセスの反復ループをユーザー側で行うことになります．一般的に，リバースコミュニケーションルーチンはダイレクトコミュニケーションルーチンよりも使用が複雑ですが，関数の評価に対してより大きな柔軟性を持ちます．

3.4 エラー処理と引数 ifail

3.4.1 エラー，失敗，警告の条件

ここでは，ライブラリルーチンで検出可能なエラー，失敗，警告の条件を説明します．これらは，ライブラリルーチンの作成者により予め想定済みの条件であって，コンパイラシステムにより検出される実行時エラー（例えば，オーバーフローや初期化忘れ等）とは別のものであることに注意してください．

以後，このドキュメント内で「エラー」という単語はルーチンが検出するエラー，失敗，警告のすべてを意味します．エラーは主に以下の 3 種類に分類されます．

(i) ルーチンが呼び出された時点で与えられた引数の値が範囲外の場合．これは計算を始める前に意味の無い値や利用可能で無い値が引数として与えられた場合のエラーです．
(ii) 計算が行われている途中で求められる結果が得られないと判断された場合．例えば，逆行列を求める計算で行列が特異であると判断されたような場合です．
(iii) 計算は終了したものの結果の信頼性が危ぶまれる場合．このような場合は警告が返されます．例えば，最適化ルーチンは局所的な最小値が求まった事を保証できない場合，警告を返します．

ライブラリはこれら **3** 種類のエラーを同様に処理します.

各々のエラーにはエラー番号が対応しています．動的メモリ割当て（セクション 3.7 参照）とライセンスチェック（セクション 3.8 参照）に関するエラー番号は，全てのライブラリルーチンで共通しているため，個々のルーチンドキュメントには記載されていません．また，最近新たに追加されたルーチンでは予期しないエラーに対しても共通の番号を用います（セクション 3.9 参照）．その他すべてのエラー番号は（エラーの説明と共に）ルーチンドキュメントのセクション 6（Error Indicators and Warnings）に列記されています．特に説明が無い限り，エラーチェックの順番はエラー番号の順番とは対応していません．（例えば，あるエラーが検出された場合に他のエラーのチェックが既に行われたかどうかは明らかではありません.）

3.4.2 ifail 引数

多くのライブラリルーチンは ifail と呼ばれる引数を持っています．この ifail 引数はライブラリのエラー処理に関連するものです.（また，いくつかのルーチンでは，エラーメッセージと警告メッセージの出力のコントロールに関連します.）

ifail には以下の **2** つの目的があります.

(i) エラーが検出された際にライブラリルーチンがどのような動作を行うべきかの指定
(ii) ルーチンを呼び出した結果（エラーが発生したかどうか）の報告

目的 (i) に関しては，ルーチンを呼び出す前に **ifail** に値を設定して行います．ifail は目的 (ii) に即して出力引数としても使われるため（定数表現では無く）変数として与えなくてはいけません.

ifail に設定する値は 0（**hard fail** オプション），1 または −1（**soft fail** オプション）のいずれかです．エラーが検出されなかった場合（計算が正常終了した場合）ifail は 0 を返し，呼出し元のプログラムは通常通り続行されます．エラーが検出された場合には，hard fail オプションか soft fail オプションかの指定によって異なる動作となります．ライブラリルーチンを呼び出す時に ifail に −1，0，1 以外の値を設定した場合，ifail にはデフォルト値 1 が用いられます.

3.4.3 Hard Fail オプション

ライブラリルーチンを呼び出す前に ifail を 0 に設定した場合，エラーが検出されるとプログラムは終了します．プログラムの終了前に以下のエラーメッセージが出力されます.

```
** ABNORMAL EXIT from NAG Library routine XXXXXX: ifail = n
** NAG hard failure - execution terminated
```

ここで XXXXXX はルーチン名で n がエラー番号です．エラー番号の説明は XXXXXX ルーチンドキュメントのセクション 6 に記載されています.

また，多くのルーチンでは上記のメッセージの直前に，より詳細なエラーメッセージが出力されます.

hard fail オプションは，ライブラリルーチンの呼び出しに失敗した（エラーが発生した）際に，プログラムを続行させたく無い場合に指定してください．エラーが発生しても，呼出し元のプログラムを続行させたい場合は，hard fail オプションを指定しないようにしてください.

3.4.4 Soft Fail オプション

ライブラリルーチンを呼び出す前に ifail を 1 または −1 に設定した場合，エラーが検出されると，ifail に適切なエラー番号が設定され，（計算はそれ以上行われずに）実行が呼出し元のプログラムに戻されます．なお，ifail に不正な値が設定された場合，ifail にはデフォルト値 1 が用いられることに注意してください．

ifail に 1 を指定した場合（**silent exit**），エラーメッセージは出力されません．

ifail に −1 を指定した場合（**noisy exit**），以下の様なエラーメッセージが出力された後に，実行が呼出し元のプログラムに戻されます．

```
** ABNORMAL EXIT from NAG Library routine XXXXXX: ifail = n
** NAG soft failure - control returned
```

ここで XXXXXX はルーチン名で n がエラー番号です．エラー番号の説明は XXXXXX ルーチンドキュメントのセクション 6 に記載されています．

また，多くのルーチンでは上記のメッセージの直前に，より詳細なエラーメッセージが出力されます．

soft fail オプションを指定した場合には，ルーチンからの復帰後に ifail の値をチェックすることが重要です． ifail が 0 以外の値であった場合は何らかのエラーを意味しますので，呼出し元のプログラムで適切な処理を行う必要があります．簡単な例で言えば，戻された ifail の値を（適切な説明と共に）出力してからプログラムを終了するといった処理です．ルーチンドキュメントのセクション 9 にある多くの Example プログラムには ifail に対するこの様な処理が含まれています．（ルーチンからのエラー復帰後に）より積極的に呼出し元のプログラムを続行したい場合には，実行を再開するのに**相応しい**箇所への分岐を持つようなプログラミングが基本となるでしょう．

soft fail オプションでは，ライブラリルーチンで検出されたエラーに対する処理はユーザー側に委ねられますので，**適切に**利用すれば，hard fail オプションに比べてより柔軟性の高いエラー処理が可能となります．特に以下の 2 つの場合にこの柔軟性が有用です．

(i) エラーや計算経過に関する追加情報が他の引数を通じて得られる場合

(ii) いくつかのルーチンでは部分的な成功があり得ます．例えば，（すべての条件が満たされているわけではないが）有望な計算結果が得られた場合，ルーチンはその計算結果と共に警告を返します．このような場合，ルーチンドキュメントのセクション 6（及び，その他の部分）のアドバイスに基づき，ユーザーはこの計算結果（部分的な成功）をある目的では適切であると見なし利用する事ができます．

3.4.5 NAG エラーメッセージの構造

ルーチンドキュメントのエラーメッセージの説明に出てくる表記 ⟨*value*⟩ はプレースホルダーです．エラーメッセージが実際に表示される時に，具体的な情報（変数値や引数名など）に置き換わります．

3.4.6 旧来のエラー処理

（主に Mark 7 および Mark 8 から導入された）いくつかのルーチンは ifail を特殊な方法で利用し，エラーメッセージと警告メッセージの出力方法の指定を行います（チャプター X04 参照）．これらのルーチンでは ifail を

$100c + 10b + a$ 形式の整数として扱います．ここで a と b は 0 か 1 かで，以下の意味を表します．

$a = 0$: hard failure　　$a = 1$: soft failure
$b = 0$: silent exit　　$b = 1$: noisy exit

詳細は各ルーチンドキュメントに記載されています．

3.5　ライブラリの入出力

ほとんどのライブラリルーチンはエラーメッセージ出力を除き，外部ファイルへの出力は行いません．すべてのエラーメッセージは論理**エラーメッセージ**装置（logical error message unit）に対して出力されます．この装置番号（多くの実装においてはデフォルトで 6 にセットされます）は，ライブラリルーチン x04aaf を用いて変更できます．

ある種のライブラリルーチンはオプションとして最終結果を，あるいは計算過程のモニタリングのために中間結果を出力することがあります．一般的にエラーメッセージ以外の出力は論理**アドバイスメッセージ**装置（logical advisory message unit）に書き出されます．この装置番号（多くの実装においては，これもデフォルトで 6 にセットされます）は，ライブラリルーチン x04abf を用いて変更できます．論理的にはエラーメッセージ装置とは異なるわけですが，実際問題としては両者の装置番号は同一のことが多いと言えます．オプション設定機能を持つルーチンのスイートには通常この装置番号を直接指定できるオプションが用意されています．

ライブラリからのすべての出力は適切にフォーマットされています．

外部ファイルからの入力を行うライブラリルーチンはわずかしかありません．そのようなルーチンはチャプター E04, E05, H に存在します．外部ファイルに対する装置番号はルーチンの引数で指定する形となっており，またすべての入力はフォーマットされているものとします．

関連する装置番号と所定の外部ファイルとのくくりつけは呼出し元のプログラムにおける OPEN 文，または OS コマンドにより行われていなくてはなりません．

3.6　外部手続き引数としての補助ルーチン

ドキュメント化され，ユーザーからの呼び出しを想定したライブラリルーチンに加え，ライブラリには数多くの補助ルーチンが収納されています．

一般的には，これらの補助ルーチンについて気にする必要はありませんが，ライブラリルーチンを呼び出す実行プログラムのメモリマップを調べたりするとその存在に気が付かれるかも知れません．唯一の例外は，ある種のライブラリルーチンを呼び出す場合に，これらの補助ルーチンの名前を外部手続き引数として指定しなくてはならない，あるいは指定できるケースがあるということです．必要な詳細はルーチンドキュメントに記載されています．そのような場合に必要となるのはルーチン名のみです．（補助ルーチンの）引数リストの詳細について**知っている必要はありません**．

NAG の補助ルーチンには，それが関係するドキュメント化されたルーチンの名称に似た名前が付けられていますが，末尾の文字は ‘Z’ や ‘Y’ 等となっています．例えば，

g13afz は g13aff から呼び出される補助ルーチンです．

一部のチャプターに含まれる補助ルーチンの場合，その名称はチャプターの第 2，第 3 の文字に 50 を加えることで得られることがあります．例えば，チャプター E04 に含まれる補助ルーチンの場合，e54nfu という名称を持っています．（e54nfu は，通常は e04nfa の qphess 引数の実際の引数として使用されます．また，e04nff で使用される同様の補助ルーチンの名称は e04nfu です．）

3.7 動的メモリ割当て

ライブラリルーチンの中にはインタフェースの簡略化のために動的にメモリを割当てるものがあります．可能な場合にはどれだけのメモリが確保されるか（通常はルーチン引数の関数式）がルーチンドキュメントに明記されています．ライブラリルーチンによって確保されたメモリはすべてリターン前に解放されます．

十分なメモリの動的割当てに失敗した場合には，ルーチンはエラー条件 ifail $= -999$ をセットすると同時にエラーメッセージを出力しリターンします．

3.8 ライセンス管理

ご使用の実装がライセンス管理されている場合，その運用の詳細情報はユーザーのローカルサイトに存在するはずです．サイト管理者にお尋ねください．お使いのマシン上で正規のライセンスが利用できるかどうかを確認するためには，ライブラリルーチン a00acf の Example プログラムを実行してください．

ライブラリからライセンス管理ルーチンを呼び出した際，万一正しいライセンスが見つからなかった場合には，ルーチンはエラー条件 ifail $= -399$ をセットすると同時にエラーメッセージを出力しリターンします．その場合，Unix ベースのシステムでは，環境変数 NAG_KUSARI_FILE に適切なライセンスファイルまでのフルパスが正しく指定されているかどうかをチェックしてください．また，ライセンスファイルの内容が適切かどうか（例えば，異なる製品のライセンスを使用してないかどうか等）をチェックしてください．すべての設定が正しいと思われる場合は，日本 NAG にお問い合わせください．

3.9 予期しないエラー

万一予期しないエラーが起こった場合でも，ルーチンはエラー番号を ifail に設定し，適切なエラーメッセージを出力して終了します．基本的には ifail に設定される番号の意味はルーチン毎に異なり，予期しないエラーに対してもルーチン毎に異なる番号が設定されます．しかし，最近新たに追加されたルーチンでは，予期しないエラーに対して共通の番号 ifail $= -99$ を用いるよう標準化が行われています．

3.10 他言語からのライブラリの呼び出し

一般的には，データ型の間に適切なマッピングが存在すれば他言語（C や Visual Basic など）からも NAG ライブラリを呼び出すことができます．

NAG はこれまでに各種 C ヘッダーファイル（各種コンパイラに対して C と Fortran のデータ型の間のマッチングを示すヘッダーファイル），ドキュメント，Example を作成してきました．これらは NAG のウェブサイトに掲載されています．

ダイナミックリンクライブラリ（DLL）実装の場合には多くの言語環境（例えば，Visual Basic, Visual Basic

for Applications (Excel), Fortran, C, C++ 等）から容易に呼び出せます．そのためのガイダンスは NAG ライブラリ DLL のユーザーノート（Users' Note）に提供されます．詳細は NAG のウェブサイトをご参照ください．

3.11 算術の考察と結果の再現性

NAG ライブラリルーチンから得られる結果は，問題の解決に使用されたアルゴリズムだけではなく，ライブラリをビルドする際に使用されたコンピューターやコンパイラの実行時ライブラリ，また実行に使われるマシンの算術特性にも依存します．

歴史的に，異なる種類のコンピューターハードウェアは異なる種類の算術システムを持つ傾向がありました．浮動小数点数の格納に，あるマシンでは基数 16 を使い，あるマシンでは基数 2 を使いました（基数が 8 や 10 といったマシンもありました）．そのような違いはライブラリプロバイダーにとって頭痛の種でした．ある算術システムで上手く動作したコードが別の算術システムでは同じように動作しないかもしれないからです．ライブラリコードをポータブルにするためには，たくさんの注意を払わなければなりませんでした．

加えて，マシンの算術がフローやエラーを起こすことがあり，掛け算や割り算などの基本的な演算が（特に非常に大きい数や非常に小さい数に対して）時おり間違った結果を与えることがありました．

浮動小数点数の算術に関する最初の IEEE 標準（ANSI/IEEE (1985)）が 1980 年代に導入された後に，この状況は大きく改善されました．現在，主なハードウェア（NAG ライブラリが動作するほとんどのハードウェア）は IEEE スタイルの基数 2 の算術を使っています．これによりポータブルコードの製造がより簡単になりました．しかし，IEEE 標準には自由裁量の部分があるため，まだ問題は残されています．例えば，算術に 80-bit 内部レジスタ（元々は 1980 年代に Intel 8087 コプロセッサで導入された）を使うハードウェアは，特にコンパイラが算術の部分式を保持する最適化コードを生成する場合には，64-bit レジスタを使うハードウェアとは微妙に異なる動きをします．

コンピューターの算術は（IEEE 標準の算術がそうであるように）一般的に有限精度です．そのため，NAG ライブラリルーチンで実装されている数値計算法のほとんどの解は（単に丸め誤差の蓄積により）厳密解の近似ということになります．

従って，二つの異なるマシンで同じデータを用いて NAG ライブラリルーチンのプログラムを実行すると，コンパイラやハードウェア，また実行時ライブラリなどの違いによって結果が異なります．大抵この違いは小さく，例えば，二つの異なるマシンで良条件の連立一次方程式を解いた場合に，二つの計算結果が最後の数ビットだけ異なるといった具合です．しかし，時には小さな違いが重要視される場合があります．例えば，条件判断がその小さな違いに依存している場合などです．最適化問題のルーチンは常に同じ局所的最適解に収束するとは限りません（常に同じ局所的最適解が得られる場合は，その旨がルーチンドキュメントに注記されています）．また，たとえ同じ局所的最適解に収束するとしても，反復回数は異なるかもしれません．

最新のハードウェアと最適化コンパイラは算術演算に更なる問題を喚起します．その一例が**ストリーミング SIMD 拡張（SSE）**命令の利用にあります．

SSE 命令は浮動小数点数算術演算の低レベルの並列化を可能にします．例えば，128-bit SSE レジスタは二つの 64-bit 倍精度の数値（または四つの 32-bit 単精度の数値）を同時に保持し，同時に演算することができます．大量のデータを扱っている場合，これは大きな時間の節約になります．

しかし，SSE 命令の効率的な使用はメモリ上のデータの並び方に依存します．メモリ間のデータの移動を行う SSE 命令は，データが 16 バイト境界に並んでいることを必要とします．もし NAG ルーチンが使用しているデータ（例えば，数値の配列へのポインタ）がたまたま上手く整列していない場合，それらの SSE 命令は使用できません．これに対して，最適化コンパイラは二つの命令ストリーム（データが上手く整列している場合とそうでない場合）を上手く生成します．

例えば，二つの n 次元ベクトル x と y の内積を計算する場合を考えてみましょう．ベクトルの内積は，二つのベクトルの相対する成分の積を取り，個々の積を足し合わせることで得られます．良い最適化コンパイラでコンパイルされたルーチンは，二つ（または四つ）の数値を一度にロードします．そして，二つ（または四つ）の数値同士の積を一度に行って最終結果に加算します．

しかし，データがメモリ上に上手く整列していない状況では（これは良くあることなのですが），一度に一つの数値同士の演算しかできません（従って，最終結果を得るのにより長い時間がかかります）．最適化コンパイラはこの状況に対応するコードパスも生成します．そして，実行時にコードは速いパスを取れるかどうかをチェックして適切に動作します．

問題は，加算の順序が変わると最終結果が変わるということです．これはコンピューターの算術が有限精度であるところから来る丸め誤差に由来します．以下の内積の代わりに，

$$s = x_1 \times y_1 + x_2 \times y_2 + x_3 \times y_3 + \cdots + x_n \times y_n$$

次のような内積が行われるかもしれません．

$$s = (x_1 \times y_1 + x_3 \times y_3) + (x_2 \times y_2 + x_4 \times y_4) + \cdots$$

どちらの方法でも同じ結果が得られるように思えますが（ただし算術は有限精度であるため，どちらの結果も近似解でしかありません），二つの結果は微妙に異なります．もしこの小さな違いが呼び出し元のプログラムで異なる分岐をもたらすならば，呼び出し元のプログラムの動作に大きな違いを生じさせることになります．

更に，同じマシンで同じデータ用いて同じプログラムを連続で 2 回実行した場合でも，結果は異なる可能性があります．これは，プログラムがロードされた時に，たまたまデータが特定の境界に整列する場合もあれば，そうでない場合もあるからです．

より新しいハードウェアにおいては，AVX 命令が 256-bit と 512-bit のレジスタを使用することにより，一度により多くの数値を演算することができます．AVX 命令に対して，データはメモリ上に 32 バイト幅で並んでいる必要があります．

NAG ライブラリルーチンによって使用されるデータは，NAG ライブラリの内部でメモリに割当てられます．NAG ライブラリルーチンは上述した計算結果の違いを最小にするために，自前のメモリ割当てルーチンを使うなどして，データが常に上手く整列するように配慮します．しかし，その配慮は部分的にコンパイラのサポートに依存しているため，常に有効とは限りません．

当然のことながら，ルーチンに渡される前のデータのメモリ割当てをライブラリルーチンはコントロールできません．もし容認できないような非決定論的な計算結果に遭遇し，それがメモリ上のデータの並び方に依るものではないかと疑われる場合は，データがメモリ上で上手く整列するように配慮することが望まれます．しかし，これをどのように行うかはご利用のシステムに依存した話となります．

マルチスレッド NAG ライブラリやマルチスレッドベンダーライブラリの並列性は，計算結果の非再現性のまた別の要因となります．ルーチンによっては，異なるコア数で実行した時に（または，同じコア数で同じ計算を繰り返した時でさえ）異なる結果を得ることがあるかもしれません．結果の再現性が重要な場合は，並列化されていないシリアル NAG ライブラリを利用するのが最良です．

3.11.1 ビット単位の再現性（Bit-wise Reproducibility（BWR））

固定長浮動小数点数（例えば，32-bit 単精度や 64-bit 倍精度など）の数学演算では，結合法則が常に満たされるとは限りません．つまり，コンピューターの計算では $a+(b+c)$ と $(a+b)+c$ の結果が異なる場合があるということです．例えば，IEEE 754 32-bit 浮動小数点数では，$2^{24}+(1-2^{24})=1$ となる一方で $(2^{24}+1)-2^{24}=0$ となります．これは，IEEE 754 32-bit 浮動小数点数の仮数が 23 ビットであるため，$2^{24}+1=2^{24}$ となるからです．BWR という用語は，コンピュータープログラム（例えば，一連のソースプログラム）が以下のような異なるコンピューター環境においてビット単位で正確に同じ答えを生成する場合を指します．

1. 異なるオペレーティングシステム（例えば，Windows 対 Linux など）
2. 異なる CPU アーキテクチャ（例えば，Intel 対 AMD または Intel Sandy Bridge 対 Intel Ivy Bridge など）
3. 異なるコンパイラバージョン
4. 異なるスレッド数

ユーザーはしばしば BWR を望みますが，しかし，これを達成することは極めて困難です．一般的に，次の条件を満たさなければなりません．

(a) 命令を常に正確に同じ順番で実行すること．
(b) 他のプロセッサでは利用できないかもしれない高度な CPU 機能（例えば，SSE3，SSE4，AVX）を使わないこと．
(c) スレッド数を常に固定すること．

条件 (a) は，最適化なしで（または，非常に制限した最適化で）コンパイルを行うことを意味します．何故なら，一般的に，コンパイラのバージョンに依って最適化の方法が異なるからです．条件 (b) は，一般的に，最も普及している基本的な SSE 命令だけを利用し，新しいプロセッサの強化された SIMD 命令は利用しない，ということを意味します．

要するに，幅広いコンピューター環境で BWR を達成するためには，パフォーマンスを犠牲にする必要があるということです．

3.11.1.1 ベンダーライブラリと条件付きビット単位の再現性（Conditional BWR（CBWR））

NAG ライブラリの実装には，ベンダーライブラリ（特に，ベンダーライブラリの線形代数ルーチン）を利用するものがあります．ベンダーライブラリから得られる計算結果については，NAG ライブラリは BWR を直接制御することができません．

CBWR を導入した一部のベンダーライブラリでは，呼び出し元のコードが一定の条件を満たしていれば，環境変数を設定することによって BWR が有効になります．しかしながら，多くの NAG ルーチンは，ベンダーライブラリの CBWR が要求する条件を満たしていません．従って，ベンダーライブラリを利用するタイプの

NAG ライブラリ実装では，異なる CPU アーキテクチャに渡って BWR を保証することはできません．

3.12 マルチスレッド

3.12.1 スレッドセーフ

マルチスレッドアプリケーションでは，チーム内の各スレッドは，同じメモリアドレス空間を共有しながら独立に命令を処理します．マルチスレッドアプリケーションが正しく動作するためには，アプリケーションから呼び出されるルーチンがスレッドセーフでなければなりません．つまり，それらのルーチンに含まれるグローバル変数が異なるスレッドから同時にアクセスされないようになっていなければなりません．これは，OpenMP にみられるような適切な同期によって保証されます．

スレッドセーフと明記されているルーチンは，複数のスレッドから安全に呼び出すことができます．また，スレッドセーフでないルーチンでも，そのルーチン自体がマルチスレッド化されている場合があります．セクション 3.12.2 で説明されているように，ルーチン内でスレッドのチームを生成して，ワークロードを共有することができます．

NAG Fortran Library のほとんどのルーチンはスレッドセーフですが，非同期グローバル変数（モジュール変数，共有（COMMON）ブロック，SAVE 属性を持つ変数など）を使用するために，スレッドセーフでないルーチンもあります．これらのルーチンは，ユーザープログラムの複数のスレッドから安全に呼び出すことはできません．詳細については，各ルーチンドキュメントのセクション 8 を参照してください．スレッドセーフでないルーチンのリストは，Thread Unsafe Routines ドキュメントをご参照ください．

NAG Fortran Library には，同じ 5 文字のルート名を共有するいくつかのルーチンペアがあります．例えば，ルーチン e04ucf と e04uca のペアです．ルーチンペアはそれぞれまったく同じ機能を持ちますが，ルーチン名の最後の文字が通常と異なる方のルーチン（大抵の場合 ‘f’ の代わりに ‘a’，例えば，e04uca）は，スレッドセーフのための引数が追加されています．ルーチンペアは，ライブラリマニュアルにはそれぞれ個別のルーチンとしてリストされていますが，同じルーチンドキュメントを共有します．

3.12.1.1 ルーチン引数を持つルーチン

一部のライブラリルーチンでは，引数にユーザーがルーチンを提供する必要があります．多くの場合，ユーザー提供ルーチンの引数リストには，グローバル変数を使わずにユーザー提供ルーチンに情報を渡すための配列引数（iuser および ruser）が含まれています．

スレッドセーフ（末尾が ‘a’）と非スレッドセーフ（末尾が ‘f’）のルーチンペアの引数リストにユーザー提供ルーチン引数がある場合，‘a’ ルーチンは追加の配列引数 iuser と ruser を持ちます．場合によっては，‘a’ ルーチンを個別の初期化ルーチンで初期化する必要があるかもしれません．詳細については，該当のルーチンドキュメントをご参照ください．

Mark 26.1 から，（iso_c_binding モジュールの）Type (c_ptr) 型の引数 cpuser が追加されました．これにより，iuser と ruser が不便な場合に，より複雑なデータ構造をユーザー提供のルーチンに簡単に渡すことができます．以下に用例を示します．

```
   Module mymodule
       Use iso_c_binding, Only: c_f_pointer, c_ptr
       Private
       Public                              :: myfun
      Type, Public                         :: mydata
        Integer                            :: nx
        Real (Kind=nag_wp), Allocatable :: x(:)
      End Type mydata
    Contains
      Subroutine myfun(...,iuser,ruser,cpuser)
        Type (c_ptr), Intent (In)          :: cpuser
        Type (mydata), Pointer             :: md
        Call c_f_pointer(cpuser,md)
             ... Use md%x and md%nx ...
      End Subroutine myfun
   End Module mymodule
   ...
   Program myprog
      Use mymodule, Only: myfun,mydata
      Use iso_c_binding, Only: c_loc, c_ptr
      Type (c_ptr)                         :: cpuser
      Type (mydata), Target                :: md
...
      md%nx = 1000
      Allocate (md%x(md%nx))
      cpuser = c_loc(md)
...
      call nagroutine(...,myfun,cpuser,iuser,ruser,ifail)
...
   End Programe
```

このメカニズムは，例えば e04stf のセクション 10 で使われています．

引数リストを介して与えられる以上の情報をユーザー提供ルーチンに与える必要がある場合は，目的のライブラリルーチンと同等のリバースコミュニケーションルーチンがあるかどうか，該当のチャプターイントロダクションを確認することをお勧めします．リバースコミュニケーションは，ユーザー提供ルーチンの引数リストが十分な柔軟性を持つことができない場合に特化した設計になっており，グローバル変数を介してデータを提供する必要もありません．リバースコミュニケーションが利用できない場合は，通常，ユーザー提供ルーチンと呼び出し元プログラムの両方からアクセス可能なグローバル変数を使用します．ただし，異なるスレッドによるグローバル変数の同時アクセスを回避できる場合にのみ（OpenMP によって threadprivate にされている場合，もしくは適切な同期を使用している場合にのみ），この方法はスレッドセーフです．

ユーザー提供ルーチンのスレッドの安全性は，NAG ライブラリのマルチスレッド版でもまた問題になります．NAG ライブラリのマルチスレッド版の多くのルーチンは，ユーザー提供ルーチンの呼び出しを内部的に並列化します．この問題はグローバル変数だけでなく iuser 配列と ruser 配列の使用方法にも影響します．このような場合，スレッドの安全性を確保するために同期が必要な場合があります．チャプター X06 には OpenMP 並列領域から呼び出されているかどうかを判断するルーチンが提供されおり，ユーザー提供ルーチン内で使用することができます．

3.12.1.2 入出力

NAG ライブラリには，エラーメッセージおよびアドバイスメッセージの出力装置番号を設定するためのルーチン（x04aaf と x04abf）があります．ただし，これらのルーチンは SAVE 文を使用して装置番号の値を保持するため，スレッドセーフではありません．

マルチスレッドアプリケーションから NAG ライブラリルーチンを呼び出す場合，エラーメッセージまたはアドバイスメッセージは各スレッドから同時に出力されるため，どのスレッドがどのメッセージを生成したか識別できず，内容が不明瞭になります．従って，エラー処理については，エラーメッセージなしの soft failure を選択すること（つまり，引数 ifail に 1 を設定すること（セクション 3.4 参照））をお勧めします．この場合，処理が呼び出し元に戻ってくる時に ifail の値を確認することが基本となります．

3.12.1.3 実装依存の問題

非常にまれなケースですが，スレッドの安全性を保証することができない実装もあります．また，実装によっては，例えば BLAS 関数を高速化するなどの理由により，ベンダーライブラリとリンクされていることがあります．それらのベンダーライブラリがスレッドセーフであることを NAG は保証できません．実装固有の情報については，該当のユーザーノートをご参照ください．

3.12.2 並列性

3.12.2.1 イントロダクション

通常の並列化されていない（シリアル）NAG ライブラリの計算時間は，使用されるプロセッサの逐次処理性能に大きく左右されます．一方で，並列化された（マルチスレッド）NAG ライブラリは，計算タスクを複数のコアに分配し並列に処理するため，プロセッサの逐次処理性能を超えて計算を行うことができます．

従来，コンピューターシステムは少数のシングルコアプロセッサで構成されていました．そして，プロセッサの処理性能の向上はクロック周波数の増加によって行われました．しかし，このクロック周波数の増加が限界に達し，処理性能を向上させる別の方法が求められました．そこで登場したのがマルチコアプロセッサです．そして，それは今や至るところで使われています．マルチコアプロセッサは，単一のコアではなく，複数のコア（各コアは基本的には CPU と小さなキャシュから成る）で構成されています．従って，現今のハードウェアリソースを最大限に利用するためには，並列性を活用することが必須となっています．

並列化の有効性は，並列プログラムが同等の逐次プログラムに比べてどの程度速いかで評価できます．これは並列高速化（parallel speedup）と呼ばれます．逐次プログラムが並列化されたときの高速化は，同じ計算問題に対して，逐次プログラムの計算時間を並列プログラムの計算時間で割ることによって定義されます．もし，

並列プログラムが n コアを使用した場合に，この高速化の値が n ならば（つまり，並列プログラムの計算時間が元の逐次プログラムの $\frac{1}{n}$ ならば），理想的な高速化が得られたことになります．使用するコア数の増加に対して，並列プログラムの高速化がこの理想的な値に近いならば，その並列プログラムはスケーラビリティ（scalability）が良いと言います．

以下の２つの要因のため，並列プログラムのスケーラビリティは理想的な値よりも小さくなります．

(a) 並列化に伴うオーバーヘッド
(b) プログラムの並列化不可能な逐次部分

オーバーヘッドは並列化に必要なセットアップだけでなく通信と同期も含みます．このようなオーバーヘッドは，コンパイラとオペレーティングシステムのライブラリの効率，そして基礎となるハードウェアに依存して異なります．プログラムの並列化不可能な逐次部分の影響は，アムダールの法則（Amdahl's law）によって理論的に（つまり，オーバーヘッドがゼロである理想的なシステムを仮定して）説明されます．アムダールの法則は，逐次部分を持つ並列プログラムの高速化に上限を設けます．r をプログラムの並列部分の割合，$s = 1 - r$ を逐次部分の割合とすると，n コアを使用した場合の高速化 S_n は次の条件を満たします．

$$S_n \leq \frac{1}{\left(s + \frac{r}{n}\right)}$$

例えば，4 分の 1 が逐次部分である並列プログラムの高速化は高々 4 となります．なぜなら，$n \to \infty$，$S_n \leq 4$ だからです．

並列化は共有メモリマシンと分散メモリマシンの二種類のシステムで利用でき，それぞれ異なるプログラミング技術を必要とします．分散メモリマシンは，ネットワークで繋がった複数のコンポーネントで構成されており，個々のコンポーネントがプロセッサとメモリ領域を持ちます．これらのコンポーネント間の通信と同期は明示的です．共有メモリマシンは，複数の（もしくは一つの）マルチコアプロセッサを持ち，これらが同じメモリ領域にアクセスします．そして，この共有メモリが通信と同期に使われます．マルチスレッド NAG ライブラリは OpenMP を用いた共有メモリ並列化を利用します（セクション 3.12.2.2 参照）．

OpenMP を用いた並列プログラムは，実行時に必要に応じて，単一のプロセスから複数のスレッドを生成します（fork）．（なお，共有メモリ並列化を利用するプログラムのことをマルチスレッドプログラムと呼びます．）スレッドは一つのマススタースレッドといくつかのスレーブスレッドで構成されるチームを形成します．これらのスレッドは，互いに独立して並列にプログラム命令を実行することができます．並列処理が一旦完了すると，スレーブスレッドはマススタースレッドに制御を戻し，次の並列処理領域まで非アクティブとなります（join）．スレッドは同じメモリアドレス空間（例えば，親プロセスのメモリアドレス空間）を共有します．そして，この共有メモリが通信と同期に使われます．OpenMP はアクセス制御のメカニズムを提供します．各スレッドは共有変数にアクセスできるだけでなく，自分だけがアクセス可能な他の変数のプライベートコピーを持つことができます．チーム内のスレッドは並列領域内に独自の並列領域を作成できます．この次のレベルの並列処理では，新しいチームを作るスレッドがそのチームのマススタースレッドになります．これをネスト並列処理（nested parallelism）と呼びます．

シリアルプログラムとの違いとしてマルチスレッドプログラムで理解しておかなければならないことは，同一の結果は保証されないし，また期待すべきでもないということです．並列プログラムでは多くの場合，同一の結果を得ることは不可能です．なぜなら，使用するスレッド数が異なると浮動小数点演算の順番が異なり

（しかし，どれも正しい），結果として丸め誤差の累積が変わるからです．結果の再現性についての更なる議論はセクション 3.11 をご参照ください．

3.12.2.2 NAG ライブラリはどのように並列化されているか?

マルチスレッド NAG ライブラリは OpenMP によるマルチスレッディングを利用しているという点でシリアル NAG ライブラリとは異なります．OpenMP は多くの異なるハードウェアプラットフォームの多くの異なるコンパイラで利用可能な共有メモリプログラミングについての移植性のある仕様です．

マルチスレッド NAG ライブラリの全てのルーチンが並列化されているわけではないので注意してください．各ルーチンの並列性とパフォーマンスについての詳細は，各ルーチンドキュメントのセクション 8 をご参照ください．

NAG ライブラリルーチンの呼び出しが並列化の恩恵を受ける状況には以下の 2 つがあります：

1. 呼び出した NAG ライブラリルーチンが OpenMP を用いて並列化された NAG 固有のルーチンである．もしくは，呼び出した NAG ライブラリルーチンが OpenMP を用いて並列化された別の NAG 固有のルーチンを内部的に呼び出している．これは，NAG ライブラリのマルチスレッド版にのみ適用されます．
2. 呼び出した NAG ライブラリルーチンが BLAS または LAPACK ルーチンを呼び出している．ご利用の NAG ライブラリ製品で使用が推奨されているベンダーライブラリは（NAG ライブラリ自体が並列化されているかどうかに関わらず）並列化されています．更なる情報は，ご利用の製品のユーザーノートをご参照ください．

NAG ライブラリルーチンの完全なリストならびに並列化の状況はセクション 3.12.3 をご参照ください．

ライブラリの非効率な使用を避けるために，ライブラリの中で OpenMP がどのように利用されているかを知ることは有益です．

並列化された NAG 固有のルーチンは，実行中に OpenMP を用いて複数の並列処理領域でスレッドチームを生成します．スレッドチームは並列領域の開始時に分岐（fork）し，並列領域の終了時に合流（join）します．fork と join は両方とも呼び出されたルーチンの内部で起こります．ただし，ユーザー提供ルーチンの中に OpenMP 指示文があれば，そこでスレッドチームが利用されるという状況はあり得ます．並列領域内に含まれていない指示文を親無し指示文（orphaned directive）と呼びます．（詳細はルーチンドキュメントのセクション 8 をご参照ください．）NAG ルーチン内の OpenMP 指示構文は NAG コード内で生成されたスレッドチームによって実行されます（つまり，ライブラリ自身には親無し指示文はありません）．本ドキュメントでは，ユーザーノートで推奨されているコンパイラの使用，特に単一の OpenMP ランタイムライブラリの使用を想定しています．従って，すべての OpenMP 環境変数はユーザーコードと NAG ルーチンに適用されます．しかし，それらをオーバーライドする仕組みを持っているベンダーライブラリには適用されないかもしれません．NAG ライブラリでは，プログラム全体のスレッドを制御するためのルーチンをチャプター X06 に提供しています．これは，NAG ライブラリによって呼び出されるベンダーライブラリ固有のスレッドにも適用されます．ユーザープログラムの並列領域から NAG ルーチンを呼び出すときには注意が必要です．ネスト並列処理（nested parallelism）が有効になっている場合（デフォルトでは無効になっています），NAG ルーチンは各スレッドからの呼び出しに対してスレッドチームを fork および join します．これによりシステムリソースの

競合が起き，パフォーマンスが著しく低下します．競合によるパフォーマンスの低下は，要求されたスレッド数がマシンの物理コア数を超える場合や，ハードウェアリソースが他のプロセス（共有システムにおける他のユーザープロセス）の実行でビジーな場合にも起こります．このような理由から，マシンで利用可能な物理コア数を考慮して，リソースの競合を最小にするスレッド数を選択する必要があります．スレッド数の設定についてのアドバイスは，ご利用の製品のユーザーノートをご参照ください．

他のスレッドメカニズムからマルチスレッド NAG ルーチンを呼び出す場合は，そのスレッドメカニズムが，ご利用のマルチスレッド NAG ライブラリをビルドする際に使用された OpenMP コンパイラランタイムと互換性があるかどうかが問題となります．更なるアドバイスは，ご利用の製品のユーザーノートをご参照ください．

NAG ルーチンの多くは並列化の対象になっていませんが，これらの並列化されていない NAG ルーチンでも，並列化された NAG ルーチン，および／または，並列化されたベンダールーチン（例えば，BLAS と LAPACK）を内部的に呼び出して利用しているルーチンは間接的に並列化の恩恵を受けます．従って，ライブラリ全体に渡り多くの箇所で並列処理は行われます．並列化によるパフォーマンスの向上は，呼び出すルーチンの種類，問題サイズと引数，システム設計そして OS 構成に依って変わってきます．もし，同様のデータサイズと引数でルーチンを頻繁に呼び出すなら，最適なパフォーマンスを得るために，色々なスレッド数を試してみることは価値があります．

一般的な指針として，以下の分野の主なルーチンは共有メモリ並列化の恩恵を受けます：

- Dense and Sparse Linear Algebra（密・スパース線形代数）
- FFTs（高速フーリエ変換）
- Random Number Generators（擬似乱数生成）
- Quadrature（数値積分）
- Partial Differential Equations（偏微分方程式）
- Interpolation（補間）
- Curve and Surface Fitting（曲線・曲面のあてはめ）
- Correlation and Regression Analysis（相関・回帰分析）
- Multivariate Methods（多変量解析）
- Time Series Analysis（時系列解析）
- Financial Option Pricing（オプションプライシング）
- Global Optimization（大域的最適化）
- Wavelets（ウェーブレット変換）

3.12.3 並列化ルーチン

NAG ライブラリのマルチスレッド版では多くのルーチンが OpenMP を用いて並列化されています．マルチスレッド版の製品コードの形式はシリアル版の ‘FL_____’ に対して ‘FS_____’ となります．詳細については，各ルーチンドキュメントのセクション 8 をご参照ください．NAG によって並列化されたルーチンのリストは Multithreaded Routines ドキュメントをご参照ください．BLAS または LAPACK ルーチンを内部的に呼び出すルーチンのリストも同じドキュメントに含まれています．NAG ライブラリが利用するベンダーライ

ブラリに含まれる BLAS および LAPACK ルーチンは，NAG ライブラリがシリアル版かマルチスレッド版かに関わらず，ベンダーライブラリ内で並列化されています．詳細についてはベンダーライブラリのドキュメントをご参照ください．NAG 製品固有の情報については，各製品のユーザーノートをご参照ください．

4 ドキュメントの使い方

4.1 マニュアルの使用

NAG Library Manual, Mark 26（ライブラリマニュアル）は NAG ライブラリに対して次の役割を果たすべく作成されています．

- 数値計算と統計解析の種々の分野に関する背景情報を提供する．
- 特定の問題の解決のためにどのライブラリルーチンを使用すべきかに関する助言を与える．
- Fortran プログラムからライブラリルーチンを呼び出し，その結果を評価する上で必要になるすべての情報を提供する．

マニュアルの先頭部（Introduction）には背景情報や追加情報に関する一般的な紹介資料が含まれています．

‘Mark 26 NAG Fortran Library News’ というドキュメントは，新たに追加されたルーチン，削除予定のルーチン，当 Mark で削除されたルーチンの詳細を記述しています．また，当 Mark のユーザーに影響を与える内部的な変更についても説明があります．

‘Advice on Replacement Calls for Withdrawn/Superseded Routines’（削除済みルーチンの代替についてのアドバイス）というドキュメントからはユーザープログラムの更新についてのアドバイスを得ることができます．

オンラインドキュメントでは Keyword and GAMS Search を用いて，キーワードでライブラリを検索することができます．検索には，各ページの右上にある Keyword Search ボックスが利用できます．

該当しそうなチャプターやルーチンが見つかったら，まず **Chapter Introduction**（チャプターイントロダクション）を読んでください．該当する数値計算分野に関する背景情報およびルーチン選択に関する推奨案（インデックス，テーブル，決定木を含む）が記述されています．

ルーチン選択が終わったら，今度は **Routine Document**（ルーチンドキュメント）の参照が必要です．個々のルーチンドキュメントは本質的に自己完結型の資料です（関連ドキュメントへの参照は含まれますが）．それには手法の説明，各引数の詳細仕様，個々のエラー復帰に関する説明，精度に関するコメント，および（ほとんどの場合）ルーチンの用法を示す Example プログラムが含まれています．場合によっては，Example プログラムの実行結果を示すプロットが付随しています．

4.2 ドキュメントの構成

ライブラリマニュアルはライブラリを使用する際に基本となるドキュメントです．それはライブラリと同一のチャプター構成を取っています．すなわち複数のライブラリルーチンから成る個々のチャプターにはマニュアルの同名のチャプターが対応しています．これらのチャプターはアルファベット順に並べられています．またマニュアルの先頭部にはライブラリの使用に際して基本となるドキュメント（Introduction）が配置されています．

それぞれのチャプターは次のドキュメントから構成されます.

- **Chapter Contents** - 例えば，D01 (quad) Chapter Contents
- **Chapter Introduction** - 例えば，D01 (quad) Chapter Introduction
- **Routine Documents** - チャプターに含まれるルーチンごとの仕様書

ルーチンドキュメントの名称はルーチン名と同一です．チャプター内においてルーチンドキュメントは短い名前のアルファベット順に並んでいます．'A' ルーチンを持つチャプターにおいては，'a' ルーチンに関する記述はペアとなる 'f' ルーチンと共に一つのルーチンドキュメントに集約されています.

すべてのルーチンドキュメントは 10 のセクションからなる同一の構成を持っています.

1. **Purpose**（目的）
2. **Specification**（仕様）
3. **Description**（説明）
4. **References**（参考文献）
5. **Arguments**（引数）（セクション 4.3 参照）
6. **Error Indicators and Warnings**（エラーと警告）
7. **Accuracy**（精度）
8. **Parallelism and Performance**（並列性とパフォーマンス）
9. **Further Comments**（コメント）
10. **Example**（使用例）（セクション 4.5 参照）

一部のドキュメント（主に，チャプター E04, E05, H）では更に 3 つのセクションが加わります.

11. **Algorithmic Details**（アルゴリズム詳細）
12. **Optional Parameters**（オプショナルパラメーター）
13. **Description of Monitoring Information**（監視情報）

上記のセクション 11. と 13. は無い場合もあります．その場合 **Optional Parameters**（オプショナルパラメーター）がセクション 11. として登場します.

4.3 引数の仕様

ルーチンドキュメントの Specification（仕様）には，C または C++ から NAG ライブラリルーチンを呼び出すための C ヘッダーインタフェースが含まれています（例えば，d03pdf/d03pda のセクション 2 を参照）．C ヘッダーインターフェースの名前は，Fortran インターフェースの短い名前にアンダースコアを付加したものです（例えば，d03faf に対して d03faf_）．同じアプローチは 'a' バージョンのルーチンにも適用されます（例えば，d03pda に対して d03pda_）.

各ルーチンドキュメントのセクション 5 には引数の仕様が引数リストに現れる順に記述されています.

4.3.1 引数の分類

引数は次のように分類されます.

Input（入力）: ルーチンを実行する際にはこれらの引数に対し値を代入しておく必要があります．これらの値はルーチンから復帰（exit）した際も変化しません.

Output（出力）：ルーチンを実行する際にはこれらの引数に対し値を代入しておく必要はありません．ルーチンの中で値がセットされます．

Input/Output（入出力）：ルーチンを実行する際にこれらの引数に対し値を代入しておく必要がありますが，これらはルーチン中で値が変更されることがあります．

Workspace（ワークスペース）：ルーチン用作業エリアとして使用される配列引数．正しい型と次元の配列をユーザーが用意する必要があります．ただし一般的に，ユーザーはその内容を知る必要はありません．

Communication Array（コミュニケーション配列）：一つのルーチン呼び出しから別のものへデータを受け渡すために使用される配列引数．

External Procedure（外部手続き）：ユーザーによって提供されなくてはならないルーチン（例えば，被積分関数の値を評価したり，中間結果を出力したりするためのもの）．通常それは呼出し元のプログラムの一部として提供されなくてはなりません．ルーチンドキュメントにはこの外部手続きの引数リストと各引数の仕様の詳細が含まれることになります．その引数はライブラリルーチンの引数と同様の形で分類されますが，ユーザーはそれを呼び出すのではなく自ら書く立場にあるため，分類の意味は異なってきます．

> ***Input***（入力）：呼び出し（entry）時，値が指定されます．手続きの中で**変更してはいけません**．
> ***Output***（出力）：手続きから復帰（exit）する前にこれらの引数に対して値をセットします．
> ***Input/Output***（入出力）：呼び出し（entry）時，値が指定されます．手続きから復帰（exit）する際に必要に応じて値をセットしてください．

セクション 3.6 に記されているように，手続きはライブラリから提供される場合もあります．そのような場合には名称の指定だけで済みます．

User Workspace（ユーザーワークスペース）：ライブラリルーチンから外部手続きに引き渡される配列引数．それらはルーチンによって使用されるものではなく，呼出し元のプログラムと外部手続きとの間での情報交換に使用されます．

Dummy（ダミー）：ルーチンでは使用されない通常変数．所定の変数または定数が提供されなくてはなりませんが，値の設定は必要ありません．（ダミー引数は多くの場合，旧バージョンでは必要だったものが互換性維持のために残されている引数です．）

4.3.2 制約条件と推奨値

入力引数の仕様中にある ‘*Constraint*:’ または ‘*Constraints*:’（制約条件）という語は，その引数に対する適正な値の範囲を規定します．

（例） *Constraint*: $\mathrm{n} > 0$

不正な引数値を用いて呼び出された場合（例えば $\mathrm{n} = 0$），ルーチンは通常エラー復帰し，ifail として 0 以外の値を返します（セクション 3.4 参照）．

タイプが Character の引数に対する制約条件としては大文字のアルファベットのみを記載しています．

（例） *Constraint*: check = ‘n’

しかし，実際には Character 引数を取るすべてのルーチンは小文字の使用を許しています．

場合によっては，新しい Mark（バージョン）で既存のルーチンが拡張され，引数の制約が緩くなることがあります．しかし，これによる既存のコードへの影響はありませんし，新しいコードでは拡張された機能を利用することができます．

‘*Suggested value*:’（推奨値）という語は，ユーザーがどのような値にすべきかわからないケースを想定して，入力引数に対する妥当な設定値を示唆します（例えば，精度や最大反復回数など）．個別の問題に対しそれらの値が適切ではないと判断される場合には異なる値を設定してください．

4.3.3 配列引数

配列引数の多くは問題のサイズに依存した寸法を持っています．Fortran の用語で言うと，それらは ‘整合寸法（adjustable dimensions）’ を持っているということになります．配列の宣言文に記述される寸法は整数変数であり，それらもまたライブラリルーチンの引数です．

例えば，ライブラリルーチンの仕様が次のように与えられたとします．

```
Subroutine <name> (m, n, a, b, ldb)
Integer              m, n, a(n), b(ldb,n), ldb
```

この例における a のような **1 次元配列**の場合，その仕様は次のような記述で始まることになります．

a(n) - Integer array（整数配列）

この場合，呼出し元の（サブ）プログラム内で宣言される配列の寸法は少なくとも引数 n に設定する値と同じ大きさでなくてはなりません．それより大きくても構いませんが，ルーチンが使用するのは最初の n 要素のみです．

上の例における b のような **2 次元配列**の場合，その仕様は次のように与えられたとします．

b(ldb,n) - Integer array（整数配列）
On entry（入力時）: $m \times n$ の行列 B

一方，引数 ldb の仕様は次のように与えられたとします．

ldb - Integer（整数）
On entry（入力時）: <name> を呼び出す（サブ）プログラム内で宣言される配列 b の第 1 次元の寸法
Constraint（制約条件）: ldb $\geq$ m

この場合，たとえルーチン内で実際に使用される行の数は引数 m によって決定されるとしても，配列 b の**第 1 次元**の寸法を引数 ldb でルーチンに与えなくてはなりません．またその寸法は少なくとも引数 m に設定する値と同じ大きさでなくてはなりません．２次元配列の部分配列にルーチンが作用できるように追加の引数 ldb が必要になります．配列の**第 2 次元**の寸法もまた，少なくとも引数 n に設定する値と同じ大きさでなくてはなりません．それより大きくても構いませんが，ルーチンが使用するのは最初の N 列のみです．

上で示したルーチンを呼び出すプログラムのサンプルコードを次に示します．

```
Integer aa(100), bb(100,50)  or     Integer Allocatable :: aa(:), bb(:,:)
ldb = 100                           Integer             :: m, n, ldb
.                                   .
.                                   .
.                                   .
m = 80                              Read(5,*) M, N
n = 20                              ldb = m
Call <name>(m,n,aa,bb,ldb)          Allocate (aa(m),bb(ldb,n))
                                    Call <name>(m,n,aa,bb,ldb)
```

多くの NAG ルーチンは，以下の様な ‘大きさ引継ぎ（assumed size）’ 配列次元で宣言される配列引数を持っています．

```
Integer          a(*), b(ldb,*)
```

しかし，呼出し元のプログラム中での配列宣言においては常にドキュメントに書かれている最小値以上の値を寸法として指定しなくてはなりません．配列に Allocatable 属性を付加すれば，コンパイル時には分からない配列の大きさを実行時に動的に割付けることができます．

配列引数を伴う NAG ルーチンの呼び出しで問題が生じた場合には，専門家に相談するか，または Fortran の書籍を調べてください．

4.4 実装依存情報

ライブラリのすべての実装に対応するために，ライブラリマニュアルでは実装によって解釈が異なる項目を区別する意味で太字イタリック体を使用しています．

一つ重要なものは ***machine precision*** という表現で，これは実数の浮動小数点数が計算機内で格納されている相対精度を意味します．例えば 10 進で約 16 桁の実装であれば，***machine precision*** は 10^{-16} に近い値を持ちます．

machine precision の正確な値はルーチン x02ajf を使って確認できます．チャプター X02 のその他のルーチンを使うと，オーバフロー用の閾値や表現可能な最大整数といった実装依存の定数値を求めることができます．詳細については X02 Chapter Introduction をご参照ください．

block size という表現はチャプター F07 と F08 の中でのみ使用されています．それはこれらのチャプターにおけるブロックアルゴリズムによって用いられているブロックサイズを表すものです．用意すべき作業エリアの量に影響が及ぶ場合にのみ，この値に留意する必要があります．関係するルーチンドキュメントとチャプターイントロダクションに記載されている引数 work と lwork についてご参照ください．

ライブラリの各実装ごとに個別の**ユーザーノート**（Users’ Note）が提供されます．これは短いドキュメントで Mark ごとに改訂されます．ほとんどの場合，機械可読（machine-readable）な形式で利用できます．それは実装に固有の追加情報，特に次のような項目を含んでいます．

- チャプター X02 ルーチンから返される値
- 出力用装置番号のデフォルト値（セクション 3.5 参照）

- 精度パラメーター nag_rp (***reduced precision***)，nag_wp (***basic precision***)，nag_hp (***additional precision***) の意味

4.5 Example プログラムと結果

ルーチンドキュメントのセクション 10 に記述されている **Example プログラム**は，ルーチンの呼び出し方を示す具体例です．修正が容易な形で作られているので，ユーザープログラムを開発する際のテンプレートとしてもご利用いただけます．

これらの Example プログラムはライブラリの実装ごとに機械可読（machine-readable）な形で配布されます．必要な修正は実施済みです．多くのサイトでは，この形式でプログラムをユーザーに公開しています．実装固有の修正を加えていない一般形式の Example プログラムは NAG のウェブサイトから直接ダウンロードすることができます．ご利用の実装に対するユーザーノート（Users' Note）には，一般形式の Example プログラムに対して必要となる変更ついての記載があります．

Example プログラムの実行結果は全ての実装で同一になるとは限りません．また，マニュアルに記載されている結果とも一致しないことがあります．

多くのルーチンドキュメントでは Example プログラムの実行結果に対してプロット図が提供されています．場合によっては，解のプロット図をより典型的なものとするために，より大きな実行結果を生成するような（若干の）変更が Example プログラムに施されています．

4.6 オンラインドキュメント

ライブラリマニュアルは次の形式で閲覧することができます．

- **HTML** - HTML, SVG, MathML によって閲覧可能なマニュアル（各ドキュメントの PDF 版へのリンクを含む）
- **PDF** - PDF のしおり（または HTML 目次ファイル）によって閲覧可能な PDF マニュアル
- **PDF（単一ファイル）** - 複数の PDF マニュアルを 1 つにまとめた単一の PDF マニュアル
- **Windows HTML ヘルプ（単一ファイル）** - Windows HTML ヘルプ形式のマニュアル

単一のファイルからなる形式はルーチンごとに一つファイルを使う形式よりもコンパクトです．例えば，マニュアル全体でテキスト検索を行うことができます．しかし，いくつかのルーチンのドキュメントしか見ないのであれば大きなサイズのファイルは不便かもしれません．

以下のセクションでは，ドキュメントを表示するために必要なソフトウェアを入手する方法を説明し，ブラウザを使う場合と使わない場合でドキュメントを閲覧する方法を説明します．

4.6.1 HTML 形式

4.6.1.1 HTML5 ファイルの表示

これらのファイルは特定のブラウザに固有の機能は使用せず W3C 勧告（HTML，MathML，SVG，CSS）に準拠しています．

これらの言語をサポートするにはブラウザのアップデートや追加フォントのインストールが必要な場合があります.

4.6.1.2 Firefox（その他 Mozilla ベースのブラウザ）

Firefox 4 以降のバージョンの Firefox ではデフォルトで HTML ファイルに MathML が表示されます.

STIX や他の OpenType 数学フォントがご利用のシステムに含まれていない場合は，これらのフォントをインストールすることによって数式の描画が改善されます．インストーラーの詳細は Firefox MathML フォントページをご参照ください.
http://www.mozilla.org/projects/mathml/fonts/

4.6.1.3 その他のブラウザ

Firefox を使っていない場合，ページの javascript は MathML を有効にするために MathJax javascript ライブラリ（http://www.mathjax.org）を読み込みます．デフォルトでは MathJax コンテンツ配信ネットワークを使用してウェブからロードされます．インターネットに接続せずにドキュメントを参照する必要がある場合は，前のセクションで説明したように Firefox を使用するか，もしくは http://docs.mathjax.org/en/latest/installation.html からダウンロードした MathJax のローカルコピーをご利用のローカルファイルサーバーまたはローカルファイルシステムに解凍してください．そして，../styles/nagmathml.js ファイルの `http://cdn.mathjax.org/mathjax/latest/`という行を，先ほどのローカルコピーを参照するように編集してください.

4.6.1.4 HTML5 ファイルの閲覧

メインの目次ファイル（html/frontmatter/manconts.html）が提供されており，個々のチャプターコンテンツにリンクしています．ブラウザを使ってこの目次ファイルから各ドキュメントを閲覧します．HTML 形式の各ルーチンドキュメントには，同等の内容の PDF ファイルへのリンクが付いています．これらの PDF ファイルは主に印刷目的で提供されています.

各ドキュメントは色々な要素（引数，セクション，チャプターコンテンツなど）へのハイパーリンクを含んでいます．各要素に対して使用される文字色を以下の表に示します.

文字色	要素
black	NAG 型
green	付録，チャプターイントロダクション，決定木，一般的なイントロダクション，セクション
grey	廃止されたドキュメント
pale blue	方程式，図形，リスト内の項目，注釈，書誌参照，表，URL，逐語的な項目，ウェブサイト
navy blue	ifail 値
red	引数名
pink	メンバ
purple	オプショナルパラメーター
royal blue	HTML 目次，Example プロット，ルーチンドキュメント，目次からの Example へのリンク

4.6.1.5 HTML5 ファイルの印刷

HTML5 ファイルをブラウザから印刷することは可能ですが，ブラウザからの印刷のサポート，特に数式の印刷のサポートは，ご利用のブラウザ，プラットフォームおよびプリンタードライバーのバージョンによって大きく異なります．

4.6.1.6 Windows HTML ヘルプ

Windows HTML ヘルプ形式のライブラリマニュアルは基本的に HTML5 ファイルを圧縮したもので, Windows HTML ヘルプビューア（MathJax のバンドル版）用にカスタマイズされています．この形式は小さく圧縮された単一のファイルであるため，マニュアル全体のテキスト検索ができるなど非常に便利です．Microsoft Windows でご利用いただけます．

4.6.2 PDF 形式

4.6.2.1 PDF ファイルの表示と印刷

Adobe Acrobat Reader をお持ちでない場合は，http://www.adobe.com/reader から無料でダウンロードすることができます．ご利用のプラットフォームに対応した Acrobat Reader があるかどうかは上記サイトをご参照ください．基本的には Acrobat Reader の利用をお勧めしますが，xpdf や ghostview などの代替 PDF ビューアでもご利用いただけます．

ローカルファイルシステムではなく http 経由で PDF ファイルをブラウズする場合，Acrobat がプラグインとして動作していないとブックマークリンクが正しく動作しません．この問題の解決には Adobe Acrobat を再インストールすることをお勧めします．

4.6.2.2 PDF ファイルの閲覧

PDF 形式のライブラリマニュアルは，複数の PDF ファイル（各ルーチンドキュメントやチャプターイントロダクションなど）のセットとして提供されます．各 PDF ファイルには，ファイル間を行き来するためのブックマークが含まれています．また，HTML 形式の目次が提供されており，ブラウザを使って目的の PDF ファイルにアクセスすることができます．その際，Acrobat をブラウザのプラグインとして使用すれば，ブラウザで PDF ファイルの表示や印刷を行うことができます．

また，単一の PDF ファイルからなるライブラリマニュアルを利用することもできます．この PDF ファイルのブックマークには全てのルーチンドキュメントへのリンクが含まれています．また，ライブラリマニュアル全体をテキスト検索することができます．

5 NAG ライブラリの設計と開発

NAG ライブラリの設計と開発に関する種々の見解，NAG の技術的方針と組織に関する情報については Ford (1982), Ford *et al.* (1979), Ford and Pool (1984), Hague *et al.* (1982) をご参照ください．

6 NAG ライブラリの標準準拠

NAG ライブラリは多くの国際標準に準拠しています．ISO Fortran 95 (1997), ANSI (1966), ANSI (1978), ANSI/IEEE POSIX (1995), Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001) をご参照ください．

7 参考文献

ACM (1960–1976) Collected algorithms from ACM index by subject to algorithms

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia http://www.netlib.org/lapack/lug

ANSI (1966) USA standard Fortran *Publication X3.9* American National Standards Institute

ANSI (1978) American National Standard Fortran *Publication X3.9* American National Standards Institute

ANSI/IEEE (1985) IEEE standard for binary floating-point arithmetic *Std 754-1985* IEEE, New York

ANSI/IEEE POSIX (1995) POSIX Standard Thread Library ANSI/IEEE POSIX 1003.1c:1995

Basic Linear Algebra Subprograms Technical (BLAST) Forum (2001) *Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard* University of Tennessee, Knoxville, Tennessee
http://www.netlib.org/blas/blast-forum/blas-report.pdf

Blackford L S, Demmel J, Dongarra J J, Duff I S, Hammarling S, Henry G, Heroux M, Kaufman L, Lumsdaine A, Petitet A, Pozo R, Remington K and Whaley R C (2002) An updated set of *Basic Linear Algebra Subprograms (BLAS) ACM Trans. Math. Software* **28** 135–151

Dongarra J J, Du Croz J J, Duff I S and Hammarling S (1990) A set of Level 3 basic linear algebra subprograms *ACM Trans. Math. Software* **16** 1–28

Dongarra J J, Du Croz J J, Hammarling S and Hanson R J (1988) An extended set of FORTRAN basic linear algebra subprograms *ACM Trans. Math. Software* **14** 1–32

Ford B (1982) Transportable numerical software *Lecture Notes in Computer Science* **142** 128–140
Springer–Verlag

Ford B, Bentley J, Du Croz J J and Hague S J (1979) The NAG Library 'machine' *Softw. Pract. Exper.* **9(1)** 65–72

Ford B and Pool J C T (1984) The evolving NAG Library service *Sources and Development of Mathematical Software* (ed W Cowell) 375–397 Prentice–Hall

Hague S J, Nugent S M and Ford B (1982) Computer-based documentation for the NAG Library *Lecture Notes in Computer Science* **142** 91–127 Springer–Verlag

ISO Fortran 95 (1997) ISO Fortran 95 programming language (ISO/IEC 1539–1:1997)

OpenMP *The OpenMP Specification for Parallel Programming* http://www.openmp.org

The BLAS Technical Forum Standard (2001) http://www.netlib.org/blas/blast-forum

---