NAG DMC

機能のご紹介

Note: 本ドキュメントは NAG DMC のご利用を検討されている方々にとって必須の資料です。

1　はじめに

本ドキュメントは NAG DMC を有効にお使いいただく上で基本となる情報について記述します。最初に NAG DMC の概要紹介と付帯する資料の説明（セクション 2 参照）から始まり、続いて NAG DMC の用法に関するキーとなる事項とそれらに関する説明資料（それぞれセクション 3、セクション 4 参照）について述べます。最後にセクション 5 では NAG のサポート情報についてまとめておきます。

2　機能概要と関連資料

2.1　機能概要

NAG DMC は実務上の問題解決を図る上で役に立つ広範な計算機能（以後、単に**関数**と呼びます）の集合体です。

NAG DMC の機能は次に示す 8 つのカテゴリ、すなわちタスクをカバーしています。

(a) データクリーニング
(b) データ変換
(c) 外れ値検出
(d) クラスタリング
(e) 分類
(f) 回帰
(g) 相関ルール
(h) ユーティリティ

2.2　関連資料

NAG DMC の説明資料は次のものから構成されています。

(i) 上記 8 種類のタスクに関する記述
(ii) 関数の仕様書

ユーザガイド中のカテゴリ (a) から (h) までの記述は該当するタスクに関する背景情報、及び NAG DMC の中のどの関数を選択すべきかに関するガイダンス情報を提供します。

一方、関数の仕様書には C/C++ プログラミング言語から関数をコールする際の仕様、及び分析に用いられるアルゴリズムについて詳細が記述されています。詳しくは“資料の使用について”をご参照ください。

これらのドキュメントは PDF(Portable Document Format) 形式で配布されるので、Adobe Acrobat Reader を用いて読むことができます。なお、Adobe Acrobat Reader は Adobe 社の Web サイト `http://www.adobe.com` から無償でダウンロードできます。また、Internet Explorer や Netscape 等の web ブラウザに適切なプラグインが組み込まれている場合には、ブラウザからの閲覧も可能です。

2.3 オンラインドキュメント

NAG の Web サイト（“サポート情報”参照）には NAG DMC の最新資料や、ダウンロード可能な各種サポートマテリアルに関するニュースが掲載されています。

2.4 実装

NAG DMC は種々のコンピュータシステム上で利用できます。それぞれのプラットフォーム（例えば Sun Solaris）上への NAG DMC の実装は NAG 社によって行われています。それぞれのサイトに適した実装がコンパイル済み、テスト済み関数ライブラリの形で配布されます。

NAG DMC の実装は本質的にみな同一機能を持っていますが、演算機構やコンパイル系の違いにより、異種プラットフォーム間では同一の演算結果が得られるとは限りません。特に数値計算の問題においては注意が必要です。

関連ドキュメントは NAG DMC のすべての実装を対象に書かれていますが、少量の実装固有の情報については User's Note という形で個別に提供されるものもあります。

2.5 数値演算の精度

NAG DMC は倍精度でのみ実装されています。

2.6 C 言語標準

NAG DMC のすべての関数は ANSI C に完全準拠しています。

3 ソフトウェアの使用

3.1 一般的なアドバイス

用いられたデータに関わりなく、NAG DMC の関数から常に有用な結果が得られるという保証はありません。次の点について配慮が必要です。

(i) タスクの定式化
(ii) 関数を使用するためのプログラミング
(iii) 結果の評価

(i) についてはユーザガイドとセクション 3.2 でより詳しく述べます。(iii) については関数の仕様書を参照ください。本セクションの残りの部分では (ii) について説明します。

3.2 タスクの定式化

NAG DMC はデータが数値形式になっていることを仮定します。文字列形式のデータは NAG DMC 関数をコールする前に数値表現に変換されていなくてはなりません。さらにユーザガイドに記述されているように、ダミー変数は 2 つ以上のカテゴリを使ってカテゴリ変数として計算しておく必要がある場合があります。

NAG DMC のほとんどの関数においてはデータ集合の中から一連のデータレコード（部分集合）を選択することができます。これは関数パラメータ **rec1** と **nrec** に適切な値をセットすることで行えます。

データ変数は次のように区分されます。

（i）カテゴリ変数：これは限られた値しか取らない変数であり、順序尺度 (ordinal scale)、名義尺度 (nominal scale) のいずれかに従います。順序尺度に従うデータ値は数直線上での意味を持ちますが、名義尺度に従うデータ値の場合にはその対応はなく、単に他の名義上の値と区別するためだけに用いられます。
（ii）連続変数： 理論上は任意の実数値（スカラー）を取ることができます。

変数はそれらが取る値によって区分されると共に、分析中で果たす役割によっても区分されます。分類、回帰型のタスクの場合、独立変数は関心の対象となる帰結（例えば分類とか予測）に影響を及ぼす（と仮定される）変数です。一方、関心の対象となる帰結に関連した変数は従属変数です。

特定のタスクに対する解を求める際に使用されるデータレコードは分類/回帰モデルに対する訓練集合 (training set) と呼ばれます。これらには独立変数の他に従属変数の値も含まれます。

3.3 プログラミングに関するアドバイス

3.3.1 一般的事項

NAG DMC とその関連資料はユーザが C によるプログラミングに関する知識をお持ちであるとの前提に立って設計され記述されています。ある NAG DMC 関数が選択された場合、それはユーザによって書かれた C プログラム (calling program) からコールされることになります。本マニュアルではユーザがそのようなプログラムを作成するに必要な C 言語に関する知識をお持ちであると仮定しています。サンプルのプログラム (calling programs) が NAG DMC と共に提供されているので、これらを使うと問題解決のためのプロトタイプを迅速に用意することができます。なお、関数の仕様書に書かれている関数パラメータ、配列サイズ、配列インデックスの指定方法には特に注意を払うようにしてください。

3.3.2 配列の参照

C 言語においては次の記法を用いることによって 2 次元の変数を宣言することができます。

```
double a[dim1][dim2];
```

この変数が関数に対するパラメータとして用いられた場合、それはコンパイラにより実質上、スタック上に `dim1*dim2` だけのメモリをアロケートされた `double` 型のポインタ`*a` として扱われます。この配列の要素、例えば a[3][5] のアドレスは `(a+3*dim2+5)` のように計算されます。これは C が行優先順 (row-major order) にデータを格納するためです。この場合、次のようなマクロ定義を用意すればこのアドレス計算のための記述を簡潔な形で行えるようになります。

```
#define A(I,J) a[(I)*dim2+J]
```

別のアプローチとしては次の構文を用いることによって、ユーザ自らが `double *`型のポインタに対し（ヒープ上の）メモリを割り当てることも可能です。

```
a = (double *)malloc(dim1*dim2*sizeof(double));
```

この配列の第 ij 要素に対するインデックスとしては、ポインタ記法`*(a+i*dim2+j)`、配列記法 `a[i*dim2+j];`、あるいはマクロ `A(I,J)` が定義されている場合にはそれを用いて `A`(i,j) のように指定することができます。

配列が対称である場合、NAG DMC はパッキング (packed storage) を用いてメモリの節約を行います。その場合のパッキングは行単位、あるいは列単位に行うことができます。

3.3.3 NAG データタイプ

NAG DMC の関数とのインタフェース上には C 標準のデータタイプ、すなわち `char`, `int`, `long`, `double` のみしか現れません。このため Java や C# 等の他の言語からも NAG DMC は容易に使えるようになっています。

従ってツリー格子のような再帰的なメモリ構造を扱う関数はインタフェース上、これらの構造に対するポインタそのものではなく、ポインタの整数キャスト (integer casts) を返します。この場合、ツリー格子上のノードへのポインタは NAG によって定義された構造体に対する適切なキャストを用いることによって回復できます。これらのキャストによって影響を受ける関数には次のようなものがあります。

nagdmc_entropy_tree
nagdmc_gini_tree
nagdmc_reg_tree
nagdmc_waid

より詳細については関数仕様書のサンプルコードセクションを参照ください。

3.3.4 メモリ管理

メモリは NAG DMC 内部で動的にアロケートされることが良くあります。その場合、すべてのメモリ要求は成功に終わったか失敗だったかがチェックされます。万一失敗に終わった場合、NAG DMC 関数は関数仕様書に記述されているような整数のエラーコードを返します。

時には関数内部でアロケートされたメモリが返される場合があります。その内容に関する記述は関数仕様書のサンプルコードセクションに適宜記されています。

例えば関数の定義

```
long *func(long a, double b);
```

は `long` 型のメモリを返します。このメモリはコールする側のプログラム中で `long *`型のパラメータ名を設定することによってアクセスされます。一例を示すと次のようになります。

```
long *retval;
retval = func(a,b);
```

関数コールからの結果に対する処理が終わったら、返されたメモリはコールした側のプログラムで解放してください。NAG DMC にはそのための関数が用意されています。

メモリの扱いに関する用例については関数仕様書、及び次のサンプルコードを参照してください。

nagdmc_impute_dist
nagdmc_impute_em
nagdmc_impute_simp

3.3.5 オプショナルパラメータ

関数の宣言文中で指定された配列で参照されないものについては 0 にセットしておいてください。関数インタフェース部におけるさらなるエラーチェックが可能になります。

3.3.6 info パラメータ

NAG DMC 内においてはパラメータ **info** が関数の実行結果に関する情報を受け渡す場として使用されます。**info** < 0 の場合は警告であり単なる情報として扱えば良いわけですが、$0 <$ **info** ≤ 100 の場合には重大なエラーにより関数の実行が途中で中断されたことを意味しています。**info** > 100 の場合はやはり関数の実行は途中で中断されますが、これらの値はユーザ提供の関数の場合に限られます。いずれにせよ関数コールを行った場合には必ずパラメータ **info** の値をチェックするようにしてください。

個々の関数から返されるエラーコードの一覧については関数仕様書の 'Parameters' セクションを参照してください。

3.4 データ関数

NAG DMC の関数に対しデータを入力する上で標準的なのはパラメータ **data** を用いる方法です。**data** パラメータは必要なデータ値をすべて含む 1 次元配列を指定します。しかし現実的な問題においては大量のデータを伴うことが少なくありません。そのような場合、すべてのデータを一遍にメモリに持ち込むことは望ましいことではなく、また物理的にも不可能なことがあります。そのような大規模なデータの分析を支援するために、NAG DMC 関数の多くがデータを細分化して扱えるようになっています。

この data chunking の機能は 3 つのパラメータ **dfun**, **comm**, **chunksize** を指定することによって有効になります。パラメータ **dfun** は一塊 (chunk) のデータを読み込むためのユーザ提供の関数を指定します。**comm** はその **dfun** に対しさらなるパラメータの受け渡しを可能にします。また **chunksize** は扱うデータのサイズをコントロールします。次は **dfun** の原型を示したものです。

```
void dfun(long irec, long chunksize, double x[], char *comm, int *ierr)
```

1: **irec** – `long` *Input*
On entry: 戻されるべき先頭のレコードに対するインデックス

2: **chunksize** – `long` *Input*
On entry: 戻されるべき連続したレコード数

3: `x`[**chunksize**∗**nvar**] – `double` *Output*
On exit: j 番目の変数 ($j = 0, 1, \ldots,$ **nvar** -1) に対する値は `x`[i∗**nvar** $+ j$] という形式で戻されなくてはならない。ただし $i = 0, 1, \ldots,$ **chunksize** -1。

4: **comm** – `char *` *Input*
On entry: **dfun** に対し引き渡される追加情報に関するパラメータ。このパラメータは'as is' で引き渡される。

5: **ierr** – `int *` *Output*
On exit: **ierr** によってポイントされる値が 100 より大きい場合、NAG DMC は即時実行を停止し、**info** がこの値をポイントすることになる。

データ関数が唯一実行しなくてはならないことは **irec** から **irec** + **chunksize** -1 までのデータレコードを配列 `x` 中にコピーすることです。しかし他のタスクを実行しても構いません。例えば回帰分析に先立ち、データのクリーニング、コード変換、カテゴリデータからのダミー変数の生成といった処理を行うことが考えられます。これらのタスクをデータ関数中で実行するメリットは、それらを少量のデータを単位に実行できる点にあります。

3.4.1 データ関数の例

サンプルのデータ関数 **nagdmc_dfun_basic** が NAG DMC と共に提供されています。この関数は入力ストリーム中から一塊のデータレコードを読み込む機能を持っています。データレコードを細分化された単位で読み込むため、その分のメモリしか必要としません。**nagdmc_dfun_basic** のような関数を以下にリストした分析関数の一つと併用することにより、どんなに大きなデータでも分析が可能になります。**nagdmc_dfun_basic** がどのように作られているかを示すために、コードの一部を注釈と共に記しておきます。このコードはファイル **nagdmc_dfun_basic.c** から抽出されたものであり、ライン番号はそのファイル上での行番号に対応します。

```
 1 void nagdmc_dfun_basic(long irec, long chunksize, double x[], char *comm,
                          int *ierr) {
 3   long r, v;
 4   FILE *fp;
 5   static long total_read_in = 0;
 6   long crec1, cnvar, cnrec;
10   sscanf(comm, "%d %d %d %p", &crec1, &cnvar, &cnrec, &fp);
46   if (feof(fp) || total_read_in >= (cnrec + crec1)) {
47     total_read_in = 0;
48     rewind(fp);
49   }
51   if (total_read_in == 0 && crec1 != 0) {
53     for (r = 0; r < crec1 && !feof(fp); r++) {
54       for (v = 0; v < cnvar && !feof(fp); v++)
55         fscanf(fp, "%*lf");
57       total_read_in++;
58     }
59   }
62   *ierr = 0;
65   for (r = 0; r < chunksize && !feof(fp); r++) {
66     for (v = 0; v < cnvar && !feof(fp); v++)
67       fscanf(fp, "%lf ", &x[r * cnvar + v]);
69     total_read_in++;
70   }
79  return;
80 }
```

行番号	説明
1	関数定義
3 - 5	変数定義
10	パラメータ **comm** で引き渡されたパラメータを回復する。
	crec1 入力ストリーム中から読み込むべき最初のレコード。これはパラメータ **rec1** に類似のもので同じ値がセットされていなくてはならない。
	cnrec 入力ストリームから読み込むべきデータレコードの総数。これはパラメータ **nrec** に類似のもので同じ値がセットされていなくてはならない。**cnrec** はチェックの目的でのみ使用される。
	fp 入力ストリームに対するポインタ。

行番号	説明
46 - 49	関数を呼ぶ側は入力データの中を何回もループできるものでなくてはならない。従ってファイルの末尾に到達した場合、または要求された数のデータレコードが読み込まれた場合には入力ストリームを rewind し、カウンタ **total_read_in** をリセットする。
51 - 59	入力ストリーム中の最初の **crec1** 個のデータをスキップする。
62	エラーコードを正常終了を示す 0 に初期化する。
65 - 70	各々 **cnvar** 個の変数を含む **chunksize** 個のデータレコードを読込み、x 中に格納する。
69	読み込まれたデータレコードの総数をカウントする。

nagdmc_dfun_basic のフルバージョンには上記以外のコードも含まれています。それらは主にユーザから指定されたパラメータのチェックを行うものです。

行番号	説明
9 - 31	**comm** が 0 でない場合には **comm** 内で指定されたパラメータをチェックし、想定外の値が検出された場合にエラーコード 101-104 を返す。
32 - 36	**comm** が 0 の場合にエラーコード 105 を返す。
39 - 43	**chunksize** を 0 にセットして **nagdmc_dfun_basic** をコールすることにより、エラーチェックカウンタ **total_read_in** のリセットと入力ストリームの rewind を可能にする。
72 - 77	入力ストリームの末尾に到達した際、想定される数のデータレコード（**total_read_in = crec1 + cnrec**）が読み込まれているかどうかをチェックする。

nagdmc_dfun_basic 中のその他のコードは空白行か、もしくはコメントを含まないものです。

入力ストリームのシーケンシャルな性格からパラメータ **irec** は **nagdmc_dfun_basic** 中で参照されていない点に注意してください。従って **irec** の用途について簡単な説明を加えておきます。**dfun** が最初にコールされた段階ではパラメータ **irec** の値はコール元関数の **rec1** と同じ値になっています。しかし次に **dfun** がコールされたときには **irec = rec1 + chunksize**, n 回目のコールでは **irec = rec1 + (n - 1) *chunksize** となっています。

3.4.2 データ関数の使用例

次の注釈付きのコードはデータ関数 **nagdmc_dfun_basic** を用いた関数 **nagdmc_linear_reg** をコールした例を示しています。

```
1 long rec1 = 0, nvar = 5, nrec = 4, dblk = 4, nxvar = 0,
      *xvar = 0, iwts = -1, yvar = 0, chunksize = 10;
2 double r2, rms, b[5], se[5], cov[15], model[48], eps = 0.0;
3 int info;
4 char *comm[20];
5 FILE *infp;
6 infp = fopen("example1.dat", "r");
7 sprintf(comm, "%d %d %d %p", rec1, nvar, nrec, infp);
```

```
8 nagdmc_linear_reg(rec1, nvar, nrec, dblk, 0, nagdmc_dfun_basic, comm,
                    chunksize, nxvar, xvar, yvar, iwts, &r2, &rms, &df,
                    b, se, cov, model, eps, &info);
```

行番号	説明
1 - 3	**nagdmc_linear_reg** で使用する変数の宣言
4	communication パラメータ用のスペース確保
5	入力ファイル用ポインタ
6	入力ファイルのオープン
7	communication パラメータに対し、先頭のレコード **rec1**, 変数の数 **nvar**, データレコード数 **nrec**, ファイルポインタ **infp** の値をセットする。
8	分析関数をコールする。データ関数が使用されているので **data** パラメータは 0 にセットされる。

データ関数を用いつつ分析を実行した例はサンプルファイル **nagdmc_reg_dfun_example.c** においても見られます。

3.4.3 分析関数

次の分析関数でデータ関数を使用することができます。

nagdmc_binomial_reg	2 項分布に従う誤差を持った線形モデル
nagdmc_dsu	サマリ統計のためのユーティリティ関数
nagdmc_linear_reg	正規分布に従う誤差を持った線形モデル
nagdmc_nrgp	クラスタリングに基づきデータレコードをグループに割り振る
nagdmc_pca	主成分分析
nagdmc_poisson_reg	ポアッソン分布に従う誤差を持った線形モデル
nagdmc_wcss	クラスタ内での 2 乗和

3.5 ライセンス管理

状況が適切であれば、NAG DMC 関数は NAG DMC に対する正当なライセンスが見当たらないことを示す −999 というコードを **info** パラメータを介して返すことがあります。

4 資料の使用について

4.1 一般的なアドバイス

ユーザガイドの先頭部に共通的な 8 つのタスクの一覧が示されています。新規のユーザはこれらのタスクのうちどれを実行したいかを決定することから始めてください。

タスクの選択が済んだら次に該当する背景情報を参照し、NAG DMC 中の適切な関数を選択します。特定のタスクに対し複数の関数が存在する場合には表を参考にした上で、適切な関数の選択を行ってください。

関数が特定されたら関数の仕様書を参照してください。それぞれの関数仕様書は実質上自己完結型に作られています（関連するドキュメントへの参照は含まれていますが)。仕様書には関数で用いられる手法の記述、関数パラメータの詳細仕様、エラー/情報コードに関する説明が含まれています。

最後に該当する関数を用いたサンプルプログラムが用意されているので、必要に応じて参照してください。ユーザご自身のタスクを開発する上でのベースとして使用することもできます。サンプルプログラムのコンパイル、リンク方法についてはプラットフォーム固有の Users' Note に記されています。

ドキュメントは PDF ファイル中に埋め込まれているリンクを使ってナビゲートできます。これらのリンクは青で表示されています。前のドキュメントに戻るには Adobe の Acrobat Reader、または web ブラウザ上で "前へ" というボタンをクリックしてください。

4.2 関数仕様書の構成

すべての関数仕様書には次のような見出しで始まるセクションが含まれています。

概要
関数の宣言
パラメータ
注釈
説明
参考資料
関連サブルーチン

また少数の仕様書（主に決定木とデータ代入関連）については次のセクションがある場合もあります。

サンプルコード

ここでは各見出しごとに概要を紹介します。

4.2.1 概要

関数の用途に関する簡単な説明。

4.2.2 関数の宣言

パラメータのタイプを規定する C 言語による関数の宣言。

4.2.3 パラメータ

箇条書き形式による各パラメータの記述。パラメータは次のように分類されます。

Input: 関数を実行する際にはこれらのパラメータに対し値をアサインしておく必要があります。これらの値は関数から exit した際も変化しません。

Output: 関数を実行する際にはこれらのパラメータに対し値をアサインしておく必要はありません。関数の中で値がセットされます。

Input/Output: 関数を実行する際にこれらのパラメータに対し値をアサインしておく必要がありますが、これらは関数中で値が変更されます。

External Procedure: コール元のプログラムの一部として提供可能な関数。その仕様書にはパラメータの一覧とパラメータ仕様詳細が含まれています。

その他のキーワードについても記しておきます。

'*Constraint*', '*Constraints*': *Input* パラメータに対して正当な値の範囲を規定します。次はその一例です。

Constraint: $\mathbf{rec1} \geq 0$

パラメータに対し不正な値が指定された場合（上の例では例えば $\mathbf{rec1} = -1$）には関数はエラー exit します。

'*Suggested Value*': *Input* パラメータに対し妥当な初期値を示唆します（例えば精度とか、反復回数の最大値等)。示された値が直面している問題に対し不適切であった場合には適宜別の設定を行ってください。

4.2.4 **注釈**

関数の宣言中で用いられているパラメータ名とアルゴリズムの記述（次項参照）中で用いられている用語との間の関連付けを行います。これによってアルゴリズムに関する数学的な背景と実際の実装との間の対応が理解しやすくなります。

4.2.5 **説明**

関数中で用いられているアルゴリズムに関する記述、及び関連事項が含まれています。

4.2.6 **参考資料**

説明の項で参照された資料の他、背景技術に関する資料が一覧として提示されます。

4.2.7 **関連サブルーチン**

他の関連する関数へのハイパーリンクリスト。例えばそれぞれの GLM（generalised linear model）関数の 'See Also' セクションには、フィットされた GLM から情報を抽出するための関数がリンクされています。

4.2.8 **サンプルコード**

本文中の論点の解説用に C のソースコードが紹介されます（例： メモリ内において再帰的データ構造を辿る機能)。

5　サポート情報

(a) ご質問等

保守サービスにご加入いただいているお客様は、電子メール、FAX、又は電話にてお問い合わせ下さい。その際に製品コード、及び保守 ID を御明記いただきますようお願い致します。受付は平日９：００～１２：００、１３：００～１７：００となります。

電子メール：　naghelp@nag-j.co.jp
ＦＡＸ：　03-5542-6312
ＴＥＬ：　03-5542-6311

(b) NAG のＷｅｂサイト

ＮＡＧのＷｅｂサイトではＮＡＧ製品及びサービス情報など定期的に更新を行っております。Ｗｅｂサイトアドレスは以下のとおりです。

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

(c) ユーザフィードバック

NAG ではユーザ様よりフィードバックをバージョンアップなどに活かして行きたいと考えています。フィードバックに御協力いただける場合は、以下のコンタクト先に記述されている連絡先にご連絡下さい。

コンタクト先情報

日本ニューメリカルアルゴリズムズグループ株式会社
（略称：日本ＮＡＧ）
東京都中央区八丁堀４－９－９ 八丁堀フロンティアビル２Ｆ

Tel: 03-5542-6311
Fax: 03-5542-6312
電子メール: naghelp@nag-j.co.jp

※ 日本ニューメリカルアルゴリズムグループ株式会社より提供されるサービス内容等は日本国内ユーザ様向け独自のものとなっております。（お問い合わせ先等）