NAG DMC

Essential Introduction

Note: *this document is essential reading for any prospective user of the NAG DMC.*

1 Introduction

This document describes information fundamental to the successful use of the NAG DMC. It begins with a description of NAG DMC and the accompanying documentation (see Section 2) and thereafter describes key issues regarding the use of NAG DMC and its documentation (see Section 3 and Section 4, respectively). Finally, Section 5 gives details of NAG's support facilities.

2 The Functions and their Documentation

2.1 The Functions

NAG DMC is a comprehensive collection of computational functions, hereafter referred to as *functions*, that can be used to solve a wide range of practical tasks.

The functionality of NAG DMC covers the eight broad categories or tasks:

(a) data cleaning;
(b) data transformations;
(c) outlier detection;
(d) clustering;
(e) classification;
(f) regression;
(g) association rules;
(h) utility functions.

2.2 The Documentation

Documentation on NAG DMC consists of:

(i) descriptions of the eight tasks listed above;
(ii) function documents.

The description of each of the categories (a) to (h) in the User Guide gives background information on the task and guidance towards the selection of appropriate functions in NAG DMC to solve it.

Function documents give detail on calling the function from the C/C++ programming language and the algorithm used in the analysis, see 'Using the Documentation' for more information.

The documentation is delivered in Portable Document Format (PDF) and can be viewed and navigated by using the Adobe Acrobat Reader which can be downloaded free of charge from the Adobe Web Site at `http://www.adobe.com`. The documentation may also be viewed and navigated in a web-browser, e.g., Internet Explorer or Netscape, if that web-browser has the suitable plug-in.

2.3 Alternative Form of Documentation

The NAG web site (see 'Support from NAG') contains up-to-date documentation on NAG DMC and news of support material available to download.

2.4 Implementations

NAG DMC is available on many different computer systems. An implementation of NAG DMC is prepared by NAG for each distinct system, e.g., the Sun Solaris implementation. The implementation is distributed to sites as a tested, compiled library of functions.

Essentially the same facilities are provided in all implementations of the NAG DMC , but, because of differences in arithmetic behavior and in the compilation system, functions cannot be expected to give identical results on different systems, especially for sensitive numerical problems.

The documentation supports all implementations of the NAG DMC, with the help of a few simple conventions, and a small amount of implementation-dependent information, which is published in a User's Note for each implementation.

2.5 Numerical Precision

NAG DMC is developed in double precision only.

2.6 C Language Standards

All functions in NAG DMC conform fully to ANSI C.

3 Using the Software

3.1 General Advice

A NAG DMC function cannot be guaranteed to return meaningful results irrespective of the data supplied to it. Care and thought must be exercised in:

(i) formulating a task;
(ii) programming the use of functions;
(iii) assessing the quality of results.

The User Guide and Section 3.2 provide more detail on (i); the function documents provide information on (iii); the remainder of this section is concerned with (ii).

3.2 Formulating a Task

NAG DMC assumes that data values are supplied in numeric form. Any data mappings from character string values to a numeric representation must be completed before calling NAG DMC functions. Furthermore, as described in the User Guide, dummy variables may need to be calculated for categorical variables with more than two categories.

Most functions in NAG DMC allow the user to select a subset of consecutive data records from a set of data, which can be achieved by setting appropriate values for the function parameters **rec1** and **nrec**.

Data variables are classified as:

(i) categorical variables which are variables that can take only a limited number of values, and may be on either the ordinal or nominal scale. Data values on the ordinal scale have a meaning on a number line, whereas the value of a nominal variable is arbitrary and only serves to distinguish between other nominal values.
(ii) continuous variables that, in theory, can take the value of any real scalar.

In addition to being classified according to the values they may take, variables are also classified in terms of their role in an analysis. For classification and regression tasks an independent variable is one which (we assume) influences the outcome of interest, e.g., a classification or prediction. The variable associated with an outcome of interest is a dependent variable.

Data records collected for use in solving a particular task are referred to as a training set of data for classification and regression models. These sets of data include values for dependent as well as independent variables in the data.

3.3 Programming Advice

3.3.1 General

NAG DMC and its documentation are designed on the assumption that the user knows how to write a calling program in C. When a suitable NAG DMC function has been selected it must be called using a suitable user-written C program, the calling program. This manual assumes that the user has sufficient knowledge of the C programming language to be able to write such a program. Example calling programs are provided with NAG DMC, and these can be used for fast prototyping of solutions. The user is also recommended to pay particular attention to the specification of the function parameters, array sizes and array indices in function documents.

3.3.2 Array References

In C it is possible to declare a two-dimensional variable using notation of the form:

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

When this variable is a parameter to a function, it is effectively treated by the compiler as a pointer, `*a`, of type `double` with an allocated memory of `dim1*dim2` on the stack. The address of an element of this array, say `a`[3][5], is then an explicit address computed to be (`a`+3*`dim2`+5), since C stores data in row-major order. The C pre-processor allows a succinct notation for computing this explicit address by using a macro definition:

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

Alternatively it is possible for a user to allocate memory explicitly (on the heap) to a pointer of type `double *`, using the form:

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

The ijth element of this array is then indexed using the pointer notation `*(a+i*dim2+j)` or by using the array notation `a[i*dim2+j]`; or by using `A`(i, j) assuming the macro `A(I,J)` is already defined.

If an array is symmetric, NAG DMC will save memory by using packed storage. Symmetric arrays can be packed by row or by column.

3.3.3 NAG Data Types

NAG DMC have been written so that the interface to each function includes only standard C types, i.e., `char`, `int`, `long` and `double`. This makes NAG DMC easy to call from some other languages, e.g., Java by using the Java Native Interface, and C#.

Hence functions that compute recursive memory structures, such as tree lattices, return in their interfaces integer casts of pointers to these structures instead of pointers to the structures themselves. The pointer to a node structure in a tree lattice is recovered by an appropriate cast to a NAG defined structure. The functions affected by these casts are:

nagdmc_entropy_tree;
nagdmc_gini_tree;
nagdmc_reg_tree;
nagdmc_waid.

The 'Explanatory Code' sections of the function documents for these functions should be consulted for further detail.

3.3.4 Memory Management

Memory is frequently dynamically allocated within the NAG DMC. All requests for memory are checked for success or failure. In the unlikely event of failure occurring the NAG DMC function returns an integer error code as documented in its function document.

Occasionally, functions return memory allocated within the function. Where appropriate, a description of the contents of the returned memory is given in its function document under the 'Explanatory Code' section.

For example, the function declaration:

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

returns memory of type `long`. This memory would be accessed through a parameter name in a calling program with a C type of `long *`, e.g.,

```
long *retval;

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

Once the results from a successful function call have been processed, the returned memory should be returned to the operating system by the user adding to the calling program the code. NAG DMC provides functions for this purpose.

For full examples dealing with returned memory see the function documentation and example code for NAG DMC functions:

nagdmc_impute_dist;

nagdmc_impute_em;
nagdmc_impute_simp.

3.3.5 Optional Parameters in Functions

Arrays specified in the declaration of a function that, because of the values of other parameters, will not be referenced should be set to 0. This allows for further error checking in the function interface.

3.3.6 info Parameter

Within NAG DMC the parameter **info** is used to return information about the exit status of a function. By convention, values of **info** < 0 are warnings and can be considered for information purposes only, values of $0 <$ **info** ≤ 100 are potentially serious errors and will have caused the function to exit prematurely. Values of **info** > 100 also cause the function to exit prematurely, however, these values are reserved for user-supplied functions. Therefore, the value of the parameter **info** must be checked after the return of each function call.

A full list of the error codes that each function may return is given in the 'Parameters' section of the function documentation.

3.4 Data Function

The standard method for supplying data to the functions within NAG DMC is through the parameter **data**. The **data** parameter specifies a one-dimensional array which holds all of the data values of interest. However, many practical problems involve large amounts of data. In such cases it is not always desirable, or even physically possible, to hold all of the data in memory at one time. In order to facilitate the analyses of large sets of data, a number of NAG DMC functions allow the data to be processed in chunks, i.e., by holding in memory at any one time only a subset of the data.

Data chunking is enabled by the use of the three parameters: **dfun**, **comm** and **chunksize**. The parameter **dfun** specifies a user-supplied function which reads in a chunk of data, **comm** allows additional parameters to be passed to **dfun** and **chunksize** controls the size of the data chunks being handled. The prototype for **dfun** is as follows:

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

1: **irec** – `long` *Input*
On entry: the index in the data of the first record returned.

2: **chunksize** – `long` *Input*
On entry: the number of consecutive records returned.

3: **x**[**chunksize**$*$**nvar**] – `double` *Output*
On exit: data values for the jth variable (for $j = 0, 1, \ldots,$ **nvar** $- 1$) must be returned in **x**$[i *$ **nvar** $+ j]$, for $i = 0, 1, \ldots,$ **chunksize** $- 1$.

4: **comm** – `char *` *Input*
On entry: a communication parameter allowing additional information to be passed to **dfun**. This parameter is passed 'as is' through the calling function.

5: **ierr** – `int *` *Output*
On exit: if the value pointed to by **ierr** on return is greater than 100, the NAG DMC function will terminate immediately and **info** will point to this value.

The only operation a data function must do is to copy data records **irec** to **irec** + **chunksize** $- 1$ into the array **x**, however, other tasks can be performed as well. For example, data can be cleaned, recoded or dummy variables constructed from categorical data, prior to a regression analysis. The benefit of performing these tasks within the data function is that they can be performed on small chunks of data rather than the entire set of data.

3.4.1 An Example Data Function

An example data function, **nagdmc_dfun_basic**, is supplied with NAG DMC. This function reads chunks of data records from an input stream. As the data records are read from the stream in chunks, only a small proportion of the set of data needs to be in memory at any one time. Using a function like **nagdmc_dfun_basic**, and one of the analysis functions listed below, allows sets of data records of an arbitrarily large size to be analysed. In order to illustrate how **nagdmc_dfun_basic** was constructed annotated code extracts are given below. These extracts are taken from the file **nagdmc_dfun_basic.c**, and the line numbering refers to the lines in that file.

```
 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 }
```

Lines	Description
1	function definition.
3 to 5	variable declarations.
10	Recover the user-supplied parameters from the communications parameter **comm**:
	crec1 the first record to be read from the input stream. This is analogous to the parameter **rec1** in the calling function and should be set to the same value.
	cnrec the total number of data records that to be read from the input stream. This is analogous to the parameter **nrec** in the calling function and should be set to the same value. **cnrec** is used only for checking purposes.
	fp pointer to the input stream.
46 to 49	The calling function needs to be able to loop through the input data a number of times. Therefore if the end of the file has been reached, or the required number of data records have been read in, rewind the input stream and reset the counter **total_read_in**.
51 to 59	skip the first **crec1** data records in the input stream.
62	initialise the error code to zero, indicating no errors have occurred.
65 to 70	read in **chunksize** data records, each containing **cnvar** variables, storing the data in **x**.

69 count the total number of data records read in.

The full version of **nagdmc_dfun_basic** has additional lines of code, mainly for checking the user-supplied parameters:

Lines	Description
9 to 31	If **comm** is not 0, check the parameters supplied within **comm** and return error codes 101 to 104 if they take unexpected values.
32 to 36	return error code 105 if **comm** is 0.
39 to 43	allow the user the option of resetting the error checking counter **total_read_in** and rewinding the input stream by calling **nagdmc_dfun_basic** on its own, with **chunksize** set to zero.
72 to 77	checks that when the end of the input stream has been reached the expected number of data records have been read in, that is **total_read_in** = **crec1** + **cnrec**.

All other lines of code in the **nagdmc_dfun_basic**, not mentioned above, are either blank or contain comments.

It should be noted that, because of the sequential nature of input streams, the parameter **irec** is not referenced in **nagdmc_dfun_basic**, therefore a brief description of the values taken by **irec** is given here. When **dfun** is first called, the parameter **irec** takes the same value as the parameter **rec1** from the calling function. At the next call to **dfun**, **irec** = **rec1** + **chunksize**, and at the nth call, **irec** = **rec1** + $(n-1) *$ **chunksize**.

3.4.2 Using a Data Function

The following is an annotated code extract for calling the function **nagdmc_linear_reg** with the data function **nagdmc_dfun_basic**:

```
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);
```

1 to 3	variable declarations used by **nagdmc_linear_reg**.
4	allocate space for the communication parameter.
5	pointer for the input file.
6	open the input file.
7	populate the communication parameter with the first record, **rec1**, number of variables, **nvar**, the number of data records, **nrec** and the file pointer, **infp**.
8	call an analysis function. As a data function is being used the **data** parameter is set to 0.

A further example of an analysis performed whilst using a data function can be found in the example file **nagdmc_reg_dfun_example.c**.

3.4.3 Functions

The following functions facilitate the use of a data function:

nagdmc_binomial_reg	linear model with binomial errors;
nagdmc_dsu	utility function for data summary statistics;
nagdmc_linear_reg	linear model with Normal errors;
nagdmc_nrgp	allocates data records to groups given a clustering;
nagdmc_pca	principal components analysis;
nagdmc_poisson_reg	linear model with Poisson errors;
nagdmc_wcss	within-cluster sums of squares.

3.5 Licence Management

Where appropriate, NAG DMC functions return an integer code of −999 through the **info** parameter indicating that a valid licence for NAG DMC was not found.

4 Using the Documentation

4.1 General Advice

At the beginning of the User Guide is a list of eight common tasks. New users should begin by determining which of these tasks they wish to accomplish.

Having found a task to accomplish, the user should read the appropriate background information and select an appropriate function in the NAG DMC. Where more than one function exists for a particular task, a table is included which gives additional information to help guide the user to an appropriate function.

Once a function has been selected, the user must consult the function document. Each function document is essentially self-contained (it may, however, contain references to related documents). It includes a description of the method used by the function, detailed specifications of the function parameters and explanations of error and information code exits.

Finally, if necessary, an example program that uses the function can be viewed for further information or used as a basis for your own task; information regarding how to compile and link the example program is given in the Users' Note for your implementation.

Documents may be navigated by using links embedded in the corresponding PDF file. These links are coloured blue. Previous documents can be recalled by using the back button on Adobe's Acrobat Reader or a web-browser.

4.2 Structure of Function Documents

All function documents contain information under the seven headings:

Purpose
Declaration
Parameters
Notation
Description
References and Further Reading
See Also

In a few documents (notably the decision tree and data imputation functions) there is the further heading:

Explanatory Code

Descriptions of these eight headings now follow.

4.2.1 Purpose

A brief description of the purpose of the function.

4.2.2 Declaration

The C language declaration of the function giving parameter types.

4.2.3 Parameters

A description of each parameter in an ordered, numerated list. Parameters are classified as follows:

Input: you must assign values to these parameters on or before entry to the function, and these values are unchanged on exit from the function.

Output: you need not assign values to these parameters on or before entry to the function; the function may assign values to them.

Input/Output: you must assign values to these parameters on or before entry to the function, and the function may then change these values.

External Procedure: a function which may be supplied as part of your calling program. Its specification includes full details of the function's parameter list and specifications of its parameters (all enclosed in a box).

The word '*Constraint*:' or '*Constraints*:' in the specification of an *Input* parameter introduces a statement of the range of valid values for that parameter, e.g.,

Constraint: **rec1** ≥ 0.

If the function is called with an invalid value for the parameter (e.g., **rec1** $= -1$), the function will take an error exit.

The phrase '*Suggested Value*:' introduces a suggestion for a reasonable initial setting for an *Input* parameter (e.g., accuracy or maximum number of iterations) in case you are unsure what value to use; you should be prepared to use a different setting if the suggested value turns out to be unsuitable for your analysis.

4.2.4 Notation

Under this heading, where appropriate, links are forged between parameter names used in the function declaration and the notation used in the algorithmic description (see below). These links make it easier to understand the relationship of the background mathematics of an algorithm to its implementation as a function.

4.2.5 Description

A full description of the algorithm used in the function is given, in addition to any relevant computational issues.

4.2.6 References and Further Reading

Where appropriate, a list of text referred to in the Description of a function and general background reading.

4.2.7 See Also

A hyper-linked list of other relevant functions, e.g., the function to extract information from a fitted generalised linear model (GLM) is listed under 'See Also' for each GLM function.

4.2.8 Explanatory Code

Under this heading additional C source code is given to aid the explanation of a point in the text, e.g., a function used to step through recursive data structures in the memory of a computer.

5 Support from NAG

5.1 Contact with NAG

Queries concerning this library should be directed initially to your local Advisory Service. If you have difficulty in making contact locally, you can contact NAG directly.

5.2 NAG Response Centres

The NAG Response Centres are available for general enquiries from all users and also for technical queries from users with Support.

The Response Centres are open during office hours, but contact is possible by fax, email and telephone (answering machine) at all times. Please see the Users Note or the NAG web sites for contact details.

When contacting one of our Response Centres it helps us to deal with you query quickly if you can quote your NAG user reference and NAG product code.

5.3 NAG Web Site

The NAG web site is an information service providing items of interest to users and prospective users of NAG products and services. The information is regularly updated and reviewed, and includes implementation availability, descriptions of products, down-loadable software and technical reports. NAG web sites can be accessed at:

`http://www.nag.co.uk` or
`http://www.nag.com` or
`http://www.naggmbh.de` or
`http://www.nag-j.co.jp`