NAG Library Function Document

nag_blgm_lm_describe_data (g22ybc)

Note: please be advised that this function is classed as 'experimental' and its interface may be developed further in the future. Please see Section 3.1.1 in How to Use the NAG Library and its Documentation for further information.

1 Purpose

nag_blgm_lm_describe_data (g22ybc) describes a data matrix.

2 Specification

```
#include <nag.h>
#include <nagg22.h>
void nag_blgm_lm_describe_data (void **hddesc, Integer nobs, Integer nvar,
     const Integer levels[], Integer lvnames, const char *vnames[],
     NagError *fail)
```

3 Description

Let D denote a data matrix with n observations on m_d independent variables, denoted $V_1, V_2, \ldots, V_{m_d}$. The jth independent variable, V_j can be classified as either binary, categorical, ordinal or continuous, where:

Binary
: V_j can take the value 1 or 0.

Categorical
: V_j can take one of L_j distinct values or levels. Each level represents a discrete category but does not necessarily imply an ordering. The value used to represent each level is therefore arbitrary and, by convention and for convenience, is taken to be the integers from 1 to L_j.

Ordinal
: As with a categorical variable V_j can take one of L_j distinct values or levels. However, unlike a categorical variable, the levels of an ordinal variable imply an ordering and hence the value used to represent each level is not arbitrary. For example, $V_j = 4$ implies a value that is twice as large as $V_j = 2$.

Continuous
: V_j can take any real value.

nag_blgm_lm_describe_data (g22ybc) returns a G22 handle containing a description of a data matrix, D. The data matrix makes no distinction between binary, ordinal or continuous variables.

A name can also be assigned to each variable. If names are not supplied then the default vector of names, {'V1','V2',...} is used.

4 References

None.

5 Arguments

1: **hddesc** – void ** *Input/Output*

On entry: must be set to **NULL**.

As an alternative an existing G22 handle may be supplied in which case this function will destroy the supplied G22 handle as if **nag_blgm_handle_free (g22zac)** had been called.

On exit: holds a G22 handle to the internal data structure containing a description of the data matrix, D. You **must not** change the G22 handle other than through the functions in Chapter g22.

2: **nobs** – Integer *Input*

On entry: n, the number of observations in the data matrix, D.

Constraint: **nobs** ≥ 0.

3: **nvar** – Integer *Input*

On entry: m_d, the number of variables in the data matrix, D.

Constraint: **nvar** ≥ 0.

4: **levels**[**nvar**] – const Integer *Input*

On entry: **levels**$[j-1]$ contains the number of levels associated with the jth variable of the data matrix, for $j = 1, 2, \ldots,$ **nvar**.

If the jth variable is binary, ordinal or continuous, **levels**$[j-1]$ should be set to 1; otherwise **levels**$[j-1]$ should be set to the number of levels associated with the jth variable and the corresponding column of the data matrix is assumed to take the value 1 to **levels**$[j-1]$.

Constraint: **levels**$[i-1] \geq 1,$ for $i = 1, 2, \ldots,$ **nvar**.

5: **lvnames** – Integer *Input*

On entry: the number of variable names supplied in **vnames**.

Constraint: **lvnames** $= 0,$ or **nvar**.

6: **vnames**[**lvnames**] – const char * *Input*

On entry: if **lvnames** $\neq 0$, **vnames**$[j-1]$ must contain the name of the jth variable, for $j = 1, 2, \ldots,$ **nvar**. If **lvnames** $= 0$, **vnames** is not referenced and may be **NULL**.

The names supplied in **vnames** should be at most 50 characters long and be unique. If a name longer than 50 characters is supplied it will be truncated.

Variable names must not contain any of the characters +.*-:^()@.

7: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_ARRAY_SIZE

On entry, **lvnames** $= \langle value \rangle$ and **nvar** $= \langle value \rangle$.
Constraint: **lvnames** $= 0,$ or **nvar**.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

NE_HANDLE

On entry, **hddesc** is not **NULL** or a recognised G22 handle.

NE_INT

On entry, **nobs** $= \langle value \rangle$.
Constraint: **nobs** ≥ 0.

On entry, **nvar** $= \langle value \rangle$.
Constraint: **nvar** ≥ 0.

NE_INT_ARRAY

On entry, $j = \langle value \rangle$ and **levels**$[j-1] = \langle value \rangle$
Constraint: **levels**$[i-1] \geq 1$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_INVALID_FORMAT

On entry, variable name i contains one more invalid characters, $i = \langle value \rangle$.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_NON_UNIQUE

On entry, variable names i and j are not unique (possibly due to truncation), $i = \langle value \rangle$ and $j = \langle value \rangle$.
Maximum variable name length is 50.

On entry, variable names i and j are not unique, $i = \langle value \rangle$ and $j = \langle value \rangle$.

NW_TRUNCATED

At least one variable name was truncated to 50 characters. Each truncated name is unique and will be used in all output.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_blgm_lm_describe_data (g22ybc) is not threaded in any implementation.

9 Further Comments

None.

10 Example

This example performs a linear regression using **nag_regsn_mult_linear (g02dac)**. The linear regression model is defined via a text string which is parsed using **nag_blgm_lm_formula (g22yac)**. The corresponding design matrix associated with the model and the dataset described via a call to **nag_blgm_lm_describe_data (g22ybc)** is generated using **nag_blgm_lm_design_matrix (g22ycc)**.

Verbose labels for the parameters of the model are constructed using information returned in **vinfo** by **nag_blgm_lm_submodel (g22ydc)**.

See also the examples in **nag_blgm_lm_formula (g22yac)**, **nag_blgm_lm_design_matrix (g22ycc)** and **nag_blgm_lm_submodel (g22ydc)**.

10.1 Program Text

```
/* nag_blgm_lm_describe_data (g22ybc) Example Program.
 *
 * Copyright 2017 Numerical Algorithms Group.
 *
 * Mark 26.1, 2016.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>
#include <nagg22.h>

#define MAX_FORMULA_LEN 200
#define MAX_VNAME_LEN 200
#define MAX_PLAB_LEN 200
#define MAX_CVALUE_LEN 200

#define DAT(I,J) dat[j*pddat+i]

char *read_line(char formula[],Integer nchar);
Integer construct_labels(Integer ip, char **plab[], char *const vnames[],
                         Integer vinfo[]);
Integer fit_lm(void *hform,Nag_IncludeIntercept intcpt,Integer nobs,Integer mx,
           double x[],Integer ldx,Integer isx[],Integer ip,double y[],
           char *plab[]);

int main(void)
{
  /* Integer scalar and array declarations */
  Integer i, j, ip = 0, pddat, ldx, lisx, lplab = 0, lvinfo,
    lvnames = 0, mx, nobs, nvar, sddat, sdx, lenlab;
  Integer exit_status = 0;
  Integer *isx = 0, *levels = 0, *vinfo = 0;
  Integer tvinfo[3];

  /* Nag Types */
  NagError fail;
  Nag_IncludeIntercept intcpt;

  /* Double scalar and array declarations */
  double *dat = 0, *x = 0, *y = 0;

  /* Character scalar and array declarations */
  char formula[MAX_FORMULA_LEN];
  char **vnames = 0, **plab = 0;

  /* Void pointers */
  void *hform = 0, *hddesc = 0, *hxdesc = 0;

  /* Initialize the error structure */
  INIT_FAIL(fail);

  printf("nag_blgm_lm_describe_data (g22ybc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
```

```
  /* Read in size of the data matrix and number of variable labels supplied */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &nobs, &nvar,
          &lvnames);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &nobs, &nvar,
        &lvnames);
#endif

  /* Allocate memory */
  if (!(levels = NAG_ALLOC(nvar, Integer)) ||
      !(vnames = NAG_ALLOC(lvnames, char *))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < lvnames; i++)
    if (!(vnames[i] = NAG_ALLOC(MAX_VNAME_LEN, char))) {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read in number of levels and names for the variables */
  for (i = 0; i < nvar; i++) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &levels[i]);
#else
    scanf("%" NAG_IFMT "", &levels[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  if (lvnames > 0) {
    for (i = 0; i < lvnames; i++)
#ifdef _WIN32
      scanf_s("%50s", vnames[i], 51);
#else
      scanf("%50s", vnames[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif
  }

  /* Call nag_blgm_lm_describe_data (g22ybc) to get a description of */
  /* the data matrix */
  nag_blgm_lm_describe_data(&hddesc,nobs,nvar,levels,lvnames,vnames,&fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_blgm_lm_describe_data (g22ybc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Read in the data matrix and response variable */
  pddat = nobs;
  sddat = nvar;
  if (!(dat = NAG_ALLOC(pddat*sddat, double)) ||
      !(y = NAG_ALLOC(nobs, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
```

```
  for (i = 0; i < nobs; i++) {
    for (j = 0; j < nvar; j++)
#ifdef _WIN32
      scanf_s("%lf", &DAT(i, j));
#else
    scanf("%lf", &DAT(i, j));
#endif
#ifdef _WIN32
    scanf_s("%lf", &y[i]);
#else
    scanf("%lf", &y[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read in the formula for the full model, remove comments and */
  /* call nag_blgm_lm_formula (g22yac) to parse it */
  read_line(formula,MAX_FORMULA_LEN);
  nag_blgm_lm_formula(&hform,formula,&fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_blgm_lm_formula (g22yac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Start of constructing the design matrix ... */

  /* Call nag_blgm_optset (g22zmc) to alter the storage order of X as */
  /* nag_regsn_mult_linear uses VAROBS storage  */
  nag_blgm_optset(hform,"Storage Order = VAROBS",&fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_blgm_optset (g22zmc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Call nag_blgm_lm_design_matrix (g22ycc) to get the size of */
  /* the design matrix */
  ldx = 0;
  sdx = 0;
  nag_blgm_lm_design_matrix(hform,hddesc,dat,pddat,sddat,&hxdesc,
                            x,ldx,sdx,&mx,&fail);
  if (fail.code != NW_ARRAY_SIZE && fail.code != NW_ALTERNATIVE) {
    printf("Error from nag_blgm_lm_design_matrix (g22ycc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Allocate design matrix */
  ldx = mx;
  sdx = nobs;
  if (!(x = NAG_ALLOC(ldx*sdx, double))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Call nag_blgm_lm_design_matrix (g22ycc) to generate the design matrix */
  nag_blgm_lm_design_matrix(hform,hddesc,dat,pddat,sddat,&hxdesc,
                            x,ldx,sdx,&mx,&fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_blgm_lm_design_matrix (g22ycc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
```

```
  /* ... End of constructing the design matrix */

  /* Start of getting the isx vector and information on parameter labels ... */
  /* Get size of output arrays used by nag_blgm_lm_submodel (g22ydc) */
  lvinfo = 3;
  lisx = lplab = lenlab = 0;
  nag_blgm_lm_submodel(hform,hxdesc,&intcpt,&ip,lisx,isx,lplab,plab,lenlab,
                       lvinfo,tvinfo, &fail);
  if (fail.code != NW_ARRAY_SIZE) {
    printf("Error from nag_blgm_lm_design_matrix (g22ydc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Allocate output arrays (we already know that lisx = mx, but */
  /* nag_blgm_lm_submodel returns it just in case) */
  lisx = tvinfo[0];
  lvinfo = tvinfo[2];
  /* We don't need plab as we are constructing our own labels from vinfo */
  lplab = 0;
  if (!(isx = NAG_ALLOC(lisx, Integer)) ||
      !(vinfo = NAG_ALLOC(lvinfo, Integer))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Call nag_blgm_lm_submodel (g22ydc) to get the isx flag */
  /* and parameter labels */
  nag_blgm_lm_submodel(hform,hxdesc,&intcpt,&ip,lisx,isx,lplab,plab,lenlab,
                       lvinfo,vinfo,&fail);

  /* Construct some verbose labels for the parameters */
  if ((exit_status = construct_labels(ip,&plab,vnames,vinfo)))
    goto END;
  /* ... End of getting the isx vector and information on parameter labels */

  /* Fit a regression model and print the results */
  exit_status = fit_lm(hform,intcpt,nobs,mx,x,ldx,isx,ip,y,plab);

 END:
  /* Call nag_blgm_handle_free (g22zac) to clean-up the g22 handles */
  nag_blgm_handle_free(&hform,&fail);
  nag_blgm_handle_free(&hddesc,&fail);
  nag_blgm_handle_free(&hxdesc,&fail);

  NAG_FREE(dat);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(levels);
  for (i = 0; i < lvnames; i++)
    NAG_FREE(vnames[i]);
  NAG_FREE(vnames);
  for (i = 0; i < ip; i++)
    NAG_FREE(plab[i]);
  NAG_FREE(plab);
  NAG_FREE(isx);
  NAG_FREE(vinfo);
  return (exit_status);
}

char *read_line(char formula[],Integer nchar) {
  /* Read in a line from stdin and remove any comments */
  char *pch;

  /* Read in the model formula */
  if (fgets(formula,nchar,stdin)) {
    /* Strip comments from formula */
    pch = strstr(formula,"::");
    if (pch) *pch = '\setminus 0';
```

```
    return formula;
  } else {
    return 0;
  }
}

Integer construct_labels(Integer ip,char **plab[],char *const vnames[],
                         Integer vinfo[]) {
  /* Construct labels from information held in VINFO */
  /* NB: For simplicity, the function contains no checks for buffer */
  /* overflow when manipulating character strings */

  /* Integer scalar and array declarations */
  Integer exit_status = 0;
  Integer b, i, j, k, li, vi;

  /* Character scalar and array declarations */
  char term[MAX_PLAB_LEN];
  char *this_lab = 0;

  if (!((*plab) = NAG_ALLOC(ip, char *))) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < ip; i++)
    if (!((*plab)[i] = NAG_ALLOC(MAX_VNAME_LEN, char))) {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  k = 0;
  for (j = 0; j < ip; j++) {
    b = vinfo[k];
    k++;
    this_lab = (*plab)[j];
    if (b == 0) {
      sprintf((*plab)[j],"%s","Intercept");
    } else {
      for (i = 0; i < b; i++) {
        vi = vinfo[k] - 1;
        li = vinfo[k+1];

        if (li != 0)
          sprintf(term,"%s (Level %" NAG_IFMT ")%c",vnames[vi],li,'\setminus 0');
        else
          strcpy(term,vnames[vi]);

        if (i == 0) {
          strcpy(this_lab,term);
          this_lab += strlen(this_lab);
        } else {
          this_lab += sprintf(this_lab," . %s",term);
        }

        /* We are ignoring the contrast identifier when */
        /* constructing these labels */
        k += 3;
      }
    }
  }

 END:

  return (exit_status);
}

Integer fit_lm(void *hform,Nag_IncludeIntercept intcpt,Integer nobs,Integer mx,
           double x[],Integer ldx,Integer isx[],Integer ip,double y[],
           char *plab[]) {
```

```
  /* Perform a multiple linear regression using */
  /* nag_regsn_mult_linear (g02dac) */

  /* Integer scalar and array declarations */
  Integer i, rank, ldq, exit_status = 0, lcvalue, ivalue;

  /* NAG types */
  Nag_Boolean svd;
  NagError fail;
  Nag_IncludeMean mean;
  Nag_VariableType optype;

  /* Double scalar and array declarations */
  double rss, tol, df, rvalue;
  double *b = 0, *cov = 0, *h = 0, *p = 0, *q = 0, *res = 0, *se = 0,
    *wt = 0, *com_ar = 0;

  /* Character scalar and array declarations */
  char cvalue[MAX_CVALUE_LEN];

  /* Initialize the error structure */
  INIT_FAIL(fail);

  /* We are assuming un-weighted data */
  ldq = (ip + 1);
  if (!(b = NAG_ALLOC(ip, double)) ||
      !(se = NAG_ALLOC(ip, double)) ||
      !(cov = NAG_ALLOC((ip * ip + ip) / 2, double)) ||
      !(res = NAG_ALLOC(nobs, double)) ||
      !(h = NAG_ALLOC(nobs, double)) ||
      !(q = NAG_ALLOC(nobs * ldq, double)) ||
      !(p = NAG_ALLOC(ip * (ip + 2), double)) ||
      !(com_ar = NAG_ALLOC(ip * ip + 5 * (ip - 1), double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Use suggested value for tolerance */
  tol = 0.000001;

  mean = (intcpt == Nag_Intercept) ? Nag_MeanInclude : Nag_MeanZero;

  /* Call nag_regsn_mult_linear (g02dac) to fit a regression model */
  nag_regsn_mult_linear(mean, nobs, x, ldx, mx, isx, ip, y,
                        wt, &rss, &df, b, se, cov, res, h, q,
                        ldq, &svd, &rank, p, tol, com_ar, &fail);

  /* Call nag_blgm_optget (g22znc) to get the formula for the model */
  /* being fit */
  lcvalue = MAX_CVALUE_LEN;
  nag_blgm_optget(hform,"Formula",&ivalue,&rvalue,cvalue,lcvalue,&optype,
                  &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_blgm_optget (g22znc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Display the results */
  printf(" Model: %s\n", cvalue);
  printf("                                  Parameter   Standard\n");
  printf(" Coefficients                     Estimate      Error\n");
  printf(" ------------------------------------------------------\n");
  for (i = 0; i < ip; i++)
    printf(" %-30s %7.3f     %7.3f\n", plab[i], b[i], se[i]);
  printf(" ------------------------------------------------------\n");
  printf(" Residual sum of squares = %9.4f\n", rss);
  printf(" Degrees of freedom      = %9.0f\n", df);
```

```
 END:
  NAG_FREE(h);
  NAG_FREE(res);
  NAG_FREE(wt);
  NAG_FREE(b);
  NAG_FREE(cov);
  NAG_FREE(p);
  NAG_FREE(q);
  NAG_FREE(com_ar);
  NAG_FREE(se);
  return exit_status;
}
```

10.2 Program Data

```
nag_blgm_lm_describe_data (g22ybc) Example Program Data
25 3 3                                  :: nobs,nvar,lvnames
3 3 1                                   :: levels
F1 F2 Con                               :: vnames
3 1 -2.4   1.16
3 3  0.2   4.96
1 3 -1.4  -1.67
2 1 -5.4 -11.80
3 3  0.2   6.03
3 2  1.4  11.70
1 2  6.8  33.34
1 2  6.7  31.97
1 1  5.3  23.93
2 3 -1.3   3.17
3 2 -3.6   1.68
3 2 -0.7   8.01
1 1  5.7  26.14
3 3  2.3  11.04
1 2  3.3  20.32
2 3 -0.5   5.62
1 1 -2.6  -6.21
1 2  3.7  22.45
1 2  0.9  10.93
3 1 -1.1   1.59
2 2  2.1  13.55
1 3  4.6  24.16
2 3  4.6  20.70
1 2  5.1  28.30
1 3  0.9   9.69                         :: dat, y
F1 + F2 + Con + F1.F2 + F1.Con + F2.Con :: formula
```

10.3 Program Results

```
nag_blgm_lm_describe_data (g22ybc) Example Program Results

 Model: F1+F2+CON+F1.F2+F1.CON+F2.CON
                              Parameter   Standard
 Coefficients                 Estimate      Error
 ------------------------------------------------
 Intercept                      3.902       0.618
 F1 (Level 2)                  -2.197       2.060
 F1 (Level 3)                   1.005       1.071
 F2 (Level 2)                   4.094       1.008
 F2 (Level 3)                   0.958       0.836
 Con (Level 1)                  3.828       0.130
 F1 (Level 2) . F2 (Level 2)    2.666       2.706
 F1 (Level 2) . F2 (Level 3)    4.399       2.398
 F1 (Level 3) . F2 (Level 2)    0.004       1.519
 F1 (Level 3) . F2 (Level 3)   -0.756       1.385
 F1 (Level 2) . Con (Level 1)  -1.327       0.270
 F1 (Level 3) . Con (Level 1)  -1.810       0.252
```

```
 F2 (Level 2) . Con (Level 1)    -0.079       0.207
 F2 (Level 3) . Con (Level 1)     0.465       0.230
 -------------------------------------------------
 Residual sum of squares =    8.2462
 Degrees of freedom      =        11
```

11 Optional Parameters

As well as the optional parameters common to all G22 handles described in **nag_blgm_optset (g22zmc)** and **nag_blgm_optget (g22znc)**, a number of additional optional parameters can be specified for a G22 handle holding the description of a data matrix as returned by **nag_blgm_lm_describe_data (g22ybc)** in **hddesc**.

Each writeable optional parameter has an associated default value; to set any of them to a non-default value, use **nag_blgm_optset (g22zmc)**. The value of an optional parameter can be queried using **nag_blgm_optget (g22znc)**.

The remainder of this section can be skipped if you wish to use the default values for all optional parameters.

The following is a list of the optional parameters available. A full description of each optional parameter is provided in Section 11.1.

Number of Observations

Number of Variables

Storage Order

11.1 Description of the Optional Parameters

For each option, we give a summary line, a description of the optional parameter and details of constraints.

The summary line contains:

a parameter value, where the letters a, i and r denote options that take character, integer and real values respectively;

the default value.

Keywords and character values are case and white space insensitive.

Number of Observations i

If queried, this optional parameter will return n, the number of observations in the data matrix.

Number of Variables i

If queried, this optional parameter will return m_d, the number of variables in the data matrix.

Storage Order a Default $=$ OBSVAR

This optional parameter states how the data matrix, D, will be stored in its input array.

If **Storage Order** $=$ OBSVAR, D_{ij}, the value for the jth variable of the ith observation of the data matrix is stored in **dat**$[(j-1) \times$ **pddat** $+ i - 1]$.

If **Storage Order** $=$ VAROBS, D_{ij}, the value for the jth variable of the ith observation of the data matrix is stored in **dat**$[(i-1) \times$ **pddat** $+ j - 1]$.

Where **dat** is the input parameter of the same name in **nag_blgm_lm_design_matrix (g22ycc)**.

Constraint: **Storage Order** $=$ OBSVAR or VAROBS.