NAG Library Function Document

nag_dummy_vars (g04eac)

1 Purpose

nag_dummy_vars (g04eac) computes orthogonal polynomial or dummy variables for a factor or classification variable.

2 Specification

```
#include <nag.h>
#include <nagg04.h>
void nag_dummy_vars (Nag_DummyType type, Integer n, Integer levels,
     const Integer factor[], double x[], Integer tdx, const double v[],
     double num_reps[], NagError *fail)
```

3 Description

In the analysis of an experimental design using a general linear model the factors or classification variables that specify the design have to be coded as dummy variables. nag_dummy_vars (g04eac) computes dummy variables that can then be used in the fitting of the general linear model using nag_regsn_mult_linear (g02dac).

If the factor of length n has k levels then the simplest representation is to define k dummy variables, X_j such that $X_j = 1$ if the factor is at level j and 0 otherwise, for $j = 1, 2, \ldots, k$. However, there is usually a mean included in the model and the sum of the dummy variables will be aliased with the mean. To avoid the extra redundant argument, $k - 1$ dummy variables can be defined as the contrasts between one level of the factor, the reference level and the remaining levels. If the reference level is the first level then the dummy variables can be defined as $X_j = 1$ if the factor is at level j and 0 otherwise, for $j = 2, 3, \ldots, k$. Alternatively, the last level can be used as the reference level.

A second way of defining the $k - 1$ dummy variables is to use a Helmert matrix in which levels $2, 3, \ldots, k$ are compared with the average effect of the previous levels. For example if $k = 4$ then the contrasts would be:

$$\begin{array}{rrrr} 1 & -1 & -1 & -1 \\ 2 & 1 & -1 & -1 \\ 3 & 0 & 2 & -1 \\ 4 & 0 & 0 & 3 \end{array}$$

Thus variable j, for $j = 1, 2, \ldots, k - 1$, is given by

$X_j = -1$ if factor is at level less than $j + 1$

$X_j = \sum_{i=1}^{j} r_i / r_{j+1}$ if factor is at level $j + 1$

$X_j = 0$ if factor is at level greater than $j + 1$

where r_j is the number of replicates of level j.

If the factor can be considered as a set of values from an underlying continuous variable then the factor can be represented by a set of $k - 1$ orthogonal polynomials representing the linear, quadratic, etc. effects of the underlying variable. The orthogonal polynomial is computed using Forsythe's algorithm (see Forsythe (1957) and Cooper (1968)). The values of the underlying continuous variable represented by the factor levels have to be supplied to the function.

The orthogonal polynomials are standardized so that the sum of squares for each dummy variable is one. For the other methods integer (± 1) representations are retained except that in the Helmert representation the code of level $j + 1$ in dummy variable j will be a fraction.

4 References

Cooper B E (1968) Algorithm AS 10. The use of orthogonal polynomials *Appl. Statist.* **17** 283–287

Forsythe G E (1957) Generation and use of orthogonal polynomials for data fitting with a digital computer *J. Soc. Indust. Appl. Math.* **5** 74–88

5 Arguments

1: **type** – Nag_DummyType *Input*

On entry: the type of dummy variable to be computed.

type = Nag_Poly
An orthogonal Polynomial representation is computed.

type = Nag_Helmert
A Helmert matrix representation is computed.

type = Nag_FirstLevel
The contrasts relative to the First level are computed.

type = Nag_LastLevel
The contrasts relative to the Last level are computed.

type = Nag_AllLevels
A complete set of dummy variables is computed.

Constraint: **type** = Nag_Poly, Nag_Helmert, Nag_FirstLevel, Nag_LastLevel or Nag_AllLevels.

2: **n** – Integer *Input*

On entry: the number of observations for which the dummy variables are to be computed, n.

Constraint: **n** $\geq$ **levels**.

3: **levels** – Integer *Input*

On entry: the number of levels of the factor, k.

Constraint: **levels** ≥ 2.

4: **factor**[**n**] – const Integer *Input*

On entry: the n values of the factor.

Constraint: $1 \leq$ **factor**$[i-1] \leq$ **levels**, for $i = 1, 2, \ldots, n$.

5: **x**[**n** $\times$ **tdx**] – double *Output*

Note: the (i, j)th element of the matrix X is stored in **x**$[(i-1) \times$ **tdx** $+ j - 1]$.

On exit: the n by k^* matrix of dummy variables, where $k^* = k - 1$ if **type** = Nag_Poly, Nag_Helmert, Nag_FirstLevel or Nag_LastLevel and $k^* = k$ if **type** = Nag_AllLevels.

6: **tdx** – Integer *Input*

On entry: the stride separating matrix column elements in the array **x**.

Constraints:

if **type** = Nag_Poly, Nag_Helmert, Nag_FirstLevel or Nag_LastLevel, **tdx** $\geq$ **levels** $- 1$;
if **type** = Nag_AllLevels, **tdx** $\geq$ **levels**.

7: **v**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **v** must be at least

levels when **type** = Nag_Poly;
1 otherwise.

On entry: if **type** = Nag_Poly, the k distinct values of the underlying variable for which the orthogonal polynomial is to be computed. If **type** $\neq$ Nag_Poly, **v** is not referenced.

Constraint: if **type** = Nag_Poly, then the k values of **v** must be distinct.

8: **num_reps**[**levels**] – double *Output*

On exit: **num_reps**[$i-1$] contains the number of replications for each level of the factor, r_i, for $i = 1, 2, \ldots, k$.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_2_INT_ARG_LT

On entry, **n** = $\langle value \rangle$ while **levels** = $\langle value \rangle$. These arguments must satisfy **n** $\geq$ **levels**.

On entry, **tdx** = $\langle value \rangle$ while **levels** = $\langle value \rangle$. These arguments must satisfy **tdx** $\geq$ **levels**.

On entry, **tdx** = $\langle value \rangle$ while **levels** $- 1 = \langle value \rangle$. These arguments must satisfy **tdx** $\geq$ **levels** $- 1$.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_ARRAY_CONS

The contents of array **v** are not valid.
Constraint: all values of **v** must be distinct.

NE_BAD_PARAM

On entry, argument **type** had an illegal value.

NE_G04EA_LEVELS

All **levels** are not represented in array factor.

NE_G04EA_ORTHO_POLY

An orthogonal polynomial has all values zero. This will be due to some values of **v** being close together. This can only occur if **type** = Nag_Poly.

NE_INT_ARG_LT

On entry, **levels** must not be less than 2: **levels** = $\langle value \rangle$.

NE_INT_ARRAY_CONS

On entry, **factor**[0] = $\langle value \rangle$.
Constraint: $1 \leq$ **factor**[0] $\leq$ **levels**.

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.

7 Accuracy

The computations are stable.

8 Parallelism and Performance

nag_dummy_vars (g04eac) is not threaded in any implementation.

9 Further Comments

Other functions for fitting polynomials can be found in Chapter e02.

10 Example

Data are read in from an experiment with four treatments and three observations per treatment with the treatment coded as a factor. nag_dummy_vars (g04eac) is used to compute the required dummy variables and the model is then fitted by nag_regsn_mult_linear (g02dac).

10.1 Program Text

```
/* nag_dummy_vars (g04eac) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>
#include <nagg04.h>

int main(void)
{
  Nag_Boolean svd;
  Integer exit_status = 0, i, *ifact = 0, ip, irank, *isx = 0, j,
         levels, m, n;
  Integer tdq, tdx;
  char nag_enum_arg[40];
  double *b = 0, *com_ar = 0, *cov = 0, df, *h = 0, *p = 0, *q = 0;
  double *rep = 0, *res = 0, tol, *v = 0, *wt = 0, *wtptr = 0, *x = 0;
  double *y = 0, rss, *se = 0;
  Nag_DummyType dum_type;
  Nag_IncludeMean mean;
  Nag_Boolean weight;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_dummy_vars (g04eac) Example Program Results\n");

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

```
#ifdef _WIN32
  scanf_s("%" NAG_IFMT " %" NAG_IFMT "", &n, &levels);
#else
  scanf("%" NAG_IFMT " %" NAG_IFMT "", &n, &levels);
#endif
#ifdef _WIN32
  scanf_s(" %39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf(" %39s", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  dum_type = (Nag_DummyType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s(" %39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf(" %39s", nag_enum_arg);
#endif
  weight = (Nag_Boolean) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s(" %39s", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf(" %39s", nag_enum_arg);
#endif
  mean = (Nag_IncludeMean) nag_enum_name_to_value(nag_enum_arg);

  if (dum_type == Nag_AllLevels)
    tdx = levels;
  else
    tdx = levels - 1;

  if (!(x = NAG_ALLOC(n * tdx, double))
      || !(rep = NAG_ALLOC(levels, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  if (dum_type == Nag_Poly) {
    if (!(v = NAG_ALLOC(levels, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    if (!(v = NAG_ALLOC(1, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  }
  if (!(wt = NAG_ALLOC(n, double))
      || !(y = NAG_ALLOC(n, double))
      || !(ifact = NAG_ALLOC(n, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  printf("\n");
  if (weight) {
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
      scanf_s("%" NAG_IFMT " %lf %lf", &ifact[i - 1], &y[i - 1], &wt[i - 1]);
#else
      scanf("%" NAG_IFMT " %lf %lf", &ifact[i - 1], &y[i - 1], &wt[i - 1]);
```

```
#endif
    wtptr = wt;
  }
  else {
    for (i = 1; i <= n; ++i)
#ifdef _WIN32
      scanf_s("%" NAG_IFMT " %lf", &ifact[i - 1], &y[i - 1]);
#else
      scanf("%" NAG_IFMT " %lf", &ifact[i - 1], &y[i - 1]);
#endif
    wtptr = 0;
  }
  if (dum_type == Nag_Poly)
    for (j = 1; j <= levels; ++j)
#ifdef _WIN32
      scanf_s("%lf", &v[j - 1]);
#else
      scanf("%lf", &v[j - 1]);
#endif

  /* Calculate dummy variables */
  /* nag_dummy_vars (g04eac).
   * Computes orthogonal polynomials or dummy variables for
   * factor/classification variable
   */
  nag_dummy_vars(dum_type, n, levels, ifact, x, tdx, v, rep, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dummy_vars (g04eac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  m = tdx;
  ip = m;
  if (mean == Nag_MeanInclude)
    ++ip;
  if (!(b = NAG_ALLOC(ip, double))
      || !(se = NAG_ALLOC(ip, double))
      || !(cov = NAG_ALLOC(ip * (ip + 1) / 2, double))
      || !(p = NAG_ALLOC(2 * ip + ip * ip, double))
      || !(com_ar = NAG_ALLOC(5 * (ip - 1) + ip * ip, double))
      || !(h = NAG_ALLOC(n, double))
      || !(res = NAG_ALLOC(n, double))
      || !(q = NAG_ALLOC(n * (ip + 1), double))
      || !(tdq = ip + 1)
      || !(isx = NAG_ALLOC(m, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  for (j = 1; j <= m; ++j)
    isx[j - 1] = 1;

  /* Set tolerance */
  tol = 1e-5;
  /* nag_regsn_mult_linear (g02dac).
   * Fits a general (multiple) linear regression model
   */
  nag_regsn_mult_linear(mean, n, x, tdx, m, isx, ip, y, wtptr, &rss, &df,
                        b, se, cov, res, h, q, tdq, &svd, &irank, p, tol,
                        com_ar, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_regsn_mult_linear (g04dac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  if (svd)
    printf(" %s%4" NAG_IFMT "\n\n", "Model not of full rank, rank = ", irank);
```

```
  printf(" Residual sum of squares =  %13.4e\n", rss);
  printf(" Degrees of freedom = %4.0f\n\n", df);
  printf("Variable       Parameter estimate    Standard error\n\n");
  for (j = 1; j <= ip; ++j)
    printf(" %6" NAG_IFMT " %20.4e %20.4e\n", j, b[j - 1], se[j - 1]);

END:
  NAG_FREE(x);
  NAG_FREE(rep);
  NAG_FREE(v);
  NAG_FREE(v);
  NAG_FREE(wt);
  NAG_FREE(y);
  NAG_FREE(ifact);
  NAG_FREE(b);
  NAG_FREE(se);
  NAG_FREE(cov);
  NAG_FREE(p);
  NAG_FREE(com_ar);
  NAG_FREE(h);
  NAG_FREE(res);
  NAG_FREE(q);
  NAG_FREE(isx);

  return exit_status;
}
```

10.2 Program Data

```
nag_dummy_vars (g04eac) Example Program Data
 12 4 Nag_AllLevels Nag_FALSE Nag_MeanInclude
1 33.63
4 39.62
2 38.18
3 41.46
4 38.02
2 35.83
4 35.99
1 36.58
3 42.92
1 37.80
3 40.43
2 37.89
```

10.3 Program Results

```
nag_dummy_vars (g04eac) Example Program Results

 Model not of full rank, rank =    4

 Residual sum of squares =     2.2227e+01
 Degrees of freedom =    8

Variable       Parameter estimate    Standard error

      1           3.0557e+01           3.8494e-01
      2           5.4467e+00           8.3896e-01
      3           6.7433e+00           8.3896e-01
      4           1.1047e+01           8.3896e-01
      5           7.3200e+00           8.3896e-01
```