NAG C 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 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, $j = 1,2,...,k$. 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 parameter, $k - 1$ 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, $j = 2, 3, \ldots, k$. 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, $j = 1, 2, \ldots, k - 1$ 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 routine.

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 Parameters

1: **type** – Nag_DummyType *Input*

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

If **type** = **Nag_Poly**, an orthogonal Polynomial representation is computed.

If **type** = **Nag_Helmert**, a Helmert matrix representation is computed.

If **type** = **Nag_FirstLevel**, the contrasts relative to the First level are computed.

If **type** = **Nag_LastLevel**, the contrasts relative to the Last level are computed.

If **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**, $i = 1, 2, \ldots, n$.

5: **x[n][tdx]** – double *Output*

Note: the second dimension of the array **x** must be at least **levels**-1 if **type** = **Nag_Poly**, **Nag_Helmert**, **Nag_FirstLevel** or **Nag_LastLevel** and **levels** if **type** = **Nag_AllLevels**.

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 second dimension of the array **x** as declared in the function from which nag_dummy_vars is called.

Constraints:

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

tdx $\geq$ **levels** if **type** = **Nag_AllLevels**.

7: **v[***dim1***]** – const double *Input*

Note: the dimension, *dim1*, of the array **v** must be at least **levels** if **type** = **Nag_Poly** and 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, $i = 1, 2, \ldots, k$.

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

The NAG error parameter (see the Essential Introduction).

5 Error Indicators and Warnings

NE_INT_ARG_LT

On entry, **levels** must not be less than 2: **levels** = <*value*>.

NE_2_INT_ARG_LT

On entry, **n** = <*value*> while **levels** = <*value*>.
These parameters must satisfy **n** $\geq$ **levels**.

On entry, **tdx** = <*value*> while **levels** = <*value*>.
These parameters must satisfy **tdx** $\geq$ **levels**.

On entry, **tdx** = <*value*> while **levels**-1 = <*value*>.
These parameters must satisfy **tdx** $\geq$ **levels**-1.

NE_BAD_PARAM

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

NE_ALLOC_FAIL

Memory allocation failed.

NE_ARRAY_CONS

The contents of array **v** are not valid.

Constraint: all values of **v** must be distinct.

NE_INT_ARRAY_CONS

On entry, **factor**[0] = <*value*>.

Constraint: $1 \leq$ **factor**[0] $\leq$ **levels**.

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_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 consult NAG for assistance.

6 Further Comments

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

6.1 Accuracy

The computations are stable.

6.2 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

7 See Also

nag_regsn_mult_linear (g02dac)
Chapter e02

8 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 is used to compute the required dummy variables and the model is then fitted by nag_regsn_mult_linear (g02dac).

8.1 Program Text

```
/* nag_dummy_vars (g04eac) Example Program.
 *
 * Copyright 2000 Numerical Algorithms Group.
 *
 * Mark 6, 2000.
 */


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

int main (void)
{
  char mean[2], type[2], weight[2];
  double *b=0, *cov=0, df, *h=0, *p=0, *q=0, *rep, *res, rss, *se=0, tol;
  double *v=0, *com_ar=0, *wtptr=0, *wt=0, *x=0, *y=0;
  Integer i, *ifact=0, ip, irank, *isx=0, j, levels, m, n, tdq, tdx;
  Integer exit_status=0;
  Boolean svd;
  Nag_DummyType dum_type;
  NagError fail;
  Nag_IncludeMean mean_enum;

  INIT_FAIL(fail);
  Vprintf("g04eac Example Program Results\n");

  /*  Skip heading in data file */
  Vscanf("%*[^\n]");

  Vscanf("%ld %ld %s %s %s", &n, &levels, type, weight, mean);
  if (*type == 'P')
    dum_type = Nag_Poly;
  else if (*type == 'H')
    dum_type = Nag_Helmert;
  else if (*type == 'F')
    dum_type = Nag_FirstLevel;
```

```
  else if (*type == 'L')
    dum_type = Nag_LastLevel;
  else if (*type == 'C')
    dum_type = Nag_AllLevels;
  else
    dum_type = (Nag_DummyType)-999;


  if (*mean == 'M')
    mean_enum = Nag_MeanInclude;
  else if (*mean == 'Z')
    mean_enum = Nag_MeanZero;
  else
    mean_enum = (Nag_IncludeMean)-999;

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

  if (!(x = NAG_ALLOC(n*tdx, double))
      || !(rep = NAG_ALLOC(levels, double)))
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  if (dum_type == Nag_Poly)
    {
      if (!(v=NAG_ALLOC(levels, double)))
{
  Vprintf("Allocation failure\n");
  exit_status = -1;
  goto END;
}
    }
  else
    {
      if (!(v=NAG_ALLOC(1, double)))
{
  Vprintf("Allocation failure\n");
  exit_status = -1;
  goto END;
}
    }
  if (!(wt = NAG_ALLOC(n, double))
      || !(y = NAG_ALLOC(n, double))
      || !(ifact = NAG_ALLOC(n, Integer)))
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  Vprintf("\n");
  if (*weight == 'W')
    {
      for (i = 1; i <= n; ++i)
Vscanf("%ld %lf %lf", &ifact[i - 1], &y[i - 1], &wt[i - 1]);
      wtptr=wt;
```

```
    }
  else
    {
      for (i = 1; i <= n; ++i)
 Vscanf("%ld %lf",  &ifact[i - 1], &y[i - 1]);
      wtptr = 0;
    }
  if (dum_type == Nag_Poly)
    for (j = 1; j <= levels; ++j)
      Vscanf("%lf", &v[j - 1]);

  /*  Calculate dummy variables */
  g04eac(dum_type, n, levels, ifact, x, tdx, v, rep, &fail);
  if (fail.code != NE_NOERROR)
    {
      Vprintf("Error from g04eac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  m = tdx;
  ip = m;
  if (mean_enum == 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)))
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

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

  /*  Set tolerance */
  tol = 1e-5;
  g02dac(mean_enum, 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)
    {
      Vprintf("Error from g04dac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;

    }

  if (svd)
    Vprintf(" %s%4ld\n\n", "Model not of full rank, rank = ", irank);

  Vprintf(" %s %12.4e\n", "Residual sum of squares = ", rss);
```

```
  Vprintf(" %s%4.0f\n\n","Degrees of freedom = ", df);
  Vprintf("%s\n\n", "Variable       Parameter estimate    Standard error");
  for (j = 1; j <= ip; ++j)
    Vprintf(" %6ld %20.4e %20.4e\n", j, b[j - 1], se[j - 1]);
 END:
  if (x) NAG_FREE(x);
  if (rep) NAG_FREE(rep);
  if (v) NAG_FREE(v);
  if (v) NAG_FREE(v);
  if (wt) NAG_FREE(wt);
  if (y) NAG_FREE(y);
  if (ifact) NAG_FREE(ifact);
  if (b) NAG_FREE(b);
  if (se) NAG_FREE(se);
  if (cov) NAG_FREE(cov);
  if (p) NAG_FREE(p);
  if (com_ar) NAG_FREE(com_ar);
  if (h) NAG_FREE(h);
  if (res) NAG_FREE(res);
  if (q) NAG_FREE(q);
  if (isx) NAG_FREE(isx);
  return exit_status;
}
```

8.2 Program Data

```
g04eac Example Program Data
 12 4 C U M
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
```

8.3 Program Results

```
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
```