NAG Library Function Document

nag_best_subset_given_size (h05abc)

1 Purpose

Given a set of m features and a scoring mechanism for any subset of those features, nag_best_subset_given_size (h05abc) selects the best n subsets of size p using a direct communication branch and bound algorithm.

2 Specification

```
#include <nag.h>
#include <nagh.h>
void nag_best_subset_given_size (Integer mincr, Integer m, Integer ip,
     Integer nbest, Integer *la, double bscore[], Integer bz[],
     void (*f)(Integer m, Integer drop, Integer lz, const Integer z[],
          Integer la, const Integer a[], double score[], Nag_Comm *comm,
          Integer *info),
     Integer mincnt, double gamma, const double acc[], Nag_Comm *comm,
     NagError *fail)
```

3 Description

Given $\Omega = \{x_i : i \in \mathbb{Z}, 1 \leq i \leq m\}$, a set of m unique features and a scoring mechanism $f(S)$ defined for all $S \subseteq \Omega$ then nag_best_subset_given_size (h05abc) is designed to find $S_{o1} \subseteq \Omega, |S_{o1}| = p$, an optimal subset of size p. Here $|S_{o1}|$ denotes the cardinality of S_{o1}, the number of elements in the set.

The definition of the optimal subset depends on the properties of the scoring mechanism, if

$$f(S_i) \leq f(S_j), \quad \text{for all } S_j \subseteq \Omega \text{ and } S_i \subseteq S_j \tag{1}$$

then the optimal subset is defined as one of the solutions to

$$\underset{S \subseteq \Omega}{\text{maximize}} f(S) \quad \text{subject to} \quad |S| = p$$

else if

$$f(S_i) \geq f(S_j), \quad \text{for all } S_j \subseteq \Omega \text{ and } S_i \subseteq S_j \tag{2}$$

then the optimal subset is defined as one of the solutions to

$$\underset{S \subseteq \Omega}{\text{minimize}} f(S) \quad \text{subject to} \quad |S| = p.$$

If neither of these properties hold then nag_best_subset_given_size (h05abc) cannot be used.

As well as returning the optimal subset, S_{o1}, nag_best_subset_given_size (h05abc) can return the best n solutions of size p. If S_{oi} denotes the ith best subset, for $i = 1, 2, \ldots, n-1$, then the $(i+1)$th best subset is defined as the solution to either

$$\underset{S \subseteq \Omega - \left\{S_{oj} : j \in \mathbb{Z}, 1 \leq j \leq i\right\}}{\text{maximize}} f(S) \quad \text{subject to} \quad |S| = p$$

or

$$\underset{S \subseteq \Omega - \left\{S_{oj} : j \in \mathbb{Z}, 1 \leq j \leq i\right\}}{\text{minimize}} f(S) \quad \text{subject to} \quad |S| = p$$

depending on the properties of f.

The solutions are found using a branch and bound method, where each node of the tree is a subset of Ω. Assuming that (1) holds then a particular node, defined by subset S_i, can be trimmed from the tree if $f(S_i) < \hat{f}(S_{on})$ where $\hat{f}(S_{on})$ is the nth highest score we have observed so far for a subset of size p, i. e., our current best guess of the score for the nth best subset. In addition, because of (1) we can also drop all nodes defined by any subset S_j where $S_j \subseteq S_i$, thus avoiding the need to enumerate the whole tree. Similar short cuts can be taken if (2) holds. A full description of this branch and bound algorithm can be found in Ridout (1988).

Rather than calculate the score at a given node of the tree nag_best_subset_given_size (h05abc) utilizes the fast branch and bound algorithm of Somol *et al.* (2004), and attempts to estimate the score where possible. For each feature, x_i, two values are stored, a count c_i and $\hat{\mu}_i$, an estimate of the contribution of that feature. An initial value of zero is used for both c_i and $\hat{\mu}_i$. At any stage of the algorithm where both $f(S)$ and $f(S - \{x_i\})$ have been calculated (as opposed to estimated), the estimated contribution of the feature x_i is updated to

$$\frac{c_i\hat{\mu}_i + \left[f(S) - f\left(S - \left\{x_j\right\}\right)\right]}{c_i + 1}$$

and c_i is incremented by 1, therefore at each stage $\hat{\mu}_i$ is the mean contribution of x_i observed so far and c_i is the number of observations used to calculate that mean.

As long as $c_i \geq k$, for the user-supplied constant k, then rather than calculating $f(S - \{x_\mathrm{i}\})$ this function estimates it using $\hat{f}(S - \{x_\mathrm{i}\}) = f(S) - \gamma\hat{\mu}_i$ or $\hat{f}(S) - \gamma\hat{\mu}_i$ if $f(S)$ has been estimated, where γ is a user-supplied scaling factor. An estimated score is never used to trim a node or returned as the optimal score.

Setting $k = 0$ in this function will cause the algorithm to always calculate the scores, returning to the branch and bound algorithm of Ridout (1988). In most cases it is preferable to use the fast branch and bound algorithm, by setting $k > 0$, unless the score function is iterative in nature, i.e., $f(S)$ must have been calculated before $f(S - \{x_i\})$ can be calculated.

nag_best_subset_given_size (h05abc) is a direct communication version of nag_best_subset_given_si ze_revcomm (h05aac).

4 References

Narendra P M and Fukunaga K (1977) A branch and bound algorithm for feature subset selection *IEEE Transactions on Computers* **9** 917–922

Ridout M S (1988) Algorithm AS 233: An improved branch and bound algorithm for feature subset selection *Journal of the Royal Statistics Society, Series C (Applied Statistics) (Volume 37)* **1** 139–147

Somol P, Pudil P and Kittler J (2004) Fast branch and bound algorithms for optimal feature selection *IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume 26)* **7** 900–912

5 Arguments

1: **mincr** – Integer *Input*

On entry: flag indicating whether the scoring function f is increasing or decreasing.

mincr $= 1$
$f(S_i) \leq f\left(S_j\right)$, i.e., the subsets with the largest score will be selected.

mincr $= 0$
$f(S_i) \geq f\left(S_j\right)$, i.e., the subsets with the smallest score will be selected.

For all $S_j \subseteq \Omega$ and $S_i \subseteq S_j$.

Constraint: **mincr** $= 0$ or 1.

2: **m** – Integer *Input*

On entry: m, the number of features in the full feature set.

Constraint: $\mathbf{m} \geq 2$.

3: **ip** – Integer *Input*

On entry: p, the number of features in the subset of interest.

Constraint: $1 \leq \mathbf{ip} \leq \mathbf{m}$.

4: **nbest** – Integer *Input*

On entry: n, the maximum number of best subsets required. The actual number of subsets returned is given by **la** on final exit. If on final exit $\mathbf{la} \neq \mathbf{nbest}$ then **fail.code** = NE_TOO_MANY is returned.

Constraint: $\mathbf{nbest} \geq 1$.

5: **la** – Integer * *Output*

On exit: the number of best subsets returned.

6: **bscore**[**nbest**] – double *Output*

On exit: holds the score for the **la** best subsets returned in **bz**.

7: **bz**[$(\mathbf{m} - \mathbf{ip}) \times \mathbf{nbest}$] – Integer *Output*

Note: where $\mathbf{BZ}(i,j)$ appears in this document, it refers to the array element $\mathbf{bz}[(j-1) \times (\mathbf{m} - \mathbf{ip}) + i - 1]$.

On exit: the jth best subset is constructed by dropping the features specified in $\mathbf{BZ}(i,j)$, for $i = 1, 2, \ldots, \mathbf{m} - \mathbf{ip}$ and $j = 1, 2, \ldots, \mathbf{la}$, from the set of all features, Ω. The score for the jth best subset is given in $\mathbf{bscore}[j-1]$.

8: **f** – function, supplied by the user *External Function*

f must evaluate the scoring function f.

The specification of **f** is:

```
void f (Integer m, Integer drop, Integer lz, const Integer z[],
     Integer la, const Integer a[], double score[], Nag_Comm *comm,
     Integer *info)
```

1: **m** – Integer *Input*

On entry: $m = |\Omega|$, the number of features in the full feature set.

2: **drop** – Integer *Input*

On entry: flag indicating whether the intermediate subsets should be constructed by dropping features from the full set ($\mathbf{drop} = 1$) or adding features to the empty set ($\mathbf{drop} = 0$). See **score** for additional details.

3: **lz** – Integer *Input*

On entry: the number of features stored in **z**.

4: **z**[**lz**] – const Integer *Input*

On entry: $\mathbf{z}[i-1]$, for $i = 1, 2, \ldots, \mathbf{lz}$, contains the list of features which, along with those specified in **a**, define the subsets whose score is required. See **score** for additional details.

5: **la** – Integer *Input*

On entry: if **la** > 0, the number of subsets for which a score must be returned.

If **la** $= 0$, the score for a single subset should be returned. See **score** for additional details.

6: **a**[**la**] – const Integer *Input*

On entry: **a**$[j-1]$, for $j = 1, 2, \ldots,$ **la**, contains the list of features which, along with those specified in **z**, define the subsets whose score is required. See **score** for additional details.

7: **score**[**max**(**la**, **1**)] – double *Output*

On exit: the value $f\left(S_j\right)$, for $j = 1, 2, \ldots,$ **la**, the score associated with the jth subset. S_j is constructed as follows:

drop $= 1$
: S_j is constructed by *dropping* the features specified in the first **lz** elements of **z** and the single feature given in **a**$[j-1]$ from the full set of features, Ω. The subset will therefore contain **m** $-$ **lz** $- 1$ features.

drop $= 0$
: S_j is constructed by *adding* the features specified in the first **lz** elements of **z** and the single feature specified in **a**$[j-1]$ to the empty set, $\emptyset$. The subset will therefore contain **lz** $+ 1$ features.

In both cases the individual features are referenced by the integers 1 to **m** with 1 indicating the first feature, 2 the second, etc., for some arbitrary ordering of the features, chosen by you prior to calling nag_best_subset_given_size (h05abc). For example, 1 might refer to the first variable in a particular set of data, 2 the second, etc..

If **la** $= 0$, the score for a single subset should be returned. This subset is constructed by adding or removing only those features specified in the first **lz** elements of **z**. If **lz** $= 0$, this subset will either be Ω or $\emptyset$.

8: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **f**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_best_subset_given_size (h05abc) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag_best_subset_given_size (h05abc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

9: **info** – Integer * *Input/Output*

On entry: **info** $= 0$.

On exit: set **info** to a nonzero value if you wish nag_best_subset_given_size (h05abc) to terminate with **fail.code** $=$ NE_USER_STOP.

9: **mincnt** – Integer *Input*

On entry: k, the minimum number of times the effect of each feature, x_i, must have been observed before $f\left(S - \{x_i\}\right)$ is estimated from $f\left(S\right)$ as opposed to being calculated directly.

If $k = 0$ then $f\left(S - \{x_i\}\right)$ is never estimated. If **mincnt** < 0 then k is set to 1.

10: **gamma** – double *Input*

On entry: γ, the scaling factor used when estimating scores. If **gamma** < 0 then $\gamma = 1$ is used.

11: **acc**[**2**] – const double *Input*

On entry: a measure of the accuracy of the scoring function, f.

Letting $a_i = \epsilon_1 |f(S_i)| + \epsilon_2$, then when confirming whether the scoring function is strictly increasing or decreasing (as described in **mincr**), or when assessing whether a node defined by subset S_i can be trimmed, then any values in the range $f(S_i) \pm a_i$ are treated as being numerically equivalent.

If $0 \leq$ **acc**[0] ≤ 1 then $\epsilon_1 =$ **acc**[0], otherwise $\epsilon_1 = 0$.

If **acc**[1] ≥ 0 then $\epsilon_2 =$ **acc**[1], otherwise $\epsilon_2 = 0$.

In most situations setting both ϵ_1 and ϵ_2 to zero should be sufficient. Using a nonzero value, when one is not required, can significantly increase the number of subsets that need to be evaluated.

12: **comm** – Nag_Comm *

The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

13: **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_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_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 2.

On entry, **mincr** = ⟨*value*⟩.
Constraint: **mincr** = 0 or 1.

On entry, **nbest** = ⟨*value*⟩.
Constraint: **nbest** ≥ 1.

NE_INT_2

On entry, **ip** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: $1 \leq$ **ip** $\leq$ **m**.

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

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_REAL

On exit from **f**, **score**[$\langle value \rangle$] = $\langle value \rangle$, which is inconsistent with the score for the parent node. Score for the parent node is $\langle value \rangle$.

NE_TOO_MANY

On entry, **nbest** = $\langle value \rangle$.
But only $\langle value \rangle$ best subsets could be calculated.

NE_USER_STOP

A nonzero value for **info** has been returned: **info** = $\langle value \rangle$.

7 Accuracy

The subsets returned by nag_best_subset_given_size (h05abc) are guaranteed to be optimal up to the accuracy of the calculated scores.

8 Parallelism and Performance

nag_best_subset_given_size (h05abc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The maximum number of unique subsets of size p from a set of m features is $N = \frac{m!}{(m-p)!p!}$. The efficiency of the branch and bound algorithm implemented in nag_best_subset_given_size (h05abc) comes from evaluating subsets at internal nodes of the tree, that is subsets with more than p features, and where possible trimming branches of the tree based on the scores at these internal nodes as described in Narendra and Fukunaga (1977). Because of this it is possible, in some circumstances, for more than N subsets to be evaluated. This will tend to happen when most of the features have a similar effect on the subset score.

If multiple optimal subsets exist with the same score, and **nbest** is too small to return them all, then the choice of which of these optimal subsets is returned is arbitrary.

10 Example

This example finds the three linear regression models, with five variables, that have the smallest residual sums of squares when fitted to a supplied dataset. The data used in this example was simulated.

10.1 Program Text

```
/* nag_best_subset_given_size (h05abc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* Pre-processor includes */
```

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>
#include <nagh05.h>

#define BZ(I,J) bz[(J)*(m - ip) + (I)]

typedef struct
{
  Integer n, tdq, tdx;
  double tol;
  double *x, *y, *b, *cov, *h, *p, *q, *res, *se, *wt, *com_ar;
  Integer *sx, cnt;
  Nag_IncludeMean mean;
} calc_subset_data;

void NAG_CALL read_subset_data(Integer m, calc_subset_data * cs);
void NAG_CALL tidy_subset_data(calc_subset_data * cs);
void NAG_CALL calc_subset_score(Integer m, Integer drop, Integer lz,
                                const Integer z[], Integer la,
                                const Integer a[], double score[],
                                Nag_Comm *comm, Integer *info);
int main(void)
{
  int exit_status = 0;

  /* Integer scalar and array declarations */
  Integer i, ip, j, la, m, mincnt, mincr, mip, nbest;
  Integer *a = 0, *bz = 0, *ibz = 0, *id = 0;

  /* NAG structures */
  NagError fail;
  Nag_Comm comm;

  /* Other data types */
  calc_subset_data cs;

  /* Double scalar and array declarations */
  double gamma;
  double *bscore = 0;
  double acc[2];

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

  printf("nag_best_subset_given_size (h05abc) Example Program Results\n\n");

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

  /* Read in the problem size */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &m, &ip, &nbest);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &m, &ip, &nbest);
#endif

  /* Read in the control parameters for the subset selection */
#ifdef _WIN32
```

```
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%lf%lf%lf%*[^\n] ", &mincr, &mincnt,
          &gamma, &acc[0], &acc[1]);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%lf%lf%lf%*[^\n] ", &mincr, &mincnt,
        &gamma, &acc[0], &acc[1]);
#endif

  /* Read in the data required for the score function */
  read_subset_data(m, &cs);

  /* Associate the data structure with comm.p */
  comm.p = (void *) &cs;

  /* Allocate memory required by the subset selection routine */
  mip = m - ip;
  if (!(a = NAG_ALLOC(MAX(nbest, m), Integer)) ||
      !(bz = NAG_ALLOC(mip * nbest, Integer)) ||
      !(bscore = NAG_ALLOC(MAX(nbest, m), double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Call the forward communication best subset routine
     (nag_best_subset_given_size (h05abc)) */
  nag_best_subset_given_size(mincr, m, ip, nbest, &la, bscore, bz,
                             calc_subset_score, mincnt, gamma, acc, &comm,
                             &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_best_subset_given_size (h05abc).\n%s\n",
           fail.message);
    if (fail.code != NE_ARG_TOO_LARGE) {
      exit_status = 1;
      goto END;
    }
  }

  /* Titles */
  printf("    Score        Feature Subset\n");
  printf("    -----        --------------\n");

  /* Display the best subsets and corresponding scores.
     nag_best_subset_given_size returns a list
     of features excluded from the best subsets, so this is inverted to give
     the set of features included in each subset */
  if (!(id = NAG_ALLOC(m, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  for (i = 0; i < la; i++) {
    printf("%12.5e ", bscore[i]);
    for (j = 0; j < m; j++)
      id[j] = 1;
    for (j = 0; j < mip; j++)
      id[BZ(j, i) - 1] = 0;
    for (j = 0; j < m; j++)
      if (id[j])
        printf(" %5" NAG_IFMT, j + 1);
    printf("\n");
  }
  printf("\n");
  if (fail.code == NE_TOO_MANY) {
    printf("%" NAG_IFMT " subsets of the requested size do not exist, "
           "only %" NAG_IFMT " are displayed.\n", nbest, la);
  }
  printf("%" NAG_IFMT " subsets evaluated in total\n", cs.cnt);

END:
```

```
  NAG_FREE(a);
  NAG_FREE(bz);
  NAG_FREE(ibz);
  NAG_FREE(bscore);
  NAG_FREE(id);

  tidy_subset_data(&cs);

  return exit_status;
}

#define CSX(I, J) cs->x[(I) * cs->tdx + J]
void NAG_CALL read_subset_data(Integer m, calc_subset_data * cs)
{
  /* Read in the data, from stdout, and allocate any arrays that are required
     by the scoring function calc_subset_score */
  Integer i, j;
  cs->sx = 0;
  cs->x = cs->y = cs->b = cs->se = cs->cov = cs->res = 0;
  cs->h = cs->q = cs->p = cs->com_ar = cs->wt = 0;

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

  /* Read in the number of observations for the data used in the linear
     regression */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &cs->n);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &cs->n);
#endif

  /* Read in the control parameters for the linear regression */
#ifdef _WIN32
  scanf_s("%lf%*[^\n] ", &cs->tol);
#else
  scanf("%lf%*[^\n] ", &cs->tol);
#endif

  /* Read in the data */
  cs->tdx = m;

  if (!(cs->x = NAG_ALLOC(cs->tdx * cs->n, double)) ||
      !(cs->y = NAG_ALLOC(cs->n, double)))
  {
    printf("Allocation failure\n");
    tidy_subset_data(cs);
    exit(-1);
  }

  /* Read in the data for the linear regression */
  for (i = 0; i < cs->n; i++) {
    for (j = 0; j < m; j++)
#ifdef _WIN32
      scanf_s("%lf", &CSX(i, j));
#else
      scanf("%lf", &CSX(i, j));
#endif
#ifdef _WIN32
    scanf_s("%lf", &cs->y[i]);
#else
    scanf("%lf", &cs->y[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
```

```
#endif
  }

  /* No intercept term and no weights */
  cs->mean = Nag_MeanZero;

  /* Allocate memory required by the regression routine */
  cs->tdq = cs->n;
  if (!(cs->sx = NAG_ALLOC(m, Integer)) ||
      !(cs->b = NAG_ALLOC(m, double)) ||
      !(cs->se = NAG_ALLOC(m, double)) ||
      !(cs->cov = NAG_ALLOC(m * (m + 1) / 2, double)) ||
      !(cs->res = NAG_ALLOC(cs->n, double)) ||
      !(cs->h = NAG_ALLOC(cs->n, double)) ||
      !(cs->q = NAG_ALLOC(cs->n * cs->tdq, double)) ||
      !(cs->p = NAG_ALLOC(2 * m + m * m, double)) ||
      !(cs->com_ar = NAG_ALLOC(5 * (m - 1) * m * m, double)))
  {
    printf("Allocation failure\n");
    tidy_subset_data(cs);
    exit(-1);
  }

  cs->cnt = 0;
}

void NAG_CALL tidy_subset_data(calc_subset_data * cs)
{
  /* Tidy up the data structure used by calc_subset_score */
  NAG_FREE(cs->sx);
  NAG_FREE(cs->x);
  NAG_FREE(cs->y);
  NAG_FREE(cs->b);
  NAG_FREE(cs->se);
  NAG_FREE(cs->cov);
  NAG_FREE(cs->res);
  NAG_FREE(cs->h);
  NAG_FREE(cs->q);
  NAG_FREE(cs->p);
  NAG_FREE(cs->com_ar);
  NAG_FREE(cs->wt);
}

void NAG_CALL calc_subset_score(Integer m, Integer drop, Integer lz,
                                const Integer z[], Integer la,
                                const Integer a[], double score[],
                                Nag_Comm *comm, Integer *info)
{
  /* Calculate the score associated with a particular set of feature subsets.

     This particular example finds the set, of a given size, of explanatory
     variables that best fit a response variable when a linear regression model
     is used. Therefore the feature set is the set of all the explanatory
     variables and the best set of features is defined as set of explanatory
     variables that gives the smallest residual sums of squares. See the
     documentation for g02dac for details on linear regression models.
   */
  calc_subset_data *cs;

  /* Integer scalar and array declarations */
  Integer i, j, inv_drop, ip, irank;

  /* NAG structures */
  NagError fail;

  /* Double scalar and array declarations */
  double rss, df;

  /* Other data types */
  Nag_Boolean svd;
```

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

  /* Point cs to the data structure stored in comm */
  cs = (calc_subset_data *) comm->p;

  /* Keep track of the number of susbets evaluated */
  cs->cnt += MAX(1, la);

  /* Set up the initial feature set.
     If drop = 0, this is the Null set (i.e. no features).
     If drop = 1 then this is the full set (i.e. all features) */
  for (i = 0; i < m; i++)
    cs->sx[i] = drop;

  /* Add (if drop = 0) or remove (if drop = 1) the all the features specified
     in Z (note: z refers to its features with values 1 to M, therefore you
     must subtract 1 when using z as an array reference) */
  inv_drop = (drop == 0) ? 1 : 0;
  for (i = 0; i < lz; i++)
    cs->sx[z[i] - 1] = inv_drop;

  /* Loop over all the elements in a, looping at least once */
  for (i = 0; i < MAX(la, 1); i++) {
    if (la > 0) {
      if (i > 0) {
        /* Reset the feature altered at the last iteration */
        cs->sx[a[i - 1] - 1] = drop;
      }

      /* Add or drop the I'th feature in A */
      cs->sx[a[i] - 1] = inv_drop;
    }

    /* Calculate number of parameters in the regression model */
    for (ip = j = 0; j < m; j++)
      ip += (cs->sx[j] == 1);

    /* Fit the regression model (g02dac) */
    nag_regsn_mult_linear(cs->mean, cs->n, cs->x, cs->tdx, m, cs->sx, ip,
                          cs->y, cs->wt, &rss, &df, cs->b, cs->se, cs->cov,
                          cs->res, cs->h, cs->q, cs->tdq, &svd, &irank, cs->p,
                          cs->tol, cs->com_ar, &fail);

    /* Return the score (the residual sums of squares) */
    score[i] = rss;
  }

  /* If g02dac fails, terminate early */
  if (fail.code != NE_NOERROR)
    *info = 1;
}
```

10.2 Program Data

```
nag_best_subset_given_size (h05abc) Example Program Data
Data required by h05abc
14 5 3                      :: m,ip,nbest
0 -1 -1.0 -1.0 -1.0         :: mincr, mincnt, gamma, acc[0], acc[1]

Data required by the scoring function
40                          :: n
1e-6                        :: tol
 -1.59   0.19   0.40   0.43  -0.40   0.79   0.06
  0.33   1.60   0.58  -1.12   1.23   1.07  -0.07          -2.44
 -0.25   0.61  -0.36   1.16   0.61  -2.05  -0.02
 -0.04   0.80  -0.73  -0.63  -0.75  -0.73   1.43          -2.97
 -2.28   0.46  -0.65   0.33   0.16  -0.21  -1.61
 -0.54   0.48   0.37  -0.95  -2.14   0.48   2.02           7.42
 -0.52   1.05   0.64   0.02  -1.12   0.23   0.06
```

```
-1.26   1.40  -0.98   2.47   0.49  -0.02  -0.05           3.00
-0.84   1.86   0.10   0.73  -1.41   0.98   0.20
-0.89   1.84   2.56   0.60  -0.12   0.71   0.23           8.83
 1.12  -0.51  -0.58   0.09  -1.14   2.11  -0.11
-0.34  -1.04  -0.43  -0.01  -0.38   1.80   0.05           0.03
 0.06   0.85  -2.09   0.22  -1.35  -0.36   1.20
 0.41   0.80  -0.28   0.18   0.27   0.92   0.63           2.57
-0.48  -1.02   0.08  -0.06   0.13  -1.18   2.30
 0.03   0.45   0.62  -1.97   0.97   0.93  -0.18           8.31
 0.08  -0.31   0.43  -0.38   0.01   1.30   0.66
 0.65  -0.59   0.76   0.04   0.17  -0.76  -0.90           4.55
 0.66   1.14   0.40   2.37   1.10   0.17  -0.38
 1.15  -1.00  -0.13  -0.69  -0.62  -0.18   0.00         -23.10
-1.08  -0.21  -1.13  -0.79  -0.76  -1.58   0.38
-0.03   1.26  -0.51  -0.75   0.86   0.29   0.68           3.38
-0.74  -1.59  -0.58  -1.09   1.18  -1.70  -1.02
 0.36   1.05   1.30  -0.98  -1.36  -1.28  -1.32          -0.13
 0.40  -1.58  -1.30  -0.10  -1.34   0.65  -0.56
 0.39  -0.73  -0.32   2.19  -0.49   0.69   0.18           5.47
 0.75  -3.09  -0.61  -1.89   0.15   0.77  -0.49
-0.63   1.20  -0.04   1.02   0.31   0.81  -0.45          13.97
-0.65   1.57  -1.50  -1.45   0.21   0.06   0.24
 2.24  -1.34   0.30   1.39  -0.38  -0.71   0.48          20.94
 1.36   1.40  -1.40  -0.90   0.36  -0.21  -0.97
 0.36  -0.26   0.08   0.06  -1.49   0.43  -1.61         -12.87
-1.01   1.50  -0.61  -0.25  -1.01  -0.43   1.90
-1.33  -0.96  -0.02   0.51  -1.38  -0.78   1.82          28.26
 1.34   1.02   3.50   0.10   0.50   0.04   0.61
-0.57  -2.69  -0.64  -0.34  -0.21  -1.97  -0.19           6.89
 0.29   0.67  -0.38  -0.63  -0.24   1.21  -0.09
 0.90  -2.20   1.72   0.29   0.66   0.19  -0.57           5.37
 0.67  -0.56  -0.41   1.22  -0.30   0.77   0.82
 0.36   1.18   1.87  -1.48   0.52   1.35   0.13          -1.50
-0.40  -1.10  -0.83   0.71   1.99  -0.24   1.30
-0.34  -0.70   0.28   0.16   0.27   0.37  -1.79         -23.00
-0.78   0.60  -0.45  -0.26  -0.23   0.89   0.87
 1.01   1.20   0.28   0.79   2.76   0.35   1.31          14.09
-1.29   0.62  -0.59   1.52   0.62   0.21   1.31
 1.09  -0.36  -0.34  -0.03  -0.59  -1.70  -0.03         -11.05
 0.40  -1.45  -0.98   2.10  -1.09  -0.53  -0.38
-1.36   0.13   0.70  -1.51   0.08  -0.62  -0.64         -32.04
 0.43  -0.86   0.70  -1.07  -0.76   0.72  -0.14
-1.58   0.00   0.58  -0.21   1.30   2.02   1.52          23.36
-0.48   0.01   1.30   0.58  -0.54   1.09   0.91
 2.90   1.32  -1.20  -0.59  -0.51   0.20  -1.74          -5.58
-1.32  -1.41  -0.58  -1.29   1.61  -0.35  -0.72
-1.92  -1.09   0.56  -0.87  -0.71   1.25   0.10           2.48
 1.43   0.69   1.34  -0.32   2.84  -1.43  -0.47
-0.01   0.83  -0.72  -0.78   0.50  -1.22   0.54          -5.30
 0.82   0.46   0.15  -0.57   0.93   1.33  -0.23
-1.07   0.76   0.25  -1.96   0.39   0.24  -0.26          -7.77
-0.91   0.23  -0.19   1.58  -0.27   0.33  -0.60
-1.39  -0.30  -0.81  -0.95   0.88  -0.09  -0.35         -34.25
 0.65  -1.14   1.18  -1.06  -0.68  -0.22   0.21
 0.94   1.08   0.81  -0.33   0.42  -0.90   0.49          26.78
-0.36  -0.50  -0.02  -0.04   0.77   0.62  -1.35
-0.64   1.20   1.22   0.18  -1.39  -0.81  -0.99         -11.85
-1.82   1.06   0.28   0.14   0.62  -0.80  -1.08
-2.15   1.37   1.57  -1.48  -0.79   0.28  -0.20          -8.62
 1.54   0.50   0.13  -0.68   0.26  -1.13   0.62
-0.43   0.39   1.14   0.15   1.03   0.46   0.40          12.35
-1.61  -0.61   0.93  -0.37   0.44  -1.45   0.58
-1.77   0.72  -2.05  -0.03  -1.24  -1.40  -0.06          -1.54
-0.48   0.67   0.04   0.27  -0.84  -0.06  -3.67
 0.09   1.66  -0.30   1.67   1.08   0.00   0.43         -16.59
-1.65  -1.16  -1.17   1.12   0.11  -0.15   0.48
-1.72   1.08  -0.94   0.49  -0.56   0.95   1.09          -8.69
-0.85  -0.02   1.18  -1.16   0.49   1.56  -0.60
```

```
  0.32   0.72  -1.20   2.52   1.78   0.16  -0.01            7.82
 -0.60  -0.73  -1.23   1.50   0.40  -0.20  -0.65
  0.68   1.09   0.40  -1.50  -2.10   0.21  -0.18          -18.56
 -0.66  -0.01  -0.01   0.85  -2.04   1.17  -0.56
  1.72  -0.18   1.14  -0.96  -0.92  -0.28   1.58           17.21 :: x,y
```

10.3 Program Results

```
nag_best_subset_given_size (h05abc) Example Program Results

    Score        Feature Subset
    -----        --------------
 1.04753e+03      4     7     8    10    14
 1.05987e+03      4     5     7     8    14
 1.07016e+03      4     5     7    10    14

45 subsets evaluated in total
```