NAG Library Function Document

nag_rand_subsamp_xyw (g05pwc)

1 Purpose

nag_rand_subsamp_xyw (g05pwc) generates a dataset suitable for use with repeated random sub-sampling validation.

2 Specification

```
#include <nag.h>
#include <nagg05.h>
```

```
void nag_rand_subsamp_xyw (Integer nt, Integer n, Integer m,
     Nag_DataByObsOrVar sordx, double x[], Integer pdx, double y[],
     double w[], Integer state[], NagError *fail)
```

3 Description

Let X_o denote a matrix of n observations on m variables and y_o and w_o each denote a vector of length n. For example, X_o might represent a matrix of independent variables, y_o the dependent variable and w_o the associated weights in a weighted regression.

nag_rand_subsamp_xyw (g05pwc) generates a series of training datasets, denoted by the matrix, vector, vector triplet (X_t, y_t, w_t) of n_t observations, and validation datasets, denoted (X_v, y_v, w_v) with n_v observations. These training and validation datasets are generated by randomly assigning each observation to either the training dataset or the validation dataset.

The resulting datasets are suitable for use with repeated random sub-sampling validation.

One of the initialization functions nag_rand_init_repeatable (g05kfc) (for a repeatable sequence if computed sequentially) or nag_rand_init_nonrepeatable (g05kgc) (for a non-repeatable sequence) must be called prior to the first call to nag_rand_subsamp_xyw (g05pwc).

4 References

None.

5 Arguments

1: **nt** – Integer *Input*

On entry: n_t, the number of observations in the training dataset.

Constraint: $1 \leq \mathbf{nt} \leq \mathbf{n}$.

2: **n** – Integer *Input*

On entry: n, the number of observations.

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

3: **m** – Integer *Input*

On entry: m, the number of variables.

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

4: **sordx** – Nag_DataByObsOrVar *Input*

On entry: determines how variables are stored in **x**.

Constraint: **sordx** = Nag_DataByVar or Nag_DataByObs.

5: **x**[*dim*] – double *Input/Output*

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

pdx × **m** when **sordx** = Nag_DataByVar;
pdx × **n** when **sordx** = Nag_DataByObs.

The way the data is stored in **x** is defined by **sordx**.

If **sordx** = Nag_DataByVar, **x**$[(j-1)\times$ **pdx** $+ i - 1]$ contains the ith observation for the jth variable, for $i = 1, 2, \dots,$ **n** and $j = 1, 2, \dots,$ **m**.

If **sordx** = Nag_DataByObs, **x**$[(i-1)\times$ **pdx** $+ j - 1]$ contains the ith observation for the jth variable, for $i = 1, 2, \dots,$ **n** and $j = 1, 2, \dots,$ **m**.

On entry: **x** must hold X_o, the values of X for the original dataset. This may be the same **x** as returned by a previous call to nag_rand_subsamp_xyw (g05pwc).

On exit: values of X for the training and validation datasets, with X_t held in observations 1 to **nt** and X_v in observations **nt** $+ 1$ to **n**.

6: **pdx** – Integer *Input*

On entry: the stride separating row elements in the two-dimensional data stored in the array **x**.

Constraints:

if **sordx** = Nag_DataByObs, **pdx** $\geq$ **m**;
otherwise **pdx** $\geq$ **n**.

7: **y**[**n**] – double *Input/Output*

If the original dataset does not include y_o then **y** must be set to **NULL**.

On entry: **y** must hold y_o, the values of y for the original dataset. This may be the same **y** as returned by a previous call to nag_rand_subsamp_xyw (g05pwc).

On exit: values of y for the training and validation datasets, with y_t held in elements 1 to **nt** and y_v in elements **nt** $+ 1$ to **n**.

8: **w**[**n**] – double *Input/Output*

If the original dataset does not include w_o then **w** must be set to **NULL**.

On entry: **w** must hold w_o, the values of w for the original dataset. This may be the same **w** as returned by a previous call to nag_rand_subsamp_xyw (g05pwc).

On exit: values of w for the training and validation datasets, with w_t held in elements 1 to **nt** and w_v in elements **nt** $+ 1$ to **n**.

9: **state**[*dim*] – Integer *Communication Array*

Note: the dimension, *dim*, of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument **state** in the previous call to nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc).

On entry: contains information on the selected base generator and its current state.

On exit: contains updated information on the state of the generator.

10: **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_ARRAY_SIZE

On entry, **pdx** $= \langle value \rangle$ and **m** $= \langle value \rangle$.
Constraint: if **sordx** = Nag_DataByObs, **pdx** $\geq$ **m**.

On entry, **pdx** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: if **sordx** = Nag_DataByVar, **pdx** $\geq$ **n**.

NE_BAD_PARAM

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

NE_INT

On entry, **m** $= \langle value \rangle$.
Constraint: **m** ≥ 1.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 1.

NE_INT_2

On entry, **nt** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: $1 \leq$ **nt** $\leq$ **n**.

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_INVALID_STATE

On entry, **state** vector has been corrupted or not initialized.

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.

7 Accuracy

Not applicable.

8 Further Comments

nag_rand_subsamp_xyw (g05pwc) will be computationality more efficient if each observation in **x** is contiguous, that is **sordx** = Nag_DataByObs.

9 Example

This example uses nag_rand_subsamp_xyw (g05pwc) to facilitate repeated random sub-sampling cross-validation.

A set of simulated data is randomly split into a training and validation datasets. nag_glm_binomial (g02gbc) is used to fit a logistic regression model to each training dataset and then nag_glm_predict (g02gpc) is used to predict the response for the observations in the validation dataset. This process is repeated 10 times.

The counts of true and false positives and negatives along with the sensitivity and specificity is then reported.

9.1 Program Text

```
/* nag_rand_subsamp_xyw (g05pwc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>
#include <nagg05.h>

int main(void)
{
  /* Integer scalar and array declarations */
  Integer fn, fp, i, ip, pdx, lstate, m,
         n, nn, np, nt, nv, obs_val, pred_val, subid,
         tn, tp, j, pdv, rank, max_iter, print_iter, nsamp, samp;
  Integer exit_status = 0, lseed = 1;
  Integer *isx = 0, *state = 0;
  Integer seed[1];

  /* NAG structures and types */
  NagError fail;
  Nag_Link link;
  Nag_IncludeMean mean;
  Nag_BaseRNG genid;
  Nag_Distributions errfn;
  Nag_Boolean vfobs;
  Nag_DataByObsOrVar sordx;

  /* Double scalar and array declarations */
  double ex_power, dev, eps, tol, df, scale;
  double *b = 0, *cov = 0, *eta = 0, *pred = 0, *se = 0, *seeta = 0,
         *sepred = 0, *v = 0, *offset = 0, *wt = 0, *x = 0, *y = 0, *t = 0;

  /* Character scalar and array declarations */
  char clink[40], cmean[40], cgenid[40];

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

  printf("nag_rand_subsamp_xyw (g05pwc) Example Program Results\n\n");

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

```
  /* Set variables required by the regression (g02gbc) ... */

  /* Read in the type of link function, whether a mean is required */
  /* and the problem size */
#ifdef _WIN32
  scanf_s("%39s%39s%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", clink,
          (unsigned)_countof(clink), cmean, (unsigned)_countof(cmean),
          &n, &m);
#else
  scanf("%39s%39s%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", clink, cmean, &n, &m);
#endif
  link = (Nag_Link) nag_enum_name_to_value(clink);
  mean = (Nag_IncludeMean) nag_enum_name_to_value(cmean);

  /* Set storage order for g05pwc */
  /* (pick the one required by g02gbc and g02gpc) */
  sordx = Nag_DataByObs;

  pdx = m;
  if (!(x = NAG_ALLOC(pdx * n, double)) ||
      !(y = NAG_ALLOC(n, double)) ||
      !(t = NAG_ALLOC(n, double)) || !(isx = NAG_ALLOC(m, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* This example is not using an offset or weights */
  offset = 0;
  wt = 0;

  /* Read in data */
  for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
#ifdef _WIN32
      scanf_s("%lf", &x[i * pdx + j]);
#else
      scanf("%lf", &x[i * pdx + j]);
#endif
    }
#ifdef _WIN32
    scanf_s("%lf%lf%*[^\n] ", &y[i], &t[i]);
#else
    scanf("%lf%lf%*[^\n] ", &y[i], &t[i]);
#endif
  }

  /* Read in variable inclusion flags */
  for (j = 0; j < m; j++) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &isx[j]);
#else
    scanf("%" NAG_IFMT "", &isx[j]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read in control parameters for the regression */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%lf%lf%" NAG_IFMT "%*[^\n] ", &print_iter, &eps,
          &tol, &max_iter);
#else
  scanf("%" NAG_IFMT "%lf%lf%" NAG_IFMT "%*[^\n] ", &print_iter, &eps,
        &tol, &max_iter);
#endif
```

```
  /* Calculate IP */
  for (ip = 0, i = 0; i < m; i++)
    ip += (isx[i] > 0);
  if (mean == Nag_MeanInclude)
    ip++;
  /* ... End of setting variables required by the regression */

  /* Set variables required by data sampling routine (g05pwc) ... */

  /* Read in the base generator information and seed */
#ifdef _WIN32
  scanf_s("%39s%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", cgenid,
          (unsigned)_countof(cgenid), &subid, &seed[0]);
#else
  scanf("%39s%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", cgenid, &subid, &seed[0]);
#endif
  genid = (Nag_BaseRNG) nag_enum_name_to_value(cgenid);

  /* Initial call to g05kfc to get size of STATE array */
  lstate = 0;
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate,
                           NAGERR_DEFAULT);

  /* Allocate state array */
  if (!(state = NAG_ALLOC(lstate, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Initialize the generator to a repeatable sequence using g05kfc */
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate,
                           NAGERR_DEFAULT);

  /* Read in the size of the training set required */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &nt);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &nt);
#endif

  /* Read in the number of sub-samples we will use */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &nsamp);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &nsamp);
#endif
  /* ... End of setting variables required by data sampling routine */

  /* Set variables required by prediction routine (g02gpc) ... */

  /* Regression is performed using g02gbc so error structure is binomial */
  errfn = Nag_Binomial;

  /* This example does not use the predicted standard errors, so */
  /* it doesn't matter what VFOBS is set to */
  vfobs = Nag_FALSE;
  /* The error and link being used in the linear model don't use scale */
  /* and ex_power so they can be set to anything */
  ex_power = 0.0;
  scale = 0.0;
  /* ... End of setting variables required by prediction routine */

  /* Calculate the size of the validation dataset */
  nv = n - nt;

  /* Allocate arrays */
  pdv = n;
  if (!(b = NAG_ALLOC(ip, double)) ||
      !(se = NAG_ALLOC(ip, double)) ||
```

```
      !(cov = NAG_ALLOC(ip * (ip + 1) / 2, double)) ||
      !(v = NAG_ALLOC(n * pdv, double)) ||
      !(eta = NAG_ALLOC(nv, double)) ||
      !(seeta = NAG_ALLOC(nv, double)) ||
      !(pred = NAG_ALLOC(nv, double)) || !(sepred = NAG_ALLOC(nv, double)))

  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Initialize counts */
  tp = tn = fp = fn = 0;

  /* Loop over each sample */
  for (samp = 1; samp <= nsamp; samp++)
  {
    /* Use g05pwc to split the data into training and validation datasets */
    nag_rand_subsamp_xyw(nt, n, m, sordx, x, pdx, y, t, state, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_rand_subsamp_xyw (g05pwc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* Call g02gbc to fit generalized linear model, with Binomial */
    /* errors to training data */
    nag_glm_binomial(link, mean, nt, x, pdx, m, isx, ip, y, t, wt,
                     offset, &dev, &df, b, &rank, se, cov, v, pdv,
                     tol, max_iter, print_iter, "", eps, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_glm_binomial (g02gbc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;

    }

    /* Call g02gpc to predict the response for the observations in the */
    /* validation dataset */
    /* We want to start passing X and T at the (NT+1)th observation, */
    /* These start at (i,j)=(nt+1,1), hence the (nt*pdx+0)th element */
    /* of X and the nt'th element of T */
    nag_glm_predict(errfn, link, mean, nv, &x[nt * pdx], pdx, m, isx, ip,
                    &t[nt], offset, wt, scale, ex_power, b, cov, vfobs, eta,
                    seeta, pred, sepred, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_glm_predict (g02gpc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

    /* Count the true/false positives/negatives */
    for (i = 0; i < nv; i++) {
      obs_val = (Integer) y[nt + i];
      pred_val = (pred[i] >= 0.5 ? 1 : 0);
      if (obs_val) {
        /* Positive */
        if (pred_val) {
          /* True positive */
          tp++;
        }
        else {
          /* False Negative */
          fn++;
        }
      }
      else {
        /* Negative */
        if (pred_val) {
          /* False positive */
```

```
          fp++;
        }
        else {
          /* True negative */
          tn++;
        }
      }
    }
  }

  /* Display results */
  np = tp + fn;
  nn = fp + tn;
  printf("                       Observed\n");
  printf("             --------------------------\n");
  printf(" Predicted | Negative  Positive   Total\n");
  printf(" --------------------------------------\n");
  printf(" Negative  | %5" NAG_IFMT "     %5" NAG_IFMT "     %5" NAG_IFMT
         "\n", tn, fn, tn + fn);
  printf(" Positive  | %5" NAG_IFMT "     %5" NAG_IFMT "     %5" NAG_IFMT
         "\n", fp, tp, fp + tp);
  printf(" Total     | %5" NAG_IFMT "     %5" NAG_IFMT "     %5" NAG_IFMT
         "\n", nn, np, nn + np);
  printf("\n");

  if (np != 0) {
    printf(" True Positive Rate (Sensitivity): %4.2f\n",
           (double) tp / (double) np);
  }
  else {
    printf(" True Positive Rate (Sensitivity): No positives in data\n");
  }
  if (nn != 0) {
    printf(" True Negative Rate (Specificity): %4.2f\n",
           (double) tn / (double) nn);
  }
  else {
    printf(" True Negative Rate (Specificity): No negatives in data\n");
  }

END:

  NAG_FREE(isx);
  NAG_FREE(state);
  NAG_FREE(b);
  NAG_FREE(cov);
  NAG_FREE(eta);
  NAG_FREE(pred);
  NAG_FREE(se);
  NAG_FREE(seeta);
  NAG_FREE(sepred);
  NAG_FREE(t);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(v);
  NAG_FREE(offset);
  NAG_FREE(wt);

  return (exit_status);
}
```

9.2 Program Data

```
nag_rand_subsamp_xyw (g05pwc) Example Program Data
 Nag_Logistic  Nag_MeanInclude  40  4 :: link, mean, n, m
 0.0 -0.1  0.0  1.0      0.0  1.0
 0.4 -1.1  1.0  1.0      1.0  1.0
-0.5  0.2  1.0  0.0      0.0  1.0
 0.6  1.1  1.0  0.0      0.0  1.0
-0.3 -1.0  1.0  1.0      0.0  1.0
```

```
 2.8 -1.8  0.0  1.0      0.0  1.0
 0.4 -0.7  0.0  1.0      1.0  1.0
-0.4 -0.3  1.0  0.0      1.0  1.0
 0.5 -2.6  0.0  0.0      1.0  1.0
-1.6 -0.3  1.0  1.0      0.0  1.0
 0.4  0.6  1.0  0.0      0.0  1.0
-1.6  0.0  1.0  1.0      1.0  1.0
 0.0  0.4  1.0  1.0      1.0  1.0
-0.1  0.7  1.0  1.0      0.0  1.0
-0.2  1.8  1.0  1.0      0.0  1.0
-0.9  0.7  1.0  1.0      0.0  1.0
-1.1 -0.5  1.0  1.0      0.0  1.0
-0.1 -2.2  1.0  1.0      1.0  1.0
-1.8 -0.5  1.0  1.0      1.0  1.0
-0.8 -0.9  0.0  1.0      1.0  1.0
 1.9 -0.1  1.0  1.0      1.0  1.0
 0.3  1.4  1.0  1.0      0.0  1.0
 0.4 -1.2  1.0  0.0      1.0  1.0
 2.2  1.8  1.0  0.0      1.0  1.0
 1.4 -0.4  0.0  1.0      1.0  1.0
 0.4  2.4  1.0  1.0      0.0  1.0
-0.6  1.1  1.0  1.0      0.0  1.0
 1.4 -0.6  1.0  1.0      1.0  1.0
-0.1 -0.1  0.0  0.0      0.0  1.0
-0.6 -0.4  0.0  0.0      0.0  1.0
 0.6 -0.2  1.0  1.0      1.0  1.0
-1.8 -0.3  1.0  1.0      1.0  1.0
-0.3  1.6  1.0  1.0      0.0  1.0
-0.6  0.8  0.0  1.0      0.0  1.0
 0.3 -0.5  0.0  0.0      1.0  1.0
 1.6  1.4  1.0  1.0      0.0  1.0
-1.1  0.6  1.0  1.0      0.0  1.0
-0.3  0.6  1.0  1.0      0.0  1.0
-0.6  0.1  1.0  1.0      0.0  1.0
 1.0  0.6  1.0  1.0      1.0  1.0     :: End of x, y, t
 1    1    1    1                     :: isx
0  0.0  0.0  0                        :: print_iter,eps,tol,max_iter
Nag_MRG32k3a 0 42321                  :: genid, subid, seed
32                                    :: nt
10                                    :: nsamp
```

9.3 Program Results

```
nag_rand_subsamp_xyw (g05pwc) Example Program Results

                       Observed
             --------------------------
 Predicted | Negative  Positive   Total
 --------------------------------------
 Negative  |    38        20        58
 Positive  |     8        14        22
 Total     |    46        34        80

 True Positive Rate (Sensitivity): 0.41
 True Negative Rate (Specificity): 0.83
```