NAG Library Function Document

nag_glm_predict (g02gpc)

1 Purpose

nag_glm_predict (g02gpc) allows prediction from a generalized linear model fit via nag_glm_normal (g02gac), nag_glm_binomial (g02gbc), nag_glm_poisson (g02gcc) or nag_glm_gamma (g02gdc).

2 Specification

```
#include <nag.h>
#include <nagg02.h>
```

```
void nag_glm_predict (Nag_Distributions errfn, Nag_Link link,
     Nag_IncludeMean mean, Integer n, const double x[], Integer tdx,
     Integer m, const Integer sx[], Integer ip, const double binom_t[],
     const double offset[], const double wt[], double scale, double ex_power,
     const double b[], const double cov[], Nag_Boolean vfobs, double eta[],
     double seeta[], double pred[], double sepred[], NagError *fail)
```

3 Description

A generalized linear model consists of the following elements:

(i) A suitable distribution for the dependent variable y.

(ii) A linear model, with linear predictor $\eta = X\beta$, where X is a matrix of independent variables and β a column vector of p parameters.

(iii) A link function $g(.)$ between the expected value of y and the linear predictor, that is $E(y) = \mu = g(\eta)$.

In order to predict from a generalized linear model, that is estimate a value for the dependent variable, y, given a set of independent variables X, the matrix X must be supplied, along with values for the parameters β and their associated variance-covariance matrix, C. Suitable values for β and C are usually estimated by first fitting the prediction model to a training dataset with known responses, using for example nag_glm_normal (g02gac), nag_glm_binomial (g02gbc), nag_glm_poisson (g02gcc) or nag_glm_gamma (g02gdc). The predicted variable, and its standard error can then be obtained from:

$$\hat{y} = g^{-1}(\eta), \qquad \mathrm{se}(\hat{y}) = \sqrt{\left(\frac{\delta g^{-1}(x)}{\delta x}\right)_{\eta} \mathrm{se}(\eta) + I_{\mathrm{fobs}}\,\mathrm{Var}(y)}$$

where

$$\eta = o + X\beta, \qquad \mathrm{se}(\eta) = \mathrm{diag}\,\sqrt{XCX^{\mathrm{T}}},$$

o is a vector of offsets and $I_{\mathrm{fobs}} = 0$, if the variance of future observations is not taken into account, and 1 otherwise. Here $\mathrm{diag}\,A$ indicates the diagonal elements of matrix A.

If required, the variance for the ith future observation, $\mathrm{Var}(y_i)$, can be calculated as:

$$\mathrm{Var}(y_i) = \frac{\phi V(\theta)}{w_i}$$

where w_i is a weight, ϕ is the scale (or dispersion) parameter, and $V(\theta)$ is the variance function. Both the scale parameter and the variance function depend on the distribution used for the y, with:

Poisson $\quad V(\theta) = \mu_i$, $\phi = 1$

binomial $\quad V(\theta) = \frac{\mu_i(t_i - \mu_i)}{t_i}$, $\phi = 1$

Normal $V(\theta) = 1$

gamma $V(\theta) = \mu_i^2$

In the cases of a Normal and gamma error structure, the scale parameter (ϕ), is supplied by you. This value is usually obtained from the function used to fit the prediction model. In many cases, for a Normal error structure, $\phi = \hat{\sigma}^2$, i.e., the estimated variance.

4 References

McCullagh P and Nelder J A (1983) *Generalized Linear Models* Chapman and Hall

5 Arguments

1: **errfn** – Nag_Distributions *Input*

On entry: indicates the distribution used to model the dependent variable, y.

errfn = Nag_Binomial
The binomial distribution is used.

errfn = Nag_Gamma
The gamma distribution is used.

errfn = Nag_Normal
The Normal (Gaussian) distribution is used.

errfn = Nag_Poisson
The Poisson distribution is used.

Constraint: **errfn** = Nag_Binomial, Nag_Gamma, Nag_Normal or Nag_Poisson.

2: **link** – Nag_Link *Input*

On entry: indicates which link function to be used.

link = Nag_Compl
A complementary log-log link is used.

link = Nag_Expo
An exponent link is used.

link = Nag_Logistic
A logistic link is used.

link = Nag_Iden
An identity link is used.

link = Nag_Log
A log link is used.

link = Nag_Probit
A probit link is used.

link = Nag_Reci
A reciprocal link is used.

link = Nag_Sqrt
A square root link is used.

Details on the functional form of the different links can be found in the g02 Chapter Introduction.

Constraints:

if **errfn** = Nag_Binomial, **link** = Nag_Compl, Nag_Logistic or Nag_Probit;
otherwise **link** = Nag_Expo, Nag_Iden, Nag_Log, Nag_Reci or Nag_Sqrt.

3: **mean** – Nag_IncludeMean *Input*

On entry: indicates if a mean term is to be included.

mean = Nag_MeanInclude
A mean term, intercept, will be included in the model.

mean = Nag_MeanZero
The model will pass through the origin, zero-point.

Constraint: **mean** = Nag_MeanInclude or Nag_MeanZero.

4: **n** – Integer *Input*

On entry: n, the number of observations.

Constraint: **n** ≥ 1.

5: **x**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **x** must be at least **n** $\times$ **tdx**.

On entry: **x**$[(i-1) \times$ **tdx** $+ j - 1]$ must contain the ith observation for the jth independent variable, for $i = 1, 2, \ldots,$ **n** and $j = 1, 2, \ldots,$ **m**.

6: **tdx** – Integer *Input*

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

Constraint: **tdx** $\geq$ **m**

7: **m** – Integer *Input*

On entry: m, the total number of independent variables.

Constraint: **m** ≥ 1.

8: **sx**[**m**] – const Integer *Input*

On entry: indicates which independent variables are to be included in the model.

If **sx**$[j-1] > 0$, the jth independent variable is included in the regression model.

Constraints:

sx$[j-1] \geq 0$, for $i = 1, 2, \ldots,$ **m**;
if **mean** = Nag_MeanInclude, exactly **ip** $- 1$ values of **sx** must be > 0;
if **mean** = Nag_MeanZero, exactly **ip** values of **sx** must be > 0.

9: **ip** – Integer *Input*

On entry: the number of independent variables in the model, including the mean or intercept if present.

Constraint: **ip** > 0.

10: **binom_t**[**n**] – const double *Input*

On entry: if **errfn** = Nag_Binomial, **binom_t**$[i-1]$ must contain the binomial denominator, t_i, for the ith observation.

Otherwise **binom_t** is not referenced and may be **NULL**.

Constraint: if **errfn** = Nag_Binomial, **binom_t**$[i-1] \geq 0.0$, for $i = 1, 2, \ldots, n$.

11: **offset**[**n**] – const double *Input*

On entry: if an offset is required then **offset**$[i-1]$ must contain the value of the offset o_i, for the ith observation. Otherwise **offset** must be supplied as **NULL**.

12: **wt**[**n**] – const double *Input*

On entry: if weighted estimates are required then **wt**$[i-1]$ must contain the weight, ω_i for the ith observation. Otherwise **wt** must be supplied as **NULL**.

If **wt**$[i-1] = 0.0$, then the ith observation is not included in the model, in which case the effective number of observations is the number of observations with positive weights.

If **wt** = **NULL**, then the effective number of observations is n.

If the variance of future observations is not included in the standard error of the predicted variable, **wt** is not referenced.

Constraint: if **wt** is not **NULL** and **vfobs** = Nag_TRUE, **wt**$[i-1] \geq 0.0$, for $i = 1, 2, \ldots,$ **n**.

13: **scale** – double *Input*

On entry: if **errfn** = Nag_Normal or Nag_Gamma and **vfobs** = Nag_TRUE, the scale parameter, ϕ.

Otherwise **scale** is not referenced and $\phi = 1$.

Constraint: if **errfn** = Nag_Normal or Nag_Gamma and **vfobs** = Nag_TRUE, **scale** > 0.0.

14: **ex_power** – double *Input*

On entry: if **link** = Nag_Expo, **ex_power** must contain the power of the exponential.

If **link** $\neq$ Nag_Expo, **ex_power** is not referenced.

Constraint: if **link** = Nag_Expo, **ex_power** $\neq 0.0$.

15: **b**[**ip**] – const double *Input*

On entry: the model parameters, β.

If **mean** = Nag_MeanInclude, **b**$[0]$ must contain the mean parameter and **b**$[i]$ the coefficient of the variable contained in the jth independent **x**, where **sx**$[j-1]$ is the ith positive value in the array **sx**.

If **mean** = Nag_MeanZero, **b**$[i-1]$ must contain the coefficient of the variable contained in the jth independent **x**, where **sx**$[j-1]$ is the ith positive value in the array **sx**.

16: **cov**[**ip** × (**ip** + **1**)/**2**] – const double *Input*

On entry: the upper triangular part of the variance-covariance matrix, C, of the model parameters. This matrix should be supplied packed by column, i.e., the covariance between parameters β_i and β_j, that is the values stored in **b**$[i-1]$ and **b**$[j-1]$, should be supplied in **cov**$[j \times (j-1)/2 + i - 1]$, for $i = 1, 2, \ldots,$ **ip** and $j = i, \ldots,$ **ip**.

Constraint: the matrix represented in **cov** must be a valid variance-covariance matrix.

17: **vfobs** – Nag_Boolean *Input*

On entry: if **vfobs** = Nag_TRUE, the variance of future observations is included in the standard error of the predicted variable (i.e., $I_{\mathrm{fobs}} = 1$), otherwise $I_{\mathrm{fobs}} = 0$.

18: **eta**[**n**] – double *Output*

On exit: the linear predictor, η.

19: **seeta**[**n**] – double *Output*

On exit: the standard error of the linear predictor, $\mathrm{se}(\eta)$.

20: **pred**[**n**] – double *Output*

On exit: the predicted value, $\hat{y}$.

21: **sepred**[**n**] – double *Output*

On exit: the standard error of the predicted value, $\mathrm{se}(\hat{y})$. If **pred**$[i-1]$ could not be calculated, then nag_glm_predict (g02gpc) returns **fail.code** = NE_INVALID_PRED, and **sepred**$[i-1]$ is set to -99.0.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

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

On entry, the error type and link function combination supplied is invalid.

NE_INT

On entry, **ip** = ⟨*value*⟩.
Constraint: **ip** > 0.

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

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

NE_INT_2

On entry, **tdx** = ⟨*value*⟩ and **m** = ⟨*value*⟩.
Constraint: **tdx** $\geq$ **m**.

NE_INT_ARRAY_CONS

On entry, **sx** not consistent with **ip**: ⟨*value*⟩ values > 0, expected ⟨*value*⟩.

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 3.6.6 in the Essential Introduction for further information.

NE_INVALID_PRED

At least one predicted value could not be calculated as required. **sepred** is set to -99.0 for affected predicted values.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_REAL

On entry, **ex_power** = 0.0.

On entry, **scale** $= \langle value \rangle$.
Constraint: **scale** > 0.0.

NE_REAL_ARRAY_CONS

On entry, **cov**$[i-1] < 0.0$ for at least one diagonal element: $i = \langle value \rangle$, **cov**$[i-1] = \langle value \rangle$.

On entry, $i = \langle value \rangle$ and **binom_t**$[i-1] = \langle value \rangle$.
Constraint: **binom_t**$[i-1] \geq 0.0$, for all i.

On entry, $i = \langle value \rangle$ and **wt**$[i-1] = \langle value \rangle$.
Constraint: **wt**$[i-1] \geq 0.0$, for all i.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_glm_predict (g02gpc) is not threaded by NAG in any implementation.

nag_glm_predict (g02gpc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

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

None.

10 Example

The model

$$y = \frac{1}{\beta_1 + \beta_2 x} + \epsilon$$

is fitted to a training dataset with five observations. The resulting model is then used to predict the response for two new observations.

10.1 Program Text

```
/* nag_glm_predict (g02gpc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>

#define T_X(I, J) t_x[(I) *t_tdx + J]
#define X(I, J)   x[(I) *tdx + J]
int main(void)
{
  /* Integer scalar and array declarations */
```

```
  Integer            i, ip, j, m, n, t_n, tdx, t_tdx, print_iter;
  Integer            exit_status = 0, tdv, rank, lx, lt_x, lv;
  Integer            *sx = 0;
  /* NAG structures */
  Nag_Link           link;
  Nag_IncludeMean    mean;
  Nag_Boolean        vfobs, weight, t_weight, ioffset, t_ioffset;
  Nag_Distributions  errfn;
  NagError           fail;
  /* Character scalar and array declarations */
  char               sioffset[40], st_ioffset[40], sweight[40], st_weight[40];
  char               slink[40], smean[40], svfobs[40];
  /* Double scalar and array declarations */
  double             rss, scale, ex_power, df;
  double             *b = 0, *cov = 0, *eta = 0, *offset = 0, *t_offset = 0;
  double             *pred = 0, *se = 0, *seeta = 0, *sepred = 0, *binom_t = 0;
  double             *v = 0, *wt = 0, *x = 0, *y = 0, *t_x = 0, *t_wt = 0;
  /* Set control parameters */
  double             eps = 0.000001;
  double             tol = 0.00005;
  Integer            max_iter = 10;

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

  printf("nag_glm_predict (g02gpc) Example Program Results\n");

  /* Skip headings in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
  /* Read in training data for model that will be used for prediction */
#ifdef _WIN32
  scanf_s("%39s %39s %39s %39s %"NAG_IFMT" %"NAG_IFMT" %lf %"NAG_IFMT"%*[^\n] ",
          slink, _countof(slink), smean, _countof(smean), st_ioffset,
          _countof(st_ioffset), st_weight, _countof(st_weight), &t_n, &m,
          &scale, &print_iter);
#else
  scanf("%39s %39s %39s %39s %"NAG_IFMT" %"NAG_IFMT" %lf %"NAG_IFMT"%*[^\n] ",
        slink, smean, st_ioffset, st_weight, &t_n, &m, &scale, &print_iter);
#endif
  /*
   * nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  link = (Nag_Link) nag_enum_name_to_value(slink);
  mean = (Nag_IncludeMean) nag_enum_name_to_value(smean);
  t_ioffset = (Nag_Boolean) nag_enum_name_to_value(st_ioffset);
  t_weight = (Nag_Boolean) nag_enum_name_to_value(st_weight);

  t_tdx = m;
  lt_x = t_tdx * t_n;

  /* Allocate memory */
  if (t_weight)
    {
      if (!(t_wt = NAG_ALLOC(t_n, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  if (t_ioffset)
```

```
    {
      if (!(t_offset = NAG_ALLOC(t_n, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  if (!(t_x = NAG_ALLOC(lt_x, double)) ||
      !(y = NAG_ALLOC(t_n, double)) ||
      !(sx = NAG_ALLOC(m, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Read in the data */
  for (i = 0; i < t_n; i++)
    {
      for (j = 0; j < m; j++)
#ifdef _WIN32
        scanf_s("%lf", &T_X(i, j));
#else
        scanf("%lf", &T_X(i, j));
#endif
#ifdef _WIN32
      scanf_s("%lf", &y[i]);
#else
      scanf("%lf", &y[i]);
#endif
      if (t_ioffset)
#ifdef _WIN32
        scanf_s("%lf", &t_offset[i]);
#else
        scanf("%lf", &t_offset[i]);
#endif
      if (t_weight)
#ifdef _WIN32
        scanf_s("%lf", &t_wt[i]);
#else
        scanf("%lf", &t_wt[i]);
#endif
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }

  for (j = 0; j < m; j++)
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[^\n] ", &sx[j]);
#else
    scanf("%"NAG_IFMT"%*[^\n] ", &sx[j]);
#endif

  if (link == Nag_Expo)
#ifdef _WIN32
    scanf_s("%lf%*[^\n] ", &ex_power);
#else
    scanf("%lf%*[^\n] ", &ex_power);
#endif
  else
    ex_power = 0.0;

  /* Calculate ip */
  ip = 0;
  for (j = 0; j < m; j++)
    if (sx[j] > 0) ip++;
  if (mean == Nag_MeanInclude)
```

```
    ip++;

  tdv = ip+6;
  lv = tdv * t_n;

  if (!(b = NAG_ALLOC(ip, double)) ||
      !(v = NAG_ALLOC(lv, double)) ||
      !(se = NAG_ALLOC(ip, double)) ||
      !(cov = NAG_ALLOC(ip*(ip+1)/2, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Call nag_glm_normal (g02gac) to fit model to training data */
  nag_glm_normal(link, mean, t_n, t_x, t_tdx, m, sx, ip, y, t_wt, t_offset,
                 &scale, ex_power, &rss, &df, b, &rank, se, cov, v, tdv,
                 tol, max_iter, print_iter, "", eps, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_glm_normal (g02gac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Display parameter estimates for training data */
  printf(
         "\nResidual sum of squares = %12.4g, Degrees of freedom = %2f\n\n",
         rss, df);
  printf("       Estimate     Standard error\n\n");
  for (i = 0; i < ip; i++)
    printf(" %14.4f %14.4f\n", b[i], se[i]);
  printf("\n");

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

  /* Read in data to predict from and check array sizes */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT" %39s %39s %39s%*[^\n] ", &n, svfobs, _countof(svfobs),
          sioffset, _countof(sioffset), sweight, _countof(sweight));
#else
  scanf("%"NAG_IFMT" %39s %39s %39s%*[^\n] ", &n, svfobs, sioffset, sweight);
#endif
  /*
   * nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  vfobs = (Nag_Boolean) nag_enum_name_to_value(svfobs);
  ioffset = (Nag_Boolean) nag_enum_name_to_value(sioffset);
  weight = (Nag_Boolean) nag_enum_name_to_value(sweight);

  if (weight)
    {
      if (!(wt = NAG_ALLOC(n, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  if (ioffset)
    {
      if (!(offset = NAG_ALLOC(n, double)))
        {
          printf("Allocation failure\n");
```

```
          exit_status = -1;
          goto END;
        }
    }

  tdx = m;
  lx = tdx * n;

  if (!(x = NAG_ALLOC(lx, double)) ||
      !(eta = NAG_ALLOC(n, double)) ||
      !(seeta = NAG_ALLOC(n, double)) ||
      !(pred = NAG_ALLOC(n, double)) ||
      !(sepred = NAG_ALLOC(n, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  for (i = 0; i < n; i++)
    {
      for (j = 0; j < m; j++)
#ifdef _WIN32
        scanf_s("%lf", &X(i, j));
#else
        scanf("%lf", &X(i, j));
#endif
      if (offset)
#ifdef _WIN32
        scanf_s("%lf", &offset[i]);
#else
        scanf("%lf", &offset[i]);
#endif
      if (weight)
#ifdef _WIN32
        scanf_s("%lf", &wt[i]);
#else
        scanf("%lf", &wt[i]);
#endif
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
    }

  /* Using nag_glm_normal (g02gac) to fit training model, so error structure
     is normal */
  errfn = Nag_Normal;

  /* Call nag_glm_predict (g02gpc) to calculate predictions */
  nag_glm_predict(errfn, link, mean, n, x, tdx, m, sx, ip, binom_t, 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;
    }

  /* Display predicted values */
  printf(
          "   I      ETA         SE(ETA)      Predicted  SE(Predicted)\n");
  printf("\n");
  for (i = 0; i < n; i++)
    {
      printf(" %3"NAG_IFMT") %10.5f    %10.5f    %10.5f    %10.5f\n", i+1,
              eta[i], seeta[i], pred[i], sepred[i]);
    }
```

```
 END:
  NAG_FREE(t_wt);
  NAG_FREE(t_x);
  NAG_FREE(y);
  NAG_FREE(sx);
  NAG_FREE(b);
  NAG_FREE(v);
  NAG_FREE(se);
  NAG_FREE(cov);
  NAG_FREE(wt);
  NAG_FREE(x);
  NAG_FREE(offset);
  NAG_FREE(eta);
  NAG_FREE(seeta);
  NAG_FREE(pred);
  NAG_FREE(sepred);

  return exit_status;
}
```

10.2 Program Data

```
nag_glm_predict (g02gpc) Example Program Data

Training Data
Nag_Reci
Nag_MeanInclude
Nag_FALSE
Nag_FALSE
5 1 0.0 0            : slink,smean,st_ioffset,st_weight,t_n,m,scale,print_iter
1.0 25.0             : t_x,y
2.0 10.0
3.0  6.0
4.0  4.0
5.0  3.0
1                    : sx

Prediction Data
2 Nag_TRUE
Nag_FALSE Nag_FALSE : n,svfobs,soffset,sweight
32.0                 : x
18.0
```

10.3 Program Results

```
nag_glm_predict (g02gpc) Example Program Results

Residual sum of squares =       0.3872, Degrees of freedom = 3.000000

       Estimate     Standard error

        -0.0239        0.0028
         0.0638        0.0026

   I      ETA         SE(ETA)      Predicted  SE(Predicted)

   1)    2.01807       0.08168       0.49552       0.35981
   2)    1.12472       0.04476       0.88911       0.36098
```