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

```c
#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 $\beta$ 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 $\beta$ and their associated variance-covariance matrix, $C$. Suitable values for $\beta$ 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:

\[
\hat{y} = g^{-1}(\eta), \quad \text{se}(\hat{y}) = \sqrt{\left(\frac{\delta g^{-1}(x)}{\delta x}\right)_{\eta} \text{se}(\eta) + I_{\text{fobs}} \text{Var}(y)}
\]

where

\[\eta = o + X\beta, \quad \text{se}(\eta) = \text{diag} \sqrt{XCX^T},\]

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

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

\[\text{Var}(y_i) = \frac{\phi V(\theta)}{w_i}\]

where $w_i$ is a weight, $\phi$ 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:

Poission \quad V(\theta) = \mu_i, \quad \phi = 1

Binomial \quad V(\theta) = \frac{\mu_i(\mu_i - \mu_i)}{\mu_i}, \quad \phi = 1
Normal \[ V(\theta) = 1 \]

gamma \[ V(\theta) = \mu^2 \]

In the cases of a Normal and gamma error structure, the scale parameter (\( \phi \)), 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


## 5 Arguments

1: \textbf{errfn} – Nag_Distributions

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

\textbf{errfn} = Nag_Binomial

The binomial distribution is used.

\textbf{errfn} = Nag_Gamma

The gamma distribution is used.

\textbf{errfn} = Nag_Normal

The Normal (Gaussian) distribution is used.

\textbf{errfn} = Nag_Poisson

The Poisson distribution is used.

Constraint: \textbf{errfn} = Nag_Binomial, Nag_Gamma, Nag_Normal or Nag_Poisson.

2: \textbf{link} – Nag_Link

On entry: indicates which link function to be used.

\textbf{link} = Nag_Compl

A complementary log-log link is used.

\textbf{link} = Nag_Expo

An exponent link is used.

\textbf{link} = Nag_Logistic

A logistic link is used.

\textbf{link} = Nag_Iden

An identity link is used.

\textbf{link} = Nag_Log

A log link is used.

\textbf{link} = Nag_Probit

A probit link is used.

\textbf{link} = Nag_Reci

A reciprocal link is used.

\textbf{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 \textbf{errfn} = Nag_Binomial, \textbf{link} = Nag_Compl, Nag_Logistic or Nag_Probit;
otherwise \textbf{link} = Nag_Expo, Nag_Iden, Nag_Log, Nag_Reci or Nag_Sqrt.
3:  \textbf{mean} - Nag\_IncludeMean \hfill \textit{Input} \\
On entry: indicates if a mean term is to be included. \\
\textbf{mean} = Nag\_MeanInclude \\
A mean term, intercept, will be included in the model. \\
\textbf{mean} = Nag\_MeanZero \\
The model will pass through the origin, zero-point. \\
\textit{Constraint:} mean = Nag\_MeanInclude or Nag\_MeanZero.

4:  \textbf{n} - Integer \hfill \textit{Input} \\
On entry: \textit{n}, the number of observations. \\
\textit{Constraint:} \textit{n} \geq 1.

5:  \textbf{x}[\text{dim}] - const double \hfill \textit{Input} \\
\textit{Note:} the dimension, \textit{dim}, of the array \textit{x} must be at least \textit{n} \times tdx. \\
On entry: \(x((i-1) \times tdx + j - 1)\) must contain the \textit{i}th observation for the \textit{j}th independent variable, for \(i = 1, 2, \ldots, \text{n}\) and \(j = 1, 2, \ldots, \text{m}\).

6:  \textbf{tdx} - Integer \hfill \textit{Input} \\
On entry: the stride separating matrix column elements in the array \textit{x}. \\
\textit{Constraint:} \textit{tdx} \geq \textit{m}.

7:  \textbf{m} - Integer \hfill \textit{Input} \\
On entry: \textit{m}, the total number of independent variables. \\
\textit{Constraint:} \textit{m} \geq 1.

8:  \textbf{sx}[\text{m}] - const Integer \hfill \textit{Input} \\
On entry: indicates which independent variables are to be included in the model. \\
If \(sx[j-1] > 0\), the \textit{j}th independent variable is included in the regression model. \\
\textit{Constraints:} \\
\(sx[j-1] \geq 0\), for \(i = 1, 2, \ldots, \text{m}\); \\
if mean = Nag\_MeanInclude, exactly \textit{ip} - 1 values of \textit{sx} must be \(> 0\); \\
if mean = Nag\_MeanZero, exactly \textit{ip} values of \textit{sx} must be \(> 0\).

9:  \textbf{ip} - Integer \hfill \textit{Input} \\
On entry: the number of independent variables in the model, including the mean or intercept if present. \\
\textit{Constraint:} \textit{ip} > 0.

10:  \textbf{binom\_t}[\text{n}] - const double \hfill \textit{Input} \\
On entry: if errfn = Nag\_Binomial, \textbf{binom\_t}[i-1] must contain the binomial denominator, \(t_i\), for the \textit{i}th observation. \\
Otherwise \textbf{binom\_t} is not referenced and may be NULL. \\
\textit{Constraint:} if errfn = Nag\_Binomial, \textbf{binom\_t}[i-1] \geq 0.0, for \(i = 1, 2, \ldots, \text{n}\).

11:  \textbf{offset}[\text{n}] - const double \hfill \textit{Input} \\
On entry: if an offset is required then \textbf{offset}[i-1] must contain the value of the offset \(o_i\), for the \textit{i}th observation. Otherwise \textbf{offset} must be supplied as NULL.
On entry: if weighted estimates are required then $wt[i-1]$ must contain the weight, $\omega_i$ for the $i$th observation. Otherwise $wt$ must be supplied as NULL.

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

If $wt = \text{NULL}$, 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 $vfobs = \text{Nag\_TRUE}$, $wt[i-1] \geq 0.0$, for $i = 1, 2, \ldots, n$.

On entry: if $\text{erf} \equiv \text{Nag\_Normal or Nag\_Gamma and } vfobs = \text{Nag\_TRUE}$, the scale parameter, $\phi$. Otherwise $scale$ is not referenced and $\phi = 1$.

Constraint: if $\text{erf} \equiv \text{Nag\_Normal or Nag\_Gamma and } vfobs = \text{Nag\_TRUE}$, $scale > 0.0$.

On entry: if $\text{link} \equiv \text{Nag\_Expo}$, $\text{ex\_power}$ must contain the power of the exponential. If $\text{link} \neq \text{Nag\_Expo}$, $\text{ex\_power}$ is not referenced.

Constraint: if $\text{link} \equiv \text{Nag\_Expo}$, $\text{ex\_power} \neq 0.0$.

On entry: the model parameters, $\beta$.

If $\text{mean} = \text{Nag\_MeanInclude}$, $b[0]$ must contain the mean parameter and $b[i]$ the coefficient of the variable contained in the $j$th independent $x$, where $sx[j-1]$ is the $i$th positive value in the array $sx$.

If $\text{mean} = \text{Nag\_MeanZero}$, $b[i-1]$ must contain the coefficient of the variable contained in the $j$th independent $x$, where $sx[j-1]$ is the $i$th positive value in the array $sx$.

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 $\beta_i$ and $\beta_j$, 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 $i = 1, 2, \ldots, \text{ip}$ and $j = i, \ldots, \text{ip}$.

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

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

On exit: the linear predictor, $\eta$.

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

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

---

**Note:**

1. $wt$ is an array that must be supplied as a double array.
2. $scale$ is a double variable.
3. $ex\_power$ is a double variable.
4. $b$ is an array that must be supplied as a double array.
5. $cov$ is an array that must be supplied as a double array.
6. $vfobs$ is a Nag_Boolean variable.
7. $eta$ is a double variable.
8. $seeta$ is a double variable.
9. $pred$ is a double variable.


21:  \texttt{sepred[n]} – double  
\textbf{Output}  
On exit: the standard error of the predicted value, se(\(\hat{y}\)). If \texttt{pred[i]} could not be calculated, then \texttt{nag_glm_predict} (g02gpc) returns \texttt{fail.code} = NE_INVALID_PRED, and \texttt{sepred[i]} is set to \(-99.0\).

22:  \texttt{fail} – NagError *  
\textbf{Input/Output}  
The NAG error argument (see Section 3.6 in the Essential Introduction).

6  Error Indicators and Warnings

\textbf{NE_ALLOC_FAIL}  
Dynamic memory allocation failed.  
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_BAD_PARAM}  
On entry, argument \texttt{value} had an illegal value.  
On entry, the error type and link function combination supplied is invalid.

\textbf{NE_INT}  
On entry, \texttt{ip} = \texttt{value}.  
Constraint: \texttt{ip} > 0.

On entry, \texttt{m} = \texttt{value}.  
Constraint: \texttt{m} \geq 1.

On entry, \texttt{n} = \texttt{value}.  
Constraint: \texttt{n} \geq 1.

\textbf{NE_INT_2}  
On entry, \texttt{tdx} = \texttt{value} and \texttt{m} = \texttt{value}.  
Constraint: \texttt{tdx} \geq \texttt{m}.

\textbf{NE_INT_ARRAY_CONS}  
On entry, \texttt{sx} not consistent with \texttt{ip}: \texttt{value} values > 0, expected \texttt{value}.

\textbf{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.

\textbf{NE_INVALID_PRED}  
At least one predicted value could not be calculated as required. \texttt{sepred} is set to \(-99.0\) for affected predicted values.

\textbf{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.

\textbf{NE_REAL}  
On entry, \texttt{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] ≥ 0.0, for all i.
On entry, i = \langle value\rangle and wt[i - 1] = \langle value\rangle.
Constraint: wt[i - 1] ≥ 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 */
#ifndef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
#ifndef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
/* Read in training data for model that will be used for prediction */
#ifndef _WIN32
    scanf_s("%39s %39s %39s %39s "%NAG_IFMT" "%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"%*[\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)) ||
!(t_ioffset = NAG_ALLOC(t_n, double)) ||
!(t_weight = NAG_ALLOC(t_n, double)))
{
    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

        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",  
rss, df);

printf(" %14.4f %14.4f\n", b[i], se[i]);

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

#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

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") %8.5f %8.5f %8.5f %8.5f
", 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

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Standard error</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.0239</td>
<td>0.0028</td>
</tr>
<tr>
<td>0.0638</td>
<td>0.0026</td>
</tr>
</tbody>
</table>

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