NAG Library Function Document

nag_mv_gaussian_mixture (g03gac)

1 Purpose

nag_mv_gaussian_mixture (g03gac) performs a mixture of Normals (Gaussians) for a given (co)variance structure.

2 Specification

```
#include <nag.h>
#include <nagg03.h>
```

```
void nag_mv_gaussian_mixture (Integer n, Integer m, const double x[],
     Integer pdx, const Integer isx[], Integer nvar, Integer ng,
     Nag_Boolean popt, double prob[], Integer tdprob, Integer *niter,
     Integer riter, double w[], double g[], Nag_VarCovar sopt, double s[],
     double f[], double tol, double *loglik, NagError *fail)
```

3 Description

A Normal (Gaussian) mixture model is a weighted sum of k group Normal densities given by,

$$p(x \mid w, \mu, \Sigma) = \sum_{j=1}^{k} w_j g(x \mid \mu_j, \Sigma_j), \quad x \in \mathbb{R}^p$$

where:

x is a p-dimensional object of interest;

w_j is the mixture weight for the jth group and $\sum_{j=1}^{k} w_j = 1$;

μ_j is a p-dimensional vector of means for the jth group;

Σ_j is the covariance structure for the jth group;

$g(\cdot)$ is the p-variate Normal density:

$$g(x \mid \mu_j, \Sigma_j) = \frac{1}{(2\pi)^{p/2} \left|\Sigma_j\right|^{1/2}} \exp\left[-\frac{1}{2}(x - \mu_j)\Sigma_j^{-1}(x - \mu_j)^{\mathrm{T}}\right].$$

Optionally, the (co)variance structure may be pooled (common to all groups) or calculated for each group, and may be full or diagonal.

4 References

Hartigan J A (1975) *Clustering Algorithms* Wiley

5 Arguments

1: **n** – Integer *Input*

On entry: n, the number of objects. There must be more objects than parameters in the model.

Constraints:

if **sopt** $=$ Nag_GroupCovar, **n** $>$ **ng** $\times$ (**nvar** $\times$ **nvar** $+$ **nvar**);
if **sopt** $=$ Nag_PooledCovar, **n** $>$ **nvar** $\times$ (**ng** $+$ **nvar**);

if **sopt** = Nag_GroupVar, $\mathbf{n} > 2 \times \mathbf{ng} \times \mathbf{nvar}$;
if **sopt** = Nag_PooledVar, $\mathbf{n} > \mathbf{nvar} \times (\mathbf{ng} + 1)$;
if **sopt** = Nag_OverallVar, $\mathbf{n} > \mathbf{nvar} \times \mathbf{ng} + 1$.

2: **m** – Integer *Input*

On entry: the total number of variables in array **x**.

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

3: **x**[**n** × **pdx**] – const double *Input*

On entry: $\mathbf{x}[(i-1) \times \mathbf{pdx} + j - 1]$ must contain the value of the jth variable for the ith object, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, \mathbf{m}$.

4: **pdx** – Integer *Input*

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

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

5: **isx**[**m**] – const Integer *Input*

On entry: if **nvar** = **m** all available variables are included in the model and **isx** is not referenced; otherwise the jth variable will be included in the analysis if $\mathbf{isx}[j-1] = 1$ and excluded if $\mathbf{isx}[j-1] = 0$, for $j = 1, 2, \ldots, \mathbf{m}$.

Constraint: if $\mathbf{nvar} \neq \mathbf{m}$, $\mathbf{isx}[j-1] = 1$ for **nvar** values of j and $\mathbf{isx}[j-1] = 0$ for the remaining $\mathbf{m} - \mathbf{nvar}$ values of j, for $j = 1, 2, \ldots, \mathbf{m}$.

6: **nvar** – Integer *Input*

On entry: p, the number of variables included in the calculations.

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

7: **ng** – Integer *Input*

On entry: k, the number of groups in the mixture model.

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

8: **popt** – Nag_Boolean *Input*

On entry: if **popt** = Nag_TRUE, the initial membership probabilities in **prob** are set internally; otherwise these probabilities must be supplied.

9: **prob**[**n** × **tdprob**] – double *Input/Output*

On entry: if **popt** ≠ Nag_TRUE, $\mathbf{prob}[(i-1) \times \mathbf{tdprob} + j - 1]$ is the probability that the ith object belongs to the jth group. (These probabilities are normalised internally.)

On exit: $\mathbf{prob}[(i-1) \times \mathbf{tdprob} + j - 1]$ is the probability of membership of the ith object to the jth group for the fitted model.

10: **tdprob** – Integer *Input*

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

Constraint: $\mathbf{tdprob} \geq \mathbf{ng}$.

11: **niter** – Integer * *Input/Output*

On entry: the maximum number of iterations.

Suggested value: 15

On exit: the number of completed iterations.

Constraint: **niter** ≥ 1.

12: **riter** – Integer *Input*

On entry: if **riter** > 0, membership probabilities are rounded to 0.0 or 1.0 after the completion of every **riter** iterations.

Suggested value: 5

13: **w**[**ng**] – double *Output*

On exit: w_j, the mixing probability for the jth group.

14: **g**[**nvar** $\times$ **ng**] – double *Output*

On exit: **g**$[(i-1) \times$ **ng** $+ j - 1]$ gives the estimated mean of the ith variable in the jth group.

15: **sopt** – Nag_VarCovar *Input*

On entry: determines the (co)variance structure:

sopt = Nag_GroupCovar
Groupwise covariance matrices.

sopt = Nag_PooledCovar
Pooled covariance matrix.

sopt = Nag_GroupVar
Groupwise variances.

sopt = Nag_PooledVar
Pooled variances.

sopt = Nag_OverallVar
Overall variance.

Constraint: **sopt** = Nag_GroupCovar, Nag_PooledCovar, Nag_GroupVar, Nag_PooledVar or Nag_OverallVar.

16: **s**[*dim*] – double *Output*

Note: the dimension, *dim*, of the array **s** must be at least $a \times b \times c$.

Where $\mathbf{S}(i,j,k)$ appears in this document, it refers to the array element **s**$[(k-1) \times a \times b + (j-1) \times a + i - 1]$.

On exit: if **sopt** = Nag_GroupCovar, $\mathbf{S}(i,j,k)$ gives the (i,j)th element of the kth group, with $a = b =$ **nvar** and $c =$ **ng**.

If **sopt** = Nag_PooledCovar, $\mathbf{S}(i,j,1)$ gives the (i,j)th element of the pooled covariance, with $a = b =$ **nvar** and $c = 1$.

If **sopt** = Nag_GroupVar, $\mathbf{S}(j,k,1)$ gives the jth variance in the kth group, with $a =$ **nvar**, $b =$ **ng** and $c = 1$.

If **sopt** = Nag_PooledVar, $\mathbf{S}(j,1,1)$ gives the jth pooled variance., with $a =$ **nvar** and $b = c = 1$

If **sopt** = Nag_OverallVar, $\mathbf{S}(1,1,1)$ gives the overall variance, with $a = b = c = 1$.

17: **f**[**n** $\times$ **ng**] – double *Output*

On exit: **f**$[(i-1) \times$ **ng** $+ j - 1]$ gives the p-variate Normal (Gaussian) density of the ith object in the jth group.

18: **tol** – double *Input*

On entry: iterations cease the first time an improvement in log-likelihood is less than **tol**. If $\mathbf{tol} \leq 0$ a value of 10^{-3} is used.

19: **loglik** – double * *Output*

On exit: the log-likelihood for the fitted mixture model.

20: **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_ARRAY_SIZE

On entry, $\mathbf{pdx} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{pdx} \geq \mathbf{n}$.

On entry, $\mathbf{tdprob} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{tdprob} \geq \mathbf{n}$.

NE_BAD_PARAM

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

NE_CLUSTER_EMPTY

An iteration cannot continue due to an empty group, try a different initial allocation.

NE_INT

On entry, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 1$.

On entry, $\mathbf{ng} = \langle value \rangle$.
Constraint: $\mathbf{ng} \geq 1$.

On entry, $\mathbf{niter} = \langle value \rangle$.
Constraint: $\mathbf{niter} \geq 1$.

NE_INT_2

On entry, $\mathbf{nvar} = \langle value \rangle$ and $\mathbf{m} = \langle value \rangle$.
Constraint: $1 \leq \mathbf{nvar} \leq \mathbf{m}$.

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 3.6.6 in the Essential Introduction for further information.

NE_MAT_NOT_POS_DEF

A covariance matrix is not positive definite, try a different initial allocation.

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_OBSERVATIONS

On entry, **n** $= \langle value \rangle$ and $p = \langle value \rangle$.
Constraint: **n** $> p$, the number of parameters, i.e., too few objects have been supplied for the model.

NE_PROBABILITY

On entry, row $\langle value \rangle$ of supplied **prob** does not sum to 1.

NE_VAR_INCL_INDICATED

On entry, **nvar** $\neq$ **m** and **isx** is invalid.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_mv_gaussian_mixture (g03gac) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_mv_gaussian_mixture (g03gac) 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

This example fits a Gaussian mixture model with pooled covariance structure to New Haven schools test data, see Table 5.1 (p. 118) in Hartigan (1975).

10.1 Program Text

```
/* nag_mv_gaussian_mixture (g03gac) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg03.h>
#include <nagx04.h>

#define S(I,J,K) s[I-1 + (J-1)*(sopt==Nag_GroupVar ?ng:nvar) + (K-1)*nvar*nvar]
#define X(I,J) x[(I-1)*tdx + J-1]
#define PROB(I,J) prob[(I-1)*tdprob + J-1]
```

```
#define G(I,J) g[(I-1)*ng + J-1]
#define F(I,J) f[(I-1)*ng + J-1]

int main(void)
{
    /* Integer scalar and array declarations */
    Integer     exit_status = 0, i, j, lens, m, n, ng, niter, nvar, riter,
        tdprob, tdx;
    Integer     *isx = 0;

    /* Double scalar and array declarations */
    double      loglik, tol;
    double      *f = 0, *g = 0, *prob = 0, *s = 0, *w = 0, *x = 0;

    /* NAG structures */
    Nag_Boolean  popt;
    Nag_VarCovar sopt;
    NagError     fail;

    /* Character scalar and array declarations */
    char         nag_enum_popt[30+1], nag_enum_sopt[30+1];

    printf("nag_mv_gaussian_mixture (g03gac) Example Program Results\n\n");
    fflush(stdout);

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

    /* Problem size */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"", &n);
#else
    scanf("%"NAG_IFMT"", &n);
#endif
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"", &m);
#else
    scanf("%"NAG_IFMT"", &m);
#endif
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"", &nvar);
#else
    scanf("%"NAG_IFMT"", &nvar);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    /* Number of groups */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"", &ng);
#else
    scanf("%"NAG_IFMT"", &ng);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    /* Scaling option */
#ifdef _WIN32
    scanf_s("%30s", nag_enum_sopt, _countof(nag_enum_sopt));
#else
    scanf("%30s", nag_enum_sopt);
```

```
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    /* Initial probabilities option */
#ifdef _WIN32
    scanf_s("%30s", nag_enum_popt, _countof(nag_enum_popt));
#else
    scanf("%30s", nag_enum_popt);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    /* Maximum number of iterations */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"", &niter);
#else
    scanf("%"NAG_IFMT"", &niter);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    /* Principal dimensions */
    tdx = nvar;
    tdprob = ng;

    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    popt = (Nag_Boolean)nag_enum_name_to_value(nag_enum_popt);
    sopt = (Nag_VarCovar)nag_enum_name_to_value(nag_enum_sopt);

    /* Variance/covariance array */
    switch (sopt)
    {
    case Nag_GroupCovar:
        lens = nvar*nvar*ng;
        break;
    case Nag_PooledCovar:
        lens = nvar*nvar;
        break;
    case Nag_GroupVar:
        lens = nvar*ng;
        break;
    case Nag_PooledVar:
        lens = nvar;
        break;
    case Nag_OverallVar:
        lens = 1;
        break;
    }

    if (!(x = NAG_ALLOC(n*tdx, double)) ||
        !(prob = NAG_ALLOC(n*tdprob, double)) ||
        !(g = NAG_ALLOC(ng*nvar, double)) ||
        !(w = NAG_ALLOC(ng, double)) ||
        !(isx = NAG_ALLOC(m, Integer)) ||
        !(f = NAG_ALLOC(ng*n, double)) ||
        !(s = NAG_ALLOC(lens, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Data matrix X */
    for (i=1; i<=n; i++)
        for (j=1;j<=m; j++)
#ifdef _WIN32
            scanf_s("%lf", &X(i,j));
#else
            scanf("%lf", &X(i,j));
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif

    /* Included variables */
    if (nvar != m)
    {
        for (j=1; j<=m; j++)
#ifdef _WIN32
            scanf_s("%"NAG_IFMT"", &isx[j-1]);
#else
            scanf("%"NAG_IFMT"", &isx[j-1]);
#endif
#ifdef _WIN32
        scanf_s("%*[^\n] ");
#else
        scanf("%*[^\n] ");
#endif
    }

    /* Optionally read initial probabilities of group membership */
    if (popt==Nag_FALSE)
    {
        for (i=1; i<=n; i++)
            for (j=1; j<=ng; j++)
#ifdef _WIN32
                scanf_s("%lf", &PROB(i,j));
#else
                scanf("%lf", &PROB(i,j));
#endif
#ifdef _WIN32
        scanf_s("%*[^\n] ");
#else
        scanf("%*[^\n] ");
#endif
    }

    /* Optimisation parameters */
    tol = 0.0;
    riter = 5;

    /* Fit the model */
    /* nag_mv_gaussian_mixture (g03gac).
     * Computes a Gaussian mixture model
     */
    INIT_FAIL(fail);
    nag_mv_gaussian_mixture(n, m, x, tdx, isx, nvar, ng, popt, prob, tdprob,
                            &niter, riter, w, g, sopt, s, f, tol, &loglik,
                            &fail);

    if (fail.code != NE_NOERROR)
      {
        printf("nag_mv_gaussian_mixture (g03gac) failed.\n%s\n",fail.message);
        exit_status = 1;
        goto END;
      }

    /* Results */
    /* nag_gen_real_mat_print (x04cac).
     * Print real general matrix (easy-to-use)
     */
    nag_gen_real_mat_print(Nag_RowMajor, Nag_GeneralMatrix, Nag_NonUnitDiag,
                           1, ng, w, ng, "Mixing proportions", NULL, &fail);

    nag_gen_real_mat_print(Nag_RowMajor, Nag_GeneralMatrix, Nag_NonUnitDiag,
                           nvar, ng, g, ng, "\n Group means", NULL, &fail);

    /* Variance/Covariance */
    switch (sopt) {
    case Nag_GroupCovar:
      for (i=1; i<=ng; i++)
        {
          nag_gen_real_mat_print(Nag_RowMajor, Nag_GeneralMatrix,
                                 Nag_NonUnitDiag, nvar, nvar, &S(1,1,i), nvar,
                                 "\n Variance-covariance matrix", NULL, &fail);
        }
      break;
    case Nag_PooledCovar:
      nag_gen_real_mat_print(Nag_RowMajor, Nag_GeneralMatrix, Nag_NonUnitDiag,
                             nvar, nvar, s, nvar,
                             "\n Pooled Variance-covariance matrix", NULL,
                             &fail);
      break;
    case Nag_GroupVar:
      nag_gen_real_mat_print(Nag_RowMajor, Nag_GeneralMatrix, Nag_NonUnitDiag,
                             nvar, ng, s, ng, "\n Groupwise Variance", NULL,
                             &fail);
      break;
    case Nag_PooledVar:
      nag_gen_real_mat_print(Nag_RowMajor, Nag_GeneralMatrix, Nag_NonUnitDiag,
                             nvar, 1, s, 1, "\n Pooled Variance", NULL, &fail);
      break;
    case Nag_OverallVar:
      printf("\n Overall Variance = %g\n", S(1,1,1));
      break;
    }

    nag_gen_real_mat_print(Nag_RowMajor, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                           ng, f, ng, "\n Densities", NULL, &fail);

    nag_gen_real_mat_print(Nag_RowMajor, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                           ng, prob, ng, "\n Membership probabilities", NULL,
                           &fail);

    printf("\nNo. iterations: %"NAG_IFMT"\n", niter);
    printf("Log-likelihood: %g\n\n", loglik);

  END:
    NAG_FREE(f);
    NAG_FREE(g);
    NAG_FREE(prob);
    NAG_FREE(s);
    NAG_FREE(w);
    NAG_FREE(x);
    NAG_FREE(isx);

    return exit_status;
}
```

10.2 Program Data

```
nag_mv_gaussian_mixture (g03gac) Example Program Data
25 4 4          : n m ip
2               : ng
Nag_PooledCovar : sopt
Nag_FALSE       : popt
15              : niter
2.7 3.2 4.5 4.8
```

```
3.9 3.8 5.9 6.2
4.8 4.1 6.8 5.5
3.1 3.5 4.3 4.6
3.4 3.7 5.1 5.6
3.1 3.4 4.1 4.7
4.6 4.4 6.6 6.1
3.1 3.3 4.0 4.9
3.8 3.7 4.7 4.9
5.2 4.9 8.2 6.9
3.9 3.8 5.2 5.4
4.1 4.0 5.6 5.6
5.7 5.1 7.0 6.3
3.0 3.2 4.5 5.0
2.9 3.3 4.5 5.1
3.4 3.3 4.4 5.0
4.0 4.2 5.2 5.4
3.0 3.0 4.6 5.0
4.0 4.1 5.9 5.8
3.0 3.2 4.4 5.1
3.6 3.6 5.3 5.4
3.1 3.2 4.6 5.0
3.2 3.3 5.4 5.3
3.0 3.4 4.2 4.7
3.8 4.0 6.9 6.7 : x
1.0 0.0
1.0 0.0
1.0 0.0
1.0 0.0
1.0 0.0
1.0 0.0
1.0 0.0
1.0 0.0
1.0 0.0
1.0 0.0
1.0 0.0
1.0 0.0
0.0 1.0
0.0 1.0
0.0 1.0
0.0 1.0
0.0 1.0
0.0 1.0
0.0 1.0
0.0 1.0
0.0 1.0
0.0 1.0
0.0 1.0
0.0 1.0
0.0 1.0 : prob
```

10.3 Program Results

```
nag_mv_gaussian_mixture (g03gac) Example Program Results

 Mixing proportions
         1       2
 1   0.4798  0.5202

 Group means
            1          2
 1      4.0041     3.3350
 2      3.9949     3.4434
 3      5.5894     4.9870
 4      5.4432     5.3602

 Pooled Variance-covariance matrix
            1          2          3          4
 1      0.4539     0.2891     0.6075     0.3413
 2      0.2891     0.2048     0.4101     0.2490
 3      0.6075     0.4101     1.0648     0.6011
 4      0.3413     0.2490     0.6011     0.3759

 Densities
              1            2
  1    2.5836e-01   1.1853e-02
  2    3.7065e-07   1.1241e-01
  3    5.3069e-03   1.8080e-06
  4    4.2461e-01   2.8584e-05
  5    5.0387e-02   1.1544e+00
  6    1.1260e+00   7.2224e-02
  7    2.0911e+00   2.1224e-02
  8    5.7856e-03   1.3227e+00
  9    1.1609e+00   2.9411e-02
 10    8.9826e-02   2.4260e-05
 11    3.0170e-01   1.0106e+00
 12    1.2930e+00   3.5422e-01
 13    2.8644e-02   6.7851e-07
 14    2.0759e-02   3.1690e+00
 15    7.6461e-02   1.5231e+00
 16    3.0279e-04   8.4017e-01
 17    5.6101e-01   4.6699e-05
 18    2.6573e-05   6.4442e-01
 19    2.1250e+00   5.1006e-02
 20    8.6822e-04   2.7626e+00
 21    1.9223e-01   2.3971e+00
 22    1.2469e-02   2.8179e+00
 23    1.8389e-02   5.3572e-01
 24    1.2409e+00   9.6489e-03
 25    2.1037e-05   4.8674e-02

 Membership probabilities
              1            2
  1    9.5018e-01   4.9823e-02
  2    3.3259e-06   1.0000e+00
  3    9.9961e-01   3.8664e-04
  4    9.9992e-01   7.9913e-05
  5    3.8999e-02   9.6100e-01
  6    9.3270e-01   6.7295e-02
  7    9.8881e-01   1.1190e-02
  8    4.1252e-03   9.9587e-01
  9    9.7252e-01   2.7479e-02
 10    9.9969e-01   3.0805e-04
 11    2.1722e-01   7.8278e-01
 12    7.6938e-01   2.3062e-01
 13    9.9997e-01   2.6937e-05
 14    6.1133e-03   9.9389e-01
 15    4.4189e-02   9.5581e-01
 16    3.5006e-04   9.9965e-01
 17    9.9990e-01   9.7029e-05
 18    4.0270e-05   9.9996e-01
 19    9.7380e-01   2.6202e-02
 20    3.0204e-04   9.9970e-01
 21    6.9471e-02   9.3053e-01
 22    4.1603e-03   9.9584e-01
 23    3.0839e-02   9.6916e-01
 24    9.9116e-01   8.8421e-03
 25    4.1534e-04   9.9958e-01

No. iterations: 14
Log-likelihood: -29.6831
```