1 Purpose

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

2 Specification

```c
#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,

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

where:

- $x$ is a $p$-dimensional object of interest;
- $w_j$ is the mixture weight for the $j$th group and $\sum_{j=1}^{k} w_j = 1$;
- $\mu_j$ is a $p$-dimensional vector of means for the $j$th group;
- $\Sigma_j$ is the covariance structure for the $j$th group;
- $g(\cdot)$ is the $p$-variate Normal density:

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

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: \textbf{n} \text{- Integer} \quad \textbf{Input}

\textit{On entry:} $n$, the number of objects. There must be more objects than parameters in the model.

\textit{Constraints:}

- if $\text{sopt} = \text{Nag\_GroupCovar}$, $n > \text{ng} \times (\text{nvar} \times \text{nvar} + \text{nvar})$;
- if $\text{sopt} = \text{Nag\_PooledCovar}$, $n > \text{nvar} \times (\text{ng} + \text{nvar})$;
if sopt = Nag_GroupVar, \( n > 2 \times ng \times nvar \);
if sopt = Nag_PooledVar, \( n > nvar \times (ng + 1) \);
if sopt = Nag_OverallVar, \( n > nvar \times ng + 1 \).

2: \( m \) – Integer  
\textit{Input}  
\textit{On entry:} the total number of variables in array \( x \).
\textit{Constraint:} \( m \geq 1 \).

3: \( x[n \times pdx] \) – const double  
\textit{Input}  
\textit{On entry:} \( x[(i - 1) \times pdx + j - 1] \) must contain the value of the \( j \)th variable for the \( i \)th object, for \( i = 1, 2, \ldots, n \) and \( j = 1, 2, \ldots, m \).

4: \( pdx \) – Integer  
\textit{Input}  
\textit{On entry:} the stride separating matrix column elements in the array \( x \).
\textit{Constraint:} \( pdx \geq m \).

5: \( isx[m] \) – const Integer  
\textit{Input}  
\textit{On entry:} if \( nvar = m \) all available variables are included in the model and \( isx \) is not referenced; otherwise the \( j \)th variable will be included in the analysis if \( isx[j - 1] = 1 \) and excluded if \( isx[j - 1] = 0 \), for \( j = 1, 2, \ldots, m \).
\textit{Constraint:} if \( nvar \neq m \), \( isx[j - 1] = 1 \) for \( nvar \) values of \( j \) and \( isx[j - 1] = 0 \) for the remaining \( m - nvar \) values of \( j \), for \( j = 1, 2, \ldots, m \).

6: \( nvar \) – Integer  
\textit{Input}  
\textit{On entry:} \( p \), the number of variables included in the calculations.
\textit{Constraint:} \( 1 \leq nvar \leq m \).

7: \( ng \) – Integer  
\textit{Input}  
\textit{On entry:} \( k \), the number of groups in the mixture model.
\textit{Constraint:} \( ng \geq 1 \).

8: \( popt \) – Nag_Boolean  
\textit{Input}  
\textit{On entry:} if \( popt = \text{Nag}\_\text{TRUE} \), the initial membership probabilities in \( prob \) are set internally; otherwise these probabilities must be supplied.

9: \( prob[n \times tdprob] \) – double  
\textit{Input/Output}  
\textit{On entry:} if \( popt \neq \text{Nag}\_\text{TRUE} \), \( prob[(i - 1) \times tdprob + j - 1] \) is the probability that the \( i \)th object belongs to the \( j \)th group. (These probabilities are normalised internally.)
\textit{On exit:} \( prob[(i - 1) \times tdprob + j - 1] \) is the probability of membership of the \( i \)th object to the \( j \)th group for the fitted model.

10: \( tdprob \) – Integer  
\textit{Input}  
\textit{On entry:} the stride separating matrix column elements in the array \( prob \).
\textit{Constraint:} \( tdprob \geq ng \).

11: \( niter \) – Integer *  
\textit{Input/Output}  
\textit{On entry:} the maximum number of iterations.
\textit{Suggested value:} 15
On exit: the number of completed iterations.

Constraint: \( \text{niter} \geq 1 \).

12: \( \text{riter} \) – Integer

On entry: if \( \text{riter} > 0 \), membership probabilities are rounded to 0.0 or 1.0 after the completion of every \( \text{riter} \) iterations.

Suggested value: 5

13: \( w\) – double

On exit: \( w_j \), the mixing probability for the \( j \)th group.

14: \( g\times n\) – double

On exit: \( g[(i - 1) \times n + j - 1] \) gives the estimated mean of the \( i \)th variable in the \( j \)th group.

15: \( \text{sopt} \) – Nag_VarCovar

On entry: determines the (co)variance structure:

\( \text{sopt} = \text{Nag\_GroupCovar} \)

Groupwise covariance matrices.

\( \text{sopt} = \text{Nag\_PooledCovar} \)

Pooled covariance matrix.

\( \text{sopt} = \text{Nag\_GroupVar} \)

Groupwise variances.

\( \text{sopt} = \text{Nag\_PooledVar} \)

Pooled variances.

\( \text{sopt} = \text{Nag\_OverallVar} \)

Overall variance.

Constraint: \( \text{sopt} = \text{Nag\_GroupCovar}, \text{Nag\_PooledCovar}, \text{Nag\_GroupVar}, \text{Nag\_PooledVar} \) or \( \text{Nag\_OverallVar} \).

16: \( s[\text{dim}] \) – double

Note: the dimension, \( \text{dim} \), of the array \( s \) must be at least \( a \times b \times c \).

Where \( S(i, j, k) \) 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 \( \text{sopt} = \text{Nag\_GroupCovar}, S(i, j, k) \) gives the \( (i, j) \)th element of the \( k \)th group, with \( a = b = n\text{var} \) and \( c = n\).

If \( \text{sopt} = \text{Nag\_PooledCovar}, S(i, j, 1) \) gives the \( (i, j) \)th element of the pooled covariance, with \( a = b = n\text{var} \) and \( c = 1 \).

If \( \text{sopt} = \text{Nag\_GroupVar}, S(j, k, 1) \) gives the \( j \)th variance in the \( k \)th group, with \( a = n\text{var}, b = n\text{g} \) and \( c = 1 \).

If \( \text{sopt} = \text{Nag\_PooledVar}, S(j, 1, 1) \) gives the \( j \)th pooled variance, with \( a = n\text{var} \) and \( b = c = 1 \).

If \( \text{sopt} = \text{Nag\_OverallVar}, S(1, 1, 1) \) gives the overall variance, with \( a = b = c = 1 \).

17: \( f[n \times n] \) – double

On exit: \( f[(i - 1) \times n + j - 1] \) gives the \( p \)-variate Normal (Gaussian) density of the \( i \)th object in the \( j \)th group.
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, \(pdx = \langle value\rangle\) and \(n = \langle value\rangle\).
Constraint: \(pdx \geq n\).
On entry, \(tdprob = \langle value\rangle\) and \(n = \langle value\rangle\).
Constraint: \(tdprob \geq n\).

**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, \(m = \langle value\rangle\).
Constraint: \(m \geq 1\).
On entry, \(ng = \langle value\rangle\).
Constraint: \(ng \geq 1\).
On entry, \(niter = \langle value\rangle\).
Constraint: \(niter \geq 1\).

**NE_INT_2**
On entry, \(nvar = \langle value\rangle\) and \(m = \langle value\rangle\).
Constraint: \(1 \leq nvar \leq m\).

**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 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.
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.

On entry, \( n = \text{value} \) and \( p = \text{value} \).
Constraint: \( n > p \), the number of parameters, i.e., too few objects have been supplied for the model.

On entry, row \( \text{value} \) of supplied prob does not sum to 1.

On entry, \( nvar \neq m \) and isx is invalid.

Not applicable.

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.

None.

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).

/* 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 <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,
     *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;
g03gac

goto END;

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

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

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

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("\n No. 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
10.3 Program Results

nag_mv_gaussian_mixture (g03gac) Example Program Results

Mixing proportions

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4798</td>
<td>0.5202</td>
</tr>
</tbody>
</table>

Group means

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.0041</td>
<td>3.3350</td>
</tr>
<tr>
<td>2</td>
<td>3.9949</td>
<td>3.4434</td>
</tr>
<tr>
<td>3</td>
<td>5.5894</td>
<td>4.9870</td>
</tr>
<tr>
<td>4</td>
<td>5.4432</td>
<td>5.3602</td>
</tr>
</tbody>
</table>

Pooled Variance-covariance matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4539</td>
<td>0.2891</td>
<td>0.6075</td>
<td>0.3413</td>
</tr>
<tr>
<td>2</td>
<td>0.2891</td>
<td>0.4101</td>
<td>0.4101</td>
<td>0.6075</td>
</tr>
<tr>
<td>3</td>
<td>0.6075</td>
<td>1.0648</td>
<td>0.6011</td>
<td>0.2490</td>
</tr>
<tr>
<td>4</td>
<td>0.3413</td>
<td>0.6075</td>
<td>0.2490</td>
<td>0.6011</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Densities</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2.5836e-01</td>
<td>1.1853e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3.7065e-07</td>
<td>1.1241e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5.3069e-03</td>
<td>1.8080e-06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4.2461e-01</td>
<td>2.8584e-05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5.0387e-02</td>
<td>1.1544e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1.1260e+00</td>
<td>7.2224e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2.0911e+00</td>
<td>2.1224e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5.7856e-03</td>
<td>1.3227e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1.1609e+00</td>
<td>2.9411e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8.9826e-02</td>
<td>2.4260e-05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3.0170e+00</td>
<td>1.0106e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1.2930e+00</td>
<td>3.5422e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>2.8644e-02</td>
<td>6.7851e-07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2.0759e-02</td>
<td>3.1690e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>7.6461e-02</td>
<td>1.5231e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>3.0279e-04</td>
<td>8.4017e-06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>5.6101e-01</td>
<td>4.6699e-05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>2.6573e-05</td>
<td>6.4442e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>2.1250e+00</td>
<td>5.1006e-02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>8.6822e-04</td>
<td>2.7626e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1.9223e+01</td>
<td>2.3971e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1.2469e+02</td>
<td>2.8179e+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1.8389e-02</td>
<td>5.3572e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>1.2409e+00</td>
<td>9.6489e-03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>2.1037e-05</td>
<td>4.8674e-02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Membership probabilities

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.5018e-01</td>
<td>4.9823e-02</td>
</tr>
<tr>
<td>2</td>
<td>3.3259e-06</td>
<td>1.0000e+00</td>
</tr>
<tr>
<td>3</td>
<td>9.9961e-01</td>
<td>3.8664e-04</td>
</tr>
<tr>
<td>4</td>
<td>9.9992e-01</td>
<td>7.9913e-05</td>
</tr>
<tr>
<td>5</td>
<td>3.8999e-02</td>
<td>9.6100e-01</td>
</tr>
<tr>
<td>6</td>
<td>9.3270e-01</td>
<td>6.7295e-02</td>
</tr>
<tr>
<td>7</td>
<td>9.8881e-01</td>
<td>1.1190e-02</td>
</tr>
<tr>
<td>8</td>
<td>4.1252e-03</td>
<td>9.9587e-01</td>
</tr>
<tr>
<td>9</td>
<td>9.7252e-01</td>
<td>2.7479e-02</td>
</tr>
<tr>
<td>10</td>
<td>9.9969e-01</td>
<td>3.0805e-04</td>
</tr>
<tr>
<td>11</td>
<td>2.1722e+01</td>
<td>7.8278e-01</td>
</tr>
<tr>
<td>12</td>
<td>7.6938e+01</td>
<td>2.3062e-01</td>
</tr>
<tr>
<td>13</td>
<td>9.9997e-01</td>
<td>2.6937e-05</td>
</tr>
<tr>
<td>14</td>
<td>6.1133e-03</td>
<td>9.9389e-01</td>
</tr>
<tr>
<td>15</td>
<td>4.4189e-02</td>
<td>9.5581e-01</td>
</tr>
<tr>
<td>16</td>
<td>3.5006e-04</td>
<td>9.9965e-01</td>
</tr>
<tr>
<td>17</td>
<td>9.9999e-01</td>
<td>9.7029e-05</td>
</tr>
<tr>
<td>18</td>
<td>4.0270e-05</td>
<td>9.9996e-01</td>
</tr>
<tr>
<td>19</td>
<td>9.7380e-01</td>
<td>2.6202e-02</td>
</tr>
<tr>
<td>20</td>
<td>3.0204e-04</td>
<td>9.9970e-01</td>
</tr>
<tr>
<td>21</td>
<td>6.9471e-02</td>
<td>9.3053e-01</td>
</tr>
<tr>
<td>22</td>
<td>4.1603e-03</td>
<td>9.9584e-01</td>
</tr>
<tr>
<td>23</td>
<td>3.0839e-02</td>
<td>9.6916e-01</td>
</tr>
<tr>
<td>24</td>
<td>9.9116e-01</td>
<td>8.8421e-03</td>
</tr>
</tbody>
</table>

No. iterations: 14
Log-likelihood: -29.6831