1 Purpose

nag_mv_fac_score (g03ccc) computes factor score coefficients from the result of fitting a factor analysis model by maximum likelihood as performed by nag_mv_factor (g03cac).

2 Specification

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

void nag_mv_fac_score (Nag_FacScoreMethod method, Nag_FacRotation rotate,
                     Integer nvar, Integer nfac, const double fl[],
                     Integer tdfl, const double psi[], const double e[],
                     const double r[], Integer tdr,
                     double fs[], Integer tdfs, NagError *fail)
```

3 Description

A factor analysis model aims to account for the covariances among \( p \) variables, observed on \( n \) individuals, in terms of a smaller number, \( k \), of unobserved variables or factors. The values of the factors for an individual are known as factor scores. nag_mv_factor (g03cac) fits the factor analysis model by maximum likelihood and returns the estimated factor loading matrix, \( A \), and the diagonal matrix of variances of the unique components, \( \Psi \). To obtain estimates of the factors, a \( p \) by \( k \) matrix of factor score coefficients, \( \Phi \), is formed. The estimated vector of factor scores, \( \hat{f} \), is then given by:

\[
\hat{f} = x^T \Phi,
\]

where \( x \) is the vector of observed variables for an individual.

There are two commonly used methods of obtaining factor score coefficients. The regression method:

\[
\Phi = \Psi^{-1} A \left( I + A^T \Psi^{-1} A \right)^{-1},
\]

and Bartlett’s method:

\[
\Phi = \Psi^{-1} A \left( A^T \Psi^{-1} A \right)^{-1}.
\]

See Lawley and Maxwell (1971) for details of both methods. In the regression method as given above, it is assumed that the factors are not correlated and have unit variance; this is true for models fitted by nag_mv_factor (g03cac). Further, for models fitted by nag_mv_factor (g03cac),

\[
A^T \Psi^{-1} A = \Theta - I,
\]

where \( \Theta \) is the diagonal matrix of eigenvalues of the matrix \( S^* \), as described in nag_mv_factor (g03cac). The factors may be orthogonally rotated using an orthogonal rotation matrix, \( R \), as computed by nag_mv_orthomax (g03bac). The factor scores for the rotated matrix are then given by \( A \hat{R} \).

4 References

5 Arguments

1: method – Nag FacScoreMethod 
   On entry: indicates which method is to be used to compute the factor score coefficients. 
   method = Nag FacScoreRegsn 
   The regression method is used. 
   method = Nag FacScoreBart 
   Bartlett’s method is used. 
   Constraint: method = Nag FacScoreRegsn or Nag FacScoreBart.

2: rotate – Nag FacRotation 
   On entry: indicates whether a rotation is to be applied. 
   rotate = Nag FacRotate 
   A rotation will be applied to the coefficients and the rotation matrix, R, must be given in r. 
   rotate = Nag FacNoRotate 
   No rotation is applied. 
   Constraint: rotate = Nag FacRotate or Nag FacNoRotate.

3: nvar – Integer 
   On entry: the number of observed variables in the factor analysis, p. 
   Constraint: nvar ≥ nfac.

4: nfac – Integer 
   On entry: the number of factors in the factor analysis, k. 
   Constraint: nfac ≥ 1.

5: fl[nvar × tdf] – const double 
   Note: the (i, j)th element of the matrix is stored in fl[(i - 1) × tdf + j - 1]. 
   On entry: the matrix of unrotated factor loadings, Λ, as returned by nag_mv_factor (g03cac).

6: tdf – Integer 
   On entry: the stride separating matrix column elements in the array fl. 
   Constraint: tdf ≥ nfac.

7: psi[nvar] – const double 
   On entry: the diagonal elements of Ψ, as returned by nag_mv_factor (g03cac). 
   Constraint: psi[i - 1] > 0.0, for i = 1, 2, ..., p.

8: e[nvar] – const double 
   On entry: the eigenvalues of the matrix S*, as returned by nag_mv_factor (g03cac). 
   Constraint: e[i - 1] > 1.0, for i = 1, 2, ..., p.

9: r[nfac × tdr] – const double 
   Note: the (i, j)th element of the matrix R is stored in r[(i - 1) × tdr + j - 1]. 
   On entry: if rotate = Nag FacRotate, then r must contain the orthogonal rotation matrix, R, as returned by nag_mv_orthomax (g03bac). 
   If rotate = Nag FacNoRotate then r need not be set.
10: tdr – Integer

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

Constraint: if rotate = Nag_FacRotate then tdr \geq nfac.

11: fs[nvar \times tdfs] – double

On exit: the matrix of factor score coefficients, \( \Phi \). 
fs[(i - 1) \times tdfs + j - 1] contains the factor score coefficient for the \( j \)th factor and the \( i \)th observed variable, for \( i = 1, 2, \ldots, p \) and \( j = 1, 2, \ldots, k \).

12: tdfs – Integer

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

Constraint: tdfs \geq nfac.

13: fail – NagError *

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

6 Error Indicators and Warnings

NE_2_INT_ARG_ENUM_CONS

On entry, tdr = \langle value \rangle while nfac = \langle value \rangle and rotate = Nag_FacRotate. These arguments must satisfy tdr \geq nfac when rotate = Nag_FacRotate.

NE_2_INT_ARG_LT

On entry, nvar = \langle value \rangle while nfac = \langle value \rangle. These arguments must satisfy nvar \geq nfac.

On entry, tdf = \langle value \rangle while nfac = \langle value \rangle. These arguments must satisfy tdf \geq nfac.

On entry, tdfs = \langle value \rangle while nfac = \langle value \rangle. These arguments must satisfy tdfs \geq nfac.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument method had an illegal value.

On entry, argument rotate had an illegal value.

NE_INT_ARG_LT

On entry, nfac = \langle value \rangle.

Constraint: nfac \geq 1.

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.

NE_REAL_ARRAY_INPUT

On entry, e[\langle value \rangle] = \langle value \rangle.

Constraint: e[\langle value \rangle] > 1.0.

On entry, psi[\langle value \rangle] = \langle value \rangle.

Constraint: psi[\langle value \rangle] > 0.0.
7 Accuracy

Accuracy will depend on the accuracy requested when computing the estimated factor loadings using nag_mv_factor (g03cac).

8 Parallelism and Performance

Not applicable.

9 Further Comments

To compute the factor scores using the factor score coefficients, the values for the observed variables first need to be standardized by subtracting the sample means and, if the factor analysis is based upon a correlation matrix, dividing by the sample standard deviations. This may be performed using nag_mv_z_scores (g03zac). The standardized variables are then post-multiplied by the factor score coefficients. This may be performed using functions from the f16 Chapter Introduction, for example nag_dgemm (f16yac).

If principal component analysis is required, the function nag_mv_prin_comp (g03aac) computes the principal component scores directly. Hence, the factor score coefficients are not needed.

10 Example

The example is taken from Lawley and Maxwell (1971). The correlation matrix for 220 observations on six school subjects is input and a factor analysis model with two factors fitted using nag_mv_factor (g03cac). The factor score coefficients are computed using the regression method.

10.1 Program Text

/* nag_mv_fac_score (g03ccc) Example Program.
 * Copyright 2014 Numerical Algorithms Group.
 */

#include <nag.h>
#include <stdio.h>
#include <string.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagg03.h>
#include <math.h>

#define FL(I, J) fl[(I) *tdfl + J]
#define FS(I, J) fs[(I) *tdfs + J]
#define R(I, J) r[(I) *tdr + J]
#define X(I, J) x[(I) *tdx + J]

int main(void)
{
  Integer exit_status = 0, i, *isx = 0, j, m, n, nfac, nvar, tdfl,
                      tdfs, tdr;
  Integer tdx;
  NagError fail;
  Nag_E04_Opt options;
  Nag_FacMat matrix;
  Nag_FacScoreMethod method;
  Nag_Boolean weight;
  char nag_enum_arg[40];
  double *com = 0, *e = 0, eps, *fl = 0, *fs = 0, *psi = 0, *r = 0;
  double *stat = 0, *wt = 0, *wtptr = 0, *x = 0;
INIT_FAIL(fail);

printf("nag_mv_fac_score (g03ccc) Example Program Results\n\n");

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

#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif

/* nag_enum_name_to_value (x04nac).
 * Converts NAG enum member name to value
 */
matrix = (Nag_FacMat) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif

#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("%\NAG_IFMT%",&nfac);
#else
  scanf("%\NAG_IFMT%",&nfac);
#endif

if (nvar >= 2 && m >= nvar && m > nvar && nvar >= nfac)
{
  if (!((com = NAG_ALLOC(nvar, double)) │
     (e = NAG_ALLOC(nvar, double)) │
     (fl = NAG_ALLOC(nvar*nfac, double)) │
     (fs = NAG_ALLOC(nvar*nfac, double)) │
     (psi = NAG_ALLOC(nvar, double)) │
     (r = NAG_ALLOC(m*m, double)) │
     (stat = NAG_ALLOC(4, double)) │
     (wt = NAG_ALLOC(n, double)) │
     (x = NAG_ALLOC((matrix == Nag_MatCorr_Covar?m:n)*m, double)) │
     (isx = NAG_ALLOC(m, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  tdf1 = nfac;
  tdfs = nfac;
  tdr = m;
  tdx = m;
}
else
{

Mark 25
printf("Invalid nvar or m or n./n");
exit_status = 1;
return exit_status;
}
if (matrix == Nag_MatCorr_Covar)
{
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < m; ++j)
            #ifdef _WIN32
                scanf_s("%lf", &X(i, j));
            #else
                scanf("%lf", &X(i, j));
            #endif
    }
}
else
{
    if (weight)
    {
        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
    #ifdef _WIN32
        scanf_s("%lf", &wt[i]);
    #else
        scanf("%lf", &wt[i]);
    #endif
        wtptr = wt;
    }
    else
    {
        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
    #ifdef _WIN32
        scanf_s("%"NAG_IFMT"", &isx[j]);
    #else
        scanf("%"NAG_IFMT"", &isx[j]);
    #endif
        }
    }
}
for (j = 0; j < m; ++j)
    #ifdef _WIN32
        scanf_s("%"NAG_IPMT"",&isx[j]);
    #else
        scanf("%"NAG_IPMT", &isx[j]);
    #endif
/* nag_opt_init (e04xic).
 * Initialization function for option setting
 */
    nag_opt_init(&options);
    options.max_iter = 500;
    options.optim_tol = 1e-3;
    eps = 1e-5;
/* nag_mv_factor (g03cac).
 * Maximum likelihood estimates of parameters
 */
    fflush(stdout);
    nag_mv_factor(matrix, n, m, x, tdx, nvar, isx, nfac, wtptr, e, 
        stat, com, psi, r, fl, tdfl, &options, eps, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("An error occurred while estimating the factor parameters./n");
        exit_status = 1;
        return exit_status;
    }
printf("Error from nag_mv_factor (g03cac).\n%s\n", fail.message);
exit_status = 1;
goto END;
}
printf("\nLoadings, Communalities and PSI\n\n");
for (i = 0; i < nvar; ++i)
{
    for (j = 0; j < nfac; ++j)
    {
        printf(" %8.3f", FL(i, j));
        printf("%8.3f%8.3f\n", com[i], psi[i]);
    }
#endif
scanf_s("%39s", nag_enum_arg, _countof(nag_enum_arg));
#else
scanf("%39s", nag_enum_arg);
#endif
method = (Nag_FacScoreMethod) nag_enum_name_to_value(nag_enum_arg);
/* nag_mv_fac_score (g03ccc).
 * Factor score coefficients, following nag_mv_factor
 * (g03cac)
 */
nag_mv_fac_score(method, Nag_FacNoRotate, nvar, nfac, fl, tdfl, psi, e,
r, tdr, fs, tdfs, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_mv_fac_score (g03ccc).\n%s\n", fail.message);
    exit_status = 1;
goto END;
}
printf("\nFactor score coefficients\n\n");
for (i = 0; i < nvar; ++i)
{
    for (j = 0; j < nfac; ++j)
    {
        printf(" %8.3f", FS(i, j));
        printf("\n");
    }
}
END:
NAG_FREE(com);
NAG_FREE(e);
NAG_FREE(fl);
NAG_FREE(fs);
NAG_FREE(psi);
NAG_FREE(r);
NAG_FREE(stat);
NAG_FREE(wt);
NAG_FREE(x);
NAG_FREE(isx);
return exit_status;

10.2 Program Data

nag_mv_fac_score (g03ccc) Example Program Data
Nag_MatCorr_Covar Nag_FALSE 220 6 6 2
1.000 0.439 0.410 0.288 0.329 0.248
0.439 1.000 0.351 0.354 0.320 0.329
0.410 0.351 1.000 0.164 0.190 0.181
0.288 0.354 0.164 1.000 0.595 0.470
0.329 0.320 0.190 0.595 1.000 0.464
0.248 0.329 0.181 0.470 0.464 1.000
1 1 1 1 1 1
Nag_FacScoreRegsn
10.3 Program Results

nag_mv_fac_score (g03ccc) Example Program Results

Parameters to e04lbc
---------------------

Number of variables........... 6

optim_tol.................. 1.00e-03  linesearch_tol......... 9.00e-01
step_max................... 1.47e+01  max_iter................ 500
print_level.............. Nag_Soln_Iter  machine precision....... 1.11e-16
deriv_check............. Nag_FALSE  outfile............... stdout

Memory allocation:
state................... User  hesl.................... User
hesd................... User

Iterations performed = 0,  function evaluations = 1
Criterion = 2.999971e-02

<table>
<thead>
<tr>
<th>Variable</th>
<th>Standardized Communalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4168</td>
</tr>
<tr>
<td>2</td>
<td>0.4138</td>
</tr>
<tr>
<td>3</td>
<td>0.3384</td>
</tr>
<tr>
<td>4</td>
<td>0.5164</td>
</tr>
<tr>
<td>5</td>
<td>0.5148</td>
</tr>
<tr>
<td>6</td>
<td>0.4127</td>
</tr>
</tbody>
</table>

Iterations performed = 1,  function evaluations = 2
Criterion = 1.579256e-02

<table>
<thead>
<tr>
<th>Variable</th>
<th>Standardized Communalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4929</td>
</tr>
<tr>
<td>2</td>
<td>0.4050</td>
</tr>
<tr>
<td>3</td>
<td>0.3664</td>
</tr>
<tr>
<td>4</td>
<td>0.6586</td>
</tr>
<tr>
<td>5</td>
<td>0.6077</td>
</tr>
<tr>
<td>6</td>
<td>0.3580</td>
</tr>
</tbody>
</table>

Iterations performed = 2,  function evaluations = 3
Criterion = 1.099067e-02

<table>
<thead>
<tr>
<th>Variable</th>
<th>Standardized Communalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4896</td>
</tr>
<tr>
<td>2</td>
<td>0.4059</td>
</tr>
<tr>
<td>3</td>
<td>0.3566</td>
</tr>
<tr>
<td>4</td>
<td>0.6277</td>
</tr>
<tr>
<td>5</td>
<td>0.5760</td>
</tr>
<tr>
<td>6</td>
<td>0.3700</td>
</tr>
</tbody>
</table>

Iterations performed = 3,  function evaluations = 4
Criterion = 1.086731e-02

<table>
<thead>
<tr>
<th>Variable</th>
<th>Standardized Communalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4898</td>
</tr>
<tr>
<td>2</td>
<td>0.4059</td>
</tr>
<tr>
<td>3</td>
<td>0.3563</td>
</tr>
<tr>
<td>4</td>
<td>0.6228</td>
</tr>
<tr>
<td>5</td>
<td>0.5688</td>
</tr>
<tr>
<td>6</td>
<td>0.3717</td>
</tr>
</tbody>
</table>

Iterations performed = 4,  function evaluations = 5
Criterion = 1.086720e-02
<table>
<thead>
<tr>
<th>Variable</th>
<th>Standardized Communalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4898</td>
</tr>
<tr>
<td>2</td>
<td>0.4059</td>
</tr>
<tr>
<td>3</td>
<td>0.3563</td>
</tr>
<tr>
<td>4</td>
<td>0.6226</td>
</tr>
<tr>
<td>5</td>
<td>0.5686</td>
</tr>
<tr>
<td>6</td>
<td>0.3718</td>
</tr>
</tbody>
</table>

Loadings, Communalities and PSI

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.553</td>
<td>-0.429</td>
<td>0.490</td>
<td>0.510</td>
</tr>
<tr>
<td>0.568</td>
<td>-0.288</td>
<td>0.406</td>
<td>0.594</td>
</tr>
<tr>
<td>0.392</td>
<td>-0.450</td>
<td>0.356</td>
<td>0.644</td>
</tr>
<tr>
<td>0.740</td>
<td>0.273</td>
<td>0.623</td>
<td>0.377</td>
</tr>
<tr>
<td>0.724</td>
<td>0.211</td>
<td>0.569</td>
<td>0.431</td>
</tr>
<tr>
<td>0.595</td>
<td>0.132</td>
<td>0.372</td>
<td>0.628</td>
</tr>
</tbody>
</table>

Factor score coefficients

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.193</td>
<td>-0.392</td>
</tr>
<tr>
<td>0.170</td>
<td>-0.226</td>
</tr>
<tr>
<td>0.109</td>
<td>-0.326</td>
</tr>
<tr>
<td>0.349</td>
<td>0.337</td>
</tr>
<tr>
<td>0.299</td>
<td>0.229</td>
</tr>
<tr>
<td>0.169</td>
<td>0.098</td>
</tr>
</tbody>
</table>