```
#include <oxstd.h>
#include <oxfloat.h>
#import <maximize>

class MLFA
{
    decl m_mY, m_iP, m_iQ;

    MLFA(const mY, const iP, const iQ);
    guess(const cN, const iP, const iQ);
    DFM(const vTheta, const cN, const iP, const iQ,
        const avBeta, const avPhi_f, const amPhi_u, const amSig_11, const amSig_22);
    SSR(const cN, const cN_1, const iP, const iQ,
        const vBeta, const vPhi_f, const mPhi_u, const dSig_11, const mSig_22,
        const amF, const amG, const amH_1, const amH_2, const amSig_vv);
    KF(const mY, const mF, const mG, const mH_1, const mH_2, const mSig_vv,
        const avLnl, const avIndex_u);
    loglikelihood(const vTheta, const adLnL, const avScore, const amHess);
    estimate(const mY, const iP, const iQ);
    index_u(const mY, const iP, const iQ, const vTheta);
    smoothing(const mY, const mF, const mG, const mH_1, const mH_2, const mSig_vv);
    index_s(const mY, const iP, const iQ, const vTheta);
}

MLFA::MLFA(const mY, const iP, const iQ)
{
    m_mY = mY;
    m_iP = iP;
    m_iQ = iQ;
}

MLFA::guess(const cN, const iP, const iQ)
{
    decl vbeta = ones(cN - 1, 1);
    decl vtheta = vbeta;
    if (iP > 0)
    {
        decl vphi_f = vec(zeros(1, iP));
        vtheta = vtheta | vphi_f;
    }
    if (iQ > 0)
    {
        decl vphi_u = vec(zeros(cN, iQ));
        vtheta = vtheta | vphi_u;
    }
    decl dsig_11 = .5;
    decl vsig_22 = .5 * ones(cN, 1);
    vtheta = vtheta | dsig_11 | vsig_22;

    return vtheta;
}

MLFA::DFM(const vTheta, const cN, const iP, const iQ,
    const avBeta, const avPhi_f, const amPhi_u, const amSig_11, const amSig_22)
{
    decl cbeta = cN - 1;
    decl cphi_f = iP;
    decl cphi_u = cN * iQ;
    decl cphi = cphi_f + cphi_u;
    decl ctheta = rows(vTheta);

```

```
62      avBeta[0] = 1 | vTheta[0: cbeta - 1];
63      avPhi_f[0] = 0;
64      if (iP > 0)
65      {
66          avPhi_f[0] = vTheta[cbeta: cbeta + cphi_f - 1]';
67      }
68      amPhi_u[0] = 0;
69      if (iQ > 0)
70      {
71          amPhi_u[0] = diag(vTheta[cbeta + cphi_f: cbeta + cphi_f + cN - 1]);
72          if (iQ > 1)
73          {
74              decl i;
75              for (i = 2; i <= iQ; ++i)
76              {
77                  amPhi_u[0] = amPhi_u[0]
78                      ~ diag(vTheta[cbeta + cphi_f + (i - 1) * cN: cbeta + cphi_f + i *
    cN - 1]);
79              }
80          }
81      }
82      amSig_11[0] = fabs(diag(vTheta[cbeta + cphi]));
83      amSig_22[0] = fabs(diag(vTheta[cbeta + cphi + 1: ctheta - 1]));
84  }
85
86  MLFA::SSR(const cN, const cN_1, const iP, const iQ,
87      const vBeta, const vPhi_f, const mPhi_u, const dSig_11, const mSig_22,
88      const amF, const amG, const amH_1, const amH_2, const amSig_vv)
89  {
90      decl cn_2 = cN - cN_1;
91      decl cbeta = cN;
92      decl cbeta_1 = cN_1;
93      decl csta = 5 + 5 * cN;
94
95      decl mf = zeros(csta, csta);
96      mf[1: 4][0: 3] = unit(4);
97      mf[5 + cN: 5 + 5 * cN - 1][5: 5 + 4 * cN - 1] = unit(4 * cN);
98      if (iP > 0)
99      {
100         mf[0][0: iP - 1] = vPhi_f;
101     }
102     if (iQ > 0)
103     {
104         mf[5: 5 + cN - 1][5: 5 + iQ * cN - 1] = mPhi_u;
105     }
106     amF[0] = mf;
107
108     decl mg = zeros(csta, 1 + cN);
109     mg[0][0] = 1;
110     mg[5: 5 + cN - 1][1: 1 + cN - 1] = unit(cN);
111     amG[0] = mg;
112
113     amH_1[0] = 0;
114     if (cN_1 > 0)
115     {
116         decl vbeta1 = vBeta[0: cbeta_1 - 1];
117         decl mh_1 = zeros(cN_1, csta);
118         mh_1[][0] = (1 / 3) * vbeta1;
119         mh_1[][1] = (2 / 3) * vbeta1;
120         mh_1[][2] = vbeta1;
121         mh_1[][3] = (2 / 3) * vbeta1;
```

```
122         mh_1[][4] = (1 / 3) * vbeta1;
123         mh_1[][5: 5 + cN_1 - 1] = (1 / 3) * unit(cN_1);
124         mh_1[][5 + cN: 5 + cN + cN_1 - 1] = (2 / 3) * unit(cN_1);
125         mh_1[][5 + 2 * cN: 5 + 2 * cN + cN_1 - 1] = unit(cN_1);
126         mh_1[][5 + 3 * cN: 5 + 3 * cN + cN_1 - 1] = (2 / 3) * unit(cN_1);
127         mh_1[][5 + 4 * cN: 5 + 4 * cN + cN_1 - 1] = (1 / 3) * unit(cN_1);
128         amH_1[0] = mh_1;
129     }
130
131     decl vbeta2 = vBeta[cbeta_1: cbeta - 1];
132     decl mh_2 = zeros(cn_2, csta);
133     mh_2[][0] = vbeta2;
134     mh_2[][5 + cN - cn_2: 5 + cN - 1] = unit(cn_2);
135     amH_2[0] = mh_2;
136
137     decl msig_vv = unit(1 + cN);
138     msig_vv[0][0] = dSig_11;
139     msig_vv[1: 1 + cN - 1][1: 1 + cN - 1] = mSig_22;
140     amSig_vv[0] = msig_vv;
141 }
142
143 MLFA::KF(const mY, const mF, const mG, const mH_1, const mH_2, const mSig_vv,
144     const avLnl, const avIndex_u)
145 {
146     decl cn = rows(mY);
147     decl cn_1 = rows(selectr(mY));
148     decl cobs = columns(mY);
149     decl csta = 5 + 5 * cn;
150
151     decl vlnl = zeros(cobs, 1);
152     decl vindex_u = zeros(cobs, 1);
153     decl vs_u = zeros(csta, 1);
154 //  decl mp_u = shape(invert(unit(csta ^ 2) - mF ** mF) * vec(mG * mSig_vv * mG'), cst
    ▹a, csta);
155     decl mp_u = zeros(csta, csta);  // approximate ML estimator
156     decl t;
157     for (t = 0; t < cobs; ++t)
158     {
159         // Prediction
160
161         decl vs_p = mF * vs_u;
162         decl mp_p = mF * mp_u * mF' + mG * mSig_vv * mG';
163
164         // Log-likelihood
165
166         decl vy = mY[][t];
167         decl mh = mH_2;
168         decl msig_ww = zeros(cn, cn);
169         if (cn_1 > 0)
170         {
171             mh = mH_1 | mH_2;
172             if (isnan(vy))
173             {
174                 vy[0: cn_1 - 1] = zeros(cn_1, 1);
175                 mh = zeros(mH_1) | mH_2;
176                 msig_ww[0: cn_1 - 1][0: cn_1 - 1] = unit(cn_1);
177             }
178         }
179         decl ve = vy - mh * vs_p;
180         decl msig_ee = mh * mp_p * mh' + msig_ww;
181         vlnl[t] = - (cn / 2) * log(2 * M_PI)
```

```
182             - (1 / 2) * log(determinant(msig_ee))
183             - (1 / 2) * ve' * invertsym(msig_ee) * ve;
184 
185         // Updating
186 
187         decl mgain = mp_p * mh' * invertsym(mh * mp_p * mh' + msig_ww);
188         vs_u = vs_p + mgain * (vy - mh * vs_p);
189         mp_u = mp_p - mgain * mh * mp_p;
190         vindex_u[t] = vs_u[0];
191     }
192     avLnl[0] = vlnl;
193     avIndex_u[0] = vindex_u;
194 }
195 
196 MLFA::loglikelihood(const vTheta, const adLnL, const avScore, const amHess)
197 {
198     decl my = m_mY;
199     decl cn = rows(my);
200     decl cn_1 = rows(selectr(my));
201     decl ip = m_iP;
202     decl iq = m_iQ;
203 
204     decl vbeta, vphi_f, mphi_u, dsig_11, msig_22;
205     DFM(vTheta, cn, ip, iq,
206         &vbeta, &vphi_f, &mphi_u, &dsig_11, &msig_22);
207 
208     decl mf, mg, mh_1, mh_2, msig_vv;
209     SSR(cn, cn_1, ip, iq, vbeta, vphi_f, mphi_u, dsig_11, msig_22,
210         &mf, &mg, &mh_1, &mh_2, &msig_vv);
211 
212     decl vlnl, vindex_u;
213     KF(my, mf, mg, mh_1, mh_2, msig_vv,
214         &vlnl, &vindex_u);
215     adLnL[0] = meanc(vlnl);
216 
217     return 1;
218 }
219 
220 MLFA::estimate(const mY, const iP, const iQ)
221 {
222     decl cn = rows(mY);
223     decl cobs = columns(mY);
224 
225     decl vtheta = guess(cn, iP, iQ);
226     decl dfunc;
227     MaxControl(-1, 1);
228     MaxBFGS(loglikelihood, &vtheta, &dfunc, 0, 1);
229 //  decl mHess;
230 //  Num2Derivative(loglikelihood, vtheta, &mHess);
231 //  print(sqrt(diagonal(invertsym(-cobs * mHess)))');
232 
233     decl dlnL = cobs * dfunc;
234     decl cpar = cn + iP + cn * iQ + cn;
235     decl dAIC = (dlnL - cpar) / cobs;
236     decl dBIC = (dlnL - cpar * log(cobs) / 2) / cobs;
237     print(iP, iQ, dlnL, dAIC, dBIC);
238 
239     return vtheta;
240 }
241 
242 MLFA::index_u(const mY, const iP, const iQ, const vTheta)
```

```
243 {
244     decl cn = rows(mY);
245     decl cn_1 = rows(selectr(mY));
246 
247     decl vbeta, vphi_f, mphi_u, dsig_11, msig_22;
248     DFM(vTheta, cn, iP, iQ,
249         &vbeta, &vphi_f, &mphi_u, &dsig_11, &msig_22);
250 
251     decl mf, mg, mh_1, mh_2, msig_vv;
252     SSR(cn, cn_1, iP, iQ, vbeta, vphi_f, mphi_u, dsig_11, msig_22,
253         &mf, &mg, &mh_1, &mh_2, &msig_vv);
254 
255     decl vlnl, vindex_u;
256     KF(mY, mf, mg, mh_1, mh_2, msig_vv,
257         &vlnl, &vindex_u);
258 
259     return vindex_u;
260 }
261 
262 MLFA::smoothing(const mY, const mF, const mG, const mH_1, const mH_2, const mSig_vv)
263 {
264     decl cn = rows(mY);
265     decl cn_1 = rows(selectr(mY));
266     decl cobs = columns(mY);
267     decl csta = 5 + 5 * cn;
268 
269     decl vs_u = zeros(csta, 1);
270     decl mp_u = zeros(csta, csta);
271     decl ms_p = zeros(csta, cobs);
272     decl mvechP_p = zeros(csta * (csta + 1) / 2, cobs);
273     decl t;
274     for (t = 0; t < cobs; ++t)
275     {
276         // Prediction
277 
278         decl vs_p = mF * vs_u;
279         decl mp_p = mF * mp_u * mF' + mG * mSig_vv * mG';
280         ms_p[][t] = vs_p;
281         mvechP_p[][t] = vech(mp_p);
282 
283         // Updating
284 
285         decl vy = mY[][t];
286         decl mh = mH_2;
287         decl msig_ww = zeros(cn, cn);
288         if (cn_1 > 0)
289         {
290             mh = mH_1 | mH_2;
291             if (isnan(vy))
292             {
293                 vy[0: cn_1 - 1] = zeros(cn_1, 1);
294                 mh = zeros(mH_1) | mH_2;
295                 msig_ww[0: cn_1 - 1][0: cn_1 - 1] = unit(cn_1);
296             }
297         }
298         decl mgain = mp_p * mh' * invertsym(mh * mp_p * mh' + msig_ww);
299         vs_u = vs_p + mgain * (vy - mh * vs_p);
300         mp_u = mp_p - mgain * mh * mp_p;
301     }
302 
303     // Smoothing
```

```
304
305     decl ms_s = zeros(csta, cobs);
306     decl vr = zeros(csta, 1);
307     for (t = cobs - 1; t >= 0; --t)
308     {
309         decl vs_p = ms_p[][t];
310         decl mp_p = unvech(mvechP_p[][t]);
311
312         decl vy = mY[][t];
313         decl mh = mH_2;
314         decl msig_ww = zeros(cn, cn);
315         if (cn_1 > 0)
316         {
317             mh = mH_1 | mH_2;
318             if (isnan(vy))
319             {
320                 vy[0: cn_1 - 1] = zeros(cn_1, 1);
321                 mh = zeros(mH_1) | mH_2;
322                 msig_ww[0: cn_1 - 1][0: cn_1 - 1] = unit(cn_1);
323             }
324         }
325         decl mgain = mp_p * mh' * invertsym(mh * mp_p * mh' + msig_ww);
326         vr = mh' * invertsym(mh * mp_p * mh' + msig_ww) * (vy - mh * vs_p)
327             + (unit(csta) - mh' * mgain') * mF' * vr;
328         ms_s[][t] = vs_p + mp_p * vr;
329     }
330
331     return ms_s;
332 }
333
334 MLFA::index_s(const mY, const iP, const iQ, const vTheta)
335 {
336     decl cn = rows(mY);
337     decl cn_1 = rows(selectr(mY));
338
339     decl vbeta, vphi_f, mphi_u, dsig_11, msig_22;
340     DFM(vTheta, cn, iP, iQ,
341         &vbeta, &vphi_f, &mphi_u, &dsig_11, &msig_22);
342
343     decl mf, mg, mh_1, mh_2, msig_vv;
344     SSR(cn, cn_1, iP, iQ, vbeta, vphi_f, mphi_u, dsig_11, msig_22,
345         &mf, &mg, &mh_1, &mh_2, &msig_vv);
346
347     decl ms_s = smoothing(mY, mf, mg, mh_1, mh_2, msig_vv);
348
349     return ms_s[0][]';
350 }
351
352 main()
353 {
354     decl time = timer();
355
356     decl mY = loadmat("F:/MixedData/MM03/mm-data/BCIQ1M42.xlsx")';
357     decl iP = 2;
358     decl iQ = 2;
359
360     decl mlfaobj = new MLFA(mY, iP, iQ);
361     decl vtheta = mlfaobj.estimate(mY, iP, iQ);
362     decl vindex_u = mlfaobj.index_u(mY, iP, iQ, vtheta);
363     decl vindex_s = mlfaobj.index_s(mY, iP, iQ, vtheta);
364     savemat("E:/MixedData/MM03/temp.xlsx", vindex_u ~ vindex_s);
```

```
365     delete mlfaobj;
366
367     print(timespan(time, timer()));
368 }
369
```