Statistics 5401
22. Factor Extraction

Gary W. Oehlert
School of Statistics
313B Ford Hall
612-625-1557
gary@stat.umn.edu

Recall the factor analysis model. Let x be a p-dimensional random variable with mean μ and variance Σ.
The orthogonal factor model assumes that we can write

$$\begin{aligned} x &= \mu + \mathbf{L}f + \epsilon \\ \Sigma &= \mathbf{V} + \Psi \\ &= \mathbf{L}\mathbf{L}' + \Psi \end{aligned}$$

where $\mathbf{V}$ has rank $m < p$, Ψ is diagonal with nonnegative diagonal elements, and $\mathbf{L}$ is $p \times m$.
The *factor extraction* step consists of finding estimates of $\mathbf{V}$ and Ψ, along with a preliminary estimate of $\mathbf{L}$.
We will discuss a handfull of methods for estimating $\mathbf{V}$, Ψ, and $\mathbf{L}$. The easy method isn't very good, and the good methods aren't very easy; such is life in the fast lane.
Remember, we can say relatively little about $\mathbf{L}$ and f without some conventions. For example:
• Changes in the mean and variance of f can be accomodated by corresponding changes in μ and $\mathbf{L}$.
• There are infinitely many equivalent $\mathbf{L}$s.
What should we assume? The orthogonal factor model assumes

$$E(f) = 0 \quad \text{and} \quad Var(f) = \mathbf{I}$$

These seem relatively safe and innocuous, but they're not enough.
In addition, a purely convenience assumption such as

$$\mathbf{L}'\mathbf{L} = \text{diagonal} \quad \text{or} \quad \mathbf{L}'\Psi^{-1}\mathbf{L} = \text{diagonal}$$

is made.
With the additional assumption that the eigenvalues of $\mathbf{V}$ are all distinct, then we've got $\mathbf{L}$ nailed down pretty well.
(Though not completely, even with these assumptions the order and signs of the columns of $\mathbf{L}$ are arbitrary.)
If $\mathbf{V}$ has any repeated eigenvalues, further assumptions are needed.
Let's do some basic accounting.
$\mathbf{S}$ has $1 + 2 + \ldots + p = p(p+1)/2$ components.
There are p uniquenesses.
$\mathbf{L}$ has mp parameters, but it has to satisfy $\mathbf{L}'\mathbf{L} =$ diagonal or equivalent, so there are only

$$p + \ldots + p - (m-1) = \frac{p(p+1)}{2} - \frac{(p-m)(p-m+1)}{2}$$

free parameters.
Thus the excess of components in $\mathbf{S}$ over parameters in our model is

$$\frac{p(p+1)}{2} - p - (\frac{p(p+1)}{2} - \frac{(p-m)(p-m+1)}{2})$$

$$= \frac{(p-m)(p-m+1)-2p}{2} = \frac{(p-m)^2-p-m}{2}$$

We shouldn't be fitting models that have more parameters than things to fit.

Excess

	m				
p	1	2	3	4	5
3	0	-2			
4	2	-1	-3		
5	5	1	-2	-4	
6	9	4	0	-3	-5
7	14	8	3	-1	-4
8	20	13	7	2	-2
9	27	19	12	6	1
10	35	26	18	11	5

Note that we can get an exact fit (using zero uniquenesses) when $m = p$.
If the correlations are all small, we can also get an exact fit using $m = p - 1$ and nonzero uniquenesses.
There are several common methods for extracting factors:

- Principal components (PC)
- Iterated principal factors (IPF)
- Unweighted least squares (ULS)
- Generalized least squares (GLS)
- Maximum likelihood (ML), which assumes multivariate normality

As might be expected, there are some close relationships between the methods.
ULS, GLS, and ML all seek to optimize a goodness of fit criterion, but each one has a slightly different criterion.
IPF is a specific algorithm to minimize the ULS criterion.
PC is the first step in IPF.
PC is easy, but it doesn't even *really* estimate $\mathbf{V}$ (in the sense of homing in to $\mathbf{V}$ for large samples).
The others are more difficult, because they involve iterative algorithms, but they will home in on $\mathbf{V}$ for large samples.
Recall that we can go back and forth between factor models for standardized and unstandardized data. Let Δ be the matrix of reciprocal standard deviations as before. Then the $\mathbf{L}$ for standardized data is just Δ times the $\mathbf{L}$ for unstandardized data. Pre- and post-multiply by Δ for transforming Ψ.
The PC, IPF, and ULS methods do NOT produce $\mathbf{L}$s that obey these relationships.
NOTE: there are many potential algorithms for ULS, GLS, and ML. We will present simple algorithms related to IPF.
These algorithms are NOT efficient. They are presented here because they are simple and easy to implement in MacAnova.
Commercial software (e.g. SAS or SPSS) will use other, more efficient, algorithms.
OK, here we go. We have the orthogonal factors model:

$$x = \mu + \mathbf{L}f + \epsilon$$

with $E(f) = 0$, Var$(f) = \mathbf{I}$, Var$(\epsilon) = \Psi$, and Cov$(f, \epsilon) = 0$.
$\hat{\mu} = \overline{\mathbf{x}}$, and we are trying to find $\hat{\mathbf{L}}$ and $\hat{\Psi}$.

$\mathbf{S}$ and $\mathbf{R}$ are the sample variance and correlation matrices, respectively.
Principal component factor extraction.
Make the eigenvector/eigenvalue decomposition of $\mathbf{R}$:

$$\mathbf{R} = \mathbf{Q}\Lambda\mathbf{Q}'$$

where Λ is the diagonal matrix of eigenvalues and $\mathbf{Q}$ is the matrix of corresponding eigenvectors.
There are p eigenvalues and their total is p, so they average to 1. One (rather arbitrary) method for choosing m is to choose m to be the number of eigenvalues greater than 1.
Let $\mathbf{Q}_m$ be the first m columns of $\mathbf{Q}$, and let $\mathbf{D}_m$ be a diagonal matrix with diagonal entries the square roots of the first m eigenvalues from Λ.
The principal components estimate of $\mathbf{L}$ is

$$\hat{\mathbf{L}} = \mathbf{Q}_m\mathbf{D}_m$$

That is, we estimate $\mathbf{L}$ by the first m eigenvectors rescaled so that their squared lengths match the eigenvalues of $\mathbf{R}$.
This uses the usual principal components rank-m approximation to the data to make a rank-m approximation of $\mathbf{R}$, which we take as an estimate of $\mathbf{V}$.
So we have

$$\hat{\mathbf{V}} = \hat{\mathbf{L}}\hat{\mathbf{L}}' = \mathbf{Q}_m\mathbf{D}_m^2\mathbf{Q}_m'$$

and

$$\hat{\Psi} = \mathbf{R} - \hat{\mathbf{V}}$$

This last part means that we take $\hat{\Psi}$ to be what ever is needed to get the diagonals of $\mathbf{R}$ and $(\hat{\mathbf{V}} + \hat{\Psi})$ to match.
The principal component factor extraction is fast and easy, but it doesn't really estimate what we want.
Suppose that by some miracle or stupendous sample size, $\mathbf{R}$ exactly equaled $\mathbf{V} + \Psi$. Even in this case, principal component factors would not reproduce $\mathbf{V}$ and Ψ.
Wood stiffness data

```
Cmd> readdata("",x1,x2,x3,x4,d)
Read from file "~/5401/JW5data/T4-3.DAT"
Column 1 saved as REAL vector x1
Column 2 saved as REAL vector x2
Column 3 saved as REAL vector x3
Column 4 saved as REAL vector x4
Column 5 saved as REAL vector d

Cmd> stiff <- hconcat(x1,x2,x3,x4)

Cmd> R <- cor(stiff);R
(1,1)          1   0.91376   0.88593   0.89812
(2,1)    0.91376         1   0.78821    0.7881
(3,1)    0.88593   0.78821         1    0.9231
(4,1)    0.89812    0.7881    0.9231         1

Cmd> eigen(R)
```

```
component: values
(1)    3.6002   0.26765  0.079069  0.053115
component: vectors
(1,1)   0.51377  -0.20607  -0.23961    0.7976
(2,1)   0.48416  -0.73169    0.1463  -0.45696
(3,1)   0.49993   0.46577   0.72995  0.017593
(4,1)   0.50169   0.45302  -0.62319  -0.39334

Cmd> L <- sqrt(3.6002)*eigen(R)$vectors[,1]

Cmd> V <- L%*%L'; V
(1,1)   0.95031   0.89555   0.92471   0.92797
(2,1)   0.89555   0.84393   0.87142   0.87449
(3,1)   0.92471   0.87142    0.8998   0.90297
(4,1)   0.92797   0.87449   0.90297   0.90615

Cmd> Psi <- dmat(diag(R)-diag(V))

Cmd> V+Psi
(1,1)         1   0.89555   0.92471   0.92797
(2,1)   0.89555         1   0.87142   0.87449
(3,1)   0.92471   0.87142         1   0.90297
(4,1)   0.92797   0.87449   0.90297         1

Cmd> R
(1,1)         1   0.91376   0.88593   0.89812
(2,1)   0.91376         1   0.78821    0.7881
(3,1)   0.88593   0.78821         1    0.9231
(4,1)   0.89812    0.7881    0.9231         1

Cmd> readdata("",x1,x2,x3,x4,gender)
Read from file "~/5401/JW5data/T6-12.DAT"
Column 1 saved as REAL vector x1
Column 2 saved as REAL vector x2
Column 3 saved as REAL vector x3
Column 4 saved as REAL vector x4
Column 5 saved as factor gender

Cmd> o2 <- hconcat(x1,x2,x3,x4)

Cmd> R <- cor(o2)

Cmd> eigenvals(R)
(1)    2.4644    1.2401   0.29153   0.00392
```

```
Cmd> out <- eigen(R)

Cmd> Qm <- out$vectors[,run(2)]

Cmd> Dm <- dmat(sqrt(out$values[run(2)]))

Cmd> L <- Qm %*% Dm
```

Should we really be fitting 2 factors for 4 components?

```
Cmd> V <- L%*%L';V
(1,1)   0.92841   0.87101   0.47346   0.48064
(2,1)   0.87101   0.95146   0.13683   0.15981
(3,1)   0.47346   0.13683   0.94482   0.91133
(4,1)   0.48064   0.15981   0.91133   0.87985

Cmd> Psi <- dmat(diag(R)-diag(V))

Cmd> R - (V+Psi)
(1,1)         0 -0.058942  0.060388 -0.090948
(2,1) -0.058942         0 -0.049591  0.074748
(3,1)  0.060388 -0.049591         0 -0.081138
(4,1) -0.090948  0.074748 -0.081138         0

Cmd> diag(V) # communalities
(1)   0.92841   0.95146   0.94482   0.87985

Cmd> 1-diag(V) # uniquenesses
(1)  0.071586  0.048536  0.055183   0.12015
```

Unweighted Least Squares factor extraction.
The ULS method tries to minimize

$$U(\hat{\mathbf{L}}, \hat{\Psi}) = ||\mathbf{S} - \hat{\Sigma}||^2 = \text{tr}[(\mathbf{S} - \hat{\Sigma})^2]$$

where $\hat{\Sigma} = \hat{\mathbf{V}} + \hat{\Psi}$.
(Recall that the (squared) norm of a matrix $\mathbf{A}$ is $||\mathbf{A}||^2 = \text{trace}(\mathbf{A}'\mathbf{A}) = \sum_{ij} a_{ij}^2$. If $\mathbf{A}$ is square and symmetric, then $||\mathbf{A}||^2 = \text{trace}(\mathbf{A}^2)$.)
The ULS criterion looks fairly sensible, but minimizing it for $\mathbf{S}$ does not lead to the corresponding minimum for $\mathbf{R}$ (unless $\mathbf{S}$ and $\mathbf{R}$ are exactly in factor form).
We will discuss the Iterated Principal Factor method for minimizing U, not be cause it is so great, but it is relatively common and also relatively easy to understand. Variations on the IPF method can be used for the better criteria.
Work with $\mathbf{R}$.
Let $\hat{\mathbf{L}}^{(i)}$ and $\hat{\Psi}^{(i)}$ be our estimates for $\mathbf{L}$ and Ψ at the ith iteration.
With $\hat{\Psi}^{(i)}$ held constant, find $\hat{\mathbf{L}}^{(i+1)}$ as the rank m principal component approximation to $\mathbf{R} - \hat{\Psi}^{(i)}$. (This is possible as long as $\mathbf{R} - \hat{\Psi}^{(i)}$ is nonnegative definite.)

With $\hat{\mathbf{L}}^{(i+1)}$ held constant, find $\hat{\Psi}^{(i+1)}$ so that $\mathbf{R}$ and $\hat{\Sigma}$ match on the diagonal.
Repeat until converged (very slow).
We need starting values. Usual practice is to specify and initial $\hat{\Psi}^{(1)}$ to get the iterations started.
One common choice is motivated as follows. The uniqueness for a component is the variance of the unique part of that component. Thus we might consider the residual variance of x_i regressed (theoretically) on the other components of x. You can show that this variance is the reciprocal of the ith diagonal element of $\mathbf{R}^{-1}$ (or $\mathbf{S}^{-1}$).
A second possibility is the uniqueness from the principal component factors.

```
Cmd> R <- cor(stiff)

Cmd> psis <- 1/diag(solve(R))

Cmd> psis
(1)  0.077308   0.15955   0.13116   0.11546

Cmd> out <- stepuls(R,psis,1)

Cmd> out
component: psi
(1)   0.05916   0.20264    0.1357   0.12173
component: loadings
(1,1)      0.96997
(2,1)      0.89295
(3,1)      0.92968
(4,1)      0.93716
component: crit
(1)    0.021165

Cmd> for(i,run(20)) {
out <- stepuls(R,out,1)
out$crit
}
(1)    0.017931
(1)    0.017265
(1)    0.017114
(1)    0.017078
(1)     0.01707
(1)    0.017068
(1)    0.017067
(1)    0.017067
...

Cmd> out
component: psi
(1)  0.033493   0.23426   0.13595   0.12353
```

```
component: loadings
(1,1)      0.98311
(2,1)      0.87507
(3,1)      0.92954
(4,1)       0.9362
component: crit
(1)     0.017067

Cmd> pcpsi <- vector(.04969,.15607,.1002,.09385)

Cmd> out <- stepuls(R,pcpsi,1)

Cmd> for(i,run(20)) {
out <- stepuls(R,out,1)
out$crit
}
(1)      0.01776
(1)     0.017197
(1)     0.017097
(1)     0.017075
(1)     0.017069
(1)     0.017067
(1)     0.017067
...

Cmd> out
component: psi
(1)  0.033493   0.23426   0.13595   0.12353
component: loadings
(1,1)      0.98311
(2,1)      0.87507
(3,1)      0.92954
(4,1)       0.9362
component: crit
(1)     0.017067

Cmd> R <- cor(o2)

Cmd> psis <- 1/diag(solve(R))

Cmd> psis
(1)   0.01238  0.015924  0.012381  0.028265

Cmd> out <- stepuls(R,psis,2)

Cmd> for(i,run(20)) {
```

```
out <- stepuls(R,out,2)
out$crit
}
(1)      0.04355
(1)     0.033646
(1)     0.029093
(1)      0.02608
ERROR: min(psi) <= 0 in macro stepuls

Cmd> out
component: psi
(1)  0.14675   0.0781  0.00559  0.30942
component: loadings
(1,1)      0.84893      -0.3641
(2,1)      0.65749     -0.69972
(3,1)      0.81365      0.57653
(4,1)      0.71842      0.41768
component: crit
(1)      0.02608

Cmd> psis <- diag(Psi) #try pc psis

Cmd> out <- stepuls(R,psis,2)

Cmd> for(i,run(20)) {
out <- stepuls(R,out,2)
out$crit
}
(1)     0.034972
(1)     0.029866
(1)     0.026654
ERROR: min(psi) <= 0 in macro stepuls

Cmd> S <- tabs(o2,covar:T) # try with S

Cmd> psis <- 1/diag(solve(S))

Cmd> out <- stepuls(S,psis,2)

Cmd> for(i,run(200)) {
out <- stepuls(S,out,2)
out$crit
}
(1)    0.0012892
...
```

```
(1)    0.0012406
(1)    0.0012405    step 157
ERROR: min(psi) <= 0 in macro stepuls

Cmd> out
component: psi
(1) 0.003172 0.00013519  0.21158   2.0056
component: loadings
(1,1)     0.040281    -0.072409
(2,1)      0.33785      -1.3468
(3,1)        0.735      0.10483
(4,1)        8.296     0.045913
component: crit
(1)    0.0012405
```

Generalized Least Squares factor extraction.
The GLS method tries to minimize

$$G(\hat{\mathbf{L}},\hat{\Psi}) = ||\mathbf{S}^{-1}(\mathbf{S}-\hat{\Sigma})||^2 = ||\mathbf{I}-\mathbf{S}^{-1}\hat{\Sigma}||^2 = \text{tr}[(\mathbf{I}-\mathbf{S}^{-1}\hat{\Sigma})^2]$$

where $\hat{\Sigma} = \hat{\mathbf{V}} + \hat{\Psi}$.
GLS is trying to minimize the sum of squared residuals between $\mathbf{S}^{-1}\hat{\Sigma}$ and $\mathbf{I}$, which should be small if $\hat{\Sigma}$ is near $\mathbf{S}$.
One advantage of this formulation is that is respects scaling, as factor models should.
Let $\mathbf{D}$ be any diagonal matrix with non-zero diagonal elements, and let $\hat{\Sigma} = \hat{\mathbf{L}}\hat{\mathbf{L}}' + \hat{\Psi}$. Consider rescaling x to $\mathbf{D}x$.

$$\begin{aligned}
&G_{\mathbf{DSD}}(\mathbf{D}\hat{\mathbf{L}},\mathbf{D}\hat{\Psi}\mathbf{D}) \\
&= ||\mathbf{I}-(\mathbf{DSD})^{-1}\mathbf{D}\hat{\Sigma}\mathbf{D}||^2 \\
&= ||\mathbf{I}-\mathbf{D}^{-1}\mathbf{S}^{-1}\mathbf{D}^{-1}\mathbf{D}\hat{\Sigma}\mathbf{D}||^2 \\
&= ||\mathbf{I}-\mathbf{D}^{-1}\mathbf{S}^{-1}\hat{\Sigma}\mathbf{D}||^2 \\
&= ||\mathbf{D}^{-1}(\mathbf{I}-\mathbf{S}^{-1}\hat{\Sigma})\mathbf{D}||^2 \\
&= \text{tr}[\mathbf{D}^{-1}(\mathbf{I}-\mathbf{S}^{-1}\hat{\Sigma})\mathbf{D}\mathbf{D}^{-1}(\mathbf{I}-\mathbf{S}^{-1}\hat{\Sigma})\mathbf{D}] \\
&= \text{tr}[\mathbf{D}^{-1}(\mathbf{I}-\mathbf{S}^{-1}\hat{\Sigma})^2\mathbf{D}] \\
&= \text{tr}[(\mathbf{I}-\mathbf{S}^{-1}\hat{\Sigma})^2\mathbf{D}\mathbf{D}^{-1}] \\
&= \text{tr}[(\mathbf{I}-\mathbf{S}^{-1}\hat{\Sigma})^2] \\
&= ||\mathbf{I}-\mathbf{S}^{-1}\hat{\Sigma}||^2 \\
&= G_{\mathbf{S}}(\hat{\mathbf{L}},\hat{\Psi})
\end{aligned}$$

Maximum likelihod factor extraction.
The ML method of factor extraction tries to minimize

$$M_{\mathbf{S}}(\hat{\mathbf{L}},\hat{\Psi}) = \text{trace}(\hat{\Sigma}^{-1}\mathbf{S}-\mathbf{I}) - \ln(\det[\hat{\Sigma}^{-1}\mathbf{S}]) \geq 0$$

The M criterion is 2/n times the log likelihood ratio test for testing the null hypothesis that Σ has factor form (with m factors) against a general alternative. This test assumes multivariate normality.
Maximum likelihood estimates $\hat{\mathbf{L}}$ and $\hat{\Psi}$ also respect scaling.
Because the M criterion comes from the likelihood ratio test, we can use it as a goodness of fit for factor models with m factors.

$$(d - (2p + 5)/6 - 2m/3)M \approx \chi^2_h$$

where d is the degrees of freedom in the estimate $\mathbf{S}$ and $h = [(p - m)^2 - p - m]/2$.
In fact, the G criterion can be used the same way.
Both GLS and ML estimates can be found with the same type of iterations found in IPF. Namely, for fixed $\hat{\Psi}$, we optimized over all possible values of $\hat{\mathbf{L}}$. Then for fixed $\hat{\mathbf{L}}$, we optimized over all possible values of $\hat{\Psi}$. The internal details of each optimization is done differ between methods. (For GLS and ML, the optimizations involve the eigenvectors and eigenvalues of $\mathbf{S}$ relative to $\hat{\Psi}$.)
In all cases, better algorithms exist.

```
Cmd> R <- cor(stiff)

Cmd> psis <- 1/diag(solve(R))

Cmd> out <- psis;out
(1)  0.077308   0.15955   0.13116   0.11546

Cmd> for(i,run(20)) {
out <- stepgls(R,out,1)
out$crit
}
(1)       1.3963
(1)      0.47823
(1)      0.47809
(1)      0.47808
(1)      0.47808
...

Cmd> out
component: psi
(1)  0.033067   0.11957  0.096378  0.072421
component: loadings
(1,1)      0.97858
(2,1)      0.90492
(3,1)      0.92847
(4,1)      0.93911
component: crit
(1)      0.47808

Cmd> dim(stiff)
(1)           30             4
```

```
Cmd> (29-(2*4+5)/6-2/3)*.4781
(1)       12.51

Cmd> ((4-1)ˆ2-4-1)/2
(1)           2

Cmd> 1-cumchi(12.51,2)
(1)   0.0019208

Cmd> out <- psis

Cmd> for(i,run(20)) {
out <- stepgls(R,out,2)
out$crit
}
(1)      1.1496
(1)   0.0095908
(1)    0.003773
(1)   0.0017404
(1)  0.00079392
...
(1)  4.6001e-09
```

Perfect fit, but we’re using too many parameters.

```
Cmd> out
component: psi
(1)  0.041172  0.088414   0.10429  0.045986
component: loadings
(1,1)      0.97182        -0.12
(2,1)      0.90139     -0.31476
(3,1)      0.93197      0.16477
(4,1)      0.95144      0.22085
component: crit
(1)  4.6001e-09

Cmd> out <- psis

Cmd> for(i,run(120)) {
out <- stepml(R,out,1)
out$crit
}
(1)     0.55611
(1)     0.52068
```

```
(1)      0.51742
(1)       0.5158
...
(1)      0.49985 iteration 120

Cmd> out
component: psi
(1)   0.02938    0.1674   0.17499   0.15803
component: loadings
(1,1)       0.9852
(2,1)      0.91247
(3,1)       0.9083
(4,1)      0.91759
component: crit
(1)      0.49985

Cmd> out <- psis

Cmd> for(i,run(60)) {
out <- stepml(R,out,2)
out$crit
}
(1)      0.25576
(1)     0.095617
(1)      0.03573
...
(1)   1.6215e-05

Cmd> out
component: psi
(1)  0.041711  0.086825   0.10328  0.047476
component: loadings
(1,1)      0.97203     -0.11597
(2,1)      0.90263     -0.31375
(3,1)      0.93175        0.169
(4,1)      0.95039        0.222
component: crit
(1)   1.6215e-05
```

ML gives us the same thing, but still too many parameters

```
Cmd> readdata("",x1,x2,x3,x4,x5,x6,x7,x8,nation)
Read from file "~/5401/JW5data/T8-6.DAT"
Column 1 saved as REAL vector x1
Column 2 saved as REAL vector x2
Column 3 saved as REAL vector x3
Column 4 saved as REAL vector x4
Column 5 saved as REAL vector x5
Column 6 saved as REAL vector x6
Column 7 saved as REAL vector x7
Column 8 saved as REAL vector x8
Column 9 saved as factor nation

Cmd> X <- hconcat(x1,x2,x3,x4,x5,x6,x7,x8)

Cmd> R <- cor(X)

Cmd> eigenvals(R)
(1)    6.6221   0.87762   0.15932   0.12405
(5)  0.07988   0.067965   0.04642    0.0226

Cmd> psis <- 1/diag(solve(R))

Cmd> out <- psis;out
(1)   0.12274   0.11198   0.15506   0.11632
(5)  0.072837     0.045  0.033244  0.095283

Cmd> for(i,run(20)) {
out <- stepgls(R,out,1)
out$crit
}
(1)       2.6906
(1)       1.3779
(1)       1.3779
(1)       1.3779
(1)       1.3779
...

Cmd> out
component: psi
(1)  0.058952  0.077214   0.13565   0.09084
(5)  0.054871   0.03495  0.014074   0.08095
component: loadings
(1,1)      0.73482
(2,1)      0.79359
(3,1)      0.85758
(4,1)      0.91956
(5,1)      0.96069
(6,1)      0.97062
(7,1)      0.98082
(8,1)      0.92114
```

```
component: crit
(1)       1.3779

Cmd> dim(X)
(1)           55            8

Cmd> (54-(2*8+5)/6-2*1/3)*1.3779
(1)       68.665

Cmd> (49-8-1)/2
(1)           20

Cmd> 1-cumchi(68.6,20)
(1)   3.0778e-07
```

Bad fit; try $m = 2$.

```
Cmd> out <- psis

Cmd> for(i,run(20)) {
out <- stepgls(R,out,2)
out$crit
}
(1)       1.8959
(1)      0.49999
(1)      0.49792
(1)      0.49771
(1)      0.49768
(1)      0.49768
(1)      0.49768
...

Cmd> out
component: psi
(1)   0.04794  0.080735   0.13671   0.09064
(5)  0.055019  0.035101  0.013557  0.081129
component: loadings
(1,1)       0.7412     -0.62803
(2,1)       0.7977     -0.52745
(3,1)      0.86017     -0.33185
(4,1)      0.92027     -0.14743
(5,1)       0.9603     -0.00791
(6,1)      0.96922      0.15677
(7,1)      0.97983      0.15715
(8,1)      0.91898      0.26115
component: crit
(1)      0.49768
```

```
Cmd> (54-(2*8+5)/6-2*2/3)*.4977
(1)        24.47

Cmd> (36-8-2)/2
(1)           13

Cmd> 1-cumchi(24.47,13)
(1)     0.027072
```

Still a bad fit; try $m = 3$ with GLS, but first try ML.

```
Cmd> out <- psis

Cmd> for(i,run(20)) {
out <- stepml(R,out,2)
out$crit
}
(1)      0.49254
(1)      0.38691
...
(1)      0.33273
(1)      0.33273
```

ML finds a better fit.

```
Cmd> out
component: psi
(1)  0.080992  0.075814   0.15143   0.13533
(5)  0.081718  0.033793  0.017957  0.085993
component: loadings
(1,1)      0.73082     -0.62041
(2,1)      0.79165     -0.54542
(3,1)      0.85494     -0.34299
(4,1)      0.91585     -0.16087
(5,1)      0.95793    -0.025686
(6,1)      0.97239      0.14375
(7,1)      0.98062      0.14294
(8,1)      0.92291       0.2495
component: crit
(1)      0.33273

Cmd> (54-(2*8+5)/6-2*2/3)*.3327
(1)       16.358

Cmd> 1-cumchi(16.358,13)
(1)      0.23034
```

So ML finds a better fitting model. Note that the loadings don't appear to be that different between ML and GLS. The first appears to be some kind of average, whereas the second looks like a trend across the variables.
Try GLS with $m = 3$

```
Cmd> for(i,run(20)) {
out <- stepgls(R,out,3)
out$crit
}
(1)       1.7696
(1)      0.16394
...
(1)      0.11541
ERROR: non-positive psi
```

```
Cmd> (54-(2*8+5)/6-2*3/3)*.1154
(1)       5.5969

Cmd> (25-8-3)/2
(1)            7

Cmd> 1-cumchi(5.59,7)
(1)      0.58835
```

A good fit.

```
Cmd> out
component: psi
(1) 0.00094961   0.10423   0.14227  0.074362
(5)   0.051118  0.036605  0.010886  0.083439
component: loadings
(1,1)      0.98067     -0.18915    0.0081194
(2,1)      0.93743     -0.02265     -0.11448
(3,1)      0.89295      0.17557     -0.16277
(4,1)      0.84493       0.3728     -0.26364
(5,1)      0.81307      0.50588     -0.17055
(6,1)      0.75288      0.62881     0.017274
(7,1)      0.76639      0.63168     0.051663
(8,1)      0.66191      0.68559     0.062691
component: crit
(1)      0.11541
```

Second and third components look a lot different from second component in $m = 2$ case.
I tried my hand a making up a new way to do ML factor extraction. It sort of works. (Hey, not too bad for an hours worth of effort.)

```
Cmd> addmacrofile("extractml.mac")

Cmd> R <- cor(stiff)
Cmd> extractml(R,1)
component: L
(1,1)      0.98525
(2,1)      0.91249
(3,1)      0.90825
(4,1)      0.91754
component: psi
(1)  0.029288   0.16735    0.1751   0.15812
component: crit
(1)      0.49985
component: status
(1)            0

Cmd> extractml(R,2)
component: L
(1,1)      0.97548            0
(2,1)      0.93657     -0.23281
(3,1)      0.90908      0.27044
(4,1)      0.92169      0.32037
component: psi
(1)  0.048954  0.067622   0.10216  0.049457
component: crit
(1)   1.4162e-05
component: status
(1)            2

Cmd> R <- cor(o2)

Cmd> extractml(R,1)
component: L
(1,1)      0.51881
(2,1)     0.076495
(3,1)      0.97957
(4,1)      0.80442
component: psi
(1)   0.69975    1.0204  5.896e-5   0.30129
component: crit
(1)       4.1476
component: status
(1)            0

Cmd> extractml(R,2)
component: L
(1,1)      0.87422            0
(2,1)      0.84993      -0.5069
```

```
(3,1)      0.49093      0.82364
(4,1)      0.53638      0.61417
component: psi
(1)   0.11854  6.82e-6   1.484-6   0.32341
component: crit
(1)       2.3324
component: status
(1)            2

Cmd> R <- cor(X)

Cmd> extractml(R,1)
component: L
(1,1)      0.66096
(2,1)      0.72428
(3,1)      0.81072
(4,1)      0.88994
(5,1)      0.94548
(6,1)      0.97467
(7,1)      0.98078
(8,1)      0.93309
component: psi
(1)   0.55537   0.45713   0.32922   0.19615
(5)  0.093479  0.036458  0.024701   0.11777
component: crit
(1)        3.138
component: status
(1)            0

Cmd> extractml(R,2)
component: L
(1,1)      0.94789            0
(2,1)      0.94515     0.095922
(3,1)      0.86345      0.29023
(4,1)      0.79068      0.46755
(5,1)       0.7358      0.59859
(6,1)      0.63691      0.73574
(7,1)      0.64375      0.74112
(8,1)      0.53051      0.78426
component: psi
(1)  0.080408  0.075249   0.15091   0.13538
(5)  0.082031  0.033528  0.017902  0.086044
component: crit
(1)      0.33314
component: status
(1)            0
```

```
Cmd> extractml(R,3)
component: L
(1,1)   0.97373            0            0
(2,1)   0.98005     0.076302            0
(3,1)   0.89604      0.21178     -0.20893
(4,1)    0.8306      0.27585     -0.51123
(5,1)   0.77855      0.49133     -0.34236
(6,1)   0.68619      0.66331      -0.2826
(7,1)   0.69493      0.67458     -0.27561
(8,1)   0.58108      0.70262     -0.31463
component: psi
(1)  0.081505  0.068479   0.14731  0.012654
(5)  0.072638  0.035238  0.014133  0.088055
component: crit
(1)      0.12064
component: status
(1)            2
```

```
extractml MACRO DOLLARS
@S <- $1
@p <- ncols(@S)
@m <- $2
@rand <- keyvalue($K,"randL","logic scalar",default:F)
if(@rand)
@L <- matrix(rnorm(@p*@m),@p)
 else
@tmp <- eigen(@S)
@L <- @tmp$vectors[,run(@m)]*sqrt(@tmp$values[run(@m)]’)

@steps <- matrix(rep(.1,@p*(@m+1)),@p)
if(@m > 1)
for(@j, run(2,@m))
for(@i,run(@j-1))
@L[@i,@j] <- 0
@steps[@i,@j] <- 0



@psi <- 1/diag(solve(@S))
@cfs <- vector(@L,@psi)
@steps <- vector(@steps)
setoptions(maxwhile:100000)
getmacros(mlfactor,silent:T)
@nmout <- neldermead(mlfactor,@cfs,@steps,@S,maxeval:100000,quad:T)
@L <- matrix(@nmout$x[run(@m*@p)],@p)
@psi <- @nmout$x[-run(@m*@p)]
@crit <- @nmout$f
structure(L:@L,psi:@psi,crit:@crit,status:@nmout$status)
%extractml%
```

```
mlfactor MACRO DOLLARS
@cfs <- $1
@S <- $2
@p <- ncols(@S)
@cfs <- matrix(@cfs,@p)
@m <- ncols(@cfs)-1
@L <- @cfs[,run(@m)]
@psi <- @cfs[,-run(@m)]
if(min(@psi) < 0)
return(1e99)

@Sig <- @L%*%@L’ + dmat(@psi)
@SS <- solve(@Sig,@S)
@det <- det(@SS)
@out <- 1e99
if(@det > 0)
@out <- trace(@SS)-@p - log(det(@SS))

@out
%mlfactor%
```