Advanced Confirmatory Factor Analysis with R

James H. Steiger

Psychology 312

Spring 2013

In a previous module, we analyzed an artificial "Athletics Data" set to illustrate several approaches to "confirmatory" factor analysis. One was truly (and rigidly) confirmatory: one starts with a particular strong model, tests it with maximum likelihood, and either rejects or accepts the model based on whether or badness-of-fit is reasonable.

When the number of variables is large, setting up a confirmatory factor analysis in R can be time-consuming. Recently, I developed a library of functions specifically designed to take most of the "busywork" out of both exploratory and confirmatory factor analysis. In this module, we see how to perform a confirmatory factor analysis with the Advanced Factor Functions library.

1 . "Pure" Confirmatory Factor Analysis

In "pure" confirmatory factor analysis, the investigor performs the following:

1. A factor loading matrix with fixed values and free parameters is specified. In traditional versions of "pure" CFA, the researcher designates many of the loadings to have fixed values of zero, and the remaining loadings to be free parameters to be estimated. Likewise, factor intercorrelations may be estimated or fixed at zero.
2. The specified confirmatory factor model is tested using standard structural equation modeling software.
3. If fit of the model is not acceptable, the model is rejected.

Consider the Athletics Data example we examined in conjunction with EFA. In our introductory handout, we tested a "pure confirmatory model" in which there were 3 uncorrelated factors called Endurance, Strength, and Hand-Eye Coordination, and that each factor has non-zero loadings on only 3 variables. This specifies that the factor pattern **F** is of the form

$$\mathbf{F} = \begin{bmatrix} \theta_1 & 0 & 0 \\ \theta_2 & 0 & 0 \\ \theta_3 & 0 & 0 \\ 0 & \theta_4 & 0 \\ 0 & \theta_5 & 0 \\ 0 & \theta_6 & 0 \\ 0 & 0 & \theta_7 \\ 0 & 0 & \theta_8 \\ 0 & 0 & \theta_9 \end{bmatrix}$$

If you were starting from scratch and specifying this model in the R program **`sem,`** it would take a lot of typing to create a model input file like the one below.

```
## Factor 1 – Endurance
Endurance -> X.1500M,  theta01, NA
Endurance -> X.2KROW,  theta02, NA
Endurance -> X.12MINTR,theta03, NA
## Factor 2 – Strength
Strength  -> BENCH,    theta04, NA
Strength  -> CURL,     theta05, NA
Strength  -> MAXPUSHU, theta06, NA
## Factor 3 – Hand-Eye Coordination
Hand-Eye  -> PINBALL,  theta07, NA
Hand-Eye  -> BILLIARD, theta08, NA
Hand-Eye  -> GOLF,     theta09, NA
## Unique Variances
X.1500M   <->  X.1500M,  theta10, NA
X.2KROW   <->  X.2KROW,  theta11, NA
X.12MINTR <->  X.12MINTR, theta12, NA
BENCH     <->  BENCH, theta13, NA
CURL      <->  CURL, theta14, NA
MAXPUSHU  <->  MAXPUSHU, theta15, NA
PINBALL   <->  PINBALL, theta16, NA
BILLIARD  <->  BILLIARD, theta17, NA
GOLF      <->  GOLF, theta18, NA
## Factor Variances fixed at 1
Endurance <->  Endurance, NA, 1
Strength  <->  Strength,  NA, 1
Hand-Eye  <->  Hand-Eye,  NA, 1
```

The **`sem`** module does offer a streamlined command, **`cfa`**, to specify a confirmatory factor model. But even this command requires typing or pasting manifest variable names.

The Advanced Factor Functions library eliminates almost all typing, whether you want a **`cfa`** specification or a full **`sem`** specification of your model.

Assume for now that you want to create the simple model specified above, and fit it to the correlation matrix for the AthleticsData file.

After loading the Hmisc library, loading the AthleticsData file and attaching it with the following 3 lines, we are ready to go.

```
> library(Hmisc)
> AthleticsData <- spss.get(“AthleticsData.sav”)
> attach(AthleticsData)
```

We begin by computing a correlation matrix for analysis.

```
> AD.R <- cor(AthleticsData)
```

After loading the **`sem`** package, we then load in the **`AdvancedFactorFunctions`** library.

```
> library(sem)
> source(“AdvancedFactorFunctions.txt”)
```

To specify and test the *pure* confirmatory factor model, use the **`QuickCFA`** function. You call this with a correlation matrix and a requested number of factors. You can optionally enter a list of factor names, but it is actually easier to simply type them into the pattern editor.

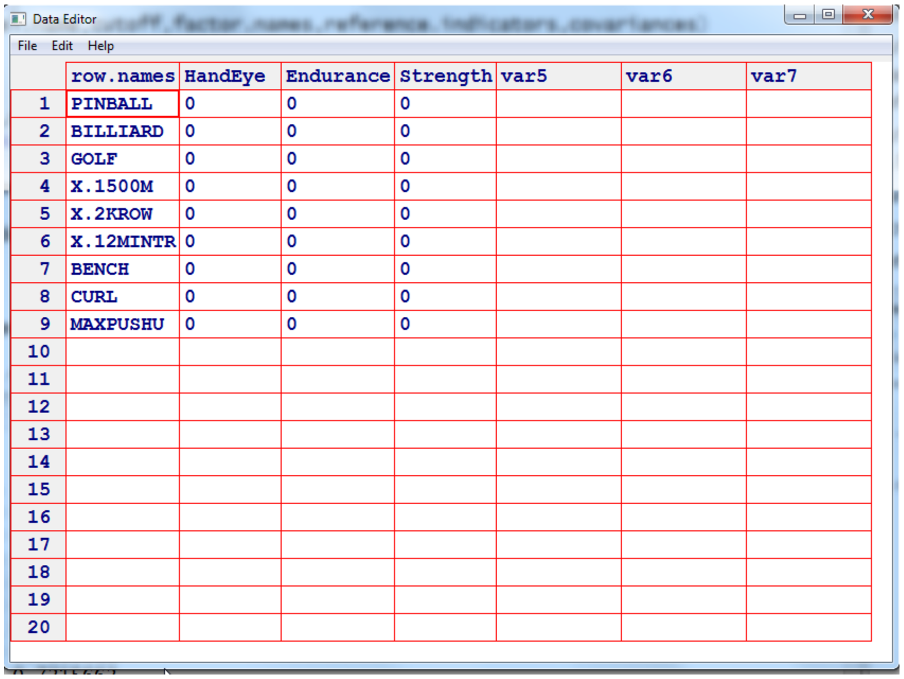
The **`QuickCFA`** function performs two steps:

1. A quick factor pattern editor is opened. The user specifies the factor pattern with 1's indicating free parameters, and 0's indicating fixed values of zero. Then this pattern is converted into a **`sem`** model specification and saved to disk.
2. The model and data are analyzed automatically, using the **`sem`** command. Results are returned as a **`sem`** object.

Each of these individual steps can be processed individually using more “fine-grained” service functions.

```
> pureCFA.fit <- QuickCFA(AD.R,3,1000,"pureCFA",c("Hand-Eye","Endurance","Strength"))
```

A window will open allowing you to edit the pattern.

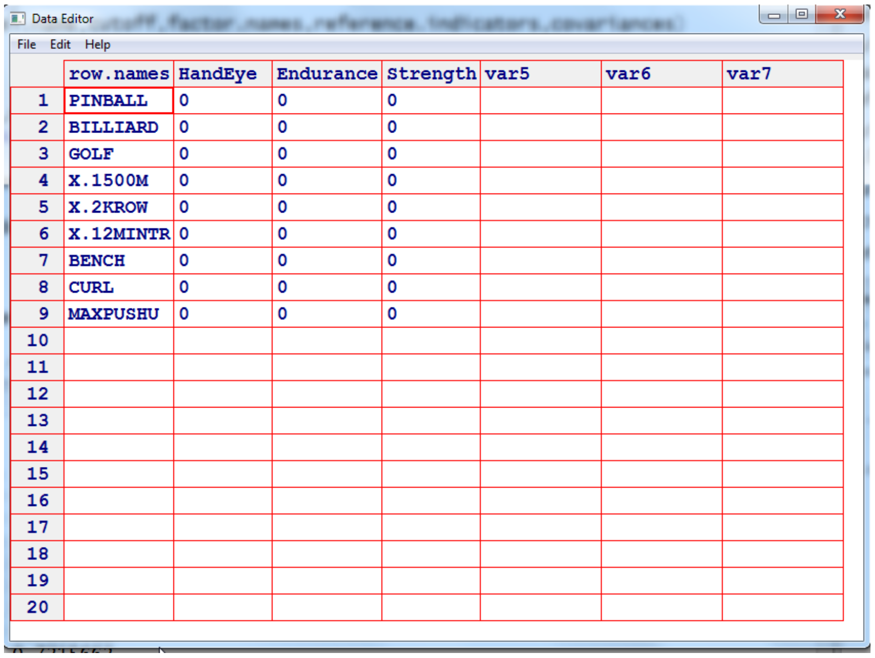


Data Editor

File Edit Help

	row.names	HandEye	Endurance	Strength	var5	var6	var7
1	PINBALL	0	0	0			
2	BILLIARD	0	0	0			
3	GOLF	0	0	0			
4	X.1500M	0	0	0			
5	X.2KROW	0	0	0			
6	X.12MINTR	0	0	0			
7	BENCH	0	0	0			
8	CURL	0	0	0			
9	MAXPUSHU	0	0	0			
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							

Change the zeros into 1's to match the hypothesized pattern. In this case, you should have



Data Editor

File Edit Help

	row.names	HandEye	Endurance	Strength	var5	var6	var7
1	PINBALL	0	0	0			
2	BILLIARD	0	0	0			
3	GOLF	0	0	0			
4	X.1500M	0	0	0			
5	X.2KROW	0	0	0			
6	X.12MINTR	0	0	0			
7	BENCH	0	0	0			
8	CURL	0	0	0			
9	MAXPUSHU	0	0	0			
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							

Close the window, and the model will be analyzed, and the results of the analysis returned. Requesting a summary of the fit produces the following:

```
> summary(pureCFA.fit)


Model Chisquare =  526.3   Df =  27 Pr(>Chisq) = 5.81e-94
 AIC =  562.3
 BIC =  339.8

 Normalized Residuals
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -7.530   0.000   0.823   1.710   2.880  14.000

 R-square for Endogenous Variables
  PINBALL  BILLIARD      GOLF   X.1500M   X.2KROW X.12MINTR     BENCH      CURL  MAXPUSHU
   0.3541    0.5765    0.5533    0.5051    0.2855    0.5639    0.6097    0.5207    0.2345

 Parameter Estimates
                         Estimate Std Error z value Pr(>|z|)
lam[PINBALL:Hand-Eye]     0.5950   0.03355   17.736   2.204e-70 PINBALL <-- HandEye
lam[BILLIARD:Hand-Eye]    0.7593   0.03470   21.885  3.644e-106 BILLIARD <-- HandEye
lam[GOLF:Hand-Eye]        0.7438   0.03458   21.512  1.196e-102 GOLF <-- HandEye
lam[X.1500M:Endurance]    0.7107   0.03740   19.005   1.550e-80 X.1500M <-- Endurance
lam[X.2KROW:Endurance]    0.5344   0.03501   15.263   1.349e-52 X.2KROW <-- Endurance
lam[X.12MINTR:Endurance]  0.7509   0.03801   19.757   7.017e-87 X.12MINTR <-- Endurance
lam[BENCH:Strength]       0.7808   0.03954   19.750   8.018e-87 BENCH <-- Strength
lam[CURL:Strength]        0.7216   0.03848   18.750   1.932e-78 CURL <-- Strength
lam[MAXPUSHU:Strength]    0.4843   0.03489   13.881   8.261e-44 MAXPUSHU <-- Strength
V[PINBALL]                0.6459   0.03553   18.178   7.719e-74 PINBALL <-> PINBALL
V[BILLIARD]               0.4235   0.03863   10.962   5.806e-28 BILLIARD <-> BILLIARD
V[GOLF]                   0.4467   0.03799   11.758   6.395e-32 GOLF <-> GOLF
V[X.1500M]                0.4949   0.04247   11.651   2.257e-31 X.1500M <-> X.1500M
V[X.2KROW]                0.7145   0.03797   18.816   5.556e-79 X.2KROW <-> X.2KROW
V[X.12MINTR]              0.4361   0.04493    9.707   2.819e-22 X.12MINTR <-> X.12MINTR
V[BENCH]                  0.3903   0.04919    7.935   2.111e-15 BENCH <-> BENCH
V[CURL]                   0.4793   0.04475   10.712   8.916e-27 CURL <-> CURL
V[MAXPUSHU]               0.7655   0.03855   19.857   9.556e-88 MAXPUSHU <-> MAXPUSHU

 Iterations =  22
```

Notice how the **sem** program gave the parameters long, involved names to try to describe their function. The **QuickCFA** function allows you to specify either correlated or orthogonal (uncorrelated) factors. In this case, we chose orthogonal factors, which is the default option.

You may find the factor pattern itself easier to read and interpret in matrix form. You can take a quick look at it with the **GetPattern** function. This function extracts the factor pattern in matrix from from the **sem** fit object:

```
> GetPattern(pureCFA.fit)
$F
            HandEye Endurance  Strength
PINBALL   0.5950341 0.0000000 0.0000000
BILLIARD  0.7593054 0.0000000 0.0000000
GOLF      0.7438495 0.0000000 0.0000000
X.1500M   0.0000000 0.7107084 0.0000000
X.2KROW   0.0000000 0.5343523 0.0000000
```

```
X.12MINTR 0.0000000 0.7509451 0.0000000
BENCH     0.0000000 0.0000000 0.7808211
CURL      0.0000000 0.0000000 0.7215662
MAXPUSHU  0.0000000 0.0000000 0.4842856

$Phi
          HandEye Endurance Strength
HandEye         1         0        0
Endurance       0         1        0
Strength        0         0        1
```

This display can be cleaned up using the **`GetPrettyPattern`** function

```
> GetPrettyPattern(pureCFA.fit)

          HandEye Endurance Strength
PINBALL   0.595
BILLIARD  0.759
GOLF      0.744
X.1500M           0.711
X.2KROW           0.534
X.12MINTR         0.751
BENCH                       0.781
CURL                        0.722
MAXPUSHU                    0.484

               HandEye Endurance Strength
SS loadings      1.484     1.355    1.365
Proportion Var   0.165     0.151    0.152
Cumulative Var   0.165     0.315    0.467

Factor Intercorrelations
          HandEye Endurance Strength
HandEye         1         0        0
Endurance       0         1        0
Strength        0         0        1
```

The “pure” confirmatory factor model was rejected by the standard chi-square test. One can calculate an RMSEA confidence interval to guard against a situation in which power is high to reject even small departures from perfect fit.

```
> RMSEA(pureCFA.fit)
$Lower.Limit
[1] 0.1260335

$Point.Estimate
[1] 0.1360501

$Upper.Limit
[1] 0.1463237

$Confidence.Level
[1] 0.9
```

By most common criteria, this model doesn’t fit well enough to be considered adequate.

In the face of such poor fit, many researchers resort to “modification indices” or other devices in order to update the model and obtain hints about where it might be inadequate. Modification indices predict, using derivatives, how much the chi-square fit statistic will improve if a particular path is added.

The **sem** module in R prints out modification indices. Some of them are illegal in a confirmatory model, and some involve freeing up error covariances in the model. The latter leads to correlated errors, which violates Spearman's original intent that factors explain the correlations among the variables (in the partial correlation sense).

A number of writers have warned about misuse of modification indices. There are a number of devices one might use to guard against improper reliance on these indices.

I've created a function, **CheckMod**, to pick out the largest modification index that is "legal." Unless you explicitly specify the option **loadings.only = FALSE,** you will be shown only modification indices corresponding to currently zero loadings.

```
> CheckMod(pureCFA.fit)
$NewPath
[1] "Endurance->MAXPUSHU,AddedTheta19, NA"

$ModIndex
[1] 186.7415
```

To update a **sem** fit object with the current modification index, use the **UseMod** function.

```
> fit2 <- UseMod(pureCFA.fit)
```

A quick check reveals that the fit is not yet what we would like:

```
> RMSEA(fit2)
$Lower.Limit
[1] 0.09436259

$Point.Estimate
[1] 0.1046251

$Upper.Limit
[1] 0.1152201

$Confidence.Level
[1] 0.9
```

We check the next modification index. It shows a potential change of 24 in the chi-square, so we modify the model again.

```
> RMSEA(fit2)
$Lower.Limit
[1] 0.09436259

$Point.Estimate
[1] 0.1046251

$Upper.Limit
[1] 0.1152201
```

```
$Confidence.Level
[1] 0.9

> CheckMod(fit2)
$NewPath
[1] "Strength->X.2KROW,AddedTheta20, NA"

$ModIndex
[1] 168.0183

> fit3 <- UseMod(fit2)
> RMSEA(fit3)
$Lower.Limit
[1] 0.05259059

$Point.Estimate
[1] 0.06337986

$Upper.Limit
[1] 0.07463124

$Confidence.Level
[1] 0.9> fit3 <- UseMod(fit2)
```

Now we have better fit.

```
> GetPrettyPattern(fit3)

          HandEye Endurance Strength
PINBALL   0.595
BILLIARD  0.759
GOLF      0.744
X.1500M           0.772
X.2KROW           0.609     0.423
X.12MINTR         0.692
BENCH                       0.812
CURL                        0.694
MAXPUSHU          0.459     0.564

               HandEye Endurance Strength
SS loadings      1.484     1.657    1.639
Proportion Var   0.165     0.184    0.182
Cumulative Var   0.165     0.349    0.531

Factor Intercorrelations
------------------------
          HandEye Endurance Strength
HandEye         1         0        0
Endurance       0         1        0
Strength        0         0        1
```

2 . Confirmatory Factor Analysis from an Exploratory Pattern

In this approach, one uses confirmatory factor modeling to “clean up” a pattern obtained by exploratory methods. The steps are

1. An exploratory factor analysis is performed with the standard steps.
2. After the number of factors is selected, the researcher chooses the best exploratory rotation, and uses that as the basis for constructing a confirmatory factor model. Essentially, any loading below a certain cutoff is fixed at 0, and all other loadings are left as free parameters.

These steps can be performed in one step using the **`QuickEFAtoCFA`** function.

```
> fit <- QuickEFAtoCFA(AD.R,3,1000)
> fit


> GetPrettyPattern(fit)
```

```
          Factor1 Factor2 Factor3
X.1500M   0.794
X.2KROW   0.645   0.474
X.12MINTR 0.674
BENCH             0.839
CURL              0.677
MAXPUSHU  0.506   0.603
PINBALL                   0.602
BILLIARD                  0.753
GOLF                      0.745

               Factor1 Factor2 Factor3
SS loadings      1.758   1.751   1.484
Proportion Var   0.195   0.195   0.165
Cumulative Var   0.195   0.390   0.555

Factor Intercorrelations
------------------------
        Factor1 Factor2 Factor3
Factor1   1.000  -0.273   0.015
Factor2  -0.273   1.000   0.207
Factor3   0.015   0.207   1.000
```

Notice that the default is a correlated factor model. By using the cov.matrix=FALSE option, one may generate an orthogonal solution.

```
> fit <- QuickEFAtoCFA(AD.R,3,1000, cov.matrix=FALSE)
> GetPrettyPattern(fit)
          Factor1 Factor2 Factor3
X.1500M   0.772
X.2KROW   0.609   0.423
X.12MINTR 0.692
BENCH             0.812
CURL              0.694
MAXPUSHU  0.459   0.564
PINBALL                   0.595
BILLIARD                  0.759
GOLF                      0.744

               Factor1 Factor2 Factor3
SS loadings      1.657   1.639   1.484
Proportion Var   0.184   0.182   0.165
Cumulative Var   0.184   0.366   0.531

Factor Intercorrelations
------------------------
        Factor1 Factor2 Factor3
Factor1       1       0       0
Factor2       0       1       0
Factor3       0       0       1
```

3. Automating the "Confirm and Update" Approach

In the introductory handout, *Confirmatory Factor Analysis with R*¸ we demonstrated how modification indices can be used to update a confirmatory factor model that does not fit well. In this section, we demonstrate how the process can be speeded up by using the results of a rotated exploratory factor analysis as the basis of a confirmatory model.

We emphasize that this technique should be used with caution. It has some real advantages, one of which is that the parameter estimates from the exploratory solution are automatically used as starting values in the confirmatory analysis.

Let's take another look at the orthogonal solution generated in the previous section.

```
> fit <- QuickEFAtoCFA(AD.R,3,1000, cov.matrix=FALSE)
> CheckMod(fit)
$NewPath
[1] "Factor2->X.1500M,AddedTheta21, NA"

$ModIndex
[1] 41.91084
> CheckMod(fit)
$NewPath
[1] "Factor2->X.1500M,AddedTheta21, NA"

$ModIndex
[1] 41.91084

> fit <- UseMod(fit)
```

With one pass from **`modIndices`**, we have achieved the same model as in the introductory handout, but in a fraction of the time. Of course, with a simple R loop, you could automate the process.

4. The "Exploratory-Confirmatory" Approach Revisited

An alternative approach, which *begins* with a purely exploratory factor analysis, was described by Karl Jöreskog in his 1978 Presidential Address to the Psychometric Society.

As described in the introductory hand out on confirmatory factor analysis, Jöreskog's approach is as follows:

- Perform an exploratory factor analysis, and decide on the number of factors, m. In many textbook examples, the decision is relatively clear cut. Be forewarned — in practice the decision may be quite difficult.
- Fit an m-factor model, and rotate to simple structure using varimax or promax. (In the original article, Jöreskog said to use promax, but he used varimax in his numerical example. We'll use varimax to replicate what he did.)

- For each column of the factor pattern, find the largest loading, then constrain *all the other loadings in that row to be zero*, and fit the resulting model as a confirmatory factor model. This confirmatory model will have exactly the same discrepancy function and χ^2 value as the exploratory factor analysis that preceded it.
- Examine the factor pattern, and test all factor loadings. Delete "non-significant" loadings from the model. After checking the fit, the user can decide whether to terminate the process, or look for more loadings to delete.

In our introductory handout, we explained how the various steps are performed, and illustrated them using the AthleticsData. We demonstrated that the first 3 steps use confirmatory factor analysis to find a specific rotational position, but that this rotational position can actually be calculated directly. Once this position has been attained in a confirmatory factor analysis, the resulting pattern is "toned up" by eliminating non-significant loadings.

The function **QuickJoreskog** does all this automatically.

```
> R <- as.matrix(Harman74.cor$cov)
> fit <- QuickJoreskog(R,4,145)
```

This analysis could take several hours to complete using the **sem** package. The many steps requiring careful scanning of output result in quite a few errors. Compare the pattern on the next page to the final one in Jöreskog (1978). His has a chi-square of 301 with 231 degrees of freedom, while the model derived automatically has values of 294 and 230. By using the service function **FAtoREF,** you can generate the intermediate "reference solution" and see that he made a couple of errors deleting and including paths. This is very easy to do when performing these operations by hand.

This solution can be "improved" by using **CheckMod** and **UseMod** and adding a few paths.

```
> GetPrettyPattern(fit)
                       Factor1 Factor2 Factor3 Factor4
GeneralInformation      0.746           0.146
PargraphComprehension   0.829
SentenceCompletion      0.889           0.141  -0.196
WordClassification      0.511   0.213   0.148
WordMeaning             0.860
VisualPerception                0.735
PaperFormBoard                  0.546
Flags                           0.587
SeriesCompletion        0.265   0.540
Addition                                0.848
CountingDots           -0.211   0.327   0.671
WordRecognition                                 0.549
NumberRecognition                               0.531
ObjectNumber                                    0.642
Cubes                           0.467
Code                                    0.423   0.393
StraightCurvedCapitals          0.448   0.428
FigureRecognition               0.322           0.386
NumberFigure                    0.184           0.499
FigureWord                                      0.490
Deduction               0.321   0.413
NumericalPuzzles                0.460   0.361
ProblemReasoning        0.319   0.399
ArithmeticProblems      0.371           0.493

               Factor1 Factor2 Factor3 Factor4
SS loadings      3.493   2.725   1.970   1.828
Proportion Var   0.146   0.114   0.082   0.076
Cumulative Var   0.146   0.259   0.341   0.417

Factor Intercorrelations
------------------------
        Factor1 Factor2 Factor3 Factor4
Factor1   1.000   0.530   0.281   0.509
Factor2   0.530   1.000   0.247   0.523
Factor3   0.281   0.247   1.000   0.445
Factor4   0.509   0.523   0.445   1.000
```