SEM 4

Professor Timothy Bates

tim.bates@ed.ac.uk

http://timbates.wikidot.com/mv-stats

Overview

- Week 1: Factor analysis
- Week 2: Confirmatory Factor Analysis
- Week 3: Path Analysis and SEM 1
- **Week 4: Path Analysis and SEM 2**
- Week 5: Complex causal modelling
  - Twins, multiple groups…

Log Likelihood Example

- Imagine we have tossed a coin 20 times
  - Observed 13 heads.
- People propose three competing models of the gambling hall
  - A friend says he wins all the time on heads: p(heads) = 1.0
  - The casino says they are fair : p(heads) = 0.5
  - A competitor says their coins are rigged : p(heads) = 0.0
- The closer the -2LL is to zero, the more likely are the data given the chosen model.
- Let's use R to calculate the -2LL for all the claimed p(heads), from 0.5 to .95 stepping by .05

Where we left off…

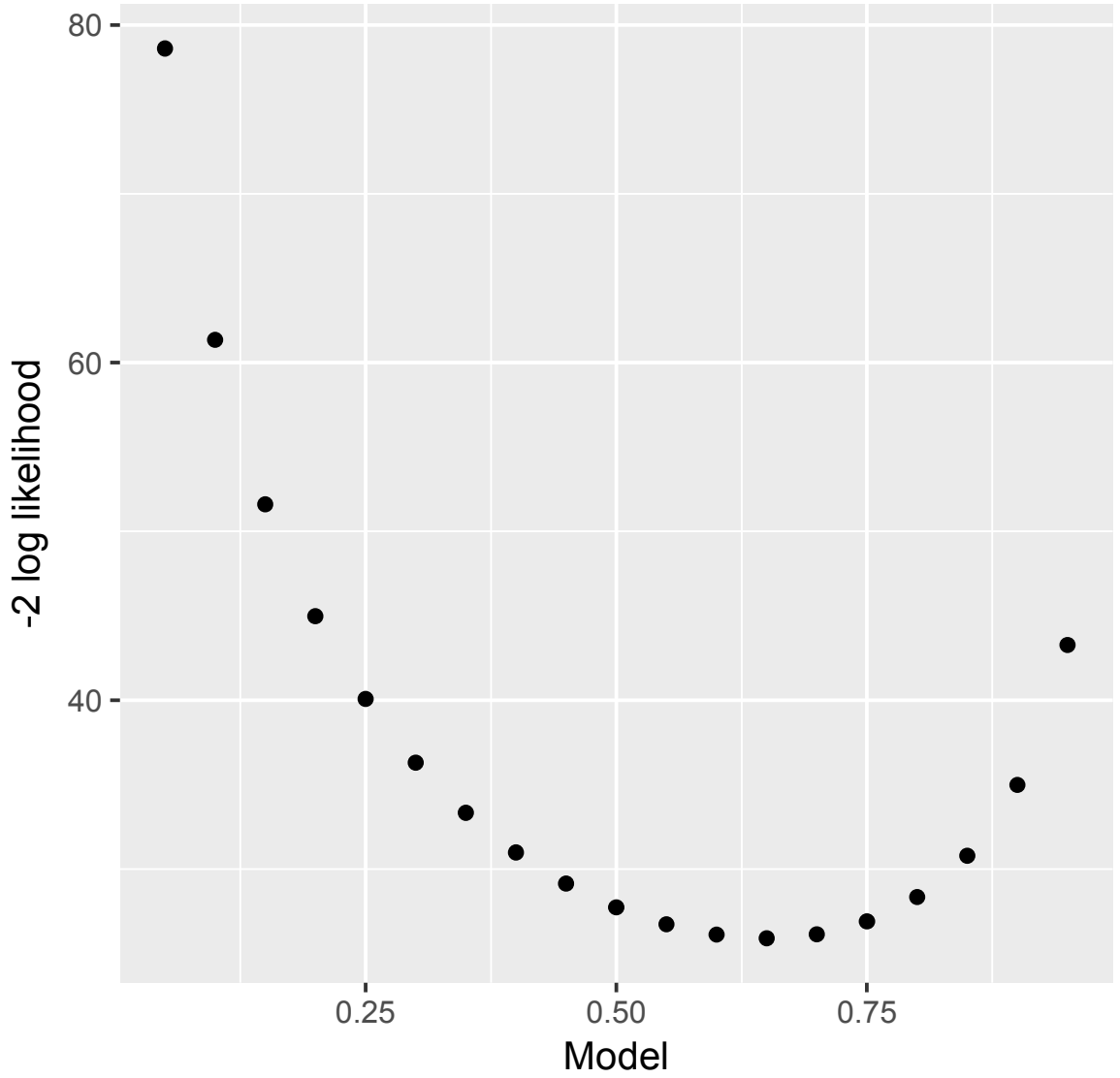
- Model specification
- Data
- **Estimation**
- Output
  - Parameters
  - Fit
- Model Comparison
- Model Updating

Compute log(likelihood)

```
model_prob = seq(.05, .95, by =.05)
minus2loglikelihood = rep(NA, length(model_prob))


i = 1
for(p in model_prob){
  # -2 Ã— log(likelihood) at a given value of p is:
  # (13 Ã— log(p)) + (17 Ã— log(1 - p))
  ll_p =  13 * log(p)
  ll_q = (20 - 13) * log(1 - p)
  minus2loglikelihood[i] = -2 * (ll_p + ll_q)
  i = i + 1
}

ggplot2::qplot(x = model_prob, y = minus2loglikelihood)
```

Compute -2*log(likelihood)

```
model_prob = seq(.05, .95, by =.05)
minus2ll   = rep(NA, length(model_prob))


i = 1
for(p in model_prob){
  ll_p =  13 * log(p)
  ll_q = (20 - 13) * log(1 - p)
  minus2ll[i] = -2 * (ll_p + ll_q)
  i = i + 1
}

qplot(model_prob, minus2ll)
```



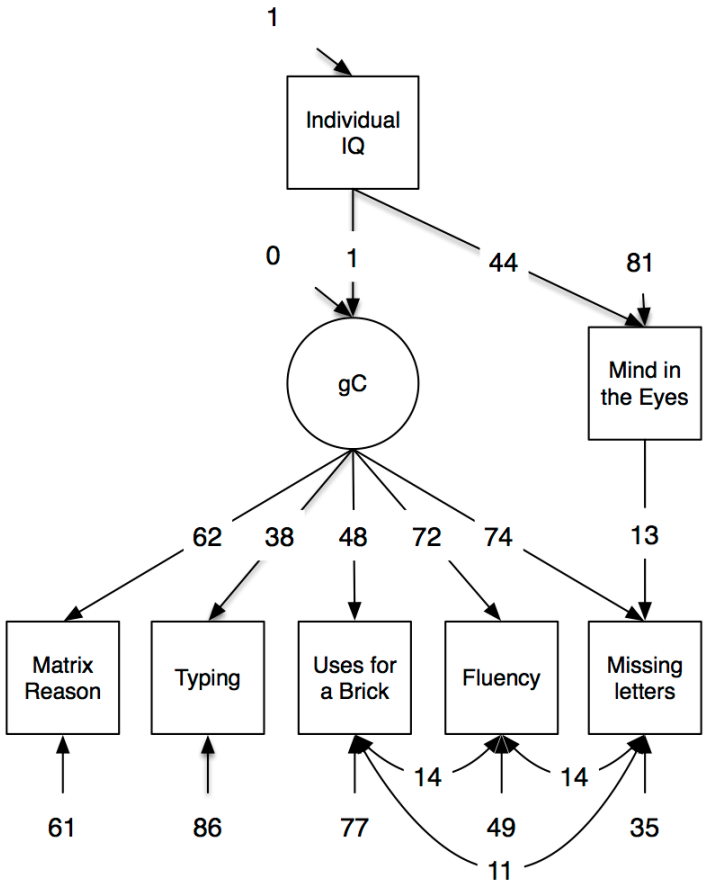
Model Fit

- Model specification
- Data
- Estimation
- **Output**
  - Parameters
  - **Fit**
- Comparison
- Updating in new samples…

Model Fit: How do we assess if a model fits "*well*" or "*poorly*", or "*better*"?

- Model fit of this specification can be evaluated by how closely the covariance of the data is recreated (likelihood), penalised by how many parameters we used.
- Key insight: Bad models don't allow very likely solutions…

If the theory that empathy causes collective IQ was true, this model would fit badly



Model Fit: How do we assess if a model fits "*well*" or "*poorly*", or "*better*"?

- **Key insight:** Bad models don't permit very likely solutions…
- We can compute the likelihood of a model, penalized by how complex it is, and thus set out criteria for good fit
  - Hu, L., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. Structural Equation Modeling, 6, 1-55.
  - Yu, C.Y. (2002). Evaluating cutoff criteria of model fit indices for latent variable models with binary and continuous outcomes. University of California, Los Angeles, Los Angeles. Retrieved from http://www.statmodel.com/download/Yudissertation.pdf

The Chi Square Test: χ2

- -2xLL is distributed as χ^2 with df degrees of freedom
  - This gives us our first test of fit: significance of χ2

- X^2 alone generates an excess of Type 1 errors due to
  - non-normality
  - small sample size
- With large sample size, it is too conservative (Type II)

Building on χ^2

- χ^2/df takes into account these limitations, but fails to adjudicate on good fit
- TLI and RMSEA build on this, with criteria for goodness (and inadequacy) of fit

Concept of worst and best fitting possible models

- Imagine we only model the mean and variance of each variable
  - *Independence* model AKA *null* model
- Imagine we add covariances between all possible variable pairings
  - *Saturated* model

- **Incremental Fit** (AKA ***relative fit***)
  - Fit relative to explaining nothing
  - Proportion of variance from 0 to 1
    - (null – myModel) /(null – Saturated)

Tucker-Lewis Index (AKA NNFI)

$$\frac{\chi^2/df(\text{Null Model}) - \chi^2/df(\text{Proposed Model})}{\chi^2/df(\text{Null Model}) - 1}$$

- Along with CFI, RMSEA, this is a core fit index
- X^2, penalized for degrees of freedom
- Good fit = TLI > .95

RMSEA:
Root Mean Square Error of Approximation

$$\frac{\sqrt{(\chi^2 - df)}}{\sqrt{|df(N - 1)|}}$$

- N = sample size
- *df* = model df
- Lower Bounded at 0.
- Good fit = RMSEA < .06

Root-mean square error of approximation

- RMSEA (Steiger & Lind, 1980)
  - Based on non-centrality parameter
  - An absolute measure index of fit
  - (relatively) independent of sample size.
    - Models with small samples, and few-df bias this upward (conservative)

  - RMSEA = sqrt((-2*ln(likelihood)-df)/(N*df))
    # < 0.06 is “good” fit and > 0.10 is ”poor” fit.

Akaike Information Criterion (AIC)

- Comparative measure of fit
  - Only meaningful when comparing models.
  - Model with the lower AIC is preferred
  - Several version exist: as long as you use the difference, all are the same
- $\chi^2 + k(k + 1) - 2df$
- k = number of variables
- AIC penalizes likelihood by 2 for each parameter added

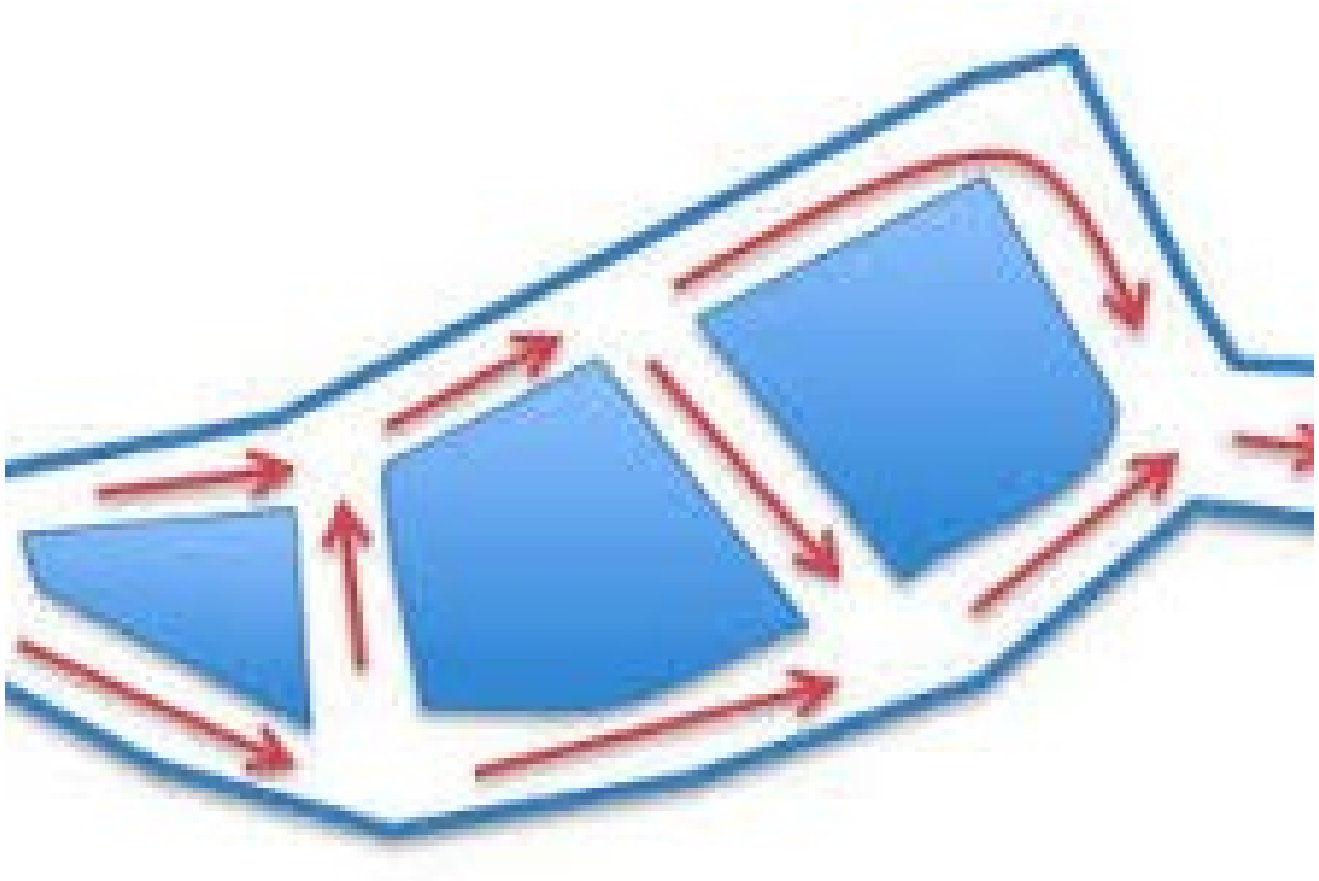
```
AIC = -2 * ln(likelihood) + 2(p + 1)
    # Model with lower AIC is preferred over model with high AIC
```

Output: Parameters and interpreting them Viewing parameters

- We can view a model as a figure or as a table
- plot(m1, means = F, std = T)
- umxSummary(m1, show = “std”)
- In the table we get parameters, and their SE, and a confidence interval.

Output: Parameters and interpreting them Getting from A to B and computing effects

- If all pathways lead to Rome, the product of the values on a path are how many people will follow that pathway
- And the total number of people is the sum of all possible routes



What are the effects predicted by my model? Path tracing rules

The expected correlation due to a chain traced between two variables is the product of the standardized path coefficients in the chain

The total expected correlation between two variables is the sum of these contributing path-chains.

- Caveats
  - Wright's rules assume a model without feedback loops: the directed graph of the model must contain no cycles.
- See also Boker 2005 (pdf)
  - BG handbook

Effects: Path tracing rules

- The rules for path tracing are:

1. You can trace backward up an arrow and then forward along the next, or forwards from one variable to the other, but never forward and then back.
   - You can't pass out of one arrow head and into another arrowhead
     - Heads-tails, or tails-heads, not heads-heads.
2. You can pass through each variable only once in a given chain of paths.
3. No more than one bi-directional arrow can be included in each path-chain.

Now we know about estimation and fit, and parameters, let's go back to model building

- Identification
  - Models must have measurement degrees of freedom to "pay" for all the paths they estimate. This is called being "identified"
- Scale
  - Latent variables need a scale if the model is to be identified.
- Constraints
  - More generally, fixing parameters (like setting the scale of a variable) imposes constraints, and these can allow identification

Identification

Model Identification: Is there a unique solution to our model?

- Model identification depends on having as many or more *known* parameters *as estimated* parameters
  - *Known* parameters = variances and covariances in data
  - *Unknown* parameters are those you are looking to estimate.

- Think of solving the equation “3 = x + y”
  - How many knowns?
  - How many unknowns?
  - How many solutions? Is it identified?

Levels of identification

- *Under-identified*
  - model has < 0 degrees of freedom.
- *Just Identified*
  - model has 0 degrees of freedom.
- *Over-Identified*
  - model has > 0 degrees of freedom.

Getting Identification via combinations of effects

Eq. 1: $x + y = 5$

Eq. 2: $2x + y = 8$

Eq. 3: $x + 2y = 9$

- EQ1 is under-identified.
- EQ1 AND EQ2 together are just identified.
- Taken together, EQ1, EQ2 & EQ3 are over identified.
- Chou & Bentler (1995)

Model Identification

- The number of known parameters is computed using the following equation:

$$(k+1)(k) / 2$$

  - K = number of observed variables
- When we subtract the number of parameters to be estimated (unknowns) from the result of the above equation, we get the degrees of freedom (df) for the model (also known as t-rule).
- In complex models, this is difficult, and a number of “short-cuts” are worth learning

Model Identification: A 3-manifest example

- Typical 3-indicator model
  - k = 3

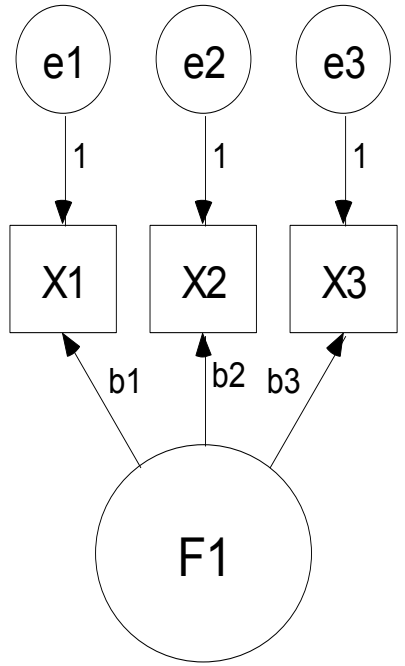
(k+1)(k)/2

(3+1)(3)/2

4x3/2

12/2

= 6 known parameters.



Model Identification: Where our “known knowns” come from

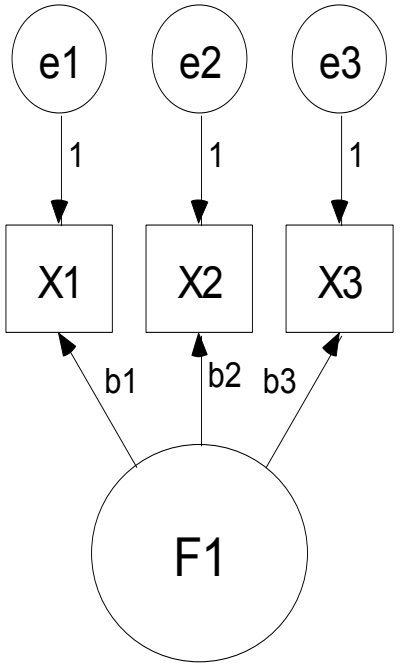
- Consider the data variance/covariance matrix
- With 3 indicators (observed variables), we have the following variance covariance matrix:

	x1	x2	X3
X1	Var(x1)		
X2	Cov(x2,x1)	Var(x2)	
X3	Cov(x3,x1)	Cov(x3,x2)	Var(x3)

3 indicator rule

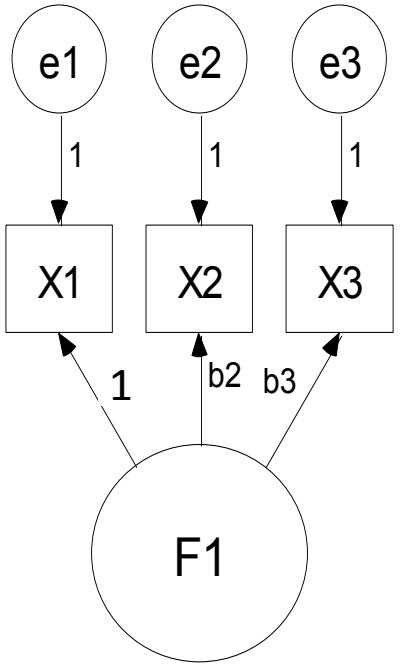
- The unknown parameters are
  - var(e1), var(e2), var(e3)
  - b1, b2, b3
  - var(f1)
- Total = 7-unknown parameters

- 6 -7 = -1 df
  - model is not identified ☹

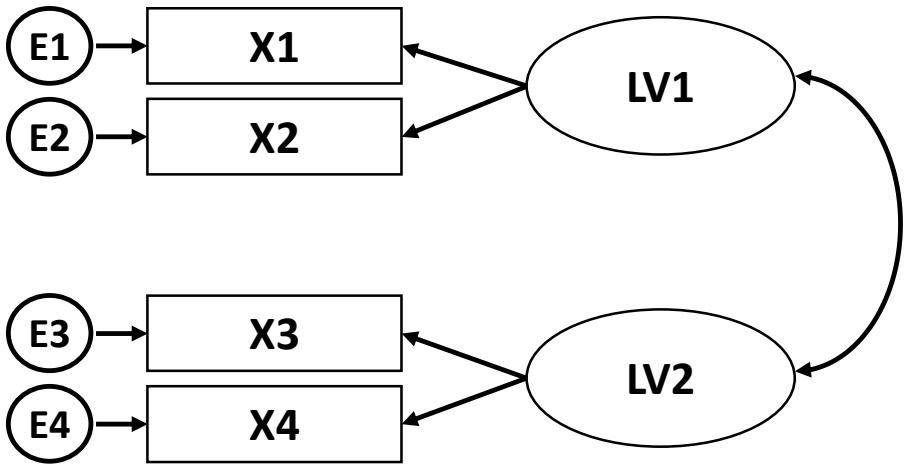


3 indicator rule: Fix variance of F1, or 1-path

- Scaling constraints, and other types of constraint (e.g. equating) “buy” df back.
- Often necessary to ensure identification.



Model Identification: 2 indicator rule



- Two indicator factors are identified if…
  1. 2 or more variables per latent factor.
  2. Each variable loads on only 1 factor.
  3. All factors have at least 1 non-0 correlation with another factor
  4. error terms are uncorrelated.

mxCheckIdentification

- This is only a cursory guide to identification
- Identification can depend on the data
- Identification depends on the parameter values
- Models can have local non-identified elements

- mxCheckIdentification can run an automated check for you

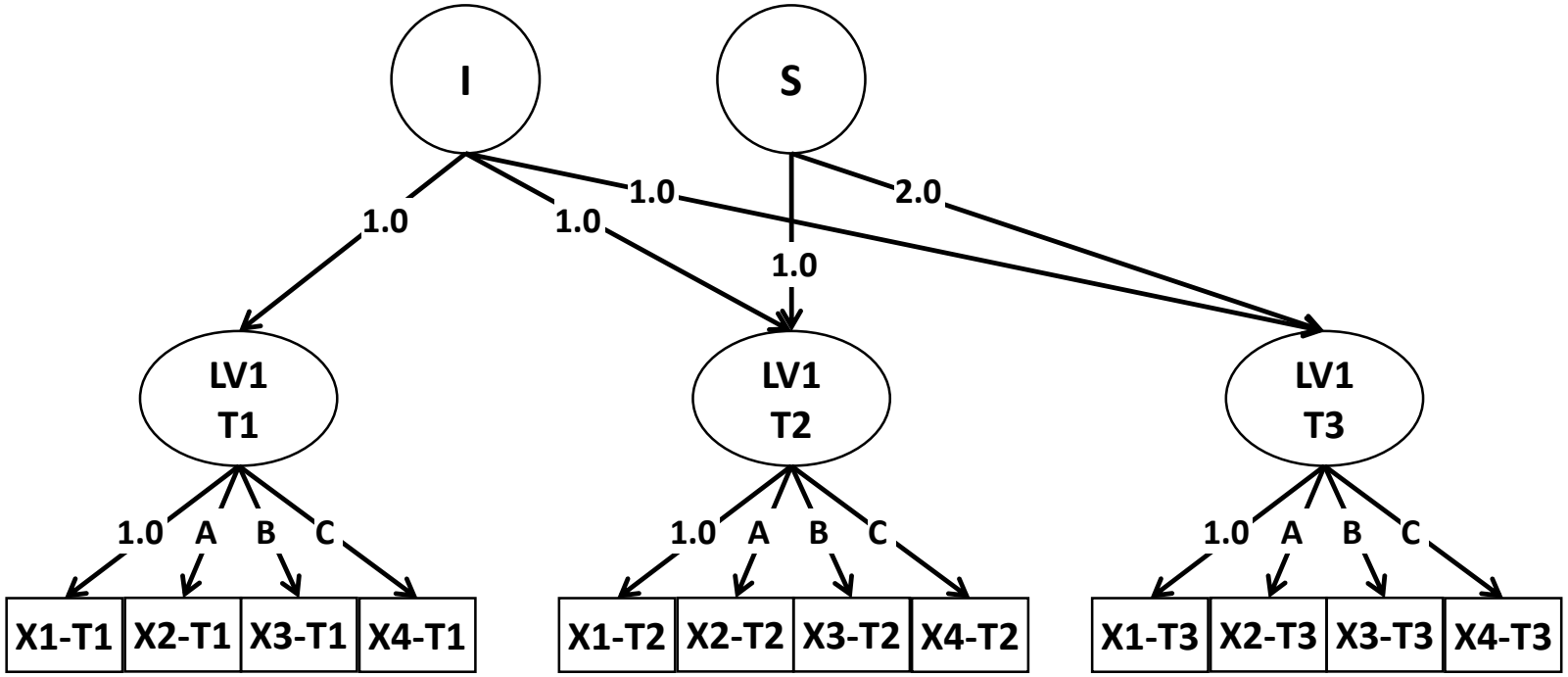
Model Specification: Scaling Latent Constructs

- Providing a scale for the latent variables allows the paths from them to have quantitative meaning.
- Two common ways to scale a construct
  - Fix the variance of the latent variable (nearly always to 1)
    - Preferred as it standardizes the latent variable and makes the paths more interpretable
  - Fix the first path from the latent variable to manifest (nearly always to 1).
    - Often seen in paths from latent residuals (impact of residual = its variance (*1)

Model Specification: Constraints

- Scaling is an example of a more general technique of constraint
- Can fix parameters at specified values
  - `umxPath("a", to "b", fixedAt = 0) # drop this path`
- Constrain within ranges
  - lbound and ubound
- Can equate paths
  - Value of path the same for males and females?
    - In umx, do this by setting parameters the same label
      - Same label, same value
  - Claw back degrees of freedom by equating item loadings
- Implement predicted model structures…

Equality Constraints to make a growth curve model



What if I change the model: Confirmatory or Exploratory?

- Testing a specific model is confirmatory.
- Model-modification is common
  - Often based on model-derived measures such as modification indices.
    - Modification indices provide information on model improvement if a given parameter is added to the model.
- After modification the model is no longer confirmatory, it is exploratory.
- Poses problems:
  - Are the additional parameters capitalizing on chance (sample specific)?
  - Are the additional parameters theoretically justified?
- Modified models require replication – rarely seen in published research.

Modification Indices
umxMI() and the hunt for better fit

- Explores the effect of adding and removing paths from a model
  - The output shows paths that would appear to lead to a better fit.
    - **note**: This is COMPLETELY POST HOC
- This is a very simplistic and automated add-one-in procedure, which can’t explore models which differ by more than a path…
- Can be very misleading: Need a mind to invent new theoretical models

Looking under the hood: Matrices, matrix algebra, and the functions that build models

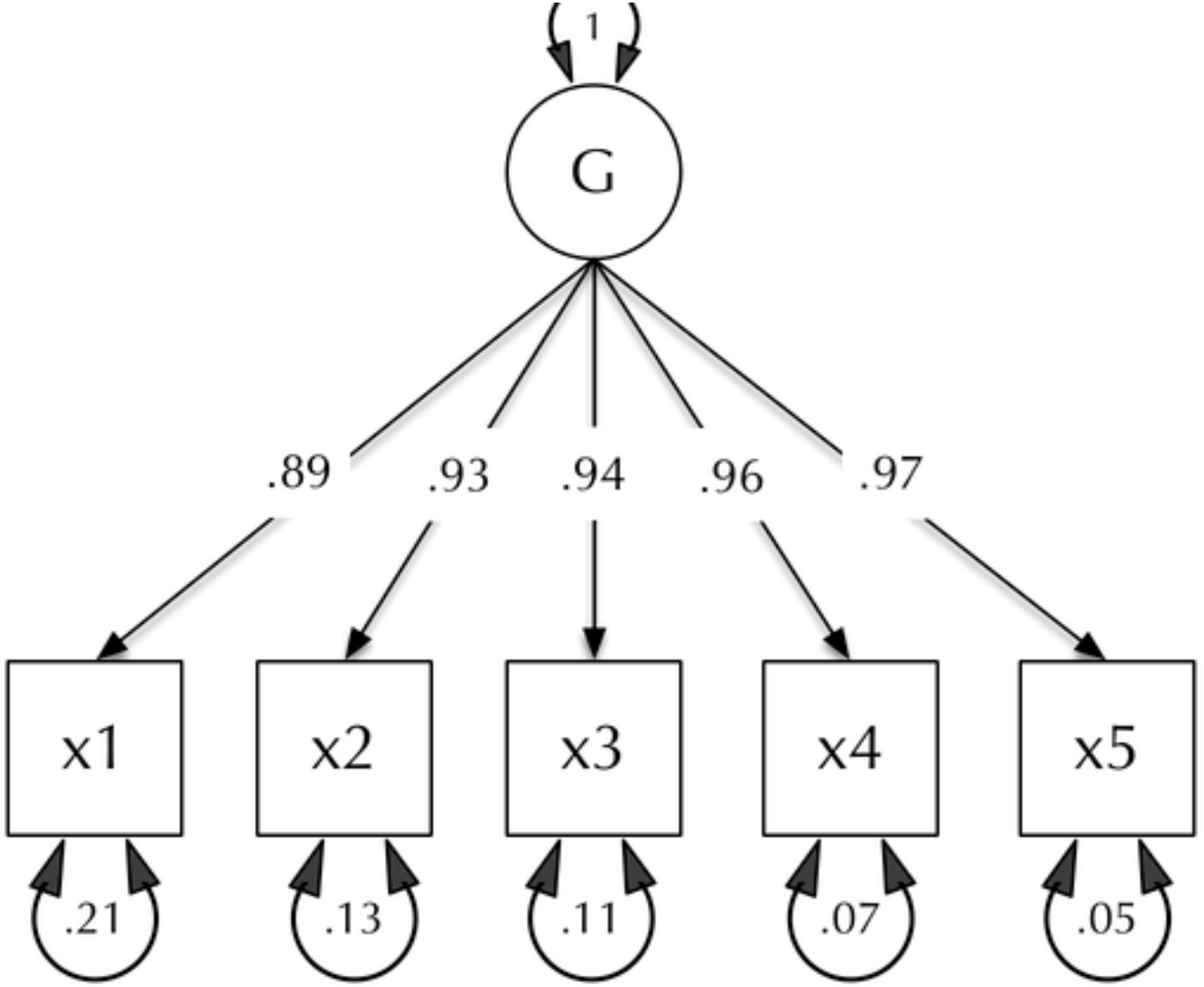
This is our 1-factor factor CFA

```
m1 <- umxRAM("One Factor",
    data = mxData(df, type = "cov", numObs = 500),
    umxPath("G", to= manifests),
    umxPath(var = manifests),
    umxPath(var ="G", fixedAt = 1)
  )
```

It makes this model

```
m1 <- umxRAM("One Factor",
    data = mxData(df, type = "cov"…),
    umxPath("G", to= manifests),
    umxPath(var = manifests),
    umxPath(var ="G", fixedAt = 1)
)
```

- plot(m1)



How did that get done under the hood? mxAlgebra; mxExpectation, mxFitFunctionML()

- Some data…
- require(OpenMx)
  data(demoOneFactor)
  manifests <- names(demoOneFactor)
  df = cov(demoOneFactor)

This is the same thing under the hood

```
m1 <- mxModel("One Factor",
    mxMatrix("Full”     , nrow = 5, ncol = 1, values = .2, free = T, name = "A"),
    mxMatrix("Symm", nrow = 1, ncol = 1, values = 1, free = F, name = "L"),
    mxMatrix("Diag”    , nrow = 5, ncol = 5, values = 1, free = T, name = "U"),
    mxAlgebra(A %*% L %*% t(A) + U, name="R"),
    mxExpectationNormal(covariance= "R", dimnames = varNames),
    mxFitFunctionML(),
    mxData(df, type = "cov", numObs = 500)
)
summary(mxRun(m1))
```

This algebra computes the expected covariance

```
mxAlgebra(A %*% L %*% t(A) + U, name="R"),
mxExpectationNormal(covariance= "R", dimnames = names(demoOneFactor)),
```

$$\mathbf{R} = \mathbf{ALA}' + \mathbf{U}$$

Constraints

- Bounding: Like that the variance of a parameter cannot be less than 0
- Theory-driven constraints include
  - Test whether boys and girls benefit equally from education
  - Constrain genetics effects in DZ twins to half that of MZ twins.
- Equating and constraining in models
  - reuse same label in different paths
  - Fix paths @ the same value
  - mxConstraint(A == B)

Algebras

- SEM can include algebraic functions
- Allows functions on paths
  - Computing parameters, like earnings are a log function of effort
  - Heritability is a function of environment
  - Variance = SD^2
- mxAlgebra()

OpenMx

- m1@matrices$A@labels
- m1 = umxLabel(m1) # add meaningful labels to the paths
- m1@matrices$A@labels

```
#   x          y
# x NA       NA
# y "x_to_y" NA
```

Matrix algebra

```
A = matrix(2, nrow = 1, ncol = 1)
B = matrix(c(.5, 1, 1.5), nrow = 1, ncol = 3)
```

A

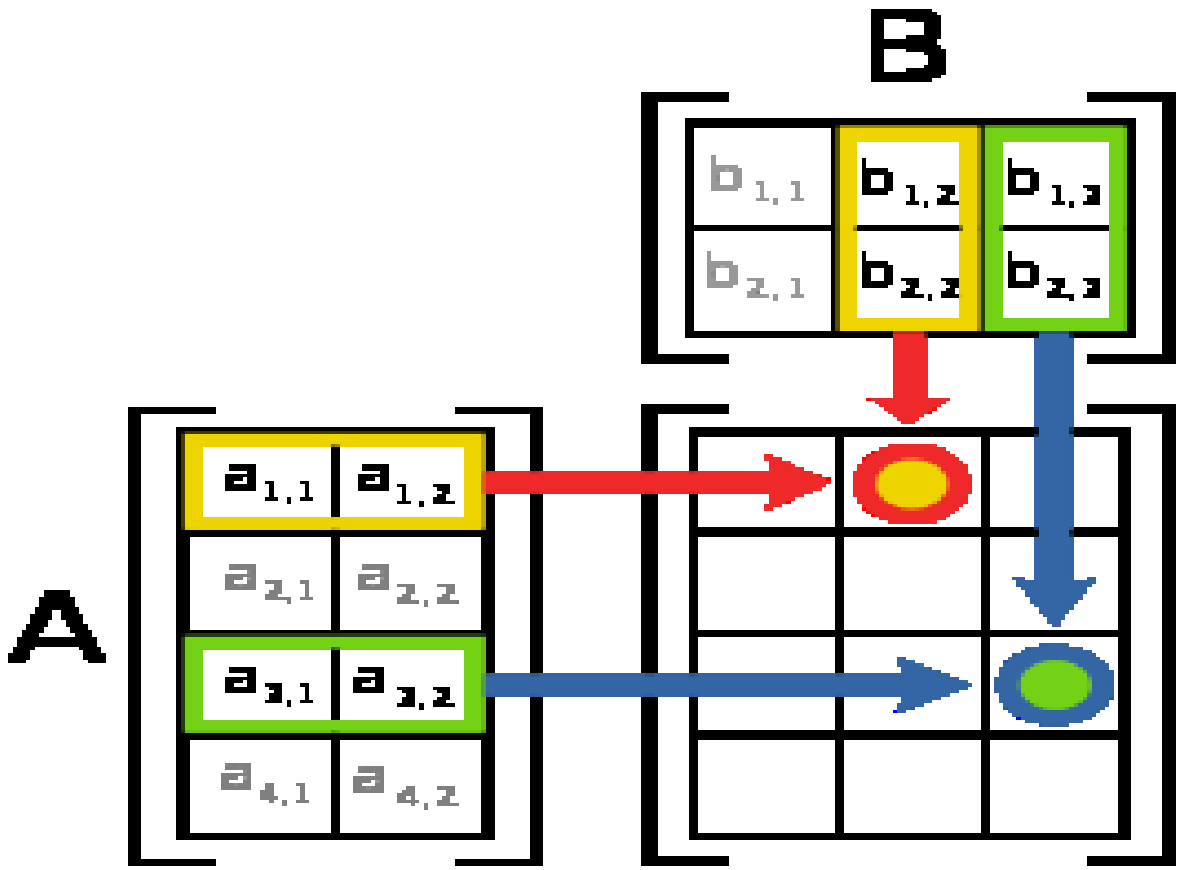
```
     [,1]
[1,]    2
```

B

```
     [,1] [,2] [,3]
[1,]  0.5    1  1.5
```

```
A %*% B
     [,1] [,2] [,3]
[1,]    1    2    3
```

http://en.wikipedia.org/wiki/Matrix_multiplication



Multiply two matrices: A × B = C

- One by one, the rows of A are dot-multiplied by the columns of B to fill the cells of C
  - Each r**A** × **B**c fills one cell of r**C**c
    - C[r, c] = sum(**A**[r,] %*% **B**[, c])
- So, to be conformable: ncol(A) == nrow(B)
- Resulting matrix has as many rows as A and as many columns as B

- Remember by “inside numbers must match” and “result has the size of the outside numbers”

How to do it

1. Figure out how big the output will be and write down an empty matrix of that size
2. Work through filling in each cell in the output matrix
   - ✓ Output[1,1] will contain the sum of A[1,] * B[,1]
   - ✓ Output cell [r,c] will contain sum(A[r,] * B[,c])

Conformability and how A*B ≠ B*A

- ```
  A = matrix(1:3, nrow = 3, ncol = 1)
  B = matrix(1:3, nrow = 1, ncol = 3)
  A %*% A
  Error in A %*% A :
    non-conformable arguments
  ```

- ```
  A %*% B
       [,1] [,2] [,3]
  [1,]    1    2    3
  [2,]    2    4    6
  [3,]    3    6    9

  B %*% A
       [,1]
  [1,]   14
  ```

Tutorial Time

- Come to the talk this afternoon too: Will be very enlightning regarding education