The Mathematica® Journal

Structural Equation Modeling

Comparing Two Approaches

Reinhard Oldenburg

Structural equational modeling is a very popular statistical technique in the social sciences, as it is very flexible and includes factor analysis, path analysis and others as special cases. While usually done with specialized programs, the same can be achieved in Mathematica, which has the benefit of allowing control of any aspect of the calculation. Moreover, a second, more flexible, approach to calculating these models is described that is conceptually much easier yet potentially more powerful. This second approach is used to describe a solution of the attenuation problem of regression.

The SEM Method

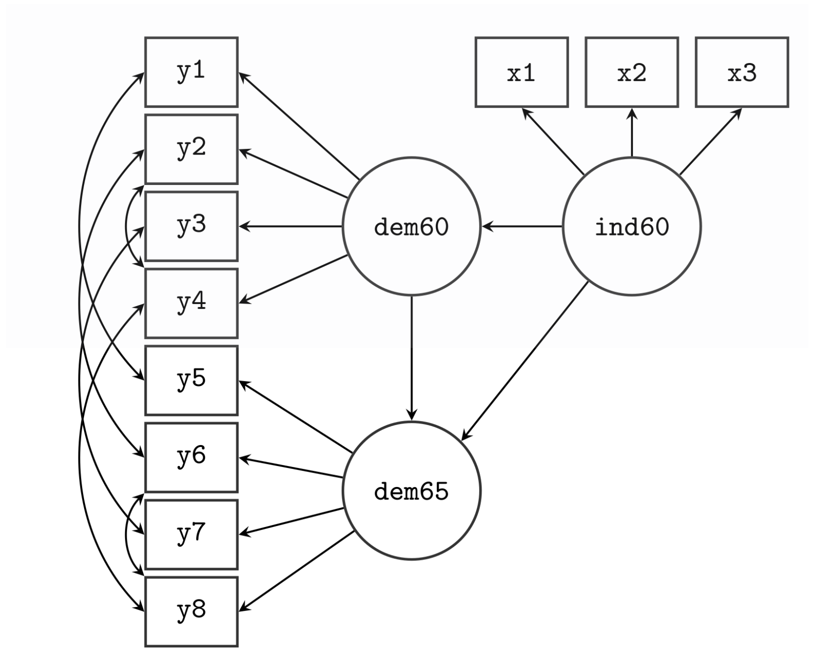
Linear structural equation modeling (SEM) is a technique that has found widespread use in many sciences in the last decades. An early foundational work is Bollen [1]; a more recent overview is provided by Hoyle [2]. The basic idea is to model the linear structure of k observed variables $x_1, \ldots, x_k$ of n cases (observations, subjects) by linear equations that may involve latent variables. These variables are not measured directly but inferred from the observed variables by their linear relation to the observed variables.

Many commercial programs (including LISREL, Amos, Mplus) and free ones (including lavaan, sem, OpenMX) have been developed to carry out the estimation procedure. From my perspective, the R package lavaan [3, 4] by Yves Rosseel is the most reliable and convenient one among the free programs. I use it as the gold standard to judge results of my own code.

This article first gives a quick overview of the standard SEM theory, then shows how to perform the calculations in Mathematica. In the last section, a second approach is discussed.

The Standard Example

There is a standard example due to Bollen that is also used in the lavaan manual. The dataset consists of observations of 11 manifest variables $x_1, x_2, x_3, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8$. SEM models are usually depicted graphically. In the lavaan documentation, this is displayed as in Figure 1.



▲ **Figure 1.** Bollen's democracy model (image from lavaan documentation [4]).

The variables x_1, x_2, x_3 are observed variables that measure the construct of industrialization in 1960, which is described by the latent variable ind60. This means that the level of industrialization is assumed to be representable by one number for each country, but this number cannot be measured directly; it has to be inferred from its linear relation to gross national product x_1, energy consumption per capita x_2 and share of industrial workers x_3. Next, dem60 and dem65 are the democracy levels in 1960 and 1965, measured by y_1, y_2, y_3, y_4 and y_5, y_6, y_7, y_8 (these indicators are freedom of the press, etc.). The data matrix consists of these 11 numbers for each of 75 countries (cases). The data is delivered with the lavaan package for R. The aim of estimating the model is twofold. First, the weights of the linear connections (represented in the picture by arrows) are estimated. These arrows encode linear equations by the rule that all arrows that end in a variable indicate a linear combination that yields the value of this variable plus some error term variable. To bring this mysterious language down to earth, here are the equations represented in Figure 1:

$$x_i = c_i \cdot \text{ind60} + e_i,\ i \in \{1, 2, 3\},$$
$$y_i = d_i \cdot \text{dem60} + f_i,\ i \in \{1, 2, 3, 4\},$$
$$y_i = d_i \cdot \text{dem65} + f_i,\ i \in \{5, 6, 7, 8\},$$
$$\text{dem60} = b_1 \cdot \text{ind60} + E_1,$$
$$\text{dem65} = b_2 \cdot \text{ind60} + b_3 \cdot \text{dem60} + E_2.$$

The variable ind60 is called an exogenous latent variable because no arrow ends there. It has no associated error variable. However, its manifest (measured) indicator variables x_1, x_2, x_3 have associated error variables e_i (they are called δ_i in [1]). The indicator variables y_1, y_2, y_3, y_4 and y_5, y_6, y_7, y_8 of the two endogenous latent variables (those latent variables where arrows end) have error variables f_i (called ε_i in [1]). The equations that relate latent and manifest variables define the measurement part of the model. The two equations (coming from three arrows) between the latent variables are the structure model, usually of most interest. Fitting the model to the data gives estimates for the weights of the arrows, $b_1, b_2, b_3, c_1, c_2, \ldots$. The second goal of SEM modeling is to check how well the structure of the model fits the data; that is, SEM is also a hypothesis-testing method.

The equations given do not yet identify all variables. Assume we have a solution of $x_i = c_i \cdot \text{ind60} + e_i$; then for any number k, the numbers $c_i' := c_i \cdot k$ and $\text{ind60}' := \text{ind60} / k$ would be solutions, too. To avoid this problem, we either fix the variance of the latent variables to be 1 or we fix some of the weights to be 1. This is the default in lavaan and we adopt it here, hence $c_1 = 1, d_1 = 1, d_5 = 1$.

■ The Standard Way of Estimating SEM

Ever since SEM's invention, SEM models are estimated by calculating the model's covariance matrix. From the data, we get the empirical covariance matrix S. On the other hand, from the model, we can calculate a theoretical covariance matrix Z between the observed variables. (Z depends on the model and thus on the parameters.) For example, one entry in this matrix would be $\text{cov}(x_1, x_2) = \text{cov}(c_1 \cdot \text{ind60} + e_1, c_2 \cdot \text{ind60} + e_2)$. Using linearity and other properties of the covariance, this boils down to a matrix with entries that are polynomials in the model parameters and the covariances and variances between latent variables and error variables. However, without further assumptions, this gives a lot of covariances (e.g. $\text{cov}(e_1, e_2)$) that are not determined by the model and hence must be estimated. As this usually leads to too much freedom, the broad assumption is that most error variables are uncorrelated. Only some covariances between error variables are not assumed to be 0; those are marked in the diagram by two-headed arrows between the observed variables. For every pair of observed variables, we calculate the covariance by using the above given model equation as replacement rules and applies linearity and independence assumptions. In the end, we get a covariance matrix Z that depends on the model parameters $b_1, b_2, b_3, c_1, c_2, \ldots$ and on the variances of the latent variables and the covariances of error variables that are not assumed to be 0. Details can be found in Bollen [1].

To fit the empirical and the theoretical covariance matrix, we have to choose these parameters to minimize some distance function. The three most common are uniform least-square, $F_{\text{ULS}} = \frac{1}{2} \text{tr}\left((S - Z)^2\right)$, generalized least-square, $F_{\text{GLS}} = \frac{1}{2} \text{tr}\left(\left(I - S^{-1} \cdot Z\right)^2\right)$ (I is the identity matrix), and maximum likelihood, $F_{\text{ML}} = \log(|Z|) + \text{tr}\left(S \cdot Z^{-1}\right) - \log(|S|) - k$ (here k is the number of manifest variables).

Now we are in the position to define a Mathematica function that performs SEM. First, we define the helper function `getAllVariables` that gets all variables contained in an expression in such a way that, for example, `x[1]` counts as one variable.

```
In[1]:= headlist = {Or, And, Equal, Unequal, Less, LessEqual,
           Greater, GreaterEqual, Inequality};
```

```
In[2]:= getAllVariablesAux[f_?NumericQ] = {};

In[3]:= getAllVariablesAux[{}] = {};

In[4]:= getAllVariablesAux[a_ → b_] :=
         Union[getAllVariablesAux[a], getAllVariablesAux[b]]

In[5]:= getAllVariablesAux[t_] /; MemberQ[headlist, t] = {};

In[6]:= getAllVariablesAux[l_List] :=
         Union@Flatten@Union@Map[getAllVariablesAux[#] &, l]

In[7]:= getAllVariablesAux[Derivative[n_Integer][f_][arg__]] :=
         getAllVariablesAux[{arg}]

In[8]:= getAllVariablesAux[f_Symbol[arg__]] :=
         Union@{
           If[
            MemberQ[Attributes[f], NumericFunction] ||
             MemberQ[headlist, f],
            getAllVariablesAux[{arg}],
            (*else*)
            f[arg]
           ]
          }

In[9]:= getAllVariablesAux[f_Symbol[arg__]] :=
         Union[{
           If[
            MemberQ[
               Attributes@f,
               NumericFunction
              ] || MemberQ[headlist, f],
            getAllVariablesAux@{arg},
            (*else*) f@arg
           ]
          }]

In[10]:= getAllVariablesAux[other_] := {other}

In[11]:= getAllVariables[a_] := Union@Flatten@getAllVariablesAux@a
```

Here is an example.

```
In[12]:= getAllVariables[1 + c x[1]^2]

Out[12]= {c, x[1]}
```

The method will be explained with Bollen’s democracy dataset, so first, we need to load this dataset. The file bollen.csv contains headers (the names of the variables are saved in the list `BollenObserved`) and a first column numbering the cases, which is dropped.

```
In[13]:= BollenData0 =
           Rest /@ Map[
             ToExpression,
             Import[
              "http://myweb.rz.uni-augsburg.de/~oldenbre/sem/bollen.
                 csv"],
             2
            ];
         BollenData = Rest@BollenData0;
         BollenObserved = First@BollenData0

Out[14]= {y1, y2, y3, y4, y5, y6, y7, y8, x1, x2, x3}
```

The data has 75 rows.

```
In[15]:= Dimensions@BollenData

Out[15]= {75, 11}
```

Here is the first row of 11 numbers.

```
In[16]:= BollenData0[[2]]

Out[16]= {2.5, 0, 3.33333, 0, 1.25, 0,
          3.72636, 3.33333, 4.44265, 3.63759, 2.55762}
```

The model itself has to be specified as a list of replacement rules that mirror the model equations discussed.

```
In[17]:= BollenRules = {
            x1 → c1 ind60 + e1,
            x2 → c2 ind60 + e2,
            x3 → c3 ind60 + e3,
            y1 → d1 dem60 + f1,
            y2 → d2 dem60 + f2,
            y3 → d3 dem60 + f3,
            y4 → d4 dem60 + f4,
            y5 → d5 dem65 + f5,
            y6 → d6 dem65 + f6,
            y7 → d7 dem65 + f7,
            y8 → d8 dem65 + f8,
            dem60 → b1 ind60 + E1,
            dem65 → b2 ind60 + b3 dem60 + E2,
            c1 → 1,
            d1 → 1,
            d5 → 1};

   In[18]:= BollenParameters = {c1, c2, c3, d1, d2, d3, d4, d5, d6,
            d7, d8, b1, b2, b3};
```

The code for the estimation function `SEM` includes some utilities. For example, it defines its own covariance and variance functions that take into account which variables are assumed to be uncorrelated. The input of `SEM` is the data matrix `data`, a matrix of numerical values, one row per case. The structural equations `modelRules` are given in the format detailed in the previous section, “The Standard Example.” Moreover, the function needs:

• the lists of free parameters, `parameters` (e.g. path weights)

• endogenous latent variables, `endogenousLatentVariables`

• exogenous latent variables, `exogenousLatentVariables`

• the list of error variables of latent variables, `errorVariablesOfLatentVariables`

• errors of exogenous manifest variables `errorsOfExogenousManifestVariables`

• errors of endogenous manifest variables `errorsOfEndogenousManifestVariables`

• a list `correlated` of pairs of error variables specifying which error variables are allowed to be correlated

The code after defining `Z` can be omitted on a first reading; it is only needed to calculate some fit indices (if required by the option `doFI`, which asks to do the fit index (FI) calculation; similarly, `doML` asks to do the maximum likelihood estimation). The estimation is done at the end of the function.

The goal of the first half of the `SEM` program is the definition of the covariance function `Cov` that takes into account the SEM assumptions: that most error variables are uncorrelated (except those specified to be correlated), leaving variances of latent variables as symbolic entities to be estimated.

This function is then used to calculate the model implied covariance matrix `Z`. Applying the model equation rules repeatedly gives a matrix that depends only on parameters, variances of latent variables and error variables and some allowed covariances of error variables. The code from the line defining `df` (the degree of freedom) onward is only important for getting fit indices. If we are only interested in estimating the model parameters, the next interesting lines are where `FindMinimum` is applied to estimate the model. As described in the introduction, there are several strategies to measure deviation of covariance matrices; for example, the definition of `uls` is a straightforward coding for minimizing $F_{\text{GLS}} = \frac{1}{2} \cdot \text{tr}\left(\left(I - S^{-1} \cdot Z\right)^2\right)$.

```
In[19]:= Options[SEM] = {doML → False, doFI → False, startValue → 1.0};
         Needs["HypothesisTesting`"];

In[21]:= SEM[data_, observed_, modelRules_, parameters_,
           exogenousLatentVariables_, endogenousLatentVariables_,
           errorVariablesOfLatentVariables_,
           errorsOfExogenousManifestVariables_,
           errorsOfEndogenousManifestVariables_, correlated_,
           OptionsPattern[]] :=
          Module[
           {n = Length@data, S = Covariance@data, Cov, Var, Z,
            i, j, ScalarQ, res = {}, Tr2, df, T, RMSEA, varr, est,
            gls, nullmodel, Znull, dfnull, Tnull, glsnull, solm,
            errorvars, errorvarsE},
           errorvarsE = Join[errorVariablesOfLatentVariables,
             errorsOfExogenousManifestVariables];
           errorvars = Join[errorvarsE,
             errorsOfEndogenousManifestVariables];
           Tr2[A_] := Tr[A.A];

           ScalarQ[a_] := True /; NumberQ[a] || MemberQ[parameters, a];
           ScalarQ[a_ + b_] := ScalarQ[a] && ScalarQ[b];
           ScalarQ[a_ b_] := ScalarQ[a] && ScalarQ[b];

           Var[a_ + b_] := Var[a] + Var[b] + 2 Cov[a, b];
           Var[s_ a_] := s^2 Var[a] /; ScalarQ[s];

           Cov[a_ + b_, c_] := Cov[a, c] + Cov[b, c];
           Cov[c_, a_ + b_] := Cov[a, c] + Cov[b, c];
           Cov[s_, b_] := 0 /; ScalarQ[s];
           Cov[b_, s_] := 0 /; ScalarQ[s];
           Cov[s_ a_, b_] := s Cov[a, b] /; ScalarQ[s];
           Cov[a_, s_ b_] := s Cov[a, b] /; ScalarQ[s];
           Cov[a_, b_] := Cov[b, a] /; Order[a, b] < 0;
           Cov[a_, a_] := Var[a];
           Cov[a_, b_] :=
            0 /; MemberQ[errorvars, a] && MemberQ[errorvars, b] &&
              Not[MemberQ[correlated, {a, b}] ||
                MemberQ[correlated, {b, a}]];
           Cov[a_, b_] :=
            0 /; MemberQ[errorsOfEndogenousManifestVariables, a] &&
              MemberQ[endogenousLatentVariables, b];
           Cov[b_, a_] :=
            0 /; MemberQ[errorsOfEndogenousManifestVariables, a] &&
              MemberQ[endogenousLatentVariables, b];
           Cov[a_, b_] :=
            0 /; MemberQ[errorvars, a] &&
              MemberQ[exogenousLatentVariables, b];
           Cov[b_, a_] :=
            0 /; MemberQ[errorvars, a] &&
              MemberQ[exogenousLatentVariables, b];
           (* Now calculate the model based symbolic covariance
            matrix Z *)
           Z =
            Table[Table[Cov[observed[[i]], observed[[j]]] //.
               modelRules, {i, Length@observed}],
             {j, Length@observed}];
           df = Length@observed (Length@observed + 1) / 2 -
             Length@getAllVariables@Z; (* degreees of freedom *)
           est[{f_, s_}] :=
            {{"df", df, "T", (n - 1) f, "ChiSq p",
              ChiSquarePValue[(n - 1) f, df], "RMSEA",
              N@Sqrt[((n - 1) f - df) / (df (n - 1))]}, s};
           If[
            OptionValue@doFI,
            Znull = Table[Table[If[i == j, varr[i], 0],
               {i, Length@observed}], {j, Length@observed}];
            dfnull = Length@observed (Length@observed + 1) / 2 -
              Length@getAllVariables@Znull;
            glsnull = FindMinimum[
              1 / 2 Tr2@IdentityMatrix[Length@S - Inverse[S].Znull],
              Map[{#, OptionValue@startValue} &,
               getAllVariables@Znull], MaxIterations → 10 000];
            nullmodel =
             FindMinimum[
              {Log@Det@Znull + Tr[S.Inverse[Znull]] - Log@Det@S -
                Length@observed, Det@Znull > 0.0},
              Map[{#[[1]], #[[2]]} &, glsnull[[2]]],
              MaxIterations → 10 000];
            Tnull = (n - 1) nullmodel[[1]];
           ];
           (* First estimation by the unweighted least square
            approach *)
           AppendTo[res,
            ULS →
             est@FindMinimum[1 / 2 Total@Map[#^2 &, Flatten[S - Z]],
               Map[{#, OptionValue@startValue} &, getAllVariables@Z],
               MaxIterations → 10 000]
           ];
           (* now estimation by the generalized least square
            approach *)
           gls = FindMinimum[
             1 / 2 Tr2[IdentityMatrix@Length@S - Inverse[S].Z],
             Map[{#, OptionValue@startValue} &, getAllVariables@Z],
             MaxIterations → 10 000];
           AppendTo[res, GLS → est@gls];
           If[OptionValue@doML || OptionValue@doFI,
            (* now estimation my maximum likelihood *)
            solm = FindMinimum[Log@Det@Z + Tr[S.Inverse[Z]] -
               Log@Det@S - Length@observed,
              Map[{#[[1]], #[[2]]} &, gls[[2]]],
              MaxIterations → 2000];
            AppendTo[res, ML → est@solm];
            If[
             OptionValue@doFI,
             res = {"df", df, "dfnull", dfnull, "T",
               (ML /. res)[[1, 4]], "Tnull", Tnull, "CFI",
               ((Tnull - dfnull) - ((ML /. res)[[1, 4]] - df)) /
                (Tnull - dfnull),
               "TLI", (Tnull / dfnull - (ML /. res)[[1, 4]] / df) /
                (Tnull / dfnull - 1),
               "NFI", (Tnull - (ML /. res)[[1, 4]]) / Tnull, res}
            ]
           ];
           res
          ]
```

Let us run the code on Bollen's model in a simplified version where no correlation of error variables is assumed. This may take several minutes.

```
In[22]:= sol1 =
          Quiet@SEM[BollenData, BollenObserved, BollenRules,
            BollenParameters, {ind60}, {dem60, dem65}, {E1, E2},
            {e1, e2, e3}, {f1, f2, f3, f4, f5, f6, f7, f8}, {},
            doFI → True, startValue → 0.5]

Out[22]= {df, 41, dfnull, 55, T, 71.4955, Tnull,
          720.912, CFI, 0.954205, TLI, 0.938568, NFI,
          0.900826, {ULS → {{df, 41, T, 844.881, ChiSq p,
              OneSidedPValue → 1.06285 × 10^-150, RMSEA, 0.51474},
             {b1 → 1.28932, b2 → 0.414362, b3 → 0.846741,
              c2 → 2.06722, c3 → 1.6326, d2 → 1.36095, d3 → 1.00928,
              d4 → 1.33757, d6 → 1.32026, d7 → 1.31953, d8 → 1.34769,
              Var$2877[e1] → 0.0195517, Var$2877[E1] → 4.01019,
              Var$2877[e2] → 0.0702004, Var$2877[E2] → 0.136539,
              Var$2877[e3] → 0.596427, Var$2877[f1] → 2.00796,
              Var$2877[f2] → 6.55861, Var$2877[f3] → 5.80287,
              Var$2877[f4] → 2.50496, Var$2877[f5] → 2.63991,
              Var$2877[f6] → 4.07917, Var$2877[f7] → 3.51127,
              Var$2877[f8] → 2.93143, Var$2877[ind60] → 0.517597}},
           GLS → {{df, 41, T, 53.6555, ChiSq p,
              OneSidedPValue → 0.0889695, RMSEA, 0.0645849},
             {b1 → 1.5478, b2 → 0.725271, b3 → 0.779033,
              c2 → 2.33574, c3 → 2.0104, d2 → 1.60797, d3 → 1.03095,
              d4 → 1.39164, d6 → 1.52055, d7 → 1.40677, d8 → 1.44845,
              Var$2877[e1] → 0.0582072, Var$2877[E1] → 3.46104,
              Var$2877[e2] → 0.138906, Var$2877[E2] → 0.274183,
              Var$2877[e3] → 0.407624, Var$2877[f1] → 1.22166,
              Var$2877[f2] → 3.57903, Var$2877[f3] → 3.78616,
              Var$2877[f4] → 1.90577, Var$2877[f5] → 1.54902,
              Var$2877[f6] → 2.22567, Var$2877[f7] → 2.74435,
              Var$2877[f8] → 2.06181, Var$2877[ind60] → 0.306766}},
         ML → {{df, 41, T, 71.4955, ChiSq p,
              OneSidedPValue → 0.00222553, RMSEA, 0.100256},
             {b1 → 1.47374, b2 → 0.453254, b3 → 0.864394,
              c2 → 2.18175, c3 → 1.81877, d2 → 1.35401, d3 → 1.04401,
              d4 → 1.29954, d6 → 1.25848, d7 → 1.28249, d8 → 1.30977,
              Var$2877[e1] → 0.0829343, Var$2877[E1] → 3.92392,
              Var$2877[e2] → 0.120038, Var$2877[E2] → 0.116466,
              Var$2877[e3] → 0.473522, Var$2877[f1] → 1.96813,
              Var$2877[f2] → 6.57733, Var$2877[f3] → 5.41208,
              Var$2877[f4] → 2.92618, Var$2877[f5] → 2.42242,
              Var$2877[f6] → 4.40157, Var$2877[f7] → 3.557,
              Var$2877[f8] → 2.98006, Var$2877[ind60] → 0.454214}}}}
```

The result combines parameter, variance and covariance estimations according to the various estimating strategies. To judge how well the model fits the data, you can set the option `doFI` to some fit indices:

• RMSEA is the root square mean error

• CFI is the comparable fit index

• TLI is the Tucker–Lewis fit index

• NFI is the normed fit index

RMSEA should be less than 0.1 or better, less than 0.05, and the last three should all be greater than 0.9 or 0.95 for good model fit.

The results of estimating using the three different methods differ somewhat. This is not a bug of our program; lavaan determines the same numbers up to several decimal places. There are results in the literature about which methods are equivalent under which conditions. For these fit indices to be interpretable, we need to assume that the data is multivariate normally distributed. If this assumption is violated, then we should judge model fit by other indices, which is beyond the scope of this article; however, they could be calculated based on the current approach as well. The book edited by Hoyle [2] gives some information on these methods.

For the original model that allows some covariances between error variables, the runtime gets worse, especially for maximum likelihood estimation. Hence, this is turned off in the following code.

```
In[23]:= sol2 =
          Quiet@SEM[BollenData, BollenObserved, BollenRules,
            BollenParameters, {ind60}, {dem60, dem65}, {E1, E2},
            {e1, e2, e3}, {f1, f2, f3, f4, f5, f6, f7, f8},
            {{f1, f5}, {f2, f4}, {f2, f6}, {f3, f7}, {f4, f8},
             {f6, f8}}, doML → False]
```

```
Out[23]= {ULS → {{df, 35, T, 269.722, ChiSq p,
            OneSidedPValue → 4.98046 × 10^-38, RMSEA, 0.301042},
          {b1 → 1.34706, b2 → 0.434015, b3 → 0.842094,
           c2 → 2.06404, c3 → 1.62794, d2 → 1.2413, d3 → 0.993572,
           d4 → 1.29367, d6 → 1.18893, d7 → 1.30967, d8 → 1.30114,
           Cov$3674[f1, f5] → 0.508461, Cov$3674[f2, f4] → 1.40076,
           Cov$3674[f2, f6] → 2.66342, Cov$3674[f3, f7] → 1.07878,
           Cov$3674[f4, f8] → 0.344901, Cov$3674[f6, f8] → 1.55098,
           Var$3674[e1] → 0.0177617, Var$3674[E1] → 4.10648,
           Var$3674[e2] → 0.0693836, Var$3674[E2] → 0.138832,
           Var$3674[e3] → 0.59955, Var$3674[f1] → 1.82961,
           Var$3674[f2] → 7.80021, Var$3674[f3] → 5.78,
           Var$3674[f4] → 2.76911, Var$3674[f5] → 2.49728,
           Var$3674[f6] → 5.25693, Var$3674[f7] → 3.37514,
           Var$3674[f8] → 3.20606, Var$3674[ind60] → 0.519387}},
        GLS → {{df, 35, T, 35.9469, ChiSq p,
            OneSidedPValue → 0.423958, RMSEA, 0.0191202},
          {b1 → 1.75509, b2 → 0.666841, b3 → 0.809661,
           c2 → 2.30078, c3 → 1.97669, d2 → 1.37207, d3 → 1.074,
           d4 → 1.27946, d6 → 1.29919, d7 → 1.38062, d8 → 1.31194,
           Cov$3674[f1, f5] → 0.419365, Cov$3674[f2, f4] → 1.43263,
           Cov$3674[f2, f6] → 1.27986, Cov$3674[f3, f7] → 0.707215,
           Cov$3674[f4, f8] → 0.296529, Cov$3674[f6, f8] → 0.970228,
           Var$3674[e1] → 0.0531497, Var$3674[E1] → 3.57272,
           Var$3674[e2] → 0.149173, Var$3674[E2] → 0.189575,
           Var$3674[e3] → 0.404741, Var$3674[f1] → 1.41245,
           Var$3674[f2] → 6.14362, Var$3674[f3] → 4.07991,
           Var$3674[f4] → 2.78536, Var$3674[f5] → 1.88536,
           Var$3674[f6] → 3.71518, Var$3674[f7] → 2.92723,
           Var$3674[f8] → 2.82744, Var$3674[ind60] → 0.321791}}}
```

The results of both models are exactly the same as calculated with lavaan.

An Alternative Approach: Case-based Estimation

When I first learned about SEM, I was puzzled by the many notions (e.g. exogenous, endogenous) and the assumptions needed. For example, I felt that correlation of error variables should be calculated by the estimation algorithm and not be set at will when specifying the model. However, these difficulties seem to play no large role in practice and there are thousands of research papers (mainly) in the social sciences that use these methods with great success. Yet, there are some reasons why the standard approach to SEM via covariance matrices can be criticized (a more detailed discussion is given in [5]). Traditional SEM:

• is well suited only for linear models (there are some nonlinear extensions, but they have not yet become mainstream)

• does not give estimates of the values of latent variables for each case (Bayesian variants can do this)

• requires the covariance matrix of observed data to be nonsingular; however, improving measurement methods in x_1, x_2, x_3, for example, may result in highly correlated measures of ind60 (in the extreme case with identical vectors of measured values) and hence their covariance matrix will be almost singular

• has resulting estimations for parameters that depend a lot on the estimation method used

• forbids certain linear models that are not identified in this approach, even though the model itself is sensible and well defined (e.g. the number of covariances of error variables allowed to be nonzero is limited, although in practice there may be correlations)

You may then wonder why the covariance matrix–based approach is so popular. I suppose that more than 40 years ago, computers were not powerful enough to deal with a full dataset, so that the information reduction by calculating the correlation matrix was essential. Since then, many powerful programs have been developed and research has been carried out that gave a good understanding of conditions under which the method works well. Moreover, the psychometric community reached a consensus on how model fit should be judged and thus studies using this method faced no problem being published.

After this discussion of pros and cons, it is time to present the following case-based approach to SEM estimation that is very easy (one may even call it naive) to implement but is also very flexible and with today's computing power, it is feasible in many real-world situations.

Hence, I propose to do SEM case-based by least-square optimization of the defects of the equations. Assume we have n observations (cases) of k variables $x_i \in \mathbb{R}^n$, $1 \leq i \leq k$. A general equational model consists of m equations $g_k(\{x_i\}, \{L_i\}, \{c_i\}) = 0$, $1 \leq k \leq m$, which involve the data, latent variables L_i, $i = 1, \ldots, l$, and parameters c_i. Then the latent variables and the parameters are estimated by minimizing $\sum_{k=1}^{m} \sum_{j=1}^{n} g_k\left(\left\{x_{i,j}\right\}, \left\{L_{i,j}\right\}, \{c_i\}\right)^2$.

Another twist is needed to get the best results, however. The above objective function gives all equations the same weight. However, it turned out (by working with simulated data where it is clear which parameters should be found) that we get better results by multiplying by a factor that gives the equations different weights, that is, $\sum_{k=1}^{m} \sum_{j=1}^{n} s_k\, g_k\left(\left\{x_{i,j}\right\}, \left\{L_{i,j}\right\}, \{c_i\}\right)^2$. The factor s_k can be modified by an option in the code that follows. Best results are obtained for $s_k = n^{-q}$, where q is the number of latent variables in g_k. The idea behind this choice is that an equation that involves only one latent variable links this variable directly to the manifest data and thus should have a high weight. In contrast, equations with many latent variables are not so close to the manifest observations and are thus are more hypothetical, so they should have a lower weight.

The model equations are not formulated as rules as for the first SEM, but as equations with the name of the error variable attached to each equation. Moreover, the dataset is not normalized, so there are nonzero intercepts in the linear equations. In the first approach this had no consequences, because such additive values are eliminated by calculating the covariance matrix, but in the SEM2 approach, intercepts must be modeled explicitly (and we have the benefit of getting estimates for them as well).

```
In[24]:= BollenEQ = {
            {dem60 == b1 ind60 + u1, e01},
            {dem65 == b2 ind60 + b3 dem60 + u2, e02},
            {x1 == 1 ind60 + t1, ee1},
            {x2 == c2 ind60 + t2, ee2},
            {x3 == c3 ind60 + t3, ee3},
            {y1 == 1 dem60 + s1, e1},
            {y2 == d2 dem60 + s2, e2},
            {y3 == d3 dem60 + s3, e3},
            {y4 == d4 dem60 + s4, e4},
            {y5 == 1 dem65 + s5, e5},
            {y6 == d6 dem65 + s6, e6},
            {y7 == d7 dem65 + s7, e7},
            {y8 == d8 dem65 + s8, e8}
          };
```

The function SEM2 that carries out the model estimation takes as input `data` and the names of the manifest (`observed`) and latent variables (`latentVariables`). At the technical heart of the function is the subroutine `EQ`. This function takes an equation involving latent variables (e.g. `dem60 == b1  ind60 + u1`) and adds to the objective function `targetFunction` the appropriate term for each case (i.e. with values from the data replacing the names of manifest variables):

$$(\mathtt{dem60}[1] - (\mathtt{b1}\,\mathtt{ind60}[1] + \mathtt{u1}[1]))^2 +$$
$$(\mathtt{dem60}[2] - (\mathtt{b1}\,\mathtt{ind60}[2] + \mathtt{u1}[2]))^2 +$$
$$\ldots +$$
$$(\mathtt{dem60}[n] - (\mathtt{b1}\,\mathtt{ind60}[n] + \mathtt{u1}[n]))^2$$

There is one option.

```
In[25]:= Options[SEM2] = {latentWeightFactor → 1};
```

```
In[26]:= SEM2[data_, observed_, latentVariables_, equations_,
           OptionsPattern[]] :=
         Module[
           {EQ, i, j, constraints = {}, targetFunction = 0},
    EQ[{eq_, err_}] := Module[
      {EEs, n = Length@data, manifest, i0, j0, weightFactor},
      vars = getAllVariables@eq;
      manifest = Intersection[observed,
        Flatten@Complement[vars, latentVariables]];
      EEs = Table[
        First@eq - Last@eq /. Union[
          Table[manifest[[j0]] →
            data[[i0]][[Position[observed, manifest[[j0]] ][[
              1, 1]] ]], {j0, Length@manifest}],
          Table[latentVariables[[j0]] →
            latentVariables[[j0]][i0],
           {j0, Length@latentVariables}]
         ],
        {i0, Length@data}
       ];
      weightFactor = OptionValue[latentWeightFactor]^
        Min[0,
         1 - Length@Intersection[vars, latentVariables]];
      targetFunction = targetFunction +
        Total@Map[#^2 &, EEs] weightFactor
     ];

    Map[EQ, equations];
    constraints = Join[constraints,
      Map[Total@Table[#[i], {i, Length@data}] == 0 &,
       latentVariables]];
    FindMinimum[
     Join[{targetFunction}, constraints],
     Map[{#, RandomReal[{0.1, 0.5}]} &,
      getAllVariables@Join[{targetFunction}, constraints]],
     MaxIterations → 10 000
    ]
   ]
```

This code estimates Bollen’s model.

```
In[27]:= estimate1 = SEM2[BollenData, BollenObserved,
            {ind60, dem60, dem65}, BollenEQ];
         {estimate1[[1]], Take[estimate1[[2]], 25]}

Out[28]= {1880.71, {b1 → 0.995703, b2 → 0.577854, b3 → 1.0518,
           c2 → 2.1902, c3 → 1.96527, d2 → 2.82293, d3 → 2.00049,
           d4 → 2.46247, d6 → 1.71956, d7 → 1.6727, d8 → 1.73107,
           s1 → 5.46467, s2 → 4.25644, s3 → 6.56311, s4 → 4.45253,
           s5 → 5.13625, s6 → 2.97807, s7 → 6.19626, s8 → 4.04339,
           t1 → 5.05438, t2 → 4.79219, t3 → 3.55769, u1 → -1.10653 × 10^-16,
           u2 → 1.96852 × 10^-17, dem60[1] → -1.61987}}
```

As mentioned, there is a version that weights equations according to the number of latent variables they have.

```
In[29]:= estimate2 = SEM2[BollenData, BollenObserved,
            {ind60, dem60, dem65}, BollenEQ,
            latentWeightFactor → Length@BollenData];
         {estimate2[[1]], Take[estimate2[[2]], 25]}
```

```
Out[30]= {1678.3, {b1 → 1.21904, b2 → 0.585325, b3 → 0.744532,
            c2 → 2.20543, c3 → 2.00439, d2 → 1.61306, d3 → 1.1638,
            d4 → 1.4081, d6 → 1.42969, d7 → 1.3938, d8 → 1.44193,
            s1 → 5.46467, s2 → 4.25644, s3 → 6.56311, s4 → 4.45253,
            s5 → 5.13625, s6 → 2.97807, s7 → 6.19626, s8 → 4.04339,
            t1 → 5.05438, t2 → 4.79219, t3 → 3.55769, u1 → -1.8441 × 10^-16,
            u2 → 6.51838 × 10^-17, dem60[1] → -2.85761}}
```

The results for the estimates differ from what is calculated in the traditional covariance matrix–based approach given for SEM. A simulation study that compares the two approaches [5] showed that in many situations the case-based approach gives better results, especially when the assumption of independent errors is violated. Moreover, the case-based approach is easily applied to nonlinear equations. However, in certain situations it may be necessary to perform the minimization with higher accuracy than provided by standard hardware floating-point numbers.

□ Application to Measurement Error

In standard linear regression $y \sim x$, one assumes that the independent variables are measured exactly, while the dependent variable has an error that is ideally normally distributed. If the independent variables are measured with error too, standard linear regression underestimates the regression coefficient. This is the famous attenuation problem and I will show how to solve it. Let us first simulate a dataset with error on both variables.

```
In[31]:= {X, Y, sx, sy} = Module[
            {
             n = 1000,
             A = 0.5, X0, Y0},
            X0 = RandomVariate[NormalDistribution[0, 1], n];
            Y0 = A X0;
            {
             X0 + RandomVariate[NormalDistribution[0, 0.3], n],
             Y0 + RandomVariate[NormalDistribution[0, 0.1], n],
             StandardDeviation[X],
             StandardDeviation[Y]
            }
           ];
```

Then linear regression underestimates the slope, which should be 0.5.

```
In[32]:= LinearModelFit[Transpose@{X, Y}, {1, x}, x]

Out[32]= FittedModel[ 0.00730558 + 0.460737 x ]
```

When using case-based modeling, several strategies are possible. We may use one or two latent variables for the true values. As the true dependent variable is just $y_0 = a\,x_0$, the following code uses just one latent variable. Another twist is that the equations are divided by the empirical standard deviations to put them on an equal footing.

```
In[33]:= a /. SEM2[
            Transpose@{X, Y}, {x, y}, {x0},
            {
             {x / sx == x0 / sx, e1},
             {y / sy == a x0 / sy, e2}
            },
            {}
           ][[2]]

Out[33]= 0.489653
```

This example shows both the power of this method and the responsibility of the modeler to set up sensible equations. If we are sure that the errors are uncorrelated, we may add `Sum[e1[i] × e2[i], {i, n}] == 0` as another constraint to further improve the estimate. This may also be done automatically with an extended version of SEM2, which will be published when its development is completed.

Summary

Two methods for the estimation of structural equational models are presented. One uses the traditional covariance matrix–based approach and is therefore restricted to linear equations, while the other approach is more general but not yet established in practice. Estimating the models is rather easy in Mathematica, but the numerical problems that arise can be demanding. The new case-based approach is very flexible and promising in certain situations where the standard approach shows limitations.

Conclusion

Case-based calculation of SEM looks very promising given the numerical power of today's computers and might give insight in situations where the restrictions of the traditional approach urge researchers into making assumptions that may not be warranted.

Acknowledgments

It is my pleasure to thank Ed Merkle and Yves Rosseel for many explanations of SEM.

References

[1] K. A. Bollen, *Structural Equations with Latent Variables*, New York: Wiley, 1989.

[2] R. H. Hoyle (ed.), *Handbook of Structural Equation Modeling*, New York: Guilford Press, 2012.

[3] K. Gana and G. Broc, *Structural Equation Modeling with lavaan*, Hoboken: John Wiley & Sons, 2019.

[4] Y. Rosseel. “lavaan.” (Aug 25, 2019) https://lavaan.ugent.be.

[5] R. Oldenburg, “Case-based vs. Covariance-based SEM,” forthcoming.

R. Oldenburg, “Structural Equation Modeling,” *The Mathematica Journal*, 2020.
https://doi.org/10.3888/tmj.22–5.

About the Author

Reinhard Oldenburg has studied physics and mathematics and received a PhD in algebra. He has been a high-school teacher and now holds a professorship in Mathematics Education at Augsburg University. His research interests are computer algebra, the logic of elementary algebra and real-world applications.

Reinhard Oldenburg
Augsburg University
Mathematics Department
Universitätsstraße 14
86159 Augsburg, Germany
reinhard.oldenburg@math.uni-augsburg.de