Introduction to the R package `plspm`

Gaston Sanchez, Laura Trinchera, Giorgio Russolillo

1 Introduction

`plspm` is an R package for performing **Partial Least Squares Path Modeling** (PLS-PM) analysis. Briefly, PLS-PM is a multivariate data analysis method for analyzing systems of relationships between multiple sets of variables. In this vignette we present a short introduction to `plspm` without providing a full description of all the package's capabilities. An extended documentation can be found in the book **PLS Path Modeling with R** freely available at: http://www.gastonsanchez.com/PLS_Path_Modeling_with_R.pdf

2 About PLS Path Modeling

Partial Least Squares Path Modeling (PLS-PM) is one of the methods from the broad family of PLS techniques. It is also known as the PLS approach to Structural Equation Modeling, and it was originally developed by Herman Wold and his research group during the 1970s and the early 1980s. Around the PLS community, the term Path Modeling is preferred to that of Structural Equation Modeling, although both terms can be found within the PLS literature. Our preferred definition of PLS-PM is based on three fundamental concepts:

1. It is a multivariate method for analyzing multiple blocks of variables
2. Each block of variables plays the role of a latent variable
3. It is assumed that there is a system of linear relationships between blocks

In other words: PLS-PM provides a framework for analyzing multiple relationships between a set of blocks of variables (or data tables). It is supposed that each block of variables is represented by a latent construct or theoretical concept; the relationships among the blocks are established taking into account previous knowledge (theory) of the phenomenon under analysis. There are plenty of references about PLS-PM, but we will only mention one from Wold, and two more recent ones:

- Esposito Vinzi V., Chin W.W., Henseler J., Wang W. (2010) Handbook of Partial Least Squares: Concepts, Methods and Applications. Springer.
- Tenenhaus M., Esposito Vinzi V., Chatelin Y.M., and Lauro C. (2005) PLS Path Modelling. *Computational Statistics & Data Analysis*, 48: 159-205.

- Wold H. (1982) Soft modeling: the basic design and some extensions. In: K.G. Joreskog & H. Wold (Eds.), *Systems under indirect observations: Causality, structure, prediction*, Part 2, pp. 1-54. Amsterdam: Holland.

3 Installation and Usage

`plspm` is freely available from the Comprehensive R Archive Network, better known as CRAN, at: http://cran.r-project.org/web/packages/plspm/index.html

Since version 0.4.0, `plspm` contains a set of features for handling non-metric data (discrete, ordinal, categorical, qualitative, etc). More information related with non-metric PLS-PM will be available in a separated vignette (coming soon).

3.1 Installation

The main version of the package is the one hosted in CRAN. You can install it like you would install any other package in R by using the function `install.packages()`. In your R console simply type:

```
# installation
install.packages("plspm")
```

Once `plspm` has been installed, you can use the function `library()` to load the package in your working session:

```
# load package 'plspm'
library("plspm")
```

3.2 Development Version

In addition to the stable version in CRAN, there is also a development version that lives in a github repository: https://github.com/gastonstat/plspm. This version will usually be the latest version that we're developing and that will eventually end up in CRAN. Most people don't need to use this version but if you feel tempted, intrigued, or adventurous, you are welcome to play with it. To download the *devel* version in R, you will need to use the package `"devtools"` —which means you have to install it first—. Once you installed `"devtools"`, type the following in your R console:

```
# load devtools
library(devtools)

# then download 'plspm' using 'install_github'
install_github("gastonstat/plspm")

# finally, load it with library()
library(plspm)
```

4 What's in plspm

plspm comes with a number of functions to perform a series of different types of analysis. The main function, which has the same name as the package, is the function plspm() which is designed for running a full PLS-PM analysis. A modified version of plspm() is its sibling function plspm.fit() which is intended to perform a PLS-PM analysis with limited results. In other words, plspm() is the deluxe version, while plspm.fit() is a minimalist option.

The accessory functions of plspm() are the plotting and the summary functions. The plot() method is a wraper of the functions innerplot() and outerplot() which allow you to display the results of the inner and outer model, respectively. In turn, the summary() function will display the results in a similar format like other standard software for PLS-PM.

In third place we have the function plspm.groups() which allows you to compare two groups (i.e. two models). This function offers two options for doing the comparison: a bootstrap t-test, and a non-parametric permutation test.

In fourth place, thanks to the collaboration of Laura Trinchera, there's the set of functions dedicated to the detection of latent classes by using **REBUS-PLS**.

Last but not least, plspm() also comes with several data sets to play with: satisfaction, mobile, spainfoot, soccer, offense, technology, oranges, wines, arizona, russett, russa, russb, and sim.data.

5 Quick Example with plspm()

In order to show a toy analysis example with plspm(), we will use the russett data set which is one of the traditional examples for presenting PLS-PM. To load the data, simply type in your R console:

```
# load data set
data(russett)
```

5.1 Data russett

The data conatins 11 variales about agricultural inequality, industrial development, and political instability measured on 47 countries, collected by Russett B. M. in his 1964 paper: The Relation of Land Tenure to Politics. *World Politics*, 16:3, pp. 442-454.

To get an idea of what the data looks like we can use the head() function which will show us the first n rows in russett

```
# take a look at the data
head(russett)
```

```
##           gini farm rent gnpr labo inst ecks death demostab demoinst dictator
## Argentina 86.3 98.2 3.52 5.92 3.22 0.07 4.06  5.38        0        1        0
## Australia 92.9 99.6 3.40 7.10 2.64 0.01 0.00  0.00        1        0        0
## Austria   74.0 97.4 2.46 6.28 3.47 0.03 1.61  0.00        0        1        0
```

```
## Belgium   58.7 85.8 4.15 6.92 2.30 0.45 2.20  0.69        1        0        0
## Bolivia   93.8 97.7 3.04 4.19 4.28 0.37 3.99  6.50        0        0        1
## Brasil    83.7 98.5 2.31 5.57 4.11 0.45 3.91  0.69        0        1        0
```

The description of each variable is given in the following table:

Table 1: Description of variables in data `russett`

Variable	Description	Block
`gini`	Inequality of land distribution	`AGRIN`
`farm`	Percentage of farmers that own half of the land	`AGRIN`
`rent`	Percentage of farmers that rent all their land	`AGRIN`
`gnpr`	Gross national product per capita	`INDEV`
`labo`	Percentage of labor force employed in agriculture	`INDEV`
`inst`	Instability of executive	`POLINS`
`ecks`	Number of violent internal war incidents	`POLINS`
`death`	Number of people killed as a result of civic group violence	`POLINS`
`demostab`	Political regime: stable democracy	`POLINS`
`demoinst`	Political regime: unstable democracy	`POLINS`
`dictator`	Political regime: dictatorship	`POLINS`

The proposed structural model consists of three latent variables: Agricultural Inequality (`AGRIN`), Industrial Development (`INDEV`), and Political Instability (`POLINS`). The model statement for the relationships between latent variables can be declared as follows:

> *The Political Instability of a country depends on both its Agricultural Inequality, and its Industrial Development.*

Besides the data, the other main ingredients that we need for running a PLS-PM analysis are: an inner model (i.e. structural model), and an outer model (i.e. measurement model). With other software that provide a graphical interface, the inner and the outer model are typically defined by drawing a path diagram. This is not the case with `plspm`. Instead, you need to define the structural relationships in matrix format, and you also need to specify the different blocks of variables. But don't be scared, this sounds more complicated than it is. Once you learn the basics, you'll realize how convenient it is to define a PLS path model for `plspm()`.

5.1.1 Path Model Matrix

The first thing to do is to define the inner model in matrix format. More specifically this implies that you need to provide the structural relationships in what we call a `path_matrix`.

To do this, you must follow a pair of important guidelines. The `path_matrix` must be a lower triangular boolean matrix. In other words, it must be a square matrix (same number of rows and columns); the elements in the diagonal and above it must be zeros; and the elements below the diagonal can be either zeros or ones. Here's how the path matrix should be defined:

```
# path matrix (inner model realtionships)
AGRIN = c(0, 0, 0)
INDEV = c(0, 0, 0)
POLINS = c(1, 1, 0)
rus_path = rbind(AGRIN, INDEV, POLINS)

# add optional column names
colnames(rus_path) = rownames(rus_path)

# how does it look like?
rus_path
```

```
##        AGRIN INDEV POLINS
## AGRIN      0     0      0
## INDEV      0     0      0
## POLINS     1     1      0
```

The way in which you should read this matrix is by "columns affecting rows". A number one in the cell i, j (i-th row and j-th column) means that column j affects row i. For instance, the one in the cell 3,1 means that `AGRIN` affects `POLINS`. The zeros in the diagonal of the matrix mean that a latent variable cannot affect itself. The zeros above the diagonal imply that PLS-PM only works wiht non-recursive models (no loops in the inner model).

We can also use the function `innerplot()` that allows us to quickly inspect the `path_matrix` in a path diagram format (and making sure it is what we want)

```
# plot the path matrix
innerplot(rus_path)
```

5.1.2 List of Blocks for Outer Model

The second ingredient is the outer model. The way in which the outer model is defined is by using a list. Basically, the idea is to tell the `plspm()` function what variables of the data set

are associated with what latent variables. Here's how you do it in R:

```
# list indicating what variables are associated with what latent variables
rus_blocks = list(1:3, 4:5, 6:11)
```

The list above contains three elements, one per each latent variable. Each element is a vector of indices. Thus, the first latent variable, `AGRIN`, is associated with the first three columns of the data set. `INDEV` is formed by the columns from 4 and 5 in the data set. In turn, `INDEV` is formed by the columns from 6 to 11.

Alternatively, you can also specify the list of `blocks` by giving the names of the variables forming each block:

```
# list indicating what variables are associated with what latent variables
rus_blocks = list(
  c("gini", "farm", "rent"),
  c("gnpr", "labo"),
  c("inst", "ecks", "death", "demostab", "demoinst", "dictator"))
```

By default, `plspm()` will set the measurement of the latent variables in reflective mode, known as *mode A* in the PLSPM world. However, it is a good idea if you explicitly provide the vector of measurement `modes` by using a character vector with as many letters as latent variables:

```
# all latent variables are measured in a reflective way
rus_modes = rep("A", 3)
```

5.2 Running `plspm()`

Now we are ready to run our first PLS path model with the function `plspm()`. You need to plug-in the data set, the `path_matrix`, the list of `blocks`, and the vector of `modes`, like this:

```
# run plspm analysis
rus_pls = plspm(russett, rus_path, rus_blocks, modes = rus_modes)

# what's in rus_pls?
rus_pls
```

```
## Partial Least Squares Path Modeling (PLS-PM)
## ---------------------------------------------
##    NAME             DESCRIPTION
## 1  $outer_model     outer model
## 2  $inner_model     inner model
## 3  $path_coefs      path coefficients matrix
## 4  $scores          latent variable scores
## 5  $crossloadings   cross-loadings
## 6  $inner_summary   summary inner model
```

```
## 7  $effects         total effects
## 8  $unidim          unidimensionality
## 9  $gof             goodness-of-fit
## 10 $boot            bootstrap results
## 11 $data            data matrix
## ---------------------------------------------
## You can also use the function 'summary'
```

What we get in `rus_pls` is an object of class `"plspm"`. Everytime you type an object of this class you will get a display with the previous list of results. For example, if you want to inspect the matrix of path coefficients, simply type:

```
# path coefficients
rus_pls$path_coefs
```

```
##            AGRIN      INDEV POLINS
## AGRIN  0.0000000  0.0000000      0
## INDEV  0.0000000  0.0000000      0
## POLINS 0.2150858 -0.6949622      0
```

Likewise, if you want to inspect the results of the inner model just type:

```
# inner model
rus_pls$inner_model
```

```
## $POLINS
##                Estimate Std. Error       t value     Pr(>|t|)
## Intercept -3.026106e-16 0.09263892 -3.266561e-15 1.000000e+00
## AGRIN      2.150858e-01 0.09749335  2.206159e+00 3.263963e-02
## INDEV     -6.949622e-01 0.09749335 -7.128304e+00 7.417367e-09
```

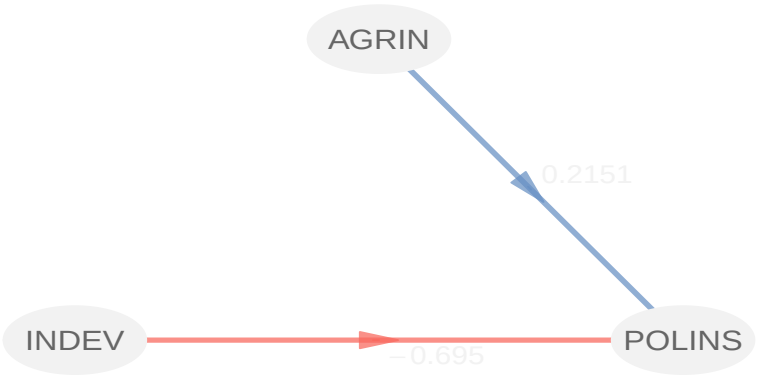
In addition, there is a `summary()` method that you can apply to any obect of class `"plspm"`. This function gives a full summary with the standard results provided in most software for PLS Path Modeling. We won't display the bunch of stuff that `summary()` provides but we recommend you to check it out in your computer:

```
# summarized results
summary(rus_pls)
```

5.3 Plotting results

One of the nice features about `plspm` is that you can also take a peek of the results using the function `plot()`. By default, this function displays the path coefficients of the inner model:

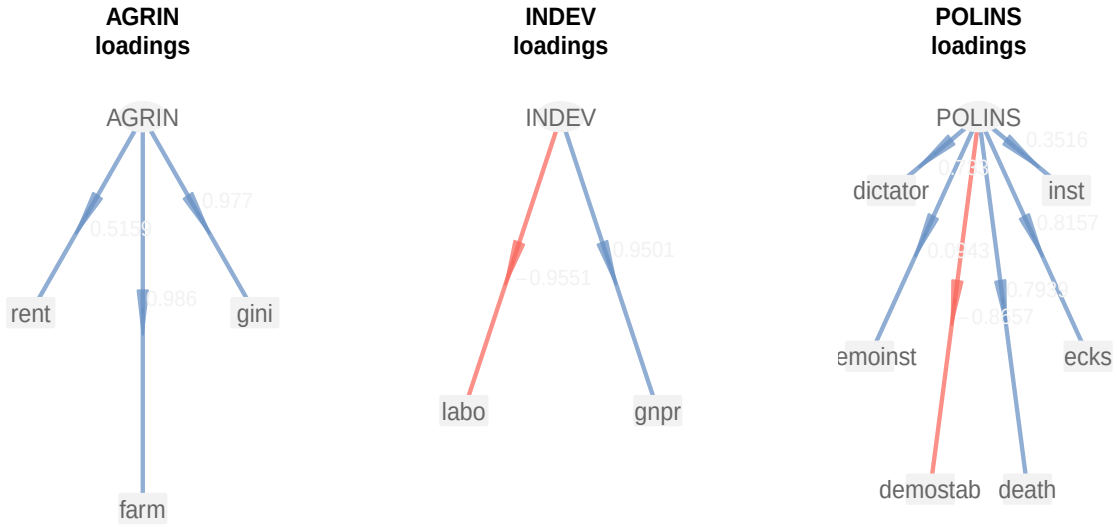
```
# plot the results (inner model)
plot(rus_pls)
```



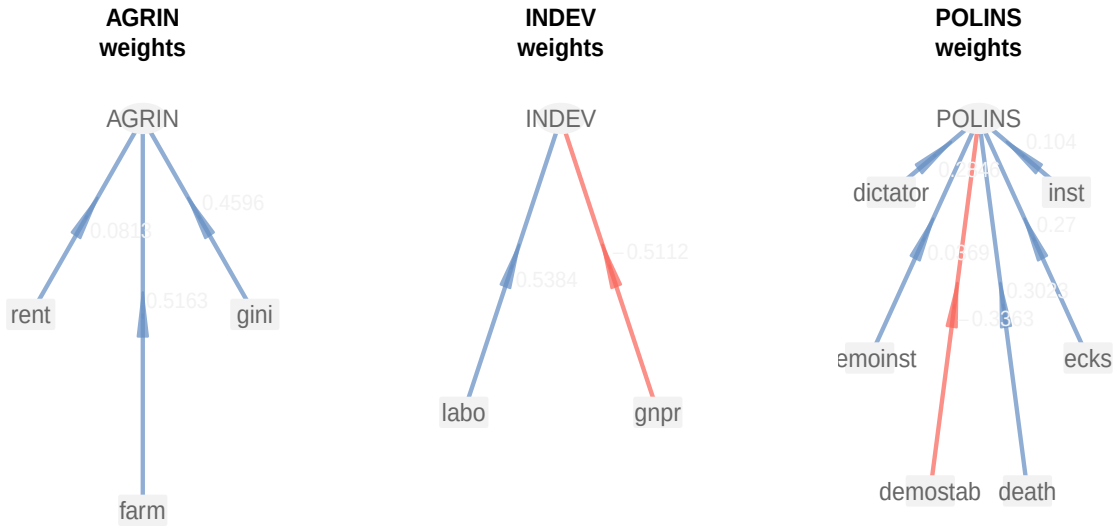
Equivalently, you can also use the function `innerplot()` to get the same plot.

In order to check the results of the outer model, say the loadings, you need to use the parameter `what` of the `plot()` function

```
# plot the loadings of the outer model
plot(rus_pls, what = "loadings", arr.width = 0.1)
```




```
# plot the weights of the outer model
plot(rus_pls, what = "weights", arr.width = 0.1)
```



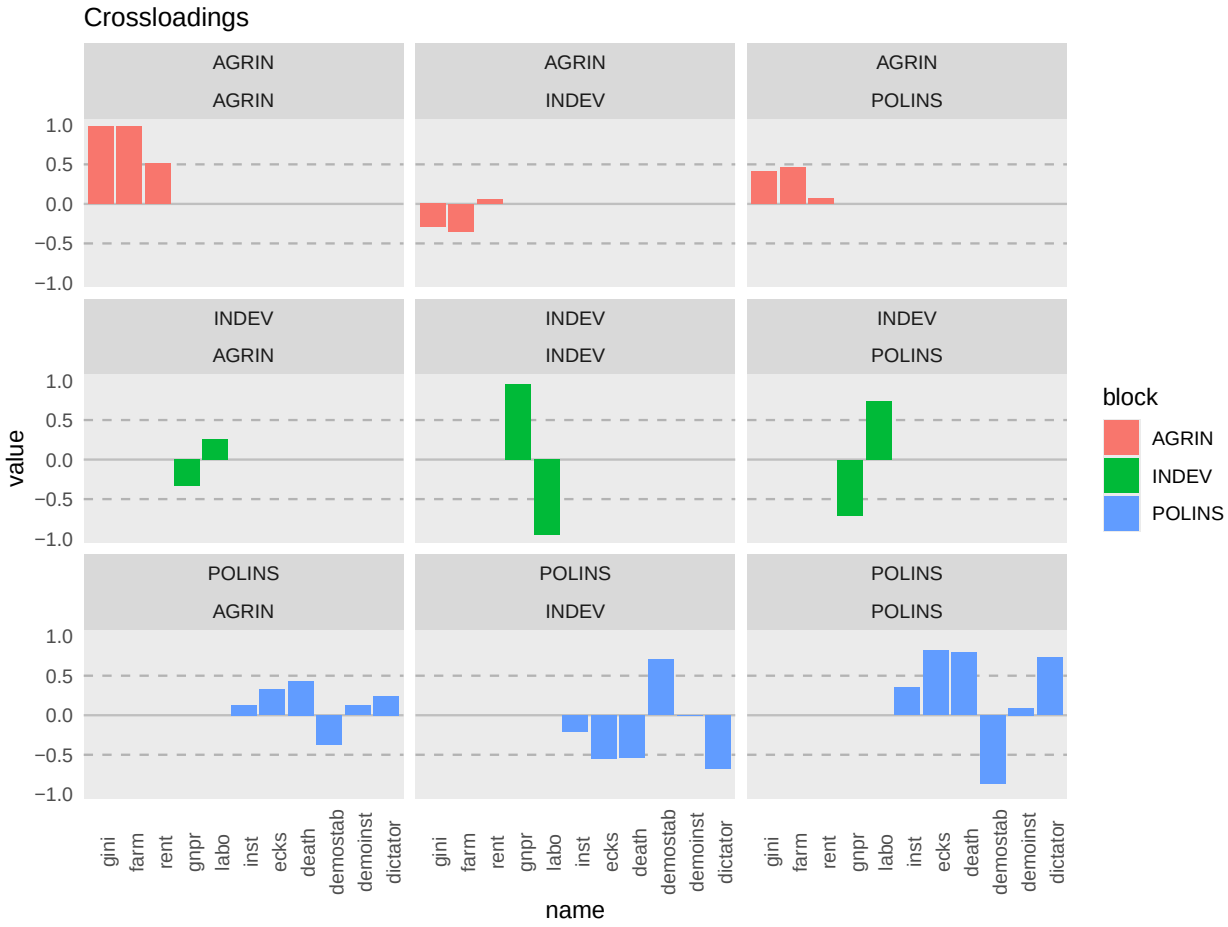
Plotting Cross-Loadings

In addition to the plotting functions provided in `plspm`, we can also use the packages `ggplot2` and `reshape` to get some nice bar-charts of the cross-loadings:

```
# load ggplot2 and reshape
library(ggplot2)
library(reshape)

# reshape crossloadings data.frame for ggplot
xloads = melt(rus_pls$crossloadings, id.vars = c("name", "block"),
              variable_name = "LV")

# bar-charts of crossloadings by block
ggplot(data = xloads,
       aes(x = name, y = value, fill = block)) +
  geom_hline(yintercept = 0, color = "gray75") +
  geom_hline(yintercept = c(-0.5, 0.5), color = "gray70", linetype = 2) +
  geom_bar(stat = 'identity', position = 'dodge') +
  facet_wrap(block ~ LV) +
  theme(axis.text.x = element_text(angle = 90),
        line = element_blank()) +
  ggtitle("Crossloadings")
```



6 Non-Metric PLS-PM

The **non-metric plspm** approach is a quantification procedure that provides "raw" manifest variables with a proper metric interval scale, independently of the measurement scale on which they have been originally observed. Each level (or distinct value) of a manifest variable is considered as a scaling parameter to optimize together with the model parameters. Scaling and model parameters (that is, quantifications and outer weights) are optimized through a modified plspm iterative algorithm. Given any combination of `mode` and `scheme`, whereas the plspm algorithm has been proved to yield outer weights that maximize a well defined criterion, **nmplspm** outer weights and quantifications maximize the same criterion. nmplspm quantifications may be subjected to different constraints; these contraints depend on which properties of the original measurement scale are retained in the quantification procedure. The user can choose the proper type of quantification for each manifest variable.

6.1 How to run a non-metric `plspm` analysis

To run a non-metric `plspm` analysis you must use the `scaling` argument of the function `plspm()` function. This argument requires a list of string vectors as long as the the list provided to the argument `blocks`. Each vector in `scaling` contains a string for each manifest

variable of a block, which indicates the type of quantification applied to the corresponding manifest variable. Currently the scaling option of the plspm function supports four types of quantifications:

- `"raw"`: the (numerical) manifest variable is not quantified. Choosing this option for every variable is equivalent to run the ordinary plspm on original variables. This is the only option which does not yield transformed variables scaled to unitary variance.
- `"num"`: a linear transformation is applied to the (numerical) manifest variable. Choosing this option for every variable is equivalent to running the ordinary plspm algorithm on scaled variables.
- `"ord"`: a monotonic transformation is applied to the manifest variable, subject to the following constraint: given two distinct attributes, the higher order attribute is quantified by a numerical value not smaller than the other. This transformation is suggested for quantifying the levels of the ordinal variables while respecting their order. It can be applied also to numerical variables for checking and overcoming non-linearity issues. In this case linearity hypothesis is relaxed in favor of a milder monotonicity hypothesis.
- `"nom"`: a non-monotonic transformation is applied to the manifest variable, subject to the following constraint: observation sharing the same attribute share the same quantification value. This transformation is recommended for quantifying nominal variables. It can be used also for exploring non-monotonic relations of an ordinal or even a numerical variable if the number of distinct values is much less then the sample size.

6.2 Running a non-metric analysis on Russett dataset

We run our analysis on the `russb` dataset. To load the data, type in your R console:

```
# load data set
data(russb)
```

This dataset contains the same information than the dataset `russett`, except that the three binary variables `demostab`, `demoinst` and `dictator` become three categories of the factor `demo`:

```
# take a look at the data
head(russb)

##           gini farm rent gnpr labo inst ecks death     demo
## Argentina 86.3 98.2 3.52 5.92 3.22 0.07 4.06  5.38 unstable
## Australia 92.9 99.6 3.40 7.10 2.64 0.01 0.00  0.00   stable
## Austria   74.0 97.4 2.46 6.28 3.47 0.03 1.61  0.00 unstable
## Belgium   58.7 85.8 4.15 6.92 2.30 0.45 2.20  0.69   stable
## Bolivia   93.8 97.7 3.04 4.19 4.28 0.37 3.99  6.50 dictator
## Brasil    83.7 98.5 2.31 5.57 4.11 0.45 3.91  0.69 unstable
```

According to Tenenhaus (1998), variables `rent`, `gnpr`, `labo`, `inst`, `ecks` and `death` need

to be transformed in order to overcome non-linearity issues. We apply an ordinal (monotonic) transformation to these variables. Moreover, we apply a nominal (non-monotonic) quantification to the categorical variable `demo`. Finally, variables `gini` and `farm` are simply standardized to unitary variance.

We create the lists `russb_scaling` and `russb_blocks` to fill the argument `scaling` and `blocks` respectively:

```
# defining the quantification constraints for each manifest variable
russb_scaling = list(c("num", "num", "ord"),
                     c("ord", "ord"),
                     c("ord", "ord", "ord", "nom"))

# defining the blocks of manifest variables
russb_blocks = list(
   c("gini", "farm", "rent"),
   c("gnpr", "labo"),
   c("inst", "ecks", "death", "demo"))
```

We have previously created the objects `rus_modes` and `rus_path`to define the modes and the relations in the inner model, so we are ready to run the PLS path model with the function `plspm()`.

Finally, we call `plspm()`

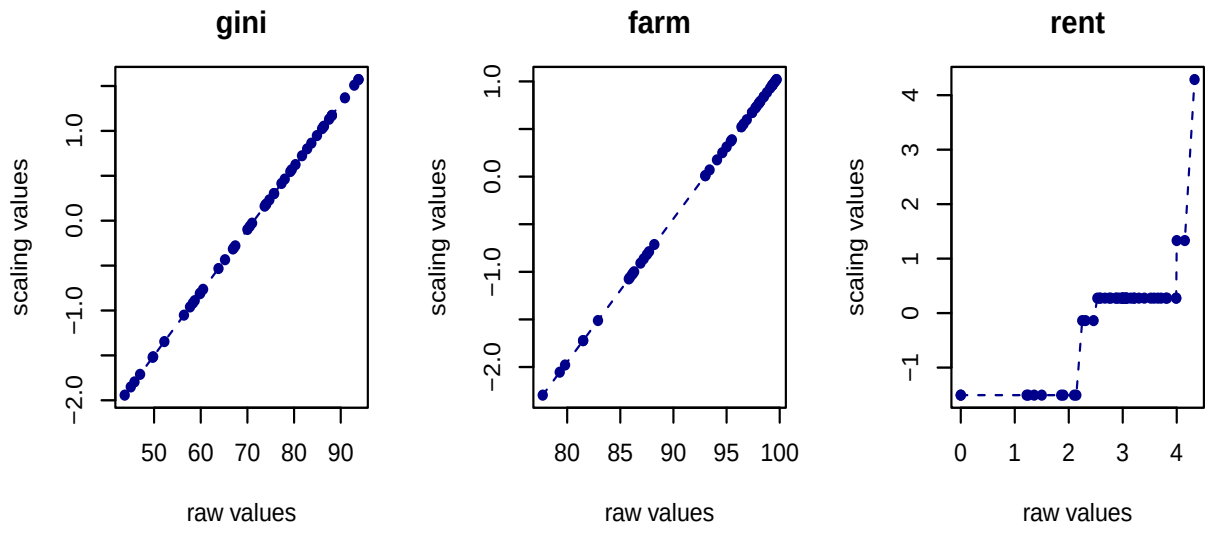
```
# running the model
russb_nmpls = plspm(russb, path = rus_path, blocks = russb_blocks,
              scaling = russb_scaling, modes = rus_modes,
              scheme = "CENTROID")
```

To obtain the quantified variables:

```
head(russb_nmpls$manifests)
```

```
##                gini       farm       rent       gnpr       labo       inst
## Argentina  1.0459001  0.7937942  0.2748046 -0.6544187 -0.1630727  0.1971539
## Australia  1.5089203  1.0047460  0.2748046  1.8677907 -1.5578828 -1.1711005
## Austria    0.1829989  0.6732503 -0.1374453  0.6023708 -0.1630727 -1.1711005
## Belgium   -0.8903660 -1.0746358  1.3317605  0.7289698 -1.5578828  0.9127150
## Bolivia    1.5720594  0.7184543  0.2748046 -1.7123359  0.8782500  0.9127150
## Brasil     0.8634983  0.8389982 -0.1374453 -0.6544187  0.8123611  0.9127150
##                ecks       death       demo
## Argentina  1.117860  0.95685251  0.1974951
## Australia -1.883948 -1.14731402 -1.3968222
## Austria   -0.542470 -1.14731402  0.1974951
## Belgium   -0.185364 -0.06412741 -1.3968222
## Bolivia    1.117860  1.61143892  0.9291196
## Brasil     1.117860 -0.06412741  0.1974951
```

```
par(mar = rep(0,4))
quantiplot(russb_nmpls, lv = "AGRIN")
```




```
par(op)
```