Composite Indicator Analysis and Optimization (CIAO) Tool

Codebook for practitioners

November 2018

David Lindén[1]

Dr. Marco Cinelli[2,3,*]

Dr. Matteo Spada[4,2]

Dr. William Becker[5]

Dr. Peter Burgherr [4,2]

[1]*Department of Sustainable Development, Environmental Science and Engineering, KTH Royal Institute of Technology, Stockholm, Sweden*

[2]*Future Resilient Systems (FRS), Swiss Federal Institute of Technology (ETH) Zürich, Singapore-ETH Centre (SEC), Singapore*

[3]*Institute of Computing Science, Poznań University of Technology, Poznań, Poland*

[4]*Technology Assessment Group, Laboratory for Energy Systems Analysis, Paul Scherrer Institute (PSI), Villigen PSI, Switzerland*

[5]*European Commission, Joint Research Centre (JRC), Ispra VA, Italy*

[*]Corresponding author: Marco Cinelli; email: marco.cinelli@put.poznan.pl

Cite as: Lindén, D., M. Cinelli, M. Spada, W. Becker, and P. Burgherr. 2018. *Composite Indicator Analysis and Optimization (CIAO) Tool.*

Program description

The Composite Indicator Analysis and Optimization (CIAO) tool is an automated menu-version of the Matlab® toolbox presented by Becker (2017) for the advanced assessment of Composite Indicators (CIs). It was developed in connection to assessing the implicit weights of a novel index – the Electricity Supply Resilience Index (ESRI), developed within the Future Resilient Systems (FRS) program, at the Singapore-ETH Centre (SEC); see Gasser et al. (2017) and Lindén (2018) for further details. The scripts for the CIAO tool are written in MathWorks's (http://www.mathworks.com) commercial software language, Matlab®, and contain five main steps: (1) a correlation analysis, (2) a weighting and aggregation of the indicators, (3) an estimation of the influence of each indicator (Si), (4) a decomposition of such influence, and (5) an optimization of the indicators' weights.

Thus, the CIAO tool allows users to:

1. Perform a detailed examination of the linear and nonlinear relationships among (i) the set of indicators and (ii) between the indicators and the Composite Indicator (CI).
2. Assess how the indicators are "balanced" within the CI, i.e. to what extent the influence of each indicator matches its assigned weight.
3. Tune the weights of the indicators so that the influence of each one matches pre-defined importance values.

Details on the methodology are presented in Becker et al. (2017) and Lindén (2018).

Please NOTE that two additional commercial toolboxes are required to run the full analysis: The Statistics and Machine Learning Toolbox[1], and the Optimization Toolbox[2]. Also, this package uses one freely available script from the MATLAB Central/File Exchange: *Random Vectors with Fixed Sum*[3]. For this analysis, it is used for generating random starting points for the weight-optimization.

This codebook presents the structure of the package to run the CIAO tool and provides details on each script.

[1] URL: https://se.mathworks.com/products/statistics.html
[2] URL: https://se.mathworks.com/products/optimization.html
[3] Author: Roger Stafford, 2006 - URL: https://se.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum

Scripts

A flowchart of the CIAO tool is shown in Fig. 1. The scripts run in the tool are described in the following sections.

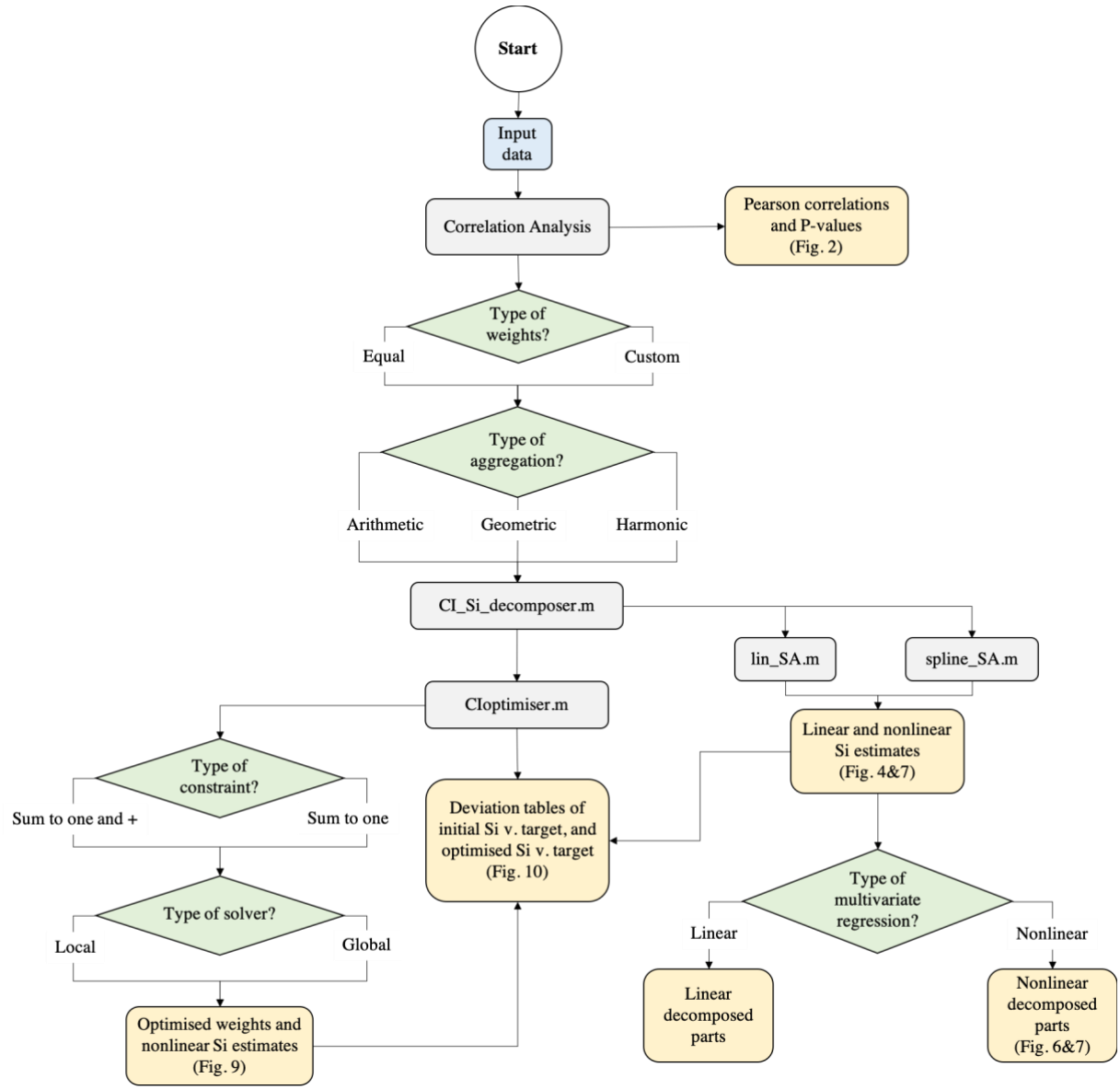

Fig. 1. Flow of the menu-version for implicit weights analysis and optimization in CIs; color coded accordingly: input (blue), outputs (orange), functions (grey), menu choices (green).

MENU_version.m

MENU_version.m is the main script of this package. By running this script, a user-friendly GUI menu is initiated. Although most sub-function settings are automated, the GUI allows the user to "click" through the analysis with a certain amount of freedom to choose different settings (see the green menu choices in Fig. 1). For an explanation on what the script needs to run, see "Input" section at page 9. Default setting loads *test_data.m*, which is a 58-by-4 matrix (see Input section). Then, the main script automatically generates a correlation matrix (with Pearson correlation coefficients) and its corresponding p-values for the matrix of indicators (Fig. 2).

```
17      %% Check correlation and P-values
18
19 -    [R,P] = corrcoef(X);
20 -    fprintf('Correlation matrix >>\n')
21 -    corr=array2table(round(R,4),'RowNames',rownamez, 'VariableNames', varnamez);
22 -    disp(corr)
23 -    fprintf('P-values >>\n')
24 -    pval=array2table(round(P,4),'RowNames',rownamez, 'VariableNames', varnamez);
25 -    disp(pval)
```

Command Window

```
Correlation matrix >>
            X_1       X_2       X_3       X_4
            ______    ______    ______    ______

    X_1         1    0.6891    0.7286    0.4145
    X_2    0.6891         1    0.8224    0.6249
    X_3    0.7286    0.8224         1    0.5997
    X_4    0.4145    0.6249    0.5997         1

P-values >>
            X_1       X_2    X_3       X_4
            ______    ___    ___    ______

    X_1         1    0      0      0.0012
    X_2         0    1      0           0
    X_3         0    0      1           0
    X_4    0.0012    0      0           1
```

Fig. 2. Output from the correlation analysis, which is obtained automatically by running the script. Note that default setting is to name variables X_i, where i=1,2,…,n.

Subsequently, the user is provided with the choice of what weights to apply to the indicators (either equal or custom) and also asked to choose an aggregation function (either arithmetic "ArAv", geometric "GeAv", or harmonic average "HarAv"); see Fig. 3. NOTE: The showcased results are obtained using the default data (*test_data.m*), aggregated with arithmetic average and equal weights.

Fig. 3. The GUI menu for the selection of weights (left) and the selection of aggregation function (right).

CI_Si_decomposer.m

CI_Si_decomposer.m is a function called by the main script *MENU_version.m*. It uses the X and y data to provide the first order sensitivity indices (i.e. influence of each indicator, Si), plus their decomposed values into correlated and uncorrelated contributions. The function runs accordingly:

function outSi = CI_Si_decomposer(X, y, plott)

The input parameters are the matrix of indicators (X) and the vector (y), which is the aggregated CI. Additionally, the choice of plotting the regression models (*plott=1*) or not (*plott=0*) can be made; *plott=1* gives scatter plots with splines and linear regression fits (Fig. 4).

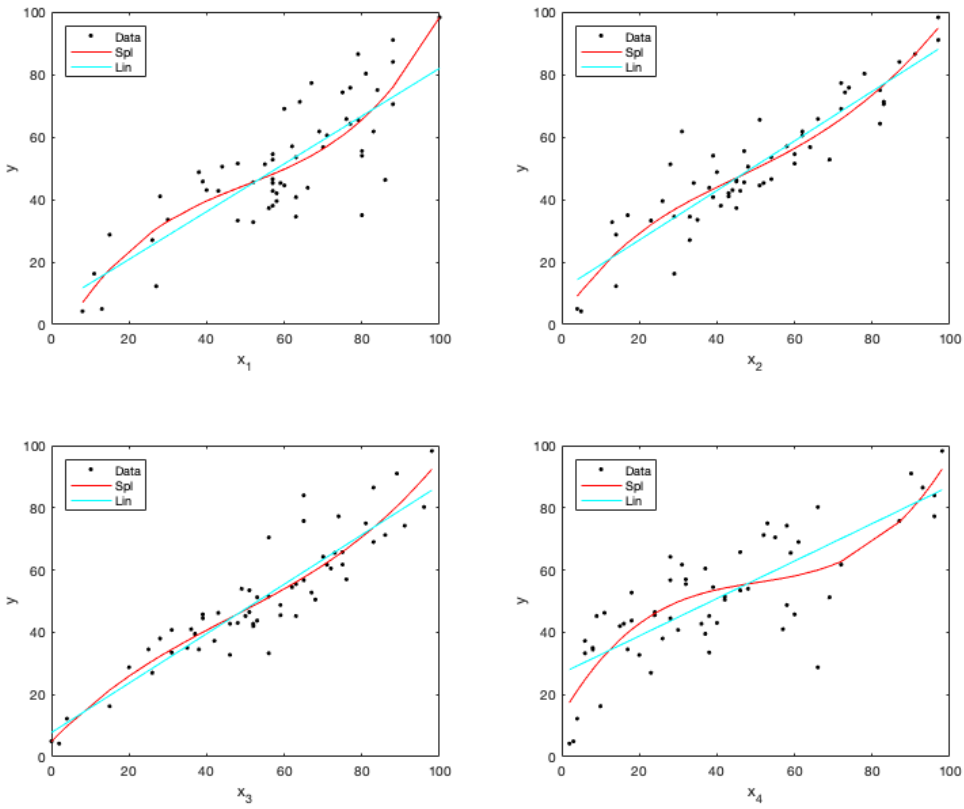

Fig. 4. Output from the regression analysis. Plots showing linear (cyan) and the nonlinear splines (red) regression models.

By running this function, three steps are performed:

First, the function calls two sub-functions for the estimation of the first order sensitivity indices (Si):

function [R2, par] = lin_SA(y, X, plott)

this calculates R^2 (R2), i.e. the linear estimates of Si, for the X and y data. The linear regression parameters (e.g. the predicted outcome variables) are stored in a structured array (*par*) with two fields, including: the predicted outcome values of the regression model (*par.yhats*), and X-values sorted in ascending order (*par.xsrts*). Also,

function [S_E, S_V, par] = spline_SA(y, X, plott, p, knots)

this uses the X and y data to calculate the Si, using nonlinear (splines) regression. The *CI_Si_decomposer.m* uses a cubic order of the spline (p=3) and a row vector of knot numbers to try (knots = [1 2 3 4 5]) as standard, but this can be altered if desired. The outputs are a column vector of first order sensitivity indices estimated via E(y|Xi) (S_E) or V(y|Xi) (S_V). The splines regression parameters are stored in a structured array (*par*) with eight fields, including: order of spline (*par.p*), trial knots (*par.trialknots*), the predicted outcome and variance values of the regression model (*par.yhats & par.varhats*), number of knots (*par.knotN*), derivatives (*par.derivs & par.derivs_xx*), and X-values sorted in ascending order (*par.xsrts*).

Second, a multivariate regression of Xi on X-i (representing the matrix X without the ith column, i.e. indicator) is carried out, using either a standard linear model, according to the method of Xu and Gertner (2008), or MATLAB's inbuilt Gaussian process regression model

(*fitrgp*)[4] for a nonlinear model. The user is free to choose between these two models, as shown in Fig. 5.

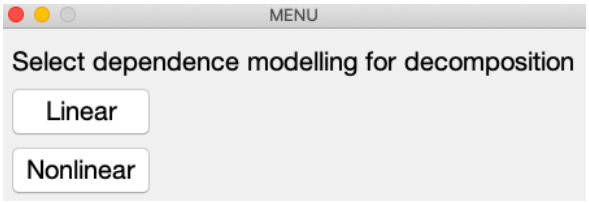

Fig. 5. The GUI for the selection of model to perform the multivariate regression.

The results from the multivariate regression are then used to obtain a residual (i.e. what is "left" after correlation is removed). Subsequently, a nonlinear regression (splines) of y on the residuals of previous regression is performed to obtain the uncorrelated part (SU). Finally, the correlated part (SC) is obtained simply by subtracting the uncorrelated from the Si, i.e. SC=Si-SU. These are the decomposed values, which are then plotted in a bar chart (Fig. 6).

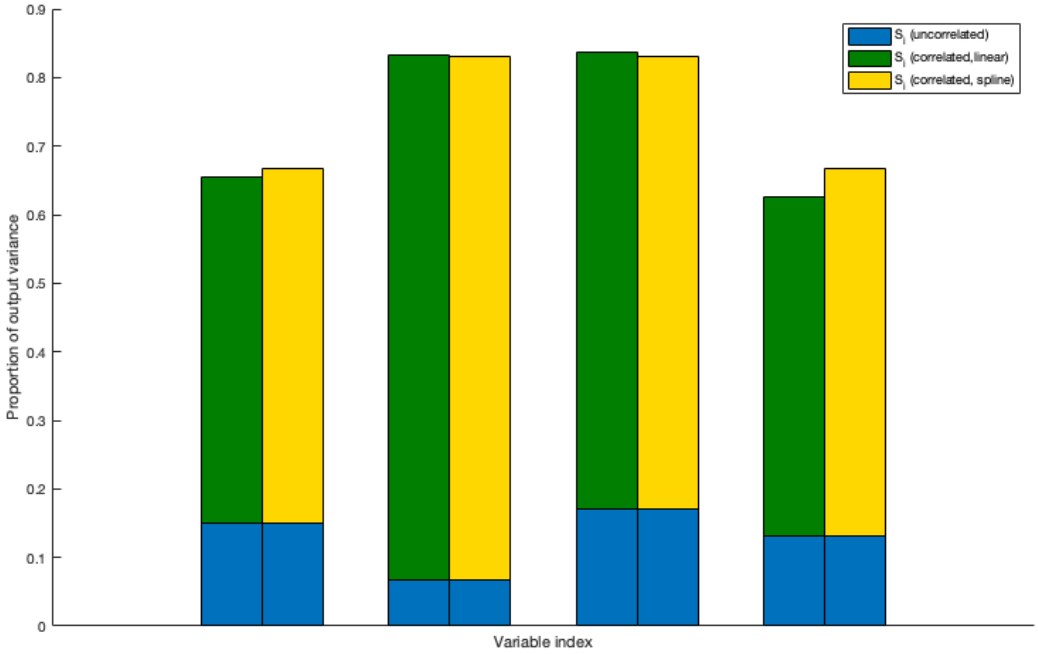

Fig. 6. The decomposed Si values; full bars represent the Si values estimated in the previous regression analysis. The uncorrelated parts (blue), estimated with nonlinear multivariate regression, and the resulting correlated parts, from either the linear (green) and nonlinear (yellow) regression results.

After these steps have been completed, the output of the *CI_Si_decomposer.m* is, beside the two plots, a structure array (*outSi*) with 3 fields containing: (1) linear regression results (*outSi.lin*), (2) spline regression results (*outSi.spline*), and (3) a summarized table of both estimates and the decomposed values (*outSi.results*), shown in Fig. 7.

[4] NOTE: for this nonlinear dependence modelling, the Statistics and Machine Learning toolbox is required. If this is not detected, linear dependence modelling will be used.

```
%% Estimation and Decomposition of influence

decomp_choice=menu('Select dependence modelling for decomposition','Linear','Nonlinear');
if decomp_choice==1
    outSi=CI_Si_decomposer(X,y,1,decomp_choice);
    fprintf('Si (lin and nonlin) and decomposed parts (linear) >>\n')
    disp(outSi.results) % display results-table with all results
elseif decomp_choice==2
    outSi=CI_Si_decomposer(X,y,1,decomp_choice);
    fprintf('Si (lin and nonlin) and decomposed parts (nonlinear) >>\n')
    disp(outSi.results) % display results-table with all results
end
```

Command Window

```
Si (lin and nonlin) and decomposed parts (nonlinear) >>
              S_L       S_Sp       SU_L       SU_Sp       SC_L       SC_Sp
              ____      ____       ____       ____        ____       ____

    X_1     0.65455   0.66823    0.15051    0.15051     0.50404    0.51772
    X_2      0.8324   0.83133   0.067245   0.067245     0.76515    0.76408
    X_3     0.83636   0.83085    0.17052    0.17052     0.66584    0.66034
    X_4     0.62556   0.66625    0.13171    0.13171     0.49385    0.53455
```

Fig. 7. Output from the estimation and decomposition of influence. These are the values used for the bar chart in Fig. 6 above. S_L and S_Sp are the full bar values, i.e. the influence (Si), estimated with linear or nonlinear (splines) regression. The SU and SC are the uncorrelated and correlated parts, respectively.

CIoptimiser.m

CIoptimiser.m is a self-contained function which optimizes CI weights according to the "target importance" of each indicator, using nonlinear regression (penalized splines). It requires two choices (Fig. 8) prior to initiating the optimization algorithm[5].

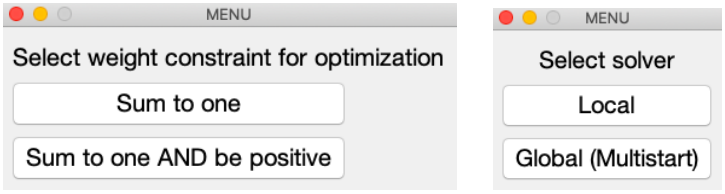

Fig. 8. The GUI for the choice of optimization constraint (left) and solver (right).

First, a constraint on the weights must be chosen. The weights are either only constrained to sum to one (meaning they can take on negative values), OR they must sum to one but also be positive. Second, the choice is between two different types of solver, either a local or a global one. Although both are based on the same search method, the Nelder-Mead simplex, they differ in two aspects: (a) staring point(s) and (b) number of runs. The local solver uses the input weights as starting point, running the search algorithm once. Conversely, the global solver (MultiStart[6]) offers an option to specify the quantity of randomly generated starting points, re-running the search method for each one. This might be desired if the local solver fails to converge to an optimal solution[7]. However, for most cases, the local solver is sufficient and is thus recommended. After these choices are made, the function runs accordingly:

function outOpt = CIoptimiser(X, Sd, constr, CIeq)

this function uses the X data to find the optimal set of weights according to a predefined "target" (*Sd*), with a constraint defined by (*constr*); *constr=1* represents the sum-to-one constraint, and

[5] NOTE: for constrained optimization, Optimization Toolbox is required. If this is not detected, will automatically revert to unconstrained.
[6] For a more detailed overview on this solver, see https://se.mathworks.com/help/gads/multistart.html
[7] For more info on this topic, see https://se.mathworks.com/help/optim/ug/local-vs-global-optima.html

constr=2 represents the sum-to-one constraint together with a positive value constraint. CIeq is the aggregation form (e.g. 'ArAv' for arithmetic) of the CI and it is set automatically by the *MENU_version.m* script according to which function was chosen for aggregation at the beginning (Fig. 9). The output of this function is a structured array (*outOpt*) with six fields containing, for example: the optimized weights (*outOpt.wopt*), Si at optimized weights (*outOpt.Sopt*) and some other solver-specific parameters.

```
81      %% Optimization
82
83 -    w_choice=menu('Select weight constraint for optimization','Sum to one','Sum to one AND be positive');
84 -    if aggregation==1
85 -        outOpt=CIoptimiser(X,w,w_choice,'ArAv');
86 -    elseif aggregation==2
87 -        outOpt=CIoptimiser(X,w,w_choice,'GeAv');
88 -    elseif aggregation==3
89 -        outOpt=CIoptimiser(X,w,w_choice,'HarAv');
```

```
Command Window
Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the optimality tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.

<stopping criteria details>

Optimised weights >>
    0.4443
    0.0782
    0.1002
    0.3773

Si at optimised weights >>
    0.7207
    0.7207
    0.7207
    0.7207
```

Fig. 9. Output from weight-optimization, using the sum to one AND positive constraint. As seen from the stopping criterion, a local solver is used. Judging by the results, it successfully finds an optimal set of weights that achieves an equal Si.

- After all the steps are carried out, the main script *MENU_version.m* automatically compiles two summarizing tables (Fig. 10), which showcase the discrepancy between initial Si values and the target values (weights), and optimized Si values and the target values (weights). For this purpose, both sets of Si values are first normalized to render them comparable to the weights (summing to one).

```
Command Window
Normalised Si at initial weights >>
          Sis_norm    Sis_target    Sis_dev     Dev_ratio
          ________    __________    ________    _________

    X_1   0.22299     0.25          -0.02701    '-11%'
    X_2   0.27742     0.25           0.02742    ' 11%'
    X_3   0.27726     0.25           0.02726    ' 11%'
    X_4   0.22233     0.25          -0.02767    '-11%'

Normalised Si at optimised weights >>
          Sisopt_norm    Sis_target    Sisopt_dev    Dev_ratio
          ___________    __________    __________    _________

    X_1   0.25           0.25          0             '0%'
    X_2   0.25           0.25          0             '0%'
    X_3   0.25           0.25          0             '0%'
    X_4   0.25           0.25          0             '0%'

fx >>
```

Fig. 10. Summarizing tables of the results, comparing both the Si values at initial weights (above) and Si values at optimized weights (below) with reference to the target values (i.e. weights), in this case equal.

Input

A matrix (X), with *N*-alternatives (rows) by *k*-variables (columns). Default setting loads *test_data.m*, which is a 58-by-4 matrix (58 alternatives, 4 indicators) containing data that is normalized to a 0-100 scale. To add your own dataset from, for example, an excel spreadsheet, simply use MATLAB's inbuilt import tool[8]. Follow the checklist below to confirm that data is imported correctly (format, name, etc.).

Checklist:

1) First, create/prepare an excel-file (*.xlsx*) with the input data. NOTE: The data must be normalized[9] prior to importing it. Example of a correct dataset can be found here.
2) When importing the data, set the "output type" to be a numeric matrix (Fig. 11).

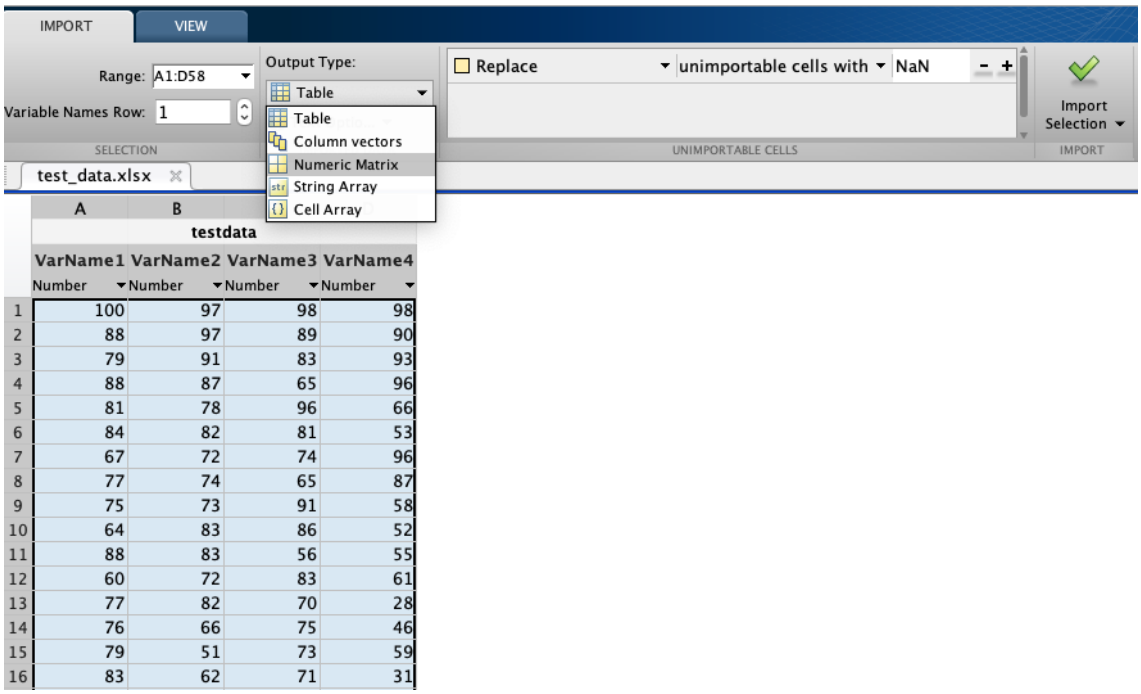

Fig. 11. Screenshot representing the choice of output types. Set to Numeric Matrix.

3) Be sure to assign your imported matrix to the global variable "X", as this is fundamental for running the scripts. This is easily done by right-clicking on the matrix name in the workspace and selecting "Rename" (Fig. 12).
4) Save your imported X-matrix as a *.mat* file in the same folder as the scripts (Fig. 12).

[8] For more info on the import tool in MATLAB see for example, https://se.mathworks.com/help/matlab/ref/importtool-app.html

[9] Examples of appropriate normalization methods, (a) Min-Max (b) Target (c) Ranking. Avoid negative values. If standardization is used, it requires that all values are brought to a positive scale. This can be done by, for example, finding the largest negative value in the data set and adding its absolute form to all values (e.g. X + abs(min(X))). This ensures that the minimum value is zero. For more details on normalization methods, see OECD (2008).

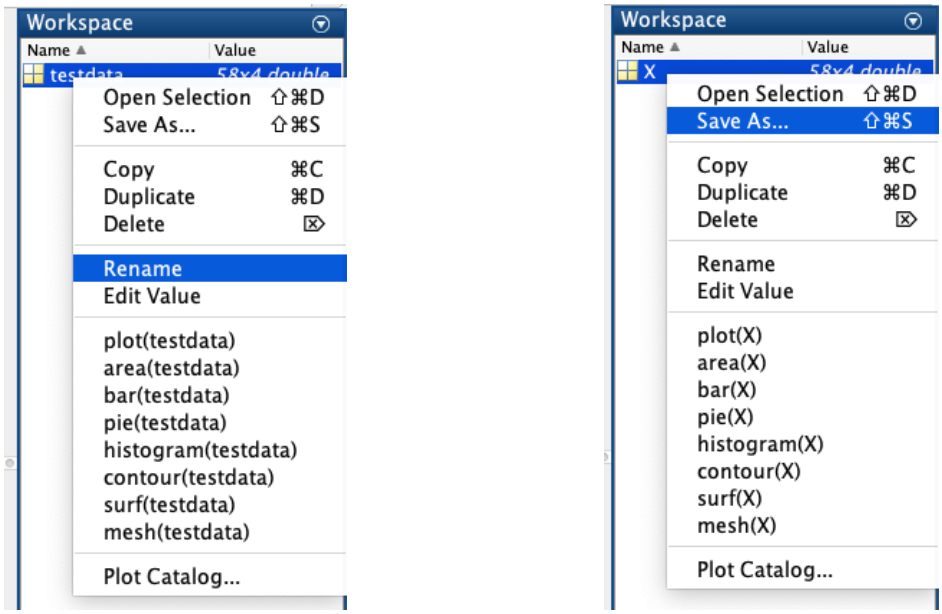

Fig. 12. Screenshots representing the renaming of the imported matrix (left). Then, when renamed/assigned to the global variable X (right), select Save As… and save as a *.mat* file.

5) In the *MENU_version.m* script, simply replace the "load *test_data"* (Fig. 13) with the name that your data is saved as.

```
%% Load input data

load test_data    % load X matrix containing indicator data
```

Fig. 13. The line in the script *MENU_version.m* that loads the X matrix for analysis.

6) Now, everything should be set for running the analysis on your data.

Output

Pearson correlations, linear and nonlinear regression estimates (Si), decomposed values (SU and SC), optimized Si and weights (wopt), and discrepancy tables.

Acknowledgments

The research was conducted at the Future Resilient Systems (FRS) at the Singapore-ETH Centre (SEC), which was established collaboratively between ETH Zürich and Singapore's National Research Foundation (FI 370074011) under its Campus for Research Excellence And Technological Enterprise (CREATE) program. This study has also been supported by the Technology Assessment Group of the Laboratory for Energy Systems Analysis at the Paul Scherrer Institute (PSI) in Switzerland. Marco Cinelli acknowledges that this project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 743553.

References

Becker, W. 2017. Matlab toolbox for analysis and adjustment of weights in composite indicators. European Commission, Joint Research Centre, http://dx.doi.org/10.13140/RG.2.2.27035.46883.

Becker, W., M. Saisana, P. Paruolo, and I. Vandecasteele. 2017. Weights and importance in composite indicators: Closing the gap. Ecological Indicators **80**:12-22.

Gasser, P., P. Lustenberger, T. Sun, W. Kim, M. Spada, P. Burgherr, S. Hirschberg, and B. Stojadinović. 2017. Security of electricity supply indicators in a resilience context. Pages 1015-1022 European Safety and Reliability Conference. 2017 Taylor & Francis Group, London, ISBN 978-1-138-62937-0.

Lindén, D. 2018. Exploration of implicit weights in composite indicators: the case of resilience assessment of countries' electricity supply. KTH Royal Institute of Technology, KTH DIVA.

OECD. 2008. Handbook on constructing composite indicators: Methodology and user guide. OECD publishing, Paris.

Xu, C., and G. Z. Gertner. 2008. Uncertainty and sensitivity analysis for models with correlated parameters. Reliability Engineering & System Safety **93**:1563-1573.