Documentation of ModelSolver

A Python class for analysing dynamic algebraic models

Magnus Kvåle Helliesen

In the series Documents, documentation, method descriptions, model descriptions and standards are published.

Symbols in tables	Symbol
Category not applicable	.
Figures do not exist at this time, because the category was not in use when the figures were collected.	
Not available	..
Figures have not been entered into our databases or are too unreliable to be published.	
Confidential	:
Figures are not published to avoid identifying persons or companies.	
Decimal punctuation mark	.

Preface

Statistics Norway is in the process of moving from proprietary software run on own servers to open-source languages run on Google Cloud. Therefore, we need to translate all programs into Python and/or R code. The national accounts division make use of an input/output model in compiling the monthly and quarterly national accounts. There does not seem to exist any package for this kind of models in either Python or R, therefore Statistics Norway has developed such a package inhouse. Documentation of the package has previously been published and will be maintained on GitHub.

Statistics Norway, February 16, 2024

Lasse Sandberg

Abstract

This paper documents the Python class `ModelSolver`. `ModelSolver` lets the user define a model object in terms of equations and endogenous variables. It contains methods to solve the model subject to data (in a `Pandas DataFrame`), as well as analysing the model using graph theory and network plots.

What sets `ModelSolver` apart from other similar solvers is that it does not require the equations of the model to be written in any particular way, or that the user associates equations with endogenous variables. Most other solvers require that either 1) the model is normalised (i.e., that the model is written in terms of endogenous variables), or 2) that the user explicitly associates equations with endogenous variables. This is non-trivial for models with lots of equations. `ModelSolver`, however, reads equations in whatever form they may be written, and performs the necessary analyses, without any input from the user other than lists of equations and endogenous variables.

`ModelSolver` was developed to facilitate solving an input-output model for the Norwegian monthly national accounts. [1] It analyses and solves the model's more than 15,500 equations over more than 30 periods in under a minute on a laptop computer.

---

[1]A detailed inventory of the Norwegian monthly national accounts can be found here: https://www.ssb.no/nasjonalregnskap-og-konjunkturer/nasjonalregnskap/artikler/quarterly-national-accounts.

Table of contents

Model classes/features	
Supported	**Not supported**
Linear, non-linear	
Static, dynamic [2]	
Explicit, implicit	
Discrete	Continuous
Deterministic	Stochastic

Table 1.1   Model classes/features supported/not supported by `ModelSolver`

1.   Introduction

The national accounts (NA) in Statistics Norway (SSB) make use of an input-output (I/O) model in the production of the monthly (and quarterly) national accounts (MNA/QNA). At present, the NA use a proprietary software package to define, analyse and solve the model. As a part of SSB's IT strategy, we are in the process of moving away from proprietary software towards open source languages, such as `Python` and `R`. A challenge that arose is that there does not seem to exist any packages in either language that meet the NA's particular needs with regards to the I/O model. Therefore, I started exploring the possibility to develop such a package myself—this work, which begun around 2021/2022, resulted in the `Python` package `ModelSolver`. At present, `ModelSolver` contains all the necessary methods to handle the NA's I/O model, but new features will be added to the package intermittently.

Although `ModelSolver` was developed in order to meet the particular needs of SSB's MNA/QNA, the package can handle a variety of models. Table 1.1 lists what model classes/features `ModelSolver` supports and does not support. The main strength of `ModelSolver` is that the equations of the model can be read in whatever form they may be formulated (not normalised in any way), that the equations may be non-linear, and that it handles fairly big models (more than 15,000 equations).

---

[2]Equations may contain lags.

2. Theoretical background

2.1 The model

Consider a model consisting of n equations, [3]

$$L_1(\mathbf{x}_t|\mathbf{z}_t) = R_1(\mathbf{x}_t|\mathbf{z}_t),$$
$$L_2(\mathbf{x}_t|\mathbf{z}_t) = R_2(\mathbf{x}_t|\mathbf{z}_t),$$
$$\vdots$$
$$L_n(\mathbf{x}_t|\mathbf{z}_t) = R_n(\mathbf{x}_t|\mathbf{z}_t),$$

for $t = T_0, T_0+1, \ldots, T_1$, where $\mathbf{x}_t = (x_{1,t}, x_{2,t}, \ldots, x_{n,t})$ is a vector of endogenous variables (and, the way it is written, the values for which solve the model), and $\mathbf{z}_t = (z_{1,t-k_1}, z_{2,t-k_2}, \ldots, z_{m,t-k_m}, \ldots, x_{i,t-\kappa_i}, \ldots)$, for $k_i \geq 0$ and $\kappa_i > 0$, is a vector of contemporaneous and lags of exogenous variables and lags of endogenous variables (or taken together: *predetermined* variables). [4] It's convenient to formulate the model using a vector function $\mathbf{F} : \mathbb{R}^n \mapsto \mathbb{R}^n$, [5]

$$\mathbf{F}(\mathbf{x}_t|\mathbf{z}_t) = \begin{pmatrix} F_1(\mathbf{x}_t|\mathbf{z}_t) \\ F_2(\mathbf{x}_t|\mathbf{z}_t) \\ \vdots \\ F_n(\mathbf{x}_t|\mathbf{z}_t) \end{pmatrix} = \begin{pmatrix} L_1(\mathbf{x}_t|\mathbf{z}_t) - R_1(\mathbf{x}_t|\mathbf{z}_t) \\ L_2(\mathbf{x}_t|\mathbf{z}_t) - R_2(\mathbf{x}_t|\mathbf{z}_t) \\ \vdots \\ L_n(\mathbf{x}_t|\mathbf{z}_t) - R_n(\mathbf{x}_t|\mathbf{z}_t) \end{pmatrix},$$

with the solution to the model being given by $\{\mathbf{x}_t\}_{t=T_0}^{T_1}$ such that

$$\mathbf{F}(\mathbf{x}_t|\mathbf{z}_t) = \mathbf{0} \text{ for } t = T_0, T_0+1, \ldots, T_1.$$

On the face of it, this is a problem which the *Newton-Raphson*-algorithm handles well. The issue, however, is that n might be quite large. In the Norwegian national accounts, for instance, n is greater than 15,500. Therefore, it is useful to analyse the system of equations before solving it, in order to break it down into minimal *simultaneous blocks* that can be solved in a particular order.

## 2.2 Block analysis and ordering

In order to analyse and divide the model into blocks, we use results from *graph theory*.

First, we construct a *bipartite graph* (BiGraph) that connects endogenous variables with equations; i.e., $B = (U, V, E)$, where U is a set containing nodes representing endogenous variables, V is a set containing nodes representing equations, and E is a set containing edges between nodes in U and V. This is illustrated in Figure 2.1 for an arbitrary model with four equations (we omit time subscripts, exogenous variables and

---

[3]L_i and R_i denote the left and right hand side of equation i, respectively. L_i or R_i may be as simple as $x_{j,t}$ or a constant, or something more complicated involving many endogenous variables.

[4]Note that $\mathbf{z}_t$ may contain only $z_{i,t-k_i}$'s or $x_{i,t-k_i}$'s or be empty.

[5]We say that $\mathbf{F}$ maps from $\mathbb{R}^n$ because exogenous variables (contemporaneous and lags) and lags of the endogenous variables are taken as given in any given period, i.e., predetermined.

Figure 2.1   Bipartite graph (BiGraph of model)
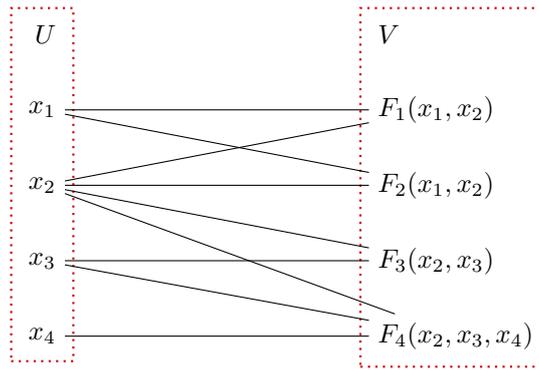
**Figure 2.2   Maximum Bipartite Match (MBM) of BiGraph**
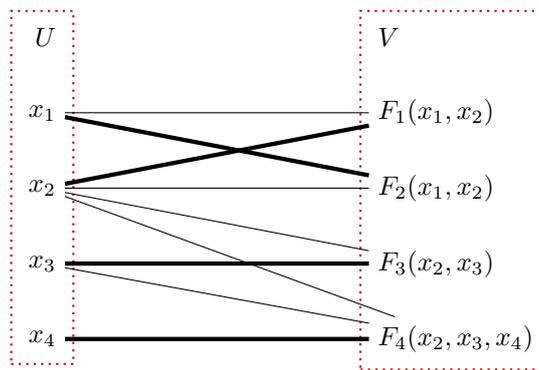
lags for notational convenience as they are not necessary for the analysis).

Next, we apply *maximum bipartite matching* (MBM), which assigns each endogenous variable to *one and only one* equation. Formally, MBM results in a function or mapping M, with $M(x_i) = F_j$ and $M^{-1}(F_j) = x_i$ for $x_i \in U$ and $F_j \in V$ (i.e., M is a *one-to-one* mapping from endogenous variables to equations, $M : U \mapsto V$ and $M^{-1} : V \mapsto U$). The MBM is arbitrary (B may have as many as $n!$ MBMs), and Figure 2.2 shows one possible solution for our arbitrary model.

We use the MBM to determine what endogenous variables impact what *other* endogenous variables. We do this by 1) assigning endogenous variables to equations (or 'labeling' the equations 'as' endogenous variables) using M^{-1}, and 2) removing the MBM edges (the MBM edges simply imply that each endogenous variable impacts itself, which, although not altering the end result, is non-informative).

This procedure leaves us with a new BiGraph $B' = (U, V', E')$, where the nodes in V' are the re-labeled nodes in V (i.e., $V' = \{M^{-1}(F_j)|F_j \in V\}$), and E' contains all the edges *except* the MBM edges (i.e., $E' = \{(x_i, M^{-1}(F_j))|(x_i, F_j) \in E \setminus M\}$). This is illustrated in Figure 2.3 for our arbitrary model. Note that, although the MBM is arbitrary, the description of what endogenous variables affect what *other* endogenous variables E' is *not* (see proof).

Next, we use B' to construct a *directed graph* (DiGraph) that shows how the endogenous variables impact each other (the causality goes from U to V in B'), i.e., $D = (U, E')$, where U (still) contains nodes representing endogenous variables, and E' now is interpreted as directed edges or *arcs* from endogenous variables to *other* endogenous variables. The DiGraph is shown in Figure 2.4 for our arbitrary model.

Figure 2.3   Graph of what endogenous variables impact what *other* endogenous variables
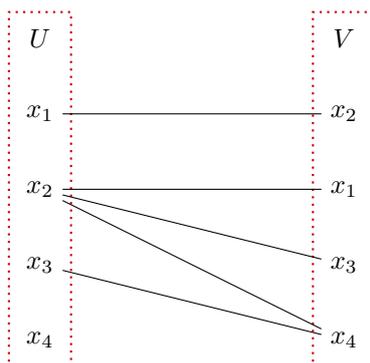
Figure 2.4   Directed graph (DiGraph) of what endogenous variables impact what *other* endogenous variables
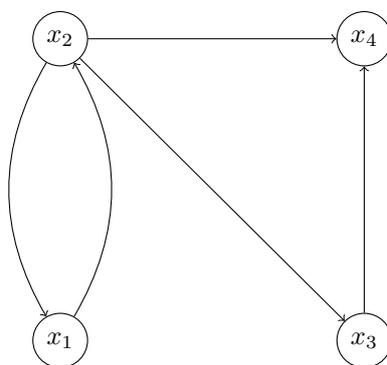
Finally, we find the *strong components* of D, as shown in Figure 2.5 for our arbitrary model. A strong component is a set of nodes that are connected such that every node can be reached from every other node in the set (traversing the arcs). A *condensation* of the DiGraph is a (new) DiGraph with each node being a strong component (of the former). The condensation can be illustrated as in Figure 2.6 for our arbitrary model. Each node of the condensation corresponds to a block of the model, and the arcs decide the order in which the blocks are to be solved. See Gilli (1992) for a discussion of strong components and block analysis.

Note that, since the MBM is a one-to-one-mapping, we may draw the DiGraph and condensation labeling the nodes with endogenous variables *or* equations, switching back and forth using the MBM.

In our arbitrary model, x_1 and x_2 must be solved first in one simultaneous block. Next, x_3 is solved (taking x_1 and x_2 as given, being predetermined by the solution to block 1). Finally, x_4 is solved (taking x_1, x_2 and x_3 as given, being predetermined by the solutions to block 1 and 2).

2.3   Simulation code

With the blocks of the model as given by the condensation of the model DiGraph, as previously discussed, we can generate *simulation code* for each block. That is *either*

   • a symbolic function (definition) that takes predetermined input and returns the endogenous value (if the block is a definition, as described below), *or*

   • a symbolic *objective function* and *Jacobian matrix* that take predetermined input and (initial) values for

Figure 2.5   **Strong components of DiGraph**
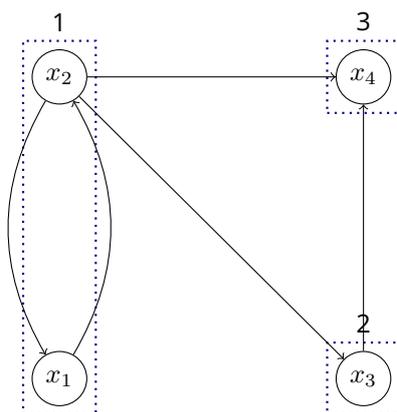
Figure 2.6   **Condensed DiGraph**
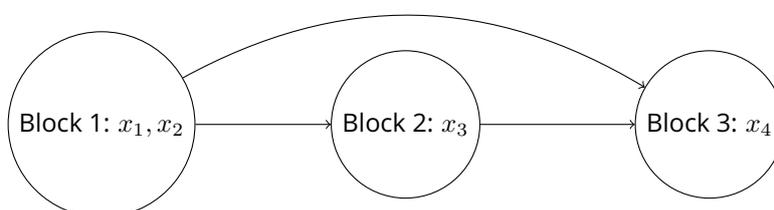
    the endogenous variables, which are to be sent to a Newton-Raphson algorithm, which in turn returns

    the endogenous value(s)

A block is a said to be a definition *if and only if* it 1) consists of one equation and 2) the only thing on the left hand side of the equation is the endogenous variable of that equation (and that endogenous variable does *not* show up on the right hand side of that equation too).

2.4   Solution

In order to solve the model numerically, we loop (nested) over time periods (chronologically) and blocks (according to the order dictated by the condensation of the model DiGraph).

If the block to be solved is a definition, as discussed above, the solution is given straight forwardly by the function defining it (given predetermined input).

If the block to be solved is not a definition, it is sent to a Newton-Raphson algorithm, which in turn returns the solution (given predetermined input, and initial values for the endogenous variables).

The $k + 1$st iteration of the Newton-Raphson algorithm is given by

$$\mathbf{x}_t^{(k+1)} = \mathbf{x}_t^{(k)} - \mathbf{J}_{\mathbf{F}}^{-1}(\mathbf{x}_t^{(k)}|\mathbf{z}_t)\mathbf{F}(\mathbf{x}_t^{(k)}|\mathbf{z}_t),$$

where $\mathbf{J}_{\mathbf{F}}(\mathbf{x}_t|\mathbf{z}_t)$ is the Jacobian matrix of $\mathbf{F}(\mathbf{x}_t|\mathbf{z}_t)$ evaluated at $\mathbf{x}_t$ and $\mathbf{z}_t$. The algorithm stops and returns $\mathbf{x}_t^{(k+1)}$ if $\max\left(\left|\mathbf{x}_t^{(k+1)} - \mathbf{x}_t^{(k)}\right|\right) \leq \varepsilon$, where ε is a tolerance level. If it so happens that $\mathbf{F}(\mathbf{x}_t^{(0)}|\mathbf{z}_t) = \mathbf{0}$, then the initial values solve the model, and as such $\mathbf{x}_t^{(0)}$ is returned as the solution.

3.   Examples of use

The `ModelSolver` object is instantiated by

```
model = ModelSolver(equations, endogenous)
```

where `equations` and `endogenous` are lists containing equations and endogenous variables as strings, e.g.,

```
equations = [
'x1 = a1',
'x2 = a2',
'0.2*x1+0.7*x2 = 0.1*ca+0.8*cb+0.3*i1',
'0.8*x1+0.3*x2 = 0.9*ca+0.2*cb+0.1*i2',
'k1 = k1(-1)+i1',
'k2 = k2(-1)+i2'
]

endogenous = ['x1', 'x2', 'ca', 'cb', 'k1', 'k2']
```

Note that the `(-#)`-syntax denotes lags of a variable; `ModelSolver` does not support leads. Mathematical functions supported are `max()`, `min()`, `log()` and `exp()`. Note also that the equations are written in a form that is neither normalised, nor associates equations with endogenous variables.

It is possible to switch endogenous and exogenous variables using

```
model.switch_endo_vars(old_endo_vars, new_endo_vars)
```

where `old_endo_vars` and `new_endo_vars` are lists. Upon invoking `switch_endo_vars`, a tuple of endogenous variables stored inside the model object is created, and `ModelSolver` immediately performs block analysis on the updated model, such that it can be solved for the updated list of endogenous variables.

3.1   Solving the model

Let `input_df` be a `Pandas DataFrame` containing exogenous variables and initial values for the endogenous variables. Then

```
solution_df = model.solve(input_df)
```

solves the model and stores the results in `solution_df`, which is a `Pandas DataFrame` *identical* to `input_df`, *but* where the endogenous variables are the solutions to the model.

3.2   Analysing the model

`ModelSolver` contains a number of methods that let the user analyse the model:

`model.describe()` describes the model in terms of number of equations and blocks, along with information about the blocks, e.g.,

```
Model consists of 6 equations in 5 blocks
4 of the blocks consist of simple definitions


4 blocks have 1 equations
1 blocks have 2 equations
```

`model.find_endo_var(<endogenous variable>[, noisy=False])` returns the block in which the endogenous variable is solved for.

`model.show_block(<block number>)` returns information about the block—endogenous variables, exogenous and predetermined variables and equations—e.g.,

```
Block consists of an equation or a system of equations


2 endogenous variables:
cb ca


4 exogenous or predetermined variables:
i2 i1 x1 x2


2 equations:
0.8*x1+0.3*x2 = 0.9*ca+0.2*cb+0.1*i2
0.2*x1+0.7*x2 = 0.1*ca+0.8*cb+0.3*i1
```

`model.show_blocks()` returns information about all the blocks of the model.

`model.trace_to_exog_vars(<block number>[, noisy=True])` traces the model DiGraph from the block back to the nodes of origin and reports what exogenous variables determine those nodes.

`model.trace_to_exog_vals(<block number>, <period index>[, noisy=True])` traces the model DiGraph back from the block to the nodes of origin, finds what exogenous variables determine those nodes, and reports values for those variables for the chosen period index (0, 1, ...).

`model.draw_blockwise_graph()` draws a network graph as shown in Figure 3.1. The method takes as arguments a variable name, and the maximum number of ancestor and descendant nodes, and maximum number of nodes in total.

`model.sensitivity(<block number>, <period index>[, method=’std’, exog_subset=None])` analyzes sensitivity of endogenous variables to exogenous variables for a specific period index (0, 1, ...). The method varies the exogneous variable by one standard deviation or one per cent (depending on option choseon by user) and reports the deviations in the block endogenous variables from the original solution.
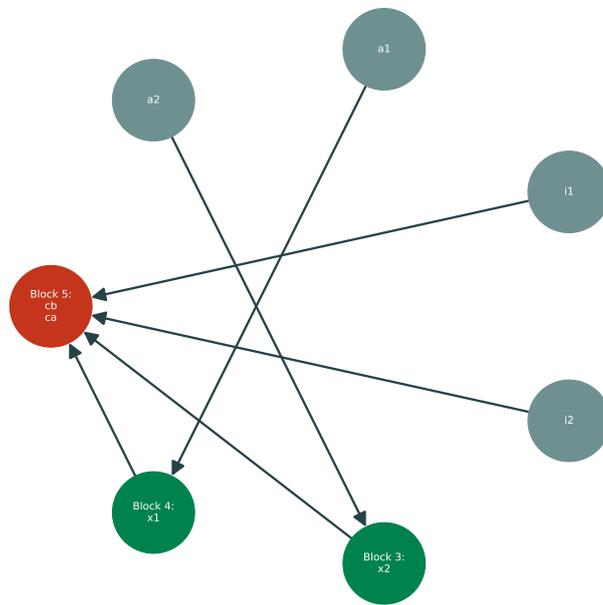
(a)   `model.draw_blockwise_graph(`**`'ca'`**`)`
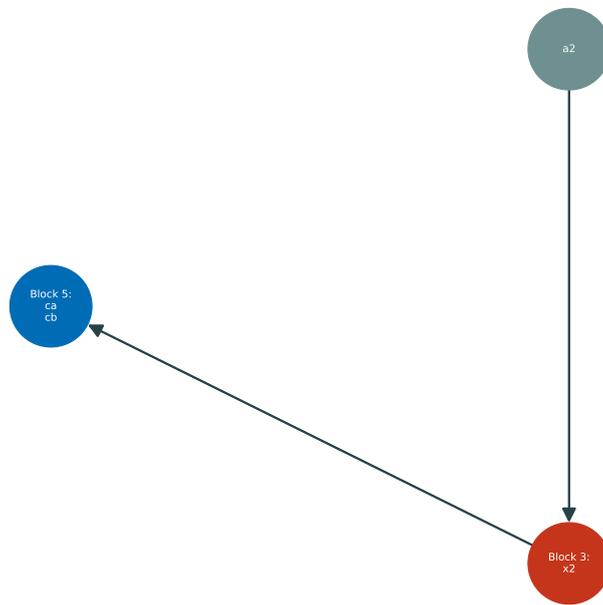
(b)   `model.draw_blockwise_graph(`**`'x2'`**`)`

Figure 3.1   Examples of network plots

4.   Dependencies

`ModelSolver` is built using the following packages:

- NumPy

- NetworkX

- Pandas

- SymEngine

- Numba

- collections

- functools

- Matplotlib

A.   Proofs

Theorem 1. *All MBMs $M_1, M_2, \ldots$ result in the same description of what endogenous variables impact what other endogenous variables $E_1' = E_2' = \ldots = E'$.*

Proof. Suppose we have a BiGraph $B = (U, V, E)$, with

$$U = \{x_1, x_2, \ldots, x_n\},$$
$$V = \{F_1, F_2, \ldots, F_n\},$$
$$E = \{(x_p, F_r), (x_p, F_s), (x_q, F_r), (x_q, F_s)\} \cup \tilde{E},$$

where $\tilde{E}$ is a set containing any *other* edges between endogenous variables (where we omit time subscripts) and equations (or functions) F than the ones explicitly given ($\tilde{E}$ must contain *at least* $n - 2$ edges for the system of equations to have a solution). Suppose further that B has two MBMs (B may have more MBMs—in fact as many as $n!$—but we shall only concern ourselves with these two for the purpose of this proof),[6]

$$M_1 = \{(x_p, F_r), (x_q, F_s)\} \cup \tilde{M},$$
$$M_2 = \{(x_p, F_s), (x_q, F_r)\} \cup \tilde{M},$$

that is, M_1 and M_2 are identical, except for two matches (the ones associating x_p and x_q with F_r and F_s; $\tilde{M}$ is a set containing all *other* matches). We use the MBMs to 'label' equations F 'as' endogenous variables x in E, using $M^{-1}(F_j) = x_i$ for $F_j \in V$ and $x_i \in U$, as discussed in the main text, i.e.,

$$E_1' = \{\cancel{(x_p, x_p)}, (x_p, x_q), (x_q, x_p), \cancel{(x_q, x_q)}\} \cup \tilde{E}',$$
$$E_2' = \{(x_p, x_q), \cancel{(x_p, x_p)}, \cancel{(x_q, x_q)}, (x_q, x_p)\} \cup \tilde{E}'.$$

where prime denotes that the set contains equations 'labeled as' endogenous variables. (We ignore the mappings from an endogenous variable onto itself.) Note that $\tilde{E}'$ is the same for E_1' and E_2'. This follows because we have chosen $\tilde{M}$ to be the same for M_1 and M_2. Therefore, we see that $E_1' = E_2'$. Since we can go from any MBM to any other switching two and two edges repeatedly, it follows that *any* MBM $M_1, M_2, \ldots$ gives the same result as *any other* $E_1' = E_2' = \ldots = E'$, which is what we set out to prove. $\square$

---

[6] As we can choose x_p, x_q, F_r and F_s, this is always the case, unless the MBM is unique (which would render the proof superfluous).

References

Gilli, M. (1992). Causal ordering and beyond. *International Economic Review*, 33(4):957–971.