# Package Guide for SolveDSGE

Richard Dennis[*]
University of Glasgow

December 30, 2014

[*]Email: richard.dennis@glasgow.ac.uk

## 1 Overview

SolveDSGE is a Julia package aimed at macroeconomics interested in solving Dynamic Stochastic General Equilibrium (DSGE) models. The package provides routines for solving rational expectations models and for solving optimal policy problems. Using this package, DSGE models can be solved in logs or levels to first- or second-order accuracy and optimal policy problems can be solved under discretion, commitment, timeless-perspective commitment, and quasi-commitment. Routines that solve robust-control versions of these policy problems are in the works. Although there is much that this package does not do, SolveDSGE offers a broad array of solution methods that can be applied provided the DSGE model can be expressed in one of several standard dynamic representations.

## 2 Installation

To date I have not requested that SolveDSGE be an official Julia package. Therefore, to install SolveDSGE you will need to type the following into the Julia REPL

```
Pkg.clone("git://github.com/RJDennis/SolveDSGE.git")
```

## 3 Solving rational expectations models

### 3.1 First-order-accurate solution methods

SolveDSGE provides a range of solution methods for computing first-order accurate solutions. Exploiting Julia's multiple dispatch all of these solution methods are called via the single command **solve_re**(). From this single command the particular solution method employed depends principally on the model type that enters the **solve_re()** function call. Models are represented in various forms that are summarized by types. The model types are

- **Blanchard_Kahn_Form**
- **Klein_Form**
- **Sims_Form**
- **Binder_Pesaran_Form**

### 3.1.1 Blanchard-Kahn form

The Blanchard-Kahn Form is given by

$$\begin{bmatrix} \mathbf{x}_{t+1} \\ \mathrm{E}_t \mathbf{y}_{t+1} \end{bmatrix} = \mathbf{A} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{y}_t \end{bmatrix} + \mathbf{C} \left[ \boldsymbol{\epsilon}_{t+1} \right],$$

$\boldsymbol{\epsilon}_t \sim i.i.d.[\mathbf{0}, \boldsymbol{\Sigma}]$, where $\mathbf{x}_t$ is an $n_x \times 1$ vector of predetermined variables and $\mathbf{y}_t$ is an $n_y \times 1$ vector of non-predetermined variables. To solve models of this form we first create the **Blanchard_Kahn_Form** type for the model, then we use **solve_re()** to solve the model. The relevant lines of code would be something like

```
cutoff = 1.0
model = Blanchard_Kahn_Form(nx,ny,a,c,sigma)
soln = solve_re(model,cutoff)
```

Here **soln** contains the solution, which is of the form

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{P}\mathbf{x}_t + \mathbf{K}\boldsymbol{\epsilon}_{t+1}, \\ \mathbf{y}_t &= \mathbf{F}\mathbf{x}_t, \end{aligned}$$

and information about the number of eigenvalues with modulus greater than cutoff and whether the “solution” returned is determinate, indeterminate, or explosive.

Models of the **Blanchard_Kahn_Form** type can also be solved using an iterative method to solve a non-symmetric, continuous, algebraic, Riccati equation. In this case the relevant lines of code might look like

```
tol = 1e-10
cutoff = 1.0
model = Blanchard_Kahn_Form(nx,ny,a,c,sigma)
soln = solve_re(model,cutoff,tol)
```

For this iterative method, the variable **cutoff** is still used to establish determinacy, but is not used to order eigenvalues.

### 3.1.2 Klein form

The Klein Form for a model is given by

$$\mathbf{B}\begin{bmatrix} \mathbf{x}_{t+1} \\ \mathrm{E}_t\mathbf{y}_{t+1} \end{bmatrix} = \mathbf{A}\begin{bmatrix} \mathbf{x}_t \\ \mathbf{y}_t \end{bmatrix} + \mathbf{C}\left[\boldsymbol{\epsilon}_{t+1}\right],$$

$\boldsymbol{\epsilon}_t \sim i.i.d.[\mathbf{0},\boldsymbol{\Sigma}]$, where $\mathbf{x}_t$ is an $n_x \times 1$ vector of predetermined variables, $\mathbf{y}_t$ is an $n_y \times 1$ vector of non-predetermined variables, and $\mathbf{B}$ need not have full rank. We can solve models of this form using the code

```
cutoff = 1.0
model = Klein_Form(nx,ny,a,b,c,sigma)
soln = solve_re(model,cutoff)
```

The composite type **soln** contains the solution which is of the form

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{P}\mathbf{x}_t + \mathbf{K}\boldsymbol{\epsilon}_{t+1}, \\ \mathbf{y}_t &= \mathbf{F}x_t, \end{aligned}$$

as well as information about the number of eigenvalues that have modulus greater than cutoff and whether the “solution” returned is determinate, indeterminate, or explosive.

### 3.1.3 Sims form

An alternative model form is used by Sims (2000) and is given by

$$\mathbf{\Gamma}_0\mathbf{z}_t = \mathbf{\Gamma}_1\mathbf{z}_{t-1} + \mathbf{C} + \mathbf{\Psi}\mathbf{v}_t + \mathbf{\Pi}\boldsymbol{\eta}_t,$$

where $\mathbf{v}_t$ is a shock process, possibly serially correlated, with mean-zero innovations whose variance-covarince matrix is given by $\mathbf{\Sigma}$. To solve models that are in this form we would do something like the following

```
cutoff = 1.0
model = Sims_Form(gamma0,gamma1,c,psi,pi,sigma)
soln = solve_re(model,cutoff)
```

Here the solution, summarized by **soln**, is of the form

$$\mathbf{z}_t = \mathbf{G}_1\mathbf{z}_{t-1} + \mathbf{c} + \mathbf{impact} \times \mathbf{v}_t + \mathbf{ywt} \times \left[\mathbf{I} - \mathbf{fmat} \times L^{-1}\right]^{-1} \times \mathbf{fwt} \times \mathbf{v}_{t+1}.$$

### 3.1.4 Binder-Pesaran form

A model is in "structural form" if it is written as

$$\mathbf{A}\mathbf{z}_t = \mathbf{A}_1\mathbf{z}_{t-1} + \mathbf{B}\mathrm{E}_t\mathbf{z}_{t+1} + \mathbf{C}\boldsymbol{\epsilon}_t,$$

where $\boldsymbol{\epsilon}_t \sim i.i.d.[\mathbf{0},\mathbf{\Sigma}]$. We have two ways of solving structural form models. The first recasts them in terms of the Klein form and here the relevant code would look something like

```
cutoff = 1.0
model = Binder_Pesaran_Form(a,a1,b,c,sigma)
soln = solve_re(model,cutoff)
```

The second method is iterative, implementing Binder and Pesaran's "brute force" method; here the code would be something like

**tol = 1e-10**

**cutoff = 1.0**

**model = Binder_Pesaran_Form(a,a1,b,c,sigma)**

**soln = solve_re(model,cutoff,tol)**

Regardless of which of the two methods is used, the solution, summarized in **soln**, has the form

$$\mathbf{z}_t = \mathbf{P}\mathbf{z}_{t-1} + \mathbf{K}\boldsymbol{\epsilon}_t.$$

As earlier, **soln** is a composite type that in addition to the solution itself also contains information about the number of eigenvalues with modulus greater than cutoff and whether the solution returned is determinate, indeterminate, or explosive.

## 3.2 Second-order-accurate solution methods

In addition to the first-order accurate solution methods documented above, SolveDSGE also contains two methods form obtaining second-order-accurate solutions to nonlinear DSGE models. As coded here, these two methods employ the same model form but differ in how the solution is computed. To employ either method the DSGE model is first expressed in the form

$$\mathrm{E}_t\mathbf{G}\left(\mathbf{x}_t, \mathbf{y}_t, \mathbf{x}_{t+1}, \mathbf{y}_{t+1}\right) = \mathbf{0}.$$

With $\mathbf{x}_t$ containing $n_x$ predetermined variables and $\mathbf{y}_t$ containing $n_y$ non-predetermined variables, $\mathbf{G}()$ is a vector containing $n_x+n_y$ functions. Bundling $x_t$ and $y_t$ into a new vector $\mathbf{z}_t = \left[\begin{array}{cc} \mathbf{x}_t^{'} & \mathbf{y}_t^{'} \end{array}\right]^{'}$ and bundling $\mathbf{z}_t$ and $\mathbf{z}_{t+1}$ into a new vector $\mathbf{p}_t = \left[\begin{array}{cc} \mathbf{z}_t^{'} & \mathbf{z}_{t+1}^{'} \end{array}\right]^{'}$ we get

$$\mathrm{E}_t\mathbf{G}\left(\mathbf{p}_t\right) = \mathbf{0}.$$

We now approximate $\mathbf{G}\left(\mathbf{p}_t\right)$ around the deterministic steady state, $\overline{\mathbf{p}}$, using a second-order Taylor expansion giving

$$\mathbf{G}\left(\mathbf{p}_t\right) \simeq \mathbf{G}_p\left(\mathbf{p}_t - \overline{\mathbf{p}}\right) + \left[\mathbf{I} \otimes \left(\mathbf{p}_t - \overline{\mathbf{p}}\right)\right]^{'} \mathbf{G}_{pp}\left[\mathbf{I} \otimes \left(\mathbf{p}_t - \overline{\mathbf{p}}\right)\right] = \mathbf{0},$$

where $\mathbf{G}_p$ is a matrix of first-derivatives and $\mathbf{G}_{pp}$ is a matrix of stacked Hessians, one Hessian for each of the $n_x + n_y$ equations.

We now recognize that some elements of $\mathbf{x}_t$ (usually the first $s$ elements) are shocks that have the form

$$\mathbf{s}_{t+1} = \mathbf{\Lambda s}_t + \boldsymbol{\eta\epsilon}_{t+1},$$

where $\boldsymbol{\epsilon}_t \sim i.i.d.[\mathbf{0}, \mathbf{\Sigma}]$. The essential components required for a second-order-accurate solution are now given by $n_x$, $n_y$, $\mathbf{G}_p$, $\mathbf{G}_{pp}$, $\boldsymbol{\eta}$, and $\mathbf{\Sigma}$.

The two model types that we consider for second-order-accurate solution methods are

- **Gomme_Klein_Form**
- **Lombardo_Sutherland_Form**

### 3.2.1 Gomme-Klein form

To compute a second-order accurate solution using the Gomme and Klein (2011) method we summarize the model in the form of the **Gomme_Klein_Form** composite type. Once this model type is constructed the model can be solved. The code to compute the solution would be something like

```
cutoff = 1.0
model = Gomme_Klein_Form(nx,ny,Gp,Gpp,eta,sigma)
soln = solve_re(model,cutoff)
```

Here **soln** is a composite type that contains the solution, which is of the form

$$\begin{aligned}
\mathbf{x}_{t+1} - \overline{\mathbf{x}} &= \frac{1}{2}\mathbf{ssh} + \mathbf{hx}\left(\mathbf{x}_t - \overline{\mathbf{x}}\right) + \frac{1}{2}\left[\mathbf{I} \otimes \left(\mathbf{x}_t - \overline{\mathbf{x}}\right)\right]' \mathbf{hxx}\left[\mathbf{I} \otimes \left(\mathbf{x}_t - \overline{\mathbf{x}}\right)\right] + \boldsymbol{\eta\epsilon}_{t+1}, \\
\mathbf{y}_t - \overline{\mathbf{y}} &= \frac{1}{2}\mathbf{ssg} + \mathbf{gx}\left(\mathbf{x}_t - \overline{\mathbf{x}}\right) + \frac{1}{2}\left[\mathbf{I} \otimes \left(\mathbf{x}_t - \overline{\mathbf{x}}\right)\right]' \mathbf{gxx}\left[\mathbf{I} \otimes \left(\mathbf{x}_t - \overline{\mathbf{x}}\right)\right],
\end{aligned}$$

where **hxx** and **gxx** are stacked matrices containing the second order coefficients for each of the $n_x$ and $n_y$ equations, respectively. **soln** also contains information about the number of eigenvalues with modulus greater than cutoff and the solution's determinacy properties, where these properties are associated with the model first-order dynamics.

### 3.2.2 Lombardo-Sutherland form

Implementing the Lombardo and Sutherland (2007) solution method is no different than for Gomme and Klein (2010). The key difference is the form in which the solution is presented. Ths code to implement the Lombardo-Sutherland method would look something like

```
cutoff = 1.0
model = Lombardo_Sutherland_Form(nx,ny,Gp,Gpp,eta,sigma)
soln = solve_re(model,cutoff)
```

where now the solution has the form

$$\begin{aligned}
\mathbf{x}_{t+1}-\overline{\mathbf{x}} &= \frac{1}{2}\mathbf{ssh}+\mathbf{hx}\left(\mathbf{x}_t-\overline{\mathbf{x}}\right)+\frac{1}{2}\mathbf{hxx}\times\mathbf{v}_t+\boldsymbol{\eta}\boldsymbol{\epsilon}_{t+1}, \\
\mathbf{y}_t-\overline{\mathbf{y}} &= \frac{1}{2}\mathbf{ssg}+\mathbf{gx}\left(\mathbf{x}_t-\overline{\mathbf{x}}\right)+\frac{1}{2}\mathbf{gxx}\times\mathbf{v}_t, \\
\mathbf{v}_t &= \boldsymbol{\Phi}\mathbf{v}_{t-1}+\boldsymbol{\Gamma}vech\left(\boldsymbol{\epsilon}_t\boldsymbol{\epsilon}_t^{'}\right)+\boldsymbol{\Psi}vec\left(\mathbf{x}_t\boldsymbol{\epsilon}_t^{'}\right),
\end{aligned}$$

with $\mathbf{v}_t$ given by

$$\mathbf{v}_t=vech\left(\mathbf{x}_t\mathbf{x}_t^{\prime}\right).$$

The solution form produced by the Lombardo-Sutherland method can be converted into that produced by the Gomme-Klein method by using the **convert_second_order** function as follows

```
new_soln = convert_second_order(soln)
```

Where **soln** is of type **Lombardo_Sutherland_Soln**, **new_soln** is of type **Gomme_Klein_Soln**.

## 4 Solving optimal policy problems

SolveDSGE provides routines for solving Linear-Quadratic (LQ) optimal policy problems. These LQ problems allow policy to be conducted under: discretion; commitment; quasi-commitment; and timeless-perspective commitment. The solutions to these four policy problems are obtained using the commands **solve_disc()**, **solve_commit()**, **solve_quasi()**, and **solve_timeless()**, respectively. The optimal policy routines are based around four model types and two solution types. At this stage not all of these policies are supported for all model-types. The four model-types and the optimal policies that they support are documented below.

### 4.1 State space form

The LQ optimal policy problem in **State_Space_Form** is described by the quadratic objective function

$$Loss = \mathrm{E}\left[\sum_{t=0}^{\infty}\beta^{t}\left(\mathbf{z}_{t}^{'}\mathbf{Q}\mathbf{z}_{t}+\mathbf{z}_{t}^{'}\mathbf{U}\mathbf{u}_{t}+\mathbf{u}_{t}^{'}\mathbf{U}^{'}\mathbf{z}_{t}+\mathbf{u}_{t}^{'}\mathbf{R}\mathbf{u}_{t}\right)\right],$$

and the linear constraints

$$\begin{bmatrix}\mathbf{x}_{t+1}\\ \mathrm{E}_{t}\mathbf{y}_{t+1}\end{bmatrix}=\mathbf{A}\begin{bmatrix}\mathbf{x}_{t}\\ \mathbf{y}_{t}\end{bmatrix}+\mathbf{B}\mathbf{u}_{t}+\mathbf{C}\left[\boldsymbol{\epsilon}_{t+1}\right],$$

where $\mathbf{x}_t$ contains $n_x$ predetermined variables, $\mathbf{y}_t$ contains $n_y$ non-predetermined variables, $\mathbf{u}_t$ contains $n_p$ policy instruments, and $\boldsymbol{\epsilon}_t$ contains $n_s$ stochastic innovations.

For this model form the following policies are supported:

- Discretion ($\mathbf{s}_t = \mathbf{x}_t$)
- Commitment $\left(\mathbf{s}_t = \begin{bmatrix}\mathbf{x}_t\\ \boldsymbol{\lambda}_t\end{bmatrix}\right)$
- Quasi-commitment $\left(\mathbf{s}_t = \begin{bmatrix}\mathbf{x}_t\\ \boldsymbol{\lambda}_t\end{bmatrix}\right)$

- Timeless-perspective commitment $\left(\mathbf{s}_t = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{x}_{t-1} \\ \mathbf{u}_{t-1} \end{bmatrix}\right)$

For each policy the solution returned is of the form

$$\begin{aligned}
\mathbf{s}_{t+1} &= \mathbf{P}\mathbf{s}_t + \mathbf{K}\boldsymbol{\epsilon}_{t+1}, \\
\mathbf{z}_t &= \mathbf{H}\mathbf{s}_t, \\
\mathbf{u}_t &= \mathbf{F}\mathbf{s}_t.
\end{aligned}$$

with this information summarized in the solution type, **State_Space_Soln**.

To solve a model for each of the policies above we would use code like the following

```
obj = State_Space_Objective(beta,q,u,r)
model = State_Space_Form(nx,ny,a,b,c,sigma)
tol = 1e-10
maxiters = 100
commit_prob = 0.75
soln_disc = solve_disc(model,obj,tol,maxiters)
soln_commit = solve_commit(model,obj,tol,maxiters)
soln_quasi = solve_quasi(model,obj,commit_prob,tol,maxiters)
soln_timeless = solve_timeless(model,obj,tol,maxiters)
```

## 4.2 Generalized state space form

The LQ optimal policy problem in **Generalized_State_Space_Form** is described by the quadratic objective function

$$Loss = \mathrm{E}\left[\sum_{t=0}^{\infty} \beta^t \left(\mathbf{z}_t^{'}\mathbf{Q}\mathbf{z}_t + \mathbf{z}_t^{'}\mathbf{U}\mathbf{u}_t + \mathbf{u}_t^{'}\mathbf{U}^{'}\mathbf{z}_t + \mathbf{u}_t^{'}\mathbf{R}\mathbf{u}_t\right)\right],$$

and the linear constraints

$$\left[\begin{array}{c}\mathbf{x}_{t+1}\\ \mathbf{A}_0\mathrm{E}_t\mathbf{y}_{t+1}\end{array}\right]=\mathbf{A}\left[\begin{array}{c}\mathbf{x}_t\\ \mathbf{y}_t\end{array}\right]+\mathbf{B}\mathbf{u}_t+\mathbf{C}\left[\boldsymbol{\epsilon}_{t+1}\right],$$

where $\mathbf{x}_t$ contains $n_x$ predetermined variables, $\mathbf{y}_t$ contains $n_y$ non-predetermined variables, $\mathbf{u}_t$ contains $n_p$ policy instruments, and $\boldsymbol{\epsilon}_t$ contains $n_s$ stochastic innovations. The **Generalized_State_Space_Form** differs from the **State_Space_Form** above through the presence of the (usually) singular leading matrix $\mathbf{A}_0$.

For this model form the following policies are supported:

- Discretion ($\mathbf{s}_t = \mathbf{x}_t$)
- Commitment $\left(\mathbf{s}_t = \left[\begin{array}{c}\mathbf{x}_t\\ \boldsymbol{\lambda}_t\end{array}\right]\right)$
- Quasi-commitment $\left(\mathbf{s}_t = \left[\begin{array}{c}\mathbf{x}_t\\ \boldsymbol{\lambda}_t\end{array}\right]\right)$
- Timeless-perspective commitment $\left(\mathbf{s}_t = \left[\begin{array}{c}\mathbf{x}_t\\ \mathbf{x}_{t-1}\\ \mathbf{u}_{t-1}\end{array}\right]\right)$

For each policy the solution returned is of the form

$$\begin{aligned}\mathbf{s}_{t+1} &= \mathbf{P}\mathbf{s}_t+\mathbf{K}\boldsymbol{\epsilon}_{t+1},\\ \mathbf{z}_t &= \mathbf{H}\mathbf{s}_t,\\ \mathbf{u}_t &= \mathbf{F}\mathbf{s}_t.\end{aligned}$$

with this information summarized in the solution type, **State_Space_Soln**.

To solve a model for each of the policies above we would use code like the following

**obj = State_Space_Objective(beta,q,u,r)**
**model = Generalized_State_Space_Form(nx,ny,a0,a,b,c,sigma)**

**tol = 1e-10**

**maxiters = 100**

**commit_prob = 0.75**

**soln_disc = solve_disc(model,obj,tol,maxiters)**

**soln_commit = solve_commit(model,obj,tol,maxiters)**

**soln_quasi = solve_quasi(model,obj,commit_prob,tol,maxiters)**

**soln_timeless = solve_timeless(model,obj,tol,maxiters)**

### 4.3 Structural form

The LQ optimal policy problem in **Structural_Form** is described by the quadratic objective function

$$Loss = \mathrm{E}\left[\sum_{t=0}^{\infty}\beta^{t}\left(\mathbf{y}_{t}^{'}\mathbf{Q}\mathbf{y}_{t}+\mathbf{u}_{t}^{'}\mathbf{R}\mathbf{u}_{t}\right)\right],$$

and the linear constraints

$$\mathbf{A}_0\mathbf{y}_t = \mathbf{A}_1\mathbf{y}_{t-1} + \mathbf{A}_2\mathrm{E}_t\mathbf{y}_{t+1} + \mathbf{A}_3\mathbf{u}_t + \mathbf{A}_5\boldsymbol{\epsilon}_t,$$

where $\mathbf{y}_t$ contains $n$ variables, $\mathbf{u}_t$ contains $n_p$ policy instruments, and $\boldsymbol{\epsilon}_t$ contains $n_s$ stochastic innovarions.

For this model form the following policies are supported:

- Discretion ($\mathbf{s}_{t-1} = \mathbf{y}_{t-1}$)
- Commitment $\left(\mathbf{s}_{t-1} = \begin{bmatrix} \mathbf{y}_{t-1} \\ \boldsymbol{\lambda}_{t-1} \end{bmatrix}\right)$

For each policy the returned solution is of the form

$$\begin{aligned} \mathbf{s}_t &= \mathbf{P}\mathbf{s}_{t-1} + \mathbf{K}\boldsymbol{\epsilon}_t, \\ \mathbf{u}_t &= \mathbf{F}\mathbf{s}_{t-1}. \end{aligned}$$

with this information contained in the solution type, **Structural_Soln**.

To solve a model for each of the policies above we would use code like the following

```
obj = Structural_Objective(beta,q,r)
model = Structural_Form(a0,a1,a2,a3,a5,sigma)
tol = 1e-10
maxiters = 100
soln_disc = solve_disc(model,obj,tol,maxiters)
soln_commit = solve_commit(model,obj,tol,maxiters)
```

## 4.4 Generalized structural form

The LQ optimal policy problem in **Generalized_Structural_Form** is described by the quadratic objective function

$$Loss = \mathrm{E}\left[\sum_{t=0}^{\infty} \beta^t \left(\mathbf{y}_t^{'}\mathbf{Q}\mathbf{y}_t + \mathbf{u}_t^{'}\mathbf{R}\mathbf{u}_t\right)\right],$$

and the linear constraints

$$\mathbf{A}_0\mathbf{y}_t = \mathbf{A}_1\mathbf{y}_{t-1} + \mathbf{A}_2\mathrm{E}_t\mathbf{y}_{t+1} + \mathbf{A}_3\mathbf{u}_t + \mathbf{A}_4\mathrm{E}_t\mathbf{u}_{t+1} + \mathbf{A}_5\boldsymbol{\epsilon}_t,$$

where $\mathbf{y}_t$ contains $n$ variables, $\mathbf{u}_t$ contains $n_p$ policy instruments, and $\boldsymbol{\epsilon}_t$ contains $n_s$ stochastic innovarions.

For this model form the following policies are supported:

- Discretion ($\mathbf{s}_{t-1} = \mathbf{y}_{t-1}$)
- Commitment $\left(\mathbf{s}_{t-1} = \begin{bmatrix} \mathbf{y}_{t-1} \\ \boldsymbol{\lambda}_{t-1} \end{bmatrix}\right)$

For each policy the returned solution is of the form

$$\begin{aligned} \mathbf{s}_t &= \mathbf{P}\mathbf{s}_{t-1} + \mathbf{K}\boldsymbol{\epsilon}_t, \\ \mathbf{u}_t &= \mathbf{F}\mathbf{s}_{t-1}. \end{aligned}$$

with this information contained in the solution type, **Structural_Soln**.

To solve a model for each of the policies above we would use code like the following

```
obj = Structural_Objective(beta,q,r)
model = Generalized_Structural_Form(a0,a1,a2,a3,a4,a5,sigma)
tol = 1e-10
maxiters = 100
soln_disc = solve_disc(model,obj,tol,maxiters)
soln_commit = solve_commit(model,obj,tol,maxiters)
```

## References

[1] Binder, M., and H. Pesaran, (1995), "Multivariate Rational Expectations Models and Macroeconomic Modeling: A Review and Some New Results," in: Pesaran, H., M. Wickens, (Eds.), *Handbook of Applied Econometrics*, Basil Blackwell, Oxford, pp.139–187.

[2] Blanchard, O., and C. Kahn, (1980), "The Solution to Linear Difference Models under Rational Expectations," *Econometrica*, 48, pp.1305–1311.

[3] Gomme, P., and P. Klein, (2011), "Second-Order Approximation of Dynamic Models Without the Use of Tensors," *Journal of Economic Dynamics and Control*, 35, pp.604–615.

[4] Klein, P., (2000), "Using the Generalized Schur Form to Solve a Multivariate Linear Rational Expectations Model," *Journal of Economic Dynamics and Control*, 24, pp.1405–1423.

[5] Lombardo, G., and A. Sutherland, (2007), "Computing Second-Order-Accurate Solutions for Rational Expectations models Using Linear Solution Methods," *Journal of Economic Dynamics and Control*, 31, pp.515–530.

[6] Sims, C., (2001), "Solving Linear Rational Expectations Models," *Computational Economics*, 20, pp.1–20.