SolveDSGE v0.3.0 — A User Guide

Richard Dennis*
University of Glasgow and CAMA

February 2020

Abstract

SolveDSGE is a Julia package for solving nonlinear Dynamic Stochastic General Equilibrium models. A varietry of solution methods are available, and they are interchangable so that one solution can be used subsequently as an initialization to obtain a more accurate solution. The package can compute one- second- and third-order perturbation solutions and Chebyshev-based, Smolyak-based and piecewise linear-based projection solutions. Once a model has been solved, the package can used used to simulate data and/or compute impulse response functions.

JEL Classification: E3, E4, E5.

*Address for Correspondence: Adam Smith Business School, University of Glasgow, Main Building, University Avenue, Glasgow G12 8QQ; email: richard.dennis@glasgow.ac.uk.

1 Introduction

SolveDSGE is a framework for solving and analyzing Dynamic Stochastic General Equilibrium (DSGE) models that is implemented in the programming language Julia. SolveDSGE will solve nonlinear DSGE models using perturbation methods, producing solutions that are accurate to first, second, and third order, but this is not its focus. The package's focus is on applying projection methods to obtain solutions that are globally accurate.

Obtaining globally accurate solutions to nonlinear DSGE models is notoriously difficult. Solutions are invariably slow to obtain and model-specific characteristics are often exploited to speed up the solution process. SolveDSGE does not exploit model-specific characteristics in order to solve a model. Instead, SolveDSGE applies the same general solution strategy to all models. Nonetheless, making use of Julia's speed, SolveDSGE allows models to be solved "relatively quickly", and it provides users with an easy, unified, way of organizing and expressing their model. At the users request, globally accurate solutions can be obtained using Chebyshev polynomials, Smolyak polynomials, or piecewise linear approximations, with the solution obtained from one approximation scheme able to be used as an initialization for the others, allowing greater speed and accuracy to be obtained via a form of homotophy.

To use SolveDSGE to solve a model, two files must be supplied. The first file (the model file) summarizes the model to be solved. The second file (the solution file) reads the model file, solves the model, and performs any post-solution analysis.

Quite a lot of time and effort has gone into writing SolveDSGE, together with the underlying modules: ChebyshevApprox, SmolyakApprox, and PiecewiseLinearApprox, but it is far from perfect. SolveDSGE may not be able to solve your model, or it may not obtain a solution quickly enough to be useful to you. You are welcome to suggest improvements to fix bugs or add functionality. At the same time, I am hopeful that you will find the package useful for your research. If it is, then please cite this User Guide and add an acknowledgement of SolveDSGE to your paper/report.

2 The model file

SolveDSGE requires that the model that is to be solved be stored in a model file. The model file is simply a text file so there is nothing particularly special about it. Every model file must contain the following five information categories: "states:", "jumps:", "shocks:", "parameters:", and "equations:"; each category name must end with a colon. Each category will begin with its name, such as "states:" and conclude with an "end". The model file can present these five categories in any order.

The information in each category can be presented with one element per line, or with multiple elements on each line with each element separated by either a comma or a semi-colon. So if the jump variables in the model are labor, consumption, and output, then this could be presented in a variety of ways, such as:

jumps:
labor
consumption
output
end

or:
jumps:
labor, consumption, output
end

or:
jumps:
labor; consumption, output
end

The first lag of a variable is denoted with a -1, so the lag of consumption is consumption(-1). Similarly the first lead of a variable is denoted with a +1, so the lead of consumption is denoted consumption(+1). The first lag of any model variable is automatically included as a state variable, second and higher lags should be given a name, defined by an equation, and included as state variables explicitly. The package may allow higher lags to processed automatically at a later stage.

Shocks in the model refers to the innovations to the shock processes, so if the shock process is given by

$$tech(+1) = rho * tech + sd * epsilon,$$

then "*tech*" will be a state variable, "*epsilon*" will be a shock, and "*rho*" and "*sd*" will be parameters. If the model is deterministic, then it will contain no shocks.

Every element in the parameters category and the equations category must contain an "=" sign, such as "$alpha = 0.33$" in the case of the parameters category and "$output = exp(tech) * capital\hat{\ }alpha * labor\hat{\ }(1.0 - alpha)$" in the case of the equations category.

2.1 Example

The following is an example of a model file for the stochastic growth model:

```
states:
cap, tech
end

jumps:
cons
end

shocks:
epsilon
end

parameters:
betta = 0.99
sigma = 1.1
delta = 0.025
alpha = 0.30
rho = 0.8
sd = 0.01
end

equations:
cap(+1) = (1.0 - delta)*cap + exp(tech)*cap^alpha - cons
cons^(-sigma) = betta*cons(+1)^(-sigma)*(1.0 - delta + alpha*exp(tech(+1))*cap(+1)^(alpha
- 1.0))
tech(+1) = rho*tech + sd*epsilon
end
```

3 Solving a model

Solving a model is straightforward; it consists of the following steps:

1. Read and process the model file. During the processing the order of variables in the system may be changed, typically the changes are to place the shocks at the top of the system. After processing is complete you will be told what the variable-order is.

2. Solve for the model's steady state.

3. Specify a SolutionScheme. A SolutionScheme specifies the solution method along with any parameters needed to implement that solution method.

4. Solve the model according to the chosen SolutionScheme.

3.1 Reading the model and solving for its steady state

To read and process a model file we simply supply the path/filename to the get_model() function, for example

dsge = get_model("c:/desktop/model.txt")

We can then solve for the model's steady state as follows

ss = compute_steady_state(dsge,tol,maxiters)

where *dsge* is the model whose steady state is to be computed, *tol* is a convergence tolerance, and *maxiters* is an integer specifying the maximum number of iterations before the function exits.

3.2 Specifying a SolutionScheme

To solve a model a SolutionScheme must be supplied. A SolutionScheme specifies the solution method and the parameters upon which this solution method relies. The solution methods in SolveDSGE are either perturbation methods or projection methods. Accordingly, the Solution-Schemes can be divided into PerturbationSchemes or ProjectionSchemes. We present each in turn.

3.2.1 PerturbationSchemes

To solve a model using a perturbation method requires and PerturbationScheme. Regardless of the model or the order of the perturbation, a PerturbationScheme is a structure with three fields: the point about which to perturb the model (the steady state), a cutoff parameter that separates unstable from stable eigenvalues (eigenvalues whose modulus is greater than cutoff will be placed in the model's unstable block), and the order of the perturbation. For a first-order perturbation, a typical PerturbationScheme might be the following

N = PerturbationScheme(ss,cutoff,"first")

while those for second and third order perturbations might be

NN = PerturbationScheme(ss,cutoff,"second")

and

NNN = PerturbationScheme(ss,cutoff,"third")

The method used to compute a first-order perturbation follows Klein (2000), that for a second-order perturbation follows Gomme and Klein (2011), while that for a third-order perturbation follows Binning (2013) with a refinement from Levintal (2017). At this point, perturbation solutions higher than third order are not supported.

3.2.2 ProjectionSchemes

ProjectionSchemes are either ChebyshevSchemes, SmolyakSchemes, or PiecewiseLinearSchemes, and for each of these there is a stochastic (for stochastic models) and a deterministic (for deterministic models) version. The SolutionScheme for the deterministic case is a special case of the stochastic one, so we focus on the stochastic case in what follows.

ChebyshevSchemes Solutions based on Chebyshev polynomials rely on and make use of all of the functionality of the module ChebyshevApprox. This means that an arbitrary number of state variables can be accommodated (if you have enough time!) and both tenser-product and complete polynomials can be used. A stochastic ChebyshevScheme requires the following arguments:

- initial_guess — This will usually be a vector containing the model's steady state. It is used as the initial guess at the solution for the case where an initializing solution is not provided (see the section on model solution below).

- node_generator — This is the name of the function used to generate the nodes for the Chebyshev polynomial. Possible options include: chebyshev_nodes and chebyshev_extrema.
- node_number — This gives the number of nodes to be used for each state variable. If there is only one state variables then node_number will be an integer. When there are two of more state variables it will be a vector of integers.
- num_quad_nodes — This is an integer specifying the number of quadrature points used to compute expectations.
- order — This defines the order of the Chebyshev polynomial to be used in the approximating functions. For a complete polynomial order will be an integer; for a tenser-product polynomial order will be a vector of integers.
- domain — This contains the domain for the state variables over which the solution is obtained. Domain will be a $2-$element vector in the one-state-variable case and a $2 \times n$ array in the n-state-variable case, with the first row of the array containing the upper values of the domain and the second row containing the lower values of the domain. If an initializing solution is provided, then the domain associated with that initializing solution can be used by setting domain to an empty array, Float64[].
- tol_fix_point_solver — This specifies the tolerance to be used in the inner loop to determine convergence at each solution node.
- tol_variables — This specifies the tolerance to be used in the outer loop to determine convergence of the overall solution.
- maxiters — This is an integer specifying the maximum number of outer-loop iterations before the solution exits.

An example of a stochastic ChebyshevScheme is

C = ChebyshevSchemeStoch(ss,chebyshev_nodes,[21,21], 9, 4,[0.1 30.0; -0.1 20.0],1e-8,1e-6,1000)

In the deterministic case the number of quadrature nodes is not needed, i.e.,

Cdet = ChebyshevSchemeDet(ss,chebyshev_nodes,[21,21],4,[0.1 30.0; -0.1 20.0],1e-8,1e-6,1000)

SmolyakSchemes Underlying the Smolyak polynomial based solution is the module SmolyakApprox. This module allows for both isotropic polynomials and ansiotropic polynomials and several different methods for producing nodes. SolveDSGE exploits all of this functionality. A stochastic SmolyakScheme requires the following arguments:

- initial_guess — This will usually be a vector containing the model's steady state. It is used as the initial guess at the solution for the case where an initializing solution is not provided (see the section on model solution below).
- node_generator —This is the name of the function used to generate the nodes for the Smolyak polynomial. Possible options include: chebyshev_gauss_lobatto and clenshaw_curtis_equidistant
- num_quad_nodes — This is an integer specifying the number of quadrature points used to compute expectations.
- layer — This is an integer (isotropic case) or a vector of integers (ansiotropic case) specifying the number of layers to be used in the approximation.
- domain — This contains the domain for the state variables over which the solution is obtained. Domain will be a $2-$element vector in the one-state-variable case and a $2 \times n$ array in the n-state-variable case, with the first row of the array containing the upper values of the domain and the second row containing the lower values of the domain. If an initializing solution is provided, then the domain associated with that initializing solution can be used by setting domain to an empty array, Float64[].
- tol_fix_point_solver — This specifies the tolerance to be used in the inner loop to determine convergence at each solution node.
- tol_variables — This specifies the tolerance to be used in the outer loop to determine convergence of the overall solution.
- maxiters — This is an integer specifying the maximum number of outer-loop iterations before the solution exits.

An example of a stochastic SmolyakScheme is

S = SmolyakSchemeStoch(ss,chebyshev_gauss_lobatto,9,3,[0.1 30.0; -0.1 20.0],1e-8,1e-6,1000)

In the deterministic case the number of quadrature nodes is not needed, i.e., *Sdet = SmolyakSchemeDet(ss,chebyshev_gauss_lobatto,3,[0.1 30.0; -0.1 20.0],1e-8,1e-6,1000)*

PiecewiseLinearSchemes To obtain piecewise linear solutions, SolveDSGE employs the module PiecewiseLinearApprox, which allows approximations over an arbitrary number of state variables. A stochastic PiecewiseLinearScheme requires the following arguments:

- initial_guess — This will usually be a vector containing the model's steady state. It is used as the initial guess at the solution for the case where an initializing solution is not provided (see the section on model solution below).
- node_number — This gives the number of nodes to be used for each state variable. If there is only one state variables then node_number will be an integer. When there are two of more state variables it will be a vector of integers.
- num_quad_nodes — This is an integer specifying the number of quadrature points used to compute expectations.
- domain — This contains the domain for the state variables over which the solution is obtained. Domain will be a 2−element vector in the one-state-variable case and a $2 \times n$ array in the n-state-variable case, with the first row of the array containing the upper values of the domain and the second row containing the lower values of the domain. If an initializing solution is provided, then the domain associated with that initializing solution can be used by setting domain to an empty array, Float64[].
- tol_fix_point_solver — This specifies the tolerance to be used in the inner loop to determine convergence at each solution node.
- tol_variables — This specifies the tolerance to be used in the outer loop to determine convergence of the overall solution.
- maxiters — This is an integer specifying the maximum number of outer-loop iterations before the solution exits.

An example of a stochastic PiecewiseLinearScheme is

P = PiecewiseLinearStoch(ss,[21,21],9,[0.1 30.0; -0.1 20.0],1e-8,1e-6,1000)

In the deterministic case the number of quadrature nodes is not needed, i,e.,

Pdet = PiecewiseLinearDet(ss,[21,21],9,[0.1 30.0; -0.1 20.0],1e-8,1e-6,1000)

3.3 Model solution

Once a SolutionScheme is specified we are in a position to solve the model. In order to do so we use the solve_model() function, which takes either two or three arguments. For a perturbation solution solve_model() requires two arguments: the model to be solved and the SolutionScheme, as follows:

soln_first_order = solve_model(dsge,N)

soln_second_order = solve_model(dsge,NN)

soln_third_order = solve_model(dsge,NNN)

Alternatively, for a projection solution solve_model() takes either two or three arguments. To provide a concrete example, suppose we wish to solve our model using Chebyshev polynomials. If we want the projection solution to be initialized using the steady state, then solve_model() requires only two arguments: the model to be solved and the SolutionScheme:

soln_chebyshev = solve_model(dsge,C)

If we want the projection solution to be initialized using the third order perturbation solution, then solve_model() requires three arguments: the model to be solved, the initializing solution, and the SolutionScheme:

soln_chebyshev = solve_model(dsge,soln_third_order,C)

Although this example uses a third order perturbation as the initializing solution, any solution (first order, second order, third order, Chebyshev, Smolyak, or piecewise linear) can be used.

3.3.1 A comment on third-order perturbation

Sometimes it can be useful to add skewness to the shocks, but this is not easy to do through the model file. If you want your shocks to be skewed, then you can access the third order perturbation solution by calling

soln_third_order = solve_third_order(dsge,NNN,skewness)

where skewness is a 2D array containing the skewness coefficients. If there is only one shock, then the skewness array is

$$skewness = E\left[\epsilon_1\epsilon_1\epsilon_1\right].$$

If there are two shocks, then the skewness array is

$$skewness = E \begin{bmatrix} \epsilon_1\epsilon_1\epsilon_1 & \epsilon_1\epsilon_1\epsilon_2 & \epsilon_1\epsilon_2\epsilon_1 & \epsilon_1\epsilon_2\epsilon_2 \\ \epsilon_2\epsilon_1\epsilon_1 & \epsilon_2\epsilon_1\epsilon_2 & \epsilon_2\epsilon_2\epsilon_1 & \epsilon_2\epsilon_2\epsilon_2 \end{bmatrix}.$$

Etc.

3.3.2 Solution structures

When a model is solved the solution is returned in the form of a structure. The exact structure returned depends on the solution method.

First-order perturbation The first-order perturbation solution takes the following form:

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{h}_\mathbf{x}\mathbf{x}_t + \mathbf{k}\boldsymbol{\epsilon}_{t+1}, \\ \mathbf{y}_t &= \mathbf{g}_\mathbf{x}\mathbf{x}_t. \end{aligned}$$

The solution structure for a stochastic first-order perturbation has the following fields:

- hbar — The steady state of the state variables
- hx — The first-order coefficients in the state-transition equation
- k — The loading matrix on the shocks in the state-transition equation.
- gbar — The steady state of the jump variables
- gx — The first-order coefficients in the jump's equation
- sigma — An identy matrix
- grc — The number of eigenvalues with modulus greater than cutoff.
- Soln_type — Either "determinate", "indeterminate", or "unstable".

The solution to a deterministic model has the same fields as the stochastic solution with the exceptions of **k** and sigma.

Second-order perturbation The second-order perturbation solution takes the following form:

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{h}_\mathbf{x}\mathbf{x}_t + \frac{1}{2}\mathbf{h}_\mathbf{ss} + \frac{1}{2}\left(\mathbf{I}\otimes\mathbf{x}_t\right)\mathbf{h}_\mathbf{xx}\left(\mathbf{I}\otimes\mathbf{x}_t\right) + \mathbf{k}\boldsymbol{\epsilon}_{t+1}, \\ \mathbf{y}_t &= \mathbf{g}_\mathbf{x}\mathbf{x}_t + \frac{1}{2}\mathbf{g}_\mathbf{ss} + \frac{1}{2}\left(\mathbf{I}\otimes\mathbf{x}_t\right)\mathbf{g}_\mathbf{xx}\left(\mathbf{I}\otimes\mathbf{x}_t\right). \end{aligned}$$

The solution structure for a stochastic second-order perturbation has the following fields:

- hbar — The steady state of the state variables
- hx — The first-order coefficients in the state-transition equation
- hss — The second-order stochastic adjustment to the mean in the state-transion equation
- hxx — The second-order coefficients in the state-transition equation
- k — The loading matrix on the shocks in the state-transition equation.
- gbar — The steady state of the jump variables
- gx — The first-order coefficients in the jump's equation
- gss — The second-order stochastic adjustment to the mean in the jump's equation
- gxx — The second-order coefficients in the jump's equation
- sigma — An identy matrix
- grc — The number of eigenvalues with modulus greater than cutoff.
- Soln_type — Either "determinate", "indeterminate", or "unstable".

The solution to a deterministic model has the same fields as the stochastic solution with the exceptions of $\mathbf{h_{ss}}$, $\mathbf{k}$, $\mathbf{g_{ss}}$, and sigma.

Third-order perturbation The third-order perturbation solution takes the following form:

$$\begin{aligned}
\mathbf{x}_{t+1} &= \mathbf{h_x}\mathbf{x}_t + \frac{1}{2}\mathbf{h_{ss}} + \frac{1}{2}\mathbf{h_{xx}}\left(\mathbf{x}_t \otimes \mathbf{x}_t\right) + \frac{1}{6}\mathbf{h_{sss}} + \frac{1}{6}\mathbf{h_{ssx}}\mathbf{x}_t + \frac{1}{6}\mathbf{h_{xxx}}\left(\mathbf{x}_t \otimes \mathbf{x}_t \otimes \mathbf{x}_t\right) + \mathbf{k}\boldsymbol{\epsilon}_{t+1}, \\
\mathbf{y}_t &= \mathbf{g_x}\mathbf{x}_t + \frac{1}{2}\mathbf{g_{ss}} + \frac{1}{2}\mathbf{g_{xx}}\left(\mathbf{x}_t \otimes \mathbf{x}_t\right) + \frac{1}{6}\mathbf{g_{sss}} + \frac{1}{6}\mathbf{g_{ssx}}\mathbf{x}_t + \frac{1}{6}\mathbf{g_{xxx}}\left(\mathbf{x}_t \otimes \mathbf{x}_t \otimes \mathbf{x}_t\right).
\end{aligned}$$

The solution structure for a stochastic third-order perturbation has the following fields:

- hbar — The steady state of the state variables
- hx — The first-order coefficients in the state-transition equation
- hss — The second-order stochastic adjustment to the mean in the state-transion equation
- hxx — The second-order coefficients in the state-transition equation
- hsss — The third-order stochastic adjustment to the mean in the state-transition equation

- hssx — The skewness adjustment othe state-transition equation
- hxxx — The third-order coefficents in the state-transition equation
- k — The loading matrix on the shocks in the state-transition equation.
- gbar — The steady state of the jump variables
- gx — The first-order coefficients in the jump's equation
- gss — The second-order stochastic adjustment othe mean in the jump's equation
- gxx — The second-order coefficients in the jump's equation
- gsss — The third-order stochastic adjustment to the mean in the jump's equation
- gssx — The skewness adjustment in the jump's equation
- gxxx — The third-order coefficients in the jump's equation
- sigma — An identy matrix
- grc — The number of eigenvalues with modulus greater than cutoff.
- Soln_type — Either "determinate", "indeterminate", or "unstable".

The solution to a deterministic model has the same fields as the stochastic solution with the exceptions of $\mathbf{h_{ss}}$, $\mathbf{h_{sss}}$, $\mathbf{h_{ssx}}$, $\mathbf{k}$, $\mathbf{g_{ss}}$, $\mathbf{g_{sss}}$, $\mathbf{g_{ssx}}$, and sigma.

Chebyshev solution The solution structure for the Chebyshev solution has the following fields:

- variables — A vector of arrays containing the solution for each variable
- weights — A vector of arrays containing the weights for the Chebyshev polynomials
- nodes — A vector of vectors containing the Chebyshev nodes
- order — The order of the Chebyshev polynomials
- domain — The domain for the state variables
- sigma — The variance-covariance matrix for the shocks
- iteration_count — The number of iterations needed to achieve convergence

The solution to a deterministic model has the same fields with the exception of sigma.

Smolyak solution The solution structure for the Smolyak solution has the following fields:

- variables — A vector of arrays containing the solution for each variable
- weights: — A vector of vectors containing the weights for the Chebyshev polynomials
- grid — A matrix containing the Smolyak grid
- multi_index — A matrix containing the multi-index underlying the polynominals
- layer — The number of layers in the approximation
- domain — The domain for the state variables
- sigma — The variance-covariance matrix for the shocks
- iteration_count — The number of iterations needed to achieve convergence

The solution to a deterministic model has the same fields with the exception of sigma.

Piecewise linear solution The solution structure for the piecewise linear solution has the following fields:

- variables — A vector of arrays containing the solution for each variable
- nodes — A vector of vectors containing the Chebyshev nodes
- domain — The domain for the state variables
- sigma — The variance-covariance matrix for the shocks
- iteration_count — The number of iterations needed to achieve convergence

The solution to a deterministic model has the same fildls with the exception of sigma.

4 Post-solution analysis

Once you have solved your model there are many things that you might want to use the solution for. Some of the more obvious things, such as simulating data from the solution and computing impulse response functions have been programmed into SolveDSGE to make things easier for you.

4.1 Simulation

To simulate data from the solution to a model the function to use is simulate(), whose arguments are a model solution, an initial state, and the number of observations to simulate. An optimal final argument is the seed for the random number generator. An example of simulate() in action might be

data_states, data_jumps = simulate(soln,[0.0, 25.0],100000)

As this example makes clear, the simulate function returns two 2D arrays. The first array contains simulated data for the state variables, the second array contains simulated data for the jump variables. The simulate function can be applied to both stochastic and deterministic models. An initial burn-in sample of 1000 observations is also generated and discarded.

4.2 Impulse response functions

Impulse responses are obtained using the impulses() function, which takes three arguments: the model solution, the length of the impulse response function (number of periods), the number of the shock to apply the impulse to, and the number of repetitions to use for the Monte Carlo integration. Responses to both a positive and a negative innovation are generated. An optimal final argument is the seed for the random number generator. An example of impulses() in use might be

pos_*responses, neg_responses = impulses(soln,50,1,10000)*

For the nonlinear solutions (second-order perturbation, third-order perturbation, and the projection-based solutions) the initial state is "integrated-out" via a Monte Carlo that averages over draws taken from the unconditional distribution of the state variables. At this stage in the package's development, the impulses need to be computed one shock at a time; this will probably change at some point.

References

[1] Andreasen, M., Fernández-Villaverde, J., and J. Rubio-Ramirez, (2017), "The Pruned State-Space System for Non-Linear DSGE Models: Theory and Empirical Applications", *Review of Economic Studies*, 0, pp. 1—49.

[2] Binning, A., (2013), "Third-order approximation of dynamic models without the use of tensors", *Norges Bank Working Paper* 2013–13.

[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] Judd, K. (1992), "Projection Methods for Solving Aggregate Growth Models", *Journal of Economic Theory*, 58, pp.410—452.

[5] Judd, K., Maliar, L., Maliar, S., and R. Valero, (2014), "Smolyak Method for Solving Dynamic Economic Models: Lagrange Interpolation, Anisotropic Grid and Adaptive Domain", *Journal of Economic Dynamics and Control*, 44, pp. 92—123.

[6] Judd, K., Maliar, L., Maliar, S., and I. Tsener, (2017), "How to Solve Dynamic Stochastic Models Computing Expectations just Once", *Quantitative Economics*, 8, pp.851—893.

[7] 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.

[8] Levintal, O., (2017), "Fifth-Order Perturbation Solution to DSGE models", *Journal of Economic Dynamics and Control*, 80, pp. 1–16.

[9] Potter, S., (2000), "Nonlinear Impulse Response Functions", *Journal of Economic Dynamics and Control*, 24, pp. 1425—1446.