Matlab Codes to Replicate "Solution Methods for Models with Rare Disasters"

Jesús Fernández-Villaverde
University of Pennsylvania

Oren Levintal*
Interdisciplinary Center (IDC) Herzliya

June 30, 2016

1 Introduction

This is a technical note that explains how to use the MATLAB codes to replicate the results in Fernández-Villaverde and Levintal (2016). The codes implement perturbation solutions of orders 1 to 5, Taylor projection of orders 1 to 3, and Smolyak collocation of any order.

A simple example can be found in folder `Simple_Example`, where the neoclassical model is solved by all methods.

If you are interested only in the Taylor projection method, read the enclosed User Guide, which explains this method in further detail.

*Correspondence: jesusfv@econ.upenn.edu (Fernández-Villaverde) and oren.levintal@idc.ac.il (Oren Levintal). Fernández-Villaverde gratefully acknowledges financial support from the National Science Foundation under Grant SES 1223271.

2 External files

The following external files are included in the package:

- The files `Monomials_1.m` and `Monomials_2.m` written by Judd, Maliar, Maliar, and Valero (2014) to discretize Gaussian shocks by monomial rules.
- The file `sym_Smolyak_Polynomial.m` is a symbolic version of `Smolyak_Polynomial.m` written by Judd, Maliar, Maliar, and Valero (2014) to compute the Smolyak basis function.
- The files `Smolyak_Grid.m` and `Smolyak_Elem_Isotrop.m` written by Judd, Maliar, Maliar, and Valero (2014) construct a Smolyak grid.
- The folder `Perturbation` contains the perturbation package written by Levintal (2015) to compute a fifth-order perturbation solution. It uses the external files `gx_hx.m` written by Schmitt-Grohé and Uribe (2004) and `gensylv.mexw64` written by `Dynare` team.
- The folder `Taylor_projection` contains the Taylor projection package written by Levintal (2016).

3 Installation

Requirements: MATLAB, MATLAB symbolic toolbox, MATLAB optimization toolbox, MATLAB control system toolbox (optional), Intel Fortran compiler (op-

tional). The package was tested on MATLAB version R2014b working on WINDOWS10 (64bit).

To install the package follow these steps:

- Add the folder `Solution_Methods` and its subfolders to the search path. For example, if the folder location is `c:\Solution_Methods` type in the command prompt:

  `addpath(genpath('c:\Solution_Methods')).`

- **MEX files**: The package includes MEX files that were compiled on WINDOWS10 (64bit). In case that these files do not work on your computer go to folder `Solution_Methods\Taylor_Projection\MEX_files` and run the file `do_mex.m` to compile the FORTRAN source codes on your system. To do so, you will need an Intel Fortran compiler that works with MATLAB.

4 Simple example

The folder `Simple_Example` solves the neoclassical growth model by the three solution methods: perturbation, Taylor projection and Smolyak collocation. Here is a brief description of these codes.

4.1 Perturbation

The folder `Simple_Example\Perturbation` solves the neoclassical growth model by perturbation. The model is solved in two stages. The first stage is performed by `prepare_model.m`. This file differentiates the model and prepares files and data that are used when the model is solved. The second stage is performed by `solve_model.m`, which solves the model for given parameter values.

The model is defined in `prepare_model.m`. The notation of the model follows Schmitt-Grohé and Uribe (2004):

$$\mathbb{E}_t f\left(y_{t+1}, y_t, x_{t+1}, x_t\right) = 0, \tag{1}$$

$$x_{t+1} = h\left(x_t\right) + \eta \epsilon_{t+1}, \tag{2}$$

$$y_t = g\left(x_t\right), \tag{3}$$

$$h\left(x_t\right) = \begin{pmatrix} \tilde{h}\left(x_t\right) \\ \Phi\left(x_t^2\right) \end{pmatrix}, \ \eta = \begin{pmatrix} 0 \\ \tilde{\eta} \end{pmatrix}. \tag{4}$$

The state and control variables are defined by symbolic vectors `x` and `y` for period t and `xp` and `yp` for period $t+1$. The model conditions are defined by the symbolic vector `f`.

The algorithm allows to define the function Φ by a separate symbolic variable `Phi`. This function is the lower block of h, which is known. It determines the evolution law of the exogenous state variables. Since this function is known, it is not approximated by the algorithm. Therefore, the evolution law of the exogenous state

variables should not be included in `f`.

After running `prepare_model.m`, you can solve the model by running `solve_model.m`. This file defines the parameter values, the matrix η, the steady state and the cross moments of the shocks up to fifth order. These cross moments are stored in structure `M`, where `M.M2` is $\mathbb{E}\epsilon^{\otimes 2}$, `M.M3` is $\mathbb{E}\epsilon^{\otimes 3}$, and so on up to fifth order. The only restriction on the distribution of ϵ is that $\mathbb{E}\epsilon = 0$.

Finally, you need to choose a solver for the Sylvester equation solved by the algorithm. There are three possibilities: 1. `'vectorize'` is good only for small models. 2. `'dlyap'` is good for larger models, but requires the MATLAB control system toolbox. 3. `'gensylv'` is recommended for very large models. It applies the algorithm of Kameník (2005), which is provided by `Dynare` as a compiled MEX file.[1] The current package includes the WINDOWS (64bit) version of this MEX file. If you work on a different operating system, you can get the appropriate version by downloading `Dynare` from `www.dynare.org` and searching for the MEX file `gensylv`. Then, add this file to folder `Solution_Methods\Perturbation\gensylv`.

The output of the perturbation algorithm is a structure, whose fields are the derivatives of g and h with respect to the state variables and the perturbation parameter. Note that the perturbation parameter is treated as an additional state variable, which is located as the last element of the state vector. Hence, derivatives with respect to the perturbation parameter are stored as derivatives with respect to the last element of `x`. For example `gx(:,end)` is the derivative of g with respect to the perturbation parameter (denoted g_σ in Schmitt-Grohé and Uribe (2004)). For

[1]For a description of `Dynare` see Adjemian, Bastani, Juillard, Mihoubi, Ratto, and Villemot (2011).

further details, see Levintal (2015).

4.2 Taylor projection

The folder `Simple_Example\Taylor projection` performs the Taylor projection method. This algorithm is also implemented in two stages, which are performed by `prepare_model.m` and `solve_model.m`. The notation of the model is identical to the perturbation algorithm. The only difference is that it allows to define auxiliary variables and auxiliary functions to speed up computations. For example, you can declare a symbolic variable `logmpkp`, which is defined by the symbolic function `logmpkp_=log(ALPHA)+logap+(ALPHA-1)*logkp`. When `logmpkp` is used in a certain model condition (e.g. in the Euler condition), the algorithm differentiates the model condition by using the chain rule. This speeds up the differentiation of the model, in particular for complicated functional forms (e.g. Epstein-Zin preferences).

The model is solved in `solve_model.m`. Few things should be noted. First, the distribution of the shocks ϵ should be discretized. This is done by the realization matrix `nep` and the probability vector `P`. Second, you need to supply an initial guess. The easiest way is to solve the model near the steady state and use a perturbation solution for the initial guess. To get a perturbation solution, you do not need to run the perturbation algorithm separately. You can do it directly from the Taylor projection algorithm, by the function `get_pert`. Finally, you need to choose the nonlinear solver (could be a simple Newton method, or some other version that is available by the MATLAB functions `fsolve` or `lsqnonlin`). For further details, see the enclosed User Guide.

4.3 Smolyak collocation

The Smolyak algorithm is also performed in two stages. The model is defined in `prepare_model` exactly as in the Taylor projection algorithm. The solution is computed in `solve_model`. First, we construct a Smolyak grid by the function `Smolyak_Grid.m` written by Judd, Maliar, Maliar, and Valero (2014). We use this (scaled) grid to construct the unscaled grid (the collocation points). Then, we take an initial guess and transform it into a Smolyak polynomial by interpolation. Finally, we solve the Smolyak coefficients by the Newton method.

5 Replication of the results

The file `replicate_all_results.m` solves the 8 models described in Fernández-Villaverde and Levintal (2016) and reports all the results of the paper. It takes about 9 hours to run. If your computer does not have sufficient memory, some algorithms may fail to solve the largest models. We were able to solve all models with 16GB RAM. It is possible to run only the cheap solutions, which takes about 1 hour.

The next sections describe the different components of the package.

5.1 The models

The models are defined in folder `all_models`. This folder contains 8 subfolders for the 8 versions of the model. In each subfolder, the model is defined in two ways: one for the perturbation algorithm and the other for the Taylor projection and Smolyak algorithms.

The equilibrium conditions of the full model are provided in the appendix of this note. Smaller versions are obtained by dropping or changing some equations, as explained in Fernández-Villaverde and Levintal (2016). We recommend to start with the perturbation code of the full model, which follows closely the model conditions presented in the appendix.

The Taylor projection and Smolyak algorithms use a slightly different code. The differences from the perturbation code are the following: (1) The size of the system is reduced as much as possible by substituting out control variables that can be defined as functions of other control and state variables. This reduces computational costs and also improves accuracy, because we approximate only the necessary variables. (2) The code uses auxiliary variables to speed up the differentiation of the model, as explained in section 4.2.

5.2 Parameters

The package contains two sets of parameters. The no-disaster parameters are defined in folder `params_SS0`. The disaster parameters are defined in folder `params_SS1`. The other codes call these parameters, whenever necessary.

5.3 Preparing files

The models are solved in two stages. The first stage prepares files and data that are necessary for solving the models. This stage is performed by the MATLAB code `prepare_all_models.m`. You can specify which models to solve and for which orders. Running this code for all models and orders may take about half an hour.

The generated files and data are stored in three folders generated automatically: `files_for_perturbation`, `files_for_smolyak` and `files_for_TaylorProjection`.

5.4 Computing the solutions

The second stage solves the models. This is done by the MATLAB code `do_solve.m`. This file calls `solve_all_models.m` that solves the models by perturbation, Taylor projection and Smolyak collocation. Again, you can specify a subset of models or solutions. However, you must have the 3rd order perturbation solution, because it is used as an initial guess for the other solutions.

5.5 Getting the results

After you run `do_solve.m`, you can get the results reported in Fernández-Villaverde and Levintal (2016). The file `accuracy_and_simulation.m` computes the model residuals across the ergodic set, and performs simulations of all models, starting at the steady state. It takes few minutes to run.

The file `present_tables.m` produces the tables in the paper. It takes few seconds.

The files `make_Figure_I.m`, `make_Figure_II.m`, `make_Figure_III.m` and `make_Figure_IV.m` produce the figures. To run these files, you must first solve version no. 8 by all solution methods.

Appendix: the detrended model

This section follows section 3.2 of the baseline DSGE model of Fernández-Villaverde and Rubio-Ramírez (2006) to detrend the model conditions described in Fernández-Villaverde and Levintal (2016). Here, we provide the full version of the model. It is coded in the MATLAB file:

`all_models\RBC_EZ_adjCost_Calvo_Taylor2_shock3\Perturbation\get_model_pert.m`.

Simpler versions of the model are coded in their respective folders by dropping/changing the relevant equations.

To stationarize the model we define: $\tilde{c}_t = \frac{c_t}{z_t}$, $\tilde{\lambda}_t = \lambda_t z_t^{\psi}$, $\tilde{r}_t = r_t \mu_t$, $\tilde{q}_t = q_t \mu_t$, $\tilde{x}_t = \frac{x_t}{z_t}$, $\tilde{w}_t = \frac{w_t}{z_t}$, $\tilde{k}_t = \frac{k_t}{z_t \mu_t}$, $\tilde{k}_t^* = \frac{k_t^*}{z_t \mu_t}$, $\tilde{y}_t = \frac{y_t}{z_t}$, $\tilde{U}_t = \frac{U_t}{z_t}$, $\tilde{U}_{l,t} = \frac{U_{l,t}}{z_t}$, $\tilde{V}_t = \frac{V_t}{z_t}$. In addition, denote: $\hat{A}_t = \frac{A_t}{A_{t-1}}$, $\hat{\mu}_t = \frac{\mu_t}{\mu_{t-1}}$, $\hat{z}_t = \frac{z_t}{z_{t-1}}$. Last, the detrended utility variables are normalized by their steady state value to avoid scaling problems.

Under the notation of Schmitt-Grohé and Uribe (2004), the exogenous state variables must be linear in the shocks. Hence, we define the following exogenous state variables:

$$d_{t+1} = \mu^d + \left(\epsilon_{d,t+1} - \mu^d\right) \tag{5}$$

$$\log \theta_{t+1} = (1 - \rho_\theta) \log \bar{\theta} + \rho_\theta \log \theta_t + \sigma_\theta \epsilon_{\theta,t+1} \tag{6}$$

$$z_{A,t+1} = \sigma_A \epsilon_{A,t+1} \tag{7}$$

$$\log \hat{\mu}_{t+1} = \Lambda_\mu + \sigma_\mu \epsilon_{\mu,t+1} \tag{8}$$

$$m_{t+1} = \sigma_m \epsilon_{m,t+1} \tag{9}$$

$$\xi_{t+1} = \rho_\xi \xi_t + \sigma_\xi \epsilon_{\xi,t+1} \tag{10}$$

The disaster state variable is denoted d_t. It is determined by the disaster shock $\epsilon_{d,t+1}$, which takes the values 1 or 0. The mean of this shock is μ^d. Since the mean is nonzero, the shock is demeaned in (5). The state variable $\log \theta_t$ is the log disaster size. The state variable $z_{A,t}$ is introduced to capture Gaussian TFP shocks, which affect the TFP growth $\log \hat{A}_t$. The state variable $\log \hat{\mu}_t$ denotes the growth of investment technology. Finally, m_t and ξ_t are the monetary shock and the time preference shock, respectively.

The following variables depend only on the exogenous variables:

$$\log \hat{A}_t = \Lambda_A + z_{A,t} - (1 - \alpha)\, d_t \theta_t$$
$$\log \hat{z}_t = \frac{1}{1 - \alpha} \log \hat{A}_t + \frac{\alpha}{1 - \alpha} \log \hat{\mu}_t.$$

The model conditions are given by the following equations:

$$\left(\frac{\tilde{V}_t}{\tilde{V}^{ss}}\right)^{1-\psi} = \left(\frac{\tilde{U}_t}{\tilde{U}^{ss}}\right)^{1-\psi} \left(\frac{\tilde{U}^{ss}}{\tilde{V}^{ss}}\right)^{1-\psi} + \beta E_t \left(\left(\frac{\tilde{V}_{t+1}}{\tilde{V}^{ss}}\right)^{1-\gamma} \hat{z}_{t+1}^{1-\gamma}\right)^{\frac{1-\psi}{1-\gamma}} \tag{11}$$

$$\tilde{U}_t = \tilde{c}_t \left(1-l_t\right)^{\nu} e^{\xi_t} \tag{12}$$

$$U_{c,t} = \left(1-l_t\right)^{\nu} e^{\xi_t} \tag{13}$$

$$\tilde{U}_{l,t} = -\nu \tilde{c}_t \left(1-l_t\right)^{\nu-1} e^{\xi_t} \tag{14}$$

$$\left(1-\psi\right)\left(\tilde{U}_t\right)^{-\psi} \tilde{U}_{l,t} = -\tilde{\lambda}_t \tilde{w}_t \tag{15}$$

$$\left(1-\psi\right)\left(\tilde{U}_t\right)^{-\psi} U_{c,t} = \tilde{\lambda}_t \tag{16}$$

$$M_{t+1} = \beta \frac{\tilde{\lambda}_{t+1}}{\tilde{\lambda}_t} \left(\hat{z}_{t+1}\right)^{-\psi} \frac{\left(\tilde{V}_{t+1}/\tilde{V}^{ss}\right)^{\psi-\gamma} \left(\hat{z}_{t+1}\right)^{\psi-\gamma}}{E_t\left(\left(\tilde{V}_{t+1}/\tilde{V}^{ss}\right)^{1-\gamma} \left(\hat{z}_{t+1}\right)^{1-\gamma}\right)^{\frac{\psi-\gamma}{1-\gamma}}} \tag{17}$$

$$E_t\left(M_{t+1} \exp\left(-d_{t+1}\theta_{t+1}\right) \frac{1}{\hat{\mu}_{t+1}} \left[\tilde{r}_{t+1} + \tilde{q}_{t+1}\left(1-\delta\right)\right]\right) = \tilde{q}_t \tag{18}$$

$$1 = \tilde{q}_t \left[1 - S\left[\frac{\tilde{x}_t}{\tilde{x}_{t-1}}\hat{z}_t\right] - S'\left[\frac{\tilde{x}_t}{\tilde{x}_{t-1}}\hat{z}_t\right] \frac{\tilde{x}_t}{\tilde{x}_{t-1}}\hat{z}_t\right] + \tag{19}$$
$$+ E_t\left(M_{t+1}\tilde{q}_{t+1} S'\left[\frac{\tilde{x}_{t+1}}{\tilde{x}_t}\hat{z}_{t+1}\right]\left(\frac{\tilde{x}_{t+1}}{\tilde{x}_t}\hat{z}_{t+1}\right)^2\right)$$

$$\tilde{y}_t = \tilde{c}_t + \tilde{x}_t \tag{20}$$

$$\tilde{k}_t^* - \left(1-\delta\right)\tilde{k}_t - \left(1 - S\left[\frac{\tilde{x}_t}{\tilde{x}_{t-1}}\hat{z}_t\right]\right)\tilde{x}_t = 0 \tag{21}$$

$$\tilde{k}_t = \frac{\tilde{k}_{t-1}^*}{\hat{z}_t \hat{\mu}_t} \exp\left(-d_t\theta_t\right) \tag{22}$$

$$\tilde{q}_t^e = E_t\left(M_{t+1}\hat{z}_{t+1}\left(\tilde{div}_{t+1} + \tilde{q}_{t+1}^e\right)\right) \tag{23}$$

$$\tilde{div}_t = \tilde{y}_t - \tilde{w}_t l_t - \tilde{x}_t \tag{24}$$

$$q_t^f = E_t M_{t+1} \tag{25}$$

The Calvo block:

$$\tilde{g}_t^1 = mc_t\tilde{y}_t^d + \theta_p E_t M_{t+1}\left(\frac{\Pi_t^\chi}{\Pi_{t+1}}\right)^{-\epsilon}\tilde{g}_{t+1}^1\hat{z}_{t+1} \tag{26}$$

$$\tilde{g}_t^2 = \Pi_t^*\tilde{y}_t^d + \theta_p E_t M_{t+1}\left(\frac{\Pi_t^\chi}{\Pi_{t+1}}\right)^{1-\epsilon}\left(\frac{\Pi_t^*}{\Pi_{t+1}^*}\right)\tilde{g}_{t+1}^2\hat{z}_{t+1} \tag{27}$$

$$\epsilon\tilde{g}_t^1 = (\epsilon - 1)\,\tilde{g}_t^2 \tag{28}$$

$$1 = \theta_p\left(\frac{\Pi_{t-1}^\chi}{\Pi_t}\right)^{1-\epsilon} + (1-\theta_p)\left(\Pi_t^*\right)^{1-\epsilon} \tag{29}$$

$$mc_t = \left(\frac{1}{1-\alpha}\right)^{1-\alpha}\left(\frac{1}{\alpha}\right)^{\alpha}\tilde{w}_t^{1-\alpha}\tilde{r}_t^{\alpha} \tag{30}$$

Optimal factor composition:

$$\frac{\tilde{k}_t}{l_t} = \frac{\alpha}{1-\alpha}\frac{\tilde{w}_t}{\tilde{r}_t} \tag{31}$$

Aggregate conditions:

$$\tilde{y}_t = \frac{\frac{\hat{A}_t}{\hat{z}_t}\left(\tilde{k}^*_{t-1}\exp\left(-d_t\theta_t\right)\right)^{\alpha} l_t^{1-\alpha} - \phi}{v_t^p} \tag{32}$$

$$v_t^p = \theta_p\left(\frac{\Pi_{t-1}^{\chi}}{\Pi_t}\right)^{-\epsilon} v_{t-1}^p + \left(1-\theta_p\right)\left(\Pi_t^*\right)^{-\epsilon}, \tag{33}$$

The Taylor rule:

$$\frac{R_t}{R} = \left(\frac{R_{t-1}}{R}\right)^{\gamma_R}\left(\left(\frac{\Pi_t}{\Pi}\right)^{\gamma_\Pi}\left(\frac{\frac{\tilde{y}_t}{\tilde{y}_{t-1}}\hat{z}_t}{\exp\left(\Lambda_y\right)}\right)^{\gamma_y}\right)^{1-\gamma_R} e^{m_t} \tag{34}$$

$$1 = E_t M_{t+1}\frac{R_t}{\Pi_{t+1}} \tag{35}$$

We define the state of the economy by the endogenous variables $\log\tilde{k}^*_{t-1}$ $\log\tilde{x}_{t-1}$, $\log\Pi_{t-1}$, $\log v^p_{t-1}$, $\log\tilde{y}_{t-1}$ and $\log R_{t-1}$, and the exogenous variables d_t, $\log\theta_t$, $z_{A,t}$, $\log\hat{\mu}_t$ m_t and ξ_t.

In flexible price models we use the following conditions instead of (26)-(35):

$$\tilde{r}_t = \alpha\hat{A}_t\hat{\mu}_t\left(\tilde{k}^*_{t-1}\exp\left(-d_t\theta_t\right)\right)^{\alpha-1} l_t^{1-\alpha} \tag{36}$$

$$\tilde{w}_t = \left(1-\alpha\right)\frac{\hat{A}_t}{\hat{z}_t}\left(\tilde{k}^*_{t-1}\exp\left(-d_t\theta_t\right)\right)^{\alpha} l_t^{-\alpha} \tag{37}$$

$$\tilde{y}_t = \frac{\hat{A}_t}{\hat{z}_t}\left(\tilde{k}^*_{t-1}\exp\left(-d_t\theta_t\right)\right)^{\alpha} l_t^{1-\alpha} - \phi \tag{38}$$

5.6 Technical tips

We found that accuracy is higher when utility variables are changed to logs. This is crucial for perturbation solutions, but improves also the other solutions. In addition, it is recommended to normalize the utility variables by their steady state values to avoid scaling problems.

Equity prices should not be normalized by dividends, because dividends could be negative. Instead, it is better to normalize equity prices by the TFP trend variable z_t, as all other variables.

The projection algorithms (i.e., Taylor projection and Smolyak collocation) work smoothly when all variables are bounded within their natural domain. For instance, a positive variable should be changed to log. Similarly, a variable that is bounded within $(0, 1)$ is changed to $\frac{\exp(x)}{1+\exp(x)}$.

In addition, projection solutions are more accurate when the number of endogenous variables that are approximated is minimal. Hence, we substitute out control variables as much as possible. One of the variables that can be substituted out is Π_t^*, because it can be defined as a function of the control variable Π_t and the (predetermined) state variable Π_{t-1} through (29):

$$1 = \theta_p \left(\frac{\Pi_{t-1}^{\chi}}{\Pi_t} \right)^{1-\epsilon} + (1 - \theta_p) \left(\Pi_t^* \right)^{1-\epsilon}.$$

The problem is that we cannot ensure that $\left(\Pi_t^*\right)^{1-\epsilon}$ is always positive. For instance, if $\theta_p \left(\frac{\Pi_{t-1}^{\chi}}{\Pi_t} \right)^{1-\epsilon}$ is larger than 1, then Π_t^* is a complex number and the Newton algorithm would fail.

To resolve this problem, note that:

$$\Pi_t^{1-\epsilon} = \theta_p \Pi_{t-1}^{\chi(1-\epsilon)} + (1-\theta_p)\left(\Pi_t \Pi_t^*\right)^{1-\epsilon}. \tag{39}$$

Hence, instead of substituting out Π_t^*, we introduce a new control variable $aux_t = \Pi_t \Pi_t^*$. Then, we approximate the control variable $\log aux_t$ and the state variable $\log \Pi_{t-1}$, while substituting out the control variable Π_t through (39), which is positive by construction. This transformation achieves our goal to reduce the number of control variables but maintains the restriction that the inflation variables should be positive.

References

Adjemian, S., H. Bastani, M. Juillard, F. Mihoubi, M. Ratto, and S. Villemot (2011): "Dynare: Reference Manual, Version 4," Dynare Working Papers 1, CEPREMAP.

Fernández-Villaverde, J., and O. Levintal (2016): "Solution Methods for Models with Rare Disasters," *Manuscript, University of Pennsylvania.*

Fernández-Villaverde, J., and J. F. Rubio-Ramírez (2006): "A Baseline DSGE Model," Discussion paper, University of Pennsylvania.

Judd, K. L., L. Maliar, S. Maliar, 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, 92–123.

Kameník, O. (2005): "Solving SDGE Models: A New Algorithm for the Sylvester Equation," *Computational Economics*, 1412.8659v1, 167–187.

Levintal, O. (2015): "Fifth Order Perturbation Solution to DSGE Models," *Manuscript, Interdisciplinary Center Herzliya*.

——— (2016): "Taylor Projection: A New Solution Method for Dynamic General Equilibrium Models," *Manuscript, Interdisciplinary Center Herzliya*.

Schmitt-Grohé, S., and M. Uribe (2004): "Solving Dynamic General Equilibrium Models Using a Second-Order Approximation to the Policy Function," *Journal of Economic Dynamics and Control*, 28, 755–775.