Python Algorithm for Solving Dynamic General Equilibrium Models Using Second-Order Approximation[*]

Sarunas Girdenas[†]and Hyun Chang Yi[‡]

June 28, 2014

Abstract

In this paper we revise Schmitt-Grohe and Uribe (2004) and Klein 200) proposed solution algorithms for dynamic stochastic general equilibrium models. We have translated it to Python programming language and at the end of this paper we provide a short guide on how to use the code. We replicate the three examples given in the original paper by Schmit-Grohe and Uribe (2004).

Keywords: DSGE models, computational economics, second-order approximation, Python

1 Introduction

Linear approximation of the first and second order to the dynamic stochastic general equilibrium (DSGE) models is probably the most common way of obtaining the solution. Papers by Schmit-Grohe and Uribe (2004) and Klein (2000) paved the methodological and computational way of handling such models. In this paper we we aim to extend their framework for Python programming language. We believe that it is important because of the following reasons: firstly, Python can reduce the computational time (hence increase efficiency), secondly, the structure of the code is much simpler and more intuitive in Python than in Matlab. In this paper we aim to provide a Python code which solves dynamic stochastic general equilibrium models using second-order approximation to policy function. We base our code on Schmitt-Grohe and Uribe (2004) paper.

2 Solution Algorithm

In this section we briefly revise theoretical solution algorithm as presented in Schmit-Grohe and Uribe (2004). We provide results for first-order and second-order approximation. This is a shorter version of the original paper, it is provided for the sake of better understanding of how the code (solution algorithm) is structured.

2.1 Model Solution

We can write dynamic general equilibrium model in the following general form as

$$E_t f(y_{t+1}, y_t, x_{t+1}, x_t) = 0 \tag{1}$$

[*]For discussion and advice we would like to thank to Tatiana Damjanovic, Tatiana Kirsanova, Keqing Liu, Jack R. Rogers and Angele Urbonaviciute.

[†]Address: Economics Department, Business School, University of Exeter, Streatham Court, Rennes Drive. EX4 4PU, UK. Email: sg325@exeter.ac.uk

[‡]Bank of Korea ...

where $E_t$is mathematical expectation conditional on information available at t. Then vector $x_t$is of dimensions $n_x \times 1$ and denotes predetermined variables and the vector $y_t$of the same dimensions of non-predetermined variables. Here we follow Klein (2000) and call variable *predetermined* when its one period ahead forecast error is exogenous and its initial value is given exogenously. Non-predetermined variables (also called forward looking) are those with endogenously given one period ahead forecast errors and initial values. Then we can partition state vector x_t as $x_t = [x_t^1;\, x_t^2]$'. The vector x_t^1consists of endogenous predetermined state variables and the vector x_t^2 of exogenous state variables. Then following Samuelson (1970), Jin and Judd (2002) and Schmit-Grohe and Uribe (2004) we set the following restrictions on the exogenous process of x_t^2:

$$x_{t+1}^2 = \Lambda x_t^2 + \tilde{\eta}\sigma\varepsilon_{t+1} \tag{2}$$

where x_t^2 and $\varepsilon_{t+1}$are both of dimensions $n_\varepsilon \times 1$. The ε_t is assumed to be i.i.d. with zero mean and I variance/covariance matrix, η (of size $n_\varepsilon \times n_\varepsilon$) is matrix of known parameters and $\sigma \geq 0$ is also known. To ensure stability, the assumption that Λhas eigenvalues which in modulus are less than one.

As an example, Schmit-Grohe and Uribe (2004) consider neoclassical growth model which has the following equilibrium conditions

$$c_t^{-\gamma} = \beta E_t\left[c_{t+1}^{-\gamma}\left(\alpha A_{t+1}k_{t+1}^{\alpha-1} + 1 - \delta\right)\right] \tag{3}$$

$$c_t + k_{t+1} = A_t k_t^\alpha + (1-\delta)\,k_t \tag{4}$$

$$\ln A_{t+1} = \rho \ln A_t + \sigma\varepsilon_{t+1} \tag{5}$$

for all $t \geqslant 0$ with given $k_0$and A_0. Assuming market clearing condition $y_t = c_t$and partitioning state vector $x_t = [k_t;\, \ln A_t]'$ we obtain

$$E_t f(y_{t+1}, y_t, x_{t+1}, x_t) = \left[\begin{array}{c} y_{1,t}^{-\gamma} - \beta E_t\left[y_{1,t+1}^{-\gamma}\left(\alpha e^{x_{2,t+1}} x_{1,t+1}^{\alpha-1} + 1 - \delta\right)\right] \\ y_{1,t} + x_{1,t+1} - e^{x_{2,t}} x_{1,t}^\alpha - (1-\delta)\, x_{1,t} \\ x_{2,t+1} - \rho x_{2,t} \end{array}\right] \tag{6}$$

notice, that here we use fact that $\ln A_t = e^{A_t}$and since we are using partitioned state vector we can rewrite the expression as $\ln A_t = e^{x_{2,t}}$, since $\ln A_t$ is the second vector of $x_t$as showed earlier.

Now we know that the solution to model (1) has this form

$$y_t = g\left(x_t, \sigma\right) \tag{7}$$

$$x_{t+1} = h\left(x_t, \sigma\right) + \eta\sigma\varepsilon_{t+1} \tag{8}$$

where $g\left(\cdot\right)$and $h\left(\cdot\right)$are functions of $x_t$and σ. Matrix η is of order $n_x \times n_\varepsilon$and looks like this

$$\eta = \left[\begin{array}{c} 0 \\ \tilde{\eta} \end{array}\right] \tag{9}$$

We aim to find second-order approximation around non-stochastic steady state where $x_t = \bar{x}$ and $\sigma = 0$. We denote steady state values of vectors using bars, namely $f\left(\bar{y}, \bar{y}, \bar{x}, \bar{x}\right) = 0$. Here we do not use time subscript because it is steady state. It is easy to see that in steady state $\bar{y} = g(\bar{x}, 0)$and $\bar{x} = h\left(\bar{x}, 0\right)$. From equation (2) we can notice that when $\sigma = 0$, then $x_{t+1} = \Lambda x_t^2$which means that we are in steady state.

Now we can substitute the proposed solution (denoted as $y_t$and $x_{t+1}$to the model equation (1) to get

$$F\left(x, \sigma\right) = E_t f\left(g\left(h\left(x_t, \sigma\right) + \eta\sigma\varepsilon_{t+1}, \sigma\right), g\left(x_t, \sigma\right), h\left(x_t, \sigma\right) + \eta\sigma\varepsilon_{t+1}, x_t\right) = 0 \tag{10}$$

Since we know that $F\left(x, \sigma\right)$must be equal to zero for all values of x and σit must be the case that the derivatives of any order of F must be equal to zero too. We can write it as

$$F_{x^k \sigma^j}\left(x, \sigma\right) = 0,\ \forall\, x, \sigma, j, k \tag{11}$$

where $F_{x^k \sigma^j}\left(x, \sigma\right)$denotes the derivative of F with respect to x taken k times and with respect to σ taken j times. Now we will approximate this solution to first and second order.

2.2 First-Order Approximation to Solution

Here we approximate the model to the first order. We would like to approximate the model around the steady state, that is $(x,\sigma) = (\bar{x}, 0)$. Then approximations will have the following form

$$g(x,\sigma) = g(\bar{x},0) + g_x(\bar{x},0)(x-\bar{x}) + g_\sigma(\bar{x},0)\sigma, \tag{12}$$

$$h(x,\sigma) = h(\bar{x},0) + h_x(\bar{x},0)(x-\bar{x}) + h_\sigma(\bar{x},0)\sigma. \tag{13}$$

Since we are looking at the steady state, we have that $g(\bar{x},0) = \bar{y}$ and $h(\bar{x},0) = \bar{x}$. The remainder of unknown coefficients could be solved by looking at

$$F_x(\bar{x},0) = 0 \tag{14}$$

$$F_\sigma(\bar{x},0) = 0 \tag{15}$$

This is because we have the fact that all derivatives of any order must be equal to zero. Now we can write the solution to the system as

$$[F(\bar{x},0)]^i_j = [f_{y'}]^i_\alpha [g_x]^\alpha_\beta [h_x]^\beta_j + [f_y]^i_\alpha [g_x]^\alpha_j + [f_{x'}]^i_\beta [h_x]^\beta_j + [f_x]^i_j = 0 \tag{16}$$

where $i = 1,...,n,\ j,\beta = 1,...n_x,\ \alpha = 1,...n_y$. The notation here is the one adopted from Schmit-Grohe and Uribe (2004) and Collard and Juillard (2001a). Here n_x denotes number of predetermined variables and n_y is the number of forward-looking variables. If take $[f_y]^i_\alpha$ then it is (i,α)element of the derivative of f with respect to y. Furthermore, this derivative is $n \times n_y$ matrix.

The derivatives of f evaluated at steady state are known, that is $(y', y, x', x) = (\bar{y}, \bar{y}, \bar{x}, \bar{x})$. Hence the expression above represents a system of $n \times n_x$quadratic equations with $n \times n_x$unknowns given by the elements of $g_{xx}$and h_{xx}matrices. In the similar manner we can solve for $g_\sigma$and h_σ:

$$[F_\sigma(\bar{x},0)]^i = \begin{array}{c} E_t\{[f_{y'}]^i_\alpha [g_x]^\alpha_\beta [h_\sigma]^\beta + [f_y]^i_\alpha [g_x]^\alpha_\beta [\eta]^\beta_\phi [\varepsilon']^\phi + [f_{y'}]^i_\alpha [g_\sigma]^\alpha \\ + [f_y]^i [g_\sigma]^\alpha + [f_{x'}]^i_\beta [h_\sigma]^\beta + [f_{x'}]^i_\beta [\eta]^\beta_\phi [\varepsilon']^\phi\} \end{array} \tag{17}$$

since ε'in steady state is equal to zero, we can simplify the above expression to

$$[F_\sigma(\bar{x},0)]^i = [f_{y'}]^i_\alpha [g_x]^\alpha_\beta [h_\sigma]^\beta + [f_{y'}]^i_\alpha [g_\sigma]^\alpha + [f_y]^i [g_\sigma]^\alpha + [f_{x'}]^i_\beta [h_\sigma]^\beta = 0. \tag{18}$$

Where $i = 1,...,n,\ \alpha = 1,...n_y,\ \beta = 1,...,n_x$and $\phi = 1,...,n_\varepsilon$. And then from here we can see that the equation above is linear and homogenous in $g_\sigma$and h_σ. Therefore if unique solution exists, it must be the case that

$$h_\sigma = 0, \tag{19}$$

$$g_\sigma = 0. \tag{20}$$

This is the theoretical result from Schmit-Grohe and Uribe (2004). It reads as follows: up to the first-order approximation, we do not need to account for the variance of shocks since it does not change steady state result. Therefore we can conclude that first-order approximation of of the expected values of $x_t$and $y_t$are equal to their non-stochastic steady state values, that is, $\bar{x}$ and $\bar{y}$.

2.3 Second-Order Approximation to Solution

Here we approximate model to the second order. For the full details one should refer to Schmit-Grohe and Uribe (2004) or Klein (2000). Here we focus on the result only omitting the derivation. Taking into account that second-order approximation is taken around the steady state again, that is around the point $(x, \sigma) = (\bar{x}, 0)$. To obtain the secon-order approximation of F we differentiate it twice with respect to x and σ and evaluating them at the steady state $(x, \sigma) = (\bar{x}, 0)$. And these derivatives must be equal to zero. We can use $F_{xx}(\bar{x}, 0)$ to identify $g_{xx}(\bar{x}, 0)$ and $h_{xx}(\bar{x}, 0)$. So we can write the second-order approximation in the following form

$$[F_{xx}(\bar{x},0)]^i_{j,k} = \left[\begin{array}{c} \left([f_{y'y'}]^i_{\alpha,\gamma}[g_x]^\gamma_\delta[h_x]^\delta_k + [f_{y'y}]^i_{\alpha,\gamma}[g_x]^\gamma_k + [f_{y'x'}]^i_{\alpha,\delta}[h_x]^\delta_k + [f_{y'x}]^i_{\alpha,k}\right)[g_x]^\alpha_\beta[h_x]^\beta_j \\ +[f_{y'}]^i_\alpha[g_{xx}]^\alpha_{\beta,\delta}[h_x]^\delta_k[h_x]^\beta_j \\ +[f_{y'}]^i_\alpha[g_x]^\alpha_\beta[h_{xx}]^\beta_{j,k} \\ +\left([f_{yy'}]^i_{\alpha,\gamma}[g_x]^\gamma_\delta[h_x]^\delta_k + [f_{yy}]^i_{\alpha,\gamma}[g_x]^\gamma_k + [f_{yx'}]^i_{\alpha,\delta}[h_x]^\delta_k + [f_{yx}]^i_{\alpha,k}\right)[g_x]^\alpha_j \\ +[f_y]^i_\alpha[g_{xx}]^\alpha_{j,k} \\ +\left([f_{x'y'}]^i_{\beta,\gamma}[g_x]^\gamma_\delta[h_x]^\delta_k + [f_{x'y}]^i_{\beta,\gamma}[g_x]^\gamma_k + [f_{x'x'}]^i_{\beta,\delta}[h_x]^\delta_k + [f_{x'x}]^i_{\beta,k}\right)[h_x]^\beta_j \\ +[f_{x'}]^i_\beta[h_{xx}]^\beta_{j,k} \\ +[f_{xy'}]^i_{j,\gamma}[g_x]^\gamma_\delta[h_x]^\delta_k + [f_{xy}]^i_{j,\gamma}[g_x]^\gamma_k + [f_{xx'}]^i_{j,\delta}[h_x]^\delta_k + [f_{xx}]^i_{j,k} = 0 \end{array}\right] \tag{21}$$

where $i = 1, ..., n,\ j, k, \beta, \delta = 1, .., n_x,\ \alpha, \gamma = 1, ..., n_y$. Because of the fact that we know the derivatives of f and the first derivatives of g and h evaluated at $(y', y, x', x) = (\bar{y}, \bar{y}, \bar{x}, \bar{x})$it follows that the above equation represents a system of $n \times n_x \times n_x$ linear equations in the $n \times n_x \times n_x$unknowns given by the elements of g_{xx} and h_{xx}. In the similar manner we can show the solution for $g_{\sigma\sigma}$ and $h_{\sigma\sigma}$. We just have to solve $F_{\sigma\sigma}(\bar{x}, 0) = 0$. We can write it as follows

$$[F_{\sigma\sigma}(\bar{x},0)]^i = \left[\begin{array}{c} [f_{y'}]^i_\alpha[g_x]^\alpha_\beta[h_{\sigma\sigma}]^\beta + [f_{y'y'}]^i_{\alpha,\gamma}[g_x]^\gamma_\delta[\eta]^\delta_\xi[g_x]^\alpha_\beta[\eta]^\beta_\phi[I]^\phi_\xi \\ +[f_{y'x'}]^i_{\alpha,\delta}[\eta]^\delta_\xi[g_x]^\alpha_\beta[\eta]^\beta_\phi[I]^\phi_\xi \\ +[f_{y'}]^i_\alpha[g_{xx}]^\alpha_{\beta,\delta}[\eta]^\delta_\xi[\eta]^\beta_\phi[I]^\phi_\xi \\ +[f_{y'}]^i_\alpha[g_{\sigma\sigma}]^\alpha \\ +[f_y]^i_\alpha[g_{\sigma\sigma}]^\alpha \\ +[f_{x'}]^i_\beta[h_{\sigma\sigma}]^\beta \\ +[f_{x'y'}]^i_{\beta,\gamma}[g_x]^\gamma_\delta[\eta]^\delta_\xi[\eta]^\beta_\phi[I]^\phi_\xi \\ +[f_{x'x'}]^i_{\beta,\delta}[\eta]^\delta_\xi[\eta]^\beta_\phi[I]^\phi_\xi = 0 \end{array}\right] \tag{22}$$

where $i = 1, .., n,\ \alpha, \gamma = 1, ..., n_y,\ \beta, \delta = 1, ..., n_x$ and $\phi, \xi = 1, .., n_\varepsilon$. This is the system of n linear equations and the n unknowns given by the elements of $g_{\sigma\sigma}$ and $h_{\sigma\sigma}$.

In Schmit-Grohe and Uribe (2004) it is showed that all the cross derivatives of $g_{x\sigma}$and $h_{x\sigma}$ are equal to zero if evaluated at the steady state $(\bar{x}$,0). They show that the system $F_{\sigma x}(\bar{x}, 0) = 0$ is a system of $n \times n_x$ equations in the $n \times n_x$ unknowns given by the elements of two matrices $g_{\sigma x}$ and $h_{\sigma x}$. Since the system is homogenous in unknowns, then the unique solution is given by

$$g_{\sigma x} = 0, \tag{23}$$

$$h_{\sigma x} = 0. \tag{24}$$

This result shows that up to the second-order approximation the coefficients of the policy function on the terms that are linear in the state vector do not depend on the size of the variance of the underlying shocks. For the detailed derivation of this result refer to Schimt-Grohe and Uribe

(2004). Then we can conclude that the solution to the system in equation (1) given by policy functions (7) and (8) could be summarized as follows:

$$g_{\sigma}\left(\bar{x},0\right)=0, \tag{25}$$

$$h_{\sigma}\left(\bar{x},0\right)=0, \tag{26}$$

$$g_{\sigma\delta}\left(\bar{x},0\right)=0, \tag{27}$$

$$h_{x\sigma}\left(\bar{x},0\right)=0. \tag{28}$$

In this paper we develop a code for Python programming language to compute the second-order approximation to the solution provided below. We base our codes on those by Sims (2000), Klein (2000) and Schmit-Grohe and Uribe (2004). We present the code in two ways. Firstly, it could be seen as a bundle of different functions (the approach presented in Klein (2000) for instance). Secondly, it could be seen as one big Python script consisting of those functions since Python allows us to specify more than one function in the script file.

3 Brief Description of Python Routine

In this section we briefly outline the main differences between our Python routine and the code provided by Schmit-Grohe and Uribe (2004). As in the previous versions of this solution algorithm, to use our routine one needs to specify only model equations and steady state solution. The code makes use of standard Python packages, such as Numpy and math (for numerical exercises), Sympy (for symbolic differentiation) and Scipy for matrix algebra. To begin with, we import them and they are the only prerequisite needed to make use of this routine.

```
import sympy as smp
import numpy as np
import math as mt
from scipy import linalg
```

After this we very closely follow the matlab files as provided by Schmit-Grohe and Uribe (2004). However, there is on particular difference in how we compute derivatives. In their original code, all the first and second-order derivatives are computed explicitly, that is, each derivative is written in a separate line. We simplify this approach and compute (and numerically evaluate) both first and second-order derivatives in one loop as showed below.

```
V    = [x,y,xp,yp] # variables
Fx   = [[] for f in F] # 1st derivatives
nFx  = [[] for f in F] # num of 1st order derivatives
Fxx  = [[[[] for w in V] for v in V] for f in F] # 2nd derivatives
nFxx = [[[[] for w in V] for v in V] for f in F] # num of 2nd order
    derivatives
fxx  = [] # intermediary matrix for calculation
nfxx = [] # intermediary matrix for calculation
# Compute Fx & Fxx and nFx & nFxx
for i,f in enumerate(F):
        f = smp.Matrix([f])
                for j,v in enumerate(V):
                        v = smp.Matrix([v])
                        fx = f.jacobian(v)
                        nfx = (fx.subs(Parameters_dic)).subs(
                            SS_Variables_dic)
                        Fx[i].append(fx)
                        nFx[i].append(nfx)
                                for p,v in enumerate(V):
```

```
                                                  v = smp.Matrix([v])
                                                  fxx = fx.jacobian(v)
                                                  Fxx[i][j][p].append(fxx)
                                                  nfxx = (fxx.subs(Parameters_dic)).
                                                      subs(SS_Variables_dic)
                                                  nFxx[i][j][p].append(nfxx)
```

They key advantage of writing code like this is the order of first and second-order derivatives matrices. Firstly, consider matrix of the second derivatives `Fxx`. It has three dimensions and is ordered in such a way, that to access the second derivative of the first model equation with respect to say first variable one just needs to write `Fxx[0][0]0[0]`. This expression points to the first function (that's why there is first zero), and to the second derivative with respect x variable in `V` array (first line of the code). Note, that we are using zero since in Python order starts at zero.

This approach let's us significantly simplify the code, saves the space and add more clarity. However, it ends up saving 4 big matrices, that is first and second-order both analytical and numerical derivatives. From the computational point of view, maybe one would like to split these matrices into a few smaller ones to save time. At the moment we leave this for further development.

It is noteworthy, that to our knowledge Python at the moment of writing this paper does not have any reliable routine for computing Shur's decomposition. Therefore we had made use of Sims (2000) matlab files, namely `qzswitch.m` and `qzdiv.m.` We have followed them closely and translated to Python. They are used as a substitute for matlab function `ordqz` which reorders Schur's decomposition matrices.

To keep the consistency we have also provided function defined in Python in exactly the same manner as it is done in matlab. However, we believe that is is more efficient to use code without defining separate functions, but for comparison purposes we have also provided functions.

4 Examples

In this section we apply our routine to the same examples as in Schmit-Grohe and Uribe (2004) paper. Namely neoclassical growth model, a two-country neoclassical model with complete asset markets by Kim and Kim (2003) and finally an asset pricing model as in Burnside (1998) and Collard and Juillard (2001b). In here we will consider only the first model (neoclassical growth model) as an example on how to use the code. However, we admit that some routines might not be very efficient and user friendly. We leave these improvements for further research.

First lines of codes are used to declare parameters and variables as sympy objects (symbolic variables).

```
#Define Parameters
SIG, DELTA, ALFA, BETTA, RHO = smp.symbols('SIG, DELTA, ALFA, BETTA, RHO')
# Define Variables
c, cp, k, kp, a, ap = smp.symbols('c, cp, k, kp, a, ap')
```

Then in the following section of the code one must specify equations that will be used for computations.

```
f1 = c + kp - (1-DELTA) * k -a * k ** ALFA
f2 = c**(-SIG) - BETTA * cp ** (-SIG) * (ap * ALFA * kp** (ALFA-1) +1 -DELTA)
f3 = smp.log(ap) - RHO * smp.log(a)
# Create function f
F = smp.Matrix([f1, f2, f3])
```

and then we put them in the matrix. In the same way we define predetermined variables (as in the original code of Schmit-Grohe and Uribe (2004)).

```
x  = smp.Matrix([k, a])
y  = smp.Matrix([c])
```

```
xp = smp.Matrix([kp, ap])
yp = smp.Matrix([cp])
```

Notice, that here we use sympy matrix. The reason for it is because we want to declare variables as objects and not as numbers.

Lastly, we may substitute logarithms in the model using

```
F = F.subs([(c,smp.exp(c)),...])
```

Steady state solution is straightforward, we just copy and paste into the code. Notice, that we use capital letters for steady state values. We also take numerical logs of the steady state values.

The last thing we do (and this most probably is not the most efficient way to handle it) is to create two dictionaries in Python: one for parameter values and other for steady state values of variables. The reason why we need this is the numerical evaluation. We use the dictionaries to evaluate first and second order derivatives.

```
# Parameters Dictionary
Parameters        = ['BETTA', 'DELTA', 'ALFA', 'RHO', 'SIG']
Param_Values      = [0.95, 1, 0.3, 0, 2]
Parameters_dic    = {} for i in range(len(Parameters)):
        Parameters_dic[Parameters[i]] = Param_Values[i]
```

The rest of the code is just an interpretation of the original codes by Schmit-Grohe and Uribe (2004). Also we have used codes written by Chris Sims as mentioned before.

The procedures are the same for any other model. We provide codes for all three examples.

5 Conclusion

We have translated Schmit-Grohe and Uribe (2004) code to Python. Our proposed routine might not be the fastest or most elegant, but it is an illustration how this code could be implemented using different programming language. Any further suggestions and improvements are more than welcome.

References

[1] Schmit-Grohe, S., Uribe, M., 2004, 'Solving Dynamic General Equilibrium Models Using a second-order Approximation to the Policy Function', *Journal of Economic Dynamics and Control*, vol. 28, pp. 755-775.

[2] Klein, P., 2000, 'Using the Generalized Schur Form to Solve a Multivariate Linear Rational Expectations Model', *Journal of Economic Dynamics and Control*, vol. 24, pp. 1405-1423.

[3] Kim, J., Kim, S., Sims, C., A., 2008, 'Calculating and Using Second-Order Accurate Solutions of Discrete Time Dynamic Equilibrium Models', *Journal of Economic Dynamics and Control,* vol. 32, pp. 3397-3414.

[4] Sims, C., A., 2000, 'Second Order Accurate Solution to Discrete Time Dynamic Equilibrium Models', Manuscript, Princeton University, Princeton, December.

[5] Collard, F., Juillard, M., 2001a, 'Perturbation Methods for Rational Expectations Models', Manuscript, CEPREMAP, Paris, February.

[6] Burnside, D., 1998, 'Solving Asset Pricing Models with Gaussian Shocks', *Journal of Economic Dynamics and Control*, vol., 22, pp. 329-340.

[7] Kim, J., Kim, ., H., 2003, 'Spurious Welfare Reversals in International Business Cycle Models', *Journal of International Economics*, vol. 60, pp. 471-500.

[8] Collard, F., Juillard, M., 2001b, 'Accuracy of Stochastic Perturbation Methods: the Case of Asset Pricing Models', *Journal of Economic Dynamics and Control,* vol. 25, pp. 979-999.