Back in Time. Fast. Improved Time Iterations.

Pablo Winant, Bank of England

June 29, 2017

1 Introduction

We present a fast iterative algorithm to find an approximate solution for a wide class of rational expectations models characterized by first order conditions. This method converges at quadratic rate and its practical implementation is several orders of magnitudes faster than existing projection methods, while producing the same solution. In particular its accuracy is the same, only limited by the space of basis functions used to interpolate the solution.

This paper is motivated by the recent interest in the economic profession to investigate the robustness of nonlinear medium scale rational equilibrium models, when the economy can be drawn away from the steady-state or when there are important nonlinearities jisuch as an occasionally binding constraint or a zero lower bound (for instance [1], [2], [3]).

In general, these models do not derive from the problem of one single optimizing agent, hence methods based on Bellman equations, like value iteration or the faster policy iterations (see [4]) are inadequate. Instead we intend to solve any model which can be characterized by a set of transition and equilibrium equations. The latter include Euler equations, asset prices, or other nonlinear relations.[1]

[1]The formulation of the model is similar to [5]. Let s_t be a vector of states and ϵ_t a vector of i.i.d. shocks. For any admissible vector of controls x_t the transition function g is:

$$s_t = g\left(s_{t-1}, x_{t-1}, \epsilon_t\right)$$

Equilibrium conditions are given by a function f such that:

$$E_{\epsilon_{t+1}}\left[f(s_t, x_t, \epsilon_{t+1}, s_{t+1}, x_{t+1})\right] = 0$$

which must be satisfied for any value of the states s_t variables. This determines optimal policy $x_t = \varphi(s_t)$.

This generic class of model, can be solved by the time iterations algorithm ([6], [7]), where a finite horizon model is solved, and the number of periods increased until convergence, or, in an equivalent formulation, by going back in time from the last period.

The contractivity of this back in time operator is a deep property of economic models. It ensures local unicity and stability of rational expectations equilibria. When the model admits a first order approximation around a steady-state, stability is locally implied by the Blanchard-Kahn conditions ([8]).

We use this property to propose a new, faster, algorithm. It reformulates the model as a big nonlinear system of equations, in the decision variables. The back-in-time stability is used to solve the Jacobian of this system, by exploiting its particular structure.

Another, and perhaps more informative, description of the algorithm would be to describe it as a way to speed up time-iterations by alternating between marginal improvements of the decision rules and successive backward iterations (which compute the cumulative errors of following a policy forever). Essentially, our method relate to Euler iterations algorithms in the same way than Howard improvement steps relate to value function iteration.

Contrary to other recent solution methods, we do not need additional assumptions such as the existence of a fixed-point representation ([9], [10]), or the predetermination of endogenous states ([11]).

Our method does not replace and does not preclude the use of dimensionality reduction algorithms (see [12]). They come with a tradeoff between dimensionality and achievable precision. By increasing computation speed and reducing memory footprint, our algorithm improves the terms of this tradeoff.

2 A fast solution algorithm

2.1 Problem setup

Let $\mathbf{s} \in \mathbb{R}^N \times \mathbb{R}^{n_d}$ be a discretized state-space, where each line contains a different vector of state.[2] Similarly, let $\mathbf{x} \in \mathbb{R}^N \times \mathbb{R}^{n_x}$ be the corresponding list of controls. The decision rule is then jointly characterized by $\mathbf{x}$ and an interpolation method $\mathcal{I}(u, \mathbf{x})$ to generalize the controls on any state u. Let (w_j, ϵ_j) be the nodes and the weights approximating an exogenous i.i.d. process.

Let $\tilde{\mathbf{x}}$ denote the set of controls representing the decision rule tomorrow. The model we try to solve is defined by the optimization errors $F(\mathbf{x}, \tilde{\mathbf{x}})$ such that:

$$\begin{aligned} \mathbf{r} &= F(\mathbf{x}, \tilde{\mathbf{x}}) \\ &= \sum_j w_j f\left(\mathbf{s}, \mathbf{x}, \mathbf{s_j}, \mathbf{x_j}\right) \end{aligned} \tag{1}$$

with

$$\begin{aligned} \mathbf{s_j} &= g(\mathbf{s}, \mathbf{x}, \epsilon_j) \\ \mathbf{x_j} &= \mathcal{I}(\mathbf{s_j}, \mathbf{s}, \mathbf{x}) \end{aligned} \tag{2}$$

A time-invariant solution $\overline{\mathbf{x}}$ of the problem satisfies:

$$G\left(\overline{\mathbf{x}}\right) = F\left(\overline{\mathbf{x}}, \overline{\mathbf{x}}\right) = 0$$

2.2 Time iterations

Let's define the time-iteration operator as the function $\mathcal{T}$ which computes $\mathbf{x}$ for any initial $\tilde{\mathbf{x}}$. Close to the solution, $\mathcal{T}$ is a (local) contraction of modulus $\lambda < 1$. This motivates the time iteration algorithm:

1. Choose initial $\mathbf{x_0}$ and termination criterion $\epsilon > 0$
2. For $n > 0$, given an initial guess $\mathbf{x_n}$,
   - compute $\mathbf{x_{n+1}} = \mathcal{T}(\mathbf{x_n})$ by solving equation $F(\mathbf{x}, \tilde{\mathbf{x}})$ in $\mathbf{x}$.
   - If $\|\mathbf{x_{n+1}} - \mathbf{x_n}\| < \epsilon$, return $\mathbf{x_n}$

Time iterations typically converge at a geometric rate λ for which a lower bound can be estimated as the limit ratio of successive approximation errors: $\limsup_n \frac{\|\mathbf{x_{n+1}} - \mathbf{x_n}\|}{\|\mathbf{x_n} - \mathbf{x_{n-1}}\|}$.

[2] For practical applications we consider both a cartesian grid and smolyak product of chebychev nodes.

2.3 Newton-Raphson method

Another approach, also known as the bruteforce approach, consists in solving the system $G(\mathbf{x}) = 0$ as a big nonlinear system in $\mathbf{x}$. A simple solution algorithm can be summarized as follows:

1. Choose initial $\mathbf{x_0}$ and termination criterion $\eta > 0$
2. For $n > 0$, given an initial guess $\mathbf{x_n}$:
   - Compute $\mathbf{r_n} = G(\mathbf{x_n})$
   - If $\|\mathbf{r_n}\| < \eta$ return $\mathbf{x_n}$
   - Compute $\mathrm{d}\,\mathbf{x_{n+1}} = -\left(G'(\mathbf{x_n})\right)^{-1}.\mathbf{r_n}$
   - Set $\mathbf{x_{n+1}} = \mathbf{x_n} + \mathrm{d}\,\mathbf{x_{n+1}}$

   When the Newton descent converges it does so at quadratic speed and takes in general less iterations than the time-iteration algorithm.[3]

   For any $\mathbf{x_n}$, we can differentiate (1) to get:

$$G'(\mathbf{x_n}).\,\mathrm{d}\,\mathbf{x} = \underbrace{F'_{\mathbf{x}}(\mathbf{x_n}, \mathbf{x_n}).\,\mathrm{d}\,\mathbf{x}}_{A_n.\,\mathrm{d}\,\mathbf{x}} + \underbrace{F'_{\tilde{\mathbf{x}}}(\mathbf{x_n}, \mathbf{x_n}).\,\mathrm{d}\,\mathbf{x}}_{-B_n.\,\mathrm{d}\,\mathbf{x}} \tag{3}$$

   In this expression, A_n and B_n are linear operators, whose expression we will explicit later.[4] The Newton step can be obtained as:

$$\mathrm{d}\,\mathbf{x}_{n+1} = -\left(A_n - B_n\right)^{-1} \mathrm{d}\,\mathbf{r}_n \tag{4}$$

   To form jacobian matrix J_n representing $A_n - B_n$ and to invert it, is a very costly operation when the number of gridpoints is high, even taking the sparsity of the jacobian into account. Figure 1 illustrates why: A_n has a simple block-diagonal structure, but B_n is more complicated. It might even have too many non-zero elements to fit into memory.

[3] In practice, a line search is made to find a scalar μ such that $\|G(\mathbf{x_n} + \mu\,\mathrm{d}\,\mathbf{x_{n+1}})\|$ is defined, optimal or improving.

[4] When M is a linear operator, we denote by $M.\mathbf{v} = M(v)$ its application to vector $\mathbf{v}$. Note that a linear operation on matrices does not need to be a matrix product in the conventional sense. For instance if M_1, M_2 are square matrices in $\mathbb{R}^n \times \mathbb{R}^n$ the application $L : \mathbb{R}^n \times \mathbb{R}^n \ni X \to M_1 X M_2$ is a linear operation since $L.(X + \lambda Y) = L(X) + \lambda L(Y)$. L is thus an element of $(\mathbb{R}^n \times \mathbb{R}^n) \times (\mathbb{R}^n \times \mathbb{R}^n)$ and a matrix representation would have a $n^2 \times n^2$ size.

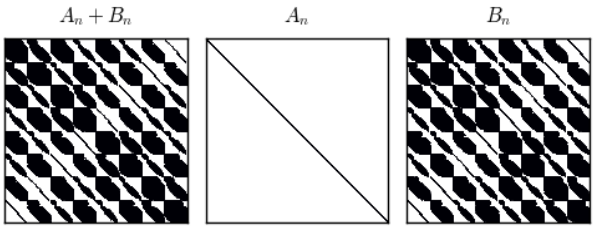

Figure 1: Structure of the jacobian for the two countries autarky model.
$A_n - B_n$ is a 10800×10800 matrix (2% nonzero elements). A sparse matrix algorithm solves for $(A_n - sB_n)Y = X$ in 40 seconds, versus 263 milliseconds for the algorithm described in the text (20 minutes for the doubling algorithm).

2.4 Time Iterations Improved

The central idea investigated in this paper, consists in replacing 4 by the Neumann series:

$$\left(I + {A_n}^{-1}B_n + \left(A_n^{-1}B_n\right)^2 + \cdots\right) {A_n}^{-1}\mathbf{r}_n \quad (5)$$

When the sum in (5) converges absolutely, it necessarily converges to the Newton-Raphson step given in 4.

To evaluate the sum, one can set a tolerance criterium $\eta > 0$ and then compute the terms π_k recursively defined by $\pi_{\mathbf{0}} = {A_n}^{-1}\mathbf{r}_n$ and $\pi_{\mathbf{k}} = A_n^{-1}B_n.\pi_{\mathbf{k-1}}$ until $\|\pi_{\mathbf{k}}\| < \eta$.

An important remark at this stage is that one never needs to form a matrix representation of $A_n^{-1}B_n$ to compute the successive terms π_n. Only requirement is an available precedure to compute $A_n^{-1}B_n.\mathbf{v}$ for any vector $\mathbf{v}$[5]

[5]If a (sparse) matrix representation $M_n = A_n^{-1}B_n$ is available, it is tempting to use a doubling algorithm to replace infinite sum (5) by infinite product

$$\sum_i (M_n)^i = (I + M_n)\prod_i \left(I + (M_n)^{2i}\right)$$

where the computation of an additional on the right hand side amounts to doubling the number of terms in the sum from the left hand side. Each new factor, requires two additional matrix multiplications. When the size of the matrices goes up, these operations become increasingly costly w.r.t. the vector multiplications involved in the infinite sum (5). We find that this approach doesn't provide any gain in our aplications.

2.5 Spectral radius of the time iteration operator

The convergence of the infinite sum in (5) is linked to the contractivity of time-iteration operator $\mathcal{T}$. Let's consider the derivative, of the time iteration step, evaluated at the solution $\overline{\mathbf{x}}$:

$$\mathcal{T}'(\overline{\mathbf{x}}).\mathrm{d}\,\mathbf{x} = -\left(\underbrace{F'_{\mathbf{x}}(\overline{\mathbf{x}},\overline{\mathbf{x}}).\mathrm{d}\,\mathbf{x}}_{A}\right)^{-1}\left(\underbrace{F'_{\tilde{\mathbf{x}}}(\overline{\mathbf{x}},\overline{\mathbf{x}})}_{B}\right)\mathrm{d}\,\mathbf{x}$$

When the model is well specified, and the approximation method adequate, the norm of linear operator $\mathcal{T}'$ is $\lambda < 1$. As a result, if $\mathbf{x_n}$ is close enough to $\overline{\mathbf{x}}$, by continuity the norm of operator $A_n^{-1}B_n$ is smaller than 1 and the sum converges.

As in the time-iteration case, the computation of sum (5) also provides a natural lower bound on the spectral norm λ as $\max_i \frac{\|\pi_{i+1}\|}{\|\pi_i\|}$. One can go further and compute the actual spectral radius by starting from a *random* initial vector $\epsilon_{\mathbf{0}}$ and looking at the successive terms $\epsilon_{\mathbf{k}} = \left(A^{-1}B\right).\epsilon_{\mathbf{k-1}}$. Then with probability one, i.e. unless $\epsilon_{\mathbf{0}}$ is very special, the ratio $\frac{\|\epsilon_{\mathbf{k}}\|}{\|\epsilon_{\mathbf{k-1}}\|}$ converges to actual radius λ.

2.6 Blanchard-Kahn conditions

If the model admits a deterministic steady-state $\overline{s}, \overline{x}$ and Blanchard-Kahn conditions are met, regular perturbation techniques provide a good initial guess for the nonlinear solution.

As a byproduct they also produce several eigenvalues $\lambda_1 < \cdots < \lambda_k < 1 < \lambda_{k+1} < \cdots < \lambda_{2n}$, where $(\lambda_i)_{i \le k}$ are the eigenvalues "selected" by

the unique convergent solution. We show in appendix B that $\frac{\lambda_k}{\lambda_{k+1}}$ is an upper bound for the norm of the time-iteration operator in the linearized model. When significantly smaller than one it implies a presumption that the model can be solved nonlinearly in a neighborhood of the steady-state.

2.7 Complementarity constraints

Suppose there are bounds $\mathbf{a}$ and $\mathbf{b}$ such that the model to solve is:

$$F(\mathbf{x}, \tilde{\mathbf{x}}) \perp \mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$$

which, by convention, is equivalent to $\overline{F}(\mathbf{x}, \tilde{\mathbf{x}}) = 0$ with

$$\overline{F}(\mathbf{x}, \tilde{\mathbf{x}}) = \min(\max(F(\mathbf{x}, \tilde{\mathbf{x}}), \mathbf{x} - \mathbf{a}), b - \mathbf{x})$$

or to the smooth variant

$$\overline{\overline{F}}(\mathbf{x}, \tilde{\mathbf{x}}) = \varphi^-(\varphi^+(F(\mathbf{x}, \tilde{\mathbf{x}}), \mathbf{x} - \mathbf{a}), b - \mathbf{x})$$

where φ^+ and φ^- are the Fisher functions.

If the model with occasionnally binding constraints, has stable backward iterations, the logic exposed before can be applied to $\overline{F}$ or $\overline{\overline{F}}$. In particular, the values and derivatives of the redefined system have the same sparsity structure as the original one.

3 Practical implementation

Differentiating (1), we get[6]:

$$\mathrm{d}\,\mathbf{r} = \overbrace{\sum_j w_j \left(f'_{\mathbf{x},\epsilon_j} + f'_{\tilde{\mathbf{s}},\epsilon_j} g'_{\mathbf{x},\epsilon_j} + f'_{\tilde{\mathbf{x}},\epsilon_j} I'_{\mathbf{s}}(\mathbf{s_j}, \tilde{\mathbf{x}}) g'_{\mathbf{x},\epsilon_j} \right)}^{A} . \mathrm{d}\,\mathbf{x} + \underbrace{\sum_j w_j f'_{\tilde{\mathbf{x}},\epsilon_j} I'_{\tilde{\mathbf{x}}}(\mathbf{s_j}, \tilde{\mathbf{x}}) . d\tilde{\mathbf{x}}}_{B . d\tilde{\mathbf{x}}} \tag{6}$$

[6]To shorten notations, for $\mathbf{v} = \mathbf{s}, \mathbf{x}, \tilde{\mathbf{s}}, \tilde{\mathbf{x}}$ we abbreviate $g'_{\mathbf{v}}(\mathbf{s}, \mathbf{x}, \epsilon_j)$ and $f'_{\mathbf{v}}(\mathbf{s}, \mathbf{x}, \mathbf{s_j}, \mathbf{x_j})$ by $g'_{\mathbf{v},\epsilon_j}$ and $f'_{\mathbf{v},\epsilon_j}$ respectively.

Let us denote by $D_{p,q} \subseteq \mathbb{R}^N \times \mathbb{R}^p \times \mathbb{R}^N \times \mathbb{R}^q$ the space of four dimensional tensors such that, for any $M \in D_{p,q}$ we have $i \neq j \implies M_{n,i,m,j}$. Elements of $D_{p,q}$ are naturally represented by a three dimensional tensor in $\mathbb{R}^N \times \mathbb{R}^p \times \mathbb{R}^q$ or by a block diagonal matrix of $\mathbb{R}^{np \times nq}$ where each block has size $p \times q$. All partial derivatives of g and f appearing in (6) as well as $I'_{\mathbf{s}}$ are elements of $D_{n_x,d}$ or D_{n_x,n_x}.

Operations (addition, multiplication, inversion), among elements of $D_{p,q}$ can be implemented efficiently. In particular, operator A is an element of D_{n_x,n_x} and can be easily inverted.

3.1 Precomputing $A^{-1}B$

The computation of $A^{-1}B . \mathrm{d}\,\pi$ involves the computation of

$$A^{-1} \sum_j w_j f'_{\tilde{\mathbf{x}},\epsilon_j} \mathcal{I}_{\mathbf{x}}(\mathbf{s_j}, \mathbf{x}) . \mathrm{d}\,\pi$$

To reduce the number of times the premultiplication by A^{-1} is applied, we can precompute the terms:

$$\mathbf{D_j} = w_j A^{-1} f'_{\mathbf{x},\epsilon_j}$$

These terms don't depend on $\mathrm{d}\,\pi$ and need to be computed only once for each Newton step. We are then left with the evaluation of:

$$\sum_j \mathbf{D_j} \mathcal{I}_{\mathbf{x}}(\mathbf{s_j}, \mathbf{x}) . \mathrm{d}\,\pi$$

3.2 Computing $\mathcal{I}'_{\mathbf{x}} . d\pi$

If $\mathcal{I}()$ is an interpolator on a linear basis, it is linear as a function of the data to be fitted, which implies:

$$\mathcal{I}'_{\mathbf{x}}(\mathbf{s_j}, \mathbf{x}) . \mathrm{d}\,\pi = \mathcal{I}(\mathbf{s_j}, \mathrm{d}\,\pi)$$

	Jacobian size	Time Iteration	Fast Newton	Fast Newton (GPU)
Model A ($9x15x15$ states)	10800^2	148s	9s	2s
Model A ($9x50x50$ states)	120000^2	1440s	52s	7s
Model B ($\mu = 1$)	300^2	4.03s	0.61s	-
Model B ($\mu = 2$)	3756^2	113.40s	2.67s	-
Model B ($\mu = 3$)	31788^2	1181.41s	25.28s	-
Model B ($\mu = 4$)	207180^2	15132.38s	612.61s	-

Table 1: Timings.

	Time Iteration				Fast Newton			
	Total	Model	Prefilter	Interpolation	Total	Model	Prefilter	Interpolation
Fraction	100%	66%	0.3%	7%	100%	46%	15%	22%
Calls		966	1632	6762		10	5560	7904

Table 2: Decomposition for model A.

The computation of $\mathcal{I}(\mathbf{s_j}, \mathrm{d}\,\pi)$ can be done using the same operations than the one used to approximate the residuals tomorrow[7].

4 Timings

The two models used to benchmark our algorithm are described in appendix A. Model A is a simple two countries economy taken from [3], with 3 continuous states, interpolated using cubic splines. Model B is a mock 12 states model made of 6 independent RBC countries. The solution of the latter is approximated using a smolyak product of chebychev polynomials, with parameters $\mu = 1, 2, 3, 4$.

Using the method described above, we implement a simple safeguarded Newton optimization[8], with tolerance $\epsilon = 10^{-8}$. After mesuring spectrum radius λ as described in section 2.5 we set termination criterium $\eta = (1 - \lambda)\epsilon$ for time-iterations.

Table 1 shows very encouraging time gains: our algorithm converges at least 15 times faster than regular time iterations for model A and 6 times faster for model B[9][10].

Table 2 shows where the gains come from: the number of model evaluations is drastically reduced (from 966 to 10) resulting in a bigger frac-

[7] The linearity of $\mathcal{I}$ is apparent in the formula:

$$\mathcal{I}(\mathbf{s_j}, \mathrm{d}\,\pi) = \Phi(\mathbf{s_j}) \underbrace{\mathcal{B}^{-1}\,\mathrm{d}\,\pi}_{d\mathbf{c}}$$

where $\mathcal{B}$ is the filtering matrix, and $\Phi()$ the function which constructs the value of basis points. Whether it is wise to use it or not depends on the approximation method:

- For smolyak polynomials, $\mathcal{B}$ is dense and there is not other way than to invert it. Hence we precompute a LU factorization in order to speed up the computation of $\mathcal{B}^{-1}\,\mathrm{d}\,\mathbf{r}$. Also the basis functions are non trivial to compute, so that it pays to precompute the the basis matrix $\Phi(\mathbf{s_j})$ only once for each Newton step.
- For cubic splines, we use filtering formulas, to find $\mathcal{B}^{-1}\,\mathrm{d}\,\mathbf{r}$ efficiently (see [13]). Also, precomputing the number of basis functions doesn't seem to offset the cost of additional memory access. Instead we use the blending formulas to compute at once $\Phi(\mathbf{s_j}).\,\mathrm{d}\,\mathbf{c}$ from the coefficients $\mathbf{c}$ only.

[8] At each step, we compute $\mathrm{d}\,\mathbf{x_n}$ and $\mathbf{x_{n+1}} = \mathbf{x} + 2^{-k}\,\mathrm{d}\,\mathbf{x_n}$ with k the smallest integer such that $\|G(\mathbf{x}_{n+1})\|$ is both, defined and smaller than $\|G(\mathbf{x_n})\|$.

[9] CPU: Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz with 16 GB memory. GPU: GTX Titan on the same machine.

[10] The comparison is probably unfair but Jesus Villaverde and Oren Levintal solve a 12 states model using smolyak polynomials in 55 ($\mu = 2$) seconds and 7741.6 seconds ($\mu = 3$). Their Jacobian is slightly bigger than ours: 39735^2 elements.

tion of total time spent performing the prefiltering and interpolation steps. Remaining operations are massively parallel and good candidates for further optimizations[11].

5 Conclusion

The life of a computational economist is plagued by a very annoying problem: a realistic nonlinear rational expectations model takes a long time to solve. Theory is of no comfort as soon as one is enstranged from the comfortable world of representative agent and complete markets models. There is often no proof of existence or global uniqueness, let alone a formal proof that iterative algorithms will converge for sure.

In this paper we try to alleviate this burden in two different ways. We show how to reduce dramatically the time needed to solve such a model, by reformulating it as one nonlinear system of equations and by exploiting the contractivity of backward iterations to solve the jacobian at each step. This contractivity is a desirable feature of any rational expectations model and when it cannot be established theoretically, our method suggest a way to test it numerically.

[11] For instance, our GPU code accelerates prefiltering 200 times, model evaluation 8 times, and interpolation 5 times.

References

[1] O. Levintal, "Solution Methods for Models with Rare Disasters," 2016.

[2] J. Fernández-Villaverde, G. Gordon, P. Guerrón-Quintana, and J. F. Rubio-Ramírez, "Nonlinear adventures at the zero lower bound," *Journal of Economic Dynamics and Control*, vol. 57, pp. 182–204, 2015.

[3] N. Coeurdacier, H. Rey, and P. Winant, "Financial Integration and Growth in a Risky World," Working Paper 21817, National Bureau of Economic Research, dec 2015.

[4] J. Rust, "Numerical dynamic programming in economics," *Handbook of Computational Economic*, vol. 1, no. November, pp. 614–722, 1996.

[5] M. J. Miranda and P. L. Fackler, *Applied Computational Economics and Finance*. Cambridge, MA, USA: MIT Press, 2002.

[6] A. Deaton and G. Laroque, "On the Behaviour of Commodity Prices," *The Review of Economic Studies*, vol. 59, no. 1, p. 1, 1992.

[7] W. J. Coleman, "Solving the Stochastic Growth Model by Policy-Function Iteration," *Journal of Business & Economic Statistics*, vol. 8, no. 1, pp. 27–29, 1990.

[8] S. Cho and A. Moreno, "The forward method as a solution refinement in rational expectations models," *Journal of Economic Dynamics and Control*, vol. 35, no. 3, pp. 257–272, 2011.

[9] W. J. den Haan and A. Marcet, "Solving the Stochastic Growth Model by Parameterizing Expectations," *Journal of Business & Economic Statistics*, vol. 8, no. 1, pp. 31–34, 1990.

[10] C. D. Carroll, "The method of endogenous gridpoints for solving dynamic stochastic optimization problems." 2006.

[11] K. L. Judd, L. Maliar, and S. Maliar, "How to Solve Dynamic Stochastic Compputing Expectations Just Once," *NBER Working Paper*, 2011.

[12] L. Maliar and S. Maliar, *Numerical methods for large scale dynamic economic models*, vol. 3. 2014.

[13] D. Ruijters and P. Thvenaz, "Gpu prefilter for accurate cubic b-spline interpolation," *The Computer Journal*, vol. 55, no. 1, p. 15, 2010.

A Models

B Relation with perturbation theory

There is a connection between the stability of backward iterations and the behaviour of the model in the neigborhood of its steady-state. In particular, we can show that a model satisfying the Blanchard-Kahn conditiosn locally, will also be stable via Backward iterations. We can also use a perturbation solution to compute an estimate for the modulus of the time-iteration operator. This are important informations, when one intends to apply global solution methods (as described in section 2) to solve medium size DSGE models.

For the sake of clarity, assume that there are not shocks. Close to the steady-state $(\overline{s}, \overline{x})$, one can use the perturbation theory to approximate the decision rule today (φ) and tomorrow $(\tilde{\varphi})$ as

$$\varphi(x) = \overline{x} + X_1(s - \overline{s}) + o(s - \overline{s})$$
$$\tilde{\varphi}(x) = \overline{x} + \tilde{X_1}(s - \overline{s}) + o(s - \overline{s})$$

B.1 Stability of backward inductions

At first order, the decisions must satisfy the Ricatti equation:

$$K_1 + K_2 X + K_3 \tilde{X} \left(K_4 + K_5 X\right)$$

Back in time iterations, consist in computing the successive terms $X_{n+1} = T(X_n)$:

$$X_{n+1} = (K_2 + K_3 X_n K_5)^{-1}(K_1 + K_3 X_n K_4)$$

This recursion does not converge in general from any initial point. If it converges to a solution X, we can study the stability of this solution by differentiating B.1 to get:

$$\underbrace{(K_2 + K_3 X K_5)\,\mathrm{d}\,X_{n+1}}_{A\mathrm{d}X_{n+1}} + \underbrace{K_3\,\mathrm{d}\,X_n \overbrace{(K_4 + K_5 X)}^{P}}_{-B\mathrm{d}X_n}$$

Based on these simple calculations, we can enumerate several conditions, which can be met selectively by a given solution:

(a) Dynamic stability: all eigenvalues of P are smaller than one.[12]

(b) Backward iterations are (locally) well defined: matrix A is invertible.

(c) Backward iterations are stable: operator $u \to A^{-1}B.u$ has all eigenvalues smaller than one.

All this conditions can be tested easily for a given solution of the linear problem. In particular, the spectral radius of $A^{-1}B.u$ provides an indication for the contractivity of the global time-iteration algorithm.

B.2 Blanchard-Kahn conditions and MOD solution

A time invariant solution to (B.1) satisfies the *algebraic* Ricatti equation if:

$$K_1 + K_2 X + K_3 X \left(K_4 + K_5 X\right)$$

All the solutions to this equation can be characterized by choosing k eigenvalues among

$$(\lambda_1 \leq \cdots \leq \lambda_{2n})$$

where $(\lambda_i)_{1 \leq i \leq 2n}$ are the eigenvalues to an associated matrix pencil. Let's denote by $\Lambda_P = (\lambda_{i_1}, \cdots, \lambda_{i_k})$ one choice of k eigenvalues, and denote by $\Lambda_Q = (\lambda_{j_1}, \lambda_{j_{2n-k}})$ the remaing ones. Each choice of eigenvalues $\Lambda_P = (\lambda_{i_1}, \cdots, \lambda_{i_k})$ corresponds to a different control X. The theory says that Λ_P are the eigenvalues of the transition matrix P while Λ_Q contains the inverse of the eigenvalues in $A^{-1}K_3$.

As a result the modulus of operator T' satisfies:

$$|||T'||| \leq |||A^{-1}K_3||| \times |||P||| = (\min_i \lambda_i)(\min_j \lambda_j)$$

In general, the inequality is not an equality and there can be several solutions satisfying (2-3) or even (1-2-3).

However, when the Blanchard-Kahn conditions are met ($\lambda_k < 1 < \lambda_{k+1}$), it is easy to check that the MOD-solution (which selects $\lambda_1, \cdots, \lambda_k$) always meets the conditions (1-2-3) and is the unique one to do so.

If $\lambda_k < \lambda_{k+1} < 1$, the MOD solution still satisfies (1-2-3), but there are exactly k other solutions satisfying (1-2) (by taking λ_{k+1} instead of λ_l for $\mu = [1, k]$).

These other solutions can be stable by time-iteration (3), or not. If $\lambda_k > 1$, the MOD solution is still well defined by time-iteration (2-3), but it is not stable. There can be other solutions which share the same property.

[12] Given the decision rule X the law of motion of the states is $s_t - \overline{s} = (g_s + g_x X)\,(s_{t-1} - \overline{s}) + o\,(s_{t-1} - \overline{s})$