CHAPTER SEVEN

Numerical Methods for Large-Scale Dynamic Economic Models

Lilia Maliar and Serguei Maliar[*]
T24, Hoover Institution, Stanford, CA, USA

Contents

[*] Chapter prepared for Handbook of Computational Economics, Volume 3, edited by Karl Schmedders and Kenneth L. Judd.

Handbook of Computational Economics, Volume 3
ISSN 1574-0021, http://dx.doi.org/10.1016/B978-0-444-52980-0.00007-4

1. INTRODUCTION

The economic literature is moving to richer and more complex dynamic models. Heterogeneous-agents models may have a large number of agents that differ in one or several dimensions, and models of firm behavior may have a large number of heterogeneous firms and different production sectors.[1] Asset-pricing models may have a large number of assets; life-cycle models have at least as many state variables as the number of periods (years); and international trade models may have state variables of both domestic and foreign countries.[2] New Keynesian models may have a large number of state variables and kinks in decision functions due to a zero lower bound on nominal interest rates.[3] Introducing new features into economic models increases their complexity and dimensionality even further.[4] Moreover, in some applications, dynamic economic models must be solved a large number of times under different parameters vectors.[5]

Dynamic economic models do not typically admit closed-form solutions. Moreover, conventional numerical solution methods—perturbation, projection, and stochastic simulation—become intractable (i.e., either infeasible or inaccurate) if the number of state variables is large. First, projection methods build on tensor-product rules; they are accurate and fast in models with few state variables but their cost grows rapidly as the number of state variables increases; see, e.g., a projection method of Judd (1992). Second, stochastic simulation methods rely on Monte Carlo integration and least-squares learning; they are feasible in high-dimensional problems but their accuracy is severely limited by the low accuracy of Monte Carlo integration; also, least-squares learning is often numerically unstable, see, e.g., a parameterized expectation algorithm of Marcet (1988). Finally, perturbation methods solve models in a steady state using Taylor expansions of the models' equations. They are also practical for solving large-scale models but the range of their

[1] Heterogeneous agents models are studied in, e.g., Gaspar and Judd (1997), Krusell and Smith (1998), Kabourov and Manovskii (2009), Heathcote et al. (2009), Guvenen (2011), Guerrieri and Lorenzoni (2011), Michelacci and Pijoan-Mas (2012), Bai et al. (2012), Dmitriev and Roberts (2012), Cogley et al. (2013). Examples of models with multiple sectors and heterogeneous firms are Krusell et al. (1997), Altig et al. (2004), Klenow and Kryvtsov (2008), Gertler and Leahy (2008), Bloom (2009).

[2] Asset-pricing models are studied in, e.g., Kocherlakota and Pistaferri (2009), Duffie (2010), Borovička and Hansen (2012), Piazzesi and Schneider (2012); life-cycle models are studied in, e.g., Krueger and Kubler (2004, 2006), Gomes et al. (2008), Glover et al. (2011), Menzio et al. (2012), Guibaud et al. (forthcoming), Hasanhodzic and Kotlikoff (2013); international trade models are studied in, e.g., Caselli et al. (2011), Manova (2013), Heathcote and Perri (2013).

[3] New Keynesian models are studied in, e.g., Smets and Wouters (2003, 2007), Christiano et al. (2005, 2011), Del Negro et al. (2007), Chari et al. (2009), Woodford (2011), Mertens and Ravn (2011), Ravenna and Walsh (2011), Hall (2012), Cogan et al. (2013).

[4] Some examples from recent literature are climate change models (Cai et al., 2012), models of information and forecasts (Coibion and Gorodnichenko, 2008; Graham and Wright, 2009; Amador and Weil, 2010; Attanasio and Pavoni, 2011; Ellison and Sargent, 2012); fiscal policy models (Golosov et al., 2011; Golosov and Sargent, 2012; Evans et al., 2012); models with a risk of consumer default (Chatterjee et al., 2007); housing and mortgage models (Nakajima and Telyukova, 2011); resource war models (Acemoglu et al., 2011); political economy models (Song et al., 2012); financial crises models (Mendoza and Bianchi, 2011; Gertler et al., 2011).

[5] One example is a nested fixed-point estimation in econometrics, see, e.g., Fernández-Villaverde and Rubio-Ramírez (2007), Winschel and Krätzig (2010), Su and Judd (2012).

accuracy is uncertain, particularly in the presence of strong nonlinearities and kinks in decision functions; see, e.g., a perturbation method of Judd and Guu (1993).

In this chapter, we show how to re-design conventional projection, stochastic simulation, and perturbation methods to make them tractable (i.e., both feasible and accurate) in broad and empirically relevant classes of dynamic economic models with finite large numbers of continuous state variables.[6]

Let us highlight four key ideas. First, to reduce the cost of projection methods, Krueger and Kubler (2004) replace an expensive tensor-product grid with a low-cost, nonproduct Smolyak sparse grid. Second, Judd et al. (2011b) introduce a generalized stochastic simulation algorithm in which inaccurate Monte Carlo integration is replaced with accurate deterministic integration and in which unstable least-squares learning is replaced with numerically stable regression methods. Third, Judd et al. (2012) propose a ε-distinguishable set method that merges stochastic simulation and projection: it uses stochastic simulation to identify a high-probability area of the state space and it uses projection-style analysis to accurately solve the model in this area. Fourth, to increase the accuracy of perturbation methods, we describe two techniques. One is a change of variables of Judd (2003): it constructs many locally equivalent Taylor expansions and chooses the one that is most accurate globally. The other is a hybrid of local and global solutions by Maliar et al. (2013) that combines local solutions produced by a perturbation method with global solutions constructed to satisfy the model's equation exactly.

Other ingredients that help us to reduce the cost of solution methods in high-dimensional problems are efficient nonproduct techniques for representing, interpolating, and approximating functions; accurate, low-cost monomial integration formulas; and derivative-free solvers. Also, we describe an endogenous grid method of Carroll (2005) and an envelope condition method of Maliar and Maliar (2013) that simplify rootfinding in the Bellman equation, thus reducing dramatically the cost of value function iteration. Furthermore, we show precomputation techniques that save on cost by constructing solution manifolds outside the main iterative cycle, namely, precomputation of intratemporal choice by Maliar and Maliar (2005a), precomputation of integrals by Judd et al. (2011d), and imperfect aggregation by Maliar and Maliar (2001).

Finally, we argue that parallel computation can bring us a further considerable reduction in computational expense. Parallel computation arises naturally on recent computers,

[6] Krusell and Smith (1998) focus on a related but mathematically different problem. They assume a continuum of agents distributed in the interval $[0, 1]$ while we consider a finitely large number of agents N. The framework of Krusell and Smith (1998) is designed for modeling the aggregate behavior of a very large number of agents that are identical in fundamentals (e.g., consumers or firms) while our framework allows for combinations of any agents (consumers, firms, government, monetary authority) as long as their total number is not too large. The computational approach of Krusell and Smith (1998) also differs from ours in that they replace the true state space (a joint distribution of individual state variables) with a reduced state space (moments of such a distribution) while we operate with the true state space.

which are equipped with multiple central processing units (CPUs) and graphics processing units (GPUs). It is reasonable to expect capacities for parallel computation will continue to grow in the future. Therefore, we complement our survey of numerical methods with a discussion of parallel computation tools that may reduce the cost of solving large-scale dynamic economic models. First, we revisit the surveyed numerical methods and we distinguish those methods that are suitable for parallelizing. Second, we review MATLAB tools that are useful for parallel computation, including parallel computing toolboxes for multiple CPUs and GPUs, a deployment tool for creating executable files and a mex tool for incorporating routines from other programming languages such as C or Fortran. Finally, we discuss how to solve large-scale applications using supercomputers; in particular, we provide illustrative examples on Blacklight supercomputer from the Pittsburgh Supercomputing Center.

The numerical techniques surveyed in this chapter proved to be remarkably successful in applications. Krueger and Kubler (2004) solve life-cycle models with 20–30 periods using a Smolyak method. Judd et al. (2011b) compute accurate quadratic solutions to a multiagent optimal growth model with up to 40 state variables using GSSA . Hasanhodzic and Kotlikoff (2013) used GSSA to solve life-cycle models with up to 80 periods. Kollmann et al. (2011b) compare the performance of six state-of-the-art methods in the context of real business cycle models of international trade with up to 20 state variables. Maliar et al. (2013) show hybrids of perturbation and global solutions that take just a few seconds to construct but that are nearly as accurate as the most sophisticated global solutions. Furthermore, the techniques described in the chapter can solve problems with kinks and strong nonlinearities; in particular, a few recent papers solve moderately large new Keynesian models with a zero lower bound on nominal interest rates: see Judd et al. (2011d, 2012), Fernández-Villaverde et al. (2012), and Aruoba and Schorfheide (2012). Also, the surveyed techniques can be used in the context of large-scale dynamic programming problems: see Maliar and Maliar (2012a,b). Finally, Aldrich et al. (2011), Valero et al. (2012), Maliar (2013), and Cai et al. (2013a,b) demonstrate that a significant reduction in cost can be achieved using parallel computation techniques and more powerful hardware and software.[7]

An important question is how to evaluate the quality of numerical approximations. In the context of complex large-scale models, it is typically hard to prove convergence theorems and to derive error bounds analytically. As an alternative, we control the quality of approximations numerically using a two-stage procedure outlined in Judd et al. (2011b).

[7] Codes are available from the authors' web sites. In particular, a Fortran code for the Smolyak method is available at http://economics.sas.upenn.edu/∼dkrueger/research.php; a MATLAB code for GSSA method is available at http://www.stanford.edu/∼maliarl; C++ / DYNARE code for the perturbation-based hybrid solution methods is available at http://www.dynare.org/sebastien/; a C++/CUDA code for a value function iteration method is available at http://www.parallelecon.com.

In Stage 1, we attempt to compute a candidate solution. If the convergence is achieved, we proceed to Stage 2, in which we subject a candidate solution to a tight accuracy check. Specifically, we construct a new set of points on which we want a numerical solution to be accurate (typically, we use a set of points produced by stochastic simulation) and we compute unit-free residuals in the model's equations in all such points. If the economic significance of the approximation errors is small, we accept a candidate solution. Otherwise, we tighten up Stage 1; for example, we use a larger number of grid points, a more flexible approximating function, a more demanding convergence criterion, etc.

We assess the performance of the surveyed numerical solution methods in the context of a canonical optimal growth model with heterogeneous agents studied in Kollmann et al. (2011b). We implement six solution methods: three global Euler equation methods (a version of the Smolyak method in line with Judd et al. (2013), a generalized stochastic simulation algorithm of Judd et al. (2011b), and an ε-distinguishable set algorithm of Judd et al. (2012)); two global dynamic programming methods (specifically, two versions of the envelope condition method of Maliar and Maliar (2012b, 2013), one that solves for value function and the other that solves for derivatives of value function); and a hybrid of the perturbation and global solutions methods of Maliar et al. (2013). In our examples, we compute polynomial approximations of degrees up to 3, while the solution methods studied in Kollmann et al. (2011b) are limited to polynomials of degrees 2.

The surveyed methods proved to be tractable, accurate, and reliable. In our experiments, unit-free residuals in the model's equation are less than 0.001% on a stochastic simulation of 10,000 observations for most accurate methods. Our main message is that all three classes of methods considered can produce highly accurate solutions if properly implemented. Stochastic simulation methods become as accurate as projection methods if they build on accurate deterministic integration methods and numerically stable regression methods. Perturbation methods can deliver high accuracy levels if their local solutions are combined with global solutions using a hybrid style of analysis. Finally, dynamic programming methods are as accurate as Euler equation methods if they approximate the derivatives of value function instead of, or in addition to, value function itself.

In our numerical analysis, we explore a variety of interpolation, integration, optimization, fitting, and other computational techniques, and we evaluate the role of these techniques in accuracy and cost of the numerical solutions methods. At the end of this numerical exploration, we provide a detailed list of practical recommendations and tips on how computational techniques can be combined, coordinated, and implemented more efficiently in the context of large-scale applications.

The rest of the paper is organized as follows: in Section 2, we discuss the related literature. In Section 3, we provide a roadmap of the chapter. Through Sections (4)–(9), we introduce computational techniques that are tractable in problems with high dimensionality. In Section 10, we focus on parallel computation techniques. In Sections 11 and 12, we assess the performance of the surveyed numerical solution methods in the context of multiagent models. Finally, in Section 13, we conclude.

2. LITERATURE REVIEW

There is a variety of numerical methods in the literature for solving dynamic economic models; for reviews, see Taylor and Uhlig (1990), Rust (1996), Gaspar and Judd (1997), Judd (1998), Marimon and Scott (1999), Santos (1999), Christiano and Fisher (2000), Miranda and Fackler (2002), Adda and Cooper (2003), Aruoba et al. (2006), Kendrik et al. (2006), Heer and Maußner (2008, 2010), Lim and McNelis (2008), Stachursky (2009), Canova (2007), Den Haan (2010), Kollmann et al. (2011b). However, many of the existing methods are subject to the "curse of dimensionality"—that is, their computational expense grows exponentially with the dimensionality of state space.[8]

High-dimensional dynamic economic models represent three main challenges to numerical methods. First, the number of arguments in decision functions increases with the dimensionality of the problem and such functions become increasingly costly to approximate numerically (also, we may have more functions to approximate). Second, the cost of integration increases as the number of exogenous random variables increases (also, we may have more integrals to approximate). Finally, larger models are normally characterized by larger and more complex systems of equations, which are more expensive to solve numerically.

Three classes of numerical solution methods in the literature are projection, perturbation, and stochastic simulation. Projection methods are used in, e.g., Judd (1992), Gaspar and Judd (1997), Christiano and Fisher (2000), Aruoba et al. (2006), and Anderson et al. (2010). Conventional projection methods are accurate and fast in models with few state variables; however, they become intractable even in medium-scale models. This is because, first, they use expensive tensor-product rules both for interpolating decision functions and for approximating integrals and, second, because they use expensive Newton's methods for solving nonlinear systems of equations.

Perturbation methods are introduced to economics in Judd and Guu (1993) and become a popular tool in the literature. For examples of applications of perturbation methods, see Gaspar and Judd (1997), Judd (1998), Collard and Juillard (2001, 2011), Jin and Judd (2002), Judd (2003), Schmitt-Grohé and Uribe (2004), Fernández-Villaverde and Rubio-Ramírez (2007), Aruoba et al. (2006), Swanson et al. (2006), Kim et al. (2008), Chen and Zadrozny (2009), Reiter (2009), Lombardo (2010), Adjemian et al. (2011), Gomme and Klein (2011), Maliar and Maliar (2011), Maliar et al. (2013), Den Haan and De Wind (2012), Mertens and Judd (2013), and Guerrieri and Iacoviello (2013) among others. Perturbation methods are practical for problems with very high dimensionality;

[8] The term *curse of dimensionality* was originally used by Richard Bellman (1961) to refer to an exponential increase in volume associated with adding extra dimensions to a mathematical space.

however, the accuracy of local solutions may deteriorate dramatically away from the steady-state point in which such solutions are computed, especially in the presence of strong nonlinearities and kinks in decision functions.

Finally, simulation-based solution methods are introduced to the literature by Fair and Taylor (1983) and Marcet (1988). The former paper presents an extended path method for solving deterministic models while the latter paper proposes a parameterized expectation algorithm (PEA) for solving economic models with uncertainty. Simulation techniques are used in the context of many other solution methods, e.g., Smith (1991, 1993), Aiyagari (1994), Rust (1997), Krusell and Smith (1998), and Maliar and Maliar (2005a), as well as in the context of learning methods: see, e.g., Marcet and Sargent (1989), Tsitsiklis (1994), Bertsekas and Tsitsiklis (1996), Pakes and McGuire (2001), Evans and Honkapohja (2001), Weintraub et al. (2008), Powell (2011), Jirnyi and Lepetyuk (2011); see Birge and Louveaux (1997) for a review of stochastic programming methods and see also Fudenberg and Levine (1993) and Cho and Sargent (2008) for a related concept of self-conforming equilibria. Simulation and learning methods are feasible in problems with high dimensionality. However, Monte Carlo integration has a low rate of convergence—a square root of the number of observations—which considerably limits the accuracy of such methods since an infeasibly long simulation is needed to attain high accuracy levels. Moreover, least-squares learning is numerically unstable in the context of stochastic simulation.[9]

The above discussion raises three questions: first, how can one reduce the cost of conventional projection methods in high-dimensional problems (while maintaining their high accuracy levels)? Second, how can one increase the accuracy of perturbation methods (while maintaining their low computational expense)? Finally, how can one enhance their numerical stability of stochastic simulation methods and increase the accuracy (while maintaining a feasible simulation length)? These questions were addressed in the literature, and we survey the findings of this literature below.

To make projection methods tractable in large-scale problems, Krueger and Kubler (2004) introduce to the economic literature a Smolyak sparse grid technique; see also Malin et al. (2011). The sparse grid technique, introduced to the literature by Smolyak (1963), selects only a small subset of tensor-product grid elements that are most important for the quality of approximation. The Smolyak grid reduces the cost of projection methods dramatically without a significant accuracy loss. Winschel and Krätzig (2010) apply the Smolyak method for developing state-space filters that are tractable in problems with high dimensionality. Fernández-Villaverde et al. (2012) use the Smolyak method to solve a new Keynesian model. Finally, Judd et al. (2013) modify and generalize the Smolyak method in various dimensions to improve its performance in economic applications.

[9] Den Haan and Marcet (1990) find that the implementation of PEA that uses least-squares learning becomes problematic even for a second-degree polynomial. Specifically, a cross term in a polynomial function is highly correlated with other terms and must be removed from a regression.

To increase the accuracy of stochastic simulation methods, Judd et al. (2011b) introduce a generalized stochastic simulation algorithm (GSSA) which uses stochastic simulation as a way to obtain grid points for computing solutions but replaces an inaccurate Monte Carlo integration method with highly accurate deterministic (quadrature and monomial) methods. Furthermore, GSSA replaces the least-squares learning method with regression methods that are robust to ill-conditioned problems, including least-squares methods that use singular value decomposition, Tikhonov regularization, least-absolute deviations methods, and the principal component regression method. The key advantage of stochastic simulation and learning methods is that they solve models only in the area of the state space where the solution "lives." Thus, they avoid the cost of finding a solution in those areas that are unlikely to happen in equilibrium. GSSA preserves this useful feature but corrects the shortcomings of the earlier stochastic simulation approaches. It delivers accuracy levels that are comparable to the best accuracy attained in the related literature.

Furthermore, Judd et al. (2010, 2012) introduce two solution methods that merge stochastic simulation and projection techniques, a cluster grid algorithm (CGA) and ε-distinguishable set method (EDS). Both methods use stochastic simulation to identify a high-probability area of the state space, cover this area with a relatively small set of representative points, and use projection-style techniques to solve a model on the constructed set (grid) of points. The two methods differ in the way in which they construct representative points: CGA partitions the simulated data into clusters and uses the centers of the clusters as grid points while EDS selects a roughly evenly spaced subset of simulated points. These two approaches are used to solve moderately large new Keynesian models, namely, Judd et al. (2011b) and Aruoba and Schorfheide (2012) use clustering techniques while Judd et al. (2012) use the EDS construction.

As far as integration is concerned, there is a variety of methods that are tractable in high-dimensional problems, e.g., Monte Carlo methods, quasi-Monte Carlo methods, and nonparametric methods; see Niederreiter (1992), Geweke (1996), Rust (1997), Judd (1998), Pagan and Ullah (1999), Scott and Sain (2005) for reviews. However, the quality of approximations of integrals differs considerably across methods. For models with smooth decision functions, deterministic Gaussian quadrature integration methods with just a few nodes are far more accurate than Monte Carlo methods with thousands of random draws; see Judd et al. (2011a,b) for numerical examples. However, the cost of Gaussian product rules is prohibitive in problems with high dimensionality; see Gaspar and Judd (1997). To ameliorate the curse of dimensionality, Judd (1998) introduces to the economic literature nonproduct monomial integration formulas. Monomial formulas turn out to be a key piece for constructing global solution methods that are tractable in high-dimensional applications. Such formulas combine a low cost with high accuracy and can be generalized to the case of correlated exogenous shocks using a Cholesky decomposition; see Judd et al. (2011b) for a detailed description of monomial formulas, as well as for a numerical assessment of the accuracy and cost of such formulas in the context of economically relevant examples.

Finally, we focus on numerical methods that can solve large systems of nonlinear equations. In the context of Euler equation approaches, Maliar et al. (2011) argue that the cost of finding a solution to a system of equilibrium conditions can be reduced by dividing the whole system into two parts: intratemporal choice conditions (those that contain variables known at time t) and intertemporal choice conditions (those that contain some variables unknown at time t). Maliar and Maliar (2013) suggest a similar construction in the context of dynamic programming methods; specifically, they separate all optimality conditions into a system of the usual equations that identifies the optimal quantities and a system of functional equations that identifies a value function.

The system of intertemporal choice conditions (functional equations) must be solved with respect to the parameters of the approximating functions. In a high-dimensional model this system is large, and solving it with Newton-style methods may be expensive. In turn, the system of intratemporal choice conditions is much smaller in size; however, Newton's methods can still be expensive because such a system must be solved a large number of times (in each grid point, time period, and future integration node). As an alternative to Newton's methods, we advocate the use of a simple derivative-free fixed-point iteration method in line with the Gauss-Jacobi and Gauss-Siedel schemes. In particular, Maliar et al. (2011) propose a simple iteration-on-allocation solver that can find the intratemporal choice in many points simultaneously and that produces essentially zero approximation errors in all intratemporal choice conditions. See Wright and Williams (1984), Miranda and Helmberger (1988), and Marcet (1988) for early applications of fixed-point iteration to economic problems; see Judd (1998) for a review of various fixed-point iteration schemes. Also, see Eaves and Schmedders (1999) and Judd et al. (2012b), respectively, for applications of homotopy methods and efficient Newton's methods to economic problems.

A February 2011 special issue of the Journal of Economic Dynamics and Control (henceforth, the JEDC project) studied the performance of six state-of-the-art solution methods in the context of an optimal growth model with heterogeneous agents (interpreted as countries). The model includes up to 10 countries (20 state variables) and features heterogeneity in preferences and technology, complete markets, capital adjustment cost, and elastic labor supply. The objectives of this project are outlined in Den Haan et al. (2011). The participating methods are a perturbation method of Kollmann et al. (2011a), a stochastic simulation and cluster grid algorithms of Maliar et al. (2011), a monomial rule Galerkin method of Pichler (2011), and a Smolyak's collocation method of Malin et al. (2011). The methodology of the numerical analysis is described in Juillard and Villemot (2011). In particular, they develop a test suite that evaluates the accuracy of solutions by computing unit-free residuals in the model's equations. The residuals are computed both on sets of points produced by stochastic simulation and on sets of points situated on a hypersphere. These two kinds of accuracy checks are introduced in Jin and Judd (2002) and Judd (1992), respectively; see also Den Haan and Marcet (1994) and Santos (2000) for other techniques for accuracy evaluation.

The results of the JEDC comparison are summarized in Kollmann et al. (2011b). The main findings are as follows: First, an increase in the degree of an approximating polynomial function by 1 increases the accuracy levels roughly by an order of magnitude (provided that other computational techniques, such as integration, fitting, intratemporal choice, etc., are sufficiently accurate). Second, methods that operate on simulation-based grids are very accurate in the high-probability area of the state space, while methods that operate on exogenous hypercube domains are less accurate in that area—although their accuracy is more uniformly distributed on a large multidimensional hypercube. Third, Monte Carlo integration is very inaccurate and restricts the overall accuracy of solutions. In contrast, monomial integration formulas are very accurate and reliable. Finally, approximating accurately the intratemporal choice is critical for the overall accuracy of solutions.

The importance of solving for intratemporal choice with a high degree of accuracy is emphasized by Maliar et al. (2011), who define the intertemporal choice (capital functions) parametrically; however, they define the intratemporal choice (consumption and leisure) nonparametrically as quantities that satisfy the intratemporal choice conditions given the future intertemporal choice. Under this construction, stochastic simulation and cluster grid algorithms of Maliar et al. (2011) solve for the intratemporal choice with essentially zero approximation errors (using an iteration-on-allocation solver). In contrast, the other methods participating in the JEDC comparison solve for some of the intertemporal choice variables parametrically and face such large errors in the intratemporal choice conditions that they dominate the overall accuracy of their solutions.

All six methods that participated in the JEDC comparison analysis work with Euler equations. In addition to the Euler equation approach, we are interested in constructing Bellman equation approaches that are tractable in high-dimensional applications. In general, there is no simple one-to-one relation between the Bellman and Euler equation approaches. For some problems, the value function is not differentiable, and we do not have Euler equations. On the contrary, for other problems (e.g., problems with distortionary taxation, externalities, etc.), we are able to derive the Euler equation even though such problems do not admit dynamic programming representations; for a general discussion of dynamic programming methods and their applications to economic problems, see Bertsekas and Tsitsiklis (1996), Rust (1996, 1997, 2008), Judd (1998), Santos (1999), Judd et al. (2003), Aruoba et al. (2006), Powell (2011), Fukushima and Waki (2011), Cai and Judd (2010, 2012), among others.

The Bellman and Euler equation approaches are affected by the curse of dimensionality in a similar way. Hence, the same kinds of remedies can be used to enhance their performance in large-scale applications, including nonproduct grid construction, monomial integration, and derivative-free solvers. However, there is an additional important computational issue that is specific to the dynamic programming methods: expensive rootfinding. To find a solution to the Bellman equation in a single grid point, conventional value function iteration (VFI) explores many different candidate points and, in each such point, it interpolates value function in many integration nodes to approximate expectations. Conventional VFI is costly even in low-dimensional problems; see Aruoba et al. (2006) for an assessment of its cost.

Two alternatives to conventional VFI are proposed in the literature. First, Carroll (2005) shows an endogenous grid method (EGM) that can significantly reduce the cost of conventional VFI by using future endogenous state variables for constructing grid points instead of the current ones; see also Barillas and Fernández-Villaverde (2007) and Villemot (2012) for applications of EGM to models with elastic labor supply and sovereign debt, respectively. Second, Maliar and Maliar (2013) show an envelope condition method (ECM) that replaces conventional expensive backward-looking iteration on value function with a cheaper, forward-looking iteration. Also, Maliar and Maliar (2013) develop versions of EGM and ECM that approximate derivatives of the value function and deliver much higher accuracy levels than similar methods approximating the value function itself. Finally, Maliar and Maliar (2012a,b) use a version of ECM to solve a multicountry model studied in Kollmann et al. (2011b) and show that value function iteration methods can successfully compete with most efficient Euler equation methods in high-dimensional applications.

Precomputation—computation of a solution to some model's equations outside the main iterative cycle—is a technique that can reduce the cost of global solution methods even further. We review three examples. First, Maliar and Maliar (2005a) introduce a technique of precomputing of intratemporal choice. It constructs the intratemporal choice manifolds outside the main iterative cycle and uses the constructed manifolds inside the cycle as if a closed-form solution were available; see Maliar et al. (2011) for a further development of this technique. Second, Judd et al. (2011d) propose a technique of precomputation of integrals. This technique makes it possible to construct conditional expectation functions in the stage of initialization of a solution algorithm and, in effect, converts a stochastic problem into a deterministic one. Finally, Maliar and Maliar (2001, 2003a) introduce an analytical technique of imperfect aggregation, which allows us to characterize the aggregate behavior of a multiagent economy in terms of a one-agent model.

There are many other numerical techniques that are useful in the context of global solution methods. In particular, Tauchen (1986) and Tauchen and Hussey (1991) propose a discretization method that approximates a Markov process with a finite-state Markov chain. Such a discretization can be performed using nonproduct rules, combined with other computational techniques that are tractable in large-scale applications. For example, Maliar et al. (2010) and Young (2010) develop variants of Krusell and Smith's (1998) method that replaces stochastic simulation with iteration on discretized ergodic distribution; the former method iterates backward as in Rios-Rull (1997) while the latter method introduces a forward iteration; see also Horvath (2012) for an extension. These two methods deliver the most accurate approximations to the aggregate law of motion in the context of Krusell and Smith's (1998) model studied in the comparison analysis of

Den Haan (2010). Of special interest are techniques that are designed for dealing with multiplicity of equilibria, such as simulation-based methods of Peralta-Alva and Santos (2005), a global solution method of Feng et al. (2009), and Gröbner bases introduced to economics by Kubler and Schmedders (2010). Finally, an interaction of a large number of heterogeneous agents is studied by agent-based computational economics. In this literature, behavioral rules of agents are not necessarily derived from optimization and that the interaction of the agents does not necessarily lead to an equilibrium; see Tesfatsion and Judd (2006) for a review.

We next focus on the perturbation class of solution methods. Specifically, we survey two techniques that increase the accuracy of local solutions produced by perturbation methods. The first technique—a change of variables—is introduced by Judd (2003), who pointed out that an ordinary Taylor expansion can be dominated in accuracy by other expansions implied by changes of variables (e.g., an expansion in levels may lead to more accurate solutions than that in logarithms or vice versa). All the expansions are equivalent in the steady state but differ globally. The goal is to choose an expansion that performs best in terms of accuracy on the relevant domain. In the context of a simple optimal growth model, Judd (2003) finds that a change of variables can increase the accuracy of the conventional perturbation method by two orders of magnitude. Fernández-Villaverde and Rubio-Ramírez (2006) show how to apply the method of change of variables to a model with uncertainty and an elastic labor supply.

The second technique—a hybrid of local and global solutions—is developed by Maliar et al. (2013). Their general presentation of the hybrid method encompasses some previous examples in the literature obtained using linear and loglinear solution methods; see Dotsey and Mao (1992) and Maliar et al. (2010, 2011). This perturbation-based method computes a standard perturbation solution, fixes some perturbation decision functions, and finds the remaining decision functions to exactly satisfy certain models' equations. The construction of the latter part of the hybrid solution mimics global solution methods: for each point of the state space considered, nonlinear equations are solved either analytically (when closed-form solutions are available) or with a numerical solver. In numerical examples studied in Maliar et al. (2013), some hybrid solutions are orders of magnitude more accurate than the original perturbation solutions and are comparable in accuracy to solutions produced by global solution methods.

Finally, there is another technique that can help us to increase the accuracy of perturbation methods. Namely, it is possible to compute Taylor expansion around stochastic steady state instead of deterministic steady state. Such a steady state is computed by taking into account the attitude of agents toward risk. This idea is developed in Juillard (2011) and Maliar and Maliar (2011): the former article computes a stochastic steady state numerically, whereas the latter article uses analytical techniques of precomputation of integrals introduced in Judd et al. (2011d).

We complement our survey of efficient numerical methods with a discussion of recent developments in hardware and software that can help to reduce the cost of large-scale problems. Parallel computation is the main tool for dealing with computationally intensive tasks in recent computer science literature. A large number of central processing units (CPUs) or graphics processing units (GPUs) are connected with a fast network and are coordinated to perform a single job. Early applications of parallel computation to economic problems are dated back to Amman (1986, 1990), Chong and Hendry (1986), Coleman (1992), Nagurney and Zhang (1998); also, see Nagurney (1996) for a survey. But after the early contributions, parallel computation received little attention in the economic literature. Recently, the situation has begun to change. In particular, Doornik et al. (2006) review applications of parallel computation in econometrics; Creel (2005, 2008) and Creel and Goffe (2008) illustrate the benefits of parallel computation in the context of several economically relevant examples; Sims et al. (2008) employ parallel computation in the context of large-scale Markov switching models; Aldrich et al. (2011), Morozov and Mathur (2012) apply GPU computation to solve dynamic economic models; Durham and Geweke (2012) use GPUs to produce sequential posterior simulators for applied Bayesian inference; Cai et al. (2012) apply high-throughput computing (Condor network) to implement value function iteration; Valero et al. (2013) review parallel computing tools available in MATLAB and illustrate their application by way of examples; and, finally, Maliar (2013) assesses efficiency of parallelization using message passing interface (MPI) and open memory programming (OpenMP) in the context of high-performance computing (a Blacklight supercomputer).

In particular, Maliar (2013) finds that information transfers on supercomputers are far more expensive than on desktops. Hence, the problem must be sufficiently large to ensure gains from parallelization. The task assigned to each core must be at least few seconds if several cores are used, and it must be a minute or more if a large number (thousands) of cores are used. Maliar (2013) also finds that for small problems, OpenMP leads to a higher efficiency of parallelization than MPI. Furthermore, Valero et al. (2013) explore options for reducing the cost of a Smolyak solution method in the context of large-scale models studied in the JEDC project. Parallelizing the Smolyak method effectively is a nontrivial task because there are large information transfers between the outer and inner loops and because certain steps should be implemented in a serial manner. Nonetheless, considerable gains from parallelization are possible even on a desktop computer. Specifically, in a model with 16 state variables, Valero et al. (2013) attain the efficiency of parallelization of nearly 90% on a four-core machine via a parfor tool. Furthermore, translating expensive parts of the MATLAB code into C++ via a mex tool also leads to a considerable reduction in computational expense in some examples. However, transferring expensive computations to GPUs does not reduce the computational expense: a high cost of transfers between CPUs and GPUs outweighs the gains from parallelization.

3. THE CHAPTER AT A GLANCE

In this section, we provide a roadmap of the chapter and highlight the key ideas using a collection of simple examples.

A Neoclassical Stochastic Growth Model

The techniques surveyed in the chapter are designed for dealing with high-dimensional problems. However, to explain these techniques, we use the simplest possible framework, the standard one-sector neoclassical growth model. Later, in Sections 11 and 12, we show how such techniques can be applied for solving large-scale heterogeneous-agents models.

We consider a *model with elastic labor supply*. The agent solves:

$$\max_{\{k_{t+1},c_t,\ell_t\}_{t=0,\ldots,\infty}} E_0 \left\{ \sum_{t=0}^{\infty} \beta^t u\left(c_t, \ell_t\right) \right\} \tag{1}$$

$$\text{s.t. } c_t + k_{t+1} = (1-\delta)\, k_t + \theta_t f\left(k_t, \ell_t\right), \tag{2}$$

$$\ln \theta_{t+1} = \rho \ln \theta_t + \sigma \epsilon_{t+1}, \quad \epsilon_{t+1} \sim \mathcal{N}(0,1), \tag{3}$$

where (k_0, θ_0) is given; E_t is the expectation operator conditional on information at time t; c_t, ℓ_t, k_{t+1}, and θ_t are consumption, labor, end-of-period capital, and productivity level, respectively; $\beta \in (0,1)$; $\delta \in (0,1]$; $\rho \in (-1,1)$; $\sigma \geq 0$; u and f are the utility and production functions, respectively, both of which are strictly increasing, continuously differentiable, and concave.

Our goal is to solve for a recursive Markov equilibrium in which the decisions on next-period capital, consumption, and labor are made according to some time-invariant state contingent functions $k' = K(k,\theta)$, $c = C(k,\theta)$, and $\ell = \mathcal{L}(k,\theta)$.

A version of model (1)–(3) in which the agent does not value leisure and supplies to the market all her time endowment is referred to as *a model with inelastic labor supply*. Formally, such a model is obtained by replacing $u(c_t,\ell_t)$ and $f(k_t,\ell_t)$ with $u(c_t)$ and $f(k_t)$ in (1) and (2), respectively.

First-Order Conditions

We assume that a solution to model (1)–(3) is interior and satisfies the first-order conditions (FOCs)

$$u_1\left(c_t,\ell_t\right) = \beta E_t \left\{ u_1\left(c_{t+1},\ell_{t+1}\right)\left[1-\delta+\theta_{t+1} f_1\left(k_{t+1},\ell_{t+1}\right)\right]\right\}, \tag{4}$$

$$u_2\left(c_t,\ell_t\right) = u_1\left(c_t,\ell_t\right)\theta_t f_2\left(k_t,\ell_t\right), \tag{5}$$

and budget constraint (2). Here, and further on, notation of type F_j stands for the first-order partial derivative of a function $F(\ldots, x_j, \ldots)$ with respect to a variable x_j. Condition (4) is called the *Euler equation*.

An Example of a Global Projection-Style Euler Equation Method

We approximate functions K, C, and $\mathcal{L}$ numerically. As a starting point, we consider a projection-style method in line with Judd (1992) that approximates these functions to satisfy (2)–(5) on a grid of points.

(EEM): A global projection-style Euler equation method.

Step 1. Choose functional forms $\widehat{K}(\cdot, b^k)$, $\widehat{C}(\cdot, b^c)$, and $\widehat{\mathcal{L}}(\cdot, b^\ell)$ for representing K, C, and $\mathcal{L}$, where b^k, b^c, b^ℓ are the coefficient vectors.
Choose a grid $\{k_m, \theta_m\}_{m=1,\ldots,M}$ on which $\widehat{K}$, $\widehat{C}$, and $\widehat{\mathcal{L}}$ are constructed.
Step 2. Choose nodes, ϵ_j, and weights, $\omega_j, j = 1, \ldots, J$, for approximating integrals.
Compute next-period productivity $\theta'_{m,j} = \theta_m^\rho \exp(\epsilon_j)$ for all j, m.
Step 3. Solve for b^k, b^c, b^ℓ that approximately satisfy the model's equations:

$$u_1(c_m, \ell_m) = \beta \sum_{j=1}^{J} \omega_j \cdot \left[u_1\left(c'_{m,j}, \ell'_{m,j}\right)\left(1 - \delta + \theta'_{m,j} f_1\left(k'_m, \ell'_{m,j}\right)\right)\right],$$

$$u_2(c_m, \ell_m) = u_1(c_m, \ell_m)\,\theta_m f_2(k_m, \ell_m),$$

$$c_m = (1 - \delta) k_m + \theta_m f(k_m, \ell_m) - k'_m.$$

We use the assumed decision functions to determine the choices in the current period $k'_m = \widehat{K}\left(k_m, \theta_m; b^k\right)$, $c_m = \widehat{C}\left(k_m, \theta_m; b^c\right)$, $\ell_m = \widehat{\mathcal{L}}\left(k_m, \theta_m; b^\ell\right)$, as well as to determine these choices in J possible future states $k''_{m,j} = \widehat{K}\left(k'_m, \theta'_{m,j}; b^k\right)$, $c'_{m,j} = \widehat{C}\left(k'_m, \theta'_{m,j}; b^c\right)$, and $\ell'_{m,j} = \widehat{\mathcal{L}}\left(k'_m, \theta'_{m,j}; b^\ell\right)$.

Unidimensional Grid Points and Basis Functions

To solve the model, we discretize the state space into a finite set of grid points $\{k_m, \theta_m\}_{m=1,\ldots,M}$. Our construction of a multidimensional grid begins with unidimensional grid points and basis functions. The simplest possible choice is a family of ordinary polynomials and a grid of uniformly spaced points but many other choices are possible. In particular, a useful alternative is a family of Chebyshev polynomials and a grid composed of extrema of Chebyshev polynomials. Such polynomials are defined in an interval $[-1, 1]$, and thus, the model's variables such as k and θ must be rescaled to be inside this interval prior to any computation. In Table 1, we compare two choices discussed above: one is ordinary polynomials and a grid of uniformly spaced points and the other is Chebyshev polynomials and a grid of their extrema.

As we see, Chebyshev polynomials are just linear combinations of ordinary polynomials. If we had an infinite arithmetic precision on a computer, it would not matter which family of polynomials we use. But with a finite number of floating points, Chebyshev polynomials have an advantage over ordinary polynomials. To see the point, in Figure 1 we plot ordinary and Chebyshev polynomials of degrees from 1 to 5.

For the ordinary polynomial family, the basis functions look very similar on $\mathbb{R}_+$. Approximation methods using ordinary polynomials may fail because they cannot

Table 1 Ordinary and Chebyshev unidimensional polynomials.

n	Ordinary polyn. of degree $n-1$	Uniform grid of points on $[-1,1]$	Chebyshev polyn. of degree $n-1$	n extrema of Chebyshev polyn. of degree $n-1$
1	1	0	1	0
2	x	$-1 \quad 1$	x	$-1 \quad 1$
3	x^2	$-1 \quad 0 \quad 1$	$2x^2-1$	$-1 \quad 0 \quad 1$
4	x^3	$-1, \quad -\frac{2}{3}, \quad \frac{2}{3}, \quad 1$	$4x^3-3x$	$-1, \quad -\frac{1}{\sqrt{2}}, \quad \frac{1}{2}, \quad 1$
5	x^4	$-1 \quad -\frac{1}{2} \quad 0 \quad -\frac{1}{2} \quad 1$	$8x^4-8x^2+1$	$-1 \quad -\frac{1}{\sqrt{2}} \quad 0 \quad -\frac{1}{\sqrt{2}} \quad 1$

Notes: Ordinary polynomial of degree $n-1$ is given by $P_{n-1}(x) = x^{n-1}$; Chebyshev polynomial of degree $n-1$ is given by $T_{n-1}(x) = cos((n-1)cos^{-1}(x))$; and finally, n extrema of Chebyshev polynomials of degree $n-1$ are given by $\zeta_j^n = -cos(\pi(j-1)/(n-1)), j = 1, \ldots, n$.

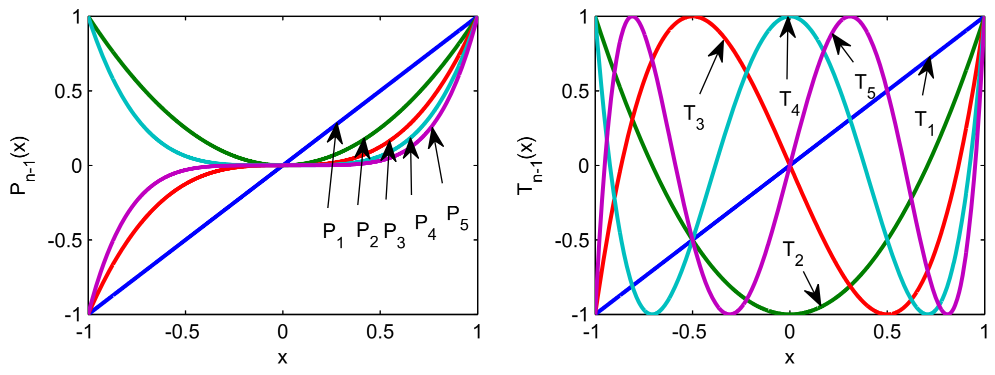


Figure 1 (a) Ordinary polynomials. (b) Chebyshev polynomials.

distinguish between similarly shaped polynomial terms such as x^2 and x^4. In contrast, for the Chebyshev polynomial family, basis functions have very different shapes and are easy to distinguish.

We now illustrate the use of Chebyshev polynomials for approximation by way of example.

Example 1. Let $f(x)$ be a function defined on an interval $[-1, 1]$, and let us approximate this function with a Chebyshev polynomial function of degree 2, i.e.,

$$f(x) \approx \widehat{f}(x; b) = b_1 + b_2 x + b_3 \left(2x^2 - 1\right).$$

We compute $b \equiv (b_1, b_2, b_3)$ so that $\widehat{f}(\cdot; b)$ and f coincide in three extrema of Chebyshev polynomials, namely, $\{-1, 0, 1\}$,

$$\widehat{f}(-1; b) = b_1 + b_2 \cdot (-1) + b_3 \left(2 \cdot (-1)^2 - 1\right) = f(-1),$$
$$\widehat{f}(0; b) = b_1 + b_2 \cdot 0 + b_3 \left(2 \cdot 0^2 - 1\right) = f(0),$$
$$\widehat{f}(1; b) = b_1 + b_2 \cdot 1 + b_3 \left(2 \cdot 1^2 - 1\right) = f(1).$$

This leads us to a system of three linear equations with three unknowns that has a unique solution

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & -1 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} f(-1) \\ f(0) \\ f(1) \end{bmatrix}$$
$$= \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} f(-1) \\ f(0) \\ f(1) \end{bmatrix} = \begin{bmatrix} \frac{f(-1)}{4} + \frac{f(0)}{2} + \frac{f(1)}{4} \\ -\frac{f(-1)}{2} + \frac{f(1)}{2} \\ \frac{f(-1)}{4} - \frac{f(0)}{2} + \frac{f(1)}{4} \end{bmatrix}.$$

It is possible to use Chebyshev polynomials with other grids, but the grid of extrema of Chebyshev polynomials is a perfect match. (The extrema listed in Table 1 are also seen in Figure 1.) First, the resulting approximations are uniformly accurate, and the error bounds can be constructed. Second, there is a unique set of coefficients such that a Chebyshev polynomial function of degree $n-1$ matches exactly n given values, and this property carries over to multidimensional approximations that build on unidimensional Chebyshev polynomials. Finally, the coefficients that we compute in our example using an inverse problem can be derived in a closed form using the property of orthogonality of Chebyshev polynomials. The advantages of Chebyshev polynomials for approximation are emphasized by Judd (1992) in the context of projection methods for solving dynamic economic models; see Judd (1998) for a further discussion of Chebyshev as well as other orthogonal polynomials (Hermite, Legendre, etc.).

Multidimensional Grid Points and Basis Functions

In Step 1 of the Euler equation algorithm, we must specify a method for approximating, representing, and interpolating two-dimensional functions. A tensor-product method constructs multidimensional grid points and basis functions using all possible combinations of unidimensional grid points and basis functions. As an example, let us approximate the capital decision function K. First, we take two grid points for each state variable, namely, $\{k_1, k_2\}$ and $\{\theta_1, \theta_2\}$, and we combine them to construct two-dimensional grid points, $\{(k_1, \theta_1), (k_1, \theta_2), (k_2, \theta_1), (k_2, \theta_2)\}$. Second, we take two basis functions for each state variable, namely, $\{1, k\}$ and $\{1, \theta\}$, and we combine them to construct two-dimensional basis functions $\{1, k, \theta, k\theta\}$. Third, we construct a flexible functional form for approximating K,

$$\widehat{K}(k, \theta; b) = b_1 + b_2 k + b_3 \theta + b_4 k\theta. \tag{6}$$

Finally, we identify the four unknown coefficients $(b_1, b_2, b_3, b_4) \equiv b$ such that $K(k, \theta)$ and $\widehat{K}(k, \theta; b)$ coincide exactly in the four grid points constructed. That is, we write

$\mathcal{B}b = w$, where

$$\mathcal{B} = \begin{bmatrix} 1 & k_1 & \theta_1 & k_1\theta_1 \\ 1 & k_1 & \theta_2 & k_1\theta_2 \\ 1 & k_2 & \theta_1 & k_2\theta_1 \\ 1 & k_2 & \theta_2 & k_2\theta_2 \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}, \quad w = \begin{bmatrix} K(k_1, \theta_1) \\ K(k_1, \theta_2) \\ K(k_2, \theta_1) \\ K(k_2, \theta_2) \end{bmatrix}. \tag{7}$$

If $\mathcal{B}$ has full rank, then coefficient vector b is uniquely determined by $b = \mathcal{B}^{-1}w$. The obtained approximation (6) can be used to interpolate the capital decision function in each point off the grid.

Tensor-product constructions are successfully used in the literature to solve economic models with few state variables; see, e.g., Judd (1992). However, the number of grid points and basis functions grows exponentially (i.e., as 2^d) with the number of state variables d. For problems with high dimensionality, we need nonproduct techniques for constructing multidimensional grid points and basis functions.

Nonproduct techniques for constructing multidimensional grids are the first essential ingredient of solution methods for high-dimensional problems. We survey several such techniques, including Smolyak (sparse), stochastic simulation, ε-distinguishable set, and cluster grids. The above techniques attack the curse of dimensionality in two different ways: one is to reduce the number of points within a fixed solution domain and the other is to reduce the size of the domain itself. To be specific, the Smolyak method uses a fixed geometry, a multidimensional hypercube, but chooses a small set of points within the hypercube. In turn, a stochastic simulation method uses an adaptive geometry: it places grid points exclusively in the area of the state space in which the solution "lives" and thus avoids the cost of computing a solution in those areas that are unlikely to happen in equilibrium. Finally, an ε-distinguishable set and cluster grid methods combine an adaptive geometry with an efficient discretization: they distinguish a high-probability area of the state space and cover such an area with a relatively small set of points. We survey these techniques in Section 4.

Numerical Integration

In Step 2 of the Euler equation algorithm, we need to specify a method for approximating integrals. As a starting point, we consider a simple two-node Gauss-Hermite quadrature method that approximates an integral of a function of a normally distributed variable $\epsilon \sim \mathcal{N}(0, \sigma^2)$ with a weighted average of just two values $\epsilon_1 = -\sigma$ and $\epsilon_2 = \sigma$ that happen with probability $\omega_1 = \omega_2 = \frac{1}{2}$, i.e.,

$$\int_{-\infty}^{\infty} G(\epsilon)\, w(\epsilon)\, d\epsilon \approx G(\epsilon_1)\,\omega_1 + G(\epsilon_2)\,\omega_2 = \frac{1}{2}\left[G(-\sigma) + G(\sigma)\right],$$

where G is a bounded continuous function, and w is a density function of a normal distribution. Alternatively, we can use a three-node Gauss-Hermite quadrature method, which uses nodes $\epsilon_1 = 0, \epsilon_2 = \sigma\sqrt{\frac{3}{2}}$, and $\epsilon_3 = -\sigma\sqrt{\frac{3}{2}}$ and weights $\omega_1 = \frac{2\sqrt{\pi}}{3}$ and $\omega_2 = \omega_3 = \frac{\sqrt{\pi}}{6}$ or even a one-node Gauss-Hermite quadrature method, which uses $\epsilon_1 = 0$ and $\omega_1 = 1$.

Another possibility is to approximate integrals using Monte Carlo integration. We can make J random draws and approximate an integral with a simple average of the draws,

$$\int_{-\infty}^{\infty} G(\epsilon)\, w(\epsilon)\, d\epsilon \approx \frac{1}{J}\sum_{j=1}^{J} G\left(\epsilon_j\right).$$

Let us compare the above integration methods using an example.

Example 2. Consider a quadratic function $G(\epsilon) = b_1 + b_2\epsilon + b_3\epsilon^2$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

(i) An exact integral is $\mathcal{I} \equiv \int_{-\infty}^{\infty}\left(b_1 + b_2\epsilon + b_3\epsilon^2\right) w(\epsilon)\, d\epsilon = b_1 + b_3\sigma^2$;
(ii) A two-node Gauss-Hermite quadrature integration method yields $\mathcal{I} \approx b_1 + b_3\sigma^2$;
(iii) A one-node Gauss-Hermite quadrature integration method yields $\mathcal{I} \approx b_1$;
(iv) A Monte Carlo integration method yields $\mathcal{I} \approx b_1 + b_2\left[\frac{1}{J}\sum_{j=1}^{J}\epsilon_j\right] + b_3\left[\frac{1}{J}\sum_{j=1}^{J}\epsilon_j^2\right]$.

Note that the quadrature method with two nodes delivers the exact value of the integral. Even with just one node, the quadrature method can deliver an accurate integral if G is close to linear (which is often the case in real business cycle models), i.e., $b_3 \approx 0$. To evaluate the accuracy of Monte Carlo integration, let us use $\sigma = 0.01$, which is consistent with the magnitude of fluctuations in real business cycle models. Let us concentrate just on the term $\frac{1}{J}\sum_{j=1}^{J}\epsilon_j$ for which the expected value and standard deviation are $E\left[\frac{1}{J}\sum_{j=1}^{J}\epsilon_j\right] = 0$ and $std\left[\frac{1}{J}\sum_{j=1}^{J}\epsilon_j\right] = \frac{\sigma}{\sqrt{J}}$, respectively. The standard deviation depends on the number of random draws: with one random draw, it is 0.01 and with 1,000,000 draws, it is $\frac{0.01}{\sqrt{1{,}000{,}000}} = 10^{-5}$. The last number represents an (expected) error in approximating the integral and places a restriction on the overall accuracy of solutions that can be attained by a solution algorithm using Monte Carlo integration. An infeasibly long simulation is needed for a Monte Carlo method to deliver the same accuracy level as that of Gauss-Hermite quadrature in our example.

Why is Monte Carlo integration inefficient in the context of numerical methods for solving dynamic economic models? This is because we compute expectations as do econometricians, who do not know the true density function of the data-generating process and have no choice but to estimate such a function from noisy data using a regression. However, when solving an economic model, we do know the process for shocks. Hence, we can construct the "true" density function and we can use such a function to compute integrals very accurately, which is done by the Gauss-Hermite quadrature method.

In principle, the Gauss-Hermite quadrature method can be extended to an arbitrary dimension using a tensor-product rule; see, e.g., Gaspar and Judd (1997). However, the number of integration nodes grows exponentially with the number of shocks in the model. To ameliorate the curse of dimensionality, we again need to avoid product rules. Monomial formulas are a nonproduct integration method that combines high accuracy and low cost; this class of integration methods is introduced to the economic literature in Judd (1998). Integration methods that are tractable in problems with a large number of shocks are surveyed in Section 5.

Optimization Methods

In Step 3 of the algorithm, we need to solve a system of nonlinear equations with respect to the unknown parameters vectors b^k, b^c, b^ℓ. In principle, this can be done with Newton-style optimization methods; see, e.g., Judd (1992). Such methods compute first and second derivatives of an objective function with respect to the unknowns and move in the direction of gradient descent until a solution is found. Newton methods are fast and efficient in small problems but become increasingly expensive when the number of unknowns increases. In high-dimensional applications, we may have thousands of parameters in approximating functions, and the cost of computing derivatives may be prohibitive. In such applications, derivative-free optimization methods are an effective alternative. In particular, a useful choice is a fixed-point iteration method that finds a root to an equation $x = F(x)$ by constructing a sequence $x^{(i+1)} = F(x^{(i)})$. We illustrate this method using an example.

Example 3. Consider an equation $x^3 - x - 1 = 0$. Let us rewrite this equation as $x = (x+1)^{1/3}$ and construct a sequence $x^{(i+1)} = (x^{(i)} + 1)^{1/3}$ starting from $x^{(0)} = 1$. This yields a sequence $x^{(1)} = 1.26, x^{(2)} = 1.31, x^{(3)} = 1.32, \ldots$, which converges to a solution.

The advantage of fixed-point iteration is that it can iterate in this simple manner on objects of any dimensionality, for example, on a vector of the polynomial coefficients. The cost of this procedure does not grow considerably with the number of polynomial coefficients. The shortcoming is that it does not always converge. For example, if we wrote the above equation as $x = x^3 - 1$ and implemented fixed-point iteration $x^{(i+1)} = (x^{(i)})^3 - 1$, we would obtain a sequence that diverges to $-\infty$ starting from $x^{(0)} = 1$.

We survey derivative-free optimization methods in Section 6, and we show how to enhance their convergence properties by using damping. We apply fixed-point iteration in two different contexts. One is to solve for parameters vectors in approximating functions and the other is to solve for quantities satisfying a system of equilibrium conditions. The latter version of fixed-point iteration is advocated in Maliar et al. (2011) and is called *iteration-on-allocation*.

Dynamic Programming Problems

Model (1)–(3) can be also written in the dynamic programming form:

$$V(k, \theta) = \max_{k', c, \ell} \left\{ u(c, \ell) + \beta E\left[V\left(k', \theta'\right)\right]\right\} \tag{8}$$

$$\text{s.t. } k' = (1-\delta)\,k + \theta f\,(k,\ell) - c, \tag{9}$$

$$\ln\theta' = \rho\ln\theta + \sigma\epsilon', \quad \epsilon' \sim \mathcal{N}\,(0,1)\,, \tag{10}$$

where $E\,[V\,(k',\theta')] \equiv E\,[V\,(k',\theta')\,|k,\theta]$ is an expectation of $V\,(k',\theta')$ conditional on state (k,θ). Here, and further on, primes on variables mean their future values.

An Example of a Global Projection-Style Dynamic Programming Algorithm

We can also use a projection-style algorithm to solve for a value function V.

(DPM): A global projection-style dynamic programming method.

Step 1. Choose functional form $\widehat{V}\,(\cdot, b^v)$ for representing V, where b^v is a coefficient vector.
Choose a grid $\{k_m, \theta_m\}_{m=1,\ldots,M}$ on which $\widehat{V}$ is constructed.
Step 2. Choose nodes, ϵ_j, and weights, $\omega_j, j = 1,\ldots,J$, for approximating integrals.
Compute next-period productivity $\theta'_{m,j} = \theta_m^{\rho}\exp(\epsilon_j)$ for all j, m.
Step 3. Solve for b^v that approximately satisfies

$$V\,(k_m,\theta_m; b^v) = \max_{\{k'_m, c_m, \ell_m\}} \left\{ u\,(c_m,\ell_m) + \beta\sum_{j=1}^{J}\left(\omega_j\left[V\left(k'_m,\theta'_{m,j}; b^v\right)\right]\right)\right\},$$
$$c_m = (1-\delta)\,k_m + \theta_m f\,(k_m,\ell_m) - k'_m.$$

Observe that this dynamic programming method also requires us to approximate multidimensional functions and integrals. Hence, our discussion about the curse of dimensionality and the need of nonproduct grids and nonproduct integration methods applies to dynamic programming algorithms as well. Furthermore, if a solution to the Bellman equation is interior, derivative-free optimization methods are also a suitable choice.

There are issues that are specific to dynamic programming problems, in addition to those issues that are common for dynamic programming and Euler equation methods. The first issue is that conventional value function iteration leads to systems of optimality conditions that are expensive to solve numerically. As an example, let us consider a version of model (8)–(10) with inelastic labor supply. We combine FOC $u'\,(c) = \beta E\,[V_1\,(k',\theta')]$ with budget constraint (10) to obtain

$$u'\left[(1-\delta)\,k + \theta f\,(k) - k'\right] = \beta E\left[V_1\left(k',\theta'\right)\right]. \tag{11}$$

We parameterize V_1 with the simplest possible first-degree polynomial $\widehat{V}_1\,(k,\theta; b) = b_1 + b_2 k + b_3\theta$. Then, (11) is

$$(1-\delta)\,k + \theta f\,(k) = k' + \left[\beta\sum_{j=1}^{J}\omega_j\left(b_1 + b_2 k' + b_3\theta^{\rho}\exp\left(\epsilon_j\right)\right)\right]^{-1/\gamma}, \tag{12}$$

where we assume $u(c) = \frac{c^{1-\gamma}-1}{1-\gamma}$ (and hence, $u'(c) = c^{-\gamma}$) for expository convenience. Solving (12) with respect to k' is expensive even in this simple case. We need to find a root numerically by computing V_1 in a large number of candidate points (k', θ'), as well as by approximating expectations for each value of (k', θ') considered. For high-dimensional problems, the cost of conventional value function iteration is prohibitive.

Two methods are proposed to simplify rootfinding to (12), one is an endogenous grid method of Carroll (2005) and the other is an envelope condition method of Maliar and Maliar (2013). The essence of Carroll's (2005) method is the following simple observation: it is easier to solve (12) with respect to k given k' than to solve it with respect to k' given k. Hence, Carroll (2005) fixes future endogenous state variable k' as grid points and treats current endogenous state variable k as an unknown.

The envelope condition method of Maliar and Maliar (2012a,b) builds on a different idea. Namely, it replaces backward-looking iteration based on a FOC by forward-looking iteration based on an envelope condition. For a version of (8)–(10) with inelastic labor supply, the envelope condition is $V_1(k, \theta) = u'(c)\left[1 - \delta + \theta f'(k)\right]$, which leads to an explicit formula for consumption

$$c = \left[\frac{V_1(k, \theta)}{1 - \delta + \theta f'(k)}\right]^{-1/\gamma}, \tag{13}$$

where we again assume $u(c) = \frac{c^{1-\gamma}-1}{1-\gamma}$. Since c follows directly from the envelope condition, and since k' follows directly from budget constraint (9), no rootfinding is needed at all in this example.

Our second issue is that the accuracy of Bellman equation approaches is typically lower than that of similar Euler equation approaches. This is because the object that is relevant for accuracy is the derivative of value function V_1 and not V itself (namely, V_1 enters optimality conditions (11) and (13) and determines the optimal quantities and not V). If we approximate V with a polynomial of degree n, we effectively approximate V_1 with a polynomial of degree $n - 1$, i.e., we effectively "lose" one polynomial degree after differentiation with the corresponding accuracy loss. To avoid this accuracy loss, Maliar and Maliar (2013) introduce variants of the endogenous grid and envelope condition methods that solve for derivatives of value function instead of or in addition to the value function itself. These variants produce highly accurate solutions.

Precomputation

Steps inside the main iterative cycle are repeated a large number of times. Computational expense can be reduced if some expensive steps are taken outside the main iterative cycle, i.e., precomputed.

As an example, consider a method of precomputation of integrals introduced in Judd et al. (2011d). Let us parameterize V by $\widehat{V}(k, \theta; b) = b_1 + b_2 k + b_3 \theta$. The key here is to observe that we can compute the expectation of $\widehat{V}$ up-front, without solving the model,

$$E\left[\widehat{V}(k',\theta';b)\right] = E\left[b_1 + b_2k' + b_3\theta'\right] = b_1 + b_2k' + b_3\widetilde{\theta},$$

where under assumption (10), we have $\widetilde{\theta}' \equiv \theta^{\rho}E\left[\exp\left(\epsilon'\right)\right]$. With this result, we rewrite Bellman equation (8) as

$$b_1 + b_2k + b_3\theta \approx \max_{\left\{k'_m, c_m, \ell_m\right\}} \left\{u\left(c_m, \ell_m\right) + \beta\left\{b_1 + b_2k' + b_3\widetilde{\theta}\right\}\right\}.$$

Effectively, this converts a stochastic Bellman equation into a deterministic one. The effect of uncertainty is fully captured by a precomputed value $\widetilde{\theta}'$. Judd et al. (2011d) also develop precomputation of integrals in the context of Euler equation methods.

Maliar and Maliar (2005b) and Maliar et al. (2011) introduce another kind of precomputation, one that constructs the solution manifolds outside the main iterative cycle. For example, consider (2) and (5),

$$c + k' = (1-\delta)\,k + \theta f\,(k,\ell)\,,$$
$$u_2\,(c,\ell) = u_1\,(c,\ell)\,\theta f_2\,(k,\ell)\,.$$

If k' is fixed, we have a system of two equations with two unknowns, c and ℓ. Solving this system once is not costly but solving it repeatedly in each grid point and integration node inside the main iterative cycle may have a considerable cost even in our simple example. We precompute the choice of c and ℓ to reduce this cost. Specifically, outside the main iterative cycle, we specify a grid of points for k, θ, k', and we find a solution for c and ℓ in each grid point. The resulting solution manifolds $\widetilde{C}\,(k,\theta,k')$ and $\widetilde{\mathcal{L}}\,(k,\theta,k')$ give us c and ℓ for each given triple (k,θ,k'). Inside the main iterative cycle, we use the precomputed manifolds to infer consumption and labor choices as if their closed-form solutions in terms of (k,θ,k') were available.

Finally, Maliar and Maliar (2001, 2003a) introduce a technique of imperfect aggregation that makes it possible to precompute aggregate decision rules in certain classes of heterogeneous-agent economies. The precomputation methods are surveyed in Section 8.

Perturbation Methods

Perturbation methods approximate a solution in just one point—a deterministic steady state—using Taylor expansions of the optimality conditions. The costs of perturbation methods do not increase rapidly with the dimensionality of the problem. However, the accuracy of a perturbation solution may deteriorate dramatically away from the steady state in which such a solution is computed.

One technique that can increase the accuracy of perturbation methods is a change of variables proposed by Judd (2003). Among many different Taylor expansions that are locally equivalent, we must choose the one that is most accurate globally. For example, consider two approximations for the capital decision function,

$$k' = b_1 + b_2 k + b_3 \theta \quad \text{and} \quad \widetilde{k}' = \widetilde{b}_1 + \widetilde{b}_2 \ln(k) + \widetilde{b}_3 \ln(\theta).$$

Let $\left(\overline{k}, \overline{\theta}\right)$ be a steady state. If we set $b_1 = \widetilde{b}_1/\overline{k}$ and $b_2 = \widetilde{b}_2/\overline{\theta}$, then k' and $\widetilde{k}'$ are locally equivalent in a sense that they have the same values and the same derivatives in the steady state. Hence, we can compare their accuracy away from the steady state and choose the one that has a higher overall accuracy.

The other technique for increasing the accuracy of perturbation methods is a hybrid of local and global solutions developed in Maliar et al. (2013). The idea is to combine local solutions produced by a perturbation method with global solutions that are constructed to satisfy the model's equations exactly. For example, assume that a first-order perturbation method delivers us a solution $k' = b_1^k + b_2^k k + b_3^k \theta$ and $c = b_1^c + b_2^c k + b_3^c \theta$ in a model with inelastic labor supply. We keep the perturbation solution for k' but replace c with a new function for consumption $\widetilde{c}$ that is constructed to satisfy (9) exactly

$$\widetilde{c} \equiv (1-\delta) k + \theta f(k) - \left(b_1^k + b_2^k k + b_3^k \theta\right).$$

The obtained $k', \widetilde{c}$ are an example of a hybrid solution. This particular hybrid solution produces a zero residual in the budget constraint unlike the original perturbation solution that produces nonzero residuals in all the model's equations. The techniques for increasing the accuracy of perturbation methods are surveyed in Section 9.

Parallel Computing

In the past decades, the speed of computers was steadily growing. However, this process has a natural limit (because the speed of electricity along the conducting material is limited and because the thickness and length of the conducting material are limited). The recent progress in solving computationally intense problems is related to parallel computation. We split a large problem into smaller subproblems, allocate the subproblems among many workers (computers), and solve them simultaneously. Each worker does not have much power but all together they can form a supercomputer. However, to benefit from this new computation technology, we need to design solution methods in a manner that is suitable for parallelizing. We also need hardware and software that support parallel computation. These issues are discussed in Section 10.

Methodology of the Numerical Analysis and Computational Details

Our solution procedure has two stages. In Stage 1, a method attempts to compute a numerical solution to a model. Provided that it succeeds, we proceed to Stage 2, in which we subject a candidate solution to a tight accuracy check. We specifically construct a set of points $\{k_i, \theta_i\}_{i=1,\ldots,I}$ that covers an area in which we want the solution to be accurate, and we compute unit-free residuals in the model's equations:

$$\mathcal{R}^{BC}(k_i,\theta_i) = \frac{(1-\delta)\,k_i + \theta_i f(k_i,\ell_i)}{c_i + k_i'} - 1, \tag{14}$$

$$\mathcal{R}^{EE}(k_i,\theta_i) = \beta E\left\{\frac{u_1\left(c_i',\ell_i'\right)}{u_1(c_i,\ell_i)}\left[1-\delta+\theta_i' f_1\left(k_i',\ell_i'\right)\right]\right\} - 1, \tag{15}$$

$$\mathcal{R}^{MUL}(k_i,\theta_i) = \frac{u_1(c_i,\ell_i)\,\theta_i f_2(k_i,\ell_i)}{u_2(c_i,\ell_i)} - 1, \tag{16}$$

where $\mathcal{R}^{BC}$, $\mathcal{R}^{EE}$, and $\mathcal{R}^{MUL}$ are the residuals in budget constraint (9), Euler equation (4), and FOC for the marginal utility of leisure (5). In the exact solution, such residuals are zero, so we can judge the quality of approximation by how far these residuals are away from zero.

In most experiments, we evaluate residuals on stochastic simulation (we use 10,200 observations and we discard the first 200 observations to eliminate the effect of initial conditions). This style of accuracy checks is introduced in Jin and Judd (2002). In some experiments, we evaluate accuracy on deterministic sets of points that cover a given area in the state space such as a hypersphere or hypercube; this kind of accuracy check is proposed by Judd (1992). We must emphasize that we never evaluate residuals on points used for computing a solution in Stage 1 (in particular, for some methods the residuals in the grid points are zeros by construction) but we do so on a new set of points constructed for Stage 2.

If either a solution method fails to converge in Stage 1 or the quality of a candidate solution in Stage 2 is economically inacceptable, we modify the algorithm's design (i.e., the number and placement of grid points, approximating functions, integration method, fitting method, etc.) and we repeat the computations until a satisfactory solution is produced.

Parameterizations of the Model

In Sections (4)–(10), we report the results of numerical experiments. In those experiments, we parameterize model (1)–(3) with elastic labor supply by

$$u(c,\ell) = \frac{c^{1-\gamma}-1}{1-\gamma} + B\frac{(1-\ell)^{1-\mu}-1}{1-\mu}, \quad \text{and} \quad \theta f(k,\ell) = \theta k^{\alpha}\ell^{1-\alpha}, \tag{17}$$

where parameters $\gamma,\mu,B>0$. We use $\alpha=0.36$, $\beta=0.99$, $\delta=0.025$, and we use $\rho=0.95$ and $\sigma=0.01$ to parameterize the stochastic process (3). For the utility parameters, our benchmark choice is $\gamma=1$ and $\mu=1$ but we also explore other values of γ and μ.

In the model with inelastic labor supply, we assume $B=0$ and $\ell=1$, i.e.,

$$u(c) = \frac{c^{1-\gamma}-1}{1-\gamma}, \quad \text{and} \quad \theta f(k) = \theta k^{\alpha}. \tag{18}$$

Finally, we also study the latter model under the assumptions of full depreciation of capital, $\delta = 1$, and the logarithmic utility function, $\gamma = 1$; this version of the model has a closed-form solution: $k_{t+1} = \beta\alpha\theta_t k_t^{\alpha}$ and $c_t = (1 - \beta\alpha)\,\theta_t k_t^{\alpha}$.

Reported Numerical Results

For each computational experiment, we report two accuracy measures, namely, the average and maximum absolute residuals across both the optimality conditions and I test points. The residuals are represented in log 10 units, for example, $\mathcal{R}^{BC}(k_i, \theta_i) = -2$

Table 2 The acronyms used.

Acronym	Meaning
CPU	Central processing unit
CGA	Cluster grid algorithm
DP	Dynamic programming
DVF	Derivative of value function
ECM	Envelope condition method
EDS	ε-distiguishable set
EGM	Endogenous grid method
FOC	First-order condition
FPI	Fixed-point iteration
GSSA	Generalized stochastic simulation algorithm
GPU	Graphics processing unit
HPC	High-performance computing
HTC	High-throughput computing
HYB	Hybrid perturbation-based method
LAD	Least-absolute deviations
LS	Least squares
MC	Monte Carlo integration
MPI	Message passing interface
MRGAL	Monomial rule Galerkin method
OpenMP	Open memory programming
PEA	Parameterized expectations algorithm
PER	Perturbation
PS	Principal component
RLAD	Regularaized least-absolute deviations
RLS	Regularized least squares
SMOL	Smolyak method
SSA	Stochastic simulation algorithm
SVD	Singular value decomposition
VF	Value function
VFI	Value function iteration
XSEDE	eXtreme Science and Engineering Discovery Environment

means that a residual in the budget constraint is $10^{-2} = 1\%$, and $\mathcal{R}^{BC}(k_i, \theta_i) = -4.5$ means such a residual is $10^{-4.5} = 0.00316\%$.

As a measure of computational expense, we report the running time. Unless specified separately, we use MATLAB, version 7.6.0.324 (R2008a) and a desktop computer with a Quad processor Intel® Core™ i7 CPU920 @2.67 GHz, RAM 6,00 GB, and Windows Vista 64 bit.

Acronyms Used

We summarize all the acronyms used in the chapter in Table 2.

4. NONPRODUCT APPROACHES TO REPRESENTING, APPROXIMATING, AND INTERPOLATING FUNCTIONS

We survey nonproduct approaches to representing, approximating, and interpolating functions that can mitigate the curse of dimensionality. Let us consider a canonical approximation problem. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a smooth function, and let $\widehat{f}(\cdot; b)$ be a parametric function of the form

$$\widehat{f}(x; b) = \sum_{i=1}^{I} b_i \Psi_i(x), \tag{19}$$

where $\Psi_i : \mathbb{R}^d \to \mathbb{R}$ is a basis function, and $b \equiv (b_1, \ldots, b_I)$ is a parameters vector. We aim to find b such that $\widehat{f}(\cdot; b) \approx f$ within a given area of $\mathbb{R}^d$.

We construct a grid of $M \geq I$ points $\{x_1, \ldots, x_M\}$ within a given area of $\mathbb{R}^d$ and compute the residuals, $\varepsilon \equiv \{\varepsilon_1, \ldots, \varepsilon_M\}$, defined as a difference between the true function, f, and its approximation, $\widehat{f}(\cdot; b)$,

$$\begin{bmatrix} \varepsilon_1 \\ \cdots \\ \varepsilon_M \end{bmatrix} = \begin{bmatrix} f(x_1) \\ \cdots \\ f(x_M) \end{bmatrix} - \begin{bmatrix} \Psi_1(x_1) & \cdots & \Psi_I(x_1) \\ \cdots & \ddots & \cdots \\ \Psi_1(x_M) & \cdots & \Psi_I(x_M) \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ \cdots \\ b_I \end{bmatrix}. \tag{20}$$

Our objective is to find b to minimize the residuals according to some norm.

An approximation method includes the following three steps:

(i) Construction of basis functions $\{\Psi_1, \ldots, \Psi_I\}$ for (19).

(ii) Selection of grid points $\{x_1, \ldots, x_M\}$ for (20).

(iii) Identification of parameters vector b that ensures that $\widehat{f}(\cdot; b) \approx f$ within a given area of $\mathbb{R}^d$.

We consider two types of approximation methods, a collocation method and a weighted residuals method. Collocation is the case when $M = I$, i.e., the number of grid points is the same as the number of basis functions. If the matrix of basis functions in the right side of (20) has full rank, we can find b that makes the residuals be equal to zero in all grid points (we have a system of M linear equations with M unknowns that admits a unique solution for b). Approximation $\widehat{f}(\cdot; b)$ coincides with true function f in all grid points, i.e., $\widehat{f}(x_m; b) = f(x_m)$ for all $x_m \in \{x_1, \ldots, x_M\}$. In this case, $\widehat{f}(\cdot; b)$ interpolates f off the grid.

A weighted residuals method is the case when $M > I$. Since there are more equations than unknowns, parameters vector b is overdetermined, and the system of equation (20) has no solution in a sense that we cannot make all the residuals be zeros in all grid points. The best we can do is to find b that minimizes the residuals according to some norm, e.g., a least-squares norm in which all the residuals have equal weights, i.e., $\min_b \sum_{m=1}^{M} \varepsilon_m^2$. This is like the usual least-squares regression in which we have many data points for identifying relatively few regression coefficients. Since approximation $\widehat{f}(\cdot; b)$ does not coincide with true function f in all grid points, we say that $\widehat{f}(\cdot; b)$ approximates f off the grid (instead of saying "interpolates").

In the remainder of the section, we consider three methods that construct the approximation of type (19) in a way which is tractable for high-dimensional problems. These are a Smolyak method, a generalized stochastic simulation method, and an ε-distinguishable set method. The first one is a collocation method, while the latter two are weighted residuals methods.

4.1 Smolyak's (1963) Sparse Grid Method

In a seminal work, Smolyak (1963) introduces a numerical technique for integrating, representing, and interpolating smooth functions on multidimensional hypercubes. The Smolyak method constructs multidimensional grid points and basis functions using non-product rules. The complexity of the Smolyak method grows polynomially with the dimensionality of the hypercube, i.e., the method is not subject to the curse of dimensionality (at least in a space of smooth functions).

The Smolyak grid was incorporated into numerical methods for solving dynamic economic models by Krueger and Kubler (2004); see also Malin et al. (2011). Furthermore, Winschel and Krätzig (2010) apply the Smolyak method for developing state-space filters that are tractable in problems with large dimensionality. Fernández-Villaverde et al. (2012) use the Smolyak method to solve a new Keynesian model. Finally, Judd et al. (2013) generalize the Smolyak method in various dimensions to improve its performance in economic applications.

We present the Smolyak method in two different ways. In Section 4.1.1, we derive the Smolyak interpolation formula using intuitive arguments which allows us to illustrate how the method works. In particular, we do not rely on the conventional interpolation formula but construct Smolyak polynomials using a Smolyak rule; also, we do not rely on the conventional formula for the Smolyak coefficients but obtain such coefficients from an inverse problem; in this presentation, we follow Judd et al. (2013). Then, in Section 4.1.2, we give a more conventional exposition of the Smolyak method in line with the analysis of Barthelmann et al. (2000), Krueger and Kubler (2004), and Malin et al. (2011). Finally, in Section 4.1.3, we discuss the relation between the analysis in Judd et al. (2013) and the conventional Smolyak interpolation in the literature.

4.1.1 How Does the Smolyak Method Work?

We introduce the Smolyak method in the context of a simple two-dimensional example. We show how the Smolyak method implements the three steps discussed in the beginning of the section, i.e., how it (i) constructs grid points, (ii) constructs a polynomial function, and (iii) identifies the polynomial coefficients for interpolation.

Smolyak Grid

The idea of the Smolyak method is appealing. Among all elements produced by a tensor-product rule, some elements are more important for the quality of approximation than the others. The Smolyak method orders all the tensor-product elements by their potential importance for the quality of approximation and selects the most important ones. A parameter, called *approximation level,* controls how many tensor-product elements are included in a specific Smolyak approximation. By increasing the approximation level, we add new elements and improve the quality of approximation.

Consider the following five unidimensional grid points $\left\{-1, \frac{-1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}, 1\right\}$.[10] If we use a tensor-product rule to construct two-dimensional grid points, we obtain 25 such grid points $\left\{(-1,-1), \left(-1, \frac{-1}{\sqrt{2}}\right), \ldots, (1,1)\right\}$.

The Smolyak method constructs grid points in a different manner. First, we use the unidimensional grid points to construct the following sequence of sets:

$i = 1 : S_1 = \{0\};$
$i = 2 : S_2 = \{0, -1, 1\};$
$i = 3 : S_3 = \left\{-1, \frac{-1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}, 1\right\}.$

The sets are nested: each subsequent set contains all the elements of the previous set.

Second, we construct all possible two-dimensional tensor products using elements from the nested sets. These products are shown in Table 3 for $i = 1, 2, 3$.

Finally, we select elements from those cells in Table 3 that satisfy a Smolyak rule: the index of a column i_1 plus the index of a row i_2 is smaller than or equal to $d + \mu$, i.e.,

$$i_1 + i_2 \leq d + \mu, \tag{21}$$

where $\mu \in \{0, 1, 2, \ldots\}$ is the approximation level parameter and d is the dimensionality of the problem (in our case, $d = 2$).

[10] These points are extrema of a Chebyshev polynomial function but this fact is not essential for the Smolyak construction; we can use other unidimensional grid points instead, for example, uniformly spaced grid points.

Table 3 Tensor products of unidimensional nested grid points for the two-dimensional case.

		$i_2=1$	$i_2=2$	$i_2=3$
	$S_{i_1}\backslash S_{i_2}$	0	$0,-1,1$	$0,-1,1,\frac{-1}{\sqrt{2}},\frac{1}{\sqrt{2}}$
$i_1=1$	0	$(0,0)$	$(0,0),(0,-1),(0,1)$	$(0,0),(0,-1),(0,1),(0,\frac{-1}{\sqrt{2}}),(0,\frac{1}{\sqrt{2}})$
$i_1=2$	0	$(0,0)$	$(0,0),(0,-1),(0,1)$	$(0,0),(0,-1),(0,1),(0,\frac{-1}{\sqrt{2}}),(0,\frac{1}{\sqrt{2}})$
	-1	$(-1,0)$	$(-1,0),(-1,-1),(-1,1)$	$(-1,0),(-1,-1),(-1,1),(-1,\frac{-1}{\sqrt{2}}),(-1,\frac{1}{\sqrt{2}})$
	1	$(1,0)$	$(1,0),(1,-1),(1,1)$	$(1,0),(1,-1),(1,1),(1,\frac{-1}{\sqrt{2}}),(1,\frac{1}{\sqrt{2}})$
$i_1=3$	0	$(0,0)$	$(0,0),(0,-1),(0,1)$	$(0,0),(0,-1),(0,1),(0,\frac{-1}{\sqrt{2}}),(0,\frac{1}{\sqrt{2}})$
	-1	$(-1,0)$	$(-1,0),(-1,-1),(-1,1)$	$(-1,0),(-1,-1),(-1,1),(-1,\frac{-1}{\sqrt{2}}),(-1,\frac{1}{\sqrt{2}})$
	1	$(1,0)$	$(1,0),(1,-1),(1,1)$	$(1,0),(1,-1),(1,1),(1,\frac{-1}{\sqrt{2}}),(1,\frac{1}{\sqrt{2}})$
	$\frac{-1}{\sqrt{2}}$	$(\frac{-1}{\sqrt{2}},0)$	$(\frac{-1}{\sqrt{2}},0),(\frac{-1}{\sqrt{2}},-1),(\frac{-1}{\sqrt{2}},1)$	$(\frac{-1}{\sqrt{2}},0),(\frac{-1}{\sqrt{2}},-1),(\frac{-1}{\sqrt{2}},1),(\frac{-1}{\sqrt{2}},\frac{-1}{\sqrt{2}}),(\frac{-1}{\sqrt{2}},\frac{1}{\sqrt{2}})$
	$\frac{1}{\sqrt{2}}$	$(\frac{1}{\sqrt{2}},0)$	$(\frac{1}{\sqrt{2}},0),(\frac{1}{\sqrt{2}},-1),(\frac{1}{\sqrt{2}},1)$	$(\frac{1}{\sqrt{2}},0),(\frac{1}{\sqrt{2}},-1),(\frac{1}{\sqrt{2}},1),(\frac{1}{\sqrt{2}},\frac{-1}{\sqrt{2}}),(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}})$

Let $\mathcal{H}^{d,\mu}$ denote the Smolyak grid for a problem with dimensionality d and approximation level μ. We use the Smolyak rule (21) to construct two-dimensional grid points for our example.

- If $\mu=0$, then (21) implies $i_1+i_2\leq 2$. The only case that satisfies this restriction is when $i_1=1$ and $i_2=1$, so that the Smolyak grid is a single point

$$\mathcal{H}^{2,0}=\{(0,0)\}. \tag{22}$$

- If $\mu=1$, then $i_1+i_2\leq 3$. The three cases that satisfy this restriction are (a) $i_1=1$, $i_2=1$; (b) $i_1=1, i_2=2$; (c) $i_1=2, i_2=1$. The nonrepeating five Smolyak points are

$$\mathcal{H}^{2,1}=\{(0,0),(-1,0),(1,0),(0,-1),(0,1)\}. \tag{23}$$

- If $\mu=2$, then $i_1+i_2\leq 4$. The six cases satisfying this restriction are (a) $i_1=1, i_2=1$; (b) $i_1=1, i_2=2$; (c) $i_1=2, i_2=1$; (d) $i_1=1, i_2=3$; (e) $i_1=2, i_2=2$; (f) $i_1=3, i_2=1$. The 13 Smolyak points are

$$\mathcal{H}^{2,2}=\Bigg\{(-1,1),(0,1),(1,1),(-1,0),(0,0),(1,0),(-1,-1),(0,-1),\\(1,-1),\left(\frac{-1}{\sqrt{2}},0\right),\left(\frac{1}{\sqrt{2}},0\right),\left(0,\frac{-1}{\sqrt{2}}\right),\left(0,\frac{1}{\sqrt{2}}\right)\Bigg\}. \tag{24}$$

Observe that the elements that satisfy (21) are situated on the diagonals that expand from the upper left corner of the table toward the lower right corner. The Smolyak grid points under $\mu=0,1,2,3$ are illustrated in Figure 2; for comparison, we also show a tensor-product grid of 9×9 points. The number of points in the Smolyak sparse grid increases polynomially with dimensionality d. In particular, for $\mu=1$ and

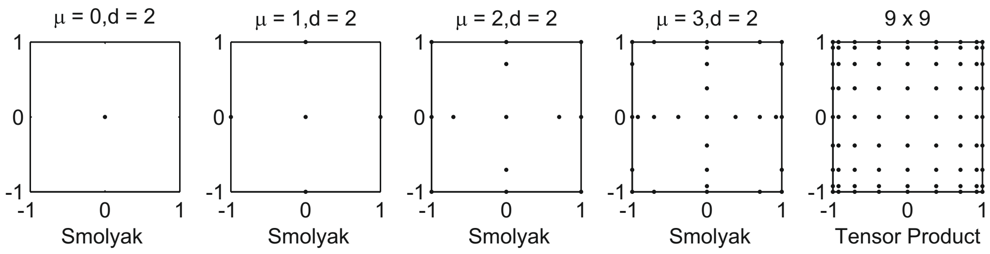


Figure 2 Smolyak grids versus tensor-product grid.

$\mu = 2$, the number of points in the Smolyak grid is $1 + 2d$ and $1 + 4d + 4d\,(d-1)$, respectively, which means that the size of the Smolyak grid grows, respectively, linearly and quadratically with d; see Table 4 for a comparison of the number of points in the Smolyak and tensor-product grids. A relatively small number of Smolyak grid points contrasts sharply with a large number of tensor-product grid points which grows exponentially with the dimensionality of the problem (curse of dimensionality).

Smolyak Polynomials

The construction of Smolyak polynomials is parallel to the construction of the Smolyak grid. We build the Smolyak multidimensional polynomial function using a family of unidimensional Chebyshev basis functions composed of five terms, $\left\{1, x, 2x^2-1, 4x^3-3x, 8x^4-8x^2+1\right\}$.[11] Again, we can construct two-dimensional basis functions using a tensor product of the unidimensional basis functions, which would give us 25 of such basis functions.

However, under the Smolyak method, we first construct a sequence of nested sets using the given unidimensional basis functions:

Table 4 Number of grid points: tensor-product grid with five points in each dimension versus Smolyak grid.

		Smolyak grid		
d	**Tensor-product grid 5^d**	$\mu = 1$	$\mu = 2$	$\mu = 3$
1	5	3	5	9
2	25	5	13	29
10	9,765,625	21	221	1581
20	95,367,431,640,625	41	841	11561

Notes: d is the number of state variables; μ is the approximation level.

[11] Again, the assumption of Chebyshev basis functions is convenient but not essential for the Smolyak construction.

$$i = 1 : S_1 = \{1\};$$
$$i = 2 : S_2 = \left\{1, x, 2x^2 - 1\right\};$$
$$i = 3 : S_3 = \left\{1, x, 2x^2 - 1, 4x^3 - 3x, 8x^4 - 8x^2 + 1\right\}.$$

Second, we list all possible two-dimensional tensor products of the unidimensional Smolyak sequences (we denote dimensions by x and y); see Table 5.

Finally, we select the elements from the table which are implied by the Smolyak rule (21). Let $\mathcal{P}^{d,\mu}$ denote the Smolyak basis functions for a problem with dimensionality d and approximation level μ. For the two-dimensional case, we have:

- If $\mu = 0$, then $i_1 + i_2 \leq 2$. The only case that satisfies this restriction is when $i_1 = 1$ and $i_2 = 1$, and thus there is just one Smolyak basis function

$$\mathcal{P}^{2,0}(x, y) = \{1\}. \tag{25}$$

- If $\mu = 1$, then $i_1 + i_2 \leq 3$. There are three combinations of i_1 and i_2 that satisfy this restriction, and the corresponding five Smolyak basis functions are

$$\mathcal{P}^{2,1}(x, y) = \left\{1, x, (2x^2 - 1), y, (2y^2 - 1)\right\}. \tag{26}$$

- If $\mu = 2$, then $i_1 + i_2 \leq 4$. There are six combinations of i_1 and i_2 that satisfy this restriction, and the 13 Smolyak basis functions are

$$\begin{aligned}\mathcal{P}^{2,2}(x, y) = \big\{&1, x, (2x^2 - 1), y, xy, (2x^2 - 1)y, (2y^2 - 1), x(2y^2 - 1),\\ &(2x^2 - 1)(2y^2 - 1), 4x^3 - 3x^2, 8x^4 - 8x^2 + 1,\\ &4y^3 - 3y^2, 8y^4 - 8y^2 + 1\big\}.\end{aligned} \tag{27}$$

By construction, in this example, the number of basis functions in Smolyak polynomial is equal to the number of points in the Smolyak grid; compare Tables 3 and 5. The same is true for a Smolyak grid $\mathcal{H}^{d,\mu}$ and Smolyak polynomial $\mathcal{P}^{d,\mu}$ under any $d \geq 1$ and $\mu \geq 0$.

Table 5 Tensor products of unidimensional nested polynomial basis for the two-dimensional case.

		$i_2 = 1$	$i_2 = 2$	$i_2 = 3$
	$S_{i_1} \backslash S_{i_2}$	1	$1, y, 2y^2 - 1$	$1, y, 2y^2 - 1, 4y^3 - 3y, 8y^4 - 8y^2 + 1$
$i_1 = 1$	1	1	$1, y, 2y^2 - 1$	$1, y, 2y^2 - 1, 4y^3 - 3y, 8y^4 - 8y^2 + 1$
	1	1	$1, y, 2y^2 - 1,$	$1, y, 2y^2 - 1, 4y^3 - 3y, 8y^4 - 8y^2 + 1$
$i_1 = 2$	x	x	$x, xy, x\left(2y^2 - 1\right),$	$x, xy, x\left(2y^2 - 1\right), x\left(4y^3 - 3y\right), \ldots$
	$2x^2 - 1$	$2x^2 - 1$	$2x^2 - 1, (2x^2 - 1)y, (2x^2 - 1)(2y^2 - 1)$	$2x^2 - 1, (2x^2 - 1)y, (2x^2 - 1)(2y^2 - 1), \ldots$
	1	1	$1, y, 2y^2 - 1$	$1, y, 2y^2 - 1, 4y^3 - 3y, 8y^4 - 8y^2 + 1$
	x	x	$x, xy, x\left(2y^2 - 1\right)$	$x, xy, x\left(2y^2 - 1\right), x\left(4y^3 - 3y\right), \ldots$
$i_1 = 3$	$2x^2 - 1$	$2x^2 - 1$	$2x^2 - 1, (2x^2 - 1)y, (2x^2 - 1)(2y^2 - 1)$	$\left(2x^2 - 1\right), \left(2x^2 - 1\right)y, \ldots$
	$4x^3 - 3x$	$4x^3 - 3x$	$\left(4x^3 - 3x\right), \left(4x^3 - 3x\right)y, \ldots$	$\left(4x^3 - 3x\right), \left(4x^3 - 3x\right)y, \ldots$
	$8x^4 - 8x^2 + 1$	$8x^4 - 8x^2 + 1$	$\left(8x^4 - 8x^2 + 1\right), \left(8x^4 - 8x^2 + 1\right)y, \ldots$	$\left(8x^4 - 8x^2 + 1\right), \left(8x^4 - 8x^2 + 1\right)y, \ldots$

Smolyak Interpolation Under $d = 2$ and $\mu = 1$

We now show Smolyak interpolation. Suppose we must interpolate a two-dimensional function f on a square $[-1,1]^2$ (i.e., we must construct a function $\widehat{f}$ that matches f exactly in a finite set of points). We consider $\mu = 1$, in which case the interpolating function is composed of family $\mathcal{P}^{2,1}$ of five Smolyak basis functions distinguished in (26),

$$\widehat{f}(x,y;b) \equiv b_{11}1 + b_{21}x + b_{31}\left(2x^2-1\right) + b_{12}y + b_{13}\left(2y^2-1\right), \tag{28}$$

where $b \equiv (b_{11}, b_{21}, b_{31}, b_{12}, b_{13})$ is a vector of five unknown coefficients.

The unknown coefficients must be chosen to match the values of $f(x,y)$ in five Smolyak grid points $\mathcal{H}^{2,1} = \{(0,0), (-1,0), (1,0), (0,-1), (0,1)\}$ distinguished in (23). This yields a system of linear equations $\mathcal{B}b = w$, where

$$\mathcal{B} = \begin{bmatrix} 1 & 0 & -1 & 0 & -1 \\ 1 & -1 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 & -1 \\ 1 & 0 & -1 & -1 & 1 \\ 1 & 0 & -1 & 1 & 1 \end{bmatrix}; \quad b = \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \\ b_{12} \\ b_{13} \end{bmatrix} \quad w \equiv \begin{bmatrix} f(0,0) \\ f(-1,0) \\ f(1,0) \\ f(0,-1) \\ f(0,1) \end{bmatrix}. \tag{29}$$

The solution to this system is given by $b = \mathcal{B}^{-1}w$,

$$\begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \\ b_{12} \\ b_{13} \end{bmatrix} = \mathcal{B}^{-1}w = \begin{bmatrix} 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 \\ -\frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & 0 & 0 & \frac{1}{4} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} f(0,0) \\ f(-1,0) \\ f(1,0) \\ f(0,-1) \\ f(0,1) \end{bmatrix} = \begin{bmatrix} \frac{f(-1,0)+f(1,0)+f(0,-1)+f(0,1)}{4} \\ \frac{-f(-1,0)+f(1,0)}{2} \\ -\frac{f(0,0)}{2} + \frac{f(-1,0)+f(1,0)}{4} \\ \frac{-f(0,-1)+f(0,1)}{2} \\ -\frac{f(0,0)}{2} + \frac{f(0,-1)+f(0,1)}{4} \end{bmatrix}. \tag{30}$$

Substituting b into the Smolyak polynomial function (28) gives us $\widehat{f}$ that can be used to interpolate f in any point of $[-1,1]^2$.

As we see, Smolyak interpolation works similarly to conventional tensor-product interpolation described in Section 2 and used in, e.g., Judd (1992). The key feature of the Smolyak method is that it uses fewer elements than does a tensor-product method. In our two-dimensional case, the difference is not so large, but in high-dimensional problems, the number of Smolyak elements is just an infinitesimally small fraction of the tensor-product elements, as Table 4 shows.

4.1.2 The Automated Smolyak Method

Our construction of Smolyak interpolation has been done manually. Tables 3 and 5 contain, respectively, repeated grid points and repeated polynomial terms. However, when constructing the interpolation function (28), we list points and basis functions in a way that avoids repetitions.

In applications, the Smolyak construction cannot rely on a visual inspection of the tables and must be automated. Smolyak (1963) does not propose an algorithmic procedure that automates the construction of multidimensional grids and basis functions. He defines these objects mathematically and provides accuracy bounds for some classes of periodic functions.

An algorithm for constructing Smolyak elements was proposed in the subsequent literature. The conventional procedure for automating the Smolyak method is based on an interpolation formula developed in Wasilkowski and Woźniakowski (1999, Lemma 1) and Delvos (1982, Theorem 1). We provide this formula below and illustrate its use by way of example.

A Formula for Smolyak Interpolation

The following formula is used to interpolate $f(x_1, \ldots, x_d)$ defined on a hypercube $[-1, 1]^d$:

$$\widehat{f}(x_1, \ldots, x_d; b) = \sum_{\max(d,\mu+1) \leq |i| \leq d+\mu} (-1)^{d+\mu-|i|} \binom{d-1}{d+\mu-|i|} p^{|i|}(x_1, \ldots, x_d), \quad (31)$$

where $|i| \equiv i_1 + \cdots + i_d$, $p^{|i|}(x_1, \ldots, x_d)$ is the sum of tensor products of unidimensional basis functions $p^{i_1,\ldots,i_d}(x_1, \ldots, x_d)$ whose indices satisfy $i_1 + \cdots + i_d = |i|$ and

$$p^{i_1,\ldots,i_d}(x_1, \ldots, x_d) = \sum_{\ell_1=1}^{m(i_1)} \cdots \sum_{\ell_d=1}^{m(i_d)} b_{\ell_1 \cdots \ell_d} \psi_{\ell_1}(x_1) \cdots \psi_{\ell_d}(x_d), \quad (32)$$

where $m(i_1), \ldots, m(i_d)$ is the number of elements in unidimensional nested sets $i_1, \ldots, i_d$, respectively; $\psi_{\ell_1}(x_1), \ldots, \psi_{\ell_d}(x_d)$ are unidimensional basis functions in dimensions $1, \ldots, d$, respectively; $\ell_d = 1, \ldots, m(i_d)$, and $b_{\ell_1 \cdots \ell_d}$ are the polynomial coefficients given by

$$b_{\ell_1 \cdots \ell_d} = \frac{2^d}{(m(i_1)-1) \cdots (m(i_d)-1)} \frac{1}{c_{\ell_1} \cdots c_{\ell_d}} \times \sum_{j_1=1}^{m(i_1)} \cdots \sum_{j_d=1}^{m(i_d)} \frac{\psi_{\ell_1}(\zeta_{j_1}) \cdots \psi_{\ell_d}(\zeta_{j_d}) \cdot f(\zeta_{j_1}, \ldots, \zeta_{j_d})}{c_{j_1} \cdots c_{j_d}}, \quad (33)$$

where $\zeta_{j_1}, \ldots, \zeta_{j_d}$ are grid points in the dimensions $j_1, \ldots, j_d$, respectively; $c_j = 2$ for $j = 1$ and $j = m(i_d)$; $c_j = 1$ for $j = 2, \ldots, m(i_d) - 1$. If along any dimension d, we have $m(i_d) = 1$, this dimension is dropped from the computation of the coefficient, i.e., $m(i_d) - 1$ is set to 1 and $c_{j_d} = c_1$ is set to 1.

Smolyak Interpolation Under $d = 2$ and $\mu = 1$ Revisited

We now illustrate the application of the interpolation formula (31) in the context of an example with $d = 2$ and $\mu = 1$ studied in Section 4.1.1. For the case of $\mu = 1$, we have $i_1 + i_2 \leq d + \mu = 3$. This is satisfied in three cases: (a) $i_1 = i_2 = 1$; (b) $i_1 = 1, i_2 = 2$; (c) $i_1 = 2, i_2 = 1$. From (32), we obtain

$$\text{(a) } p^{1,1} = \sum_{\ell_1=1}^{m(1)} \sum_{\ell_2=1}^{m(1)} b_{\ell_1 \ell_2} \psi_{\ell_1}(x) \psi_{\ell_2}(y) = b_{11},$$

$$\text{(b) } p^{1,2} = \sum_{\ell_1=1}^{m(1)} \sum_{\ell_2=1}^{m(2)} b_{\ell_1 \ell_2} \psi_{\ell_1}(x) \psi_{\ell_2}(y) = b_{11} + b_{12}\psi_2(y) + b_{13}\psi_3(y), \tag{34}$$

$$\text{(c) } p^{2,1} = \sum_{\ell_1=1}^{m(2)} \sum_{\ell_2=1}^{m(1)} b_{\ell_1 \ell_2} \psi_{\ell_1}(x) \psi_{\ell_2}(y) = b_{11} + b_{21}\psi_2(x) + b_{31}\psi_3(x). \tag{35}$$

Collecting the elements p^{i_1,i_2} with the same sum of i_1 and i_2, i.e., $|i| \equiv i_1 + i_2$, we obtain

$$p^{|2|} \equiv p^{1,1}, \tag{36}$$

$$p^{|3|} \equiv p^{2,1} + p^{1,2}. \tag{37}$$

The Smolyak polynomial function (31) for the case $d = 2$ and $\mu = 1$ is given by

$$\begin{aligned}
\widehat{f}(x, y; b) &= \sum_{\max(d,\mu+1) \leq |i| \leq d+\mu} (-1)^{d+\mu-|i|} \binom{d-1}{d+\mu-|i|} p^{|i|} \\
&= \sum_{2 \leq |i| \leq 3} (-1)^{3-|i|} \binom{1}{3-|i|} p^{|i|} = \sum_{2 \leq |i| \leq 3} (-1)^{3-|i|} \frac{1}{(3-|i|)!} p^{|i|} \\
&= (-1) \times p^{|2|} + 1 \times p^{|3|} = (-1) \times p^{1,1} + 1 \times (p^{2,1} + p^{1,2}) \\
&= -b_{11} + b_{11} + b_{21}\psi_2(x) + b_{31}\psi_3(x) + b_{11} + b_{12}\psi_2(y) + b_{13}\psi_3(y) \\
&= b_{11} + b_{21}\psi_2(x) + b_{31}\psi_3(x) + b_{12}\psi_2(y) + b_{13}\psi_3(y).
\end{aligned} \tag{38}$$

Formula (38) coincides with (28) if instead of ψ's in (38), we substitute the corresponding Chebyshev unidimensional basis functions.

Let us now compute the coefficients using (33). Coefficient b_{21} appears in $p^{2,1}$ with $m(i_1) = m(2) = 3$ and $m(i_2) = m(1) = 1$. The Smolyak nested sets of grid points are $S_1 = \{0\}$ and $S_2 = \{0, -1, 1\}$, i.e., the first three grid points are $\zeta_1 = 0, \zeta_2 = -1, \zeta_3 = 1$.

Condition (33) implies

$$\begin{aligned}
b_{21} &= \frac{2^2}{3-1} \cdot \frac{1}{c_2 \cdot c_1} \sum_{j_1=1}^{3} \frac{\psi_2\left(\zeta_{j_1}\right) \cdot \psi_1\left(\zeta_1\right) \cdot f\left(\zeta_{j_1}, \zeta_1\right)}{c_{j_1} \cdot 1} \\
&= \frac{\psi_2\left(\zeta_1\right) \cdot f\left(\zeta_1, \zeta_1\right)}{c_1} + \frac{\psi_2\left(\zeta_2\right) \cdot f\left(\zeta_2, \zeta_1\right)}{c_2} + \frac{\psi_2\left(\zeta_3\right) \cdot f\left(\zeta_3, \zeta_1\right)}{c_3} \\
&= \frac{-1 \cdot f\left(-1, 0\right)}{2} + \frac{1 \cdot f\left(1, 0\right)}{2};
\end{aligned}$$

and b_{12} is obtained similarly,

$$b_{12} = -\frac{f\left(0, -1\right)}{2} + \frac{f\left(0, 1\right)}{2}.$$

Coefficient b_{31} is given by

$$\begin{aligned}
b_{31} &= \frac{2}{3-1} \frac{1}{c_3 \cdot c_1} \sum_{j_1=1}^{3} \frac{\psi_3\left(\zeta_{j_1}\right) \cdot \psi_1\left(\zeta_1\right) \cdot f\left(\zeta_{j_1}, \zeta_1\right)}{c_{j_1}} \\
&= \frac{1}{2}\left[\frac{1 \cdot f\left(-1, 0\right)}{2} - f(0,0) + \frac{1 \cdot f\left(1, 0\right)}{2}\right] \\
&= -\frac{f\left(0, 0\right)}{2} + \frac{f\left(-1, 0\right) + f\left(1, 0\right)}{4},
\end{aligned}$$

and b_{13} is obtained similarly

$$b_{13} = -\frac{f\left(0, 0\right)}{2} + \frac{f\left(0, -1\right) + f\left(0, 1\right)}{4}.$$

Conditions defining b_{21}, b_{12}, b_{31}, and b_{13} are the same as those we had in (30). Formula (33) cannot be used to find b_{11}. We use (38) instead, which implies

$$\begin{aligned}
\widehat{f}\left(0, 0; b\right) &= b_{11} - b_{31} - b_{13} \\
&= b_{11} + \frac{f\left(0, 0\right)}{2} - \frac{f\left(-1, 0\right) + f\left(1, 0\right)}{4} + \frac{f\left(0, 0\right)}{2} - \frac{f\left(0, -1\right) + f\left(0, 1\right)}{4}.
\end{aligned}$$

Since under interpolation, we must have $\widehat{f}\left(0, 0; b\right) = f\left(0, 0\right)$, the last formula implies

$$b_{11} = \frac{f\left(-1, 0\right) + f\left(1, 0\right) + f\left(0, -1\right) + f\left(0, 1\right)}{4},$$

which coincides with the condition for b_{11} in (30).

4.1.3 Conventional Interpolation Formula with Repetitions of Elements Is Inefficient

The conventional interpolation formula (31)–(33) is inefficient. First, it creates a long list with many repeated elements from all tensors that satisfy

$$\max(d, \mu + 1) \leq i_1 + \cdots + i_d \leq d + \mu, \tag{39}$$

and then, it assigns weights to each element in the list such that the repeated elements are cancelled out.

The number of repeated elements in the list increases with both the dimensionality of the problem and the approximation level. In particular, for our two-dimensional example, the list involves seven elements under $\mu = 1$ (i.e., two elements are repeated) and 25 elements under $\mu = 2$ (i.e., 12 elements are repeated). In high-dimensional applications, the number of repetitions is hideous, which seriously limits the capacities of the Smolyak method.

To summarize, the original Smolyak's (1963) construction provides us with a set of grid points and basis functions that are an efficient choice for interpolation. However, the formula based on (39) is an inefficient way to arrive to such grid points and basis functions. Judd et al. (2013) show a Smolyak interpolation formula that avoids repetitions of elements and that is an efficient alternative to the conventional interpolation formula based on (39).

4.2 Generalized Stochastic Simulation Algorithm

The idea of finding solutions to economic models using stochastic simulation dates back to Marcet's (1988) parameterized expectations algorithm (PEA), which was later developed in Den Haan and Marcet (1990); see Marcet and Lorenzoni (1999) for a detailed description of this algorithm. PEA is feasible in high-dimensional problems but its overall accuracy is limited by a low accuracy of Monte Carlo integration. Moreover, PEA builds on least-squares learning which is numerically unstable in the context of stochastic simulation. Judd et al. (2009, 2011b) propose a generalized stochastic simulation algorithm (GSSA), in which inaccurate Monte Carlo integration is replaced with accurate deterministic integration, and in which unstable least-squares learning is replaced with regression methods suitable for dealing with ill-conditioned problems. The resulting GSSA method is numerically stable and delivers accuracy of solutions comparable to the best accuracy attained in the related literature. In particular, Hasanhodzic and Kotlikoff (2013) applied GSSA to solve life-cycle models with up to 80 periods.

Again, for the sake of expositional convenience, we work with a simple two-dimensional example. In Sections 4.2.1, 4.2.2, and 4.2.3, we show how stochastic simulation methods implement the three steps outlined in the beginning of Section 4: (i) constructs a grid of simulated points, (ii) constructs an approximating function using some nonproduct rules, and (iii) identifies the parameters in the approximating function by way of regression. In the remainder of the section, we discuss advantages and shortcomings of stochastic simulation methods, provide a description of PEA and GSSA, and discuss computational techniques that can improve the performance of stochastic simulation methods in problems with high dimensionality.

4.2.1 A Grid of Simulated Points

A stochastic simulation approach solves a dynamic economic model on a grid composed of simulated points. Constructing such a grid is simple. As an example, consider the neoclassical stochastic growth model (1)–(3) described in Section 3. We draw and fix a sequence of productivity shocks $\{\epsilon_{t+1}\}_{t=0,\ldots,T}$ and construct productivity levels $\{\theta_{t+1}\}_{t=0,\ldots,T}$ using (3). We guess a parametric function $\widehat{K}(\cdot; b)$ that approximates the true capital decision function, K, choose initial condition (k_0, θ_0), fix the coefficient vector b, and simulate the model forward using $k_{t+1} = \widehat{K}(k_t, \theta_t; b)$. The resulting series $\{k_t, \theta_t\}_{t=1,\ldots,T}$ are used as a grid for finding a solution.

4.2.2 An Approximating Function

As a set of basis functions for approximation, we can use any family of functions which is sufficiently flexible (to ensure a sufficiently high accuracy), and which does not grow too rapidly with the dimensionality of the problem (to ensure a reasonably low cost). In particular, Judd et al. (2011b) advocate families of complete polynomials that are linear in coefficients b, i.e.,

$$\widehat{K}(k, \theta; b) = \sum_{i=1}^{I} b_i \Psi_i(k, \theta), \tag{40}$$

where $\{\Psi_i \mid i = 1, \ldots, I\}$ is a set of basis functions and $b \equiv (b_1, \ldots, b_I)^\top \in \mathbb{R}^I$. The basis functions can be ordinary polynomials, e.g., $\widehat{K}(k, \theta; b) = b_1 + b_2 k + b_3 \theta$, or some other polynomials; for example, Judd et al. (2011b) and Maliar et al. (2011) combine a grid of points produced by stochastic simulation with Hermite and Smolyak polynomials, respectively. The numbers of terms in a complete polynomial function of degrees one, two, and three are given by $1 + d$, $1 + d + \frac{d(d+1)}{2}$, $1 + d + \frac{d(d+1)}{2} + d^2 + \frac{d(d-1)(d-2)}{6}$, i.e., they grow linearly, quadratically, and cubically, respectively.

4.2.3 Finding the Polynomial Coefficients by Way of Regression

We estimate the coefficients in (40) by way of regression

$$y_t = X_t b + \varepsilon_t, \tag{41}$$

where y_t is the model's variable for which we aim to approximate the decision function, e.g., $y_t = k_{t+1}$; X_t is the list of basis functions in (40), e.g., $X_t \equiv \left[1, k_t, \theta_t, k_t^2, k_t\theta_t, \theta_t^2, \ldots, \theta_t^L\right] \in \mathbb{R}^I$ with L being the polynomial degree; and ε_t is the residual between the true value of y_t and fitted value $\widehat{y}_t = X_t b$. Regression Equation (41) is a special case of the general approximation problem (20).

If simulated points are used as a grid, then basis functions do not have the property of orthogonality. In particular, there is no guarantee that any specific polynomial function composed of I basis functions will pass exactly through I simulated points, unlike they do in the Smolyak collocation method. Therefore, we will not try to fit the approximating function exactly to any of the simulated points; instead, we will proceed as is done in the regression analysis in econometrics. Namely, we will choose a simulation length to be larger than the number of explanatory variables, i.e., $T > I$, and we will identify coefficients b by minimizing the residuals in regression equation (41). As we will see, the way in which the regression coefficients are estimated will play an important role in the accuracy and numerical stability of stochastic simulation methods.

4.2.4 Advantages of the Stochastic Simulation Approach

The key advantage of the stochastic simulation approach is that it produces an adaptive grid. Points that are realized in simulation belong to a high-probability set of a given model; we also refer to this set as *an essentially ergodic set*; see Judd et al. (2012) for a formal definition of an essentially ergodic set. By focusing just on those points, we can save on the cost of computing a solution in those areas that are not visited in equilibrium. How much can one save on cost by focusing on the essentially ergodic set instead of a multidimensional hypercube that encloses such a set? As an illustration, in Figure 3, we plot the simulated series of capital and productivity levels for the one-agent model (1)–(3). The essentially ergodic set takes the form of an oval, and the rectangular area that sits outside of the oval's boundaries is effectively never visited. In the two-dimensional case, a circle inscribed within a square occupies about 79% of the area of the square, and an oval inscribed in this way occupies an even smaller area. Thus, the essentially ergodic set is at least 21% smaller than the hypercube enclosing this set. In general, the ratio of the volume of a d-dimensional hypersphere of diameter 1, $\mathcal{V}_O^d$, to the volume of a

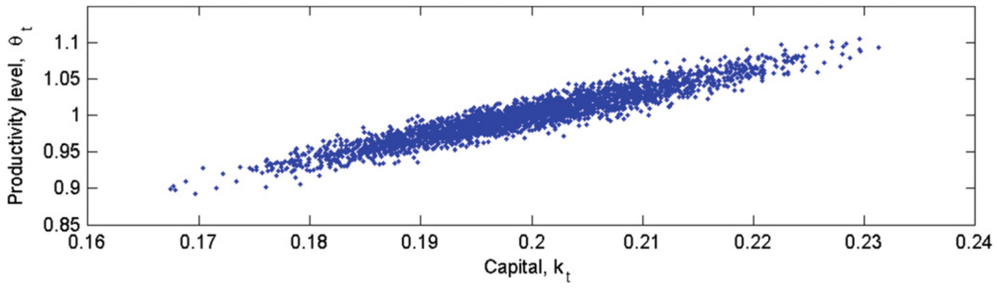


Figure 3 A high probability set in the one-agent model.

d-dimensional hypercube of width 1, $\mathcal{V}_{\square}^{d}$, is

$$\frac{\mathcal{V}_{O}^{d}}{\mathcal{V}_{\square}^{d}} = \begin{cases} \frac{(\pi/2)^{\frac{d-1}{2}}}{1\cdot 3\cdot \ldots \cdot d} & \text{for} \quad d = 1, 3, 5, \ldots \\ \frac{(\pi/2)^{\frac{d}{2}}}{2\cdot 4\cdot \ldots \cdot d} & \text{for} \quad d = 2, 4, 6, \ldots \end{cases} \tag{42}$$

The ratio $\frac{\mathcal{V}_{O}^{d}}{\mathcal{V}_{\square}^{d}}$ declines very rapidly with the dimensionality of the state space. For example, for dimensions three, four, five, ten, and thirty, this ratio is $0.52, 0.31, 0.16, 3 \cdot 10^{-3}$, and $2 \cdot 10^{-14}$, respectively.

This example suggests that focusing on the right geometry can be critical for accuracy and cost in high-dimensional applications. Potential benefits are twofold: first, when computing a solution on an essentially ergodic set, we face just a fraction of the cost we would have faced on a hypercube grid, used in conventional projection methods. The higher the dimensionality of a problem, the larger the reduction in cost is likely to be. Second, when fitting a polynomial on an essentially ergodic set, stochastic simulation methods focus on a relevant solution domain and can get a better fit inside this domain than methods that operate on a hypercube and that face a trade-off between the fit inside and outside the relevant domain.

4.2.5 Marcet's (1988) Parameterized Expectations Algorithm

Marcet's (1988) parameterized expectations algorithm (PEA) approximates the expectation function in the right side of the Euler equation by an exponentiated polynomial of logarithm of the state variables,

$$\begin{aligned} &E_t \left[u_1 \left(c_{t+1}, \ell_{t+1}\right) \left(1 - \delta + \theta_{t+1} f_1 \left(k_{t+1}, \ell_{t+1}\right)\right)\right] \\ &\quad \approx \widehat{\Phi} \left(k_t, \theta_t; b\right) = \exp \left(b_1 + b_2 \ln \left(k_t\right) + b_3 \ln \left(\theta_t\right) + \cdots + b_I \left[\ln \left(\theta_t\right)\right]^L\right), \end{aligned} \tag{43}$$

where L is a degree of approximating polynomial. This also identifies the left side of the Euler equation as $u_1 \left(c_t, \ell_t\right) = \beta \widehat{\Phi} \left(k_t, \theta_t; b\right)$.[12] Using the assumed $\widehat{\Phi}$, PEA simulates the model forward to construct the series $\{\theta_t, k_t\}_{t=1,\ldots,T}$. Next, it approximates the conditional expectation function by the realized value of the integrand for each $t = 1, \ldots, T-1$,

$$\begin{aligned} &E_t \left[u_1 \left(c_{t+1}, \ell_{t+1}\right) \left(1 - \delta + \theta_{t+1} f_1 \left(k'_{t+1}, \ell'_{t+1}\right)\right)\right] \equiv E_t \left[y_{t+1}\right] \\ &\quad \approx u_1 \left(c_{t+1}, \ell_{t+1}\right) \left[1 - \delta + \theta_{t+1} f_1 \left(k'_{t+1}, \ell'_{t+1}\right)\right] \equiv y_{t+1}. \end{aligned} \tag{44}$$

Finally, PEA fits values of y_{t+1} to $\widehat{\Phi} \left(k_t, \theta_t; b\right)$ using a nonlinear least-squares (NLLS) regression to compute $\widehat{b}$; it iterates on b until convergence.

[12] The name of Marcet's (1988) method highlights the way in which it deals with uncertainty; namely, it parameterizes a conditional expectation function and approximates such a function from simulated data using a Monte Carlo integration method (combined with a least-squares learning).

Marcet's (1988) implementation of a stochastic simulation algorithm has important shortcomings that limit both accuracy and numerical stability. First, the PEA uses stochastic simulation not only to construct a grid of points for approximating the solution but also as a set of nodes for integration. However, the one-node Monte Carlo method used has low accuracy (unless the simulation length is impractically long). This is not surprising given that PEA replaces an expectation of a random variable $E_t\left[y_{t+1}\right]$ with just one realization of this variable y_{t+1}; see Section 5.3 for a discussion of the accuracy of this integration method.

Second, PEA relies on standard least-squares (LS) learning methods for fitting the approximating functions to simulated data, and such LS methods are numerically unstable in the given context (beyond the first-degree polynomials). This is because monomial terms constructed on simulated series are highly correlated in the studied class of economic models. The multicollinearity problem is severe even under low-degree polynomials. In particular, Den Haan and Marcet (1990) report that in a second-degree PEA solution, the cross term $\ln k_t \ln \theta_t$ in (43) is highly correlated with the other terms and must be removed from the regression.

Finally, PEA builds on a nonlinear approximating function, which is an additional source of numerical instability and computational expense. The resulting nonlinear regression model must be estimated with NLLS methods. Such methods need a good initial guess, may deliver multiple minima, and on many occasions fail to converge; moreover, nonlinear optimization is costly because it requires computing Jacobian and Hessian matrices; see Christiano and Fisher (2000) for a discussion.

4.2.6 Generalized Stochastic Simulation Algorithm by Judd et al. (2011b)

Judd et al. (2011b) propose a generalized stochastic simulation algorithm (GSSA) that does not suffer from the above shortcomings. First, GSSA uses stochastic simulation only for constructing a grid on which a solution is approximated; however, it computes integrals using accurate quadrature and monomial integration formulas described in Sections 5.1 and 5.2, respectively. Second, GSSA stabilizes stochastic simulation by using regression methods that are suitable for ill-conditioned problems. Finally, GSSA uses families of basis functions of type (40) that are linear in coefficients b, and it builds the fitting step on a linear regression model that can be estimated with simple and reliable linear approximation methods. As a result, GSSA can deliver high-degree polynomial approximations and attain accuracy of solutions comparable to the best accuracy in the related literature. Below, we show regression methods and other techniques that GSSA uses to stabilize stochastic simulation.

Ill-Conditioned LS Problem

We now explain when the multicollinearity arises and show how it affects the regression outcomes. We consider the simplest possible approximating family, the one composed of ordinary polynomials. The objective is to fit simulated series for a variable y_t on an ordinary polynomial function of the current state variables $X_t b$ as shown in (41).

Consider the standard LS approach to regression equation (41):

$$\min_b \left\| y - Xb \right\|_2^2 = \min_b \left[y - Xb \right]^\top \left[y - Xb \right], \tag{45}$$

where $\|\cdot\|_2$ denotes the L_2 vector norm; $y \equiv (y_0, \ldots, y_{T-1})^\top \in \mathbb{R}^T$ and $X \equiv (X_0, \ldots, X_{T-1})^\top \in \mathbb{R}^{T \times I}$.

The solution to (45) is

$$\widehat{b} = \left(X^\top X \right)^{-1} X^\top y. \tag{46}$$

LS problem (45) is often ill-conditioned when X is generated by stochastic simulation. To measure the degree of ill-conditioning, we use the condition number of matrix $X^\top X$, denoted by $\mathcal{K}\left(X^\top X\right)$ and defined as follows: We order the eigenvalues, λ_i, $i = 1, \ldots, I$, of $X^\top X$ by their magnitude, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_I \geq 0$, and find a ratio of the largest eigenvalue, λ_1, of $X^\top X$ to its smallest eigenvalue, λ_I, i.e., $\mathcal{K}\left(X^\top X\right) \equiv \lambda_1 / \lambda_I$. The eigenvalues of $X^\top X$ are defined by the standard eigenvalue decomposition $X^\top X = V \Lambda V^\top$, where $\Lambda \in \mathbb{R}^{I \times I}$ is a diagonal matrix of eigenvalues of $X^\top X$, and $V \in \mathbb{R}^{I \times I}$ is an orthogonal matrix of eigenvectors of $X^\top X$. A large condition number implies that $X^\top X$ is close to being singular and not invertible and tells us that any linear operation, such as (46), is very sensitive to perturbation and numerical errors (such as round-off errors).

Two causes of ill-conditioning are multicollinearity and poor scaling of the variables constituting X. Multicollinearity occurs when the variables forming X are significantly correlated. The following example illustrates the effects of multicollinearity on the LS solution (we analyze the sensitivity to changes in y but the results are similar for the sensitivity to changes in X).

Example 4. Let $X = \begin{bmatrix} 1+\phi & 1 \\ 1 & 1+\phi \end{bmatrix}$ with $\phi \neq 0$. Then, $\mathcal{K}\left(X^\top X\right) = \left(1 + \frac{2}{\phi}\right)^2$. Let $y = (0,0)^\top$. Thus, OLS solution (46) is $\left(\widehat{b}_1, \widehat{b}_2\right) = (0,0)$. Suppose y is perturbed by a small amount, i.e., $y = (\varepsilon_1, \varepsilon_2)^\top$. Then, the OLS solution is

$$\widehat{b}_1 = \frac{1}{\phi} \left[\frac{\varepsilon_1 (1+\phi) - \varepsilon_2}{2+\phi} \right] \quad \text{and} \quad \widehat{b}_2 = \frac{1}{\phi} \left[\frac{\varepsilon_2 (1+\phi) - \varepsilon_1}{2+\phi} \right]. \tag{47}$$

The sensitivity of $\widehat{b}_1$ and $\widehat{b}_2$ to perturbation in y is proportional to $1/\phi$ (increases with $\mathcal{K}\left(X^\top X\right)$).

The scaling problem arises when the columns of X have significantly different means and variances (due to differential scaling among either the state variables, k_t and θ_t, or their functions, e.g., k_t and k_t^5). A column with only very small entries will be treated as if it were a column of zeros. The next example illustrates the effect of the scaling problem.

Example 5. Let $X = \begin{bmatrix} 1 & 0 \\ 0 & \phi \end{bmatrix}$ with $\phi \neq 0$. Then, $\mathcal{K}\left(X^\top X\right) = 1/\phi$. Let $y = (0,0)^\top$. Thus, OLS solution (46) is $\left(\widehat{b}_1, \widehat{b}_2\right) = (0,0)$. Suppose y is perturbed by a small amount, i.e., $y = (\varepsilon_1, \varepsilon_2)^\top$. The OLS solution is

$$\widehat{b}_1 = \varepsilon_1 \quad \text{and} \quad \widehat{b}_2 = \frac{\varepsilon_2}{\phi}. \tag{48}$$

Sensitivity of $\widehat{b}_2$ to perturbation in y is proportional to $1/\phi$ (and $\mathcal{K}\left(X^\top X\right)$).

A comparison of Examples 1 and 2 shows that multicollinearity and poor scaling magnify the impact of perturbations on the OLS solution. Each iteration of a stochastic simulation algorithm produces changes in simulated data (perturbations). In the presence of ill-conditioning, these changes, together with numerical errors, may induce large, erratic jumps in the regression coefficients and failures to converge.

Data Normalization

Data normalization addresses the scaling issues highlighted in Example 2. We center and scale both the response variable y and the explanatory variables of X to have a zero mean and unit standard deviation. We then estimate a regression model without an intercept to obtain the vector of coefficients $\left(\widehat{b}_1^+, \ldots, \widehat{b}_I^+\right)$. We finally restore the coefficients $\widehat{b}_1, \ldots, \widehat{b}_I$ and the intercept $\widehat{b}_0$ in the original (unnormalized) regression model according to $\widehat{b}_i = \left(\sigma_y/\sigma_{x_i}\right)\widehat{b}_i^+$, $i = 1, \ldots, I$, and $\widehat{b}_0 = \overline{y} - \sum_{i=1}^{I} \widehat{b}_i \overline{x}_i$, where $\overline{y}$ and $\overline{x}_i$ are the sample means, and σ_y and σ_{x_i} are the sample standard deviations of the original unnormalized variables y and x_i, respectively.

Choosing a Family of Basis Functions

An obvious example of a family of type (40) is an ordinary polynomial family, $P_n(x) = x^n$, $n = 0, 1, \ldots$. However, as we argued in Section 3, the basis functions of this family look very similar (e.g., $P_2(x) = x^2$ looks similar to $P_4(x) = x^4$, and $P_3(x) = x^3$ looks similar to $P_5(x) = x^5$). As a result, the explanatory variables in the regression equation are likely to be strongly correlated (i.e., the LS problem is ill-conditioned) and estimation methods (e.g., OLS) may fail because they cannot distinguish between similarly shaped polynomial terms.

In contrast, for families of orthogonal polynomials (e.g., Hermite, Chebyshev, Legendre), basis functions have very different shapes and, hence, the multicollinearity problem is likely to manifest to a smaller degree, if at all. In the context of GSSA, we consider the family of Hermite polynomials. Such polynomials can be defined with a simple recursive formula: $H_0(x) = 1$, $H_1(x) = x$, and $H_n(x) = xH_{n-1}(x) - (n-1)H_{n-2}(x)$, which yields $H_2(x) = x^2 - 1$, $H_3(x) = x^3 - 3x$, $H_4(x) = x^4 - 6x^2 + 3$, and $H_5(x) = x^5 - 10x^3 + 15x$.

Two points are in order. First, while Hermite polynomials are orthogonal under the Gaussian density function, they are not orthogonal under the ergodic measure derived from stochastic simulation. Still, Hermite polynomials are far less correlated than ordinary polynomials which suffice to avoid ill-conditioning. Second, even though using Hermite polynomials helps us avoid ill-conditioning in one variable, they may not suffice to deal with multicollinearity across variables. For example, if k_t and θ_t are perfectly correlated, certain Hermite polynomial terms for k_t and θ_t, such as $H_2(k_t) = k_t^2 - 1$ and $H_2(\theta_t) = \theta_t^2 - 1$, are also perfectly correlated and, hence, X is singular. Thus, we may still need to complement Hermite polynomials with regression methods that suit ill-conditioned problems.[13]

Approximation Methods Suitable for Dealing with Multicollinearity

We now review fitting (regression) techniques that are suitable for dealing with ill-conditioned problems. We also discuss the choice of a family of approximating functions; see Judd et al. (2011b) for more details.

LS-SVD We now present two LS methods that are more robust to ill-conditioning than the standard OLS method. The first approach, called *LS using SVD (LS-SVD)*, relies on a singular value decomposition (SVD) of X. We use the SVD of X to rewrite OLS solution (46) in a way that does not require an explicit computation of $\left(X^\top X\right)^{-1}$. For a matrix $X \in \mathbb{R}^{T\times I}$ with $T > I$, an SVD decomposition is

$$X = USV^\top, \tag{49}$$

where $U \in \mathbb{R}^{T\times I}$ and $V \in \mathbb{R}^{I\times I}$ are orthogonal matrices and $S \in \mathbb{R}^{I\times I}$ is a diagonal matrix with diagonal entries $s_1 \geq s_2 \geq \cdots \geq s_I \geq 0$, known as *singular values* of X.[14] The condition number of X is its largest singular value divided by its smallest singular value, $\mathcal{K}(X) = s_1/s_I$. The singular values of X are related to the eigenvalues of $X^\top X$ by $s_i = \sqrt{\lambda_i}$. This implies that $\mathcal{K}(X) = \mathcal{K}(S) = \sqrt{\mathcal{K}(X^\top X)}$. The OLS estimator $\widehat{b} = \left(X^\top X\right)^{-1} X^\top y$ in terms of SVD (49) is

$$\widehat{b} = VS^{-1}U^\top y. \tag{50}$$

With an infinite-precision computer, OLS formula (46) and LS-SVD formula (50) give us identical estimates of b. With a finite-precision computer, the standard OLS estimator cannot be computed reliably if $X^\top X$ is ill-conditioned. However, it is still possible that S is sufficiently well-conditioned so that the LS-SVD estimator can be computed successfully.[15]

[13] Christiano and Fisher (2000) found that multicollinearity can plague the regression step even with orthogonal (Chebyshev) polynomials as basis functions.

[14] For a description of methods for computing the SVD of a matrix, see, e.g., Golub and Van Loan (1996), pp. 448–460. Routines that compute the SVD are readily available in modern programming languages.

[15] Another decomposition of X that leads to a numerically stable LS approach is a QR factorization; see, e.g., Davidson and MacKinnon (1993), pp. 30–31, and Golub and Van Loan (1996), p. 239.

RLS-Tikhonov The second approach, called *regularized LS using Tikhonov regularization (RLS-Tikhonov)*, imposes penalties based on the size of the regression coefficients. In essence, a regularization method replaces an ill-conditioned problem with a well-conditioned problem that gives a similar answer. In statistics, Tikhonov regularization is known as *ridge regression* and is classified as a shrinkage method because it shrinks the norm of estimated coefficient vector relative to the nonregularized solution. Formally, Tikhonov regularization imposes an L_2 penalty on the magnitude of the regression-coefficient vector; i.e., for a regularization parameter $\eta \geq 0$, the vector $b(\eta)$ solves

$$\min_b \|y - Xb\|_2^2 + \eta \|b\|_2^2 = \min_b (y - Xb)^\top (y - Xb) + \eta b^\top b, \tag{51}$$

where $y \in \mathbb{R}^T$ and $X \in \mathbb{R}^{T \times I}$ are centered and scaled, and $b \in \mathbb{R}^I$. The parameter η controls the amount by which the regression coefficients are shrunk, with larger values of η leading to greater shrinkage.

Finding an FOC of (51) with respect to b gives us the following estimator

$$\widehat{b}(\eta) = \left(X^\top X + \eta I_I\right)^{-1} X^\top y, \tag{52}$$

where I_I is an identity matrix of order I. Note that Tikhonov regularization adds a positive constant multiple of the identity matrix to $X^\top X$ prior to inversion. Thus, if $X^\top X$ is nearly singular, the matrix $X^\top X + \eta I_I$ is less singular, reducing problems in computing $\widehat{b}(\eta)$.

LAD Approaches LAD, or L_1, regression methods use linear programming to minimize the sum of absolute deviations. LAD methods do not construct $\left(X^\top X\right)^{-1}$ and avoid the ill-conditioning. The basic LAD method solves the optimization problem

$$\min_b \|y - Xb\|_1 = \min_b 1_T^\top |y - Xb|, \tag{53}$$

where $\|\cdot\|_1$ denotes the L_1 vector norm, and $|\cdot|$ denotes the absolute value. Without a loss of generality, we assume that X and y are centered and scaled.

There is no explicit solution to LAD problem (53), but the LAD problem (53) is equivalent to the linear programming problem:

$$\min_{g,b} 1_T^\top g \tag{54}$$

$$\text{s.t.} \quad -g \leq y - Xb \leq g, \tag{55}$$

where $g \in \mathbb{R}^T$. The problem has $I + T$ unknowns. Although this formulation of the LAD problem is intuitive, it is not the most suitable for a numerical analysis. Judd et al. (2011b) show primal and dual implementations of the LAD problems that are more convenient for numerical treatment.

Regularized LAD Approaches Similar to the LS case, we can modify the original LAD problem (53) to incorporate an L_1 penalty on the coefficient vector b. We refer to the resulting problem as a *regularized LAD* (RLAD). Like Tikhonov regularization, the RLAD problem shrinks the values of the coefficients toward zero. Introducing an L_1 penalty in place of the L_2 penalty used in Tikhonov regularization allows us to have the same benefits of shrinking the coefficients but in the context of a linear programming approach. Formally, for a given regularization parameter $\eta \geq 0$, the RLAD problem finds a vector $b(\eta)$ that solves

$$\min_b \|y - Xb\|_1 + \eta \|b\|_1 = \min_b 1_T^\top |y - Xb| + \eta 1_I^\top |b|, \tag{56}$$

where $y \in \mathbb{R}^T$ and $X \in \mathbb{R}^{T\times I}$ are centered and scaled, and $b \in \mathbb{R}^I$. See Judd et al. (2011b) for a detailed description of the primal and dual formulations of the regularized LAD problems.

Principal Component Method A principal component method reduces the multicollinearity in the data to a given target level. Let $X \in \mathbb{R}^{T\times I}$ be a matrix of centered and scaled explanatory variables and consider the SVD of X defined in (49). Let us make a linear transformation of X using $Z \equiv XV$, where $Z \in \mathbb{R}^{T\times I}$ and $V \in \mathbb{R}^{I\times I}$ is the matrix of singular vectors of X defined by (49). Vectors $z_1, \ldots, z_I$ are called *principal components* of X. They are orthogonal, $z_j^\top z_i = 0$ for any $j \neq i$, and their norms are related to singular values s_i by $z_i^\top z_i = s_i^2$. Principal components have two noteworthy properties. First, the sample mean of each principal component z_i is equal to zero, since it is given by a linear combination of centered variables $X_1, \ldots, X_I$, each of which has a zero mean; second, the variance of each principal component is equal to s_i^2/T, because we have $z_i^\top z_i = s_i^2$.

By construction of the SVD, if z_i has a zero variance (equivalently, a zero singular value, $s_i = 0$), then all entries of z_i are equal to zero, $z_i = (0, \ldots, 0)^\top$, which implies that variables $x_1, \ldots, x_I$ constituting this particular principal component are linearly dependent. Therefore, we can reduce the degrees of ill-conditioning of X to some target level by excluding low-variance principal components corresponding to small singular values.

Let κ represent the largest condition number of X that we are willing to tolerate. Let us compute the ratios of the largest singular value to all other singular values, $\frac{s_1}{s_2}, \ldots, \frac{s_1}{s_I}$. (Recall that the last ratio is the actual condition number of matrix X; $\mathcal{K}(X) = \mathcal{K}(S) = \frac{s_1}{s_I}$). Let $Z^r \equiv (z_1, \ldots, z_r) \in \mathbb{R}^{T\times r}$ be the first r principal components for which $\frac{s_1}{s_i} \leq \kappa$, and let us remove the last $I - r$ principal components for which $\frac{s_1}{s_i} > \kappa$. By construction, Z^r has a condition number which is smaller than or equal to κ.

Let us consider regression equation (41) and let us approximate Xb using Z^r such that $Xb = XVV^{-1}b \approx XV^r (V^r)^{-1} b(\kappa) = Z^r \vartheta^r$, where $V^r = (v_1, \ldots, v_r) \in \mathbb{R}^{I\times r}$ contains the first r right singular vectors of X and $\vartheta^r \equiv (V^r)^{-1} b(\kappa) \in \mathbb{R}^r$. The resulting regression equation is

$$y = Z^r \vartheta^r + \varepsilon, \tag{57}$$

where y is centered and scaled. The coefficients ϑ^r can be estimated by any of the methods described in this section. For example, we can compute OLS estimator (46). Once we compute $\widehat{\vartheta}^r$, we can recover the coefficients as $\widehat{b}(\kappa) = V^r \widehat{\vartheta}^r \in \mathbb{R}^I$. Instead of the principal component methods, we can use a truncated SVD method; see Judd et al. (2011b) for a description of this method.

4.2.7 Numerical Illustration of the Importance of the Approximating Function and Fitting Method

As an illustration, we consider the one-agent model with inelastic labor supply under parameterization (18) that allows for a closed-form solution. Judd et al. (2011b) use the GSSA method to solve this model under a simulation length $T = 10{,}000$; see Judd et al. (2011b) for details. In Table 6, we provide the results that illustrate the importance of a choice of the approximating function. When the OLS method is used with unnormalized data and ordinary polynomials, we cannot go beyond the second-degree polynomial approximations. The data normalization improves the performance of the OLS method; however, we still cannot calculate more than a third-degree polynomial approximation. The introduction of Hermite polynomials completely resolves the ill-conditioning of the LS problem: OLS can compute all five degrees of the polynomial approximations, and the accuracy of these approximations improves systematically as we move from the first- to the fifth-degree polynomials, e.g., the average Euler equation residuals decrease from 10^{-4} to 10^{-9}.

In Table 7, we illustrate the importance of the fitting methods. The RLS-Tikhonov method leads to visibly less accurate solutions than the LS-SVD method. This happens because RLS-Tikhonov and LS-SVD work with different objects: the former works with a very ill-conditioned matrix $X^\top X$, while the latter works with a better conditioned matrix S. Under RLAD-DP and LS-SVD, the solutions are very accurate: the average

Table 6 Conventional OLS in the one-agent model with a closed-form solution: ordinary polynomial with unnormalized and normalized data and Hermite polynomials.[a]

	Ordinary polynomials						Hermite polynomials		
	Unnormalized data			Normalized data			Unnormalized data		
Polynomial degree	L_1	L_∞	CPU	L_1	L_∞	CPU	L_1	L_∞	CPU
1st	−3.52	−2.45	0.8	−3.52	−2.45	1	−3.52	−2.45	1
2nd	−5.46	−4.17	3.1	−5.45	−4.17	3	−5.46	−4.17	4
3rd	–	–	–	−6.84	−5.36	5	−6.84	−5.36	6
4th	–	–	–	–	–	–	−7.97	−6.35	8
5th	–	–	–	–	–	–	−9.09	−7.29	10

[a] *Notes:* L_1 and L_∞ are, respectively, the average and maximum of absolute residuals across optimality conditions and test points (in log10 units). CPU is the time necessary for computing a solution (in seconds). These results are reproduced from Judd et al. (2011b), Table 1.

Table 7 Other fitting methods in the one-agent model with a closed-form solution: RLS-Tikhonov, RLAD-DP and LS-SVD.[a]

	RLS-Tikhonov			RLAD			LS-SVD		
Polynomial degree	L_1	L_∞	CPU	L_1	L_∞	CPU	L_1	L_∞	CPU
1st	−3.52	−2.45	1	−3.52	−2.45	3	−3.52	−2.45	1
2nd	−5.46	−4.17	3	−5.55	−4.12	10	−5.46	−4.17	4
3rd	−6.84	−5.36	5	−6.98	−5.25	19	−6.84	−5.36	6
4th	−6.97	−5.63	8	−8.17	−6.13	45	−7.97	−6.35	8
5th	–	–	–	−8.17	−6.15	71	−9.08	−7.25	9

[a] *Notes:* L_1 and L_∞ are, respectively, the average and maximum of absolute residuals across optimality conditions and test points (in log10 units). CPU is the time necessary for computing a solution (in seconds). RLS-Tikhonov uses ordinary polynomials and $\eta = 10^{-10}$. RLAD uses ordinary polynomials and $\eta = 10^{-6}$. LS-SVD uses Hermite polynomials. These results are reproduced from Judd et al. (2011b), Table 1. RLAD corresponds to RLAD-DP in Judd et al. (2011b), where DP stands for a dual problem.

Euler equation residuals decrease from 10^{-4} to 10^{-9} when the polynomial degree increases from one to five.

4.3 ε-Distinguishable Set and Cluster Grid Algorithms

The Smolyak method provides an efficient nonproduct discretization of a hypercube; however, the hypercube itself is not an efficient choice for a solution domain. In order to capture all points that are visited in equilibrium, the hypercube must typically contain large areas that are unlikely to happen in equilibrium. In turn, simulated points used by GSSA occupy a much smaller area than the hypercube and allow us to focus on a relevant solution domain. However, a set of simulated points itself is not an efficient discretization of such a domain because a grid of simulated points is unevenly spaced; has many closely located, redundant points; and contains some points in low-density regions.

Judd et al. (2012) introduce an *ε-distinguishable set* (EDS) technique that selects a subset of points situated at the distance of at least ε from one another. This construction combines the best features of stochastic simulation and hypercube grids; namely, it combines an adaptive geometry with efficient discretization. In Figure 4a and b, we shows an example of such a grid for the one-agent model.

As is seen in the figure, the resulting EDS covers the high-probability area of the state space roughly uniformly. The EDS grid can be combined with any family of approximating functions, in particular with those of type (40). As in the case of stochastic simulation methods, we use more EDS grid points than the parameters in the approximating function, and we identify the parameters of such a function using regression methods that are robust to ill-conditioned problems. Therefore, we just focus on explaining how the EDS grid is constructed.

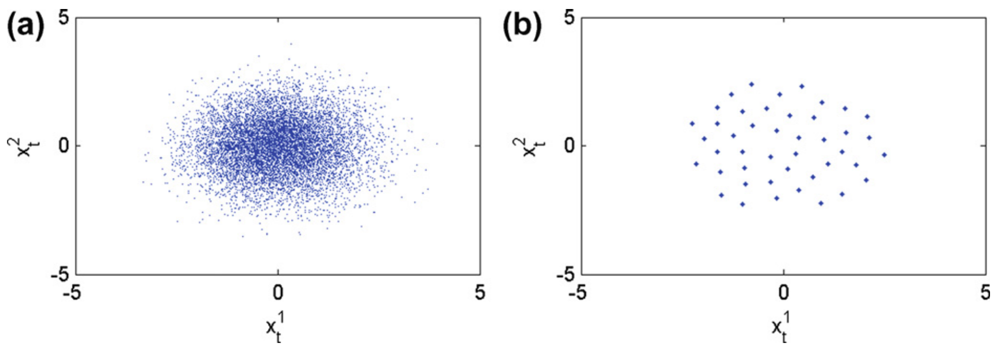


Figure 4 (a) A set of simulated points. (b) A subset of simulated points.

4.3.1 Eliminating Simulated Points Outside the High-Probability Set

Let us assume some equilibrium law of motion φ for state variables

$$x_{t+1} = \varphi\left(x_t, \epsilon_{t+1}\right), \quad t = 0, 1, \ldots, \tag{58}$$

where $x_t \in \mathbb{R}^d$ is a vector of d (exogenous and endogenous) state variables. We first select a subset of simulated points which belong to an essentially ergodic set $\mathcal{A}^\eta$ using the following algorithm.

(Algorithm $\mathcal{A}^\eta$): An algorithm for selecting points within an essentially ergodic set.

Step 1. Simulate $\varphi\left(x_t, \epsilon_{t+1}\right)$ for T periods.
Step 2. Select each κth point to get a set P of n points $x_1, \ldots, x_n \in X \subseteq \mathbb{R}^d$.
Step 3. Estimate the density function $\widehat{g}\left(x_i\right) \approx g\left(x_i\right)$ for all $x_i \in P$.
Step 4. Remove all points for which the density is below η.

In Step 2, we include only each κth observation in the sample P to make random draws (approximately) independent. As far as Step 3 is concerned, there are various methods in statistics that can be used to estimate the density function from a given set of data; see Scott and Sain (2005) for a review. We use one such method, namely, a multivariate kernel algorithm with a normal kernel which estimates the density function in a point x as

$$\widehat{g}\left(x\right) = \frac{1}{n\left(2\pi\right)^{d/2}\overline{h}^d}\sum_{i=1}^{n}\exp\left[-\frac{D\left(x, x_i\right)}{2\overline{h}^2}\right], \tag{59}$$

where $\overline{h}$ is the bandwidth parameter and $D\left(x, x_i\right)$ is the distance between x and x_i. The complexity of Algorithm $\mathcal{A}^\eta$ is $O(n^2)$ because it requires us to compute pairwise distances between all sample points. Finally, in Step 3, we do not choose the density cutoff η but a fraction of the sample to be removed, Δ, which is related to η by

$p(\eta) = \int_{g(x) \geq \eta} g(x)\, dx = 1 - \Delta$. For example, $\Delta = 0.05$ means that we remove the 5% of the sample which has the lowest density.

4.3.2 Constructing an ε-Distinguishable Set of Points

Formally, we define an EDS as follows. Let (X, D) be a bounded metric space. A set P^{ε} consisting of points $x_1^{\varepsilon}, \ldots, x_M^{\varepsilon} \in X \subseteq \mathbb{R}^d$ is called ε-*distinguishable* if $D(x_i^{\varepsilon}, x_j^{\varepsilon}) > \varepsilon$ for all $1 \leq i, j \leq M : i \neq j$, where $\varepsilon > 0$ is a parameter. EDSs are used in mathematical literature that studies entropy; see Temlyakov (2011) for a review. This literature focuses on a problem of constructing an EDS that covers a given subset of $\mathbb{R}^d$ (such as a multidimensional hypercube). Judd et al. (2012) study a different problem: they construct an EDS for a given discrete set of points. To this purpose, they introduce the following algorithm.

```
(Algorithm $P^{\varepsilon}$): An algorithm for constructing an EDS.

Let $P$ be a set of $n$ simulated points $x_1, \ldots, x_n \in X \subseteq \mathbb{R}^d$.
Let $P^{\varepsilon}$ begin as an empty set, $P^{\varepsilon} = \{\varnothing\}$.
Step 1. Select $x_i \in P$. Compute $D(x_i, x_j)$ to all $x_j$ in $P$.
Step 2. Eliminate from $P$ all $x_j$ for which $D(x_i, x_j) < \varepsilon$.
Step 3. Add $x_i$ to $P^{\varepsilon}$ and eliminate it from $P$.

Iterate on Steps 1–3 until all points are eliminated from $P$.
```

The complexity of Algorithm P^{ε} is of the order $O(nM)$. When no points are eliminated from P, i.e., $M = n$, the complexity is quadratic, $O(n^2)$, which is the upper bound on cost. However, the number of points M in an EDS is bounded from above if X is bounded. This means that asymptotically, when $n \to \infty$, the complexity of Algorithm P^{ε} is linear, $O(n)$.

The distance between simulated points depends on measurement units of and correlation between variables (this affects the resulting EDS). Therefore, prior to using Algorithms $\mathcal{A}^{\eta}$ and P^{ε}, we normalize and orthogonalize the simulated data using the principal component (PC) transformation. As a measure of distance between two observations x_i and x_j, we use the Euclidean distance between their PCs, namely, $D(x_i, x_j) = \left[\sum_{\ell=1}^{d} (\mathrm{PC}_i^{\ell} - \mathrm{PC}_j^{\ell})^2\right]^{1/2}$, where all principal components $\mathrm{PC}^1, \ldots, \mathrm{PC}^d$ are normalized to have unit variance.

In Figure 5, we illustrate the described two-step procedure by way of an example of the one-agent model.

We simulate time series for capital and productivity level of 1,000,000 periods, and we select a sample of 10,000 observations by taking each 100th point (to make the draws independent); see Figure 5a. We orthogonalize the data using PC transformation and we normalize the PCs to a unit variance; see Figure 5b. We estimate the density function using the multivariate kernel algorithm with the standard bandwidth of $\overline{h} = n^{-1/(d+4)}$, and we remove from the sample 5% of simulated points in which the density was lowest;

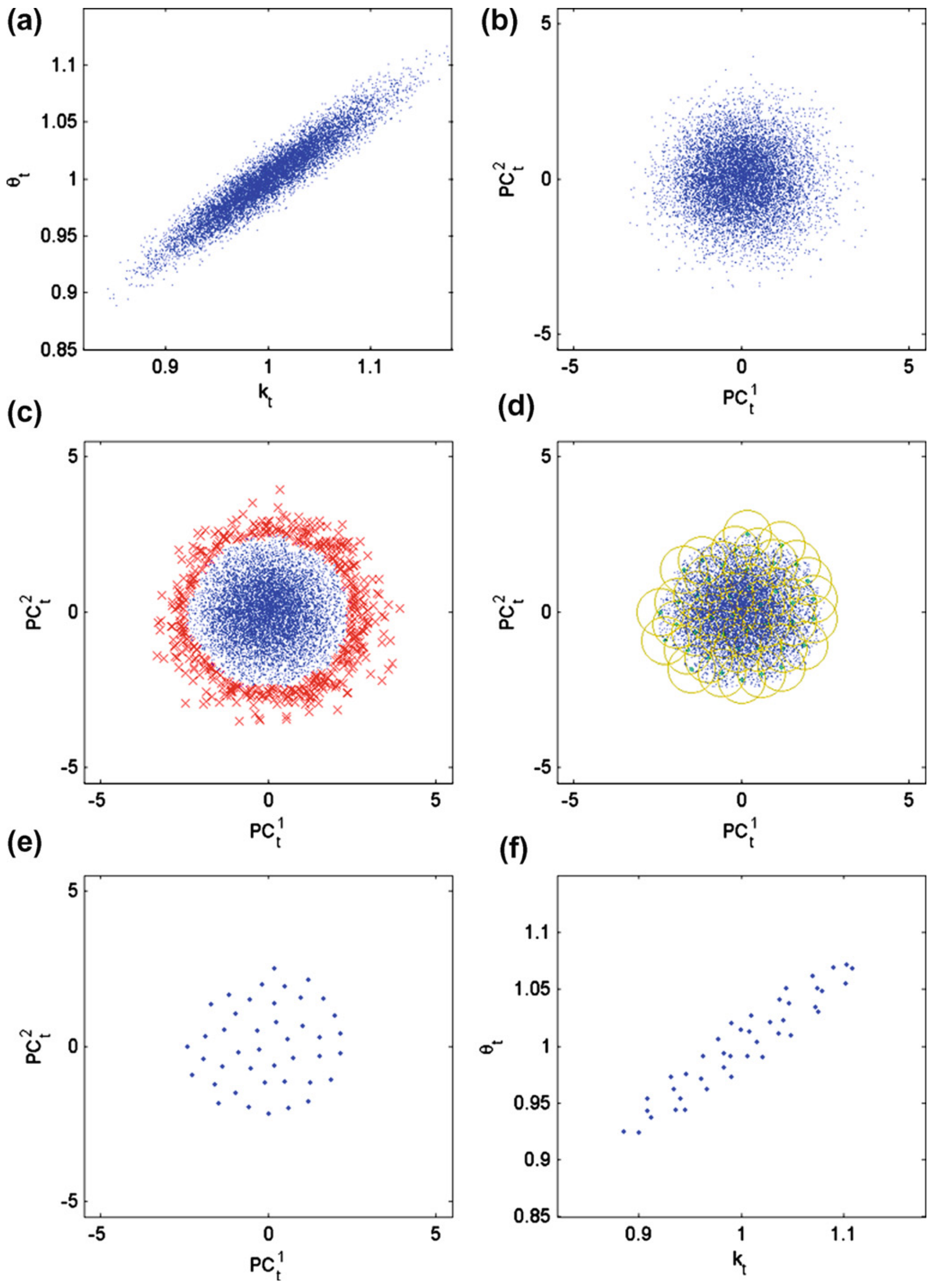


Figure 5 (a) Simulated points. (b) Principal components (PCs). (c) Density levels on PCs. (d) Constructing EDS. (e) EDS on PCs. (f) EDS on original data.

see Figure 5c. We construct an EDS; see Figure 5d. We plot such a set in the PC and original coordinates in Figure 5e and f, respectively. The constructed EDS appears to be evenly spaced.

More details about the EDS sets and their applications for solving dynamic economic models are given in Judd et al. (2012); in particular, this paper establishes the dispersion, cardinality, and degree of uniformity of the EDS grid. Also, Judd et al. (2012) perform the worst-case analysis and relate the results to recent mathematical literature on covering problems (see Temlyakov, 2011) and random sequential packing problems (see Baryshnikov et al., 2008). Judd et al. (2012) use the EDS algorithm to compute accurate quadratic solutions to a multicountry neoclassical growth model with up to 80 state variables, as well as to solve a large-scale new Keynesian model with a zero lower bound on nominal interest rates.

4.3.3 Other Grids on the Ergodic Set

We have described just one specific procedure for forming a discrete approximation to the essentially ergodic set of an economic model. Below, we outline other techniques that can be used for this purpose.

First, the two-step procedure outlined above has a complexity of order $O(n^2)$ because the kernel algorithm computes pairwise distances between all observations in the sample. This is not a problem for the size of applications we study in this chapter; however, it is expensive for large samples. Judd et al. (2012) describe an alternative procedure that has a lower complexity of $O(nM)$. The complexity is reduced by inverting the steps in the two-step procedure: first an EDS is constructed on all simulated points, and then the points in which the density is low are removed.

Second, one can use methods from cluster analysis to select a set of representative points from a given set of simulated points (instead of constructing an EDS set). We partition the simulated data into clusters (groups of closely located points) and we replace each cluster with one representative point. Examples of cluster grids are shown in Figure 6. The clustering techniques are demonstrated in Judd et al. (2012a) using two clustering methods: an agglomerative hierarchical and a K-means method. Such techniques were used to produce all the results in Judd et al. (2010, 2011d) and were found to lead to highly

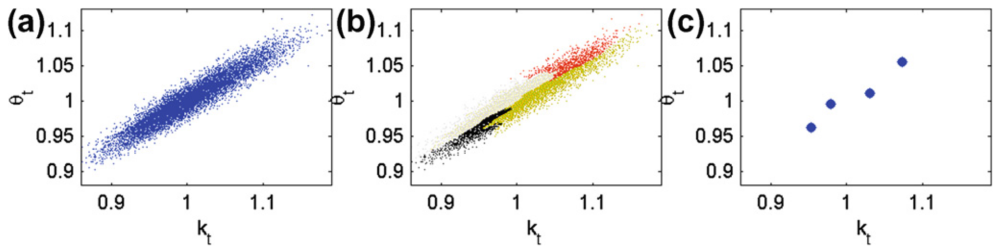


Figure 6 (a) The ergodic set. (b) Four clusters. (c) The clusters' centers.

accurate solutions. However, the cost of constructing cluster grids is higher than that of EDS grids. Judd et al. (2011d) and Aruoba and Schorfheide (2012) use clustering techniques to solve a new Keynesian model with a zero lower bound on nominal interest rates.

4.3.4 Numerical Illustration of the Accuracy of the Smolyak Method

Judd et al. (2012) compare the accuracy of solutions under the conventional Smolyak grid and EDS grids in the one-agent model with inelastic labor supply parameterized by (18). To make the results comparable, Judd et al. (2012) construct an EDS grid with the same target number of points (namely, 13) as in the Smolyak grid and use ordinary polynomials up to degree three under both the Smolyak and EDS grids. The accuracy is evaluated on two sets of points: one is a stochastic simulation of 10,000 points and the other is a set of 100×100 points that are uniformly spaced on the same solution domain as the one used by the Smolyak method. The results are shown in Table 8.

The accuracy ranking depends critically on the choice of points in which the accuracy is evaluated. The EDS method is significantly more accurate than the Smolyak method when the accuracy is evaluated on a stochastic simulation. This is because under an adaptive domain, we fit a polynomial function directly in an essentially ergodic set; while under the conventional hypercube grid, we fit a polynomial function in a larger hypercube domain and face a trade-off between the fit inside and outside the essentially ergodic set.

Nevertheless, the Smolyak method produces smaller maximum residuals than the EDS method when the accuracy is evaluated on a deterministic hypercube domain. This is because the EDS method is designed to be accurate in the ergodic set (simulated points), and its accuracy decreases away from the ergodic set while the conventional Smolyak (1963) method produces solutions that are uniformly accurate on a larger hypercube domain.

Table 8 Smolyak grid versus EDS grid in the one-agent model.[a]

	Accuracy on a simulation				**Accuracy on a hypercube**			
	Smolyak grid		**EDS grid**		**Smolyak grid**		**EDS grid**	
Polynomial degree	L_1	L_∞	L_1	L_∞	L_1	L_∞	L_1	L_∞
1st	−3.31	−2.94	−4.23	−3.31	−3.25	−2.54	−3.26	−2.38
2nd	−4.74	−4.17	−5.89	−4.87	−4.32	−3.80	−4.41	−3.25
3rd	−5.27	−5.13	−7.19	−6.16	−5.39	−4.78	−5.44	−4.11

[a] *Notes:* L_1 and L_∞ are, respectively, the average and maximum of absolute residuals across optimality condition and test points (in log10 units) on a stochastic simulation of 10,000 observations. The results are reproduced from Judd et al. (2012a), Table 2.

5. APPROXIMATION OF INTEGRALS

We study a class of methods that approximates multidimensional integrals by a weighted sum of integrands, i.e.,

$$E\left[G\left(\epsilon\right)\right]=\int_{\mathbb{R}^N}G\left(\epsilon\right)w\left(\epsilon\right)d\epsilon\approx\sum_{j=1}^{J}\omega_jG\left(\epsilon_j\right),\tag{60}$$

where $\epsilon\equiv\left(\epsilon^1,\ldots,\epsilon^N\right)^\top\in\mathbb{R}^N$ is a vector of uncorrelated variables; $G:\mathbb{R}^N\rightarrow\mathbb{R}$ is a continuous bounded function; w is a density function normalized by $\int_{\mathbb{R}^N}w\left(\epsilon\right)d\epsilon=1$; $\{\epsilon_j\}_{j=1,\ldots,J}$ is a set of nodes; and $\{\omega_j\}_{j=1,\ldots,J}$ is a set of weights normalized by $\sum_{j=1}^{J}\omega_j=1$.

We restrict attention to the case of random variables that are normally distributed with zero mean and unit standard deviation, i.e., $\epsilon\sim\mathcal{N}(0_N,I_N)$, where $0_N\equiv(0,\ldots,0)^\top\in\mathbb{R}^N$ is a vector of zeros and $I_N\in\mathbb{R}^{N\times N}$ is an identity matrix; and $w\left(\epsilon\right)=(2\pi)^{-N/2}$ $\exp\left[-\frac{1}{2}\epsilon^\top\epsilon\right]$ is a density function of a multivariate normal distribution. However, our analysis is easy to extend to other random processes.

Typically, there is a trade-off between accuracy and cost of integration methods: having more nodes leads to a more accurate approximation of integrals but is also more expensive. Hence, we describe several alternative integration methods that differ in accuracy and cost. While we describe the integration rules for the case of uncorrelated variables, at the end of the section we show how all the results can be generalized for the case of correlated variables using a Cholesky decomposition.

5.1 Gauss-Hermite Product Quadrature Rules

The Gauss-Hermite quadrature method provides a set of integration nodes $\{\epsilon_j\}_{j=1,\ldots,J}$ and weights $\{\omega_j\}_{j=1,\ldots,J}$ for approximation of a unidimensional version of integral (60),

$$\int_{\mathbb{R}}G\left(\epsilon\right)w\left(\epsilon\right)d\epsilon\approx\sum_{j=1}^{J}\omega_jG\left(\epsilon_j\right).\tag{61}$$

In Section 3, we show nodes and weights for one-, two-, and three-node Gauss-Hermite quadrature methods.

We can extend the unidimensional Gauss-Hermite quadrature rule to the multidimensional quadrature rule by way of a tensor-product rule:

$$\int_{\mathbb{R}^N}G\left(\epsilon\right)w\left(\epsilon\right)d\epsilon\approx\sum_{j_1=1}^{J_1}\cdots\sum_{j_N=1}^{J_N}\omega_{j_1}^1\cdots\omega_{j_N}^N\cdot G\left(\epsilon_{j_1}^1,\ldots,\epsilon_{j_N}^N\right),\tag{62}$$

where $\{\omega_{j_h}^h\}_{j_h=1,\ldots,J_h}$ and $\{\epsilon_{j_h}^h\}_{j_h=1,\ldots,J_h}$ are, respectively, weights and nodes in a dimension h derived from the unidimensional Gauss-Hermite quadrature rule (note that, in general, the number of nodes in one dimension, J_h, can differ across dimensions). The total number of nodes is given by the product $J_1 J_2 \cdots J_N$. The total number of nodes grows exponentially with the dimensionality N; under the assumption that $J_h = J$ for all dimensions, the total number of nodes is J^N.

Consider a two-node Gauss-Hermite quadrature rule for a model with two shocks; then we have $(\theta_j^h)' = \theta^\rho \exp(\epsilon_j^h)$; see Figure 7a.

Example 6. Let $\epsilon^h \sim \mathcal{N}(0,1)$, $h = 1, 2$ be uncorrelated normally distributed variables. A two-node Gauss-Hermite quadrature rule, denoted by $Q(2)$, has four nodes

value\node	***j* = 1**	***j* = 2**	***j* = 3**	***j* = 4**
ϵ_j^1	1	−1	1	−1
ϵ_j^2	1	1	−1	−1
ω_j	1/4	1/4	1/4	1/4

The expectation of a function $G(\epsilon)$ can be approximated as

$$E[G(\epsilon)] \approx \frac{1}{4}[G(1,1) + G(-1,1) + G(1,-1) + G(-1,-1)].$$

Under a J-node Gauss-Hermite product rule, the number of nodes grows rapidly with the number of exogenous random variables, N. Even if there are just two nodes for each random variable, the total number of nodes is prohibitively large for large N; for example, if $N = 100$, we have $2^N \approx 10^{30}$ nodes. This makes product rules impractical for high-dimensional applications.

There is a specific case of Gauss-Hermite quadrature rule, a one-node quadrature rule $Q(1)$, which is the cheapest possible deterministic integration method; see Figure 7b.

Example 7. Let $\epsilon^h \sim \mathcal{N}(0,1)$, $h = 1, 2$ be uncorrelated normally distributed variables. A one-node Gauss-Hermite quadrature rule, denoted by $Q(1)$, has one node

value\node	***j* = 1**
ϵ_j^1	0
ϵ_j^2	0
ω_j	1

The expectation of a function $G(\epsilon)$ can be approximated as

$$E[G(\epsilon)] \approx G(0,0).$$

This rule has just one node independently of the number of shocks N. In the context of one- and multicountry growth models, Maliar et al. (2011) find that this simple rule leads to just a slightly lower accuracy of solutions than the most accurate deterministic methods.

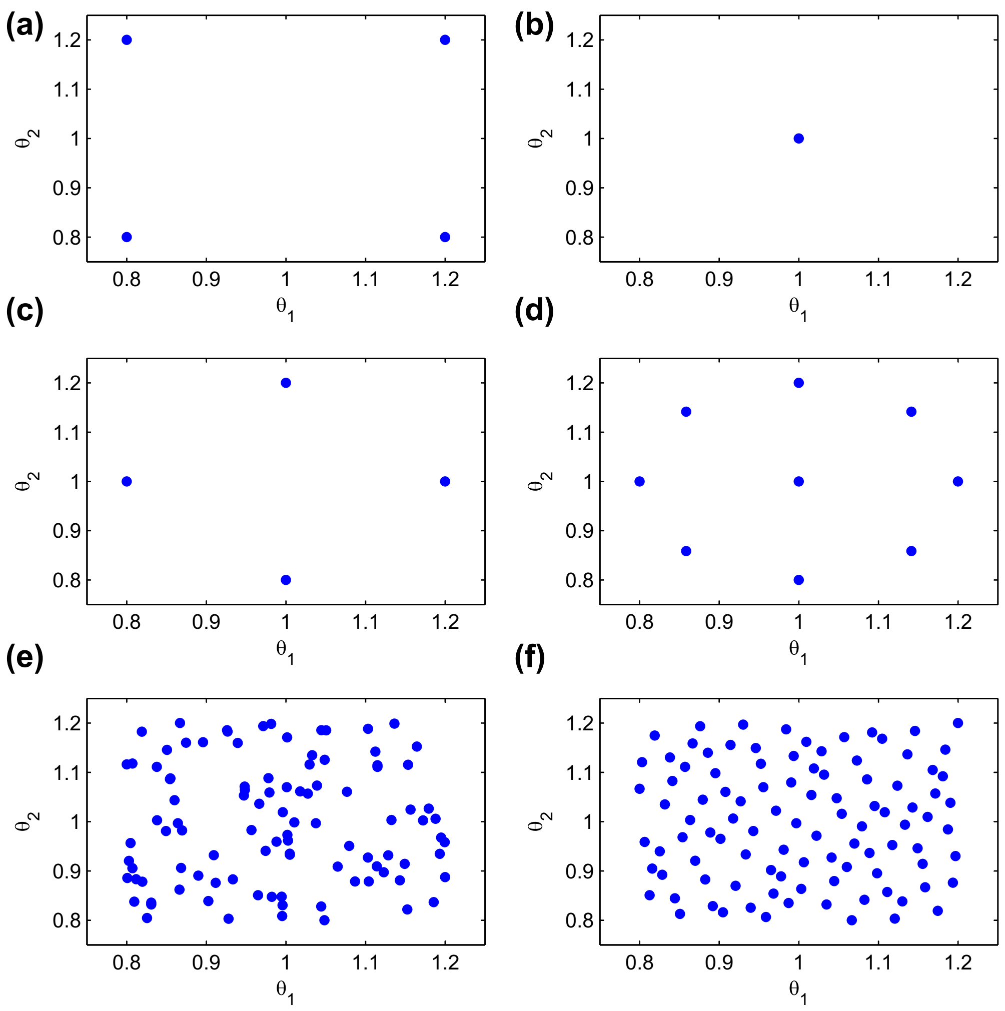


Figure 7 (a) Quadrature rule Q(2). (b) Quadrature rule Q(1). (c) Monomial rule M1. (d) Monomial rule M2. (e) Monte Carlo draws. (f) Sobol grid.

5.2 Monomial Rules

Monomial integration rules are nonproduct: they construct a relatively small set of nodes distributed in some way within a multidimensional hypercube. The computational expense of monomial rules grows only polynomially with the dimensionality of the problem, which makes them feasible for problems with large dimensionality. Monomial rules are introduced to economic literature in Judd (1998). Monomial formulas are used for approximating integrals by all global methods focusing on large-scale applications, e.g., Judd et al. (2010, 2011a,b, 2012), Malin et al. (2011), Maliar et al. (2011), and

Pichler (2011). The last paper also uses monomial rules for constructing a grid of points for finding a solution. Finally, Juillard and Villemot (2011) use monomial formulas for implementing the accuracy checks.

5.2.1 Monomial Rule M1 with 2N Nodes

Consider the following simple example of the monomial rule with $2N$ nodes which we denote $M1$:

$$E\left[G(\epsilon)\right] \approx \frac{1}{2N}\sum_{h=1}^{N}\left[G\left(R\iota^h\right)+G\left(-R\iota^h\right)\right], \tag{63}$$

where $\epsilon \sim \mathcal{N}(0_N, I_N)$ and $R \equiv \sqrt{N}$, and $\iota^h \in \mathbb{R}^N$ is a vector whose hth element is equal to one and the remaining elements are equal to zero, i.e., $\iota^h \equiv (0,\ldots,1,\ldots,0)^\top$.

$M1$ constructs nodes by considering deviations of each random variable holding the other random variables fixed to their expected values. Let us illustrate this rule with a two-dimensional example.

Example 8. Let $\epsilon^h \sim \mathcal{N}(0,1)$, $h = 1, 2$ be uncorrelated normally distributed variables. A monomial rule $M1$ has four nodes; see Figure 7c

value\node	***j* = 1**	***j* = 2**	***j* = 3**	***j* = 4**
ϵ_j^1	$\sqrt{2}$	$-\sqrt{2}$	0	0
ϵ_j^2	0	0	$\sqrt{2}$	$-\sqrt{2}$
ω_j	1/4	1/4	1/4	1/4

The expectation of a function $G(\epsilon)$ is approximated as

$$E\left[G(\epsilon)\right] = \frac{1}{4}\left[G\left(\sqrt{2},0\right)+G\left(-\sqrt{2},0\right)+G\left(0,\sqrt{2}\right)+G\left(0,-\sqrt{2}\right)\right].$$

The integration nodes produced by this monomial rule are shown in Figure 7c. Since the cost of $M1$ increases with N only linearly, this rule is feasible for the approximation of integrals with very large dimensionality. For example, with $N = 100$, the total number of nodes is just $2N = 200$.

5.2.2 Monomial Rule M2 with $2N^2 + 1$ Nodes

Many other monomial rules are available in the mathematical literature; see Stroud (1971), pp. 315–329, and Judd (1998), p. 275, for a discussion. Below, we show a monomial rule with $2N^2+1$ nodes, denoted by $M2$,

$$E\left[G(\epsilon)\right] \approx \frac{2}{2+N}G(0,\ldots,0) + \frac{4-N}{2(2+N)^2}\sum_{h=1}^{N}\left[G\left(R\iota^h\right)+G\left(-R\iota^h\right)\right] + \frac{1}{(N+2)^2}\sum_{h=1}^{N-1}\sum_{s=h+1}^{N}G\left(\pm D\iota^h \pm D\iota^s\right), \tag{64}$$

where $\epsilon \sim \mathcal{N}(0_N, I_N)$, $R \equiv \sqrt{2+N}$ and $D \equiv \sqrt{\frac{2+N}{2}}$. A two-dimensional example of this rule is illustrated in the example below.

Example 9. Let $\epsilon^h \sim \mathcal{N}(0,1)$, $h = 1,2$ be uncorrelated normally distributed variables. A monomial rule $M1$ has nine nodes; see Figure 7d.

value\node	***j* = 1**	***j* = 2**	***j* = 3**	***j* = 4**	***j* = 5**	***j* = 6**	***j* = 7**	***j* = 8**	***j* = 9**
ϵ_j^1	0	2	-2	0	0	$\sqrt{2}$	$\sqrt{2}$	$-\sqrt{2}$	$-\sqrt{2}$
ϵ_j^2	0	0	0	2	-2	$\sqrt{2}$	$-\sqrt{2}$	$\sqrt{2}$	$-\sqrt{2}$
ω_j	$\frac{1}{2}$	1/16	1/16	1/16	1/16	1/16	1/16	1/16	1/16

The expectation of a function $G(\epsilon)$ can be approximated as

$$E\left[G(\epsilon)\right] \approx \frac{1}{2}G(0,0) + \frac{1}{16}\left[G(2,0)+G(-2,0)+G(0,2)+G(0,-2)\right] + \frac{1}{16}\left[G\left(\sqrt{2},\sqrt{2}\right)+G\left(\sqrt{2},-\sqrt{2}\right) + G\left(-\sqrt{2},\sqrt{2}\right)+G\left(-\sqrt{2},\sqrt{2}\right)\right].$$

The cost of $M2$ grows quadratically with N but it is still feasible with high dimension. For example, with $N = 100$, the total number of nodes is just $2N^2+1 = 20{,}001$, which is still manageable.

5.3 Monte Carlo Integration Method

A J-node Monte Carlo integration method, denoted by $MC(J)$, draws J shocks, $\{\epsilon_j\}_{j=1,\ldots,J}$ and approximates the integral by the average of the integrand's values across the shocks

$$E\left[G(\epsilon)\right] = \frac{1}{J}\sum_{j=1}^{J}G\left(\epsilon_j\right). \tag{65}$$

See Figure 7e for a Monte Carlo method with 100 realizations. A one-node version of the Monte Carlo integration method is used in Marcet's (1988) PEA (namely, the future expectation of a random variable is approximated by one realization of the variable). As we argued in Section 3, this method is inaccurate and limits the overall accuracy of solutions produced by numerical solution methods.

5.4 Quasi-Monte Carlo Integration Methods

Quasi-Monte Carlo methods construct a set of nodes that are uniformly spaced in a given area of the state space. Normally, the nodes are constructed in a multidimensional hypercube. It is possible to construct a uniformly spaced set of nodes in a hypercube using a tensor-product grid of unidimensional uniformly spaced nodes, but this technique is subject to the curse of dimensionality. Quasi-Monte Carlo methods construct a uniformly spaced grid of nodes using nonproduct techniques. The number of grid points is controlled by a researcher. The accuracy of quasi-Monte Carlo methods is generally higher than that of Monte Carlo methods. Applications of quasi-Monte Carlo sequences to solving dynamic economic models are studied in Rust (1997); see also Geweke (1996) for a discussion.

Our presentation relies on the standard notion of uniformity in the literature, which is the discrepancy of a given set of points from the uniformly distributed set of points; see Niederreiter (1992), p. 14. Let P be a set consisting of points $x_1, \ldots, x_n \in X \subseteq \mathbb{R}^d$, and let J be a family of Lebesgue-measurable subsets of X. The discrepancy of P under J is given by $\mathcal{D}_n(P; \mathcal{J}) = \sup_{J \in \mathcal{J}} \left| \frac{C(P;J)}{n} - \lambda(J) \right|$, where $C(P; J)$ counts the number of points from P in J, and $\lambda(J)$ is a Lebesgue measure of J. Thus, $\mathcal{D}_n(P; \mathcal{J})$ measures the discrepancy between the fraction of points $\frac{C(P;J)}{n}$ contained in J and the fraction of space $\lambda(J)$ occupied by J. If the discrepancy is low, $\mathcal{D}_n(P; \mathcal{J}) \approx 0$, the distribution of points in X is close to uniform.

The measure of discrepancy commonly used in the literature is the star discrepancy. The star discrepancy $\mathcal{D}_n^*(P; \mathcal{J})$ is defined as the discrepancy of P over the family J generated by the intersection of all subintervals of $\mathbb{R}^d$ of the form $\Pi_{i=1}^d [-\infty, v_i)$, where $v_i > 0$. Let S be a sequence of elements on X, and let $x_1, \ldots, x_n \in X \subseteq \mathbb{R}^d$ be the first n terms of S. Niederreiter (1992), p. 32, suggests calling a sequence S low discrepancy if $\mathcal{D}_n^*(S; \mathcal{J}) = O\left(n^{-1} (\log n)^d\right)$, i.e., if the star discrepancy converges to zero asymptotically at a rate at least of order $n^{-1} (\log n)^d$.

The star discrepancy of points which are randomly drawn from a uniform distribution $[0, 1]^d$ also converges to zero asymptotically, $\lim_{n \to \infty} \mathcal{D}_n^*(S; \mathcal{J}) = 0$, a.e. but its rate of convergence $(\log \log n)^{1/2} (2n)^{-1/2}$ is far lower than that of low-discrepancy sequences. This rate of convergence follows directly from the law of iterated logarithm by Kiefer (1961); see Niederreiter (1992), pp. 166–168, for a general discussion on how to use Kiefer's (1961) results for assessing the discrepancy of random sequences.

The simplest example of a sequence, equidistributed on $[0, 1]$, is a Weyl sequence $x_n = \{n\vartheta\}$, where ϑ is irrational and $\{x\}$ is a fractional part of x. For example, let $\vartheta = \sqrt{2}$. Then, we have $x_1 = \left\{1 \cdot \sqrt{2}\right\} = \{1.4142\} = 0.4142$, $x_2 = \left\{2 \cdot \sqrt{2}\right\} = \{2.8284\} = 0.8284$, etc. This sequence can be generalized to a d-dimensional case using d distinct and linearly independent irrational numbers $\vartheta_1, \ldots, \vartheta_d$; this yields a sequence $x_n = (\{n\vartheta_1\}, \ldots, \{n\vartheta_d\})$ whose points are equidistributed over the $[0, 1]^d$. There is a variety of other low-discrepancy sequences in the literature including Haber, Baker, Niederreiter, and Sobol sequences; see Niederreiter (1992) for a review. As an example, we draw a two-dimensional Sobol sequence with 100 points; see Figure 7f. Thus, to perform integration, we use a low-discrepancy sequence in the same way as we use a number of Monte Carlo random draws.

5.5 Nonparametric Kernel-Density Methods

Suppose we observe a set of data $\{x_i, G_i\}_{i=1,\ldots,I}$, and we must compute the conditional expectation function $E\left[G|x_j\right]$, for some $j \in \{1, \ldots, I\}$. A general class of nonparametric estimators can be written as

$$E\left[G|x_j\right] = \sum_{i=1}^{I} w_{i,j} G_i, \tag{66}$$

where $w_{i,j} = W\left(D\left(x_i, x_j\right)\right)$ is a weight function that satisfies $\sum_{i=1}^{I} w_{i,j} = 1$ and $W' < 0$, with $D\left(x_i, x_j\right)$ being a distance between points x_i and x_j. In other words, expectation $E\left[G|x_j\right]$ is given by a weighted sum of all observations in the sample, and a weight $w_{i,j}$ placed on each observation x_i depends on how far this particular observation is from observation x_j in which the expectation is evaluated. Nadaraya (1964) and Watson (1964) propose to construct the weights using the multivariate normal kernel

$$w_{i,j} = \frac{\exp\left[-\frac{D\left(x_i, x_j\right)}{2\overline{h}^2}\right]}{\sum_{i=1}^{I} \exp\left[-\frac{D\left(x_i, x_j\right)}{2\overline{h}^2}\right]}, \tag{67}$$

where $\overline{h}$ is a parameter which effectively represents the width of the interval in which the expectation is estimated. There are many other alternatives for the weight function in the literature on nonparametric estimation; see Pagan and Ullah (1999). The difficult problem in statistics is to choose the width parameter $\overline{h}$. There are two procedures in the literature for identifying this parameter, namely, *cross-validation* and *plug-in*. Typically, an iterative procedure is used: assume some $\overline{h}$, compute the expectation function, and check $\overline{h}$ by using some criterion; and iterate until a fixed point $\overline{h}$ is obtained. A recent example of application of nonparametric methods for solving dynamic economic models is Jirnyi and Lepetyuk (2011).

5.6 Approximating a Markov Process Using a Markov Chain

There are other combinations of techniques of potential interest that are not described above. For example, Tauchen (1986) shows how to accurately approximate unidimensional integrals by discretizing an autoregressive process into a finite-state Markov chain; and Tauchen and Hussey (1991) generalize their analysis to multidimensional cases. Such a discretization can be performed using nonproduct rules that are tractable in high-dimensional applications. This discretization method can be combined with other computational techniques surveyed in the present chapter to construct numerical solution methods that are tractable in high-dimensional applications.

5.7 Correlated Shocks and Cholesky Decomposition

We finally show how to extend the described integration methods to the case when the shocks are correlated. To be specific, we evaluate a multidimensional integral of the form

$$E\left[G\left(\epsilon\right)\right]=\int_{\mathbb{R}^N} G\left(\epsilon\right)w\left(\epsilon\right)d\epsilon, \tag{68}$$

where $\epsilon \equiv \left(\epsilon^1,\ldots,\epsilon^N\right)^\top \in \mathbb{R}^N$ follows a multivariate normal distribution, $\epsilon \sim \mathcal{N}\left(\mu,\Sigma\right)$, with $\mu \equiv \left(\mu^1,\ldots,\mu^N\right)^\top \in \mathbb{R}^N$ being a vector of means and $\Sigma \in \mathbb{R}^{N\times N}$ being a variance-covariance matrix, and $w\left(\epsilon\right)$ is a density function of the multivariate Normal distribution,

$$w\left(\epsilon\right)=\left(2\pi\right)^{-N/2}\det\left(\Sigma\right)^{-1/2}\exp\left[-\frac{1}{2}\left(\epsilon-\mu\right)^\top\Sigma^{-1}\left(\epsilon-\mu\right)\right], \tag{69}$$

with $\det\left(\Sigma\right)$ denoting the determinant of Σ.

If random variables $\epsilon^1,\ldots,\epsilon^N$ are correlated, we must rewrite the integral in terms of uncorrelated variables prior to numerical integration. Given that Σ is symmetric and positive-definite, it has a Cholesky decomposition, $\Sigma=\Omega\Omega^\top$, where Ω is a lower triangular matrix with strictly positive diagonal entries. The Cholesky decomposition of Σ allows us to transform correlated variables ϵ into uncorrelated ν with the following linear change of variables:

$$\nu=\Omega^{-1}\left(\epsilon-\mu\right). \tag{70}$$

Note that $d\epsilon=\det\left(\Omega\right)d\nu$. Using (70) and taking into account that $\Sigma^{-1}=\left(\Omega^{-1}\right)^\top\Omega^{-1}$ and that $\det\left(\Sigma\right)=\left[\det\left(\Omega\right)\right]^2$, we obtain

$$\int_{\mathbb{R}^N} G\left(\epsilon\right)w\left(\epsilon\right)d\epsilon=\left(2\pi\right)^{-N/2}\int_{\mathbb{R}^N} G\left(\Omega\nu+\mu\right)\exp\left(-\frac{\nu^\top\nu}{2}\right)d\nu$$

$$= \int_{\mathbb{R}^N} G(\Omega \nu + \mu) w(\nu)$$
$$\approx \sum_{j=1}^{J} \omega_j G(\Omega \nu_j + \mu) \equiv \sum_{j=1}^{J} \omega_j G(\epsilon_j), \qquad (71)$$

where $\nu \sim \mathcal{N}(0_N, I_N)$, with $0_N \equiv (0, \ldots, 0)^\top \in \mathbb{R}^N$. Comparing (60) and (71) shows that with correlated shocks, we can use the same formula as with uncorrelated normalized shocks with the only modification that instead of a node implied by a given integration rule we use a transformed node $\epsilon_j \equiv \Omega \nu_j + \mu$, where ν_j is a node for uncorrelated shocks.

5.8 Numerical Illustration of the Importance of the Integration Method

We compare the simulation-based and nonparametric methods with the parametric ones. As we will see, the performance of simulation-based integration is poor, and the performance of the nonparametric integration methods is even poorer. The reason is that we disregard information about the true density function by replacing it with the estimated (noisy) density function. Econometricians use the estimated density function because the true one is not available. However, in economic models we define the process for shocks and we do know density functions. We can do much better by using deterministic (quadrature and monomial) rules rather than simulation-based methods. We provide an illustration below.

5.8.1 Monte Carlo Integration

The Monte Carlo integration method is feasible in high-dimensional applications (because we control the number of draws) but its accuracy is generally far lower than that of the deterministic integration methods. Judd et al. (2011b) study a version of the one-agent model with inelastic labor supply under assumption (18). They assess the performance of GSSA based on the Monte Carlo method, $MC(J)$, with $J = 1$ and $J = 2,000$ random nodes in each simulated point and, the Gauss-Hermite quadrature method, $Q(J)$, with $J = 1$ and $J = 2$ deterministic nodes. The results are provided in Table 9. Under Monte Carlo integration, high-degree polynomials do not necessarily lead to more accurate solutions than low-degree polynomials because the overall accuracy is dominated by errors produced by an integration method. Surprisingly, a one-node Gauss-Hermite quadrature method, $Q(1)$, leads to more accurate solutions than a 2,000-node Monte Carlo method $MC(2,000)$. A two-node Gauss-Hermite quadrature method, $Q(2)$, produces very accurate solutions with the residuals of order 10^{-8}. Increasing the number of quadrature nodes does not visibly improve the accuracy. Monomial integration rules deliver the same (high) accuracy levels as the quadrature rules; see Judd et al. (2011b) for more details.

Table 9 Monte Carlo integration and quadrature integration in the one-agent model.[a]

Polynomial degree	MC(1)			MC(2000)			Q(1)			Q(2)		
	$T = 10,000$											
	L_1	L_∞	CPU	L_1	L_∞	CPU	L_1	L_∞	CPU	L_1	L_∞	CPU
1st	−4.26	−3.37	1	−4.40	−3.48	1236	−4.35	−3.37	15	−4.36	−3.37	16
2nd	−4.42	−3.69	11	−6.04	−4.93	1711	−5.99	−4.94	32	−6.13	−4.92	27
3rd	−4.32	−3.37	25	−6.15	−5.07	2198	−6.32	−5.90	45	−7.48	6.01	35
4th	−4.31	−2.98	47	−6.08	−4.71	3337	−6.32	−6.18	53	−8.72	−7.10	44
5th	−4.23	−3.30	80	−6.07	−4.70	4551	−6.32	−6.18	62	−8.91	−7.26	51
	$T = 100,000$											
1st	−4.39	−3.40	4				−4.36	−3.40	117	−4.37	−3.39	113
2nd	−4.87	−3.96	79				−6.03	−4.94	281	−6.16	−4.94	188
3rd	−4.86	−3.60	184	Ran out of memory			−6.32	−5.93	387	−7.52	−6.04	260
4th	−4.72	−3.43	341				−6.32	−6.19	470	−8.78	−7.18	335
5th	−4.71	−3.44	623				−6.32	−6.19	548	−8.98	−7.35	406

[a] *Notes:* L_1 and L_∞ are, respectively, the average and maximum of absolute residuals across optimality conditions and test points (in log10 units). CPU is the time necessary for computing a solution (in seconds). T is the simulation length. $MC(J)$ and $Q(J)$ denote J-node Monte Carlo and quadrature integration methods, respectively. These results are reproduced from Judd et al. (2011b), Table 2.

The above results illustrate a slow $\sqrt{J}$ rate of convergence of the Monte Carlo method. To achieve high accuracy of solutions under the Monte Carlo method, we must increase the sample size by many orders of magnitude, which is infeasible in practice.

5.8.2 Nonparametric Kernel-Density Integration Method

It might be tempting to solve for decision functions using a fully nonparametric solution method. The advantage of this approach is that we do not need to make any assumptions about a functional form for decision functions. Instead, we just simulate the model, compute the expectation using the simulated data, and let the model itself choose what is the best possible approximation. Unfortunately, such nonparametric methods are not competitive in the studied applications due to their high computational expense and low rate of convergence (equivalently, low accuracy).

As an illustration, we assess the performance of a nonparametric kernel-density method in the context of the model with inelastic labor supply under the assumptions (18). To give the best chance to the nonparametric method, we choose the width parameter, $\overline{h}$, optimally, namely, we compute an accurate parametric solution, and we find the value of $\overline{h}$ for which conditional expectations delivered by the parametric and nonparametric methods are as close as possible on a set of the simulated points considered (this technique is not feasible in statistics where the true expectation function is unknown). We find the

Table 10 Nonparametric kernel-density integration method.[a]

Polynomial degree	**T = 1000**			**T = 3000**			**T = 10,000**		
	L_1	**L_∞**	**CPU**	**L_1**	**L_∞**	**CPU**	**L_1**	**L_∞**	**CPU**
1st	−3.35	−1.94	2429.3	−3.31	−1.87	11373.3	−3.62	−2.21	62051.8
2nd	−3.37	−2.22		−3.32	−1.97		−3.62	−2.23	
3rd	–	–		–	–		−3.64	−1.98	
4th	–	–		–	–		−3.64	−1.82	
5th	−3.31	−1.04		−3.16	−1.68		−3.63	−1.75	

[a] *Notes:* L_1 and L_∞ are, respectively, the average and maximum of absolute approximation errors across optimality conditions and test points (in log10 units). CPU is the time necessary for computing a solution (in seconds). T is the sample size.

solutions under $T = 1,000$, $T = 3,000$, and $T = 10{,}000$. The solution procedure is expensive since we have to find an expectation in each simulation point by averaging up all simulated points. Furthermore, there is no natural way to assess the accuracy of nonparametric solutions. We therefore take a time-series solution delivered by the nonparametric method, regress it on an ordinary polynomial function of degrees from one to five, and use the resulting parametric function to run accuracy checks. The results are shown in Table 10. As we see, the performance of the nonparametric integration method is poor. The running time is large and increases rapidly with the simulation length. Under $T = 1,000$ and $T = 3,000$, the accuracy is so low that the solution is numerically unstable in simulation. Under $T = 10{,}000$, the numerical stability is achieved but the accuracy is still very low compared to parametric solutions reported in Table 9.

The rate of convergence of the Nadaraya-Watson method we use is always slower than that of the Monte Carlo method and decreases with the number of random variables with respect to which the expectation is evaluated. When there are k random variables and J simulated points, and when we use the same value of width $\overline{h}$ for all random variables, the convergence rate is of the order $J^{2/(k+4)}$; see Bierens (1994), Theorem 10.2.2. For our example, $k = 1$ and hence, the convergence rate is of the order $J^{2/5}$. There are other nonparametric estimators, e.g., local linear regressions, but their convergence properties are also poor.

6. DERIVATIVE-FREE OPTIMIZATION METHODS

In this section, we focus on solving systems of nonlinear equations that arise in the context of the Euler equation methods. Our main tool is a simple derivative-free fixed-point iteration (FPI) method. Assume that we need to solve an equation $x = F(x)$ (any equation can be represented in this form). We implement the following iterative procedure.

(FPI): Fixed-point iteration.

```
Initialization. Fix initial guess $x^{(0)}$, a norm $\|\cdot\|$, and a convergence criterion $\varpi$.
Step 1. On iteration $i$, compute $\widehat{x} = F\left(x^{(i)}\right)$.
Step 2. If $\left\|\widehat{x} - x^{(i)}\right\| < \varpi$, then stop.
    Otherwise, set $x^{(i+1)} = \widehat{x}$ and go to Step 1.
```

FPI has two advantages over Newton-style methods. First, FPI does not require us to compute derivatives (such as Jacobian or Hessian); as a result, its cost does not increase considerably with the dimensionality of the problem. Second, FPI can iterate on any object x at once (variable, vector of variables, matrix, time series, vector of coefficients, etc.), while Newton-style methods compute a solution point-by-point and are more difficult to vectorize or parallelize.

The drawback of the FPI method is that it may fail to converge. However, damping can often help us to achieve the convergence. Instead of a full updating at the end of the iteration $x^{(i+1)} = \widehat{x}$, we use a partial updating $x^{(i+1)} = \xi\widehat{x} + (1-\xi)\, x^{(i)}$ for some $\xi \in (0,1]$. By varying a damping parameter ξ, we can control how much x changes from one iteration to another and avoid explosive paths. FPI with damping systematically converged in all our numerical experiments. Note that we need to adjust the convergence criterion ϖ to the damping parameter ξ, for example, by replacing ϖ with $\frac{\varpi}{\xi}$; otherwise, the convergence will critically depend on the value of ξ assumed and can be trivially achieved by using a very small value of ξ. In the remainder of the section, we illustrate the applications of FPI by way of examples.

6.1 Separation of Intertemporal and Intratemporal Choices

Equilibrium conditions of dynamic economic models are not symmetric in a sense that some of them contain only current variables (known at t), while the others contain both current and future variables (unknown at t); we refer to these two groups of conditions as the *intratemporal and intertemporal choice conditions*, respectively. Maliar et al. (2011) show how to exploit this asymmetry of the equilibrium conditions to solve for equilibrium more efficiently. Specifically, they find solutions to the intertemporal choice conditions in terms of state contingent functions; however, they find solutions to the intratemporal choice conditions in terms of equilibrium quantities. Below, we show how this construction can be implemented in the context of model (1)–(3).

6.1.1 Intratemporal Choice FOCs

Let us collect the intratemporal choice conditions of problem (1)–(3) that contain only the variables known at time t:

$$u_2(c,\ell) = u_1(c,\ell)\,\theta f_\ell(k,\ell), \tag{72}$$

$$c + k' = (1-\delta)\,k + \theta f(k,\ell). \tag{73}$$

The above system contains five variables, namely, c, ℓ, k', k, θ. We can solve it with respect to two variables if the remaining three variables are fixed. In the benchmark case, we fix (k, θ) and k', and we solve for c, ℓ.

6.1.2 Intertemporal Choice FOCs

The FOC of problem (1)–(3) with respect to capital contains both current and future variables

$$u_1(c, \ell) = \beta E\left\{u_1\left(c', \ell'\right)\left[1 - \delta + \theta' f_1\left(k', \ell'\right)\right]\right\}. \tag{74}$$

Note that (74) cannot be treated in the same way as (72), (73). In particular, fixing c, ℓ is not sufficient for computing k' from (74). We need a method that allows us to infer the values of future variables c', ℓ' in all possible future economy's states (k', θ').

To find a solution to the intertemporal choice conditions, we use a Markov structure of the decision functions, namely, we parameterize $K(k, \theta)$ with a flexible functional form $\widehat{K}(k, \theta; b)$, and we solve for a coefficient vector b that approximately satisfies (74).

6.1.3 A Global Euler Equation Method with a Separation of Intertemporal and Intratemporal Choices

We now outline a global Euler equation method that finds a solution to (1)–(3) by separating the intertemporal and intratemporal choice conditions.

The inner and outer loops in the algorithm correspond to the intratemporal and intertemporal planner's choice, respectively. Note that planner has to solve for the intratemporal choice $J + 1$ times, one time at present and J times in all possible future states (integration nodes). The problems solved at Step 3a and 3b are identical. We next discuss how to find a solution to the intratemporal and intertemporal choice conditions in more detail.

6.2 The Intratemporal Choice Quantities

Consider system (72), (73) under utility and production function parameterizations in (17),

$$B\ell^{-\mu} = c^{-\gamma}(1-\alpha)\theta k^{\alpha}(1-\ell)^{-\alpha}, \tag{75}$$

$$c = \theta k^{\alpha}\ell^{1-\alpha} + (1-\delta)k - k'. \tag{76}$$

This system has a small number of equations, and, in principle, Newton solvers can be used to solve for c and ℓ in terms of (k, θ, k'). However, we must solve this system a large number of times, namely, for each value of $\left(k_m, \theta_m, k'_m\right)$ on the grid, and for J future states $\left(k'_m, \theta'_{m,j}, k''_{m,j}\right)$ corresponding to each grid point, i.e., $M \times (J + 1)$ times in total for every iteration on the outer loop. The cost of solving this system repeatedly can be large, especially in high-dimensional problems.

Maliar et al. (2011) develop a derivative-free solver called *iteration-on-allocation*. This method solves a system of nonlinear equations by performing FPI on the intratemporal

(EEM2): A global Euler equation method with inner and outer loops.

Step 1. Choose a grid $\{k_m, \theta_m\}_{m=1,\ldots,M}$ on which decision functions are approximated. Choose a flexible functional form $\widehat{K}(\cdot; b)$ for approximating decision function K.
Step 2. Choose nodes, ϵ_j, and weights, $\omega_j, j = 1, \ldots, J$, for approximating integrals. Compute next-period productivity $\theta'_{m,j} = \theta_m^{\rho} \exp(\epsilon_j)$ for all j, m.
Step 3. Solve for b, $\{c_m, \ell_m\}$, and $\left\{c'_{m,j}, \ell'_{m,j}\right\}$ for all j such that
–*3a (inner loop):* The quantities $\{c_m, \ell_m\}$ solve:

$$u_2(c_m, \ell_m) = u_1(c_m, \ell_m)\,\theta_m f_2(k_m, \ell_m),$$
$$c_m + k'_m = \theta_m f(k_m, \ell_m) + (1-\delta)k_m,$$

given $k'_m = \widehat{K}(k_m, \theta_m; b)$.
–*3b:* In each state $j = 1, \ldots, J$, the quantities $\left\{c'_{m,j}, \ell'_{m,j}\right\}$ solve:

$$u_2\left(c'_{m,j}, \ell'_{m,j}\right) = u_1\left(c'_{m,j}, \ell'_{m,j}\right)\theta'_{m,j} f_2\left(k'_m, \ell'_{m,j}\right),$$
$$c'_{m,j} + k''_{m,j} = \theta'_{m,j} f\left(k'_m, \ell'_{m,j}\right) + (1-\delta)k'_m,$$

given $k'_m = \widehat{K}(k_m, \theta_m; b)$ and $k''_{m,j} = \widehat{K}\left(k'_m, \theta'_{m,j}; b\right)$.
–*3c (outer loop):* The function $k' = \widehat{K}(k, \theta; b)$ solves:

$$u_1(c_m, \ell_m) = \beta \sum_{j=1}^{J} \omega_j \cdot \left[u_1\left(c'_{m,j}, \ell'_{m,j}\right)\left(1 - \delta + \theta'_{m,j} f_1\left(k'_m, \ell'_{m,j}\right)\right)\right],$$

given $\{c_m, \ell_m\}$ and $\left\{c'_{m,j}, \ell'_{m,j}\right\}$ for all j.

choice variables in line with Gauss-Jacobi and Gauss-Siedel iteration schemes; see Judd (1998) for a detailed description of these schemes. Below, we illustrate iteration-on-allocation in the context of system (75), (76).

Example 10. We represent system (75), (76) in the form $\widehat{\ell} = F(\ell)$ suitable for FPI:

$$\widehat{c} = \left[\frac{B(1-\ell)^{-\mu}}{(1-\alpha)\,\theta k^{\alpha} \ell^{-\alpha}}\right]^{-1/\gamma}, \tag{77}$$

$$\widehat{\ell} = \left[\frac{\widehat{c} + k' - (1-\delta)k}{\theta k^{\alpha}}\right]^{1/(1-\alpha)}. \tag{78}$$

Initialization. Fix initial guess $\ell^{(0)}$, a norm $\|\cdot\|$, and a convergence criterion ϖ.
Step 1. On iteration i, compute $\widehat{c}$ and $\widehat{\ell}$ using (77), (78), respectively.
Step 2. If $\left\|\widehat{\ell} - \ell^{(i)}\right\| < \frac{\varpi}{\xi}$, then stop.
Otherwise, set $\ell^{(i+1)} = \xi \ell^{(i)} + (1-\xi)\widehat{\ell}$, *where* $\xi \in (0, 1]$ and go to Step 1.

Therefore, we apply iteration-on-allocation $M \times (J+1)$ times to system (77), (78). Namely, we solve for quantities $\{c_m, \ell_m\}$ given $\left(k_m, \theta_m, k'_m\right)$ in all grid points, and we solve for quantities $\left\{c'_{m,j}, \ell'_{m,j}\right\}$ given $\left(k'_m, \theta'_{m,j}, k''_{m,j}\right)$ for all j, m.

The iteration-on-allocation method has the following valuable feature: it delivers exact solutions to the intratemporal choice condition both at present and future (up to a given degree of precision). The residuals in the intratemporal choice conditions are all zeros by construction, and the only condition that has nonzero residuals is the Euler equation (74).

The iteration-on-allocation method is used by Maliar et al. (2011) to compute the intratemporal choice in the context of the JEDC comparison analysis. A high accuracy of iteration-on-allocation gives an important advantage in accuracy to the stochastic simulation and cluster grid algorithms of Maliar et al. (2011) over the other methods participating in the JEDC comparison analysis; see Kollmann et al. (2011b). Those other methods approximate some of the intratemporal choice variables in terms of state contingent functions and face large residuals in some intratemporal choice conditions which dominate the overall accuracy of solutions.

Finally, let us mention that many different implementations of the iteration-on-allocation method are possible. First, in Eq. (78), we can use consumption obtained in the previous iteration, c, instead of consumption obtained in the current iteration, $\widehat{c}$ (this is the difference between the Gauss-Jacobi and Gauss-Siedel iterative schemes). Second, we can switch the order of equations, as well as change the way in which we express the variables from the equations. For example, we can express labor from the first equation (75), and we can express consumption from the second equation (76). A specific implementation can affect the numerical stability and speed of convergence of the iteration-on-allocation method.

6.3 The Intertemporal Choice Functions

Consider Euler equation (74) on a grid of points $\{k_m, \theta_m\}_{m=1,\dots,M}$,

$$u_1(c_m, \ell_m) = \beta \sum_{j=1}^{J} \omega_j \cdot \left[u_1\left(c'_{m,j}, \ell'_{m,j}\right)\left(1 - \delta + \theta'_{m,j} f_1\left(k'_m, \ell'_{m,j}\right)\right)\right]. \tag{79}$$

In the outer loop, we must find a vector of parameters b such that (79) is satisfied either exactly (collocation methods) or approximately (weighted residuals methods) on a given grid of points. Again, this is possible to do using Newton-style solvers; see, e.g., Judd (1992) and Gaspar and Judd (1997). To be specific, we fix some b, compute $k'_m = \widehat{K}(k_m, \theta_m; b)$ and $k''_{m,j} = \widehat{K}\left(k'_m, \theta'_{m,j}; b\right)$, solve for $\{c_m, \ell_m\}$ and $\left\{c'_{m,j}, \ell'_{m,j}\right\}$ from the intratemporal choice conditions, and substitute the results in the Euler equation (79) to evaluate the residuals. We subsequently perform iterations on b until the residuals are minimized. However, this procedure is expensive when the number of coefficients in b is large. For problems with high dimensionality, our preferred method is again FPI.

Example 11. To implement FPI on b, we represent (79) in the form $\widehat{k}'_m = F\left(k'_m\right)$ suitable for FPI as follows:

$$\widehat{k}'_m = \underbrace{\beta \sum_{j=1}^{J} \omega_j \cdot \left[\frac{u_1\left(c'_{m,j}, \ell'_{m,j}\right)}{u_1\left(c_m, \ell_m\right)}\left(1-\delta+\theta'_{m,j} f_1\left(k'_m, \ell'_{m,j}\right)\right)\right] k'_m}_{\equiv F\left(k'_m\right)}. \tag{80}$$

Initialization. Fix initial guess $b^{(0)}$, a norm $\|\cdot\|$, and a convergence criterion ϖ.
Step 1. On iteration i, use $b^{(i)}$ to compute $\left\{k'_m, c_m, \ell_m\right\}$ and $\left\{k'_{m,j}, c'_{m,j}, \ell'_{m,j}\right\}_{j=1,\ldots,J}$ and substitute them in the right side of (80) to find $\widehat{k}'_m$.
Step 2. *If* $\left\|\widehat{k}'_m - k'_m\right\| < \frac{\varpi}{\xi}$, then stop.
Otherwise, find $\widehat{b}$ that solves $\min_b \left\|\widehat{k}'_m - \widehat{K}\left(k_m, \theta_m; b\right)\right\|$, set $b^{(i+1)} = \xi b^{(i)} + (1-\xi)\widehat{b}$, where $\xi \in (0,1]$, and go to Step 1.

In the true solution, we have a fixed-point property $\widehat{k}'_m = k'_m$, and these two variables cancel out, making equations (79) and (80) to be equivalent. However, in the iterative process, we have $\widehat{k}'_m \neq k'_m$, and the difference between those quantities guides us in the process of convergence to the true solution.

We can use different FPI representations of the Euler equation to parameterize decision functions of other variables; for example, we can parameterize the consumption decision function by premultiplying the left and right sides of Euler equation (74) by $\widehat{c}_m$ and c_m, respectively; see Den Haan (1990) and Marcet and Lorenzoni (1999) for related examples.

Among many possible variants of FPI for finding b, we distinguish one special case, which is known as time iteration and is used in, e.g., the Smolyak method of Malin et al. (2011). In the context of our example, we fix b and represent all choice variables as functions of k'_m, namely, we first identify $k''_{m,j} = \widehat{K}\left(k'_m, \theta'_{m,j}; b\right)$ and we then identify $\left\{c_m, \ell_m\right\}$ and $\left\{c'_{m,j}, \ell'_{m,j}\right\}$ to satisfy the intratemporal choice conditions under given k'_m and $k''_{m,j}$. With this representation, we can implement the following iterative process.

Example 12. We rewrite Euler equation (79) to implement FPI of type $\widehat{b} = F(b)$:

$$\begin{aligned} & u_1\left(c_m\left(\cdot, \widehat{k}'_m\right), \ell_m\left(\cdot, \widehat{k}'_m\right)\right) \\ & \quad = \beta \sum_{j=1}^{J} \omega_j \cdot \left[u_1\left(c'_{m,j}\left(\cdot, \widehat{k}'_m\right), \ell'_{m,j}\left(\cdot, \widehat{k}'_m\right)\right)\left(1-\delta+\theta'_{m,j} f_1\left(\widehat{k}'_m, \ell'_{m,j}\left(\cdot, \widehat{k}'_m\right)\right)\right)\right]. \end{aligned} \tag{81}$$

Initialization. Fix an initial guess $b^{(0)}$, a norm $\|\cdot\|$, and a convergence criterion ϖ.
Step 1. On iteration i, solve for $\widehat{k}'_m$ satisfying (81) under $b^{(i)}$ and compute $k'_m = \widehat{K}\left(k_m, \theta_m; b^{(i)}\right)$.
Step 2. *If* $\left\|\widehat{k}'_m - k'_m\right\| < \frac{\varpi}{\xi}$, then stop.

Otherwise, find $\widehat{b}$ that solves $\min_b \left\| \widehat{k}'_m - \widehat{K}(k_m, \theta_m; b) \right\|$, set $b^{(i+1)} = \xi b^{(i)} + (1-\xi)\widehat{b}$, where $\xi \in (0, 1]$, and go to Step 1.

The name *time iteration* emphasizes the similarity of this iterative process to value function iteration. Namely, we restrict future end-of-period capital $k''_{m,j} = \widehat{K}\left(k'_m, \theta'_{m,j}; b\right)$, and we solve for current end-of-period capital k'_m, in the same way as in the case of dynamic programming methods, we fix the future value function, V_{t+1}, in the right side of the Bellman equation, and we solve for the present value function, V_t, in the left side of the Bellman equation; see Section 7 for more details. Some theoretical arguments suggest that time iteration is more stable numerically than other versions of FPI (because for some simple equilibrium problem, time iteration possesses the property of contraction mapping as does value function iteration; see Judd (1998) for a discussion). However, the contraction mapping is not preserved in more complex equilibrium problems so it is not clear which of many possible versions of FPI will perform best in terms of convergence.

Certainly, time iteration is far more expensive than our benchmark simple FPI described in Example 11. Even in the simple growth model, it requires us to find $\widehat{k}'_m$ satisfying (81) using a numerical solver. In more complex high-dimensional problems, time iteration is even more expensive because it requires us to solve a system of several Euler equations with respect to several unknowns. In contrast, FPI of type (80) is trivial to implement in problems with any dimensionality since only straightforward calculations are needed.

6.4 Coordination Between the Intratemporal and Intertemporal Choices

A specific way in which we define and coordinate the intertemporal and intratemporal choices can significantly affect the accuracy and speed of a solution method.

First, under some parameterizations, the system of intratemporal choice conditions can be easier to solve than under others. For example, Maliar and Maliar (2005b) use the labor decision function $\widehat{\mathcal{L}}(k_m, \theta_m; b)$ as an intertemporal choice instead of the capital decision function $\widehat{K}(k_m, \theta_m; b)$. In this case, we fix (k, θ, ℓ) and solve (72), (73) with respect to (c, k'). Under parameterization (17), the solution to the resulting intratemporal choice conditions can be characterized analytically.

Example 13. We represent system (72), (73) in the form

$$c = \left[\frac{B(1-\ell)^{-\mu}}{(1-\alpha)\theta k^{\alpha}\ell^{-\alpha}}\right]^{-1/\gamma},$$

$$k' = \theta k^{\alpha}\ell^{1-\alpha} + (1-\delta)k - c,$$

Second, we must compute the intertemporal and intratemporal choices in a way that delivers similar accuracy levels in all models' equations. If some decision functions are computed less accurately than the others, the overall accuracy of solutions will be

Table 11 Intertemporal choice in the two-agent model: one policy function versus iteration-on-allocation (in both the solution and simulation procedures).[a]

Optimality condition	Alternative 1		Alternative 2	
	L_1	L_∞	L_1	L_∞
Euler equation	−6.06	−4.57	−5.74	−4.35
Budget constraint	−4.54	−3.09	–	–
Overall	−4.54	−3.09	−5.74	−4.35

[a] *Notes:* L_1 and L_∞ are, respectively, the average and maximum of absolute residuals across test points (in log10 units). An entry "–" is used if accuracy measure is below −10 (such residuals are viewed as negligible). The results are reproduced from Maliar et al. (2011), Table 1.

determined by the least accurate decision function; see Maliar et al. (2011) for related examples and further discussion.

Finally, solving for the intratemporal choice with a high degree of accuracy may be unnecessary in every outer-loop iteration. For example, consider Example 10. Instead of solving for the intratemporal choice accurately in initial outer-loop iterations, we perform a fixed number of iterations in the inner loop, let us say 10. (Each time, we start the inner loop iterations from quantities $\{\ell_m\}, \left\{\ell'_{m,j}\right\}_{j=1,\dots,J}$ that were obtained at the end of the previous outer-loop iteration). As the capital decision function $\widehat{K}$ refines along the outer-loop iterations, so does the intertemporal choice; this technique is used by Maliar et al. (2011) to reduce the computational expense of finding the intratemporal choice in problems with high dimensionality. A possible shortcoming of this technique is that too inaccurate calculations of the intratemporal choice may reduce the numerical stability and may slow down the convergence of $\widehat{K}$ in the outer loop; to deal with this shortcoming, one must choose the damping parameter appropriately.

6.5 Numerical Illustration of the Importance of Coordination

Maliar et al. (2011) illustrate the importance of coordination between different computational techniques by showing an example in which the total accuracy is dominated by the least accurate decision function. Specifically, they consider two alternatives in the context of a two-agent version of model (1)–(3) with inelastic labor supply:

Alternative 1. Approximate consumption of the first agent, c^1, with a second-degree polynomial (the capital decision function is also approximated with a second-degree polynomial), and compute consumption of the second agent, c^2, to satisfy an FOC with respect to consumption.
Alternative 2. Solve for c^1 and c^2 to satisfy exactly the budget constraint and an FOC with respect to consumption using iteration-on-allocation.

Thus, under alternative 1, c^1 is approximated with error and this error is transmitted to c^2, while under alternative 2, c^1 and c^2 are calculated exactly. Table 11 shows the results.

As we can see, a parametric polynomial approximation of the intratemporal choice (i.e., alternative 1) reduces the overall accuracy of solutions by about an order of magnitude compared to the case when all the intratemporal choice variables are computed accurately using iteration-on-allocation (i.e., alternative 2). In this particular example, to have a comparable accuracy in the capital and consumption choices under alternative 1, one needs to approximate the consumption decision function with a polynomial of a degree higher than that used for parametric approximations of the capital decision function.

7. DYNAMIC PROGRAMMING METHODS FOR HIGH-DIMENSIONAL PROBLEMS

The efficient techniques for interpolation, integration, and optimization described in Sections 4, 5, and 6, respectively, apply to dynamic programming (DP) methods in the same way as they do to the Euler equation methods. DP methods which are analogs to the previously described Euler equation methods are developed and tested. In particular, Maliar and Maliar (2005a) introduce a solution method that approximates the value function by simulation and use this method to solve a multisector model with six state variables. Judd et al. (2011d) apply a value function iteration version of the GSSA method to solve a multicountry model with up to 20 countries (40 state variables). Judd et al. (2012) implement a value function iteration method operating on the EDS grid. Finally, Maliar and Maliar (2012a,b) use a version of ECM to solve a multicountry model studied in Kollmann et al. (2011b).

We therefore concentrate on two questions that are specific to DP methods. First, conventional value function iteration (VFI) leads to systems of equations that are expensive to solve, especially in high-dimensional applications. We describe two approaches that reduce the cost of conventional VFI: an *endogenous grid method* (EGM) of Carroll (2005) and an *envelope condition method* (ECM) of Maliar and Maliar (2013). Second, value function iteration produces less accurate solutions than Euler equation methods. We describe versions of EGM and ECM methods developed in Maliar and Maliar (2013) that approximate derivatives of the value function instead of, or in addition to, the value function itself and that deliver high accuracy levels comparable to those produced by Euler equation methods. We illustrate the above methods in the context of the one-agent model. In Section 12, we use a version of ECM to solve large-scale DP problems.

7.1 Conventional Value Function Iteration

As a starting point, we describe conventional VFI. To simplify the exposition, we consider a version of model (1)–(3) with inelastic labor supply. In the DP form, this model can be written as

$$V(k,\theta) = \max_{k',c} \left\{ u(c) + \beta E\left[V\left(k',\theta'\right)\right]\right\} \tag{82}$$

$$\text{s.t. } k' = (1-\delta)\,k + \theta f(k) - c, \tag{83}$$

$$\ln\theta' = \rho\ln\theta + \sigma\epsilon', \quad \epsilon' \sim \mathcal{N}(0,1). \tag{84}$$

If a solution to Bellman (82)–(84) is interior, the optimality quantities satisfy the FOC,

$$u'(c) = \beta E\left[V_1\left(k',\theta'\right)\right], \tag{85}$$

Conventional VFI goes backward: it guesses a value function in period $t+1$, and it solves for a value function in period t using the Bellman equation; this procedure is referred to as *time iteration* in the literature.

(VFI): Conventional value function iteration with inner and outer loops.

Step 1. Choose a grid $\{k_m,\theta_m\}_{m=1,\ldots,M}$ on which value function is approximated. Choose a flexible functional form $\widehat{V}(\cdot;b)$ for approximating value function V.
Step 2. Choose nodes, ϵ_j, and weights, $\omega_j, j=1,\ldots,J$, for approximating integrals. Compute next-period productivity $\theta'_{m,j} = \theta_m^{\rho}\exp(\epsilon_j)$ for all j, m.
Step 3. Solve for b and $\left\{c_m,k'_m\right\}$ such that
–*3a (inner loop):* The quantities $\left\{c_m,k'_m\right\}$ solve:

$$u'(c_m) = \beta E\left[V_1\left(k'_m,\theta'_m\right)\right],$$
$$c_m + k'_m = \theta_m f(k_m) + (1-\delta)\,k_m,$$

given $V(k_m,\theta_m)$.
–*3c (outer loop):* The value function $\widehat{V}(k,\theta;b)$ solves:

$$\widehat{V}(k_m,\theta_m;b) = u(c_m) + \beta\sum_{j=1}^{J}\omega_j\cdot\left[\widehat{V}\left(k'_m,\theta'_{m,j};b\right)\right],$$

given $\left\{c_m,k'_m\right\}$.

Here, we approximate V parametrically, namely, we parameterize V with a flexible functional form $\widehat{V}(\cdot;b)$, and we solve for a coefficient vector b that approximately satisfies (82). In turn, the policy functions satisfying (83), (85) are computed in terms of quantities, i.e., nonparametrically. Conditions (83), (85) can be viewed as intratemporal choice conditions, and condition (82) can be viewed as an intertemporal choice condition. This separation is parallel to the one introduced in Maliar et al. (2011) for the Euler equation methods and is discussed in Section 6.

The main shortcoming of conventional VFI is that the system of equations in the inner loop is expensive to solve numerically. Indeed, consider (83) and (85) under our benchmark parameterization.

Example 14. (Inner loop): Under (18), combining (83) and (85) yields

$$\left(k' - (1-\delta)\,k - \theta k^{\alpha}\right)^{-\gamma} = \beta E\left[V_1\left(k',\theta'\right)\right]. \tag{86}$$

We must solve (86) with respect to k' given (k,θ).

To find a root to (86), we must explore many different candidate points k'. For each candidate point, we must interpolate V_1 to J new values (k', θ'), as well as to approximate conditional expectation $E[V_1(k', \theta')]$. The cost of this procedure is high even in models with few state variables; see Aruoba et al. (2006) for examples assessing the cost of VFI.

7.2 Reducing the Cost of Dynamic Programming Methods

We now describe two methods that reduce the cost of conventional VFI: an endogenous grid method of Carroll (2005) and an envelope condition method of Maliar and Maliar (2013).

7.2.1 Endogenous Grid Method

Carroll (2005) introduces an endogenous grid method (EGM) that simplifies rootfinding under VFI. The key idea of EGM is to construct a grid on future endogenous state variables instead of current endogenous state variables. EGM dominates conventional VFI because in a typical economic model, it is easier to solve for k given k' than to solve for k' given k. Below, we outline EGM for our example (the steps that coincide under conventional VFI and EGM are omitted).

(EGM): An endogenous grid method with inner and outer loops.

```
Step 1. Choose a grid $\{k'_m, \theta_m\}_{m=1,...,M}$ on which value function is approximated.
...
Step 3. Solve for $b$ and $\{c_m, k_m\}$ such that
–3a (inner loop): The quantities $\{c_m, k_m\}$ solve:
        $u'(c_m) = \beta E\left[V_1\left(k'_m, \theta'_{m,j}\right)\right]$,
        $c_m + k'_m = \theta_m f(k_m) + (1-\delta) k_m$,
given $V(k_m, \theta_m)$.
...
```

Since the values of k' are fixed (they are grid points), it is possible to up-front compute $E[V(k', \theta')] \equiv W(k', \theta)$ and $E[V_1(k', \theta')] \equiv W_1(k', \theta)$. Now, system (83) and (85) can be written as follows:

Example 15. (Inner loop): Under (18), we use (83) to express $c = [\beta W_1(k', \theta)]^{-1/\gamma}$, and we rewrite (85) as

$$(1-\delta) k + \theta k^{\alpha} = \left[\beta W_1\left(k', \theta\right)\right]^{-1/\gamma} + k'. \tag{87}$$

We must solve (87) with respect to k given (k', θ).

Equation (87) is easier to solve numerically than (86) because it does not involve either interpolation or approximation of a conditional expectation. Furthermore, Carroll (2005) finds a clever change of variables that simplifies rootfinding even further and effectively allows us to solve (87) in a closed form. He introduces a new variable $Y \equiv (1-\delta)\,k + \theta k^{\alpha} = c + k'$, which allows us to rewrite Bellman equation (82)–(84) as

$$V\left(Y,\theta\right) = \max_{k'} \left\{ \frac{c^{1-\gamma}-1}{1-\gamma} + \beta E\left[V\left(Y',\theta'\right)\right] \right\}, \tag{88}$$

$$\text{s.t. } c = Y - k', \tag{89}$$

$$Y' = (1-\delta)\,k' + \theta'\left(k'\right)^{\alpha}, \tag{90}$$

where $E\left[V\left(Y',\theta'\right)\right] = W\left(Y',\theta\right)$. The FOC of 88,89,90 is

$$u'\left(c\right) = \beta E\left[V_1\left(Y',\theta'\right)\left(1-\delta+\alpha\theta'\left(k'\right)^{\alpha-1}\right)\right]. \tag{91}$$

FOC (91) allows us to compute c directly since $Y' \equiv (1-\delta)\,k' + \theta'\,(k')^{\alpha}$ follows directly from the (endogenous) grid points and so does the conditional expectation in the right side of (91). Given c, we find $Y = k' + c$ and, subsequently, we find $V\left(Y,\theta\right)$ in the left side of (88). This provides a basis for iterating on the Bellman equation (88) without using a numerical solver. Once the convergence is achieved, we find k that corresponds to the constructed endogenous grid by solving $Y \equiv (1-\delta)\,k + \theta k^{\alpha}$ with respect to k for each value Y. This step does require us to use a numerical solver, but just once, at the very end of the iteration procedure.

7.2.2 Envelope Condition Method

The envelope condition method (ECM) of Maliar and Maliar (2013) simplifies rootfinding using a different mechanism. First, ECM does not perform conventional backward iteration on the Bellman equation but iterates forward. Second, to construct policy functions, ECM uses the envelope condition instead of the FOCs used by conventional VFI and EGM. The systems of equations produced by ECM are typically easier to solve than those produced by conventional VFI. In this sense, ECM is similar to EGM.

For problem (82)–(84), the envelope condition is given by

$$V_1\left(k,\theta\right) = u'\left(c\right)\left[1-\delta+\theta f_1\left(k\right)\right]. \tag{92}$$

Typically, the envelope condition used to derive the Euler equation (92) is updated to get $V_1\left(k',a'\right)$ and the result is substituted into (85) to eliminate the unknown derivative of the value function). In ECM, the envelope condition is used to solve for the intratemporal choice by combining (92) and (85).

ECM proceeds as follows (the steps that coincide under VFI and ECM are omitted): Consider again our benchmark example under parameterization (18).

Example 16. (Inner loop): Under (18), we solve envelope condition (92) with respect to c in a closed form:

$$c = \left(\frac{V_1\left(k,\theta\right)}{1-\delta+\alpha\theta\left(k\right)^{\alpha-1}}\right)^{-1/\gamma}. \tag{93}$$

(ECM): An envelope condition method with inner and outer loops.

```
Step 1. Choose a grid $\{k_m, \theta_m\}_{m=1,\ldots,M}$ on which the value function is approximated.
...
Step 3. Solve for $b$ and $\{c_m, k'_m\}$ such that
–3a (inner loop): The quantities $\{c_m, k'_m\}$ solve:
    $V_1(k_m, \theta_m) = u'(c_m)\left[1 - \delta + \theta_m f'(k_m)\right]$,
    $c_m + k'_m = \theta_m f(k_m) + (1 - \delta) k_m$,
given $V(k_m, \theta_m)$.
...
```

We compute k' from budget constraint (85) given (k, θ).

In this example, ECM is simpler than Carroll's (2005) EGM since all the policy functions can be constructed analytically and a solver need not be used at all (not even once).

7.2.3 Shall We Use the Envelope Condition with Endogenous Grid?

Four combinations are possible to obtain from two alternative conditions for V_1 (i.e., FOC (83) and envelope condition (92)) and two alternative grids (i.e., exogenous and endogenous). So far, we have distinguished two competitive methods: one is EGM of Carroll (2005) (FOC (83) and endogenous grid) and the other is our ECM (envelope condition (92) and exogenous grid). Conventional VFI (FOC (83) and exogenous grid) is not competitive. Therefore, we are left to check the remaining combination (envelope condition (92) and endogenous grid).

Example 17. (Inner loop): Combining (92) and (93) yields

$$(1-\delta)k + \theta k^{\alpha} = \left(\frac{V_1(k,\theta)}{1-\delta+\alpha\theta(k)^{\alpha-1}}\right)^{-1/\gamma} + k'. \tag{94}$$

We must solve (94) for k given (k', θ).

Solving (94) involves costly evaluation of $V_1(k, \theta)$ for many candidate solution points (k, θ). We conclude that the combination of the envelope condition and endogenous grid does not lead to a competitive method. Our results are suggestive for other applications.

7.2.4 EGM and ECM in a Model with Elastic Labor Supply

Maliar and Maliar (2013) apply ECM and EGM for solving the one-agent model with elastic labor supply under assumption (17). Under EGM, Carroll's (2005) change of variables does not avoid rootfinding in the model with elastic labor supply. The variable $Y' = \theta' f(k', \ell') + k'$ depends on future labor ℓ', and $E[V(Y', \theta')]$ cannot be computed without specifying a labor policy function. Barillas and Fernández-Villaverde (2007) propose a way of extending EGM to the model with elastic labor supply. Namely, they fix a policy function for labor $\ell = \mathcal{L}(k', \theta)$, construct the grid of (Y', θ), solve the model on that grid holding $\mathcal{L}$ fixed, and use the solution to reevaluate $\mathcal{L}$; they iterate on these steps until $\mathcal{L}$ converges. See also Villemot (2012) for an application of EGM for solving a model of sovereign debt.

The implementation of EGM in Maliar and Maliar (2013) for the model with elastic labor supply differs from that in Barillas and Fernández-Villaverde (2007). First, Maliar and Maliar (2013) use future endogenous state variables for constructing grid points but they do not use Carroll's (2005) change of variables. Second, to deal with rootfinding, Maliar and Maliar (2013) use a numerical solver, while Barillas and Fernández-Villaverde (2007) iterate on a state contingent policy function for labor $\mathcal{L}(k', \theta)$. Maliar and Maliar (2013) find that ECM and EGM have very similar performances in that they produce solutions of similar (high) accuracy and have similar computational expenses. Even though neither ECM nor EGM allows us to solve the system of intratemporal choice conditions in a closed form, these two methods still lead to far simpler systems of equations than does conventional VFI.

7.3 Increasing the Accuracy of Dynamic Programming Methods

So far, we have been concerned with the inner loop: all the methods we have considered differ in the way in which they solve for the intratemporal choice. Maliar and Maliar (2012b, 2013) modify the outer loop as well by introducing versions of EGM and ECM that solve for the derivative of value function, V_1, instead of or in addition to the value function itself V. Since only the derivative of the value function enters optimality conditions (83) and (92) but not the value function itself, we can find a solution to problem (82)–(84) without computing V explicitly.

In terms of our example, combining (83) and (92) yields

$$V_1(k,\theta) = \beta\left[1-\delta+\theta f_1(k,\ell)\right] E\left[V_1\left(k',\theta'\right)\right]. \tag{95}$$

This equation is used in Maliar and Maliar (2012b, 2013) to implement iteration on the derivative of the value function similar to the one we implement for value function using the Bellman equation. Iteration on the derivative of the value function is compatible with all previously described DP methods such as conventional VFI, EGM, and ECM. In particular, ECM modifies as follows:

(ECM-DVF): An algorithm iterating on derivatives of value function.

…

–3c (outer loop): The derivative of value function, $\widehat{V}_1(k,\theta;b)$, solves:

$$\widehat{V}_1(k_m,\theta_m;b) = \beta\left[1-\delta+\alpha\theta_m k_m^{\alpha-1}\right]\sum_{j=1}^{J}\omega_j \widehat{V}_1\left(k'_m,\theta_{m,j};b\right),$$

given $\left\{c_m, k'_m\right\}$.

In some experiments, Maliar and Maliar (2012a,b) approximate the value function V and its derivative V_1 jointly. A robust finding is that methods approximating derivatives of value function lead to far more accurate solutions than the methods approximating the value function itself.

7.4 Numerical Illustration of Dynamic Programming Methods

Under Maliar and Maliar's (2012a) implementation, the rootfinding problems under EGM and ECM are comparable in their complexity. In both cases, we must find a solution to a nonlinear equation in each grid point. Such an equation does not involve either interpolation or approximation of expectations. Below, we reproduce from Maliar and Maliar (2013) the results about a numerical comparison of ECM with EGM in the context of the model with elastic labor supply parameterized by (17).

We first solve for V by iterating on the Bellman equation (82)–(84); we refer to the corresponding methods as EGM-VF and ECM-VF. The results are shown in Table 12. The performance of EGM-VF and ECM-VF is very similar. EGM-VF produces slightly smaller maximum residuals, while ECM-VF produces slightly smaller average residuals. EGM-VF is somewhat slower than ECM-VF.

We next solve for V_1 by iterating on (95); we call the corresponding methods EGM-DVF and ECM-DVF. The results are provided in Table 13. Again, EGM-DVF and ECM-DVF perform very similarly. Both methods deliver accuracy levels that are about an order of magnitude higher than those of EGM-VF and ECM-VF.

Iterating on (95) produces more accurate solutions than iterating on (82) because the object that is relevant for accuracy is V_1 and not V (V_1 identifies the model's variables via (83), (85), and (92)). Approximating a supplementary object V and computing its derivative V_1 involves an accuracy loss compared to the case when we approximate the relevant object V_1 directly. For example, if we approximate V with a polynomial, we effectively approximate V_1 with a polynomial which is one degree lower, i.e., we "lose"

Table 12 Accuracy and speed of EGM-VF and ECM-VF in the one-agent model with elastic labor supply.[a]

Polynomial degree	EGM-VF			ECM-VF		
	L_1	L_∞	CPU	L_1	L_∞	CPU
1st	–	–	–	–	–	–
2nd	−3.28	−2.81	8.3	−3.34	−2.75	5.8
3rd	−4.31	−3.99	8.9	−4.38	−3.87	7.2
4th	−5.32	−4.96	7.3	−5.45	−4.86	5.8
5th	−6.37	−5.85	6.5	−6.57	−5.72	4.7

[a] *Notes:* L_1 and L_∞ are, respectively, the average and maximum of absolute residuals across optimality condition and test points (in log10 units) on a stochastic simulation of 10,000 observations; CPU is the time necessary for computing a solution (in seconds). These results are reproduced from Maliar and Maliar (2013), Table 1.

Table 13 Accuracy and speed of EGM-DVF and ECM-DVF in the one-agent model with elastic labor supply.[a]

Polynomial degree	**EGM-DVF**			**ECM-DVF**		
	L_1	L_∞	CPU	L_1	L_∞	CPU
1st	−3.03	−2.87	8.1	−3.08	−2.92	7.2
2nd	−4.13	−3.82	7.2	−4.18	−3.91	6.5
3rd	−5.06	−4.77	7.3	−5.20	−4.87	6.7
4th	−6.09	−5.64	7.4	−6.29	−5.72	6.8
5th	−7.12	−6.26	7.6	−7.36	−6.32	6.9

[a] *Notes:* L_1 and L_∞ are, respectively, the average and maximum of absolute residuals across optimality condition and test points (in log10 units) on a stochastic simulation of 10,000 observations; CPU is the time necessary for computing a solution (in seconds). These results are reproduced from Maliar and Maliar (2013), Table 2.

one polynomial degree. In our experiments, EGM-DVF and ECM-DVF attain accuracy levels that are comparable to the highest accuracy attained by the Euler equation methods.

8. PRECOMPUTATION TECHNIQUES

Precomputation is a technique that computes solutions to some model's equations outside the main iterative cycle. Precomputation techniques can be analytical, numerical, or a combination of both. We describe two numerical techniques: a technique of precomputing integrals in the intratemporal choice introduced in Judd et al. (2011d) and a technique of precomputing the intratemporal choice manifolds introduced in Maliar and Maliar (2005b) and developed in Maliar et al. (2011). We then briefly discuss an analytical technique of *imperfect aggregation* of Maliar and Maliar (2001, 2003a) which allows us to characterize aggregate behavior of certain classes of heterogeneous-agent economies in terms of one-agent models.

8.1 Precomputation of Integrals

In existing iterative methods for solving dynamic stochastic models, a conditional expectation function is computed in each iteration. Recomputing expectations in each iteration is costly. Judd et al. (2011d) show a simple technique that makes it possible to compute conditional expectation functions just once, in the stage of initialization. The proposed technique is called *precomputation of integrals*. We first show how to precompute the expectation for polynomial functions, and we then use the resulting polynomial functions to approximate expectations in the Euler and Bellman equations outside the main iterative loop.

8.1.1 Precomputation of Expectations for Polynomial Functions

Let us consider the one-agent model 1, 2, 3. Consider a complete ordinary polynomial function in terms of the current state (k, θ),

$$\mathcal{P}(k, \theta; b) = b_1 + b_2 k + b_3 \theta + b_4 k^2 + b_5 k\theta + b_6 \theta^2 + \cdots + b_I \theta^L, \tag{96}$$

where $b \equiv (b_1, b_1, \ldots, b_I) \in \mathbb{R}^I$ is a vector of polynomial coefficients, and L is a polynomial degree. Taking into account that k' is known at present and that $\theta' = \theta^\rho \exp(\epsilon')$, we can represent the conditional expectation of $\mathcal{P}(k', \theta'; b)$ as follows:

$$\begin{aligned}
&E\left[\mathcal{P}\left(k', \theta'; b\right)\right] \\
&\quad = E\left[b_1 + b_2 k' + b_3 \theta^\rho \exp\left(\epsilon'\right) + b_4\left(k'\right)^2 + b_5 k' \theta^\rho \exp\left(\epsilon'\right) + \cdots + b_I \theta^{L\rho} \exp\left(L\epsilon'\right)\right] \\
&\quad = b_1 \mathcal{I}_1 + b_2 \mathcal{I}_2 k' + b_3 \mathcal{I}_3 \theta^\rho + b_4 \mathcal{I}_4 \left(k'\right)^2 + b_5 \mathcal{I}_5 k' \theta^\rho + \cdots + b_I \mathcal{I}_I \theta^{L\rho} \\
&\quad \equiv \mathcal{P}\left(k', \theta^\rho; b\mathcal{I}\right), \qquad (97)
\end{aligned}$$

where $b\mathcal{I} \equiv (b_1 \mathcal{I}_1, \ldots, b_I \mathcal{I}_I) \in \mathbb{R}^I$, and $\mathcal{I}_1 = \mathcal{I}_2 = \mathcal{I}_4 = \mathcal{I}_7 = \cdots = e_0, \mathcal{I}_3 = \mathcal{I}_5 = \mathcal{I}_8 = \cdots = e_1, \mathcal{I}_6 = \mathcal{I}_9 = \cdots = e_2, \ldots, \mathcal{I}_I = e_L$ with

$$e_l \equiv E\left[\exp\left(l\epsilon'\right)\right] = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} \exp\left(l\epsilon'\right) \exp\left(-\frac{\left(\epsilon'\right)^2}{2\sigma^2}\right) d\epsilon'. \qquad (98)$$

The integrals $\mathcal{I}_1, \ldots, \mathcal{I}_I$ can be computed up-front without solving the model (i.e., precomputed). Once $\mathcal{I}$'s are computed, an evaluation of conditional expectation becomes very simple. Namely, the conditional expectation of a polynomial function is given by the same polynomial function but evaluated at a different coefficient vector, i.e., $E[\mathcal{P}(k', \theta'; b)] = \mathcal{P}(k', \theta^\rho; b\mathcal{I})$. Effectively, precomputation of integrals converts a stochastic problem into a deterministic one.

8.1.2 Precomputation of the Expectation in the Euler Equation

We now show how to precompute conditional expectation functions in Euler equation (74). Let us parameterize a variable inside the conditional expectation function with a polynomial function,

$$u_1(c, \ell)\left[1 - \delta + \theta f_1(k, \ell)\right] = b_1 + b_2 k + \cdots + b_I \theta^L. \qquad (99)$$

By (99), we can write Euler equation (74) as

$$\begin{aligned}
u_1(c, \ell) &= \beta E\left\{u_1\left(c', \ell'\right)\left[1 - \delta + \theta' f_1\left(k', \ell'\right)\right]\right\} \\
&= \beta E\left[b_1 + b_2 k' + \cdots + b_I \theta^{L\rho} \exp\left(L\varepsilon'\right)\right] \\
&= \beta\left[b_1 \mathcal{I}_1 + b_2 \mathcal{I}_2 k' + \cdots + b_I \mathcal{I}_I \theta^{L\rho}\right], \qquad (100)
\end{aligned}$$

where $\mathcal{I}_1, \ldots, \mathcal{I}_I$ are integrals that are known (precomputed) using (98). With this result, we formulate a system of four equations to feed into a grid-based Euler equation method:

$$u_1(c_m, \ell_m)\left[1 - \delta + \theta_m f_1(k_m, \ell_m)\right] = b_1 + b_2 k_m + \cdots + b_I \theta_m^L, \qquad (101)$$

$$u_2(c_m, \ell_m) = u_1(c_m, \ell_m)\, \theta_m f_2(k_m, \ell_m), \qquad (102)$$

$$c_m + k'_m = \theta_m f(k_m, \ell_m) + (1-\delta) k_m, \tag{103}$$

$$u_1(c_m, \ell_m) = \beta \left[b_1 \mathcal{I}_1 + b_2 \mathcal{I}_2 k'_m + \cdots + b_I \mathcal{I}_I \theta_m^{\rho L} \right]. \tag{104}$$

We can now implement the following iterative scheme: fix some b, use the intratemporal choice equations 101, 102, 103 to solve for $\left\{c_m, \ell_m, k'_m\right\}$ in each grid point (k_m, θ_m), and use the intertemporal choice equation (104) to recompute b. There is an explicit OLS formula for b, namely, $b = (X'X)^{-1} X' u_1$, where $X = \left[\beta \mathcal{I}_1, \mathcal{I}_2 k'_m, \ldots, \mathcal{I}_I \theta_m^{\rho L}\right]$ is a matrix of the coefficients and $u_1 \equiv (u_1(c_1, \ell_1), \ldots, u_1(c_M, \ell_M))^{\top}$ is a vector of marginal utilities. However, the OLS method can be numerically unstable, and we may need to use regression methods that are robust to ill-conditioning studied in Section 4.2.

With precomputation of integrals, we can also construct a version of the Euler equation method that operates on an endogenous grid $\left\{k'_m, \theta_m\right\}_{m=1,\ldots,M}$ and that is analogous to Carroll's (2005) EGM. The iterative scheme modifies as follows: fix some b, use the intratemporal choice equations (102)–(104) to solve for $\{c_m, \ell_m, k_m\}$ in each grid point $\left(k'_m, \theta_m\right)$, and use the intertemporal choice condition (101) to recompute b. Again, there is an explicit OLS formula for $b = (X'X)^{-1} X' u_1^{+}$, where $X = \left[1, k_m, \ldots, \theta_m^{\rho L}\right]$ is a matrix of the coefficients and variable u_1^{+} is defined as $u_1^{+} = (u_1(c_1, \ell_1)[1-\delta+\theta_1 f_1(k_1, \ell_1)], \ldots, u_1(c_M, \ell_M)[1-\delta+\theta_M f_1(k_M, \ell_M)])^{\top}$.

8.1.3 Precomputation of the Expectation in the Bellman Equation

We now show how to precompute conditional expectation functions in Bellman equation 8, 9, 10. Let us parameterize the value function with complete ordinary polynomial,

$$V(k, \theta) \approx \widehat{V}(k, \theta; b) = b_1 + b_2 k + \cdots + b_I \theta^L. \tag{105}$$

Then, according to (97), expectation $E[V(k', \theta')]$ can be precomputed as

$$E\left[V\left(k', \theta'\right)\right] \approx E\left[\widehat{V}\left(k', \theta'; b\right)\right] = b_1 \mathcal{I}_1 + b_2 \mathcal{I}_2 k'_m + \cdots + b_I \mathcal{I}_I \theta_m^{\rho L}. \tag{106}$$

With this result, we rewrite the Bellman equation (8) as

$$u(c, \ell) = b_1 (1 - \beta \mathcal{I}_1) + b_2 \left(k - \beta \mathcal{I}_2 k'\right) + \cdots + b_I \left(\theta^L - \beta \mathcal{I}_I \theta^{\rho L}\right). \tag{107}$$

The system of the optimality conditions under precomputation of integrals becomes

$$u_1(c_m, \ell_m)[1-\delta+\theta_m f_1(k_m . \ell_m)] = \frac{\partial}{\partial k_m}\left[b_1 + b_2 k_m + \cdots + b_I \theta_m^L\right], \tag{108}$$

$$u_2(c_m, \ell_m) = u_1(c_m, \ell_m) \theta_m f_2(k_m, \ell_m), \tag{109}$$

$$c_m + k'_m = \theta_m f(k_m, \ell_m) + (1-\delta) k_m, \tag{110}$$

$$u(c_m, \ell_m) = b_1 (1 - \beta \mathcal{I}_1) + b_2 \left(k_m - \beta \mathcal{I}_2 k'_m\right) + \cdots + b_I \left(\theta_m^L - \beta \mathcal{I}_I \theta_m^{\rho L}\right). \tag{111}$$

A version of ECM with precomputation of integrals is as follows: fix some b, use intratemporal choice equations (108)–(110) to solve for $\left\{c_m, \ell_m, k'_m\right\}$ in each grid point (k_m, θ_m), and use the intertemporal choice equation (111) to recompute b; iterate until convergence.

Precomputation of integrals can also be performed for the DP methods iterating on derivatives of value function as discussed in Section 7. Interestingly, a method that iterates on a derivative of value function coincides with the previously described Euler equation method that solves (101)–(104). This is because in this specific example, a variable in (99) which is parameterized to precompute the expectations in the Euler equation (74) coincides with the derivative $V_1(k, \theta)$. However, precomputation of integrals in the Euler equation is possible even for those problems for which we do not have a Bellman equation.

Similar to the Euler equation case, there is an explicit OLS formula for $b = (X'X)^{-1} X'u$, where $X = \left[(1 - \beta\mathcal{I}_1), (k_m - \beta\mathcal{I}_2 k'_m), \ldots, (\theta_m^L - \beta\mathcal{I}_I \theta_m^{\rho L})\right]$ is a matrix of coefficients, and $u = (u(c_1, \ell_1), \ \ldots, u(c_M, \ell_M))^\top$ is a vector of values of the utility functions. Finally, we can construct a version of Carroll's (2005) EGM with precomputation of integrals. In this case, we compute the intratemporal choice $\{c_m, \ell_m, k_m\}$ on the grid $\left\{k'_m, \theta_m\right\}_{m=1,\ldots,M}$ to satisfy (109)–(111), and we use (108) to recompute b.

8.1.4 Relation of Precomputation of Integrals to the Literature

Effectively, precomputation of integrals allows us to solve stochastic economic models as if they were deterministic, with the corresponding reduction in cost. Precomputation of integrals is a very general technique that can be applied to any set of equations that contains conditional expectations, including the Bellman and Euler equations. Furthermore, precomputation of integrals is compatible with a variety of approximating functions, solution domains, integration rules, fitting methods, and iterative schemes for finding unknown parameters of approximating functions. That is, apart from precomputation of integrals, the rest of our solution procedure is standard. Finally, given that we must approximate integrals just once, we can use very accurate integration methods that would be intractable if integrals were computed inside an iterative cycle.

We must emphasize that, in order to precompute integrals, it is critical to parameterize the integrand, i.e., the expression inside the expectation function. Other parameterizations such as the capital function or the expectation function do not allow us to precompute integrals using accurate deterministic integration methods.

There are two other methods of approximating integrals that have similar computational complexity as does the method of precomputation of integrals (they also evaluate integrals in just one point). First, Marcet (1988) approximates an expectation function in each simulated point using a next-period realization of the integrand (which is a version of a Monte Carlo method with one random draw); the accuracy of this approximation is low and limits dramatically the accuracy of solutions; see our discussion in Section 5.8. Second, Judd et al. (2011a) propose to use a one-node Gauss-Hermite quadrature method, which approximates the integral with the expected value of the integrand; this method is far more accurate than the one-node Monte Carlo integration and is even more accurate than Monte Carlo integration with thousands of random nodes in the class of problems studied. Still, for high-degree polynomials, the one-node quadrature method limits the accuracy of solutions. In contrast, the method of precomputation of integrals delivers the highest possible accuracy that can be produced by deterministic integration methods at the lowest possible cost.

8.1.5 Numerical Illustration of the Precomputation Methods

In Table 14, we provide the results for the model with inelastic labor supply parameterized by (18) that are delivered by the Bellman and Euler equation methods with precomputation of integrals (these results are reproduced from Judd et al. (2011d)).

As we can see, the residuals decrease with each polynomial degree by one or more orders of magnitude. High-degree polynomial solutions are very accurate. Under ECM-VF-Prec (which is a DP algorithm), the maximum residuals for the fifth-degree polynomial approximations are of the order 10^{-6}. Under ECM-DVF-Prec (which is an Euler equation algorithm), the maximum residuals are about one order of magnitude smaller. The tendencies are similar to those we observe for the corresponding methods without precomputation; see Tables 12 and 13 for a comparison.

8.2 Precomputation of Intratemporal Choice Manifolds

The precomputation technique of intratemporal choice manifolds constructs solutions to the intratemporal choice conditions outside the main iterative cycle and uses the constructed solutions to infer the intratemporal choice inside the main iterative cycle. Consider the intratemporal choice conditions for model (1)–(3) under parameterization (17)

$$B\ell^{-\mu} = c^{-\gamma}\left(1-\alpha\right)\theta k^{\alpha}\left(1-\ell\right)^{-\alpha}, \tag{112}$$

$$c = \theta k^{\alpha}\ell^{1-\alpha} + \left(1-\delta\right)k - k'. \tag{113}$$

Table 14 Accuracy and speed of the Bellman equation algorithm in the one-agent model with inelastic labor supply.[a]

Polynomial degree	**ECM-VF-Prec**			**ECM-DVF-Prec**		
	L_1	**L_∞**	**CPU**	**L_1**	**L_∞**	**CPU**
1st	−1.64	−1.63	8.60	−3.39	−3.24	1.20
2nd	−3.65	−3.42	0.14	−4.64	−4.21	1.16
3rd	−4.83	−4.39	0.16	−5.68	−5.19	0.72
4th	−5.96	−5.36	0.17	−6.83	−6.18	0.73
5th	−7.12	−6.43	0.18	−8.01	−7.32	0.40

[a] *Notes:* L_1 and L_∞ are, respectively, the average and maximum of absolute residuals across optimality condition and test points (in log10 units) on a stochastic simulation of 10,000 observations; CPU is the time necessary for computing a solution (in seconds). The results for ECM-VF-Prec and ECM-DVF-Prec are reproduced from Judd et al. (2011d), Tables 1 and 2, respectively.

We denote a solution for c and ℓ for given (k, θ, k') as $c = \widetilde{C}(k, \theta, k')$ and $\ell = \widetilde{\mathcal{L}}(k, \theta, k')$, respectively, and we call $\widetilde{C}$ and $\widetilde{\mathcal{L}}$ *solution manifolds*.

Observe that it is sufficient for us to construct a manifold for labor $\widetilde{\mathcal{L}}$ since we can infer consumption from (113) if labor is given. To compute the intratemporal choice inside the main iterative cycle, first we use $\widehat{K}(k, \theta; b)$ to find k' and k''; then we use $\widetilde{\mathcal{L}}$ to compute $\ell = \widetilde{\mathcal{L}}(k, \theta, k')$ and $\ell' = \widetilde{\mathcal{L}}(k', \theta', k'')$; and, finally, we can get c and c' from (113). In the remainder of the section, we focus on constructing $\widetilde{\mathcal{L}}$.

In simple cases, the intratemporal choice manifolds can be constructed analytically. Consider the following example.

Example 18. Assume (17) under $\gamma = \mu = 0$. Then, system (112), (113) admits a closed-form solution in the form

$$\ell = B^{-1/\alpha}\left[(1-\alpha)\theta\right]^{1/\alpha} k. \tag{114}$$

However, under more general parameterizations, $\widetilde{\mathcal{L}}$ cannot be constructed analytically from (112), (113). We therefore construct it numerically.

Example 19. Assume (17). Consider system (112), (113). We proceed as follows:
– Outside the main iterative cycle:

(i) Take a grid $\left\{k_i, \theta_i, k_i'\right\}_{i=1,\ldots,I}$ within the relevant range.
(ii) For each grid point i, compute the values of c_i and ℓ_i that satisfy (112), (113) using a numerical solver.
(iii) Construct a labor manifold $\widetilde{\mathcal{L}}(k, \theta, k')$ (by fitting $\ell_1, \ldots, \ell_I$ to some parametric function, such as polynomial function, splines, etc., on the grid $\left\{k_i, \theta_i, k_i'\right\}_{i=1,\ldots,I}$).

In the above example, we had to compute values for labor using a numerical solver. We now show how to "invert" the problem and to precompute the same manifold but without using a solver.

Example 20. Assume (17). Rewrite system (112), (113) as follows

$$k = \left[\frac{u_2(c,\ell)}{u_1(c,\ell)\,\theta\,(1-\alpha)\,(1-\ell)^{1-\alpha}}\right]^{1/\alpha}, \tag{115}$$

$$k' = \theta f(k,\ell) + (1-\delta)k - c. \tag{116}$$

We proceed as follows:
– Outside the main iterative cycle:

(i) Take a grid $\{\theta_i, c_i, \ell_i\}_{i=1,\ldots,I}$ within the relevant range.
(ii) For each grid point i, compute analytically the values of k_i, k_i' from (115), (116).
(iii) Construct a labor manifold $\widetilde{\mathcal{L}}(k, \theta, k')$ (by fitting the values $\ell_1, \ldots, \ell_I$ to a polynomial function, splines, etc., on the grid $\left\{k_i, \theta_i, k_i'\right\}_{i=1,\ldots,I}$).

Maliar and Maliar (2005b) show an example of precomputation of the labor choice in a similar model but assuming that the intertemporal choice is parameterized with a consumption decision function instead of a capital decision function. In this case, the labor choice can be precomputed in terms of just one artificial variable and, thus, is easier to construct in the main iterative cycle; see the example below.

Example 21. Assume (17). Consider the system (112), (113). Let us define a new variable z by

$$\ell^{-\mu}(1-\ell)^{\alpha} = \left[\widehat{C}(k,\theta;b)\right]^{-\gamma}(1-\alpha)\,\theta k^{\alpha}/B \equiv z. \tag{117}$$

We proceed as follows:

– Outside the main iterative cycle:

(i) Take a grid $\{z_i\}_{i=1,\ldots,I}$ within the relevant range.

(ii) For each grid point i, compute the value of variable ℓ_i satisfying (117).

(iii) Construct a labor manifold $\widetilde{\mathcal{L}}(z)$ (by fitting the values $\ell_1,\ldots,\ell_I$ to a polynomial function, splines, etc., on the grid z_i, $i=1,\ldots,I$).

Inside the main iterative cycle, given a value of $z=\widehat{C}(k,\theta;b)^{-\gamma}(1-\alpha)\,\theta k^{\alpha}/B$, we compute ℓ using the precomputed manifolds $\ell=\widetilde{\mathcal{L}}(z)$, and we compute k' using (113).

We can also invert the task and construct the same manifold by choosing a grid for $\{\ell_i\}_{i=1,\ldots,I}$ and by solving for z using (117), in which case a numerical solver is not needed. See Maliar et al. (2011) for examples of precomputation of the intratemporal choice manifolds for heterogeneous-agent economies.

Three comments are in order: First, the solution manifold $\widetilde{\mathcal{L}}$ depends on three arguments (k,θ,k') instead of the conventional decision function that depends on two arguments (k,θ): the relation between these objects is given by $\mathcal{L}(k,\theta)=\widetilde{\mathcal{L}}(k,\theta,K(k,\theta))$, where K is the optimal capital decision function; see Maliar et al. (2011) for a discussion. Second, there are many ways to define the intertemporal (and thus, intratemporal) choice manifolds; it is possible to fix any three variables in (112), (113) and solve for the remaining two variables; for example, we can fix (ℓ,k,θ) and find c,k'. Finally, precomputation of the intratemporal choice can also be implemented in the context of DP methods described in Section 7.

8.3 Precomputation of Aggregate Decision Rules

Maliar and Maliar (2001, 2003a) study complete-market heterogeneous-agents economies in which agents differ in initial endowments and labor productivities (under some parameterizations, their model is identical to the multicountry model studied in Section 11). In particular, they show that under the standard assumptions of the CRRA or addilog utility functions and the Cobb-Douglas production function, it is possible to characterize the aggregate behavior of a heterogeneous-agents economy in terms of a one-agent model. This kind of result is referred to as *imperfect aggregation* and is in effect an analytical precomputation of some aggregate decision functions outside the main iterative cycle.

Imperfect aggregation is not an aggregation result in the traditional sense of Gorman (1953) since the behavior of the composite consumer does depend on a joint distribution of heterogeneity parameters. It is a special case of the aggregation in equilibrium point by Constantinides (1982) in which the social utility function can be constructed analytically. (In general, such utility function depends on the distribution of individual characteristics in a manner which is difficult to characterize). Models derived from aggregation are of additional interest because having explicit conditions relating aggregate variables is convenient for numerical work. See Maliar et al. (2011) for imperfect aggregation results in the context of the multicountry models studied in the JEDC comparison analysis.

9. LOCAL (PERTURBATION) METHODS

A local perturbation method computes a solution to an economic model in just one point—a deterministic steady state—using Taylor expansions of optimality conditions. Perturbation methods are introduced to economics in Judd and Guu (1993) and became a popular tool in the literature. The main advantage of perturbation methods is their low computational expense. The main shortcoming is that the accuracy of perturbation solutions may decrease rapidly away from the steady state. In this section, we discuss two techniques that can increase the accuracy of perturbation methods: a change of variables introduced in Judd (2003) and a hybrid of local and global solutions advocated in Maliar et al. (2013).[16] We present these techniques in the context of Euler equation methods; however, our results apply to DP methods as well. See Judd (1998) for perturbation methods in the context of value function iteration.

9.1 Plain Perturbation Method

To present the perturbation techniques, we use a version of the one-agent model (1)–(3) with inelastic labor supply:

$$\max_{\{k_{t+1},c_t\}_{t=0,\ldots,\infty}} E_0 \left\{ \sum_{t=0}^{\infty} \beta^t u\left(c_t\right) \right\} \tag{118}$$

$$\text{s.t. } c_t + k_{t+1} = k_t + \theta_t f\left(k_t\right), \tag{119}$$

$$\ln \theta_{t+1} = \rho \ln \theta_t + \sigma \epsilon_{t+1}, \quad \epsilon_{t+1} \sim \mathcal{N}\left(0,1\right), \tag{120}$$

where (k_0, θ_0) is given. This model's presentation differs from that in (1)–(3) in two respects: First, we use a separate notation for a net of depreciation production function

[16] There is another technique that can help us to increase the accuracy of perturbation methods, namely, computing Taylor expansions around stochastic steady state instead of deterministic one. Two variants of this technique are developed in Juillard (2011) and Maliar and Maliar (2011): the former paper computes the stochastic steady state numerically, while the latter paper uses analytical construction based on precomputation of integrals of Judd et al. (2011d). We do not survey this technique in detail because it is not yet tested in the context of large-scale models.

$\theta_t f(k_t) \equiv \theta_t F(k_t) - \delta k_t$, where F is the gross production function. This helps us to simplify the presentation. Second, we explicitly introduce σ into the process for productivity (120). This is done because we compute a Taylor expansion not only with respect to the two state variables (k_t, θ_t) but also with respect to parameter σ. In addition, we assume that $u(c_t) = \frac{c_t^{1-\gamma}-1}{1-\gamma}$.

We approximate the capital decision function in the form $k' = K(k, \theta; \sigma)$ using the Taylor expansion around the deterministic steady state $k^* = K(k^*, 1; 0)$, $\theta^* = 1, \sigma^* = 0$, as follows:

$$\begin{aligned} k' = K(k,\theta;\sigma) &= k^* + K_k(k,\theta;\sigma)|_{k^*,1;0}\left(k-k^*\right) + K_\theta(k,\theta;\sigma)|_{k^*,1;0}(\theta-1) \\ &+ K_\sigma(k,\theta;\sigma)|_{k^*,1;0}(\sigma-0) + \frac{1}{2}K_{kk}(k,\theta;\sigma)|_{k^*,1;0}\left(k-k^*\right)^2 \\ &+ K_{k\theta}(k,\theta;\sigma)|_{k^*,1;0}\left(k-k^*\right)(\theta-1) + K_{k\sigma}(k,\theta;\sigma)|_{k^*,1;0}\left(k-k^*\right)(\sigma-0) \\ &+ \cdots + \frac{1}{2}K_{\sigma\sigma}(k,\theta;\sigma)|_{k^*,1;0}(\sigma-0)^2 + \cdots \end{aligned}$$

In this section, notation of type G_x and G_{xy} stands for the first-order partial derivative of a function $G(\ldots, x, y, \ldots)$ with respect to a variable x and the second-order partial derivative with respect to variables x and y, respectively. The steady-state value $\theta^* = 1$ follows by (120).

To identify the derivatives of $K(k, \theta; \sigma)$, we use a procedure suggested by an implicit function theorem. We write the Euler equation of problem (118)–(120) as follows:

$$\Gamma(k,\theta;\sigma) \equiv c^{-\gamma} - \beta E\left\{\left(c'\right)^{-\gamma}\left[1 + \theta' f_k(K(k,\theta;\sigma))\right]\right\} = 0, \qquad (121)$$

where θ' is determined by (120), and c and c' are determined by (119) in the current and next periods, respectively,

$$\begin{aligned} \theta' &= \theta^\rho \exp\left(\sigma\epsilon'\right), \\ c &= \theta f(k) + k - K(k,\theta;\sigma), \\ c' &= \theta' f(K(k,\theta;\sigma)) + K(k,\theta;\sigma) - K\left(K(k,\theta;\sigma), \theta'; \sigma\right). \end{aligned}$$

By definition, $\Gamma(k, \theta; \sigma)$ is a zero function in the true solution since the Euler equation must be satisfied exactly for any $(k, \theta; \sigma)$. In particular, the value of Γ must be zero in the steady state, $\Gamma(k^*, 1; 0) = 0$, and all the derivatives of Γ must be also zero in the steady state, $\Gamma_k(k,\theta;\sigma)|_{k^*,1;0} = 0$, $\Gamma_\theta(k,\theta;\sigma)|_{k^*,1;0} = 0$, $\Gamma_\sigma(k,\theta;\sigma)|_{k^*,1;0} = 0$, $\Gamma_{kk}(k,\theta;\sigma)|_{k^*,1;0} = 0$, etc. This fact suggests a procedure for approximating a solution to the model. Namely, we evaluate the value of Γ and its derivatives in the steady state and set them to zero to identify the unknown coefficients in the capital decision function $K(k, \theta; \sigma)$.

We start by finding the steady state. Given $\Gamma_k(k,\theta;\sigma)|_{k^*,1;0} = 0$, (121) implies $1 + f_k(k^*) = 1/\beta$, which identifies $k^* = f_k^{-1}(1/\beta - 1)$ and $c^* = f(k^*)$). Let us show how to derive the first-order perturbation solution.

First, $\Gamma_k(k,\theta;\sigma)$ is given by

$$\Gamma_k(k,\theta;\sigma) = -\gamma c^{-\gamma-1}\frac{\partial c}{\partial k} - \beta E\left\{\left(c'\right)^{-\gamma}\theta^{\rho}\exp\left(\epsilon'\sigma\right)f_{kk}\left(k'\right)K_k(k,\theta;\sigma)\right\}$$
$$+\beta E\left\{\gamma\left(c'\right)^{-\gamma-1}\frac{\partial c'}{\partial k'}K_k(k,\theta;\sigma)\cdot\left[1+\theta' f_k(K(k,\theta;\sigma))\right]\right\},$$

where

$$\frac{\partial c}{\partial k} = 1 + \theta f_k(k) - K_k(k,\theta;\sigma).$$

Note that $\frac{\partial c}{\partial k}\Big|_{k^*,1;0} = \frac{\partial c'}{\partial k'}\Big|_{k^*,1;0}$. Evaluating $\Gamma_k(k,\theta;\sigma)$ in the steady state and equalizing it to zero, we obtain

$$\Gamma_k\left(k^*,1;0\right) = -\gamma\left(c^*\right)^{-\gamma-1}\left[\frac{1}{\beta} - K_k\left(k^*,1;0\right)\right] - \beta\left(c^*\right)^{-\gamma}f_{kk}\left(k^*\right)K_k\left(k^*,1;0\right)$$
$$+\gamma\left(c^*\right)^{-\gamma-1}\left[\frac{1}{\beta} - K_k\left(k^*,1;0\right)\right]K_k\left(k^*,1;0\right) = 0.$$

Rearranging the terms yields

$$\gamma\left(c^*\right)^{-\gamma-1}\left[-\left(K_k\left(k^*,1;0\right)\right)^2 + K_k\left(k^*,1;0\right)\left(1+\frac{1}{\beta}-\frac{\beta c^*}{\gamma}f_{kk}\left(k^*\right)\right)-\frac{1}{\beta}\right] = 0.$$

This is a quadratic equation in one unknown, $K_k(k^*,1;0)$.

Second, $\Gamma_\theta(k,\theta;\sigma)$ is given by

$$\Gamma_\theta(k,\theta;\sigma) = -\gamma c^{-\gamma-1}\frac{\partial c}{\partial \theta} - \beta E\left\{(-\gamma)\left(c'\right)^{-\gamma-1}\frac{\partial c'}{\partial \theta}\left[1+\theta' f_k\left(k'\right)\right]\right.$$
$$\left.-\beta c^{-\gamma}\left[\frac{\partial \theta'}{\partial \theta}f_k\left(k'\right) + \theta' f_{kk}\left(k'\right)K_\theta(k,\theta;\sigma)\right]\right\},$$

where

$$\frac{\partial c}{\partial \theta} = f(k) - K_\theta(k,\theta;\sigma),$$
$$\frac{\partial c'}{\partial \theta} = K_\theta(k,\theta;\sigma)\left[1+\theta' f_k\left(k'\right)\right] - K_k\left(k',\theta';\sigma\right)$$
$$+\rho\theta^{\rho-1}\exp\left(\sigma\epsilon'\right)\cdot\left[K_\theta\left(k',\theta';\sigma\right)+f\left(k'\right)\right],$$
$$\frac{\partial \theta'}{\partial \theta} = \rho\theta^{\rho-1}\exp\left(\sigma\epsilon'\right).$$

Evaluating $\Gamma_\theta(k, \theta; \sigma)$ in the steady state and equalizing it to zero, we obtain

$$\Gamma_\theta\left(k^*, 1; 0\right) = -\gamma\left(c^*\right)^{-\gamma-1}\left[f\left(k^*\right) - \left(1 + \frac{1}{\beta} + \rho - \frac{\beta c^*}{\gamma} f_{kk}\left(k^*\right)\right) K_\theta\left(k^*, 1; 0\right) \cdot \right.$$
$$\left. + \rho f\left(k^*\right) + \frac{\beta c^*}{\gamma}\rho f_k\left(k^*\right) - K_k\left(k^*, 1; 0\right)\right] = 0.$$

This gives us a solution for $K_\theta(k^*, 1; 0)$:

$$K_\theta\left(k^*, 1; 0\right) = \frac{f(k^*)(1-\rho) - \frac{\beta c^*}{\gamma}\rho f_k(k^*)}{1 + \frac{1}{\beta} - \rho - K_k(k^*, 1; 0) - \frac{\beta c^*}{\gamma} f_{kk}(k^*)}.$$

Finally, $\Gamma_\sigma(k, \theta; \sigma)$ is given by

$$\Gamma_\sigma(k, \theta; \sigma) = -\gamma c^{-\gamma-1}\frac{\partial c}{\partial \sigma} - \beta E\left\{(-\gamma)\left(c'\right)^{-\gamma-1}\frac{\partial c'}{\partial \sigma}\left[1 + \theta' f_k\left(k'\right)\right]\right.$$
$$\left. - c^{-\gamma}\theta' f_{kk}\left(k'\right) K_\sigma(k, \theta; \sigma) + \frac{\partial \theta'}{\partial \sigma} k'\right\},$$

where

$$\frac{\partial c}{\partial \sigma} = -K_\sigma(k, \theta; \sigma),$$
$$\frac{\partial c'}{\partial \sigma} = \left[1 + \theta' f_k\left(k'\right) - K_k\left(k', \theta'; \sigma\right)\right] K_\sigma(k, \theta; \sigma) - K_\sigma\left(k', \theta'; \sigma\right)$$
$$+ \frac{\partial \theta'}{\partial \sigma}\left[f\left(k'\right) - K_{\theta'}\left(k', \theta'; \sigma\right)\right],$$
$$\frac{\partial \theta'}{\partial \sigma} = \theta^\rho \epsilon' \exp\left(\sigma \epsilon'\right),$$

$$\frac{\partial c'}{\partial \sigma} = \frac{\partial \theta'}{\partial \sigma} f(k') + \theta' f_k(k') K_\sigma(k, \theta; \sigma) + K_\sigma(k, \theta; \sigma)$$
$$- \left[K_{k'}\left(k', \theta'; \sigma\right) K_\sigma(k, \theta; \sigma) + K_{\theta'}\left(k', \theta'; \sigma\right)\frac{\partial \theta'}{\partial \sigma} + K_\sigma\left(k', \theta'; \sigma\right)\right].$$

Evaluating $\Gamma_\sigma(k, \theta; \sigma)$ in the steady state and equalizing it to zero, we obtain

$$\Gamma_\sigma\left(k^*, 1; 0\right) = K_\sigma\left(k^*, 1; 0\right)\gamma\left(c^*\right)^{-\gamma-1} \cdot$$
$$\cdot\left[\frac{\beta c^*}{\gamma} f_{kk}\left(k^*\right) - \frac{1}{\beta} + K_k\left(k^*, 1; 0\right)\right] = 0.$$

This implies $K_\sigma(k^*, 1; 0) = 0$.

After all the derivatives of $K(k, \theta; \sigma)$ are found, we can form a first-order perturbation solution for capital,

$$k' \approx k^* + K_k\left(k^*, 1; 0\right)\left(k - k^*\right) + K_\theta\left(k^*, 1; 0\right)(\theta - 1) + K_\sigma\left(k^*, 1; 0\right)(\sigma - 0). \quad (122)$$

Similarly, we can write a first-order perturbation solution for consumption $C(k, \theta; \sigma)$,

$$c \approx c^* + C_k\left(k^*, 1; 0\right)\left(k - k^*\right) + C_\theta\left(k^*, 1; 0\right)(\theta - 1) + C_\sigma\left(k^*, 1; 0\right)(\sigma - 0), \quad (123)$$

where $C_k(k^*, 1; 0) \equiv \frac{\partial c}{\partial k}\Big|_{k^*,1;0}$, $C_\theta(k^*, 1; 0) \equiv \frac{\partial c}{\partial \theta}\Big|_{k^*,1;0}$, and $C_\sigma(k^*, 1; 0) \equiv \frac{\partial c}{\partial \sigma}\Big|_{k^*,1;0}$. Finally, the first-order perturbation approximation for the productivity shocks $\Theta(\theta; \sigma)$ is

$$\theta' \approx \theta^* + \Theta_\theta(1; 0)(\theta - 1) + \Theta_\sigma(1; 0)(\sigma - 0), \quad (124)$$

where $\Theta_\theta(1; 0) \equiv \frac{\partial \theta'}{\partial \theta}\Big|_{1;0} = \rho\theta^{\rho-1}\exp(\sigma\epsilon')\Big|_{1;0} = \rho$ and $\Theta_\theta(1; 0) \equiv \frac{\partial \theta'}{\partial \sigma}\Big|_{1;0} = \theta^\rho \epsilon' \exp(\sigma\epsilon')\Big|_{1;0} = 0$. The first-order terms $K_\sigma(k^*, 1; 0)$, $C_\sigma(k^*, 1; 0)$, and $\Theta_\sigma(1; 0)$ are equal to zero in (122), (123) and (124), respectively. However, the higher-order derivatives of K, C, and Θ with respect to σ are not equal to zero, which makes higher-order perturbation solutions depend on the volatility of shocks.

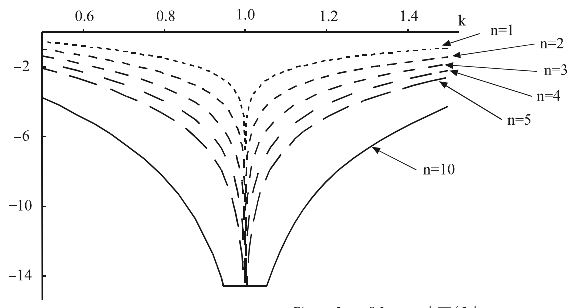
9.2 Advantages and Shortcomings of Perturbation Methods

Constructing perturbation solutions by hand is tedious even in the case of first-order approximation, as our example demonstrates. Fortunately, perturbation methods can be easily automated. Perturbation software commonly used in economics is Dynare. This software platform can solve, simulate, and estimate a wide class of economic models and delivers standard perturbation solutions up to the third order.[17] Automated software for solving economic models is also developed by Jin and Judd (2002), Swanson et al. (2002), and Schmitt-Grohé and Uribe (2004). There are perturbation methods that can solve models with kinks using penalty functions; see Kim et al. (2010) and Mertens and Judd (2013).

An important advantage of perturbation methods is that they have low computational expense, which makes them an ideal candidate for solving problem with high dimensionality. The shortcoming of perturbation methods is that the accuracy of local solutions may decrease rapidly away from the steady state. To provide an illustration, we reproduce some results from Judd and Guu (1993). In Figure 8, we plot the size of unit-free residuals in the Euler equation (in log 10 units) depending on how far we deviate from the steady state where the solution is computed (in this figure, n denotes an order of approximation).

We observe that the linear perturbation solutions are very inaccurate. The quadratic perturbation solutions are more accurate but are still not acceptable even in a relatively small range around the steady state (e.g., $k \in [0.8, 1.2]$). The perturbation solutions of

[17] See http://www.dynare.org and Adjemian et al. (2011) for more details on Dynare.



Graph of $\log_{10}|E(k)|$

- Linear approximation is very poor even for k close to steady state
- Order 2 is better but still not acceptable for even $k = .9, 1.1$
- Order 10 is excellent for $k \in [.6, 1.4]$

Figure 8 Global Quality of Asymptotic Approximations. *Notes:* Reproduced from Judd and Guu (1993) with kind permission from Springer Science+Business.

order 10 are of acceptable accuracy in a large range (e.g., $k \in [0.6, 1.4]$). Thus, high-order approximations are needed to attain accurate solutions. See also Kollmann et al. (2011b) and Judd et al. (2012) for the accuracy results of perturbation methods in the context of a large-scale multicountry model and a new Keynesian model, respectively.[18]

9.3 Change of Variables

Judd (2003) shows that the accuracy of perturbation methods can be considerably increased using a nonlinear change of variables. Specifically, he shows that the ordinary Taylor series expansion can be dominated in accuracy by other expansions implied by changes of variables. All the expansions are locally equivalent but differ globally. We must choose the one that performs best in terms of accuracy on the domain of interest. In the context of a deterministic optimal growth model, Judd (2003) finds that using alternative expansions can increase the accuracy of the conventional perturbation method by two orders of magnitude. Fernández-Villaverde and Rubio-Ramírez (2006) show how to apply the method of change of variables to a model with uncertainty and elastic labor supply.

9.3.1 An Example of the Change of Variables Technique

Let us consider the one-agent model (118)–(120) with inelastic labor supply as an example. Suppose $k' = K(k, \theta; \sigma)$ is approximated as was discussed in Section 9 and that we have computed all the derivatives $K_k(k^*, 1; 0)$, $K_\theta(k^*, 1; 0)$, and $K_\sigma(k^*, 1; 0)$. The

[18] In particular, in the comparison analysis of Kollmann et al. (2011b), the first- and second-order perturbation methods of Kollmann et al. (2011a) produce maximum residuals of 6.3% and 1.35% on a stochastic simulation, and they produce maximum residuals of 65% and 50% on a 30% deviation from the steady state. Moreover, perturbation quadratic solutions are up to three orders of magnitude less accurate than global quadratic solutions.

objective is to find a change of variables $x = \varphi(k)$ and $x' = \varphi(k')$, with x being a new variable, such that a Taylor approximation (122) of x' in terms of a new variable x has two properties: (i) the derivatives are the same in the steady state; and (ii) the accuracy declines less rapidly when we deviate from the steady state.

Using the identity, $k = \varphi^{-1}(\varphi(k))$, we rewrite the capital function $k' = K(k, \theta; \sigma)$ as $\varphi^{-1}(\varphi(k')) = K\left(\varphi^{-1}(\varphi(k)), \theta; \sigma\right)$. With this result, we obtain a decision function for x

$$x' = \varphi\left(K\left(\varphi^{-1}(x), \theta; \sigma\right)\right). \tag{125}$$

Let us find a first-order Taylor expansion of the decision function for x (125) around $x^*, \theta^* = 1$, and $\sigma^* = 0$:

$$\begin{aligned} x' &\approx x^* + \left.\frac{\partial x'}{\partial x}\right|_{x^*,1;0}\left(x - x^*\right) + \left.\frac{\partial x'}{\partial \theta}\right|_{x^*,1;0}\left(\theta - \theta^*\right) + \left.\frac{\partial x'}{\partial \sigma}\right|_{x^*,1;0}(\sigma - 0) \\ &= x^* + \varphi'\left(K\left(\varphi^{-1}(x), \theta; \sigma\right)\right)\Big|_{x^*,1;0} \times \left\{ \frac{K_k\left(\varphi^{-1}(x), \theta; \sigma\right)\Big|_{x^*,1;0}}{\varphi'(k)\big|_{k^*}}\left(x - x^*\right) \right. \\ &\quad \left. + K_\theta\left(\varphi^{-1}(x), \theta; \sigma\right)\Big|_{x^*,1;0}(\theta - 1) + K_\sigma\left(\varphi^{-1}(x), \theta; \sigma\right)\Big|_{x^*,1;0}\sigma \right\}. \end{aligned}$$

Substituting the steady-state value k^* instead of $\varphi^{-1}(x^*)$, we obtain the formula for the change of variables

$$x' \approx \varphi\left(k^*\right) + K_k\left(k^*, 1; 0\right)\left(x - x^*\right) + \varphi'\left(k^*\right)\left\{K_\theta\left(k^*, 1; 0\right)(\theta - 1) + K_\sigma\left(k^*, 1; 0\right)\sigma\right\}. \tag{126}$$

Consider now specific examples. In the trivial case, $x = k = \varphi(k)$, we have $\varphi'(k^*) = 1$, and we are back to (122). Under a logarithmic transformation of, i.e., $x = \varphi(k) = \ln(k)$, condition (126) implies

$$\begin{aligned} \ln k' &\approx \ln k^* + K_k\left(k^*, 1; 0\right)\left(\ln k - \ln k^*\right) \\ &\quad + \frac{K_\theta\left(k^*, 1; 0\right)}{k^*}(\theta - 1) + \frac{K_\sigma\left(k^*, 1; 0\right)}{k^*}\sigma. \end{aligned} \tag{127}$$

For a power transformation $x = \varphi(k) = k^\eta$, we obtain

$$\begin{aligned} \left(k'\right)^\eta &\approx \left(k^*\right)^\eta + K_k\left(k^*, 1; 0\right)\left(k^\eta - \left(k^*\right)^\eta\right) \\ &\quad + \eta\frac{K_\theta\left(k^*, 1; 0\right)}{\left(k^*\right)^{1-\eta}}(\theta - 1) + \eta\frac{K_\sigma\left(k^*, 1; 0\right)}{\left(k^*\right)^{1-\eta}}\sigma. \end{aligned} \tag{128}$$

In this way, we can try many different transformations (at a low cost). For each candidate change of variables, we evaluate accuracy using a selected accuracy test (e.g., we can evaluate the Euler equation residuals on a stochastic simulation), and we choose the representation that yields the most accurate solution globally.

Table 15 Perturbation in levels and logarithms in the one-agent model.[a]

Order	Perturbation in levels	Perturbation in logarithms
1	−1.25	−1.50
2	−1.50	−3.29
3	−1.72	−3.92
4	−1.92	−4.50

[a] *Notes:* The numbers in the table are the maximum absolute Euler equation residuals (in log10 units); $u(c) = \ln c$; $f(k) = k + 4/19k^{1/4}$. These results are reproduced from Judd (2003), Table 2.

Three observations are in order. First, we need not compute anything new to implement the change of variables: the derivatives K_k, K_θ, and K_σ which determine the coefficients in (127) and (128) are the same as that we found in Section 9.1 using the plain perturbation method. Second, in the above example, we apply the same change of variables to k and k'. More generally, we can use other transformations. For example, for k, we may keep the same transformation as before, $x = \varphi(k)$, but for k', we may use a different transformation $z' = \phi(k')$ so that $k' = \phi^{-1}(z')$. Finally, we may also apply some changes of variables to c and θ. Ideally, a selection of the right change of variables must be automated; see Judd (2003) for more details.

9.3.2 Numerical Illustration of the Change of Variables Technique

For a simple one-agent model with inelastic labor supply, Judd (2003) shows that logarithmic and power transformations of the variables may decrease the maximum residuals in the optimality conditions by several orders of magnitude. In Table 15, we reproduce his results for the logarithmic transformations of k and k' (perturbation in logarithms) compared to ordinary perturbation in levels. As we can see, perturbation in logarithms produces significantly more accurate approximations than perturbation in levels; however, this accuracy ranking is model-specific and does not need to hold for other models.

9.4 Hybrid of Local and Global Solutions

Maliar et al. (2013) develop a hybrid perturbation-based solution method that combines local and global approximation techniques. (The term *hybrid method* indicates that we combine policy functions produced by different solution methods.) This hybrid method computes a plain perturbation solution, fixes some perturbation functions, and replaces the rest of the perturbation functions by new functions that are constructed to satisfy the model's conditions exactly. The construction of these new functions mimics global solution methods: for each point of the state space considered, we solve nonlinear equations either analytically (when closed-form solutions are available) or with a numerical solver. If the perturbation functions that were used to construct a hybrid solution are accurate, then the entire hybrid solution will inherit their high accuracy; such a hybrid solution may be far more accurate than the original perturbation solution. The cost of the hybrid method is essentially the same as that of the standard perturbation method. The hybrid method of Maliar et al. (2013) encompasses previous examples constructed on the basis of the standard loglinearization method in Dotsey and Mao (1992) and Maliar et al. (2011).

9.4.1 Description of the Hybrid Method

To exposit the idea of the hybrid method, we use the one-agent model (118)–(120) with inelastic labor supply. We assume that a plain perturbation method delivers an approximate solution in the form of two decision functions $\widehat{K} \approx K$ and $\widehat{C} \approx C$, such as (122), (123) constructed in Section 9.1. Let us assume that the decision functions are not sufficiently accurate for our purpose.

We attempt to improve on the accuracy in the following way. We fix one decision function from the benchmark solution, for example, $\widehat{K}$, and we solve for the other decision function, $\widetilde{C} \equiv \widetilde{C}(\widehat{K})$, to satisfy some model's nonlinear optimality conditions taking $\widehat{K}$ as given.

For model (118)–(120), we have two optimality conditions,

$$u'(c_t) = \beta E_t \left\{u'(c_{t+1}) \left[1 + \theta_{t+1} f'(k_{t+1})\right]\right\}, \tag{129}$$

$$k_{t+1} = k_t + \theta_t f(k_t) - c_t. \tag{130}$$

By considering all possible combinations of the two decision functions and the two optimality conditions, we construct four hybrid solutions $\left\{\widehat{K}, \widetilde{C}_{BC}\right\}, \left\{\widehat{K}, \widetilde{C}_{EE}\right\}, \left\{\widehat{C}, \widetilde{K}_{BC}\right\}$, and $\left\{\widehat{C}, \widetilde{K}_{EE}\right\}$, as follows:

HYB1: Fix $\widehat{K}$ and define $\widetilde{C}_{BC}$ using (130),

$$\widetilde{C}_{BC}(k_t, \theta_t) = k_t + \theta_t f(k_t) - \widehat{K}(k_t, \theta_t).$$

HYB2: Fix $\widehat{K}$ and define $\widetilde{C}_{EE}$ using (129),

$$u'\left(\widetilde{C}_{EE}(k_t, \theta_t)\right) = \beta E_t \left\{u'\left[\widetilde{C}_{EE}\left(\widehat{K}(k_t, \theta_t), \theta_{t+1}\right)\right]\left[1 + \theta_{t+1} f'\left(\widehat{K}(k_t, \theta_t)\right)\right]\right\}.$$

HYB3: Fix $\widehat{C}$ and define $\widetilde{K}_{BC}$ using (130),

$$\widetilde{K}_{BC}(k_t, \theta_t) = k_t + \theta_t f(k_t) - \widehat{C}(k_t, \theta_t).$$

HYB4: Fix $\widehat{C}$ and define $\widetilde{K}_{EE}$ using (129),

$$u'\left(\widehat{C}(k_t, \theta_t)\right) = \beta E_t \left\{u'\left(\widehat{C}\left(\widetilde{K}_{EE}(k_t, \theta_t), \theta_{t+1}\right)\right)\left[1 + \theta_{t+1} f'\left(\widetilde{K}_{EE}(k_t, \theta_t)\right)\right]\right\}.$$

On the basis of this example, we can make the following observations: first, multiple hybrid solutions can be constructed for a given benchmark solution; in our example, there are four hybrid solutions. Second, the hybrid method mimics global solution methods in a sense that functions $\widetilde{C}_{BC}$, $\widetilde{C}_{EE}$, $\widetilde{K}_{BC}$, and $\widetilde{K}_{EE}$ are defined to satisfy the corresponding nonlinear optimality conditions globally, for any point (k_t, θ_t) of the state space considered. Third, a hybrid solution can be either more accurate or less accurate than the benchmark solution. Assume that in the benchmark solution, $\widehat{K}$ is accurate and $\widehat{C}$ is not. Then, the hybrid solutions based on $\widehat{K}$ (i.e., HYB1 and HYB2) will be more accurate, while the hybrid solutions based on $\widehat{C}$ (i.e., HYB3 and HYB4) will be less accurate than the benchmark solution. Finally, hybrid solutions can differ in cost considerably. In our example, HYB1 and HYB3 are obtained using simple closed-form expressions, while HYB2 and HYB4 are defined implicitly and are far more costly to compute.

9.4.2 Numerical Illustration of the Hybrid Method

A potential usefulness of the hybrid solution method is well seen in examples constructed using the standard loglinearization method. In particular, Maliar et al. (2011) take the capital decision function produced by the standard loglinearization method and accurately solve for consumption and labor to satisfy the intertemporal choice conditions (using the iteration-on-allocation method) in the context of a two-agent version studied in the JEDC comparison analysis. In Table 16, we compare the accuracy of the resulting hybrid method, HYB, with that of the plain first-order and second-order perturbation methods (in levels), denoted PER1 and PER2, respectively. As the table indicates, the first-order hybrid method produces more accurate linear approximations (by more than an order of magnitude) than PER1. In terms of the maximum residuals, it is even more accurate than PER2.

9.5 Numerical Instability of High-Order Perturbation Solutions in Simulation

An important shortcoming of plain perturbation methods is a numerical instability in simulation. The instability occurs because perturbation methods produce an approximation that is valid only locally, i.e., in the steady-state point and its small neighborhood. If we use this approximation for simulation, there is a chance that the simulated series become explosive when some realizations of shocks drive the process outside the accuracy range. The chance of nonstationary behavior is higher if the model has a high degree of nonlinearity.

Table 16 Perturbation method versus hybrid method in the two-agent model.[a]

PER1		PER2		HYB	
L_1	L_∞	L_1	L_∞	L_1	L_∞
−3.53	−1.45	−4.84	−2.30	−4.56	−2.84

[a] *Notes:* L_1 and L_∞ are, respectively, the average and maximum of absolute residuals across optimality conditions and test points (in log10 units). PER1 and PER2 are the first- and second-order perturbation solutions (in levels), and HYB are the hybrid solutions. These results are reproduced from Maliar et al. (2011), Table 5.

Pruning methods try to address this shortcoming of the standard perturbation methods. The term *pruning* is introduced by Kim et al. (2008) who point out that a simulation of high-order perturbation solutions may produce explosive time series. To restore numerical stability, they propose to replace cross-products of variables in the second-order perturbation solution with cross-products of variables obtained from the first-order perturbation solution. Other papers that focus on stabilizing perturbation methods are Lombardo (2010) and Den Haan and De Wind (2012). In particular, the latter paper uses a fixed-point iteration technique that is similar in spirit to iteration-on-allocation described in Maliar et al. (2010, 2011) for constructing hybrid-like solutions.

10. PARALLEL COMPUTATION

Technological progress constantly increases the speed of computers. Moore (1965) made an observation that the number of transistors on integrated circuits doubles approximately every 2 years, and the speed of processors doubles approximately every 18 months (both because the number of transistors increases and because transistors become faster). The Moore law continues to hold meaning that in 10 years, computers will become about 100 times faster.

What happens if we cannot wait for 10 years or if a 100-time increase in speed is not sufficient for our purpose? There is another important source of growth of computational power that is available at present, namely, parallel computation: we connect a number of processors together and use them to perform a single job. Serial desktop computers have several central processing units (CPUs) and may have hundreds of graphics processing units (GPUs), and a considerable reduction in computational expense may be possible. Supercomputers have many more cores (hundreds of thousands) and have graphical cards with a huge number of GPUs. Each processor in a supercomputer is not (far) more powerful than a processor on our desktop but pooling their efforts gives them a high computational power. Running a job on 10,000 cores in parallel can increase the speed of our computation up to a factor of 10,000. This is what supercomputers are.

Early applications of parallel computation to economic problems are Amman (1986, 1990), Chong and Hendry (1986), Coleman (1992), Nagurney (1996), Nagurney and Zhang (1998). More recent applications include Doornik et al. (2006), Creel (2005, 2008), Creel and Goffe (2008), Sims et al. (2008), Aldrich et al. (2011), Morozov and Mathur (2012), Durham and Geweke (2012), Cai et al. (2012), Valero et al. (2013), and Maliar (2013), among others.

The possibility of parallel computation raises new issues. First, to take advantage of this technology, we must design algorithms in a manner which is suitable for parallelization. Second, we need hardware and software that support parallel computation. Finally, we must write a code that splits a large job into smaller tasks, that exchanges information between different cores in the process of computation, and that gathers the information to produce final output. We discuss these issues in the remainder of the section.

10.1 Serial Versus Parallel Computation

Assume that we must execute N tasks. Traditional, one-core serial computation requires us to process the tasks one-by-one. The running time for executing N tasks is N times larger than that for executing one task. If we have multiple cores, we can parallelize the computation. The easiest case for parallelization is when tasks are independent and no information is shared (exchanged) during the process of computation. This case is known in computer science literature as *naturally parallelizable jobs*. A more general case requires us to exchange information between the cores (implementing different tasks) during the computational process; see Table 17.

A coordinating unit, called *a core 0* or *master*, assigns tasks to multiple cores (workers) and performs the information exchange if needed. Assuming that the implementation of parallel computation has no additional cost, the parallelized algorithm is N times faster than the serial one if tasks are independent. When the information exchange is needed, the gain from parallelization depends on a specific application and a specific way in which the code is implemented.

10.1.1 Applications with Independent Tasks

The restriction that tasks are independent is obviously limiting. Still, there are many interesting applications for which this restriction is satisfied. In econometrics, we may need to run many regressions of a dependent variable on different combinations of independent variables, and we may run each such a regression on a separate core; see Doornik et al. (2006) for a review of applications of parallel computation in econometrics. In a similar vein, we may need to solve an economic model under a large number of different parameterizations either because we want to study how the properties of the solution depend on a specific parameterization (sensitivity analysis) or because we want to produce multiple data sets for estimating the model's parameters (e.g., nested fixed-point estimation; see Fernández-Villaverde and Rubio-Ramírez (2007), Winschel and Krätzig (2010), and Su and Judd (2012) for related examples). In this case, we solve a model under each given parameter vector on a separate core. Other examples of naturally parallelizable jobs

Table 17 Serial versus parallel computation.

Serial computation	**Parallel computation with independent tasks**	**Parallel computation with dependent tasks**
Assignation of tasks	Assignation of tasks	Interconnection
$\Downarrow$	$\Downarrow$ $\Downarrow$ $\Downarrow$	$\Updownarrow$ $\Updownarrow$ $\Updownarrow$
Task 1	Task 1 … Task N	Task 1 … Task N
…		
Task N		

are matrix multiplication, exhaustive search over a discrete set of elements, optimization of a function over a region of state space, etc.

10.1.2 Applications with Dependent Tasks

In general, a given problem will contain some tasks that can be parallelized, some tasks that cannot be parallelized, and some tasks that can be parallelized but will require information exchange during their implementation (dependence). This is in particular true for the solution methods described in Sections (4)–(9). Loops are an obvious candidate for parallelization. However, after each iteration on a loop, we must typically gather the output produced by all workers and to combine it in order to produce an input for the next iteration. Some of the surveyed computational techniques are designed to be parallelizable. In particular, Maliar et al. (2011) and Maliar and Maliar (2013) propose a separation of the model's equations into the intratemporal and intertemporal choice equations under the Euler equation and dynamic programming methods. Such a separation provides a simple way to parallelize computation. Namely, we can first produce current and future endogenous state variables in all grid points and or integration nodes, and we can then solve for the intratemporal choice in each grid point and integration node using a separate core; see Sections 6 and 7 for a discussion.

10.1.3 Speedup and Efficiency of Parallelization

Two measures that are used to characterize gains from parallelization are the speedup and efficiency of parallelization. The speedup is defined as a ratio

$$\mathcal{S}(N) = \tau_1/\tau_N, \tag{131}$$

where τ_1 and τ_N are the times for executing a job on one core and N cores, respectively. In turn, the efficiency of parallelization is defined as

$$\mathcal{E}(N) = \frac{\tau_1/\tau_N}{N}. \tag{132}$$

The efficiency shows gains from parallelization τ_1/τ_N relative to the number of cores used N. The upper bounds of (131) and (132) are $\mathcal{S}(N) \leq N$ and $\mathcal{E}(N) \leq 1$, and they are attained if a job is perfectly parallelizable and the cost of information transfers is zero.

The typical code has some parts that are parallelizable and other parts that must be executed serially. Nonparallelizable parts of the code restrict the speedup and efficiency of parallelization that can be attained in a given application. Moreover, the speedup and efficiency of parallelization depend on the cost of implementing the parallelization procedure and information transfers.

Consider the case when the parallelizable part of the code consists of a number of tasks that require the same execution time. The following condition can be used to characterize the speedup

$$\mathcal{S}(N) = \frac{\tau_1}{\tau_1^p/N + \tau_1^{np} + \tau^{tf}(N)}, \tag{133}$$

where τ_1^p and τ_1^{np} denote the execution times for parallelizable and nonparallelizable parts of the code, respectively, $\tau_1^p + \tau_1^{np} = \tau_1$, and $\tau^{tf}(N)$ denotes the time for information transfers. The measure (133) is a version of Amdahl's (1967) law, which we augment to include the cost of information transfers $\tau^{tf}(N)$; see Nagurney (1996) for a further discussion.

Two implications of the Amdahl's (1967) law are the following. First, (133) implies that $\mathcal{S}(N) \leq \frac{\tau_1}{\tau_1^{np}}$, i.e., the fraction of the code that is not parallelizable, $\frac{\tau_1}{\tau_1^{np}}$, is a bottleneck for the speedup. For example, if 50% of the code is not parallelizable, i.e., $\frac{\tau_1^{np}}{\tau_1} = 0.5$, we can reduce the running time of a code at most by a factor of 2, no matter how many cores we employ. Second, (133) shows that the speedup can be smaller than 1 (which is parallel slowdown) if the cost of information transfers $\tau^{tf}(N)$ is very large.

It is straightforward to describe the implications of Amdahl's (1967) law for the efficiency measure (132). Namely, efficiency of parallelization decreases with the number of cores $\mathcal{E}(N) = \frac{\tau_1}{\tau_1^p + N\left(\tau_1^{np} + \tau^{tf}(N)\right)}$ reaching zero in the limit $N \to \infty$, which is another way to say that the gains from parallelization are bounded from above by the execution time of tasks that cannot be parallelized. Furthermore, $\mathcal{E}(N)$ decreases with the costs of parallelization and transfer, and it can be arbitrary close to zero if such costs are very large.

Finally, different tasks executed in a parallel manner may differ in the amount of time necessary for their execution. For example, when searching for a maximum of a function over different regions of state space, a numerical solver may need considerably more time for finding a maximum in some regions than in others. The most expensive region will determine the speedup and efficiency of parallelization since all the workers will have to wait until the slowest worker catches up. The cost of information transfers may also differ across parallelizable tasks in some applications. These issues must be taken into account when designing codes for parallel computation.

10.2 Parallel Computation on a Desktop Using MATLAB

Most economic researchers write codes for solving dynamic economic models using a desktop and serial MATLAB software. The advantages of this choice are three: first, the user can concentrate exclusively on computations without being concerned with all subtle issues related to the operational system and computer architecture. Second, communication among multiple cores of a desktop is very fast, and can be much faster than that among cores in a cluster or supercomputer. Finally, MATLAB itself is a convenient choice. It is widespread, well-supported, and has an extensive documentation; it is easy to learn and to use; and it has many preprogrammed routines that can be easily integrated into other codes.

In particular, MATLAB provides a simple way to parallelize computations on either CPUs or GPUs; no additional software is needed. Valero et al. (2013) survey parallel computation tools available in MATLAB; below, we reproduce some discussion from this paper.

First, MATLAB has a "Parallel Computing Toolbox" which allows us to allocate the computational effort in different CPUs on multicore computers; see MATLAB (2011). Parallelizing in MATLAB is simple, we just need to replace "*for*" with its parallel computing version "*parfor*." To use it, we must tell to the computer what part of the code we want to parallelize and when we want to finish parallel computation and to gather the output; this is done using "*matlabpool open/matlabpool close*." Some restrictions must be imposed on variables that are used in parallel computing.

Second, MATLAB has tools for working with GPUs. Functions "*gpuArray*" and "*gather*" transmit data from CPUs to GPUs and vice versa, respectively. It is also possible to generate the data directly in GPUs which can help to save on transfer time, for example, "*parallel.gpu.GPUArray.zeros* (*d*)" creates an array of zeros of dimension *d* in GPUs. Once you allocate an array in a GPU, MATLAB has many functions which allow you to work directly there. A useful function in MATLAB is "*arrayfun*" which, instead of multiple calls to perform separate GPU operations, makes one call to a vectorized GPU operation (it is analogous to MATLAB's vectorized operations). MATLAB has functions that allow us to use native languages for GPU programming; for example, it allows us to use CUDA with the NVIDIA graphical card.[19]

Finally, MATLAB has other tools that are useful for parallel computation. "Jacket" and "GPUMat" toolboxes are useful alternatives to the standard "Parallel Computation Toolbox." Another useful tool is "*deploytool*" which allows us to convert MATLAB codes into executable files. Also, MATLAB codes can be translated to other languages such as C/C++ or Fortran source code using the "*mex*" tool. Depending on a specific version of MATLAB, different software may be needed to create the mex files; we use a Microsoft Windows SDK 7.1 with NET Framework 4.0. Function "*coder.extrinsic*" makes it possible to export to other languages some functions that are specific to MATLAB, e.g., "*tic*" and "*toc*." The above tools allow us to run MATLAB codes on machines that do not have MATLAB installed.

The drawbacks of MATLAB are typical for high-level programming languages: it is relatively slow and does not take into account the hardware characteristics to optimize computations. Also, MATLAB is not free software and can be executed only on those machines that have licenses.

Many other programming languages can be used on a desktop instead of MATLAB. Octave is almost identical to MATLAB and is freeware. Another freeware is Python, a

[19] Currently, NVIDIA graphical cards are the most developed ones for scientific computation, and they use a language that has a basis in C, called CUDA; see Sanders and Kandrot (2010).

high-level language that is growing in popularity. Significant speedups of calculations can be obtained using low-level languages such as C or Fortran. However, these languages may be more complicated to learn and to use. There are also languages that are specially designed to work with GPUs. A pioneering paper by Aldrich et al. (2010) shows how to apply GPUs using CUDA for solving economic problems; see also Morozov and Mathur (2012). CUDA can only be used with NVIDIA GPUs. However, there is software that can manage other kinds of GPUs, for example, OPEN CL (see http://www.khronos.org), VIENNA CL (http://viennacl.sourceforge.net); see Gallant's (2012) conference slides for a concise and informative discussion and further references.

10.2.1 Numerical Example of GPU Computation Using MATLAB

The following example is borrowed from Valero et al. (2013). Consider a function with a unique input x, which is randomly drawn from a uniform distribution $[0, 1]$

$$y = \sin(3x) + \cos(\pi x) + \frac{x^5}{5} + \sqrt{x}\arccos(x) + 8x\exp(x). \tag{134}$$

We approximate the expectation of y using a Monte Carlo integration method $E(y) \approx \frac{1}{n}\sum_{i=1}^{n} y_i$.

The calculations are implemented in GPUs using MATLAB. We report separately the cases with a single and double precision because the precision significantly affects the speed of GPU computations. To illustrate the cost of transfers between CPUs and GPUs, we report two speedups (a ratio of CPU to GPU running times, see definition (131)): the speedup without information transfers between CPUs and GPUs and that with such transfers.[20] We first report the results obtained with "*gpuArray*" and "*gather*" functions in Figure 9a and b for the single and double precision cases, respectively.

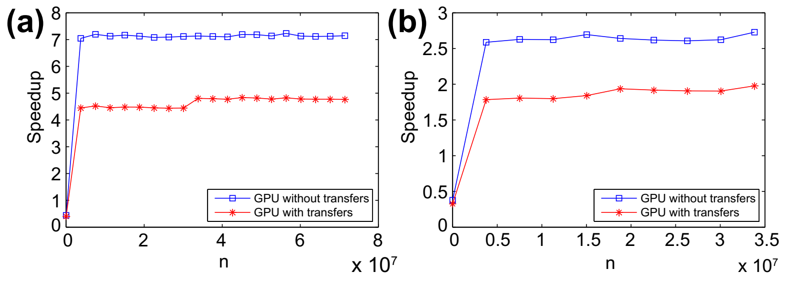


Figure 9 (a) Speedup with GPUs: single precision. (b) Speedup with GPUs: double precision.

[20] The comparison here is not completely fair. By default, MATLAB makes the use of multithreading capabilities of the computer. If we limit MATLAB to a single computational thread, all the reported speedups will increase by around a factor of 3–4.

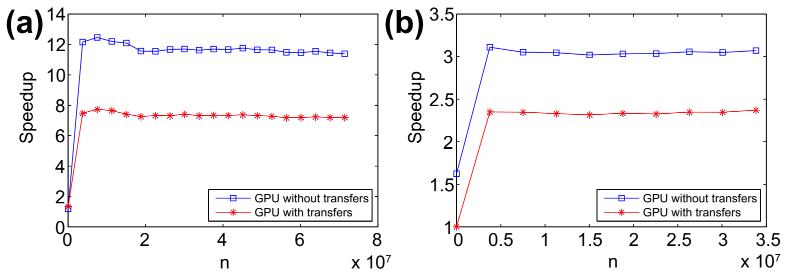


Figure 10 (a) Speedup with GPUs (vectorized): single precision. (b) Speedup with GPUs (vectorized): double precision.

If the transfer time is included, we observe speedups of order 4.5 and 1.7 under the single and double precision, respectively, when n is sufficiently large (for small n, the speedup is negative as the gain from GPUs does not compensate for the high cost of transferring information between CPUs and GPUs).

We next repeat the computations using a vectorized version of the code built on "*arrayfun*"; see Figure 10a and 10b for single and double precision, respectively.

This improves the performance of GPUs considerably. Now, the speedups are of order 7 and 2.4 under the single and double precision, respectively. "*Arrayfun*" also allows for better allocations of memory. Our hardware for GPU computations is: Intel$^{(\circledR)}$ Core$^{(\text{TM})}$ i7-2600 CPU @ 3.400 GHz with RAM 12.0 GB and GPU GeFoce GT 545, and we use MATLAB 2012a.

10.3 Parallel Computation on Supercomputers

Clusters of computers and supercomputers can solve certain problems of very large size. Clusters are networks that have 16–1,000 cores, and supercomputers may have hundreds of thousands of cores, as well as graphical cards with a huge number of GPUs. However, two issues must be mentioned. First, the information exchange between CPUs or GPUs is generally slower than that between cores on a desktop, which may reduce the gains from parallelization in some applications. Second, using supercomputers requires certain knowledge of the computer architecture and the operational system (typically, Unix), as well as software that distributes and exchanges information among different cores, and the programming can be a nontrivial task.

Three different types of supercomputers are distinguished in the computer science literature; see Blood (2011).

1. High-performance computing (HPC) runs one large application across multiple cores. The user is assigned a fixed number of processors for a fixed amount of time, and this time is over if not used.

2. High-throughput computing (HTC) runs many small applications at once. The user gets a certain number of cores that are available at that time, and this computer time would be wasted otherwise.
3. Data-intensive computing focuses on input-output operations, where data manipulation dominates computation.

We are primarily interested in the first two types of supercomputing, HPC and HTC. An important issue for parallel computation is how to share the memory. Two main alternatives are shared memory and distributed memory.

- *Shared memory programming.* There is a global memory which is accessible by all processors, although processors may also have their local memory. For example, OpenMP software splits loops between multiple threads and shares information through common variables in memory; see http://www.openmp.org.
- *Distributed memory programming.* Processors possess their own memory and must send messages to each other in order to retrieve information from memories of other processors. MPI is a commonly used software for passing messages between the processors; see http://www.mpi-forum.org.

The advantage of shared memory is that it is easier to work with and it can be used to parallelize already existing serial codes. The drawbacks are that the possibilities of parallelization are limited and that sharing memory between threads can be perilous. The advantage of distributed memory is that it can work with a very large number of cores and is ubiquitous but it is also more difficult to program. There are also hybrids that use distributed memory programming for a coarse parallelization and that use shared memory programming for a fine parallelization.

High computational power becomes increasingly accessible to economists. In particular, the eXtreme Science and Engineering Discovery Environment (XSEDE) portal financed by the NSF provides access to supercomputers for US academic/nonprofit institutions. Currently, XSEDE is composed of 17 service providers around the world; see https://portal.xsede.org. We discuss two examples of supercomputers within the XSEDE portal, namely, Blacklight and Condor; see http://www.psc.edu/index.php/computing-resources/blacklight and https://www.xsede.org/purdue-condor, respectively.

Blacklight is an example of an HPC machine. It consists of 256 nodes each of which holds 16 cores, 4096 cores in total. Each core has a clock rate of 2.27 GHz and 8 Gbytes of memory. The total floating point capability of the machine is 37 Tflops, and the total memory capacity of the machine is 32 Tbytes. Blacklight has many software packages installed including C, C++, Fortran, R, Python, MATLAB, etc., as well as facilities for running MPI and OpenMP programs. See Maliar (2013) for an assessment of the efficiency of parallelization using MPI and OpenMP on a Blacklight supercomputer.

Condor is an example of an HTC machine. It is composed of a large net of computers. Computers in the net belong to priority users and are not always free (our own computers can become a part of the Condor network if we give them a permission). Condor software detects computers that are not currently occupied by priority users and assigns tasks to them. It passes messages between masters and workers, queues the tasks, detects failures and interruptions, collects the output, and delivers it to users. The Condor network is slower than that of HPC machines but the speed of communication is not essential for many applications, in particular, for those that are naturally parallelizable. Cai et al. (2013a,b) show how to use the Condor network to solve dynamic programming problems.

Computer time can be also bought in the Internet at relatively low prices. For example, Amazon Elastic Compute Cloud provides the possibility to pay for computing capacity by the hour; see http://aws.amazon.com/ec2/#pricing.

MATLAB is of a limited use on supercomputers. For example, at the moment, Blacklight has MATLAB licenses just on 32 cores, meaning that only a tiny fraction of its total capacity is available to MATLAB users. Such users have two alternatives. The first one is to convert MATLAB files into executable files as was discussed earlier (this can work well if we have independent tasks). The other alternative is to use Octave, which is a freeware clone of MATLAB. The problem is that there are still some differences between MATLAB and Octave, and it could happen that the MATLAB code does not work under Octave as expected. For working with a supercomputer, a better alternative is to use languages that are freeware and have no restrictions on the number of licenses (C, R, Python, etc.). These languages have developed parallel programming tools that can be used with MPI or OpenMP.

10.3.1 Numerical Example of Parallel Computation Using a Blacklight Supercomputer

For desktops, the information exchange between CPUs is very fast. For supercomputers, the information exchange is far slower and may reduce dramatically gains from parallelization even in applications that are naturally suitable for parallelization. It is therefore of interest to determine how large a task per core should be to obtain sufficiently high gains from parallelization on supercomputers.

In the following example, we assess the efficiency of parallelization using MPI on a Blacklight supercomputer; this example is borrowed from Maliar (2013). Let us again consider the problem of approximating the expectation of y, defined in (134) using Monte Carlo integration. The calculations are implemented on Blacklight using C and MPI (with point-to-point communication). In the code, each core (process) runs a copy of the executable (single program, multiple data), takes the portion of the work according to its rank, and works independently of the other cores, except when communicating. For each simulation length n (the size of the problem), we run four experiments in which we vary the number of cores, 1, 16, 32, and 128. In each experiment, we solve the problem of the same size n. That is, depending on the number of cores assumed, each core performs tasks of the corresponding size. For example, if the number of cores is 16,

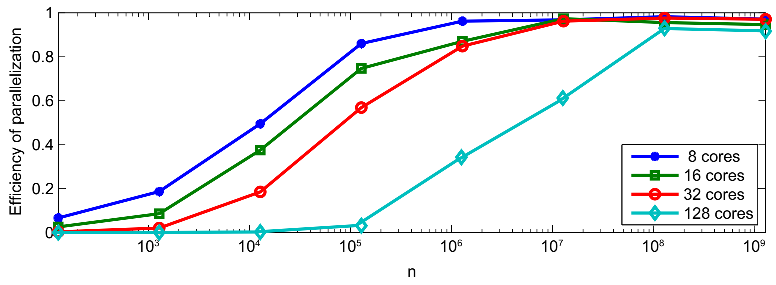


Figure 11 Efficiency of parallelization on Blacklight supercomputer.

each core processes $n/16$ observations (we use n which are multiples of 128). The results are provided in Figure 11.

In the figure, we compute the efficiency of parallelization as defined in (132). For small n, the efficiency of parallelization is low because the cost of information transfer overweighs the gains from parallelization. However, as n increases, the efficiency of parallelization steadily increases, approaching unity. We also observe that the efficiency of parallelization depends on the number of cores used: with 16 cores, the efficiency of parallelization of 90% is reached for 20 s problem (2.5 s per core), while with 128 cores, a comparable efficiency of parallelization is reached only for 2,000 s problem (15.6 s per core). Our sensitivity experiments (not reported) had shown that for larger numbers of cores, the size of the task per core must be a minute or even more to achieve high efficiency of parallelization.

11. NUMERICAL ANALYSIS OF A HIGH-DIMENSIONAL MODEL

In the previous sections, we have surveyed a collection of efficient computational techniques in the context of simple examples. Now, we show how these techniques can be combined into numerical methods that are suitable for solving large-scale economic models.

11.1 The Model

As an example of a high-dimensional problem, we consider a stylized stochastic growth model with N heterogeneous agents (interpreted as countries). Each country is characterized by a capital stock and a productivity level, so that there are $2N$ state variables. By varying N, we can control the dimensionality of the problem. In addition to a potentially large number of state variables, the model features elastic labor supply, heterogeneity in fundamentals, and adjustment cost for capital.

Time is discrete, and the horizon is infinite, $t = 0, 1, \ldots$. The world economy consists of a finite number of countries, $N \geq 1$, and each country is populated by one (representative) consumer. A social planner maximizes a weighted sum of expected lifetime utilities of the consumers

$$\max_{\left\{k_{t+1}^h, c_t^h, \ell_t^h\right\}_{t=0,\ldots,\infty}^{h=1,\ldots,N}} E_0 \left\{ \sum_{h=1}^{N} \tau^h \left(\sum_{t=0}^{\infty} \beta^t u^h \left(c_t^h, \ell_t^h\right) \right) \right\} \tag{135}$$

subject to the aggregate resource constraint, i.e.,

$$\sum_{h=1}^{N} c_t^h = \sum_{h=1}^{N} \left[\theta_t^h f^h \left(k_t^h, \ell_t^h\right) - \frac{\phi}{2} k_t^h \left(\frac{k_{t+1}^h}{k_t^h} - 1 \right)^2 + k_t^h - k_{t+1}^h \right], \tag{136}$$

where E_t is the operator of conditional expectation conditional on k_t and θ_t; $c_t^h, \ell_t^h, k_{t+1}^h$, θ_t^h, u^h, f^h, and τ^h are consumption, labor, end-of-period capital, productivity level, utility function, production function, and welfare weight of a country $h \in \{1, \ldots, N\}$, respectively; $c_t^h, \ell_t^h, k_{t+1}^h \geq 0$; $\beta \in [0, 1)$ is the discount factor; ϕ is the adjustment-cost parameter; and $\delta \in (0, 1]$ is the depreciation rate. Initial condition, $\left(k_0^1, \ldots, k_0^N\right)$ and $\left(\theta_0^1, \ldots, \theta_0^N\right)$, is given. We assume that $\tau^h > 0$ and that u^h and f^h are strictly increasing and strictly concave and satisfy the Inada types of conditions for all h. To simplify the presentation, we consider the production function net of depreciation $\theta_t^h f^h \left(k_t^h, \ell_t^h\right) \equiv \theta_t^h F^h \left(k_t^h, \ell_t^h\right) - \delta k_t^h$, where F^h is a gross production function and δ is the depreciation rate, $\delta \in [0, 1]$.

Future productivity levels of countries are unknown and depend on randomly drawn productivity shocks,

$$\ln \theta_{t+1}^h = \rho \ln \theta_t^h + \sigma \epsilon_{t+1}^h, \tag{137}$$

where $\rho \in (-1, 1)$ is the autocorrelation coefficient of the productivity level; $\sigma > 0$ determines the standard deviation of the productivity level; and $\left(\epsilon_{t+1}^1, \ldots, \epsilon_{t+1}^N\right)^\top \sim \mathcal{N}\left(0_N, \Sigma\right)$ is a vector of productivity shocks with $0_N \in \mathbb{R}^N$ being a vector of zero means and $\Sigma \in \mathbb{R}^{N \times N}$ being a variance-covariance matrix. Thus, we allow for the case when productivity shocks of different countries are correlated. A planner's solution is given by decision functions $c_t^h = C^h \left(\boldsymbol{k}_t, \boldsymbol{\theta}_t\right), \ell_t^h = \mathcal{L}^h \left(\boldsymbol{k}_t, \boldsymbol{\theta}_t\right), k_{t+1}^h = K^h \left(\boldsymbol{k}_t, \boldsymbol{\theta}_t\right)$, and $\lambda_t = \Lambda \left(\boldsymbol{k}_t, \boldsymbol{\theta}_t\right)$, where $\boldsymbol{k}_t \equiv \left(k_t^1, \ldots, k_t^N\right)$ and $\boldsymbol{\theta}_t \equiv \left(\theta_t^1, \ldots, \theta_t^N\right)$.

11.2 Methods Participating in the JEDC Project

Model (135)–(137) has been studied in the February 2011 special issue of the *Journal of Economic Dynamics and Control*. Detailed descriptions of the numerical methods that participate in the JEDC comparison analysis can be found in the individual papers of the participants of the JEDC project. In this section, we provide a brief description of these algorithms and summarize their main implementation details in Table 18.

Perturbation Methods, PER

The *first- and second-order perturbation* methods (PER1) and (PER2) of Kollmann et al. (2011a) compute perturbation solutions in logarithms of the state variables. These two methods approximate the decision functions of all individual variables (consumption,

Table 18 Implementation of the algorithms participating in the JEDC project.[a]

	PER	SSA	CGA	MRGAL	SMOL-MKK
Individual article	Kollmann et al. (2011b)	Maliar et al. (2011)	Maliar et al. (2011)	Pichler (2011)	Malin et al. (2011)
Program. language	MATLAB	MATLAB	MATLAB	MATLAB	Fortran
Functions parameterized	$C^h, \mathcal{L}^h, K^h, I^h, \Lambda$ in logs, for all h	K^h, for all h	K^h, for all h	K^h, for all h and C	$K^h(\cdot)$ for all h and C^1
Functional form	1st and 2d degree compl. ordin. polyn.	1st degree compl. ordin. polyn.	2d degree compl. Cheb. polyn.	2d degree compl. ordin. polyn.	Subset of 4th degree compl. Cheby. poly.
# Basis functions	$1+d$ and $1+d+\frac{d(d+1)}{2}$	$1+d$	$1+d+\frac{d(d+1)}{2}$	$1+d+\frac{d(d+1)}{2}$	$1+4d+4\frac{d(d+1)}{2}$
Number of grid points	–	10,000	500	$2d^2+1$	$1+4d+4\frac{d(d+1)}{2}$
Intratem. choice	–	Iteration-on-allocation	Iteration-on-allocation	Precomputation	Newton's solver
Integration	–	Monte Carlo	Monomial	Monomial	Monomial
Solving for poly. coeff.	Taylor expansions	FPI	FPI	Newton's solver	Time iteration

[a]*Notes:* I^h is the decision function for investment of country h; C is the decision function for aggregate consumption; $d = 2N$ is the model's dimensionality. A part of this table is reproduced from Kollmann et al. (2011b), Table 2.

labor, capital, investment) and the Lagrange multiplier associated with the aggregate resource constraint using Taylor expansions of the equilibrium conditions.

Stochastic Simulation Algorithm, SSA

The *stochastic simulation algorithm* (SSA) of Maliar et al. (2010) is a variant of GSSA that computes linear polynomial approximations using a one-node Monte Carlo integration method. It parameterizes the capital decision functions of all countries, solves for polynomial coefficients of approximating functions using FPI, and solves for consumption and labor nonparameterically, using iteration-on-allocation.

Cluster Grid Algorithm, CGA

The *cluster grid algorithm* (CGA) of Maliar et al. (2011) constructs a grid for finding a solution using methods from cluster analysis; see our discussion in Section 4.3.3. CGA computes quadratic polynomial approximations. It parameterizes the capital decision functions of all countries and it finds the coefficients using FPI. For integration, it uses a monomial rule. CGA computes consumption and labor nonparameterically, using iteration-on-allocation.

Monomial Rule Galerkin Method, MRGAL

The *monomial rule Galerkin* algorithm (MRGAL) of Pichler (2011) uses a set of points produced by monomial rules both as a grid for finding a solution and as a set of nodes for integration. MRGAL uses second-degree Chebyshev polynomials of state variable to approximate the decision functions for capital of all countries and the aggregate consumption function. It solves for the polynomial coefficients using a Newton-type solver that minimizes a weighted sum of residuals in the equilibrium conditions.

Smolyak Method, SMOL-MKK

The *Smolyak-collocation* method (SMOL-MKK) of Malin et al. (2011) solves the model on a Smolyak grid with an approximation level $\mu = 2$. SMOL-MKK approximates the capital decision functions of all countries and the consumption decision function of the first country using a second-degree Smolyak polynomial function of state variables. It uses monomial rules to approximate integrals, and it uses time iteration to compute fixed-point polynomial coefficients.

11.3 Global Euler Equation Methods

We implement several other methods for solving model (135)–(137), in addition to the six solution methods studied in the JEDC comparison analysis. Below, we elaborate a description of the three global Euler equation methods, SMOL-JMMV, GSSA, and EDS, outlined in Sections 4.1, 4.2, and 4.3, respectively. We compute both second- and third-degree polynomial approximations whereas the solution methods of the JEDC project are limited to second-degree polynomial approximations. We summarize the implementation of these methods in Table 19.

Table 19 Implementation of our Euler equation algorithms.[a]

	SMOL-JMMV	**GSSA**	**EDS**
Articles	Judd et al. (2013)	Judd et al. (2011b)	Judd et al. (2012)
Program. language	MATLAB	MATLAB	MATLAB
Functions parameterized	K^h for all h	K^h for all h	K^h for all h
Functional form	Subset of 4^{th} degree compl. Cheb. poly.	2^d and 3^d degree compl. ordin. polyn.	2^d and 3^d degree compl. ordin. poly.
# Basis functions	$1 + 4d + 4\frac{d(d+1)}{2}$	$1 + d + \frac{d(d+1)}{2}$ and n_3	$1 + d + \frac{d(d+1)}{2}$ and n_3
Number of grid points	$1 + 4d + 4\frac{d(d+1)}{2}$	2000	$2\left(1 + d + \frac{d(d+1)}{2}\right)$ and $2n_3$
Intratemporal choice	Iteration-on-allocation	Iteration-on-allocation	Iteration-on-allocation
Integration	Monomial	MC, monomial, quadrature	Monomial
Solving for polyn. coeff.	FPI	FPI	FPI

[a] *Notes:* "MC" means Monte Carlo integration; d is the model's dimensionality, $d = 2N$; FPI means fixed-point iteration; $n_3 = 1 + d + d(d+1)/2 + d^2 + \frac{d(d-1)(d-2)}{6}$ is the number of basis functions in complete ordinary polynomial of degree 3.

11.3.1 First-Order Conditions

We assume that the planner's solution to model (135)–(137) is interior and, hence, satisfies the FOCs given by

$$\lambda_t \left[1 + \phi \left(\frac{k_{t+1}^h}{k_t^h} - 1\right)\right] = \beta E_t \left\{\lambda_{t+1}\left[1 + \frac{\phi}{2}\left(\left(\frac{k_{t+2}^h}{k_{t+1}^h}\right)^2 - 1\right) + \theta_{t+1}^h f_k^h\left(k_{t+1}^h, \ell_{t+1}^h\right)\right]\right\}, \quad (138)$$

$$u_c^h\left(c_t^h, \ell_t^h\right)\tau^h = \lambda_t, \quad (139)$$

$$u_\ell^h\left(c_t^h, \ell_t^h\right)\tau^h = -\lambda_t \theta_t^h f_\ell^h\left(k_t^h, \ell_t^h\right), \quad (140)$$

where λ_t is the Lagrange multiplier associated with aggregate resource constraint (136). Here, and further on, notation of type F_x stands for the first-order partial derivative of a function $F(\ldots, x, \ldots)$ with respect to a variable x.

11.3.2 Separating the Intertemporal and Intratemporal Choices

The global Euler equation methods aim at solving (136)–(140). We separate the equilibrium conditions into the intertemporal and intratemporal choice conditions as described in Section 6.

Regarding the intertemporal choice, we combine (138) and (139) to eliminate λ_t, premultiply both sides with k_{t+1}^h, and rearrange the terms to obtain

$$k_{t+1}^h = \beta E_t \left\{\frac{u_c^h\left(c_{t+1}^h, \ell_{t+1}^h\right)}{u_c^h\left(c_t^h, \ell_t^h\right) o_t^h}\left[\pi_{t+1}^h + \theta_{t+1}^h f_k^h\left(k_{t+1}^h, \ell_{t+1}^h\right)\right] k_{t+1}^h\right\} \approx \widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right), \quad (141)$$

where new variables o_t^h and π_t^h are introduced for compactness,

$$o_t^h \equiv 1 + \phi\left(\frac{k_{t+1}^h}{k_t^h} - 1\right), \quad (142)$$

$$\pi_t^h \equiv +\frac{\phi}{2}\left(\left(\frac{k_{t+1}^h}{k_t^h}\right)^2 - 1\right). \quad (143)$$

We parameterize the right side of (141) with a flexible functional form. Our goal is to find parameters vectors $\left\{b^h\right\}^{h=1,\ldots,N}$ such that $\widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right)$ solves (141) for $h = 1, \ldots, N$. This representation of the intratemporal choice follows Example 11 in Section 6 and allows us to implement FPI without the need of a numerical solver. All three global Euler equation methods described in this section use parameterization (141) for characterizing the intertemporal choice.

Concerning the intratemporal choice, we do not use state contingent functions but solve for quantities $\{c_t, \ell_t\}$ that satisfy (136), (139), and (140). In principle, this can be done with any numerical method that is suitable for solving systems of nonlinear equations, for example, with a Newton's method. However, we advocate the use of derivative-free solvers; see Section 6 for a discussion. In Section 11.6, we show how to implement one such method, iteration-on-allocation, for the studied multicountry model.

11.3.3 Smolyak Method with Iteration-on-Allocation and FPI

We consider a version of the Smolyak algorithm, SMOL-JMMV, that builds on Judd et al. (2013). Also, Valero et al. (2013) test this algorithm in the context of a similar multicountry model. This method differs from the SMOL-MKK of Malin et al. (2011) in three respects. SMOL-JMMV solves for the intratemporal choice in terms of quantities (instead of state-contingent functions); this increases the accuracy of solutions. SMOL-JMMV solves for equilibrium decision rules using FPI (instead of time iteration); this decreases computational expense. Finally, SMOL-JMMV constructs the interpolation formula in a way that avoids costly repetitions of grid points and polynomial terms and computes the polynomial coefficients from a linear system of equations (instead of the conventional interpolation formula with repetitions); see Section 4 for a discussion.

Steps of SMOL-JMMV

- *Initialization:*
  - **(a)** Choose approximation level μ.
  - **(b)** Parameterize $\left(k^h\right)' = K^h(\boldsymbol{k}, \boldsymbol{\theta})$ with Smolyak polynomials $\widehat{K}^h\left(\boldsymbol{k},\boldsymbol{\theta}; b^h\right)$ constructed using Chebyshev unidimensional basis functions.
  - **(c)** Construct a Smolyak grid $\mathcal{H}^{2N,\mu} = \left\{\left(\boldsymbol{x}_m, \boldsymbol{y}_m\right)\right\}_{m=1,\ldots,M}$ on the hypercube $[-1,1]^{2N}$ using the Smolyak interpolation method described in Section 4.1, where $\boldsymbol{x}_m \equiv \left(x_m^1, \ldots, x_m^N\right)$ and $\boldsymbol{y}_m \equiv \left(y_m^1, \ldots, y_m^N\right)$.
  - **(d)** Compute the Smolyak basis functions $\mathcal{P}^{2N,\mu}$ in each grid point m. The resulting $M \times M$ matrix is $\mathcal{B}$.
  - **(e)** Choose the relevant ranges of values for $(\boldsymbol{k}, \boldsymbol{\theta})$ on which a solution is computed. The resulting hypercube is $\left[\underline{k}^1, \overline{k}^1\right] \times \cdots \times \left[\underline{\theta}^N, \overline{\theta}^N\right]$.
  - **(f)** Construct a mapping between points $(\boldsymbol{k}_m, \boldsymbol{\theta}_m)$ in the original hypercube $\left[\underline{k}^1, \overline{k}^1\right] \times \cdots \times \left[\underline{\theta}^N, \overline{\theta}^N\right]$ and points $\left(\boldsymbol{x}_m, \boldsymbol{y}_m\right)$ in the normalized hypercube $[-1,1]^{2N}$ using a linear change of variables:

$$x_m^h = 2\frac{k_m^h - \underline{k}^h}{\overline{k}^h - \underline{k}^h} - 1 \quad \text{and} \quad y_m^h = 2\frac{\theta_m^h - \underline{\theta}^h}{\overline{\theta}^h - \underline{\theta}^h} - 1, \tag{144}$$

where $\boldsymbol{k}_m \equiv \left(k_m^1, \ldots, k_m^N\right)$ and $\boldsymbol{\theta}_m \equiv \left(y_m^1, \ldots, y_m^N\right)$.

(g) Choose integration nodes, $\boldsymbol{\epsilon}_j = \left(\epsilon_j^1, \ldots, \epsilon_j^N\right)$, and weights, $\omega_j, j = 1, \ldots, J$.

(h) Construct next-period productivity, $\boldsymbol{\theta}'_{m,j} = \left(\left(\theta^h_{m,j}\right)', \ldots, \left(\theta^h_{m,j}\right)'\right)$ with $\left(\theta^h_{m,j}\right)' = \left(\theta^h_m\right)^\rho \exp\left(\epsilon^h_j\right)$ for all j and m.

(i) Make an initial guess on the coefficient vectors $\left(b^1\right)^{(1)}, \ldots, \left(b^N\right)^{(1)}$.

- *Iterative cycle. At iteration i, given $\left(b^1\right)^{(i)}, \ldots, \left(b^N\right)^{(i)}$, perform the following steps.*
- Step 1. *Computation of the capital choice.*
  Compute $\left(k^h_m\right)' = \mathcal{B}_m \left(b^h\right)^{(i)}$, where $\mathcal{B}_m$ is the mth row of $\mathcal{B}$ for $m = 1, \ldots, M$.
- Step 2. *Computation of the intratemporal choice.*
  Compute $\{\boldsymbol{c}_m, \boldsymbol{\ell}_m\}$ satisfying (136), (139), and (140) given $\left\{\boldsymbol{k}_m, \boldsymbol{\theta}_m, \boldsymbol{k}'_m\right\}$ for $m = 1, \ldots, M$.
- Step 3. *Approximation of conditional expectation.*
  For $m = 1, \ldots, M$,

  (a) compute:
  - $\left(\boldsymbol{x}'_{m,j}, \boldsymbol{y}'_{m,j}\right)$ that correspond to $\left(\boldsymbol{k}'_m, \boldsymbol{\theta}'_{m,j}\right)$ using the inverse of the transformation (144);
  - the Smolyak basis functions $\mathcal{P}^{2N,\mu}$ in each point $\left(\boldsymbol{x}'_m, \boldsymbol{y}'_{m,j}\right)$; the resulting $M \times M \times J$ matrix is $\mathcal{B}'_{m,j}$;
  - $\left(k^h_{m,j}\right)'' = \mathcal{B}'_{m,j} \left(b^h\right)^{(i)}$**,** where $\mathcal{B}'_{m,j}$ is a vector of basis functions evaluated in $\left(\boldsymbol{k}'_m, \boldsymbol{\theta}'_{m,j}\right)$ using the transformation (144) for all j;
  - $\left\{\boldsymbol{c}'_{m,j}, \boldsymbol{\ell}'_{m,j}\right\}$ satisfying (136), (139), and (140) given $\left\{\boldsymbol{k}'_m, \boldsymbol{\theta}'_{m,j}, \boldsymbol{k}''_{m,j}\right\}$ for $m = 1, \ldots, M$;

  (b) evaluate conditional expectation:

$$e^h_m \equiv \beta \sum_{j=1}^{J} \left\{ \omega_j \left(\frac{u^h_c\left(\left(c^h_{m,j}\right)', \left(\ell^h_{m,j}\right)'\right)}{u^h_c\left(c^h_m, \ell^h_m\right) o^h_m} \left[\left(\pi^h_{m,j}\right)' + \left(\theta^h_{m,j}\right)' f^h_k\left(\left(k^h_m\right)', \left(\ell^h_{m,j}\right)'\right)\right] \left(k^h_m\right)' \right) \right\},$$

  where o^h_m and $\left(\pi^h_{m,j}\right)'$ are given by (142) and (143), respectively.
- Step 4. *Computation of the intertemporal choice.*
  Find $\left\{\widehat{b}^h\right\}^{h=1,\ldots,N}$ that solves $e^h_m = \mathcal{B}_m \widehat{b}^h$, i.e., $\widehat{b}^h = \mathcal{B}_m^{-1} e^h_m$.

- Step 5. *Updating of the coefficient vectors.*
  For each $h = 1, \ldots, N$, compute the coefficient vector for the subsequent iteration $i + 1$ using fixed-point iteration,

$$\left(b^h\right)^{(i+1)} = (1 - \xi) \left(b^h\right)^{(i)} + \xi \widehat{b}^h, \tag{145}$$

  where $\xi \in (0, 1)$ is a damping parameter.

Iterate on Steps 1–5 until convergence of the solution,

$$\frac{1}{MN\xi} \sum_{m=1}^{M} \sum_{h=1}^{N} \left| \frac{\left(\left(k_m^h\right)'\right)^{(i+1)} - \left(\left(k_m^h\right)'\right)^{(i)}}{\left(\left(k_m^h\right)'\right)^{(i)}} \right| < 10^{-\vartheta}, \tag{146}$$

where $\left(\left(k_m^h\right)'\right)^{(i+1)}$ and $\left(\left(k_m^h\right)'\right)^{(i)}$ are the hth country's capital choices on the grid obtained on iterations $i + 1$ and i, respectively, and $\vartheta > 0$.

Computational Details

To start iterations, we use an arbitrary initial guess on the capital decision function, $k_{t+1}^h = 0.9k_t^h + 0.1\theta_t^h$, for all $h = 1, \ldots, N$ (this guess matches the steady-state level of capital). We use a Smolyak polynomial function with an approximation level $\mu = 2$ (such a function has four times more coefficients than the ordinary polynomial function of degree two). We transform $(\boldsymbol{k}_m, \boldsymbol{\theta}_m)$ into $(\boldsymbol{x}_m, \boldsymbol{y}_m)$ using a linear change of variable (144) because unidimensional Chebyshev basis functions are defined in the interval $[-1, 1]$ and, thus, the Smolyak grid $(\boldsymbol{x}_m, \boldsymbol{y}_m)$ is constructed inside a hypercube $[-1, 1]^{2N}$, whereas the original capital and productivity levels are not necessarily in this range. To approximate integrals, we use a monomial integration rule $M1$ with $2N$ nodes. We set the damping parameter in FPI (145) at $\xi = 0.05$, and we set the tolerance parameter at $\vartheta = 7$ in convergence criterion (147).

11.3.4 Generalized Stochastic Simulation Algorithm

In the JEDC comparison analysis, Maliar et al. (2011) implement a version of the GSSA algorithm that computes linear solutions using a one-node Monte Carlo integration method (we refer to this method as SSA). In this section, we implement other versions of GSSA, those that use Monte Carlo, Gauss-Hermite quadrature and monomial rules. Also, we compute polynomial approximations up to degree 3. In particular, these choices allow us to assess the role of approximation and integration methods in the accuracy and speed of GSSA.

Steps of GSSA

- *Initialization:*
  (a) Parameterize $k_{t+1}^h = K^h(\boldsymbol{k}_t, \boldsymbol{\theta}_t)$ with a flexible functional form $\widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right)$.

(b) Fix simulations length T and initial condition $(\boldsymbol{k}_0, \boldsymbol{\theta}_0)$. Draw a sequence of productivity shocks $\{\boldsymbol{\epsilon}_t\}_{t=1,\ldots,T}$. Compute and fix the sequence of productivity levels $\{\boldsymbol{\theta}_{t+1}\}_{t=0,\ldots,T-1}$ using equation (137).

(c) Choose integration nodes, $\boldsymbol{\epsilon}_j = \left(\epsilon_j^1, \ldots, \epsilon_j^N\right)$, and weights, $\omega_j, j = 1, \ldots, J$.

(d) Construct next-period productivity, $\boldsymbol{\theta}_{t+1,j} = \left(\theta_{t+1,j}^h, \ldots, \theta_{t+1,j}^h\right)$ with $\theta_{t+1,j}^h = \left(\theta_t^h\right)^\rho \exp\left(\epsilon_j^h\right)$ for all j.

(e) Make an initial guess on the coefficient vectors $\left(b^1\right)^{(1)}, \ldots, \left(b^N\right)^{(1)}$.

Iterative cycle. At iteration i, given $\left(b^1\right)^{(i)}, \ldots, \left(b^N\right)^{(i)}$, perform the following steps.

- Step 1. *Simulation of the solution.*
  Use the assumed capital decision functions $k_{t+1}^h = \widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; (b^h)^{(i)}\right), h = 1, \ldots, N$, to recursively calculate a sequence of capital stocks $\{\boldsymbol{k}_{t+1}\}_{t=0,\ldots,T}$ corresponding to a given sequence of productivity levels $\{\boldsymbol{\theta}_t\}_{t=0,\ldots,T}$.
- Step 2. *Computation of the intratemporal choice.*
  Compute $\{\boldsymbol{c}_t, \boldsymbol{\ell}_t\}$ satisfying (136), (139), and (140) given $\{\boldsymbol{k}_t, \boldsymbol{\theta}_t, \boldsymbol{k}_{t+1}\}$ for $t = 0, \ldots, T$.
- Step 3. *Approximation of conditional expectation.*
  For $t = 0, \ldots, T-1$,

  (a) compute:
  - $k_{t+2,j}^h = \widehat{K}^h\left(\boldsymbol{k}_{t+1}, \boldsymbol{\theta}_{t+1,j}; (b^h)^{(i)}\right)$ for all j;
  - $\left\{\boldsymbol{c}_{t+1,j}, \boldsymbol{\ell}_{t+1,j}\right\}$ satisfying (136), (139), and (140) for given $\left\{\boldsymbol{k}_{t+1}, \boldsymbol{\theta}_{t+1,j}, \boldsymbol{k}_{t+2,j}\right\}$;

  (b) evaluate conditional expectation:

$$e_t^h \equiv \beta \sum_{j=1}^{J} \left\{ \omega_{t,j} \cdot \left(\frac{u_c^h\left(c_{t+1,j}^h, \ell_{t+1,j}^h\right)}{u_c^h\left(c_t^h, \ell_t^h\right) o_t^h} \left[\pi_{t+1,j}^h + \theta_{t+1,j}^h f_k^h\left(k_{t+1}^h, \ell_{t+1,j}^h\right)\right] k_{t+1}^h \right) \right\},$$

  where o_t^h and $\pi_{t+1,j}^h$ are given by (142) and (143).
- Step 4. *Computation of the intertemporal choice.*
  Find $\left\{\widehat{b}^h\right\}^{h=1,\ldots,N}$ such that

$$\widehat{b}^h \equiv \arg\min_{b^h} \sum_{t=1}^{T} \left\| e_t^h - \widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right) \right\|.$$

- *Updating of the coefficients vectors.*
  For each $h = 1, \ldots, N$, compute the coefficient vector for the subsequent iteration $i + 1$ using FPI (145).

Iterate on Steps 1–5 until convergence,

$$\frac{1}{TN\xi}\sum_{t=1}^{T}\sum_{h=1}^{N}\left|\frac{\left(k_{t+1}^{h}\right)^{(i+1)}-\left(k_{t+1}^{h}\right)^{(i)}}{\left(k_{t+1}^{h}\right)^{(i)}}\right|<10^{-\vartheta}, \tag{147}$$

where $\left(k_{t+1}^{h}\right)^{(i+1)}$ and $\left(k_{t+1}^{h}\right)^{(i)}$ are the hth country's capital stocks obtained on iterations $i+1$ and i, respectively, and $\vartheta > 0$.

Computational Details

To start the iterative process, we use the same (arbitrary) initial guess as in the Smolyak method: $k_{t+1}^{h} = 0.9k_t^h + 0.1\theta_t^h$ for all $h = 1, \ldots, N$. The simulation length for finding solutions is $T = 2,000$. Initial capital and productivity level are set at their steady-state values: $k_0^h = 1$ and $\theta_0^h = 1$ for all $h = 1, \ldots, N$. We compute complete ordinary polynomial approximations of degrees 2/3. In Step 4, we approximate integrals using three different methods: a one-node Monte Carlo integration method, a one-node Gauss-Hermite quadrature rule (i.e., we assume that the future has just one possible state $\epsilon_h = 0$ for all h), and a monomial rule $M1$ with $2N$ nodes. In Step 5, we use a least-squares truncated QR factorization method, which is robust to ill-conditioning. We set the damping parameter in FPI (145) at $\xi = 0.05$ and we set the tolerance parameter at $\vartheta = 7$ in convergence criterion (147).

11.3.5 ε-Distinguishable Set Algorithm

We consider a version of the EDS algorithm studied in Judd et al. (2012). The number of grid points in the EDS grid is smaller than the number of simulated points considered by GSSA; namely, we set the number of grid points, which is two times larger than the number of polynomial coefficients to estimate. This allows us to substantially reduce the cost relative to GSSA.

Steps of EDS

- *Initialization:*
  - **(a)** Parameterize $k_{t+1}^{h} = K^h(\boldsymbol{k}_t, \boldsymbol{\theta}_t)$ with a flexible functional form $\widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right)$.
  - **(b)** Fix simulations length T and initial condition $(\boldsymbol{k}_0, \boldsymbol{\theta}_0)$. Draw $\{\boldsymbol{\epsilon}_t\}_{t=1,\ldots,T}$. Compute and fix $\{\boldsymbol{\theta}_{t+1}\}_{t=0,\ldots,T-1}$ using equation (137).
  - **(c)** Choose integration nodes, $\boldsymbol{\epsilon}_j = \left(\epsilon_j^1, \ldots, \epsilon_j^N\right)$, and weights, ω_j , $j = 1, \ldots, J$.
  - **(d)** Make an initial guess on the coefficient vectors $\left(b^1\right)^{(1)}, \ldots, \left(b^N\right)^{(1)}$.
- Step 1. *Construction of a grid and the corresponding next-period productivity.*
  - **(a)** Use the assumed capital decision functions $k_{t+1}^{h} = \widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right)$, $h = 1, \ldots, N$, to recursively calculate a sequence of capital stocks $\{\boldsymbol{k}_{t+1}\}_{t=0,\ldots,T}$ corresponding to a given sequence of productivity levels $\{\boldsymbol{\theta}_t\}_{t=0,\ldots,T}$.

(b) Construct a grid $\Gamma = \{\boldsymbol{k}_m, \boldsymbol{\theta}_m\}_{m=1,\ldots,M}$ using the EDS algorithm described in Section 4.3.

(c) Construct next-period productivity, $\boldsymbol{\theta}'_{m,j} \equiv \left(\left(\theta^h_{m,j}\right)', \ldots, \left(\theta^h_{m,j}\right)'\right)$ with $\left(\theta^h_{m,j}\right)' = \left(\theta^h_m\right)^{\rho} \exp\left(\epsilon^h_j\right)$ for all j and m.

Iterative cycle. At iteration i, given $\left(b^1\right)^{(i)}, \ldots, \left(b^N\right)^{(i)}$*, perform the following steps.*

- Step 2. *Computation of the capital choice.*
  Compute $\boldsymbol{k}'_m = \widehat{K}^h\left(\boldsymbol{k}_m, \boldsymbol{\theta}_m; \left(b^h\right)^{(i)}\right)$ for $m = 1, \ldots, M$.
- Step 3. *Computation of the intratemporal choice.*
  Compute $\{\boldsymbol{c}_m, \boldsymbol{\ell}_m\}$ satisfying (136), (139), and (140) given $\left\{\boldsymbol{k}_m, \boldsymbol{\theta}_m, \boldsymbol{k}'_m\right\}$ for $m = 1, \ldots, M$.
- Step 4. *Approximation of conditional expectation.*
  For $m = 1, \ldots, M$,

  (a) compute:
  – $\left(k^h_{m,j}\right)'' = \widehat{K}^h\left(\boldsymbol{k}'_m, \boldsymbol{\theta}'_{m,j}; \left(b^h\right)^{(i)}\right)$ for all j;
  – $\left\{\boldsymbol{c}'_{m,j}, \boldsymbol{\ell}'_{m,j}\right\}$ satisfying (136), (139), and (140) given $\left\{\boldsymbol{k}'_m, \boldsymbol{\theta}'_{m,j}, \boldsymbol{k}''_{m,j}\right\}$ for $m = 1, \ldots, M$;

  (b) evaluate conditional expectation:

$$e^h_m \equiv \beta \sum_{j=1}^{J} \left\{ \omega_j \left(\frac{u^h_c\left(\left(c^h_{m,j}\right)', \left(\ell^h_{m,j}\right)'\right)}{u^h_c\left(c^h_m, \ell^h_m\right) o^h_m} \right.\right.$$

$$\left.\left. \left[\left(\pi^h_{m,j}\right)' + \left(\theta^h_{m,j}\right)' f^h_k\left(\left(k^h_m\right)', \left(\ell^h_{m,j}\right)'\right)\right] \left(k^h_m\right)' \right) \right\},$$

  where o^h_m and $\left(\pi^h_{m,j}\right)'$ are given by (142) and (143), respectively.

- Step 5. *Computation of the intertemporal choice.*
  Find $\left\{\widehat{b}^h\right\}^{h=1,\ldots,N}$ such that

$$\widehat{b}^h \equiv \arg\min_{b^h} \sum_{m=1}^{M} \left\| e^h_m - \widehat{K}^h\left(\boldsymbol{k}_m, \boldsymbol{\theta}_m; b^h\right) \right\|.$$

- Step 6. *Updating of the coefficient vectors.*
  For each $h = 1, \ldots, N$, compute the coefficient vector for the subsequent iteration $i + 1$ using FPI (145).

Iterate on Steps 2–6 until convergence of the solution,

$$\frac{1}{MN\xi}\sum_{m=1}^{M}\sum_{h=1}^{N}\left|\frac{\left(\left(k_m^h\right)'\right)^{(i+1)}-\left(\left(k_m^h\right)'\right)^{(i)}}{\left(\left(k_m^h\right)'\right)^{(i)}}\right| < 10^{-\vartheta},$$

where $\left(\left(k_m^h\right)'\right)^{(i+1)}$ and $\left(\left(k_m^h\right)'\right)^{(i)}$ are the hth country's capital choices on the grid obtained on iterations $i+1$ and i, respectively, and $\vartheta > 0$. Iterate on Steps 1–6 until convergence of the grid.

Computational Details

As in the case of the previous two algorithms, we start simulation from the steady state and assume $k_{t+1}^h = 0.9k_t^h + 0.1\theta_t^h$ as an initial guess for all $h = 1, \ldots, N$. We use complete ordinary polynomial functions of degrees two and three. The simulation length for constructing the EDS grid is $T = 10{,}000$, and the number of grid points is twice as large as the number of polynomial coefficients to estimate; for example, for the second-degree polynomial case, we have $M = 2\left(1 + 4d + \frac{d(d+1)}{2}\right)$ grid points. In Step 4, to approximate integrals, we use a monomial integration rule $M1$ with $2N$ nodes as in (63). In Step 5, we estimate the regression equation by using a least-squares truncated QR factorization method. The damping parameter is $\xi = 0.05$ and the tolerance parameter is $\vartheta = 7$.

11.4 Dynamic Programming Methods

We next implement dynamic programming methods. Specifically, we consider two versions of ECM, one that approximates the value function and the other that approximates derivatives of the value function. As introduced in Maliar and Maliar (2013), the ECM methods rely on a product grid for in two-dimensional examples. In a following-up paper, Maliar and Maliar (2012a,b) implement simulation-based versions of the ECM method that are tractable in high-dimensional applications. These versions of ECM are applied to solve multiagent models studied in the JEDC comparison analysis of Kollmann et al. (2011b). The design of ECM follows the design of GSSA, namely, we combine simulation-based grids with monomial integration rules, numerically stable regression methods, and an FPI method for finding the polynomial coefficients. The results of Maliar and Maliar (2012a,b) are reproduced below. We summarize the implementation of the ECM algorithms in the first two columns of Table 20.

11.4.1 Bellman Equation, FOCs, and Envelope Condition

Let us write problem (135)–(137) in the DP form

$$V(\boldsymbol{k},\boldsymbol{\theta}) = \max_{\left\{\left(k^h\right)', c^h, \ell^h\right\}^{h=1,\ldots,N}} \left\{\sum_{h=1}^{N} \tau^h u^h\left(c^h, \ell^h\right) + \beta E\left[V\left(\boldsymbol{k}', \boldsymbol{\theta}'\right)\right]\right\} \qquad (148)$$

Table 20 Implementation of our Bellman equation algorithms and hybrid algorithm.[a]

	ECM-VF	**ECM-DVF**	**HYB**
Articles	Maliar and Maliar (2012)	Maliar and Maliar (2012)	Maliar et al. (2012)
Program. language	MATLAB	MATLAB	C++, Dynare
Functions parameterized	K^h for all h, and V	K^h for all h, V_{k^h} for all h	K^h for all h
Functional form	2^d and 3^d degree compl. ordin. polyn.	2^d and 3^d degree compl. ordin. polyn.	1^{st}, 2^d, and 3^d degree compl. ordin. polyn.
# Basis functions	$1+d+\frac{d(d+1)}{2}$ and n_3	$1+d+\frac{d(d+1)}{2}$ and n_3	$1+d$, $1+d+\frac{d(d+1)}{2}$ and n_3
Number of grid points	2000	2000	–
Intratemporal choice	Iteration-on-allocation	Iteration-on-allocation	Iteration-on-allocation
Integration	Monomial	Monomial	–
Solving for polyn. coeff.	FPI	FPI	–

[a] *Notes:* $d = 2N$ is the model's dimensionality; n_3 is the number of basis functions in complete ordinary polynomial of degree 3, $n_3 = 1+d+d(d+1)/2+d^2+d(d-1)(d-2)/6$.

$$\text{s.t.} \sum_{h=1}^{N} c^h = \sum_{h=1}^{N} \left[\theta^h f^h \left(k^h, \ell^h\right) - \frac{\phi}{2} k^h \left(\frac{\left(k^h\right)'}{k^h} - 1 \right)^2 + k^h - \left(k^h\right)' \right], \tag{149}$$

$$\left(\theta^h\right)' = \rho \ln \theta^h + \sigma \left(\epsilon^h\right)', \tag{150}$$

where V is the optimal value function, $\left(\left(\epsilon^1\right)', \ldots, \left(\epsilon^N\right)'\right)^{\top} \sim \mathcal{N}\left(0_N, \Sigma\right)$; $\boldsymbol{k} \equiv \left(k^1, \ldots, k^N\right)$ and $\boldsymbol{\theta} \equiv \left(\theta^1, \ldots, \theta^N\right)$.

Again, we assume that the solution to DP problem (148)–(150) is interior and, hence, satisfies the FOCs and envelope condition, which are, respectively, given by

$$\beta E\left[V_{k^h}\left(\boldsymbol{k}', \boldsymbol{\theta}'\right)\right] = \lambda \left[1 + \phi \cdot \left(\frac{k_{t+1}^h}{k_t^h} - 1 \right) \right], \tag{151}$$

$$\tau^h u_c^h \left(c^h, \ell^h\right) = \lambda, \tag{152}$$

$$u_\ell^h \left(c^h, \ell^h\right) \tau^h = -\lambda \theta^h f_\ell^h \left(k^h, \ell^h\right), \tag{153}$$

$$V_{k^h}\left(\boldsymbol{k}, \boldsymbol{\theta}\right) = \lambda \left[1 + \theta^h f_k^h \left(k^h, \ell^h\right) + \frac{\phi}{2} \left(\left(\frac{\left(k^h\right)'}{k^h} \right)^2 - 1 \right) \right], \tag{154}$$

where λ is the Lagrange multiplier associated with aggregate resource constraint (149).

A solution to dynamic programming problem (148)–(150) is an optimal value function V that satisfies (148)–(154). For our planner's problem, the Bellman equation implies the Euler equation: by updating envelope condition (154) for the next period and by substituting the resulting condition into (151), we obtain Euler equation (138).

11.4.2 Separating the Intertemporal and Intratemporal Choices

The value function iteration aims at approximating a solution to (148)–(154). As in the case of the Euler equation methods, we separate the optimality conditions into the intertemporal and intratemporal choice conditions; see Section 7.

Regarding the intertemporal choice, we combine (152) and (154) to obtain

$$u_c^h \left(c_t^h, \ell_t^h\right) \left[\pi_t^h + \theta_t^h f_k^h \left(k_t^h, \ell_t^h\right)\right] = V_{k^h}\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t\right), \tag{155}$$

where o_t^h and π_t^h are given by (142) and (143), respectively. We next obtain a condition that relates today's and tomorrow's derivatives of the value function by combining FOC (151) and envelope condition (154) to eliminate λ_t,

$$V_{k^h}\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t\right) = \frac{\beta \left[\pi_t^h + \theta_t^h f_k^h \left(k_t^h, \ell_t^h\right)\right]}{o_t^h} E_t \left[V_{k^h}\left(\boldsymbol{k}_{t+1}, \boldsymbol{\theta}_{t+1}\right)\right], \tag{156}$$

where o_t^h and π_t^h are given by (142) and (143), respectively. This condition is parallel to (95) for the one-agent model. We next premultiply (156) from both sides with k_{t+1}^h, rearrange the terms, and parameterize it with a flexible functional form to obtain

$$k_{t+1}^h = \frac{\beta E_t\left[V_{k^h}\left(\boldsymbol{k}_{t+1}, \boldsymbol{\theta}_{t+1}\right)\right]}{o_t^h V_{k^h}\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t\right)}\left[\pi_t^h + \theta_t^h f_k^h\left(k_t^h, \ell_t^h\right)\right] k_{t+1}^h \approx \widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right). \tag{157}$$

The above condition is similar to Euler equation (141) and is written in a way that is convenient for an iterative process.

As far as the intratemporal choice is concerned, conditions (149), (152), and (153) under DP methods are identical to (136), (139), and (140) under the Euler equation methods. As a result, we can use an identical solver for finding the intratemporal choice (we use the iteration-on-allocation procedure described in Section 11.6). We shall recall an important advantage of the DP approaches over the Euler equation approaches. In the former case, we only need to know the intratemporal choice in the current state, while in the latter case, we must also find such a choice in J possible future states (integration nodes). To be more specific, GSSA, EDS, and SMOL-JMMV have to solve for $\left\{\boldsymbol{c}'_{m,j}, \boldsymbol{\ell}'_{m,j}\right\}$ satisfying (136), (139), and (140), which is expensive. We do not need to solve for those quantities under the DP approach described in this section.

11.4.3 Envelope Condition Method Iterating on Value Function

ECM-VF is a variant of the ECM method that iterates on value function.

Steps of ECM-VF

- *Initialization:*
  - **(a)** Parameterize $K^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t\right)$ and $V\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t\right)$ with flexible functional forms $\widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right)$ and $\widehat{V}\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; \varpi\right)$, respectively.
  - **(b)** Fix simulation length T and initial condition $\left(\boldsymbol{k}_0, \boldsymbol{\theta}_0\right)$. Draw and fix for all simulations a sequence of productivity levels $\{\boldsymbol{\theta}_t\}_{t=1,\ldots,T}$ using (137).
  - **(c)** Choose integration nodes, $\boldsymbol{\epsilon}_j = \left(\epsilon_j^1, \ldots, \epsilon_j^N\right)$, and weights, $\omega_j, j = 1, \ldots, J$.
  - **(d)** Construct next-period productivity, $\boldsymbol{\theta}_{t+1,j} = \left(\theta_{t+1,j}^h, \ldots, \theta_{t+1,j}^h\right)$ with $\theta_{t+1,j}^h = \left(\theta_t^h\right)^\rho \exp\left(\epsilon_j^h\right)$ for all j.
  - **(e)** Make an initial guess on the coefficient vectors $\left(b^1\right)^{(1)}, \ldots, \left(b^h\right)^{(1)}$.

  Iterative cycle. At iteration i, given $\left(b^1\right)^{(i)}, \ldots, \left(b^N\right)^{(i)}$, perform the following steps.
- Step 1. *Simulation of the solution.*
  Use the assumed capital decision functions $k_{t+1}^h = \widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; (b^h)^{(i)}\right), h = 1, \ldots, N$, to recursively calculate a sequence of capital stocks $\{\boldsymbol{k}_{t+1}\}_{t=0,\ldots,T}$ corresponding to a given sequence of productivity levels $\{\boldsymbol{\theta}_t\}_{t=0,\ldots,T}$.
- Step 2. *Computation of the intratemporal choice.*
  Compute $\{\boldsymbol{c}_t, \boldsymbol{\ell}_t\}$ satisfying (136), (139), and (140), given $\{\boldsymbol{k}_t, \boldsymbol{\theta}_t, \boldsymbol{k}_{t+1}\}$ for $t = 0, \ldots, T$.
- Step 3. *Approximation of conditional expectation.*
  For $t = 0, \ldots, T-1$,

(a) compute $k^h_{t+1} = \widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; (b^h)^{(i)}\right)$ and π^h_t;
(b) find $\widehat{\varpi}$ satisfying $V(\boldsymbol{k}_t, \boldsymbol{\theta}_t; \widehat{\varpi}) = \sum_{h=1}^{N} \tau^h u^h\left(c^h_t, \ell^h_t\right) + \beta \sum_{j=1}^{J} \omega_j V\left(\boldsymbol{k}_{t+1}, \boldsymbol{\theta}_{t+1,j}; \widehat{\varpi}\right)$;
(c) use $V(\cdot; \widehat{\varpi})$ to find $V_{k^h}(\boldsymbol{k}_t, \boldsymbol{\theta}_t; \widehat{\varpi})$ and to infer $V_{k^h}\left(\boldsymbol{k}_{t+1}, \boldsymbol{\theta}_{t+1,j}; \widehat{\varpi}\right)$ for $j = 1, \ldots, J$;
(d) evaluate conditional expectation in (157) and compute

$$e^h_t \equiv \frac{\beta \sum_{j=1}^{J} \omega_j \cdot V_{k^h}\left(\boldsymbol{k}_{t+1}, \boldsymbol{\theta}_{t+1,j}; \widehat{\varpi}\right)}{o^h_t V_{k^h}(\boldsymbol{k}_t, \boldsymbol{\theta}_t; \widehat{\varpi})}\left[\pi^h_t + \theta^h_t f^h_k\left(k^h_t, \ell^h_t\right)\right] k^h_{t+1},$$

where o^h_t and π^h_t are given by (142) and (143), respectively.

- Step 4. *Computation of b^h that fits the values $\widehat{k}^h_{t+1}$ on the grid.*

Find $\left\{\widehat{b}^h\right\}^{h=1,\ldots,N}$ such that

$$\widehat{b}^h \equiv \arg\min_{b^h} \sum_{t=1}^{T} \left\| e^h_t - \widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right)\right\|.$$

- Step 5. *Updating of the coefficient vectors.* For each country $h = 1, \ldots, N$, compute the coefficient vector for the subsequent iteration $i + 1$ using FPI (145).
  Iterate on Steps 1–5 until the convergence criterion (147) is satisfied.

Computational Details

To start the iterative process, we use an initial guess: $k^h_{t+1} = 0.9k^h_t + 0.1\theta^h_t$ for all $h = 1, \ldots, N$, and we assume $k^h_0 = 1$ and $\theta^h_0 = 1$ for all $h = 1, \ldots, N$. The simulation length is $T = 2,000$. We use complete ordinary polynomials of degrees two and three. To approximate integrals in Step 3, we use a monomial integration rule $M1$ with $2N$ nodes as in (63). In Step 4, we use a least-squares truncated QR factorization method. The damping parameter is $\xi = 0.05$ and the tolerance parameter is $\vartheta = 7$.

11.4.4 Envelope Condition Method Solving for Derivatives of Value Function

ECM-DVF is a variant of the ECM method that iterates on derivatives of value function. We use (155) to approximate the derivative of value function, $V_{k^h}(\cdot) \approx \widehat{V}_{k^h}\left(\cdot; \varpi^h\right)$.

The Steps of ECM-DVF

The steps that are identical to those in ECM-VF are omitted.

- *Initialization:*

(a) Parameterize $K^h(\boldsymbol{k}_t, \boldsymbol{\theta}_t)$ and $V_{k^h}(\boldsymbol{k}_t, \boldsymbol{\theta}_t)$ with flexible functional forms $\widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right)$ and $\widehat{V}_{k^h}\left(\boldsymbol{k}_t, \boldsymbol{\theta}; \varpi^h\right)$, respectively (in both cases, we use ordinary polynomials).

…

Iterative cycle. At iteration i, given $\left(b^1\right)^{(i)}, \ldots, \left(b^N\right)^{(i)}$, perform the following steps.

- Step 1.

  …
- **(b)** find $d_t^h \equiv u_c^h\left(c_t^h, \ell_t^h\right)\left[\pi_t^h + \theta_t^h f_k^h\left(k_t^h, \ell_t^h\right)\right]$ and find $\widehat{\varpi}^h \equiv \arg\min_{\varpi^h} \left\| d_t^h - V_{k^h}\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; \varpi^h\right)\right\|$, $h = 1, \ldots, N$;

  (c) use $V_{k^h}\left(\cdot; \widehat{\varpi}^h\right)$ to infer $V_{k^h}\left(\boldsymbol{k}_{t+1}, \boldsymbol{\theta}_{t+1,j}; \widehat{\varpi}^h\right)$ for $j = 1, \ldots, J$;

  …
- Step 3. *Approximation of conditional expectation.*

  For $t = 0, \ldots, T-1$,

  (a) compute $k_{t+1}^h = \widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; \left(b^h\right)^{(i)}\right)$ and π_t^h;

  (b) find derivative of value function in (155), $d_t^h \equiv u_c^h\left(c_t^h, \ell_t^h\right)\left[\pi_t^h + \theta_t^h f_k^h\left(k_t^h, \ell_t^h\right)\right]$;

  (c) find $\left\{\widehat{\varpi}^h\right\}^{h=1,\ldots,N}$ such that

$$\widehat{\varpi}^h \equiv \arg\min_{\varpi^h} \left\| d_t^h - \widehat{V}_{k^h}\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; \varpi^h\right)\right\|;$$

  (d) evaluate conditional expectation in (157) and compute

$$e_t^h \equiv \frac{\beta \sum_{j=1}^J \omega_j \cdot \widehat{V}_{k^h}\left(\boldsymbol{k}_{t+1}, \left(\theta_{t+1,j}^h\right)'; \varpi^h\right)}{o_t^h \widehat{V}_{k^h}\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; \varpi^h\right)} \left[\pi_t^h + \theta_t^h f_k^h\left(k_t^h, \ell_t^h\right)\right] k_{t+1}^h,$$

  where o_t^h and π_t^h are given by (142) and (143), respectively.
- …

Computational Details

The computational choice for ECM-DVF is the same as that for ECM-VF.

11.5 Hybrid of Local and Global Solutions

We consider a version of the hybrid perturbation-based method of Maliar et al. (2013) who implement this algorithm for model (135)–(137) studied here; we reproduce the results from that paper. The implementation of the hybrid perturbation-based method is summarized in the last column of Table 20.

As a first step, we compute a solution to model (135)–(137) using the standard perturbation method in levels; to this purpose, we use the Dynare software. We keep the capital policy functions $\widehat{K}^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right)$, $h = 1, \ldots, N$ that are delivered by Dynare and we discard all the other policy functions (i.e., consumption and labor).

When simulating the solutions and evaluating their accuracy, we proceed as under global Euler equation methods. Namely, we first construct a path for capital, $\{\boldsymbol{k}_{i+1}\}_{\tau=0,\ldots,\mathcal{T}}$, and subsequently fill in the corresponding intratemporal allocations $\{\boldsymbol{c}_i, \boldsymbol{\ell}_i\}_{i=1,\ldots,I}$ by solving (136), (139), and (140) for each given $\{\boldsymbol{k}_i, \boldsymbol{\theta}_i, \boldsymbol{k}_{i+1}\}$. In this step, we use the iteration-on-allocation solver as described in Section 11.6.

11.6 Solving for Consumption and Labor Using Iteration-on-Allocation

Maliar et al. (2011) emphasize the importance of solving accurately for intratemporal choice for the overall accuracy of solutions; see their Table 1 and see our Table 11 and a related discussion. Also, the comparison results of Kollmann et al. (2011b) lead to the same conclusion: the residuals in the intratemporal choice conditions are larger than the residuals in the intertemporal choice conditions by an order of magnitude and they drive the overall accuracy down for all the methods. An exception is two methods described in Maliar et al. (2011), which solve for all the intratemporal choice variables exactly (up to a given degree of accuracy). Below, we describe the iteration-on-allocation method which is used by the two methods studied in Maliar et al. (2011), as well as by all methods presented in Section 11.

Parameterization of the Model

We solve one of the multicountry models studied in the comparison analysis of Kollmann et al. (2011b), namely, Model II with an asymmetric specification.[21] The parameterization of this model is described in Juillard and Villemot (2011). The utility and production functions are given by

$$u^h\left(c_t^h,\ell_t^h\right)=\frac{\left(c_t^h\right)^{1-1/\gamma^h}}{1-1/\gamma^h}-B^h\frac{\left(\ell_t^h\right)^{1+1/\eta^h}}{1+1/\eta^h},\qquad \theta_t^h f^h\left(k_t^h,\ell_t^h\right)\equiv\theta_t^h A\left(k_t^h\right)^{\alpha}\left(\ell_t^h\right)^{1-\alpha}-\delta k^h, \tag{158}$$

where $\left\{\gamma^h,B^h,\eta^h\right\}$ are the utility-function parameters; α is the capital share in production; A is the normalizing constant in output. The country-specific utility-function parameters γ^h and η^h are uniformly distributed in the intervals $[0.25,1]$ and $[0.1,1]$ across countries $h=1,\ldots,N$, respectively. The values of common-for-all-countries parameters are $\alpha=0.36,\beta=0.99,\delta=0.025,\sigma=0.01,\rho=0.95,\phi=0.5$. The steady-state level of productivity is normalized to one, $\overline{\theta}^h=1$. We also normalize the steady-state levels of capital and labor to one, $\overline{k}^h=1,\overline{\ell}^h=1$, which implies $\overline{c}^h=A,\overline{\lambda}=1$ and leads to $A=\frac{1-\beta}{\alpha\beta},\tau^h=u^h(A,1)$, and $B^h=(1-\alpha)A^{1-1/\gamma^h}$. We chose this parameterization because it represents all challenges posed in the JEDC comparison analysis, namely, a large number of state variables, endogenous labor-leisure choice, heterogeneity in fundamentals, and the absence of closed-form expressions for next-period state and control variables in terms of the current state variables.[22]

Iteration-on-Allocation

We now show how to solve for the intratemporal choice under assumptions (158) using the iteration-on-allocation solver. Our objective is to solve for $\{c,\ell\}$ that satisfy (136),

[21] In total, there are four models, Models I–IV, studied in Kollmann et al. (2011b), and each of these models has symmetric and asymmetric specifications.

[22] Model I has a degenerate labor-leisure choice and Models III and IV are identical to Model II up to specific assumptions about preferences and technologies. Juillard and Villemot (2011) provide a description of all models studied in the comparison analysis of Kollmann et al. (2011b).

(139), and (140) for given $\{\boldsymbol{k}, \boldsymbol{\theta}, \boldsymbol{k}'\}$. Under parameterization (158), these conditions can be represented as

$$\ell^h = \left[\frac{\theta^h \left(k^h\right)^\alpha \tau^1 B^1}{\theta^1 \left(k^1\right)^\alpha \tau^h B^h}\right]^{\frac{\eta^h}{1+\alpha\eta^h}} \left(\ell^1\right)^{\frac{\eta^h\left(1+\alpha\eta^1\right)}{\eta^1\left(1+\alpha\eta^h\right)}} \quad h = 2, \ldots, N, \tag{159}$$

$$c^h = \left[\frac{(1-\alpha)\,\theta^h A \left(k^h\right)^\alpha \left(\ell^h\right)^{-\alpha}}{B^h \left(\ell^h\right)^{1/\eta^h}}\right]^{\gamma^h}, \tag{160}$$

$$\widehat{\ell^1} = \left[\frac{\sum_{h=1}^N \left[c^h + \left(k^h\right)' - k^h + \frac{\phi}{2} k^h \left(\frac{\left(k^h\right)'}{k^h} - 1\right)^2\right] - \sum_{h=2}^N \theta^h A \left(k^h\right)^\alpha \left(\ell^h\right)^{1-\alpha}}{\theta^1 A \left(k^1\right)^\alpha}\right]^{\frac{1}{1-\alpha}}, \tag{161}$$

where $\widehat{\ell^1}$ is a new value of labor of country 1. Condition (159) is obtained by finding a ratio of (139) for two agents, condition (160) follows by combining (139) and (140), and condition (161) follows from (136). For given $\{\boldsymbol{k}, \boldsymbol{\theta}, \boldsymbol{k}'\}$, equations (159)–(161) define a mapping $\widetilde{\ell}^1 = \Gamma\left(\ell^1\right)$. We iterate on labor of the first country, ℓ^1, as follows: assume some initial ℓ^1; compute $\ell^h, h = 2, \ldots, N$, from (159); find $c^h, h = 1, \ldots, N$, from (160); obtain $\widehat{\ell^1}$ from (161) and stop if $\ell^1 = \widehat{\ell^1}$ with a given degree of accuracy; compute the next-iteration input as $(1-\varsigma)\,\ell^1 + \varsigma\widehat{\ell^1}$ and go to the next iteration, where $\varsigma = 0.01$ is a damping parameter. Our criterion of convergence is that the average difference of labor choices in two subsequent iterations is smaller than 10^{-10}.

When Iteration-on-Allocation is Used

Iteration-on-allocation allows us to solve for $\{\boldsymbol{c}, \boldsymbol{\ell}\}$ that satisfy (136), (139), and (140) for any given triple $\{\boldsymbol{k}, \boldsymbol{\theta}, \boldsymbol{k}'\}$. For stochastic simulation methods, we use iteration-on-allocations twice: first, we find $\{\boldsymbol{c}_t, \boldsymbol{\ell}_t\}$ that solve (136), (139), and (140) for all $(\boldsymbol{k}_t, \boldsymbol{\theta}_t, \boldsymbol{k}_{t+1})$ that are realized in simulation. Second, we find $\left\{\boldsymbol{c}_{t+1,j}, \boldsymbol{\ell}_{t+1,j}\right\}$ that solve the same system (136), (139), and (140) in J integration nodes that correspond to each simulation point, i.e., we find the intratemporal choice for all possible state $(\boldsymbol{k}_{t+1}, \boldsymbol{\theta}_{t+1,j}, \boldsymbol{k}_{t+2,j})$ that may occur at $t+1$ given the state $(\boldsymbol{k}_t, \boldsymbol{\theta}_t, \boldsymbol{k}_{t+1})$ at t. Thus, in the inner loop, we solve for the intratemporal choice $T \times (J+1)$ times: for T simulated points and for $T \times J$ possible future states.

Similarly, for projection methods operating on a prespecified grid, we use iteration-on-allocation twice: first, we find $\{\boldsymbol{c}, \boldsymbol{\ell}\}$ that solve (136), (139), and (140) for all grid points $\left(\boldsymbol{k}_m, \boldsymbol{\theta}_m, \boldsymbol{k}'_m\right)$, where $\boldsymbol{k}'_m$ is determined by an intertemporal choice function. Second, we find $\left\{\boldsymbol{c}'_{m,j}, \boldsymbol{\ell}'_{m,j}\right\}$ that solve the same system (136), (139), and (140) for J integration nodes that correspond to each grid point, i.e., for all $\left(\boldsymbol{k}'_m, \boldsymbol{\theta}'_{m,j}, \boldsymbol{k}''_{m,j}\right)$ where $\boldsymbol{k}''_{m,j}$ is also known. Thus, in the inner loop, we solve for the intratemporal choice $M \times (J+1)$ times: for M grid points and for $M \times J$ possible future states.

Finally, we also use iteration-on-allocation when simulating the model and evaluating the accuracy of solutions. Namely, we first construct a path for capital, $\{\boldsymbol{k}_{\tau+1}\}_{t=0,\ldots,\mathcal{T}}$, and subsequently fill in the corresponding intratemporal allocations $\{\boldsymbol{c}_\tau, \boldsymbol{\ell}_\tau\}_{\tau=1,\ldots,\mathcal{T}}$ by solving (136), (139), and (140) for each given $\{\boldsymbol{k}_\tau, \boldsymbol{\theta}_\tau, \boldsymbol{k}_{\tau+1}\}$. This procedure leads to very accurate solutions. However, since the intratemporal choice is defined implicitly, simulating the model is more expensive than under explicitly defined intratemporal choice functions. We use a vectorized version of the iteration-on-allocation method that is very fast in MATLAB.

Initial Guess and Bounds

To start the iteration-on-allocation method, we assume that consumption and labor are equal to their steady-state values. To enhance the numerical stability on initial iterations when the solution is inaccurate, we impose fixed lower and upper bounds (equal to 50% and 150% of the steady-state level, respectively) on labor. This technique is similar to the moving bounds used in Maliar and Maliar (2003b) to restrict explosive simulated series. With the bounds imposed, the iteration-on-allocation procedure was numerically stable and converged to a fixed point at a good pace in all of our experiments.

Partial Convergence

We shall draw attention to an important aspect of the implementation of iteration-on-allocation in iterative cycles. Finding consumption and labor allocations with a high degree of accuracy on each outer-loop iteration has a high computational cost and is in fact of no use, since on the next iteration we must recompute consumption and labor allocations for a different intratemporal choice. We thus do not target any accuracy criteria in consumption and labor allocations in each iteration of the outer loop, but instead perform 10 subiterations on (159)–(161). We store in memory consumption and labor allocations obtained at the end of the outer loop, and we use these allocations as inputs for the next round of the iteration-on-allocation process. Thus, as the decision functions for capital are refined along the iterations, so do our consumption and labor allocations.

11.7 Accuracy Measures

The measure of accuracy of solutions is the size of residuals in $3N+1$ optimality conditions, namely, optimality conditions (138)–(140) for $h = 1, \ldots, N$ and aggregate resource constraint (136). We represent all the conditions in a unit-free form by

$$\mathcal{R}_i^{EE,h} = \beta E\left\{\frac{\lambda_i'\left[1+\frac{\phi}{2}\left(\left(\frac{\left(k_i^h\right)'}{k_i^h}\right)^2-1\right)+\left(\theta_i^h\right)' f_k^h\left(\left(k_i^h\right)',\left(\ell_i^h\right)'\right)\right]}{\lambda_i\left[1+\phi\left(\frac{\left(k_i^h\right)'}{k_i^h}-1\right)\right]}\right\}-1, \qquad (162)$$

$$\mathcal{R}_i^{MUC,h} = \frac{\lambda_i}{u_c^h\left(c_i^h,\ell_i^h\right)\tau^h}-1, \qquad (163)$$

$$\mathcal{R}_i^{MUL,h} = \frac{\lambda_i \theta_i^h f_\ell^h\left(k_i^h, \ell_i^h\right)}{u_\ell^h\left(c_i^h, \ell_i^h\right)\tau^h} + 1, \tag{164}$$

$$\mathcal{R}_i^{BC} = \frac{\sum_{h=1}^N \left[\frac{\phi}{2} k_i^h \left(\frac{\left(k_i^h\right)'}{k_i^h} - 1\right)^2\right]}{\sum_{h=1}^N \left[c_i^h + \left(k_i^h\right)' - k_i^h - \theta_i^h f^h\left(k_i^h, \ell_i^h\right)\right]} - 1. \tag{165}$$

We report two accuracy statistics, namely, the average and maximum absolute residuals across $3N + 1$ conditions computed in a given set of points.

Domains on Which Accuracy Is Evaluated

We use two alternatives for a choice of points $\{(\boldsymbol{k}_i, \boldsymbol{\theta}_i)\}_{i=1,\ldots,I}$ in which the residuals are evaluated: one is deterministic and the other is simulated. Our definition of implementation of accuracy checks is the same as the one used in Juillard and Villemot (2011).

Test 1. A deterministic set of points constructed to represent a given area of the state space. We fix a value of $r > 0$ and draw 1,000 points $\{(\boldsymbol{k}_i, \boldsymbol{\theta}_i)\}_{i=1,\ldots,1{,}000}$ such that the Euclidean distance between each point and the steady state is exactly r, i.e., $\|(\boldsymbol{k}_i, \boldsymbol{\theta}_i) - (\mathbf{1}, \mathbf{1})\| = r$, where $\mathbf{1} \in \mathbb{R}^N$ represents the steady state, and $\|\cdot\|$ is the Euclidean norm. We consider three different values for $r = \{0.01, 0.10, 0.30\}$. These values allow us to judge how accurate the solution is on a short, medium, and long distance from the steady state.

Test 2. A set of simulated points constructed to represent an essentially ergodic set. We draw a sequence of $I = 10{,}200$ shocks, calculate $\{\boldsymbol{\theta}_i\}_{i=1,\ldots,10{,}200}$ using (137), and simulate $\{\boldsymbol{k}_i\}_{i=1,\ldots,10{,}200}$ using the approximate solution $\left(k_i^h\right)' = \widehat{K}^h(\boldsymbol{k}_i, \boldsymbol{\theta}_i)$. We start simulation from the steady state $(\boldsymbol{k}_0, \boldsymbol{\theta}_0) = (\mathbf{1}, \mathbf{1})$, and we disregard the first 200 points to eliminate the effect of the initial conditions.

11.8 Explicit Versus Implicit Solutions

Suppose a numerical algorithm has delivered a solution to model (135)–(137) in the form of the capital decision functions $\left(k^h\right)' = \widehat{K}^h\left(\boldsymbol{k}, \boldsymbol{\theta}; b^h\right), h = 1, \ldots, N$. Consider two alternative methods for simulating the intratemporal choice:

1. Construct a parametric function $\ell^h = \widehat{\mathcal{L}}^h\left(\boldsymbol{k}, \boldsymbol{\theta}; b^h\right)$ and use this function to find ℓ^h for each given $(\boldsymbol{k}_t, \boldsymbol{\theta}_t)$ in simulation (the corresponding c^h can be found from FOC (140)).

2. Find ℓ^h and c^h that solve system (136), (139), and (140) using a numerical solver (i.e., Newton's method, iteration-on-allocation, etc.) for each given $(\boldsymbol{k}_t, \boldsymbol{\theta}_t)$.

It is clear that there is a trade-off between the accuracy and cost: providing explicit decision functions allows us to simulate a solution more rapidly; however, it may result in lower accuracy compared to the case when the intratemporal choice is characterized implicitly. In the context of the studied model, Maliar et al. (2011) find that alternative 1 reduces accuracy by almost an order of magnitude compared to Alternative 2. The cost of

Alternative 2 is moderate if one uses the iteration-on-allocation solver; see Maliar et al. (2011), Table 2.

The same kind of trade-off is observed in the context of DP methods. That is, we can construct explicitly both the value function, V, and some decision rules such as K^h, C^h, and $\mathcal{L}^h$, or we can construct explicitly only the value function V and define the decision rules implicitly as a solution to optimality conditions in each point considered.

12. NUMERICAL RESULTS FOR THE MULTICOUNTRY MODEL

In this section, we assess the performance of the surveyed numerical methods in the context of the multicountry model (135)–(137). We end the section by providing practical recommendations on how to apply numerical solution methods more efficiently in high-dimensional problems.

12.1 Projection Methods

We consider five projection methods; we list them in Table 21. CGA and MRGAL compute a second-degree polynomial approximation; the EDS method computes both second- and third-degree polynomial approximations (we call them EDS2 and EDS3, respectively); and SMOL-JMMV and SMOL-MKK compute an approximation that uses a mixture of the second- and fourth-degree Chebyshev polynomial terms.

Running Times

For each method in Table 22, we report the running times needed to solve the models with $N = 2, 4, 6, 8$ countries. A precise comparison of the computational cost of the studied methods is not possible because different methods are implemented using different hardware and software. Nonetheless, the running times in the table provide us with some idea about the cost of different methods. MRGAL and SMOL-MKK use Newton's solvers which are fast for small N (i.e., $N = 2$ and $N = 4$) but expensive for large N (MRGAL uses a Newton's method to solve for the coefficients of the approximating polynomials, and SMOL-MKK uses such a method to implement time iteration). In turn, the SMOL-JMMV, EDS, and CGA methods use FPI that avoids the need of numerical solvers. EDS2 becomes fastest for larger N (i.e., $N = 6$ and $N = 8$). Overall, the cost of the EDS method

Table 21 Participating projection algorithms.

Abbreviation	Name and the article
SMOL-JMMV	Smolyak method with iteration-on-allocation and FPI of Judd et al. (2013)
EDS	ε-distiguishable set algorithm of Judd et al. (2012)
CGA	Cluster grid algorithm of Maliar et al. (2011)
MRGAL	Monomial rule Galerkin method of Pichler (2011)
SMOL-MKK	Smolyak method of Malin et al. (2011)

Table 22 Running times (in seconds) for the projection methods.[a]

N	SMOL-JMMV	EDS2	EDS3	CGA	MRGAL	SMOL-MKK
2	109.5	13	20	1602.1	1.4	2.2
4	526.3	80	40	3293.4	16.0	117.1
6	1491.6	135	152	7155.7	197.3	1674.9
8	4216.7	496	4279	27785.8	2373.8	12126.8

[a]*Notes:* The following software and hardware are employed: MATLAB 7 on an Intel(R) Core(TM) 2 CPU, 3 GHz, 4 GB RAM (for MRGAL); Intel Fortran 8.0 on an Intel (R) Xeon (TM) 2.80 GHz CPU, 12 GB RAM (for SMOL-MKK); MATLAB 7 on a Quad Intel(R) Core(TM) i7 CPU920, 2.67 GHz, 6 GB RAM (for EDS and CGA); MATLAB 7 on Intel(R) Core(TM) i7 CPU920 2.67 GHz, 6 GB RAM (for SMOL-JMMV). The results for MRGAL, CGA, and SMOL-MKK are reproduced from Kollmann et al. (2011b), Table 3. The remaining results are our own computations.

grows slowly with N even under third-degree polynomials. The difference between CGA and EDS consists not only in using distinct grid points (cluster grid versus EDS grid) but also in using different integration formulas (EDS uses a cheap monomial formula $M1$ and CGA implements a two-step procedure, first using $M1$ and then using $Q(2)$); the differing cost of integration accounts for most of the difference in cost between EDS and CGA.

Accuracy on a Sphere

In Table 23, we report the absolute size of unit-free residuals on spheres of different radii; see Test 1 in Section 11.7. There are several differences across the methods that critically affect their accuracy. First, the EDS and CGA methods find a solution in a high-probability area of the state space while SMOL-JMMV, MRGAL, and SMOL-MKK operate on hypercubes that are exogenous to the model. Second, SMOL-JMMV, EDS, and CGA solve for the intratemporal choice exactly (using iteration-on-allocation), while MRGAL and SMOL-MKK approximate some of the intratemporal choice variables with state contingent functions. Finally, SMOL-JMMV and SMOL-MKK have more flexible approximating functions than do EDS2, CGA, and MRGAL methods (Smolyak polynomial consists of a mixture of second- and fourth-degree polynomial terms and has four times more terms than a complete second-degree polynomial).

The accuracy ranking of the studied methods is affected by all the above factors. The ergodic set methods (EDS and CGA) fit polynomials in a smaller area than do the SMOL-JMMV, MRGAL, and SMOL-MKK methods that operate on exogenous grids. As a result, the former methods tend to deliver more accurate solutions near the steady state than the latter methods. The SMOL-JMMV, EDS3, and SMOL-MKK have an advantage over the other methods because they use more flexible approximating functions. However, the overall accuracy of the MRGAL and SMOL-MKK methods is dominated by large errors in the intratemporal choice conditions. In particular, Kollmann et al. (2011b) compare the size of residuals across the model's equations and find that the residuals in the intratemporal choice conditions are larger than those in the intertemporal choice conditions for the MRGAL and SMOL-MKK methods. In contrast, for the SMOL-JMMV, EDS, and CGA methods, such errors are zero by construction. Finally, EDS2 performs somewhat better than CGA, which suggests that a uniform EDS grid leads to

Table 23 Accuracy of the projection methods on a sphere.

N	**SMOL-JMMV**	**EDS2**	**EDS3**	**CGA**	**MRGAL**	**SMOL-MKK**
Radius $r = 0.01$						
2	−6.75	−6.00	−6.94	−6.02	−5.00	−5.00
4	−6.30	−6.24	−7.65	−6.26	−5.11	−4.78
6	−6.47	−6.31	−7.62	−6.13	−4.95	−4.66
8	−6.36	−5.95	−7.48	−6.03	−4.98	−4.52
Radius $r = 0.10$						
2	−5.85	−4.55	−5.86	−4.28	−4.19	−4.40
4	−5.61	−4.78	−5.95	−4.58	−4.40	−4.53
6	−5.67	−5.07	−6.14	−4.88	−4.43	−4.52
8	−5.80	−5.10	−6.16	−4.96	−4.52	−4.45
Radius $r = 0.30$						
2	−4.60	−3.18	−4.11	−2.82	−2.50	−3.29
4	−4.61	−3.47	−4.30	−3.12	−2.91	−3.73
6	−4.68	−3.50	−4.39	−3.43	−3.23	−3.86
8	−4.83	−3.63	−4.46	−3.71	−3.57	−4.07

Notes: The numbers reported are maximum absolute residuals (in log10 units) across all equilibrium conditions and 1,000 draws of state variables located on spheres in the state space (centered at steady state) with radii 0.01, 0.10, and 0.30, respectively. The results for MRGAL, CGA, and SMOL-MKK are reproduced from Kollmann et al. (2011b), Table 4. The remaining results are our own computations.

Table 24 Accuracy of the projection methods on a stochastic simulation.

N	**SMOL-JMMV**	**EDS2**	**EDS3**	**CGA**	**MRGAL**	**SMOL-MKK**
Average residuals						
2	−6.70	−6.20	−7.40	−6.38	−5.69	−5.78
4	−6.24	−6.23	−7.42	−6.42	−5.92	−5.93
6	−6.13	−6.28	−7.50	−6.45	−6.05	−5.99
8	−6.13	−6.31	−7.45	−6.40	−6.09	−5.96
Maximum residuals						
2	−4.42	−4.34	−5.33	−4.50	−3.50	−3.64
4	−4.27	−4.43	−5.30	−4.48	−3.80	−3.63
6	−4.35	−4.46	−5.40	−4.39	−3.85	−3.83
8	−4.41	−4.63	−5.48	−4.54	−3.91	−3.77

Notes: The numbers reported are average (top panel) and maximum (bottom panel) absolute residuals (log10 units), taken across all equilibrium conditions and all periods for a stochastic simulation of 10,000 periods. The results for MRGAL, CGA, and SMOL-MKK are reproduced from Kollmann et al. (2011b), Table 5. The remaining results are our own computations.

more accurate approximations than a grid of clusters, which is less uniform and mimics the density function of the underlying distribution.

Accuracy on a Stochastic Simulation

In Table 24, we report the absolute size of unit-free residuals on a stochastic simulation; see Test 2 in Section 11.7. The tendencies here are similar to those we observed in the test on the spheres. Focusing on the essentially ergodic set, having a more flexible approximating function, and solving for the intratemporal choice accurately are factors that increase the accuracy of solutions. Moreover, since the accuracy is evaluated on a stochastic simulation, the ergodic set methods, EDS and CGA, have the highest chance to be accurate (because the area in which the accuracy is evaluated coincides with the area in which their polynomials are fit) and as a result, overperform the SMOL-JMMV, MRGAL, and SMOL methods which are designed to perform well on exogenous domains.

12.2 Generalized Stochastic Simulation Methods

Stochastic simulation methods find solutions by recomputing the series of endogenous state variables on each iteration (the exogenous state variables are held fixed). We consider two stochastic simulation methods, SSA and GSSA. The former is a version of the GSSA algorithm that participated in the comparison analysis of Kollmann et al. (2011b); it computes linear solutions using a one-node Monte Carlo integration rule. The latter version of GSSA is implemented exclusively in this chapter; we compute second- and third-degree polynomial approximations using three alternative integration rules, namely, one-node Monte Carlo, one-node Gauss-Hermite quadrature, and $2N$-node monomial rules described in Sections 5.3, 5.1, and 5.2, respectively. The corresponding methods are referred to as GSSA-MC, GSSA-Q(1), and GSSA-M1, respectively. For SSA studied in Kollmann et al. (2011b), a simulation is of length $T = 10{,}000$ observations. For GSSA-MC, GSSA-Q(1), and GSSA-M1, we use a shorter simulation length, $T = 2,000$.

We list the studied methods in Table 25. A detailed description of SSA is provided in Maliar et al. (2011). A general description of GSSA is provided in Section 4.2; in

Table 25 Participating stochastic simulation methods.

Abbreviation	Name
SSA	Stochastic simulation algorithm of Maliar et al. (2011)
GSSA-MC	Generalized stochastic simulation algorithm of Judd et al. (2011b) using a one-node MC integration
GSSA-Q(1)	Generalized stochastic simulation algorithm of Judd et al. (2011b) using a one-node quadrature
GSSA-M1	Generalized stochastic simulation algorithm of Judd et al. (2011b) using a monomial rule

Table 26 Running times (in seconds) of the stochastic simulation methods.

N	SSA	GSSA2-MC	GSSA3-MC	GSSA2-Q(1)	GSSA3-Q(1)	GSSA2-M1	GSSA3-M1
2	1118.1	930	1560	1134	1759	5062	24530
4	2248.0	2355	10516	2475	12002	14018	22391
6	3360.4	5052	–	5694	67407	19612	34457
8	4833.4	10057	–	11654	282437	34171	110041

Notes: The software and hardware employed are as follows: MATLAB 7 on a Quad Intel$^{(\text{®})}$ Core$^{(\text{™})}$ i7 CPU920, 2.67GHz, 6 GB RAM. The results for SSA are reproduced from Kollmann et al. (2011b), Table 3 (in terms of notations used for the other methods, SSA means GSSA1-MC). The remaining results are our own computations.

Section 11.3.4, we elaborate a description of this method for the studied multicountry model. More details about GSSA are available in Judd et al. (2011b).

Running Times

Table 26 reports the running times for the GSSA methods considered. All the results are produced by employing the same software and hardware, so that the comparison of running times across the studied methods is informative. The table shows that having more integration nodes and polynomial terms increases the cost, and the higher is the dimensionality of the problem, the larger is the increase in cost. For example, while under $N = 2$, the difference in the running time between GSSA2-Q(1) and GSSA3-Q(1) is by a factor less than 2, and under $N = 8$, this difference is larger than by a factor 20.

Accuracy on a Sphere

The results for the accuracy test on a sphere are presented in Table 27. The following tendencies are observed: First, the accuracy of solutions increases with both the degree of an approximating polynomial function and the accuracy of integration methods (GSSA3-M1 is the most accurate method in the table). Second, Monte Carlo integration is substantially less accurate than the quadrature and monomial rules, especially for $r = 0.3$. Moreover, GSSA3-MC explodes for $N = 6, 8$. Among the three methods that use Monte Carlo integration, SSA (that computes linear solutions) is the most accurate, and GSSA3-MC is the least accurate. In this case, large integration errors dominate the accuracy of solutions and a higher degree of polynomials does not lead to more accurate solutions. Furthermore, a simple one-node deterministic rule $Q(1)$ leads to sufficiently accurate solutions, in particular, for a large radius, $r = 0.3$. Finally, under accurate monomial integration rule $M1$, the accuracy levels produced by stochastic simulation methods are comparable to those produced by the projection methods studied in Section 12.1.

Accuracy on a Stochastic Simulation

The results for the accuracy test on a stochastic simulation are presented in Table 28. Again, the tendencies are similar to those we have observed in the test on the spheres. Accurate

Table 27 Accuracy of the stochastic simulation methods on a sphere.

N	SSA	GSSA2-MC	GSSA3-MC	GSSA2-Q(1)	GSSA3-Q(1)	GSSA2-M1	GSSA3-M1
Radius *r* = 0.01							
2	−4.21	−4.33	−4.05	−4.98	−5.04	−6.11	−7.59
4	−4.00	−3.63	−3.63	−5.11	−4.85	−6.35	−7.69
6	−4.12	−3.58	–	−4.97	−4.81	−6.43	−7.78
8	−4.02	−3.52	–	−4.94	−4.79	−6.45	−7.13
Radius *r* = 0.10							
2	−2.97	−3.26	−2.92	−4.50	−5.01	−4.60	−5.96
4	−3.03	−2.58	−2.57	−4.41	−4.84	−5.09	−5.96
6	−3.14	−2.48	–	−4.42	−4.80	−5.05	−6.07
8	−3.15	−2.30	–	−4.47	−4.78	−5.13	−5.55
Radius *r* = 0.30							
2	−2.03	−2.31	−1.57	−3.20	−4.06	−3.21	−4.19
4	−2.20	−1.51	−1.20	−3.23	−4.29	−3.60	−4.38
6	−2.43	−1.53	–	−3.22	−4.38	−3.49	−4.38
8	−2.41	−1.21	–	−3.33	−4.43	−3.64	−4.00

Notes: The numbers reported are maximum absolute residuals (in log10 units) across all equilibrium conditions and 1,000 draws of state variables located on spheres in the state space (centered at steady state) with radii 0.01, 0.10, and 0.30. The results for SSA are reproduced from Kollmann et al. (2011b), Table 4.

Table 28 Accuracy of the stochastic simulation methods on a simulation.

N	SSA	GSSA2-MC	GSSA3-MC	GSSA2-Q(1)	GSSA3-Q(1)	GSSA2-M1	GSSA3-M1
Average residuals							
2	−4.79	−4.77	−4.56	−5.70	−5.72	−6.31	−7.48
4	−4.69	−4.05	–	−5.71	−5.51	−6.32	−7.45
6	−4.71	−3.90	–	−5.59	−5.46	−6.39	−7.39
8	−4.65	−3.83	–	−5.54	−5.44	−6.41	−6.87
Maximum residuals							
2	−3.12	−3.10	−2.53	−4.24	−4.82	−4.28	−5.22
4	−3.17	−2.57	–	−3.91	−4.71	−4.45	−5.27
6	−3.05	−2.42	–	−4.13	−4.74	−4.50	−5.19
8	−3.14	−2.30	–	−4.26	−4.73	−4.58	−5.14

Notes: The numbers reported are averages (top panel) and maximum (bottom panel) absolute residuals (in log10 units), computed across all equilibrium conditions and all periods for a stochastic simulation of 10,000 periods. The results for SSA are reproduced from Kollmann et al. (2011b), Table 5.

integration methods and flexible approximating functions are important for accurate solutions. The performance of the Monte Carlo integration method is poor, while a simple $Q(1)$ method produces sufficiently accurate solutions in the studied examples.

12.3 Dynamic Programming Methods

In this section, we study the ECM method. Recall that the ECM method has the advantage over the Euler equation methods that it solves for control variables only at present and does not need to find such variables in all integration nodes. This advantage can be especially important in high-dimensional problems as the number of integration nodes grows with dimensionality. We consider two versions of ECM, one that solves for value function, ECM-VF, and the other that solves for derivatives of value function, ECM-DVF. We use a $2N$-node monomial integration rule and we assume the simulation length of $T = 2,000$ observations. The considered methods are listed in Table 29.

A general description of ECM-VF and ECM-DVF methods is provided in Section 7; in Sections 11.4.3 and 11.4.4, we elaborate a description of these methods for the studied multicountry model. More details about these methods are available from Maliar and Maliar (2012a,b, 2013). Both ECM-VF and ECM-DVF compute second- and third-degree polynomial approximations.

Running Times

In Table 30, we provide running times for the ECM methods. We observe that the convergence of ECM-VF is considerably faster than that of ECM-DVF.

Table 29 Participating Bellman methods.

Abbreviation	Name and the article
ECM-VF	Envelope condition method iterating on value function of Maliar and Maliar (2012a)
ECM-DVF	Envelope condition method iterating on derivatives of value function of Maliar and Maliar (2012a)

Table 30 Running times (in seconds) of the ECM methods.

N	ECM2-VF	ECM3-VF	ECM2-DVF	ECM3-DVF
2	29	34	1189	1734
4	155	1402	2039	8092
6	629	21809	2723	38698
8	2888	89872	4541	165911

Notes: The software and hardware employed are as follows: MATLAB 7 on a Quad Intel$^{(®)}$ Core$^{(™)}$ i7 CPU920, 2.67 GHz, 6 GB RAM. These results are reproduced from Maliar and Maliar (2012a), Table 2.

Table 31 Accuracy of the ECM methods on a sphere.

N	ECM2-VF	ECM3-VF	ECM2-DVF	ECM3-DVF
Radius $r = 0.01$				
2	−3.66	−3.71	−5.73	−6.95
4	−3.66	−3.68	−5.61	−6.52
6	−3.64	−3.66	−5.71	−6.37
8	−3.62	−3.68	−5.65	−6.46
Radius $r = 0.10$				
2	−2.69	−2.72	−4.49	−5.56
4	−2.70	−2.74	−4.61	−5.51
6	−2.68	−2.69	−4.48	−5.12
8	−2.65	−2.66	−4.34	−4.82
Radius $r = 0.30$				
2	−2.13	−2.18	−3.14	−4.08
4	−2.13	−2.20	−3.53	−3.98
6	−2.08	−2.16	−3.44	−3.74
8	−2.04	−2.01	−3.31	−3.40

Notes: The numbers reported are maximum absolute residuals (in log10 units) across all equilibrium conditions and 1,000 draws of state variables located on spheres in the state space (centered at steady state) with radii 0.01, 0.10, and 0.30. These results are reproduced from Maliar and Maliar (2012a), Tables 3 and 4, respectively.

Table 32 Accuracy of the ECM methods on a simulation.

N	ECM2-VF	ECM3-VF	ECM1-DVF	ECM2-DVF	ECM3-DVF
Average residuals					
2	−3.97	−4.01	−4.82	−6.06	−7.10
4	−3.86	−3.90	−4.82	−6.01	−6.87
6	−3.84	−3.88	−4.83	−5.88	−6.61
8	−3.83	−3.90	−4.84	−5.72	−6.46
Maximum residuals					
2	−2.51	−2.51	−3.01	−4.21	−4.93
4	−2.48	−2.50	−3.19	−4.32	−4.89
6	−2.47	−2.51	−3.26	−4.27	−4.76
8	−2.48	−2.48	−3.39	−4.16	−4.71

Notes: The numbers reported are averages (top panel) and maximum (bottom panel) absolute residuals (in log10 units), computed across all equilibrium conditions and all periods for a stochastic simulation of 10,000 periods. These results are reproduced from Maliar and Maliar (2012a), Table 2.

Accuracy on a Sphere

In Table 31, we report the results for the accuracy test on a sphere. We observe that ECM-VF is considerably less accurate than ECM-DVF given the same degree of approximating the polynomial. This is because if we approximate V with a polynomial of some degree, we effectively approximate V_k with a polynomial of one degree less, i.e., we "lose" one polynomial degree.

Accuracy on a Stochastic Simulation

In Table 32, we provide the results for the accuracy test on a stochastic simulation. Again, ECM-DVF is considerably more accurate than ECM-DVF. The ECM-DVF solutions are comparable in accuracy to the GSSA solutions; for comparison, see GSSA-M1 in Table 28 that uses the same integration method, $M1$, as does ECM-DVF. We conclude that value function iteration methods that approximate derivatives of value function can successfully compete with the Euler equation methods.

12.4 Local Solution Methods

We assess the performance of perturbation methods and show that the accuracy of local solutions can be considerably increased by using the techniques of a change of variables of Judd (2003) and a hybrid of local and global solutions of Maliar et al. (2013). In Table 33, we list the three perturbation-based methods considered. PER-LOG is a perturbation method in logarithms of Kollmann et al. (2011a) which participate in the JEDC comparison analysis. PER-L and HYB-L are the standard and hybrid perturbation methods in levels that are described in Sections 9.1 and 9.4, respectively; see Maliar et al. (2011) for a detailed description of these methods. PER-LOG computes perturbation solutions of orders one and two, while PER-L and HYB-L compute perturbation solutions of orders one, two, and three.

Running Times

The running times for all the perturbation methods are small; see Table 34. For example, the running time for PER1-LOG is one or two milliseconds, and that for PER2-LOG is a few seconds.

Table 33 Participating perturbation methods.

Abbreviation	Name
PER-LOG	Perturbation method in logs of variables of Kollmann et al. (2011b)
PER-L	Perturbation method in levels of variables of Maliar et al. (2012)
HYB-L	Perturbation-based hybrid method of Maliar et al. (2012)

Table 34 Running times (in seconds) for the perturbation methods.

N	PER1-L	PER2-L	PER3-L	PER1-LOG	PER2-LOG
2	0.0	0.0	0.0	0.1	0.3
4	0.1	0.1	0.5	0.1	0.5
6	0.1	1.1	1.6	0.2	1.8
8	0.0	0.1	1.3	0.2	4.3

Notes: The following software and hardware are employed: MATLAB 7 on an Intel(®) Pentium(®) 4 CPU, 3.06 GHz, 960 MB RAM (for PER-LOG); Dynare and C++ on workstation with two quad-core Intel® Xeon X5460 processors (clocked at 3.16 GHz), 8 GB of RAM, and running 64 bit Debian GNU/Linux (for PER-L and HPER-L). The results for PER-L and HPER-L are reproduced from Maliar et al. (2012), Tables 1 and 2, and the results for PER-LOG are reproduced from Kollmann et al. (2011b), Table 5.

12.4.1 The Importance of the Change of Variables

The importance of the change of variables can be seen by comparing the results produced by the loglinear perturbation solution methods of Kollmann et al. (2011a) and the linear perturbation solution methods of Maliar et al. (2013).

Table 35 Accuracy of the perturbation methods on a sphere.

N	PER1-L	PER2-L	PER3-L	PER1-LOG	PER2-LOG
Radius $r = 0.01$					
2	−4.32	−5.26	−6.69	−3.04	−5.01
4	−4.34	−5.25	−6.34	−3.26	−4.92
6	−4.43	−5.29	−6.26	−3.50	−5.10
8	−4.60	−5.34	−6.23	−3.65	−5.35
Radius $r = 0.10$					
2	−2.25	−3.64	−4.76	−1.07	−2.29
4	−2.27	−3.73	−5.26	−1.27	−1.93
6	−2.42	−3.88	−5.37	−1.48	−2.22
8	−2.55	−3.93	−5.48	−1.65	−2.41
Radius $r = 0.30$					
2	−1.11	−1.99	−2.03	−0.29	−0.68
4	−1.13	−2.10	−3.13	−0.41	−0.53
6	−1.31	−2.36	−3.41	−0.54	−0.79
8	−1.47	−2.46	−3.51	−0.69	−1.02

Notes: The numbers reported are maximum absolute residuals (in log10 units) across all equilibrium conditions and 1,000 draws of state variables located on spheres in the state space (centered at steady state) with radii 0.01, 0.10, and 0.30. The results for PER-LOG are reproduced from Kollmann et al. (2011b), Table 4, and those for PER-L are reproduced from Maliar et al. (2012), Tables 1 and 2.

Table 36 Accuracy of the perturbation methods on a stochastic simulation.

N	PER1-L	PER2-L	PER3-L	PER1-LOG	PER2-LOG
Average residuals					
2	−3.05	−4.21	−5.18	−3.53	−4.84
4	−3.10	−4.37	−5.39	−3.67	−4.82
6	−3.22	−4.58	−5.65	−3.78	−4.92
8	−3.12	−4.43	−5.48	−3.86	−4.99
Maximum residuals					
2	−1.55	−2.25	−2.97	−1.45	−2.30
4	−1.58	−2.54	−3.39	−1.44	−2.32
6	−1.64	−2.48	−3.34	−1.55	−2.16
8	−1.58	−2.40	−3.22	−1.67	−2.38

Notes: The numbers reported, in log10 units, are averages (top panel) and maxima (bottom panel) of absolute values of the model's residuals, where the averages/maxima are taken across all equilibrium conditions and all dates for a stochastic simulation run of 10,000 periods. The results for PER-LOG are reproduced from Kollmann et al. (2011b), Table 5, and those for PER-L are reproduced from Maliar et al. (2012), Tables 1 and 2.

Accuracy on a Sphere

The results for the accuracy test on a sphere are reported in Table 35. Two tendencies are observed from the table. First, the perturbation method in levels, PER-L, performs very well; it delivers accuracy levels that are comparable to those produced by global projection and stochastic simulation methods (given the same degree of an approximating polynomial function); for a comparison, see Tables 23 and 27, respectively. However, the performance of the perturbation method in logarithms, PER-LOG, is relatively poor. Even for the second-order method PER2-LOG, the maximum residuals can be as large as 8.5%, and they can be even larger for the first-order method PER1-LOG. We therefore conjecture that for this specific model, a change of variables from logarithms to levels increases the accuracy of solutions, especially for large deviations from the steady state.

Accuracy on a Stochastic Simulation

The results for the accuracy test on a stochastic simulation are provided in Table 36. Surprisingly, there is not much difference between PER-L and PER-LOG when the accuracy is evaluated on a stochastic simulation: the maximum residuals for the two methods are practically identical.

12.4.2 The Benefits of Hybrid Solutions

To implement the hybrid perturbation-based method, we fix the capital decision functions $k_{t+1}^h = K^h\left(\boldsymbol{k}_t, \boldsymbol{\theta}_t; b^h\right), h = 1, \ldots, N$ that are produced by the PER-L method, and we find $2N + 1$ allocations $\{\boldsymbol{c}_t, \boldsymbol{\ell}_t, \lambda_t\}$ satisfying (136), (139), and (140) using the

Table 37 Perturbation versus hybrid perturbation-based methods in the multicountry model with $N = 8$.

	1st order		2nd order		3rd order	
	PER	**HYB**	**PER**	**HYB**	**PER**	**HYB**
Solution time	0.03	0.03	0.06	0.06	1.32	1.32
Radius $r = 0.01$						
EulerEq	−6.26	−3.85	−7.07	−6.36	−7.85	−7.37
MUCons	−5.28	–	−5.41	–	−6.23	–
MULabor	−4.60	–	−5.34	–	−6.57	–
WorResConst	−5.25	–	−6.22	–	−7.65	–
Overall	−4.60	−3.85	−5.34	−6.36	−6.23	−7.37
Radius $r = 0.10$						
EulerEq	−4.20	−3.23	−5.04	−4.63	−6.76	−6.11
MUCons	−3.28	–	−4.54	–	−6.02	–
MULabor	−2.55	–	−3.93	–	−5.48	–
WorResConst	−3.25	–	−4.75	–	−6.18	–
Overall	−2.55	−3.23	−3.93	−4.63	−5.48	−6.11
Radius $r = 0.30$						
EulerEq	−3.17	−2.14	−3.57	−3.06	−4.68	−4.22
MUCons	−2.32	–	−3.77	–	−4.74	–
MULabor	−1.47	–	−2.46	–	−3.51	–
WorResConst	−2.29	–	−3.38	–	−4.29	–
Overall	−1.47	−2.14	−2.46	−3.06	−3.51	−4.22
Stochastic simulation						
EulerEq	−3.53	−2.30	−3.49	−3.45	−4.37	−5.02
MUCons	−1.79	–	−2.66	–	−3.52	–
MULabor	−1.58	–	−2.40	–	−3.22	–
ResConst	−2.22	–	−3.44	–	−4.48	–
Overall	−1.58	−2.30	−2.40	−3.45	−3.22	−5.02

Notes: Both PER and HYB compute solutions in levels. For each model equation (listed in the first column), the table reports maximum absolute errors in log10 units across countries and test points. For panels "Radius $r = 0.01$," "Radius $r = 0.10$," and "Radius $r = 0.30$," the set of test points is 1,000 draws of state variables located on spheres with radii 0.01, 0.10, and 0.30, respectively; for panel "stochastic simulation," the set of test points is a stochastic simulation of 10,000 periods. An entry "–" is used if accuracy measure is below −10 (such errors are viewed as negligible), PER is PER-L, and HYB is HYB-L.

Model equations are as follows: "EulerEq" is Euler equation (74); "MUCons" equates the (scaled) marginal utility of consumption to the Lagrange multiplier, see (72); "MULabor" equates the (scaled) marginal utility of labor to marginal productivity of labor multiplied by the Lagrange multiplier, see (73); "WorResConst" is world resource constraint (136); "Overall" is the maximum error across all the model's equations; "Solution time" is time for computing a solution. The results for PER and HYB are reproduced from Maliar et al. (2012), Table 2 with kind permission from Springer Science+Business Media B.V.

iteration-on-allocation numerical solver as described in Section 11.6. The results are presented in Table 37. We observe that the plain perturbation method, PER-L, produces nonnegligible residuals in all the model's equations, while the hybrid method, HYB, produces nonnegligible residuals only in the Euler equations (the quantities delivered by the iteration-on-allocation solver, by construction, satisfy the intratemporal conditions exactly). In terms of maximum size of the residuals, the hybrid solutions are sufficiently more accurate than the plain perturbation solutions. The difference in accuracy between PER-L and HYB-L reaches almost two orders of magnitude in the test on a stochastic simulation.

12.5 Speeding up Computations in MATLAB

In this section, we provide some results on how the cost of numerical methods can be reduced in large-scale applications using MATLAB. Our presentation closely follows Valero et al. (2013), who explore several options for reducing the cost of a Smolyak solution method in the context of model (135)–(137). We focus on three different tools available in MATLAB: parallel computation on multiple CPUs using a "*parfor*" routine; automated translations of the code to C using a "*mex*" routine; and parallel computation using multiple GPUs. We refer to the standard MATLAB and these three alternative implementations as "standard," "parfor," "mex," and "GPU," respectively.

The literature on parallel computing often considers examples in which gains from parallelization are readily obtained. In contrast, the Smolyak method studied in Valero et al. (2013) is a challenging case for parallelization. First, there are large information transfers between the outer and inner loops in the Smolyak method and second, a large fraction of the Smolyak code must be implemented in a serial manner. The running times for our experiments are provided in Table 38 (the solutions delivered are identical in all cases).

Our main finding is that parallel computation using multiple CPUs can speed up the methods for solving dynamic economic models if the dimensionality of the problem is high. To be specific, the parfor implementation does not speed up computations under $N = 2$; however, it is almost four times faster under $N = 8$. The efficiency of paralleliza-

Table 38 Running times (in seconds) depending on the implementaion in MATLAB: parfor, mex, GPU.

N	standard	parfor	mex	GPU
2	109.5	106.8	11.9	24453
4	526.3	302.4	202.9	150800
6	1491.6	517.8	1545.5	–
8	4216.7	1206.6	9272.7	–

Notes: The following software and hardware are employed: MATLAB 2012a, Intel$^{(\text{®})}$ Core$^{(\text{TM})}$ i7-2600 CPU @ 3.400 GHz with RAM 12.0 GB and GPU GeFoce GT 545. Also, to compile mex functions, we use Microsoft Windows SDK 7.1 with NET Framework 4.0. These results are reproduced from Valero et al. (2013), Table 2.

tion (132) in the last case is nearly 90% on our four-core machine, namely, $\frac{4216.7}{1206.6\times 4} \approx 0.88$. The mex translation brings speedups in problems with low dimensionality but performs poorly when dimensionality increases. This is because for large problems, nonoptimized C code produced by a mex routine is less efficient than a vectorized MATLAB code. Finally, parallel computation using multiple GPUs does poorly in our case because of standard limitations of GPUs (namely, a high cost of transfers) and because of limitations of the MATLAB language in operating GPUs.

12.6 Practical Recommendations About Solving High-Dimensional Problems: Summary

When designing a solution method, we pursue two main objectives: a high accuracy of solutions and a low computational expense. There is a nontrivial trade-off between these two objectives. We now provide some considerations on how to take into account such a trade-off in order to make solution methods more effective in high-dimensional applications.

(i) Producing accurate solutions is costly in high-dimensional applications. The numbers of grid points, integration nodes, and polynomial terms, as well as the number and complexity of the model's equations, grow rapidly all together with dimensionality of the state space. Accuracy levels of 10^{-9} that one easily achieves in some model with two state variables are unlikely to be feasible in a similar model with one hundred state variables. Therefore, the first question one must ask is: "How much accuracy do I need in a given application?" The answer depends on the economic significance of the approximation errors. Solutions that are accurate for one purpose may be inaccurate for another purpose. For example, a perturbation method that has large 5% errors in the model's variables may still be sufficiently accurate for the purpose of evaluating second moments, since a typical sampling error in empirical data is still larger. However, this accuracy level is not sufficient for forecasting; for example, predicting that the US growth rate could be anything within the 5% interval $[-3\%, 7\%]$ is too loose to be useful.

(ii) Given a target accuracy level, the second question one must ask is: "What combination of techniques can attain the given accuracy level at the lowest possible cost?" Here, the cost must be understood as the sum of both running time and programmer's time. Perturbation methods (incorporated, e.g., in the Dynare platform) are often the cheapest possible alternative in all respects and can deliver accurate solutions to many smooth problems, especially if one uses changes of variables and hybrids of local and global solutions described in this section. Global solution methods are more accurate and flexible but also more costly in terms of both the running time and the programmer's efforts.

(iii) In the case of global solution methods, coordination in the choice of computational techniques is important for making a solution method cost-efficient. All computational techniques employed must be suitable for high-dimensional applications. For example, if one uses a tensor-product grid, the cost will be prohibitive no matter how efficiently we compute integrals or solve for the intratemporal choice. Moreover, all the techniques used must match each other in terms of attainable accuracy. For example, if one uses Monte Carlo integration, the solutions will be inaccurate no matter how sophisticated is the grid and how flexible is the approximating function.

(iv) Among the global solution methods considered, GSSA delivers a high accuracy of solutions and is very simple to program. It is an especially convenient choice if one needs a solution to be accurate in the high-probability area of the state space (i.e., on a set of points produced by stochastic simulation). The EDS and cluster grid methods require more programming efforts but are also faster. A Smolyak method produces solutions that are uniformly accurate in a hypercube (at the cost of a somewhat lower accuracy in the high-probability area). It is an especially useful choice when the solution must be accurate not only inside but also outside the high-probability area of the state space, for example, when modeling the evolution of a development economy that starts off far below the steady state.

(v) To keep the exposition simple, we limit ourselves to solving stylized one- and multiagent growth models. However, the surveyed techniques can be used to solve a variety of other interesting economic models, such as new Keynesian models, life-cycle models, heterogeneous-agents models, asset-pricing models, multisector models, multicountry models, climate change models, etc. The solution procedures will be similar to those we described in the chapter. Namely, under global solution methods, we parameterize some model's variables (such as decision functions of heterogeneous agents, firms, countries, etc.) with flexible functional forms, approximate integrals using some discretization method, and solve the resulting systems of the model's equations with respect to parameters of the approximating functions. Under local solution methods, we produce a perturbation solution and modify this solution to increase its accuracy. Examples of solution procedures for life-cycle models can be found in Krueger and Kubler (2004), and those for new Keynesian models can be found in Judd et al. (2011d, 2012), Fernández-Villaverde et al. (2012), and Aruoba and Schorfheide (2012).

(vi) Orthogonal polynomial families (such as Chebyshev or Hermite) are convenient for numerical work. They prevent us from having ill-conditioned inverse problems, ensure well-defined interpolation, and allow us to derive the interpolation coefficients analytically. However, they are also more costly to construct than the ordinary polynomials and require data transformations. Ordinary polynomials are a possible choice if combined with numerically stable regression methods and if the number of grid points is somewhat larger than the number of polynomial coefficients.

(vii) In the fitting step, we recommend avoiding standard least-squares regression methods (OLS, Gauss-Newton method) and using instead methods that can handle ill-conditioned problems, such as least-squares methods using either QR factorization or SVD or Tikhonov regularization. If the problem is not ill-conditioned, these methods give the same (or almost the same) answer as the standard least-squares method. However, if the problem is ill-conditioned, the standard least-squares methods will fail, while these other methods will succeed.

(viii) For approximating integrals, we recommend using monomial formulas (combined with Cholesky decomposition in the case of serially correlated shocks). The monomial formula with $2N$ nodes produces very accurate approximations; the formula with $2N^2 + 1$ is even more accurate. We recommend avoiding the use of simulation-based integration methods (such as Monte Carlo, learning, and nonparametric ones) because their convergence rate (accuracy) is low.

(ix) For solving systems of nonlinear equations, we recommend using a fixed-point iteration method instead of quasi-Newton's methods, especially if the system of equations is large. Iteration-on-allocation is a simple and effective way to find a solution to a system of the intratemporal choice conditions.

(x) In the case of dynamic programming, conventional VFI based on time iteration is expensive. Using other, cheaper versions of fixed-point iteration for VFI, such as endogenous grid and envelope condition methods, can help us to substantially reduce the cost.

(xi) It is important to accurately solve all the model's equations, in particular the intratemporal choice ones. If one equation is solved inaccurately, the entire solution is inaccurate. Furthermore, it is important to solve accurately the model's equations not only when computing solutions but also when simulating the model and evaluating the accuracy of solutions.

(xii) Precomputation can save a lot of time: instead of computing the same thing all over again, we compute it just once, at the beginning of a solution procedure.

(xiii) It is useful to check for codes that are available on the Internet before writing your own software. For example, a MATLAB code accompanying the GSSA method includes generic routines for implementing many numerically stable LS and LAD methods, a routine for generating multidimensional polynomials, and a routine for multidimensional Gauss-Hermite quadrature and monomial integration methods. The code also contains a test suite for evaluating the accuracy of solutions. Not only can this code solve the studied examples but it can be easily adapted to other problems in which the reader may be interested.

(xiv) Parallel computation is a promising tool for many problems but it is not automatically useful in every possible context. Not every method is naturally parallelizable. In some cases, the cost of transfers between the cores can outweigh the gains from parallelization. Also, we must design numerical methods in a way that is suitable for parallelization.

(xv) MATLAB is a useful tool when working on a desktop. It is not well suited for supercomputers, in particular because there is a license limitation. To benefit from supercomputers, one must first make an investment in learning some lower-level programming languages (such as Fortran, C, or Python) as well as learning the Unix operational system and software that supports parallel computation (such as Open MP or MPI).

(xvi) Last but not least, it is always necessary to check the quality of the approximations obtained by, for example, evaluating unit-free residuals on a set of points constructed to represent the domain of interest. The code may have bugs or the accuracy may be insufficient; we are at risk of producing a bunch of random numbers if we do not run accuracy checks.

13. CONCLUSION

Recent developments in the field of numerical analysis have extended the horizons of what was thought unfeasible just a few years ago. First of all, these are novel solution methods that are tractable, accurate, and reliable in large-scale applications. We build such methods using nonproduct grids, monomial integration methods, derivative-free solvers, and numerically stable regression methods. To simplify rootfinding in the Bellman equation, we employ endogenous grid and envelope conditions methods. To avoid repeating some computations, we use precomputation techniques. Finally, to increase accuracy of perturbation methods, we use changes of variables and construct hybrids of local and global solutions. Taken together, these techniques enable us to accurately solve models with nearly a hundred of state variables using a standard desktop computer and serial MATLAB software.

Parallel computing opens another dimension in numerical analysis of economic models. Gains from parallelization are possible even on desktop computers with few cores. Supercomputers have thousands and thousands of CPUs and GPUs that can be coordinated for computationally intensive tasks. Also, they have large memories to record the results. We hope that these new capacities and possibilities will bring economic research to a qualitatively new level in terms of generality, empirical relevance, and rigor of results.

ACKNOWLEDGMENTS

Lilia Maliar and Serguei Maliar acknowledge support from the Hoover Institution and Department of Economics at Stanford University, University of Alicante, Ivie, MCI, and FEDER funds under the Projects SEJ-2007-62656 and ECO2012-36719. We thank the editors Karl Schmedders and Kenneth L. Judd as well as two anonymous referees for many valuable comments. We thank Eric Aldrich, Markus Baldauf, Phillip Blood, Yongyang Cai, Kenneth L. Judd, and Rafael Valero for useful discussions of several issues related to parallel computation. Juan Mora López made useful suggestions about the use of nonparametric statistics in the context of numerical solution methods. We acknowledge XSEDE Grant TG-ASC120048, and we thank Phillip Blood and Rick Costa, scientific specialists from the Pittsburgh Supercomputing Center, for technical support. Much of the material in this chapter builds on our prior joint work with Kenneth L. Judd, Rafael Valero, and Sébastien Villemot. We benefited from the comments of seminar participants at

Autonomous University of Barcelona, Birkbeck University of London, Canadian Central Bank, Center for Financial Studies at the European Central Bank, Cornell University, Federal Reserve Bank of San Francisco, Paris School of Economics, Santa Clara University, Stanford University, University of Alicante, University of Bilbao, University of California at Berkeley, University of Chicago, University of Edinburgh, and University of Oxford. We used the material from this chapter in the graduate courses at Stanford University, University of Chicago (ICE 2012), and University of Alicante, and we received a useful feedback from many graduate students, in particular, Monica Bhole, Daniel-Oliver Garcia-Macia, Moritz Lenel, Jessie Li, Davide Malacrino, Erik Madsen, Inna Tsener, and Rafael Valero.

REFERENCES

Acemoglu, D., Golosov, M., Tsyvinski, A., Yared, P., 2011. A dynamic theory of resource wars. Quarterly Journal of Economics 127 (1), 283–331.

Adda, J., Cooper, R., 2003. Dynamic Economics: Quantitative Methods and Applications. The MIT Press, Cambridge, Massachusetts, London, England.

Adjemian, S., Bastani, H., Juillard, M., Mihoubi, F., Perendia, G., Ratto, M., Villemot, S., 2011. Dynare: reference manual, version 4. Dynare Working Papers 1, CEPREMAP.

Aiyagari, R., 1994. Uninsured idiosyncratic risk and aggregate saving. Quarterly Journal of Economics 109, 659–684.

Aldrich, E.M., Fernández-Villaverde, J., Gallant, R., Rubio-Ramírez, J., 2011. Tapping the supercomputer under your desk: solving dynamic equilibrium models with graphics processors. Journal of Economic Dynamics and Control, Elsevier 35 (3), 386–393.

Altig, D., Christiano, L., Eichenbaum, M., 2004. Firm-specific capital, nominal rigidities and the business cycle. Review of Economic Dynamics 14 (2), 225–247.

Amador, M., Weil, P.-O., 2010. Learning from prices: public communication and welfare. Journal of Political Economy 118 (5), 866–907.

Amdahl, G. 1967. The validity of single processor approach to achieving large scale computing capabilities. in: AFIPS procedings, pp. 483–485.

Amman, H., 1986. Are supercomputers useful for optimal control experiments? Journal of Economic Dynamics and Control 10, 127–130.

Amman, H., 1990. Implementing stochastic control software on supercomputing machines. Journal of Economic Dynamics and Control 14, 265–279.

Anderson, G., Kim, J., Yun, T., 2010. Using a projection method to analyze inflation bias in a micro-founded model. Journal of Economic Dynamics and Control 34 (9), 1572–1581.

Aruoba, S.B., Schorfheide, F., 2012. Macroeconomic dynamics near the ZLB: a tale of two equilibria. <http://www.ssc.upenn.edu/schorf/papers/AS-ZLB.pdf>.

Aruoba, S.B., Fernández-Villaverde, J., Rubio-Ramírez, J., 2006. Comparing solution methods for dynamic equilibrium economies. Journal of Economic Dynamics and Control 30, 2477–2508.

Attanasio, O., Pavoni, N., 2011. Risk sharing in private information models with asset accumulation: explaining the asset smoothness of consumption. Econometrica 79 (4), 1027–1068.

Bai, Y., Ríos-Rull, J.-V., Storesletten, K., 2012. Demand shocks as productivity shocks. Manuscript.

Barillas, F., Fernández-Villaverde, J., 2007. A generalization of the endogenous grid method. Journal of Economic Dynamics and Control, Elsevier 31, 2698–2712.

Barthelmann, V., Novak, E., Ritter, K., 2000. High dimensional polynomial interpolation on sparse grids. Advances in Computational Mathematics 12, 73–288.

Baryshnikov, Yu., Eichelbacker, P., Schreiber, T., Yukich, J.E., 2008. Moderate deviations for some point measures in geometric probability. Annales de l'Institut Henri Poincaré – Probabilités et Statistiques 44, 422–446.

Bellman, R.E., 1961. Adaptive Control Processes. Princeton University Press, Princeton, NJ.

Bertsekas, D., Tsitsiklis, J., 1996. Neuro-Dynamic Programming. Optimization and Neural computation series. Athena Scientific, Belmont, Massachusetts.

Bierens, H.J., 1994. Topics in Advanced Econometrics. Cambridge University Press.

Birge, J.R., Louveaux, F.V., 1997. Introduction to Stochastic Programming. Springer-Verlag, New York.

Blood, P. 2011. Getting started using national computing resources. <http://staff.psc.edu/blood/ICE11/XSEDEICEJuly2011.pdf>.

Bloom, N., 2009. The impact of uncertainty shocks. Econometrica 77 (3), 623–685.

Borovička, Hansen, L.P., 2012. Examining macroeconomic models through the lens of asset pricing. Federal Reserve Bank of Chicago Working Paper 01.

Cai, Y., Judd, K.L., 2012. Dynamic programming with shape-preserving rational spline Hermite interpolation. Economics Letters 117, 161–164.

Cai, Y., Judd, K.L., Lontzek, T., 2012. DSICE: a dynamic stochastic integrated model of climate and economy. Manuscript.

Cai, Y., Judd, K.L., Lontzek, T., 2013a. Continuous-time methods for integrated assessment models. NBER Working Paper 18365.

Cai, Y., Judd, K.L., Train, G., Wright, S. 2013b. Solving dynamic programming problems on a computational grid. NBER Working Paper 18714.

Canova, F., 2007. Methods for Applied Macroeconomic Research. Princeton University Press.

Carroll, K., 2005. The method of endogenous grid points for solving dynamic stochastic optimal problems. Economic letters 91, 312–320.

Caselli, F., Koren, M., Lisicky, M., Tenreyro, S., 2011. Diversification through trade. Manuscript.

Chari, V.V., Kehoe, P., McGrattan, E., 2009. New Keynesian models: not yet useful for policy analysis. American Economic Journal: Macroeconomics 1 (1), 242–266.

Chatterjee, S., Corbae, D., Nakajima, M., Ríos-Rull, J.-V., 2007. A quatitative theory of unsecured consumer credit with risk of default. Econometrica 75 (6), 1525–1589.

Chen, B., Zadrozny, P., 2009. Multi-step perturbation solution of nonlinear differentiable equations applied to an econometric analysis of productivity. Computational Statistics and Data Analysis 53 (6), 2061–2074.

Cho, I.C., Sargent, T.J., 2008. Self-confirming equilibrium. In: Durlauf, S., Blume, L. (Eds.), The New Palgrave Dictionary of Economics, Palgrave Macmillan.

Chong, Y., Hendry, D., 1986. Econometric evaluation of linear macroeconomic models. The Review of Economic Studies 53 (4), 671–690.

Christiano, L., Fisher, D., 2000. Algorithms for solving dynamic models with occasionally binding constraints. Journal of Economic Dynamics and Control 24, 1179–1232.

Christiano, L., Eichenbaum, M., Evans, C., 2005. Nominal rigidities and the dynamic effects of a shock to monetary policy. Journal of Political Economy 113 (1), 1–45.

Christiano, L., Eichenbaum, M., Rebelo, S., 2011. When is the government spending multiplier large? Journal of Political Economy 119 (1), 78–121.

Cogan, J., Taylor, L., Wieland, V., Wolters, M., 2013. Fiscal consolidation strategy. Journal of Economic Dynamics and Control 37, 404–421.

Cogley, T., Sargent, T., Tsyrennikov, V., 2013. Wealth dynamics in a bond economy with heterogeneous beliefs. Manuscript.

Coibion, O., Gorodnichenko, Y., 2008. What can survey forecasts tell us about informational rigidities? Journal of Political Economy 120 (1), 116–159.

Coleman, W., 1992. Solving nonlinear dynamic models on parallel computers. Discussion Paper 66, Institute for Empirical Macroeconomics, Federal Reserve Bank of Minneapolis.

Collard, F., Juillard, M., 2001. Accuracy of stochastic perturbation methods: the case of asset pricing models. Journal of Economic Dynamics and Control 25, 979–999.

Constantinides, G., 1982. Intertemporal asset pricing with heterogeneous consumers and without demand aggregation. Journal of Business 55, 253–267.

Creel, M., 2005. User-friendly parallel computations with econometric examples. Computational Economics 26 (2), 107–128.

Creel, M., 2008. Using parallelization to solve a macroeconomic model: a parallel parameterized expectations algorithm. Computational Economics 32, 343–352.

Creel, M., Goffe, W., 2008. Multi-core CPUs, clusters, and grid computing: a tutorial. Computational Economics 32 (4), 353–382.

Davidson, R., MacKinnon, J., 1993. Estimation and Inference in-Econometrics. Oxford University Press, New York, Oxford.

Del Negro, M., Schorfheide, F., Smets, F., Wouters, R., 2007. On the fit of new Keynesian models. Journal of Business and Economic Statistics 25 (2), 123–143.

Den Haan, W., 1990. The optimal inflation path in a Sidrauski-type model with uncertainty. Journal of Monetary Economics 25, 389–409.

Den Haan, W., 2010. Comparison of solutions to the incomplete markets model with aggregate uncertainty. Journal of Economic Dynamics and Control 34, 4–27.

Den Haan, De Wind, J., 2012. Nonlinear and stable perturbation-based approximations. Journal of Economic Dynamics and Control 36 (10), 1477–1497.

Den Haan, W., Marcet, A., 1990. Solving the stochastic growth model by parameterized expectations. Journal of Business and Economic Statistics 8, 31–34.

Den Haan, W., Marcet, A., 1994. Accuracy in simulations. Review of Economic Studies 6, 3–17.

Den Haan, W., Judd, K.L., Juillard, M., 2011. Computational suite of models with heterogeneous agents II: multicountry real business cycle models. Journal of Economic Dynamics and Control 35, 175–177.

Dmitriev, A., Roberts, I., 2012. International business cycles with complete markets. Journal of Economic Dynamics and Control 36 (6), 862–875.

Doornik, J.A., Hendry, D.F., Shephard, N. 2006. Parallel computation in econometrics: A simplified approach. In: Kontoghiorghes, E.J. (Ed.), Handbook of Parallel Computing and Statistics. Chapman & Hall/CRC, London, pp. 449–476.

Dotsey, M., Mao, C.S., 1992. How well do linear approximation methods work? The production tax case. Journal of Monetary Economics 29, 25–58.

Duffie, D., 2010. Presidential address: asset pricing dynamics with slow-moving capital. Journal of Finance LXV(4), 1237–1267.

Durham, G., Geweke, J., 2012. Adaptive sequential posterior simulators for massively parallel computing environments. Manuscript.

Eaves, B., Schmedders, K., 1999. General equilibrium models and homotopy methods. Journal of Economic Dynamics and Control 23, 1249–1279.

Ellison, M., Sargent, T., 2012. A defence of the FOMC. International Economic Review 53 (4), 1047–1065.

Evans G.W., Honkapohja, S., 2001. Learning and Expectations in Macroeconomics. Princeton University Press.

Evans, R., Kotlikoff, L., Phillips, K., 2012. Game over: simulating unsustainable fiscal policy. In: Fiscal Policy after the Financial Crisis National Bureau of Economic Research, NBER Chapters.

Fair, R., Taylor, J., 1983. Solution and maximum likelihood estimation of dynamic nonlinear rational expectation models. Econometrica 51, 1169–1185.

Feng, Z., Miao, J., Peralta-Alva, A., Santos, M., 2009. Numerical simulation of nonoptimal dynamic equilibrium models. Working papers Federal Reserve Bank of St. Louis 018.

Fernández-Villaverde, J., Rubio-Ramírez, J., 2006. Solving DSGE models with perturbation methods and a change of variables. Journal of Economic Dynamics and Control 30, 2509–2531.

Fernández-Villaverde, J., Rubio-Ramírez, J., 2007. Estimating macroeconomic models: a likelihood approach. Review of Economic Studies 74, 1059–1087.

Fernández-Villaverde, J., Gordon, G., Guerrón-Quintana, P., Rubio-Ramírez, J., 2012. Nonlinear adventures at the zero lower bound. NBER Working Paper 18058.

Fudenberg, D., Levine, D., 1993. Self-confirming equilibrium. Econometrica 61, 523–545.

Fukushima, K., Waki, Y., 2011. A Polyhedral Approximation Approach to Concave Numerical Dynamic Programming. Manuscript.

Gallant, R.A., 2012. Parallelization strategies: hardware and software (two decades of personal experience). <http://www.duke.edu/ arg>.

Gaspar, J., Judd, K.L., 1997. Solving large-scale rational-expectations models. Macroeconomic Dynamics 1, 45–75.

Gertler, M., Leahy, J., 2008. A Phillips curve with an Ss foundation. Journal of Political Economy 110 (3), 533–572.

Gertler, M., Kiyotaki, N., Queralto, A., 2011. Financial crises, bank risk exposure and government financial policy. Manuscript.

Geweke, J., 1996. Monte Carlo simulation and numerical integration. In: Amman, H., Kendrick, D., Rust, J. (Eds.), Handbook of Computational Economics. Elsevier Science, Amsterdam, pp. 733–800.

Glover A., Heathcote, J., Krueger, D., Ríos-Rull, J.-V., 2011. Intergenerational redistribution in the great recession. NBER Working Paper 16924.

Golosov, M., Sargent, T., 2012. Taxation, redistribution, and debt with aggregate shocks. Manuscript.

Golosov, M., Troshkin, M., Tsyvinski, A., 2011. Optimal dynamic taxes. NBER Working Paper 17642.

Golub, G., Van Loan, C., 1996. Matrix Computations. The Johns Hopkins University Press, Baltimore and London.

Gomes, F., Kotlikoff, L., Viceira, L., 2008. Optimal life-cycle investing with flexible labor supply: a welfare analysis of life-cycle funds. American Economic Review: Papers and Proceedings 98 (2), 297–303.

Gomme, P., Klein, P., 2011. Second-order approximation of dynamic models without the use of tensors. Journal of Economic Dynamics and Control 35, 604–615.

Gorman, W., 1953. Community preference field. Econometrica 21, 63–80.

Graham, L., Wright, S., 2009. Information, heterogeneity and market incompleteness. Journal of Monetary Economics 57 (2), 164–174.

Guerrieri, L., Iacoviello, M., 2013. OccBin: A toolkit for solving dynamic models with occasionally binding constraints easily. Manuscript.

Guerrieri, V., Lorenzoni, G., 2011. Credit crises, precautionary savings, and the liquidity trap. NBER Working Papers 17583.

Guibaud, S., Nosbusch, Y., Vayanos, D., forthcoming. Bond market clienteles, the yield curve, and the optimal maturity structure of government debt. Review of Financial Studies.

Guvenen, F., 2011. Macroeconomics with heterogeneity: a practical guide. NBER Working Papers 17622.

Hall, R.E., 2012. Quantifying the Forces Leading to the Collapse of GDP after the Financial Crisis. Manuscript.

Hasanhodzic, J., Kotlikoff, L.J., 2013. Generational risk – is it a big deal?: Simulating an 80-period OLG model with aggregate shocks. NBER 19179.

Heathcote, J., Perri, F., 2013. The international diversification puzzle is not as bad as you think. Manuscript.

Heathcote, J., Storesletten, K., Violante, G., 2009. Quantitative macroeconomics with heterogeneous households. Annual Review of Economics, Annual Reviews 1 (1), 319–354.

Heer, B., Maußner, A., 2008. Computation of business cycle models: a comparison of numerical methods. Macroeconomic Dynamics 12, 641–663.

Heer, B., Maußner, A., 2010. Dynamic General Equilibrium Modeling. Springer-Verlag, Berlin Heidelberg.

Hoerl, A., Kennard, R., 1970. Ridge regression: biased estimation for nonorthogonal problems. Technometrics 12, 69–82.

Horvath, M., 2012. Computational accuracy and distributional analysis in models with incomplete markets and aggregate uncertainty. Economic Letters 117 (1), 276–279.

Jin, H., Judd, K.L., 2002. Perturbation methods for general dynamic stochastic models. Stanford University. Manuscript.

Jirnyi, A., Lepetyuk, V., 2011. A reinforcement learning approach to solving incomplete market models with aggregate uncertainty. IVIE Working Paper, Series AD 21.

Judd, K., 1992. Projection methods for solving aggregate growth models. Journal of Economic Theory 58, 410–452.

Judd, K., 1998. Numerical Methods in Economics. MIT Press, Cambridge, MA.

Judd, K., 2003. Perturbation methods with nonlinear changes of variables. Manuscript.

Judd, K., Guu, S., 1993. Perturbation solution methods for economic growth models. In: Varian, H. (Ed.), Economic and Financial Modeling with Mathematica, Springer Verlag, pp. 80–103.

Judd, K., Yeltekin, S., Conklin, J., 2003. Computing supergame equilibria. Econometrica 71 (1239), 1254.

Judd, K.L., Maliar, L., Maliar, S., 2009. Numerically stable stochastic simulation approaches for solving dynamic economic models. NBER Working Paper 15296.

Judd, K.L., Maliar, L., Maliar, S., 2010. A cluster-grid projection method: solving problems with high dimensionality. NBER Working Paper 15965.

Judd, K.L., Maliar, L., Maliar, S., 2011a. One-node quadrature beats Monte Carlo: a generalized stochastic simulation algorithm. NBER Working Paper 16708.

Judd, K.L., Maliar, L., Maliar, S., 2011b. Numerically stable and accurate stochastic simulation approaches for solving dynamic models. Quantitative Economics 2, 173–210.

Judd, K.L., Maliar, L., Maliar, S., 2011d. How to solve dynamic stochastic models computing expectations just once. NBER Working Paper 17418.

Judd, K.L., Maliar, L., Maliar, S., 2012a. Merging simulation and projection approaches to solve high-dimensional problems. NBER Working Paper 18501.

Judd, K.L., Renner, P., Schmedders, K., 2012b. Finding all pure-strategy equilibria in games with continuous strategies. Quantitative Economics 3, 289–331.

Judd, K.L., Maliar, L., Maliar, S., Valero, R., 2013. Smolyak method for solving dynamic economic models: Lagrange interpolation, anisotropic grid and adaptive domain. NBER 19326.

Juillard, M. 2011. Local approximation of DSGE models around the risky steady state. Wp.comunite 0087, Department of Communication, University of Teramo.

Juillard, M., Villemot, S., 2011. Multi-country real business cycle models: accuracy tests and testing bench. Journal of Economic Dynamics and Control 35, 178–185.

Kabourov, G., Manovskii, I., 2009. Occupational mobility and wage inequality. Review of Economic Studies 76 (2), 731–759.

Kendrik, D., Ruben Mercado, P., Amman, H.M., 2006. Computational Economics, Princeton University Press.

Kiefer, J., 1961. On large deviations of the empiric D.F. of vector change variables and a law of the iterated logarithm. Pacific Journal of Mathematics 11, 649–660.

Kim, J., Kim, S., Schaumburg, E., Sims, C.A., 2008. Calculating and using second-order accurate solutions of discrete time dynamic equilibrium models. Journal of Economic Dynamics and Control 32, 3397–3414.

Kim, S., Kollmann, R., Kim, J., 2010. Solving the incomplete market model with aggregate uncertainty using a perturbation method. Journal of Economics Dynamics and Control 34, 50–58.

Klenow, P. Kryvtsov, O. 2008. State-dependent or time-dependent pricing: does it matter for recent US inflation? Quarterly Journal of Economics CXXIII (3), 863–904.

Kocherlakota, N., Pistaferri, L., 2009. Asset pricing implications of Pareto optimality with private information. Journal of Political Economy 117 (3), 555–590.

Kollmann, R., Kim, S., Kim, J., 2011a. Solving the multi-country real business cycle model using a perturbation method. Journal of Economic Dynamics and Control 35, 203–206.

Kollmann, R., Maliar, S., Malin, B., Pichler, P., 2011b. Comparison of solutions to the multi-country real business cycle model. Journal of Economic Dynamics and Control 35, 186–202.

Krueger, D., Kubler, F., 2004. Computing equilibrium in OLG models with production. Journal of Economic Dynamics and Control 28, 1411–1436.

Krueger, D., Kubler, F., 2006. Pareto-improving social security reform when financial markets are incomplete? American Economic Review 96 (3), 737–755.

Krusell, P., Smith, A., 1998. Income and wealth heterogeneity in the macroeconomy. Journal of Political Economy 106, 868–896.

Krusell, P., Ohanian, L., Ríos-Rull, J.-V., Violante, G., 1997. Capital-skill complementarity and inequality: a macroeconomic analysis. Econometrica 68 (5), 1029–1053

Kubler, F., Schmedders, K., 2010. Tackling multiplicity of equilibria with Gröbner bases. Operations Research 58, 1037–1050.

Lim, G., McNelis, P., 2008. Computational Macroeconomics for the Open Economy. The MIT Press, Cambridge, Massachusetts, London, England.

Lombardo, G., 2010. On approximating DSGE models by series expansions. European Central Bank Working Paper 1264.

Maliar, L., 2013. Assessing gains from parallel computation on supercomputers. Manuscript.

Maliar, L., Maliar, S., 2001. Heterogeneity in capital and skills in a neoclassical stochastic growth model. Journal of Economic Dynamics and Control 25, 1367–1397.

Maliar, L., Maliar, S., 2003a. The representative consumer in the neoclassical growth model with idiosyncratic shocks. Review of Economic Dynamics 6, 362–380.

Maliar, L., Maliar, S. 2003b. Parameterized expectations algorithm and the moving bounds. Journal of Business and Economic Statistics 21, 88–92.

Maliar, L., Maliar, S., 2005a. Solving nonlinear stochastic growth models: iterating on value function by simulations. Economics Letters 87, 135–140.

Maliar, L., Maliar, S., 2005b. Parameterized expectations algorithm: how to solve for labor easily. Computational Economics 25, 269–274.

Maliar, L., Maliar, S. 2011. Perturbation with precomputation of integrals. Manuscript.

Maliar, L., Maliar, S. 2012a. Solving the multi-country real business cycle model using an envelope-condition method. Manuscript.

Maliar, L., Maliar, S., 2012b. Value function iteration for problems with high dimensionality: An envelope-condition method. Manuscript.

Maliar, L., Maliar, S., 2013. Envelope condition method versus endogenous grid method for solving dynamic programming problems. Economics Letters 120, 262–266.

Maliar, L., Maliar, S., Valli, F., 2010. Solving the incomplete markets model with aggregate uncertainty using the Krusell-Smith algorithm. Journal of Economic Dynamics and Control 34 (special issue), 42–49.

Maliar, S., Maliar, L., Judd, K.L., 2011. Solving the multi-country real business cycle model using ergodic set methods. Journal of Economic Dynamic and Control 35, 207–228.

Maliar, L., Maliar, S., Villemot, S., 2013. Taking perturbation to the accuracy frontier: a hybrid of local and global solutions. Computational Economics 42, 307–325.

Malin, B., Krueger, D., Kubler, F., 2011. Solving the multi-country real business cycle model using a Smolyak-collocation method. Journal of Economic Dynamics and Control 35, 229–239.

Manova, K., 2013. Credit constraints, heterogeneous firms, and international trade. Review of Economic Studies 80, 711–744.

Marcet, A., 1988. Solution of nonlinear models by parameterizing expectations. Carnegie Mellon University, Manuscript.

Marcet, A., Lorenzoni, G., 1999. The parameterized expectation approach: some practical issues. In: Marimon, R., Scott, A. (Eds.), Computational Methods for Study of Dynamic Economies. Oxford University Press, New York, pp. 143–171.

Marcet, A., Sargent, T., 1989. Convergence of least-squares learning in environments with hidden state variables and private information. Journal of Political Economy 97, 1306–1322.

Marimon, R., Scott, A., 1999. Computational Methods for Study of Dynamic Economies. Oxford University Press, New York.

MATLAB, 2011. MATLAB parallel computing toolbox. <http://www.mathworks.com/products/parallel-computing/description5.html>.

Mendoza, E.G., Bianchi, J., 2011. Overborrowing, financial crises and 'macro-prudential' policy? IMF Working Papers 11/24, International Monetary Fund.

Menzio, G., Telyukova, I., Visschers, L., 2012. Directed search over the life cycle. NBER Working Papers 17746.

Mertens, T., Judd, K.L., 2013. Equilibrium existence and approximation for incomplete market models with substantial heterogeneity. Manuscript.

Mertens, K., Ravn, M., 2011. Credit channels in a liquidity trap. CEPR Discussion Paper 8322.

Michelacci, C., Pijoan-Mas, J., 2012. Intertemporal labour supply with search frictions. Review of Economic Studies (2012) 79, 899–931.

Miranda, M., Fackler, P., 2002. Applied Computational Economics and Finance. MIT Press, Cambridge.

Miranda, M., Helmberger, P., 1988. The effects of commodity price stabilization programs. American Economic Review 78, 46–58.

Moore, G.E., 1965. Cramming more components onto integrated circuits. Electronics 38 (8), 1965.

Morozov, S., Mathur, S., 2012. Massively parallel computation using graphics processors with application to optimal experimentation in dynamic control. Computational Economics 40, 151–182.

Nadaraya, E.A., 1964. On Estimating Regression. Theory of probability and its applications 10, 186–190.

Nagurney, A., 1996. Parallel computation. In: Amman, H.M., Kendrick, D.A., Rust, J. (Eds.), Handbook of Computational Economics, vol 1. Elsevier, Amsterdam, pp. 336–401.

Nagurney, A., Zhang, D., 1998. A massively parallel implementation of discrete-time algorithm for the computation of dynamic elastic demand and traffic problems modeled as projected dynamical systems. Journal of Economic Dynamics and Control 22 (8–9), 1467–1485.

Nakajima, M., Telyukova, I., 2011. Reverse mortgage loans: a quantitative analysis. Manuscript.

Niederreiter, H., 1992. Random Number Generation and Quasi-Monte Carlo Methods. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania.

Pagan, Ullah, 1999. Nonparametric Econometrics. Cambridge University Press, New York.

Pakes, A., McGuire, P., 2001. Stochastic algorithms, symmetric Markov perfect equilibria, and the 'curse' of dimensionality. Econometrica 69, 1261–1281.

Peralta-Alva, A., Santos, M., 2005. Accuracy of simulations for stochastic dynamic models. Econometrica 73, 1939–1976.

Piazzesi, M., Schneider, M. 2012. Inflation and the price of real assets. Staff Report 423, Federal Reserve Bank of Minneapolis.

Pichler, P., 2011. Solving the multi-country real business cycle model using a monomial rule Galerkin method. Journal of Economic Dynamics and Control 35, 240–251.

Powell, W., 2011. Approximate Dynamic Programming. Wiley, Hoboken, New Jersey.

Ravenna, F., Walsh, C., 2011. Welfare-based optimal monetary policy with unemployment and sticky prices: a linear-quadratic framework. American Economic Journal: Macroeconomics 3, 130–162.

Reiter, M., 2009. Solving heterogeneous-agent models by projection and perturbation. Journal of Economic Dynamics and Control 33 (3), 649–665.

Rios-Rull, J.V., 1997. Computing of equilibria in heterogeneous agent models. Federal Reserve Bank of Minneapolis Staff. Report 231.

Rust, J., 1996. Numerical dynamic programming in economics. In: Amman, H., Kendrick, D., Rust, J. (Eds.), Handbook of Computational Economics. Elsevier Science, Amsterdam, pp. 619–722.

Rust, J., 1997. Using randomization to break the curse of dimensionality. Econometrica 65, 487–516.

Rust, J. 2008., Dynamic programming. In: Durlauf, S., Blume, L. (Eds.), The New Palgrave Dictionary of Economics. Palgrave Macmillan.

Sanders, J., Kandrot, E., 2010. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, Upper Saddle River, NJ.

Santos, M., 1999. Numerical solution of dynamic economic models. In: Taylor, J., Woodford, M. (Eds.), Handbook of Macroeconomics. Elsevier Science, Amsterdam, pp. 312–382.

Santos, M., 2000. Accuracy of numerical solutions using the Euler equation residuals. Econometrica 68, 1377–1402.

Schmitt-Grohé, S., Uribe, M., 2004. Solving dynamic general equilibrium models using a second-order approximation to the policy function. Journal of Economic Dynamics and Control 28 (4), 755–775.

Scott, D., Sain, S., 2005. Multidimensional density estimation. In: Rao, C., Wegman, E., Solka, J. (Eds.), Handbook of Statistics, vol 24. Elsevier B. V., Amsterdam, pp. 229–261.

Sims, C., Waggoner, D., Zha, T., 2008. Methods for inference in large-scale multiple equation Markov-switching models. Journal of Econometrics 142 (2), 255–274.

Smets, F., Wouters, R., 2003. An estimated dynamic stochastic general equilibrium model of the Euro area. Journal of the European Economic Association 1 (5), 1123–1175.

Smets, F., Wouters, R., 2007. Shocks and frictions in US business cycles: a Bayesian DSGE approach. American Economic Review 97 (3), 586–606.

Smith, A., 1991. Solving stochastic dynamic programming problems using rules of thumb. Queen's University. Economics Department, Discussion Paper 816.

Smith, A., 1993. Estimating nonlinear time-series models using simulated vector autoregressions. Journal of Applied Econometrics 8, S63–S84.

Smolyak, S., 1963. Quadrature and interpolation formulas for tensor products of certain classes of functions. Soviet Mathematics, Doklady 4, 240–243.

Song, Z., Storesletten, K., Zilibotti, F., 2012. Rotten parents and disciplined children: a politico-economic theory of public expenditure and debt. Econometrica 80 (6), 2785–2803.

Stachursky, J., 2009. Economic Dynamics: Theory and Computation. MIT Press, Cambridge.

Stroud, A., 1971. Approximate Integration of Multiple Integrals. Prentice Hall, Englewood Cliffs, New Jersey.

Su, C.L., Judd, K.L., 2012. Constrained optimization approaches to estimation of structural models. Econometrica 80 (5), 2213–2230.

Swanson, E., Anderson, G., Levin, A., 2002. Higher-order perturbation solutions to dynamic, discrete-time rational expectations models. Manuscript.

Swanson, E., Anderson, G., Levin, A., 2006. Higher-order perturbation solutions to dynamic, discrete-time rational expectations models. Federal Reserve Bank of San Francisco working paper 1.

Tauchen, G., 1986. Finite state Markov chain approximations to univariate and vector autoregressions. Economic Letters 20, 177–181.

Tauchen, G., Hussey, R., 1991. Quadrature-based methods for obtaining approximate solutions to nonlinear asset pricing models. Econometrica 59, 371–396.

Taylor, J., Uhlig, H., 1990. Solving nonlinear stochastic growth models: a comparison of alternative solution methods. Journal of Business and Economic Statistics 8, 1–17.

Temlyakov, V., 2011. Greedy Approximation. Cambridge University Press, Cambridge.

Tesfatsion, L., Judd, K.L., 2006. Handbook of computational economics. In: Agent-Based Computational Economics, Vol 2. Elsevier Science, Amsterdam.

Tsitsiklis, J., 1994. Asynchronous stochastic approximation and Q-Learning. Machine Learning 16, 185–202.

Valero, R., Maliar, L., Maliar, S., 2013. Parallel speedup or parallel slowdown: is parallel computation useful for solving large-scale dynamic economic models? Manuscript.

Villemot, S. 2012. Accelerating the resolution of sovereign debt models using an endogenous grid method. Dynare working paper 17, <http://www.dynare.org/wp>.

Watson, G.S. 1964. Smooth regression analysis. Shankya Series A 26, 359–372.

Weintraub, G., Benkard, C.L., Roy, B.V., 2008. Markov perfect industry dynamics with many firms. Econometrica 2008, 1375–1411.

Winschel, V., Krätzig, M., 2010. Solving, estimating and selecting nonlinear dynamic models without the curse of dimensionality. Econometrica 78 (2), 803–821.

Woodford, M. 2011. Optimal monetary stabilization policy, in: Friedman, B.M., Woodford, M. (Eds.), Handbook of Monetary Economics, vol. 3B. Elsevier, Amsterdam, pp. 723–828.

Wright, B., Williams, J., 1984. The welfare effects of the introduction of storage. Quarterly Journal of Economics 99, 169–192.

Young, E., 2010. Solving the incomplete markets model with aggregate uncertainty using the Krusell-Smith algorithm and non-stochastic simulations. Journal of Economic Dynamics and Control 34, 36–41.