Journal of Economic Dynamics & Control 44 (2014) 92–123

Contents lists available at ScienceDirect

Journal of Economic Dynamics & Control

journal homepage: www.elsevier.com/locate/jedc

Smolyak method for solving dynamic economic models: Lagrange interpolation, anisotropic grid and adaptive domain

Kenneth L. Judd[a], Lilia Maliar[b,c,*], Serguei Maliar[c,d], Rafael Valero[c]

[a] Hoover Institution, 434 Galvez Mall, Stanford University, Stanford, CA 94305-6010, USA
[b] Department of Economics, 579 Serra Mall, Stanford University, Stanford, CA 94305-6072, USA
[c] Department of Fundamentos del Análisis Económico, University of Alicante, Campus de San Vicente, 03080 Alicante, Spain
[d] Leavey School of Business, Lucas Hall 124, Santa Clara University, Santa Clara, CA, 95053, USA

ARTICLE INFO

Article history:
Received 17 August 2013
Received in revised form
8 March 2014
Accepted 9 March 2014
Available online 3 April 2014

JEL classification:
C63
C68

Keywords:
Smolyak method
Sparse grid
Adaptive domain
Projection
Anisotropic grid
High-dimensional problem

ABSTRACT

We show how to enhance the performance of a Smolyak method for solving dynamic economic models. First, we propose a more efficient implementation of the Smolyak method for interpolation, namely, we show how to avoid costly evaluations of repeated basis functions in the conventional Smolyak formula. Second, we extend the Smolyak method to include anisotropic constructions that allow us to target higher quality of approximation in some dimensions than in others. Third, we show how to effectively adapt the Smolyak hypercube to a solution domain of a given economic model. Finally, we argue that in large-scale economic applications, a solution algorithm based on Smolyak interpolation has substantially lower expense when it uses derivative-free fixed-point iteration instead of standard time iteration. In the context of one- and multi-agent optimal growth models, we find that the proposed modifications to the conventional Smolyak method lead to substantial increases in accuracy and speed.

1. Introduction

In a seminal paper, Smolyak (1963) proposed a sparse-grid method that allows to efficiently represent, integrate and interpolate functions on multidimensional hypercubes. The Smolyak method is not subject to the curse of dimensionality and can be used to solve large-scale applications. A pioneering work of Krueger and Kubler (2004) introduced the Smolyak method to economics in the context of a projection-style iterative method for solving multi-period overlapping generation models. Smolyak methods have been also used to solve portfolio-choice problems (Gavilan-Gonzalez and Rojas, 2009); to develop state-space filters tractable in large-scale problems (Winschel and Krätzig, 2010); to solve models with infinitely lived heterogenous agents (Malin et al., 2011; Gordon, 2011; Brumm and Scheidegger, 2013); and to solve new Keynesian models (Fernández-Villaverde et al., 2012).

While the Smolyak method enables us to study far larger problems than do tensor-product methods, its computational expense still grows rapidly with the dimensionality of the problem. In particular, Krueger and Kubler (2004) and Malin et al. (2011) document a high computational cost of their solution methods when the number of state variables exceeds twenty.

* Corresponding author at: Department of Economics, 579 Serra Mall, Stanford University, Stanford, CA 94305-6072, USA. Tel.: +1 6507259069.
E-mail address: maliarl@stanford.edu (L. Maliar).

http://dx.doi.org/10.1016/j.jedc.2014.03.003
0165-1889/© 2014 Elsevier B.V. All rights reserved.

In the paper, we show a more efficient implementation of the Smolyak method that reduces its computational expense, and we propose extensions of the Smolyak method that enable us to more effectively solve dynamic economic models.[1]

First, the conventional Smolyak formula is inefficient. To interpolate a function in a given point, it first causes the computer to create and evaluate a long list of repeated basis functions, and it then constructs linear combinations of such functions to get rid off repetitions. In high-dimensional problems, the number of repetitions is large and slows down computations dramatically. We offer a way to avoid costly evaluations of the repeated basis functions: Instead of conventional nested-set generators, we introduce disjoint-set generators. Nested sets include one another, and as a result, their tensor products contain repeated elements but our disjoint sets do not. This is why our implementation of the Smolyak formula does not have repetitions.[2]

An efficient implementation of the Smolyak method is especially important in the context of numerical methods for solving dynamic economic models which require us to interpolate decision and value functions a very large number of times, e.g., in each grid point, integration node or time period. We save on cost every time when we perform an evaluation of the Smolyak interpolant.

To compute the interpolation coefficients, we use a universal Lagrange interpolation technique instead of the conventional closed-form expressions. Namely, we proceed in three steps: (i) construct M Smolyak grid points; (ii) construct M corresponding Smolyak basis functions; and (iii) interpolate the values of the true function at the grid points using the basis functions. We then solve a system of M linear equations in M unknowns. The cost of solving this system can be high but it is a fixed cost in the context of iterative methods for solving dynamic economic models. Namely, we argue that an expensive inverse in the Lagrange inverse problem can be precomputed up-front (as it does not change along iterations). Furthermore, to ensure numerical stability of a solution to the Lagrange inverse problem, we use families of orthogonal basis functions, such as a Chebyshev family.

Second, the conventional Smolyak formula is symmetric in a sense that it has the same number of grid points and basis functions for all variables. To increase the quality of approximation, one must equally increase the number of grid points and basis functions for all variables, which may be costly or even infeasible in large-scale applications. In the paper, we present an anisotropic version of the Smolyak method that allows for asymmetric treatments of variables, namely, it enables us to separately choose the accuracy level for each dimension with the aim of increasing the quality of approximation. In economic applications, variables do not enter symmetrically: decision or value functions may have more curvature in some variables than in others; some variables may have larger ranges of values than others; and finally, some variables may be more important than the others. For example, in heterogeneous-agent economies, an agent's decision functions may depend more on her own capital stock than on the other agents' capital stocks (e.g., Kollmann et al., 2011); also, we may need more grid points for accurate approximation of endogenous than exogenous state variables (e.g., models based on Tauchen and Hussy's, 1991 approximation of shocks). An anisotropic version of the Smolyak method allows us to take into account a specific structure of decision or value functions to solve economic models more efficiently.

Third, the Smolyak method constructs grid points within a normalized multidimensional hypercube. In economic applications, we must also specify how the model's state variables are mapped into the Smolyak hypercube. The way in which this mapping is constructed can dramatically affect the effective size of a solution domain, and hence, the quality of approximation. In the paper, we show how to effectively adapt the Smolyak grid to a solution domain of a given economic model. We specifically construct a parallelotope that encloses a high-probability area of the state space of the given model, and we reduce the size of the parallelotope to minimum by reorienting it with a principle-component transformation of state variables. Judd et al. (2011) find that solution algorithms that focus on “a relevant domain” are more accurate (in that domain) than solution algorithms that focus on a larger (and partially irrelevant) domain and that face therefore a trade off between the fit inside and outside the relevant domain. For the same reason, an adaptive domain increases the accuracy of the Smolyak method.

Finally, the Smolyak method for interpolation is just one ingredient of a numerical method for solving dynamic economic models. In particular, Krueger and Kubler (2004) and Malin et al. (2011) complemented Smolyak interpolation with other computational techniques that are tractable in large-scale problems, such as Chebyshev polynomials, monomial integration and a learning-style procedure for finding polynomial coefficients. Nonetheless, there is one technique – time iteration – that is expensive in their version of their numerical procedure. Time iteration is traditionally used in dynamic programming: given functional forms for future value function, it solves for current value function using a numerical solver. It works similarly in the context of the Euler equation methods: given functional forms for future decision functions, it solves for current decision functions using a numerical solver. However, there is a simple derivative-free alternative to time iteration – fixed point iteration – that can solve large systems of equations using only straightforward calculations, avoiding thus the need of a numerical solver. Fixed-point iteration is commonly used in the context of solution methods for dynamic economic models; see, e.g., Wright and Williams (1984), Miranda and Helmberger (1988), Marcet (1988), Den Haan (1990), Marcet and Lorenzoni (1999), Gaspar and Judd (1997), Judd et al. (2010, 2011), Maliar and Maliar (2014b). We assess how the cost of a Smolyak-based projection method for solving dynamic economic models depends on a specific iterative scheme used. We find that time iteration may dominate fixed-point iteration in small-scale applications but the situation reverses

[1] We provide a MATLAB code that implements the computational techniques developed in the paper. Also, Coleman and Lyon (2013) provide Python and Julia codes that implement some of these techniques.

[2] Our exposition was informed by our personal communication with Sergey Smolyak.