Standardized projection algorithms to solve dynamic economic models

Sijmen Duineveld*

December 3, 2021

Abstract

We evaluate the performance of several projection algorithms with three Dynamic (Stochastic) General Equilibrium (DSGE) models. The algorithms are standardized and implemented in a projection method toolbox. We use cubic splines, complete Chebyshev polynomials, and Smolyak-Chebyshev polynomials as basis functions. For a small scale, and near-linear model complete Chebyshev polynomials with Galerkin's method perform best. For more complex models or less linear models cubic splines perform better in terms of accuracy and speed than global polynomials.

The first DSGE model that we use to evaluate the algorithms is a standard RBC model, which is near-linear and has two state variables. For such a simple model a maximum error of 10^{-6} can be achieved in less than 0.05 seconds with all three basis functions. The second model is an RBC model with habits in consumption and investment

*I thank Daniel Fehrle, Alfred Maussner, and Christopher Heiberger for our discussions on projection methods, and I thank Alfred Maussner, Rafa Valero and Joris de Wind for providing their codes on projection methods.
Email: s.a.duineveld@outlook.com
Declarations of interest: none

adjustment costs, and has four state variables and two policy variables. For this model splines perform best and they achieve a maximum error of 10^{-5} in about 5 seconds. The third model is a highly non-linear limit cycle model, which is also best solved with a spline.

1 Introduction

Dynamic Stochastic General Equilibrium (DSGE) models are at the core of modern macroeconomics. These models are usually solved with perturbation techniques. The main reason for the popularity of perturbation methods is the availability of software packages such as Dynare. In addition, perturbation techniques are fast and deemed accurate enough for most applications. The disadvantage of perturbation methods is that they only provide a local solution (Fernández-Villaverde et al., 2016), which may result in large errors further away from the steady state. Projection methods on the other hand are rarely used. The main reason is that they are perceived to be hard to code (Fernández-Villaverde et al., 2016).

The first contribution of this paper is to describe standardized projection algorithms as implemented in the Promes Toolbox[1], which to the best of our knowledge is the only toolbox that can solve dynamic economic models with projection methods. This toolbox only requires the user to code a model file that computes the Euler residuals, set the interval where the policy function should be approximated, select an algorithm, and supply an initial guess for the policy function. Based on these inputs the toolbox constructs the appropriate grid, and solves the policy function using the selected algorithm. The toolbox can also be used to evaluate the policy function given the state variables.

The literature on projection methods describes several algorithms. The papers by Judd (1992), and Gaspar and Judd (1997) use Chebyshev polyno-

[1]The Promes Toolbox is written for Matlab. The most recent version is available at www.promestoolbox.com.

mials, while McGrattan (1996) uses Finite Elements. The papers by Aruoba et al. (2006) and Fernández-Villaverde et al. (2016) discuss both polynomials and Finite Elements. For small, near-linear models projection yields very accurate solutions, especially when the exogenous variable is discretized. Relative to global polynomials Finite Elements are better at capturing local behavior (Fernández-Villaverde et al., 2016). For this reason they are more accurate, especially for highly non-linear models. For example, Finite Elements can be customized to increase the number of gridpoints where the accuracy is low. Finite Elements are harder to implement though, and are in general slower than polynomials.

To solve high dimensional systems Smolyak's algorithm is powerful, because it uses a sparse grid, and sparse polynomials. This sparseness is very effective at addressing the curse of the dimensionality, which puts a bound on more traditional projection algorithms. Applications of this algorithm in economic models were first described in the papers by Krueger and Kubler (2004) and Malin et al. (2011). Examples with many dimensions are Fernández-Villaverde et al. (2015) who use a model with 5 state variables, Fernández-Villaverde and Levintal (2018) who use a model with 12 state variables, and Malin et al. (2011) who use a model with 20 continuous state variables.

The economic literature is relatively silent on the use of splines, or piecewise polynomials, in combination with projection methods. Two exceptions that discuss splines in economic models are Judd et al. (2000) and Judd et al. (2003). These authors use cubic splines to solve a heterogeneous agent model with a borrowing constraint.

Most of the literature on projection methods describes applications to relatively standard models as in Judd (1992), McGrattan (1996), Gaspar and Judd (1997), Aruoba et al. (2006) and Fernández-Villaverde et al. (2016). Some papers describe the application of projection methods to non-standard models. Fernández-Villaverde et al. (2015) analyze a model with a Zero Lower Bound, and five state variables, which is solved with the Smolyak

algorithm, and results in errors between 10^{-3} and 10^{-5}. Caldara et al. (2012) and Fernández-Villaverde and Levintal (2018) analyze a rare disaster model. McGrattan (1996) discusses a model with an inequality constraint, and solves this with a penalty function.

The second contribution of our paper is to compare the performance of splines with global (Chebyshev) polynomials in several models with continuously differentiable policy functions. To be more precise we evaluate three types of basis functions. The first are cubic splines, the second complete Chebyshev polynomials, and the third sparse Chebyshev polynomials as used in Smolyak's algorithm described by Judd et al. (2014)[2].

We evaluate the performance in terms of accuracy and computation time[3] for three models. The first model is a Standard RBC model with two continuous state variables and one policy variable. This model is near-linear and shows that projection methods can be both fast and accurate for such a simple model. For each of the three basis functions we obtain errors smaller than 10^{-6} is less than 0.05 seconds. Especially Galerkin with Chebyshev polynomials performs very well in terms of speed and accuracy. This method achieves a maximum error smaller than 10^{-13} in less than 0.3 seconds.

Next we analyze the performance of the algorithms in an RBC model with Habits in consumption and Investment Adjustment costs (HIA model). This model has four continuous state variables, and two policy variables. For this model a third order perturbation solution results in an error of 10^{-2}. Projection methods can achieve a similar accuracy level in about 1 second. When a higher accuracy level is required splines outperform global polynomials.

The third contribution of this paper is the application of projection algo-

[2]The Promes toolbox uses the code provided by these authors to implement the Smolyak algorithm, see Footnote 19.

[3]All reported computation times are on a PC with Matlab 2019a and a Ryzen 2700x processor, and without any parallel computing. We do not use parallel computing, because we would use parallel computing in the outer most loop during a calibration procedure. The computation time of each iteration is then similar to the reported time.

rithms to a model featuring an attracting limit cycle, which is the third and last model we analyze. Limit cycle models have regained interest in macroeconomics (Beaudry et al., 2020), but little is known about solving these models. Galizia (2021) shows that models with an attracting limit cycle can be solved with perturbation methods, although most perturbation software packages have not yet implemented this method[4]. The model equations in Beaudry et al. (2020) are restricted to have at most cubic terms, which will be approximated well with a third order perturbation solution. Instead we choose a highly non-linear example described in Duineveld (2021), which is an RBC model with external habits in the labor supply. To illustrate the performance we use the deterministic version of the model[5], which has two continuous state variables.

For this model splines perform better than global polynomials. The third order perturbation approximation is very inaccurate, and simulations are completely out of phase after just one cycle. Smolyak's algorithm does not converge to the solution, even when we start from a highly accurate solution. The reason is that Smolyak's algorithm does not preserve the shape of the policy function sufficiently.

The paper is organized as follows. Section 2 describes the projection algorithms. Each section thereafter discusses a model where projection methods are applied to. The performance measures for each model consist of the accuracy and computation time. Section 3 analyzes the performance of the algorithms for a Standard RBC model. In Section 4 an RBC model with Habits in consumption and Investment Adjustment is used to compare the algorithms. Section 5 analyzes the performance of the algorithms for a limit cycle model with external habits in the labor supply (Duineveld, 2021). Fi-

[4]The only known exception is the CSD toolbox, which can solve limit cycle models with perturbation methods up to an order three approximation. See https://www.saduineveld.com/tools.

[5]In Duineveld (2021) the model includes stochastic Total Factor Productivity, and has therefore one state variable more.

nally we draw a conclusion in Section 6.

2 Projection algorithms

The goal of each of the algorithms is to numerically approximate a policy function that solves a recursive dynamic optimization problem. A policy function gives the control (or policy) variable Y as a function of the state variables x. If the exact policy function is $Y(x)$ the algorithm approximates this function with $\hat{Y}(x;\theta)$ where θ is a vector of parameters of the basis function, either a spline or a polynomial in this paper.

The objective of projection methods is to find the policy function that solves the dynamic optimization problem. The algorithms require a function that computes the Euler residuals for a given approximation of the policy function. In practice the residual function $R(x;\theta)$ is a function with the model equations, which is best explained with a simple example.

Assume households face the dynamic problem:

$$\begin{aligned} &\max \sum_{t=1}^{\infty} \beta^{t-1} \log(C_t) \\ &\text{s.t. } K_{t+1} + C_t = K_t^{\alpha} \end{aligned}$$

Taking the First Order Conditions we derive the Euler equation:

$$C_t^{-1} = \beta C_{t+1}^{-1} \alpha K_{t+1}^{\alpha-1} \tag{1}$$

We choose consumption C_t as the policy variable, which is a function of the state variable capital K_t. There exists a policy function $C_t = C(K_t)$, which exactly solves this dynamic optimization problem. Instead we numerically approximate the policy function with $\hat{C}(K_t;\theta)$. The approximation

$\hat{C}(K_t;\theta)$ will not exactly solve this dynamic system, and we need to compute the errors with a residual function.

Given the approximation of consumption we can compute next period's variables:

$$\hat{K}_{t+1} = \hat{K}(K_t;\theta) = K_t^{\alpha} - \hat{C}(K_t;\theta) \tag{2}$$

$$\hat{C}_{t+1} = \hat{C}\left(\hat{K}_{t+1};\theta\right) \tag{3}$$

The residuals in the Euler equation (1) as a result of the approximation are:

$$R(K_t;\theta) = \beta\hat{C}\left(\hat{K}(K_t;\theta);\theta\right)^{-1}\alpha\hat{K}(K_t;\theta)^{\alpha-1} - \hat{C}(K_t;\theta)^{-1} \tag{4}$$

The objective is to find the policy function that minimizes the residuals on a specified interval of the state variables. This interval is determined by a lower and upper bound of each state variable. Within this interval discrete points, or gridpoints, are chosen, where we evaluate the Euler residuals.

2.1 Projection Algorithms

Projection method algorithms consist of three main choices[6]. The first choice are the basis functions used for the approximation of the policy function. The most popular choices are Chebyshev polynomials, Finite Elements and Splines. The second choice is the projection condition of which Galerkin, Collocation, and Minimization of the Squared Errors are the most widely known. Galerkin's method sets the basis functions such that they are orthogonal to the errors, similar to the Method of Moments (Judd, 1998). Collocation solves the model at the gridpoints.

[6]Gaspar and Judd (1997, Table 3) mention a fourth choice, which is the integration method to compute the expected value of future states of the economy.

The third choice is the method to solve the objective function. For our algorithms we use two solution methods. The first is a Newton-type of non-linear equation solver[7]. When a Newton-type of solver is used to solve the model at the gridpoints we call this Direct Computation. The second option is Time Iteration, which is an iterative scheme described in Subsection 2.3.

We do not describe the Fixed Point algorithm (Miranda and Helmberger, 1988) to solve the objective function as advocated by Gaspar and Judd (1997). This method is known to perform well, because it is is derivative free. The method is not included, because the format of the Promes Toolbox requires a residual function that computes the Euler residuals. This contrasts with the Fixed Point algorithm, which requires a model function that returns the policy variables as output (Gaspar and Judd, 1997).

From all the possible combination of basis functions, projection conditions and solution methods we have chosen the combinations which perform best. In total we evaluate three types of basis functions, each with two different algorithms. The first type of basis functions are splines, which are determined with the collocation projection condition. The solution at gridpoints is obtained using either Direct Computation or Time Iteration. A spline is fitted through the solution at the gridpoints.

The second type of basis functions are complete Chebyshev polynomials, which is used for two algorithms. One algorithm is based on Galerkin's projection condition, where the coefficients are obtained using a Newton-type of solver. The algorithm sets each coefficient of the polynomial such that each polynomial term is orthogonal to the errors of the residual function. The other algorithm with complete Chebyshev polynomials uses Time Iteration to obtain the solution at the gridpoints. The coefficients of the polynomial are determined by Minimization of the Squared Errors at the gridpoints[8].

The third basis function is based on Smolyak's algorithm. The Smolyak

[7] We use Matlab's `fsolve`.

[8] See Footnote 18.

approximation uses a sparse Chebyshev polynomial in combination with a sparse grid. The projection condition is collocation. As with splines the solution at the gridpoints is obtained with either Direct Computation or Time Iteration.

Galerkin's method with Chebyshev polynomials is fast and accurate for small scale near-linear models. For more complex models or less linear models splines perform better in terms of accuracy and speed. Splines are especially robust in combination with Time Iteration. Time Iteration is the only method that should theoretically converge to the saddle path stable solution, although this requires that the shape of the policy function is preserved (Judd, 1998). Cubic splines will preserve the shape of the policy function better than global polynomials. Or as De Boor (1978) puts it: "... the essential limitation of polynomial approximation: If the function to be approximated is badly behaved anywhere in the interval of approximation, then the approximation is poor everywhere. This global dependence on local properties can be avoided when using piecewise polynomial approximants."

Smolyak's algorithm is primarily recommended for models with a high number of state variables, because it is the most effective method to address the curse of the dimensionality. The reason is that the number of gridpoints grows only polynomially in the number of state variables, while the number of gridpoints grows exponentially for the other algorithms.

Some caution is warranted because Smolyak's algorithms might approximate the shape of a policy function poorly, which can result in convergence problems. For example, in the Limit Cycle model of Section 5 the Smolyak algorithms diverges even from a very accurate solution, because it fails to preserve the shape sufficiently.

2.2 Defining a grid

To solve the policy function the residuals are calculated on an interval of the n state variables where we want the approximation to be good. The intervals

are defined by the lower and upper bounds $[\underline{x}^i, \overline{x}^i]$ for $i = [1, \ldots, n]$.

For all methods except Smolyak's algorithm[9] we use a Cartesian product to construct the grid. For each state variable i a set of q^i nodes $X^i = \left\{x_1^i, x_2^i, \ldots, x_{q^i}^i\right\}$ are defined on the interval between the lower and upper bound. For splines we use equidistant nodes, and for complete Chebyshev polynomials we use the Chebyshev nodes. We call the Cartesian product of these sets the initial grid:

$$\mathcal{X} = X^1 \times \ldots \times X^n \tag{5}$$

This set consists of $m = \prod_{i=1}^{n} q^i$ points where the residual function is evaluated.

.

2.3 Solving a model at the gridpoints

For the collocation and Minimization of Squared Errors projection condition we solve the model at the gridpoints with either Direct Computation or Time Iteration. Direct Computation uses a non-linear equation solver based on a Newton-type of algorithm. Time Iteration is a method specifically designed to solve recursive dynamic optimization problems. With Time Iteration we solve for period's t choices, holding the policy function in period $t+1$ constant. This makes each iteration computationally less intensive, but also less effective than with Direct Computation.

For both Direct Computation and Time Iteration the solution at the gridpoints uniquely defines the coefficients θ of the basis function, either a spline or a polynomial. Assume the solution is $\tilde{Y}$ at the gridpoints $x \in \mathcal{X}$. The coefficients of the basis function θ can be determined by some function:

$$\theta = \Omega\left(x, \tilde{Y}\right) \tag{6}$$

[9]See Subsection 2.6 for the construction of the grid with Smolyak's algorithm.

For collocation we can either (a) solve the policy variable at the gridpoints, and determine the coefficients with (6), or by (b) directly set the coefficients θ such that the model is solved at the gridpoints. With Time Iteration we use option (a), because that ensures that the Jacobian matrix of the system of equations is sparse. With splines in combination with Direct Computation we use (a), because the number of gridpoints is smaller than the number of coefficients, and results in a smaller Jacobian matrix. With Smolyak's algorithm in combination with Direct Computation we choose (b). We discuss both Direct Computation and Time Iteration using the simple example described earlier.

Direct Computation For Direct Computation the residuals can be written as a function of either the coefficients θ or the choice variable $\tilde{C}$ at the gridpoints. In equation (4) the residuals were a function of the coefficients θ. Using (6) the alternative formulation of (4) in terms of choice variable $\tilde{C}$ is:

$$R\left(K_t, \tilde{C}_t\right) = R\left(K_t; \Omega\left(K_t, \tilde{C}_t\right)\right)$$

This is a system of m equations, where m is the total number of gridpoints[10]. Note that a change in $\tilde{C}_{i,t}$ at gridpoint i affects the coefficients θ, which also affects the solution at other gridpoints $j \neq i$.

The system of equations can be solved with a Newton-type of non-linear equation solver[11]. This requires the numerical approximation of the $m \times m$ Jacobian. As the dense Jacobian has m^2 elements each iteration is computationally expensive for a large number of gridpoints m. As Direct Computation is basically a Newton-type of solver it will converge at least quadratically close to the solution (Judd, 1998)

[10] See equation (5).

[11] We use Matlab's `fsolve` with the 'trust-region-dogleg' algorithm.

Time Iteration The Time Iteration algorithm is described by Judd (1998). Compared to Direct Computation it economizes on the iteration step. In iteration j the algorithm solves for period t choices $\tilde{C}_t^{\,j}$, while using the coefficients of the previous iteration θ^{j-1} for period $t+1$ choices. For the simple example we replace (2) and (3) with:

$$\begin{aligned}\hat{K}_{t+1} &= K_t^{\alpha} - \tilde{C}_t^j = \hat{K}\left(K_t, \tilde{C}_t^j\right) \\ \hat{C}_{t+1} &= \hat{C}\left(\hat{K}_{t+1}; \theta^{j-1}\right) \\ &= \hat{C}\left(\hat{K}\left(K_t, \tilde{C}_t^j\right); \theta^{j-1}\right)\end{aligned}$$

This gives us the residual function:

$$R\left(K_t, \tilde{C}_t^j; \theta^{j-1}\right) = \beta \hat{C}\left(\hat{K}\left(K_t, \tilde{C}_t^j\right); \theta^{j-1}\right)^{-1} \alpha \hat{K}\left(K_t, \tilde{C}_t^j\right)^{\alpha-1} - \left(\tilde{C}_t^j\right)^{-1} \quad (7)$$

Equation (7) defines a system of m non-linear equations in as many unknowns $\tilde{C}_t^j$. We solve this system of equations with a Newton type of algorithm[12]. As the residual at gridpoint i only depends on $\tilde{C}_{i,t}^j$ the Jacobian matrix is sparse with entries on the diagonal only[13]. In addition, we do not have to recompute the spline or polynomial when numerically approximating the Jacobian. This makes the algorithm efficient for recursive dynamic problems. After solving the system of equations (7) for the policy variable $\tilde{C}_t^j$ we update the coefficients θ using (6), and repeat the process until convergence.

We use two stopping criteria, which both have to be satisfied. The first criterion is the maximum absolute difference in the policy variable between iterations, which has to satisfy $\max\left|\tilde{Y}^j - \tilde{Y}^{j-1}\right| \leq \epsilon^d$. The second criterion is the maximum Euler residual at the gridpoints when using the updated

[12]Matlab's `fsolve` with the 'trust-region' algorithm for square problems. This algorithm allows the use of a sparse Jacobian.

[13]With multiple policy functions the Jacobian consists of repeated blocks of diagonal matrices.

policy θ^j for both current and next period's policy. Formally this stopping criterion is $\max \left| R\left(x, \tilde{Y}^j; \theta^j\right)\right| \leq \epsilon^r$, where x are the state variables.

For consistency with the other methods we use an alternative notation for (7) in the examples of the next sections. The alternative notation uses the coefficients θ to define the policy functions $\hat{C}\left(K_t; \theta^j\right) = \tilde{C}_t^j$, and $\hat{K}\left(K_t; \theta^j\right) = \hat{K}\left(K_t, \tilde{C}_t^j\right)$. This results in:

$$R\left(K_t; \theta\right) = \beta \hat{C}\left(\hat{K}\left(K_t; \theta^j\right); \theta^{j-1}\right)^{-1} \alpha \hat{K}\left(K_t; \theta^j\right)^{\alpha-1} - \hat{C}\left(K_t; \theta^j\right)^{-1}$$

instead of (7).

2.4 Splines

Spline basis functions are best used in combination with collocation. The policy variable(s) $\tilde{Y}$ at each gridpoint $x \in \mathcal{X}$ is obtained with either Direct Computation or Time Iteration. The solution at the gridpoints determines the parameters θ of a piece-wise cubic polynomial, or spline as in (6). For simplicity we use a grid with equidistant nodes.

To determine the coefficients we use not-a-knot end conditions, which results in a twice differentiable spline[14]. When such a cubic spline is used to approximate a four times differentiable function the convergence is $\mathcal{O}\left(q^{-4}\right)$, where q is the number of nodes per dimension (De Boor, 1978).

For the univariate case with q data points $(x_1, y_1), \ldots, (x_q, y_q)$ a cubic spline takes the piece-wise form:

[14]Matlab's `griddedInterpolant` with the method set to 'spline'.

$$
\begin{aligned}
S_1(x) &= y_1 + \theta_{1,1}\Delta x_1 + \theta_{1,2}\Delta x_1^2 + \theta_{1,3}\Delta x_1^3 \quad \text{for } x \in [x_1, x_2] \\
S_2(x) &= y_2 + \theta_{2,1}\Delta x_2 + \theta_{2,2}\Delta x_2^2 + \theta_{2,3}\Delta x_2^3 \quad \text{for } x \in [x_2, x_3] \\
&\vdots \quad \vdots \\
S_{q-1}(x) &= y_{q-1} + \theta_{q-1,1}\Delta x_{q-1} + \theta_{q-1,2}\Delta x_{q-1}^2 + \theta_{q-1,3}\Delta x_{q-1}^3 \quad \text{for } x \in [x_{q-1}, x_q]
\end{aligned}
$$

with $\Delta x_i = x - x_i$.

This univariate spline has $3(q-1)$ coefficients, which can be solved with the following conditions. At the interior points the function needs to be continuous, which gives us $q-1$ conditions:

$$S_i(x_{i+1}) = y_{i+1}$$

In addition the first and second derivative have to be continuous at the interior points, which gives us two times $q-2$ conditions:

$$
\begin{aligned}
S_i'(x_{i+1}) &= S_{i+1}'(x_{i+1}) \\
S_i''(x_{i+1}) &= S_{i+1}''(x_{i+1})
\end{aligned}
$$

The not-a-knot end conditions require that the third derivative is also continuous at the gridpoints x_2 and x_{q-1}:

$$
\begin{aligned}
S_1'''(x_2) &= S_2'''(x_2) \\
S_{q-2}'''(x_{q-1}) &= S_{q-1}'''(x_{q-1})
\end{aligned}
$$

This yields a linear system of $3(q-1)$ equations in the univariate case. With multi-dimensional interpolation each dimension is treated independently, and

sequential one-dimensional interpolation is carried out[15].

2.5 Complete Chebyshev polynomials

The most commonly used basis functions are Chebyshev polynomials. Chebyshev polynomials are superior to monomial basis functions for three reasons. The first is that Chebyshev polynomials of the first kind are orthogonal to the weight $\frac{1}{\sqrt{1-x^2}}$ on the interval $[-1, 1]$ (Judd, 1998). For Chebyshev polynomials $T_i(x)$ of degree i we have:

$$\int_{-1}^{1} T_i(x) T_j(x) \frac{dx}{\sqrt{1-x^2}} = 0 \quad \text{if } i \neq j$$

Note that this orthogonality is defined for the one dimensional case. For multiple dimensions we could use the tensor product of these one-dimensional polynomials as they would be orthogonal to the product norm (Judd, 1992). However, for tensor products the number of coefficients increases exponentially with the number of state variables. For this reason we use complete polynomials[16], which only increase polynomially with the number of dimensions.

The second reason for the superiority of Chebyshev polynomials is that they are scaled such that the absolute value of the extrema never exceeds 1. The third reason is that they are very effective at reducing Runge's phenomenon. Runge's phenomenon is that polynomial interpolation results in oscillation at the edges of an interval for polynomials of high degree when equidistant nodes are used. This phenomenon is avoided with Chebyshev nodes.

To make use of the favorable properties of Chebyshev polynomials it is

[15]Matlab does not specify the algorithm for multi-dimensional interpolation, but the results are equivalent to sequential one dimensional interpolation.

[16]See Judd (1998) for a definition.

necessary to linearly map variables from the interval $[\underline{x}, \overline{x}]$ to $[-1, 1]$. This transformation is given by:

$$\tilde{x}(x) = 2\frac{x - \underline{x}}{\overline{x} - \underline{x}} - 1 \tag{8}$$

which we call the scaling down of variables. The inverse of this map, which we call scaling up, is:

$$x(\tilde{x}) = \frac{(\tilde{x} + 1)(\overline{x} - \underline{x})}{2} + \underline{x} \tag{9}$$

Chebyshev polynomials of the first kind are defined for the scaled down variables $\tilde{x}$. The polynomials have the recurrent relation:

$$\begin{aligned} T_0(\tilde{x}) &= 1 \\ T_1(\tilde{x}) &= \tilde{x} \\ T_{j+1}(\tilde{x}) &= 2\tilde{x}T_j(\tilde{x}) - T_{j-1}(\tilde{x}) \end{aligned} \tag{10}$$

where subscripts are the degree of the polynomial. The algorithm uses complete polynomials, because a complete polynomial achieves almost the same accuracy as a tensor product, despite having a lower number of coefficients (Judd, 1992). For a complete polynomial the number of coefficients grows polynomially in the number of dimensions, while tensor product grow exponentially (Judd, 1998).

For example, the complete Chebyshev polynomial of degree two with two variables x_1 and x_2 is:

$$\hat{H}(\tilde{x}; \theta) = \theta_1 + \theta_2\tilde{x}_1 + \theta_3\tilde{x}_2 + \theta_4\left(2\tilde{x}_1^2 - 1\right) + \theta_5\tilde{x}_1\tilde{x}_2 + \theta_6\left(2\tilde{x}_2^2 - 1\right) \tag{11}$$

where θ are the coefficients on the polynomial terms $i = 1, \ldots, P$. We refer to the Chebyshev polynomial terms as $\psi_i(\tilde{x}_1, \tilde{x}_2)$. The approximation of the policy function with a complete Chebyshev polynomial is:

$$\hat{H}(x;\theta) = \sum_{i=1}^{P} \theta_i \psi_i(\tilde{x}(x)) \tag{12}$$

where $\tilde{x}(x)$ scales each of the n state variable down as in equation (8).

The Chebyshev nodes are chosen on the interval $[-1, 1]$. For q nodes these are determined by the formula:

$$\tilde{x}_i = \cos\left(\frac{2i-1}{2q}\pi\right)$$

for $i = 1, \ldots, q$. The q nodes are the roots of the polynomial of degree q. For example for $q = 2$ the nodes are the roots of $2\tilde{x}^2 - 1$, which are $\pm\frac{1}{2}\sqrt{2}$. Note that these roots do not include the bounds $[-1, 1]$. In fact, the nodes are quite far from the boundaries for low order polynomials.

With Galerkin's method we obtain the coefficients θ in (12) as follows. We calculate the product of the residual function (4) and each polynomial term $\psi_j(\tilde{x}(x))$ at all gridpoints. The objective is to set the sum of these products to zero. The objective is to solve the system of equations:

$$0 = \sum_{x \in \mathcal{X}} R(x;\theta)\, \psi_j(\tilde{x}(x)) \tag{13}$$

As there are $j = 1, \ldots, P$ coefficients θ_j this is a system of P-equations in P-unknowns. This system is solved using a non-linear equations solver, based on a Newton-type of algorithm[17]. If multiple policy variables need to

[17]We use Matlab's `fsolve` with the 'trust-region-dogleg' algorithm.

be solved each policy variable has its own residual function. With d policy variables this is a system of $d \times P$ equations.

For a large number of coefficients it is more efficient to solve the coefficients of the complete polynomial with Time Iteration in combination with the Minimization of the Squared Errors. Time Iteration solves the problem at the gridpoints. Given the solution at the gridpoints the coefficients are obtained by solving a least square problem[18].

2.6 Smolyak's algorithm

Smolyak's algorithm can be implemented in various ways. We use the method described by Judd et al. (2014)[19]. The algorithm constructs a sparse grid consisting of Chebyshev extrema. The solution at the gridpoints determines the coefficients of a sparse Chebyshev polynomial. The (sparse) gridpoints are concentrated on the axis and the corners of the grid[20]. The solution at the gridpoints is computed with Direct Computation or Time Iteration. With Direct Computation we use the coefficients θ as choice variables. With Time Iteration we solve the policy variable at the gridpoints, and obtain the coefficients by solving a linear system of equations.

For the construction of the isotropic sparse grid[21] we largely follow the exposition by Fernández-Villaverde et al. (2016). First we choose the accuracy parameter μ[22]. The degree of the Chebyshev polynomial will be 2^μ. For accuracy $\mu > 0$ there are $q_\mu = 2^\mu + 1$ number of nodes in each dimension,

[18] Matlab's `mldivide` gives the least-squares solution of a linear system of equations $Ax = B$ when it is overidentified. This approach is similar to solving equation (16) as discussed in Subsection 2.6.

[19] We implemented the provided Matlab code: Rafa Valero (2021), Smolyak Anisotropic Grid (https://www.mathworks.com/matlabcentral/fileexchange/50963-smolyak-anisotropic-grid), MATLAB Central File Exchange. Retrieved November, 2021.

[20] See for example Figure 11 in Fernández-Villaverde et al. (2016).

[21] The construction of the anisotropic grid as described by Judd et al. (2014) is very similar. The accuracy parameter μ is then specified for each dimension.

[22] In the notation of Fernández-Villaverde et al. (2016): $\mu = q - n$.

and for $\mu = 0$ there is one node $q_0 = 1$.

The extrema of a univariate Chebyshev polynomial (also called Gauss-Lobotto nodes) for given μ with $j = 1, \ldots, q_\mu$ are (Judd et al., 2014):

$$\zeta_j^\mu = \begin{cases} 0 & \text{for } \mu = 0 \\ -\cos\left(\frac{j-1}{q_\mu - 1}\pi\right) & \text{for } \mu > 0 \end{cases}$$

We define the nested sets:

$$\mathcal{G}^\mu = \left\{\zeta_1^\mu, \ldots, \zeta_{q_\mu}^\mu\right\}$$

where $\mathcal{G}^\mu \subset \mathcal{G}^{\mu+1}$. The first three sets are: $\mathcal{G}^0 = \{0\}$, $\mathcal{G}^1 = \{-1, 0, 1\}$, and $\mathcal{G}^2 = \left\{-1, -\frac{\sqrt{2}}{2}, 0, \frac{\sqrt{2}}{2}, 1\right\}$. We introduce the notation $T_k^i = T_k(\tilde{x}_i)$, which is the univariate Chebyshev basis function of degree k in dimension i as defined in (10). The nodes $\mathcal{G}^\mu$ correspond to the extrema of the basis functions $T_0, \ldots, T_{2^\mu}$, with the extremum of T_0 set at 0.

The multivariate sparse grid is a union of the Cartesian products[23]:

$$\mathbb{G}(\mu, n) = \bigcup_{\sum \mu_n = \mu} (\mathcal{G}^{\mu_1} \times \ldots \times \mathcal{G}^{\mu_n}) \tag{14}$$

For example with $n = 2$ dimensions and $\mu = 1$ (meaning a degree $2^\mu = 2$ polynomial) we get:

[23]Fernández-Villaverde et al. (2016) include Cartesian products with $\sum \mu_n < \mu$, but since $\mathcal{G}^\mu \subset \mathcal{G}^{\mu+1}$ these lower ranked Cartesian products are redundant. For example $\mathcal{G}^1 \times \mathcal{G}^0 \subset \mathcal{G}^2 \times \mathcal{G}^0$

$$\begin{aligned}\mathbb{G}(1,2) &= \bigcup_{\sum \mu_n = 1} (\mathcal{G}^{\mu_1} \times \mathcal{G}^{\mu_2}) \\ &= \left(\mathcal{G}^1 \times \mathcal{G}^0\right) \cup \left(\mathcal{G}^0 \times \mathcal{G}^1\right) \\ &= \{(-1,0),(0,0),(1,0)\} \cup \{(0,-1),(0,0),(0,1)\} \\ &= \{(0,0),(-1,0),(1,0),(0,-1),(0,1)\}\end{aligned}$$

Similarly with $n = 2$ dimensions and $\mu = 2$ (meaning a degree $2^\mu = 4$ polynomial) we get:

$$\begin{aligned}\mathbb{G}(2,2) &= \bigcup_{\sum \mu_n = 2} (\mathcal{G}^{\mu_1} \times \mathcal{G}^{\mu_2}) \\ &= \left(\mathcal{G}^2 \times \mathcal{G}^0\right) \cup \left(\mathcal{G}^0 \times \mathcal{G}^2\right) \cup \left(\mathcal{G}^1 \times \mathcal{G}^1\right) \cup \left(\mathcal{G}^1 \times \mathcal{G}^1\right)\end{aligned}$$

Note that $\mathcal{G}^1 \times \mathcal{G}^0 \subset \mathcal{G}^2 \times \mathcal{G}^0$, so $\mathbb{G}(1,2) \subset \mathbb{G}(2,2)$.

The sparse grid exactly identifies the coefficients of a polynomial, and we can infer from the grid, which polynomial terms are included. For example, for a two dimensional grid ($n = 2$) and an accuracy $\mu = 1$ we get a degree $2^\mu = 2$ polynomial. The grid consists of the sets $(\mathcal{G}^1 \times \mathcal{G}^0) \cup (\mathcal{G}^0 \times \mathcal{G}^1)$. This corresponds to a polynomial consisting of only univariate terms T_0, T_1^i, and T_2^i for $i = 1, 2$. The degree 2 bivariate terms $T_1^i T_1^j$ for $i \neq j$ are omitted.

To generalize this we define the set of univariate Chebyshev polynomials up to order k as $\mathcal{T}_i^k = \{T_0^i, T_1^i, \ldots, T_k^i\}$. Note that for accuracy μ the degree of the polynomial is 2^μ. For example, in two dimensions with $\mu_1 = 2$ and $\mu_2 = 1$ the Cartesian product $\mathcal{G}^2 \times \mathcal{G}^1$ defines 15 gridpoints. The corresponding tensor product $\mathcal{T}_1^{2^{\mu_1}} \otimes \mathcal{T}_2^{2^{\mu_2}}$ is a set of 15 bivariate polynomials:

$$\mathcal{T}_1^4 \otimes \mathcal{T}_2^2 = \left\{ \begin{array}{lll} T_0^1 T_0^2, & \ldots & , T_4^1 T_0^2 \\ T_0^1 T_1^2, & \ldots & , T_4^1 T_1^2 \\ T_0^1 T_2^2, & \ldots & , T_4^1 T_2^2 \end{array} \right\} \tag{15}$$

This sparse grid and sparse set of polynomials is very effective at tackling the curse of the dimensionality. A standard Cartesian product with q nodes in n dimensions consists of a total of q^n nodes. We would need at least 5 gridpoints in each dimension to estimate a degree 4 complete polynomial. For 8 dimensions this would result in $5^8 = 390,625$ gridpoints to estimate a total of 495 coefficients.

The Smolyak algorithm with a sparse degree 4 polynomial (ie. $\mu = 2$) in 8 dimensions results in only 145 nodes and coefficients. The resulting polynomial consists of the univariate terms T_0, T_1^i, T_2^i, T_3^i, and T_4^i for $i = 1, \ldots, 8$, and all possible combinations of the bivariate terms $T_1^i T_1^j$, $T_2^i T_1^j$, and $T_2^i T_2^j$ for $i \neq j$. Compared to a complete polynomial 350 terms are omitted by the Smolyak algorithm: the degree 3 terms $T_1^i T_1^j T_1^l$, and the degree 4 terms $T_1^i T_1^j T_1^l T_1^m$, $T_2^i T_1^j T_1^l$ and $T_3^i T_1^j$ for all possible combination with $i \neq j \neq l \neq m$. In general, a degree 4 Smolyak polynomial does not contain polynomial terms consisting of more than 2 variables.

Following the explanation by Judd et al. (2014) the polynomial results in the approximation of the policy variable:

$$\hat{Y}(x;\theta) = \sum_{i=1}^{m} \theta_i \varphi_i(x)$$

where m is the total number of polynomial terms, equal to the number of gridpoints, and $\varphi_i(x)$ are the sets of multivariate polynomial terms as in the example (15).

With Time Iteration we get a linear system of equations given the solution

at the gridpoints:

$$\begin{bmatrix} \hat{Y}(x_1;\theta) \\ \vdots \\ \hat{Y}(x_m;\theta) \end{bmatrix} = \begin{bmatrix} \varphi_1(x_1) & \cdots & \varphi_m(x_1) \\ \vdots & \ddots & \vdots \\ \varphi_1(x_m) & \cdots & \varphi_m(x_m) \end{bmatrix} \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix} \tag{16}$$

or in matrix notation $\hat{Y} = \Phi\theta$. The coefficients θ can be determined using matrix inversion $\theta = \Phi^{-1}\hat{Y}$.

2.7 Implementation of algorithms

In the next sections we discuss the performance of the described algorithms for three models. We implement the algorithms using a simple approach, where we use pre-determined lower and upper bounds for the grid, and a linear policy function as initial guess. For the discussed models this suffices to converge to the saddle path stable solution. The results are thus obtained with a single-step approach.

In practice one will usually have to use a multi-step approach (Fernández-Villaverde et al., 2016). One reason is that we cannot know the appropriate bounds of the grid without simulating the model. Another reason is that the algorithms might not converge to the saddle path stable solution from a simple initial guess. In that case one would first have to improve the initial guess, before computing the final solution. For example, Fernández-Villaverde et al. (2016) first solve a low order approximation, and then increase the order of the approximation. A multi-step approach can also be used to improve the accuracy. For example, we can improve the accuracy by increasing the order of the polynomial, increasing the number of nodes for splines, or use asymmetric grids or polynomials.

3 Application to Standard RBC model

In this section we discuss the performance of the projection algorithms for a standard RBC model. The model consists of two continuous state variables, capital K_t and Total Factor Productivity (TFP) Z_t. Capital is determined endogenously, while TFP follows a stochastic process.

The objective function of the agent is:

$$\begin{aligned} &\max E_0 \sum_{t=0}^{\infty} \beta^t \frac{C_t^{1-\nu}}{1-\nu} - \chi \frac{H_t^{1+\frac{1}{\eta}}}{1+\frac{1}{\eta}} \\ &\text{s.t. } C_t + K_{t+1} = Z_t K_t^{\alpha} H_t^{1-\alpha} + (1-\delta) K_t \end{aligned} \tag{17}$$

The law of motion for Total Factor Productivity is given by:

$$z_t = \rho_z z_{t-1} + \sigma_z \epsilon_t \tag{18}$$

where a small case indicate logs, ie. $z_t = \log(Z_t)$. The autocorrelation coefficient is ρ_z, and the shocks are scaled by σ_z. The shocks are standard normally distributed $\epsilon_t \sim \mathcal{N}(0,1)$.

The First Order Conditions yield:

$$C_t^{-\nu} = \beta E_t C_{t+1}^{-\nu} \left[Z_{t+1} \alpha K_{t+1}^{\alpha-1} H_{t+1}^{1-\alpha} + 1 - \delta \right] \tag{19}$$

$$H_t = \left[\frac{1-\alpha}{\chi} C_t^{-\nu} Z_t K_t^{\alpha} \right]^{\frac{\eta}{1+\alpha\eta}} \tag{20}$$

Equation (19) is the dynamic optimality condition, and (20) is a static equation that determines the labor supply, given the state variables and consumption.

3.1 Residual function for Standard RBC model

To solve the model we want to numerically approximate the solution that solves the Euler equation (19). We choose consumption as the policy variable. The exact solution gives consumption as function of the state variables $C_t = C(K_t, Z_t)$. We approximate the policy function numerically $\hat{C}_t = \hat{C}(K_t, Z_t; \theta)$, where θ is the vector of parameters[24]. These parameters define either a spline or a polynomial. We use θ^j to denote period t choices and θ^{j-1} for period $t+1$ choices. We only need this distinction for Time Iteration though. For the other methods these policies are the same $\theta^j = \theta^{j-1}$.

To determine the coefficients θ we need a residual function that computes the errors when we use the approximation instead of the exact solution. This residual function is based on the Euler equation (19). We first introduce the functions:

$$H_t = H(K_t, Z_t, C_t) \tag{21}$$

$$\begin{aligned} K_{t+1} &= Z_t K_t^{\alpha} H(K_t, Z_t, C_t)^{1-\alpha} + (1-\delta) K_t - C_t \\ &= K(K_t, Z_t, C_t) \end{aligned} \tag{22}$$

For a given approximation of the policy function $\hat{C}_t = \hat{C}(K_t, Z_t; \theta^j)$ we can compute $\hat{H}_t = H\left(K_t, Z_t, \hat{C}(K_t, Z_t; \theta^j)\right)$ and $\hat{K}_{t+1} = K\left(K_t, Z_t, \hat{C}(K_t, Z_t; \theta^j)\right)$. Note that K_{t+1} depends on period t choices determined by θ^j.

Next we need to evaluate the right-hand side of the Euler equation (19), which consists of time $t+1$ variables. Using (21) we define a function G for part of the right-hand side of the Euler equation:

[24] In practice we use control and state variables in logs to approximate the policy function $\log\left(\hat{C}_t\right) = \hat{c}(\log(K_t), \log(Z_t); \theta)$.

$$G\left(K_{t+1}, Z_{t+1}, C_{t+1}\right) = C_{t+1}^{-\nu}\left[Z_{t+1}\alpha K_{t+1}^{\alpha-1} H\left(K_{t+1}, Z_{t+1}, C_{t+1}\right)^{1-\alpha} + 1 - \delta\right]$$

The approximation of G is:

$$\hat{\Phi}\left(K_{t+1}, Z_{t+1}; \theta^{j-1}\right) = G\left(K_{t+1}, Z_{t+1}, \hat{C}\left(K_{t+1}, Z_{t+1}; \theta^{j-1}\right)\right) \tag{23}$$

The right-hand side of the Euler equation (19) includes an expectation operator. We use Gauss-Hermite quadrature to approximate the expected value. Assume we have a function $f\left(z_{t+1}, x\right)$ and z_{t+1} is governed by (18) with standard normally distributed shocks ϵ_{t+1}. The Gauss-Hermite approximation is then:

$$E_t f\left(z_{t+1}, x\right) \approx \sum_{l=1}^{L} \frac{\omega_l}{\sqrt{\pi}} f\left(\rho_z z_t + \sigma_z \sqrt{2}\zeta_l, x\right) \tag{24}$$

where ζ_l are the Gauss-Hermite nodes, and ω_l are the Gauss-Hermite weights (see Appendix A for the derivation).

With the Gauss-Hermite formula (24) the approximation of the expected value of (23) becomes:

$$\begin{aligned}
&\Psi\left(K_t, Z_t; \theta\right) \approx E_t\left\{\hat{\Phi}\left(K\left(K_t, Z_t, \hat{C}\left(K_t, Z_t; \theta^j\right)\right), Z_{t+1}; \theta^{j-1}\right)\right\} \\
&= \sum_{l=1}^{L} \frac{\omega_l}{\sqrt{\pi}} \hat{\Phi}\left(K\left(K_t, Z_t, \hat{C}\left(K_t, Z_t; \theta^j\right)\right), \exp\left(\rho_z \log\left(Z_t\right) + \sigma_z\sqrt{2}\zeta_l\right); \theta^{j-1}\right)
\end{aligned} \tag{25}$$

After substituting this expression into the Euler equation (19) we obtain a residual function that depends on K_t, Z_t and the parameters θ:

Table 1: Parameters of Standard RBC model

Parameter	Value
Exponent capital (α)	0.36
Discount factor (β)	0.985
Depreciation (δ)	0.025
Risk aversion (ν)	2.00
Labor supply el. (η)	4.00
Scalar labor supply (χ)	1.00
Autoc. TFP (ρ_z)	0.95
St.d. shocks (σ_z)	0.01

$$R\left(K_t, Z_t; \theta\right) = \beta \Psi\left(K_t, Z_t; \theta\right) / \hat{C}\left(K_t, Z_t; \theta\right)^{-\nu} - 1 \tag{26}$$

We divided the Euler equation by $\hat{C}\left(K_t, Z_t; \theta\right)^{-\nu}$ to ensure the approximation is good over the whole interval. Without this scaling the normalized Euler Equation Errors (see Subsection 3.2) will be larger for high levels of consumption. The reason is that stopping criteria will result in similar absolute errors in the objective function over the whole interval. Without scaling the Euler equation (19) that would imply similar errors in the marginal utility of consumption $C^{-\nu}$. As the marginal utility decreases with consumption the errors in consumption would then increase with the level of consumption.

3.2 Performance for Standard RBC model

To evaluate each of the five algorithms we evaluate the accuracy and computation time for the Standard RBC model. The parameters of the model are listed in Table 1. We set the boundaries of the grid at $k_{ss} \pm 0.1275$ for capital (in logs) and $\pm 2.6 \cdot \sigma_z \sqrt{\frac{1}{1-\rho_z^2}}$ for TFP (in logs). These values ensure that roughly 99% of all observations in a simulation fall within the grid. To approximate the expected value we use 5 Gauss-Hermite nodes for both the residual function (26), and the computation of the normalized errors (see

equation (27) below).

As initial guess for the policy function we use a linear guess:

$$c = c_{ss} + c_k \left(k - k_{ss}\right) + c_z \left(z - z_{ss}\right)$$

where smaller cases refer to variables in logs, ie. $x = \log\left(X\right)$ for $X = [K, Z, C]$. We have used an initial guess with $c_k = 0.25$ and $c_z = 0.25$, which is not too close to the first order perturbation solution ($c_k = 0.346$ and $c_z = 0.353$). We used tight stopping criteria to obtain very good approximations at the cost of computation times. We reduced the tolerances to 10^{-12} for all solvers[25], and used $\epsilon^d = \epsilon^r = 10^{-12}$ for Time Iteration[26].

As measure for the accuracy we use the maximum of the normalized Euler Equation Errors (EEE) in consumption equivalent units (Judd, 1992), which is a dimension free measure and a standard accuracy measure[27]. A normalized Euler Equation Errors of 0.01 means the error in the Euler Equation in consumption terms is about 1%. The normalized Euler Equation Error is:

$$EEE\left(K_t, Z_t; \theta\right) = \frac{\left[\beta \Psi\left(K_t, Z_t; \theta\right)\right]^{-1/\nu}}{\hat{C}\left(K_t, Z_t; \theta\right)} - 1 \tag{27}$$

We measure the maximum normalized Euler residuals *on grid* and *off grid*. The *on grid* errors are the errors on the initial grid[28], which is used to find the optimal solution. The *off grid* errors, referred to as 'the errors', are measured on an equidistant grid with 1,000 nodes in each dimension. For the spline and Smolyak algorithms the *on grid* errors should be close to zero, because the approximation passes through the solution at the gridpoints.

[25]The tolerances for the Matlab solvers are 'OptimalityTolerance', 'FunctionTolerance' and 'StepTolerance'.

[26]See Subsection 2.4 for the definitions.

[27]See for example Fernández-Villaverde et al. (2016) and Guerrieri and Iacoviello (2015)

[28]See Subsection 2.2.

Similar errors on and off the grid indicate that errors on the grid drive the errors off the grid. In that case tighter stopping criteria for the spline and Smolyak algorithms can lower errors on the grid, which should lower errors off the grid as well. This will not be the case for the complete polynomials, because they are overidentified as the number of gridpoints is higher than the number of parameters[29].

All computation times are based on the average of solving the model 5 times, including the construction of the grid. The evaluation of the errors is not included in the computation time. The computation of simulations is relatively fast though[30]. A simulation consisting of 1,000 series, each with a length of 1,000 periods, is computed in less than 0.2 seconds for the spline methods. For the Chebyshev polynomials computation of these 1,000,000 points takes about 0.16 seconds for a degree 1 polynomial approximation, and 0.65 seconds for a degree 7 polynomial. For the Smolyak solution it takes 0.18 seconds when $\mu = 1$ (degree 2 polynomial) and 0.65 seconds when $\mu = 4$ (degree 16 polynomial).

For comparison we have also included the performance results of the perturbation solution of orders 1 to 3. For these results we used the CSD toolbox for Matlab[31], which has much less overhead costs than Dynare. The CSD toolbox will therefore be faster than Dynare when solving a simple model only once. The solutions of the CSD toolbox and Dynare are practically identical with maximum differences in the state variables (in logs) smaller than 10^{-12} after a simulation of 1,000 periods.

For splines we vary the number of nodes in each dimension. For the complete Chebyshev polynomials we use approximations of order 1 to 7. We used the minimum number of nodes (the order plus 1) in each dimension,

[29]The exception are policy functions with one state variable and the number of nodes set to the order plus 1.

[30]The policy function is evaluated in vectorized format, which reduces computation times in Matlab.

[31]See www.saduineveld.com/tools

because increasing the number of nodes usually has little effect on the accuracy (Fernández-Villaverde et al., 2016)[32]. For the Smolyak algorithm we vary the accuracy parameter μ.

The results in terms of accuracy and computation time are shown in Table 2. The most noticeable are the low computation times for projection methods. For all three basis functions a maximum (off grid) error of 10^{-6} can obtained in less than 0.05 seconds. In fact, projection can be faster and more accurate than perturbation when we solve a model only once[33].

The algorithm using a Spline with Direct Computation is very fast and accurate for a low number of gridpoints[34]. Direct Computation is fast in that case, because the Jacobian matrix is small, and the Newton-solver converges at least quadratically close to the solution (Judd, 1998). This also results in small *on grid* errors. Since the dense Jacobian is $m \times m$ and will be approximated with finite differences computation time increases rapidly with the total number of nodes m. For larger grids Time Iteration therefore outperforms Direct Computation. The algorithm performs well in terms of accuracy as *(off grid)* errors range between 10^{-6} for 3 nodes in each dimension, and 10^{-10} for 15 nodes. It is slower than the methods relying on polynomials though, but the computation time is still below 1 second for a total of 225 gridpoints.

The algorithm Spline with Time Iteration is slower than Direction Computation for a low number of gridpoints, and faster for a high number of gridpoints. The computation time is just above 1 second for a total of 225 gridpoints. The reason for the high computation time is that convergence

[32]Numerical tests confirmed that in general a higher number of nodes has little effect on the accuracy for complete Chebyshev polynomials, which are already overidentified.

[33]For perturbation methods computation times can be reduced significantly during a calibration exercise. The reason is that symbolic differentiation is a time consuming process for perturbation methods, which only has to be carried out once for a given model.

[34]With 25 and 50 nodes the equation solver does not converge to a solution from a poor initial guess. The reason is that the system of equations is highly non-linear, and therefore poorly approximated with a local linearization (ie. the Jacobian matrix). With a better initial guess ($c_k = 0.35$ and $c_z = 0.35$) the solver does converge for 25 and 50 nodes.

Table 2: Performance for Standard RBC model

	Spline, Direct Computation						
Nodes per dim.	3	5	7	10	15		
Total Nodes	9	25	49	100	225		
Comp. time	0.05	0.06	0.12	0.27	0.73		
EEE, off grid	-6.3	-8.9	-9.5	-10.1	-10.8		
EEE, on grid	-15.4	-15.4	-14.9	-14.5	-14.1		
	Spline, Time Iteration						
Nodes per dim.	3	5	7	10	15	25	50
Total Nodes	9	25	49	100	225	625	2500
Comp. time	0.81	0.85	0.90	0.99	1.21	2.05	5.00
EEE, off grid	-6.3	-8.9	-9.5	-10.1	-10.8	-11.7	-12.3
EEE, on grid	-12.3	-12.3	-12.3	-12.3	-12.3	-12.3	-12.3
	Complete Chebyshev poly., Galerkin						
Order	1	2	3	4	5	6	7
Total Nodes	4	9	16	25	36	49	64
Comp. time	0.04	0.02	0.04	0.07	0.11	0.18	0.28
EEE, off grid	-3.4	-5.6	-7.1	-8.8	-10.8	-12.0	-13.4
EEE, on grid	-3.9	-6.0	-7.4	-9.2	-10.9	-12.3	-13.7
	Smolyak, Direct Computation						
Accuracy (μ)	1	2	3	4			
Total Nodes	5	13	29	65			
Comp. time	0.04	0.04	0.15	0.64			
EEE, off grid	-3.7	-7.5	-11.1	-12.8			
EEE, on grid	-15.6	-15.5	-15.1	-12.8			
	Perturbation						
Order	1	2	3				
Comp. time	0.15	0.18	0.30				
EEE (off grid)	-3.32	-4.44	-6.38				

Computation times in seconds. Errors are the maximum Euler Equation Errors, in absolute values and log10. 'Off grid' refers to the equidistant grid with 1 million nodes, while 'on grid' refers to the initial grid used to solve the model.

is linear at best Judd (1998). In addition we used relatively tight stopping criteria[35] with $\epsilon^d = \epsilon^r = 10^{-12}$. With looser stopping criteria $\epsilon^d = \epsilon^r = 10^{-6}$

[35]See Subsection 2.4 for the definitions.

computation times can be reduced by a factor 4. With 50 nodes errors can be reduced to 10^{-12} in accordance with an error decay of $\mathcal{O}\left(q^{-4}\right)$ for four times differentiable functions, where q is the number of nodes in each dimension (De Boor, 1978).

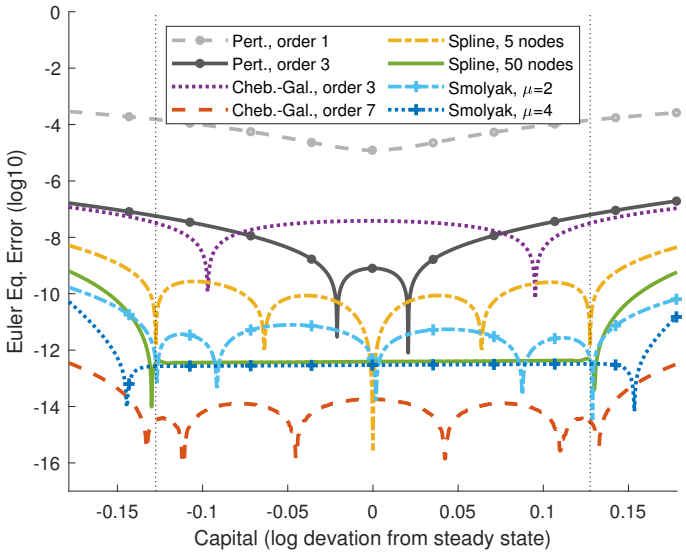
For both spline algorithms the errors on the grid are much smaller than the errors off the grid, except with 50 nodes. This indicates *off grid* errors result from interpolation for 25 or less nodes. For Time Iteration with 50 nodes in each dimension the interpolation errors become almost zero as *on* and *off grid* errors are almost equal. In this case tighter stopping criteria are necessary to reduce errors further.

The algorithm using complete Chebyshev polynomials with Galerkin projection is both fast and accurate. For example, the third order approximation is found in 0.06 seconds, and has a maximum error of 10^{-7}. Increasing the order of the approximation to 7 we can reduce the maximum error to 10^{-13}, computed in just 0.3 seconds.

For this simple model Smolyak's algorithm with Direct Computation is also fast and accurate. For a required accuracy level it is slower than a complete Chebyshev polynomial with Galerkin projection though. The main reason is that the Smolyak polynomial approximates the policy function less efficiently. For example, for the Smolyak algorithm with $\mu = 4$ the errors are -12.8 in log10. This approximation has 65 gridpoints and coefficients, and includes a degree 16 monomial. With a complete Chebyshev polynomials we obtain a smaller error of -13.4 in log10 with a degree 7 polynomial, which has 36 coefficients and requires 64 gridpoints.

We may conclude that Chebyshev polynomials with Galerkin projection are the best choice for this model as the algorithm is both fast and accurate. For low accuracy levels the performance differences between Chebyshev polynomials, Splines and Smolyak's algorithm are small. We should however be careful drawing any general conclusions based on these results, because the model has only two state variables and is near-linear. This is reflected in the

Figure 1: Euler Equation Errors for RBC model ($Z = 1$)



Vertical dotted lines are the bounds of the grid.

small errors for low order approximations. Such small errors are unlikely in models with more state variables or in models that are less well-behaved as discussed in the next two sections.

To conclude this section we plot the normalized Euler residuals for a selection of algorithms in Figure 1[36]. The errors are shown as a function of the capital stock when TFP is at the steady state level. Figures for other values of TFP look similar[37]. For the first and third order perturbation solution the errors increase with the distance from the steady state. This is expected as perturbation solutions are local by definition. For projection methods errors are more constant inside the grid, consistent with their global nature. For projection methods errors can increase rapidly outside the boundaries of the grid.

[36]Note the similarity with Figure 15 in Fernández-Villaverde et al. (2016) on page 626.

[37]The exception is that the errors of the perturbation solution are more skewed towards one end. Also the figures with TFP on the x-axis look similar (for different values of the capital stock).

For the third order Chebyshev-Galerkin solution the maximum error is only slightly smaller than for the third order perturbation solution. The errors are very small for the seventh order Chebyshev-Galerkin solution. For the Smolyak algorithm with $\mu = 4$ (degree 16 polynomial) and the Spline with 50 nodes the Euler residuals are very constant inside the bounds of the grid, and of similar magnitude.

4 Application to HIA model

In this section we analyze the performance of the algorithms in an RBC model with Habits in consumption, and Investment Adjustment costs (HIA). The agents in the model maximize expected utility:

$$\max E_0 \sum_{t=0}^{\infty} \beta^t \frac{\left(C_t / H_{t-1}\right)^{1-\nu}}{1-\nu}$$

with $H_{t-1} = C_{t-1}^{\gamma}$ and $\gamma \in [0, 1)$. The optimization is subject to the constraints:

$$C_t + I_t = Z_t K_t^{\alpha} \tag{28}$$

$$K_{t+1} = (1-\delta) K_t + I_t \left(1 - \varphi\left(\frac{I_t}{I_{t-1}}\right)\right) \tag{29}$$

The First Order Conditions yield:

$$\Lambda_t = \Upsilon_t \left[1 - \varphi\left(\frac{I_t}{I_{t-1}}\right) - \frac{I_t}{I_{t-1}}\varphi'\left(\frac{I_t}{I_{t-1}}\right)\right] + \beta E_t \Upsilon_{t+1} \left(\frac{I_{t+1}}{I_t^2}\right)^2 \varphi'\left(\frac{I_{t+1}}{I_t}\right) \tag{30}$$

$$\Upsilon_t = \beta E_t \left\{\Lambda_{t+1} \alpha Z_{t+1} K_{t+1}^{\alpha-1} + \Upsilon_{t+1}(1-\delta)\right\} \tag{31}$$

$$C_t = H_{t-1} \left(\Lambda_t H_{t-1}\right)^{\frac{1}{-\nu}} \tag{32}$$

In addition we define $\varphi\left(\frac{I_t}{I_{t-1}}\right) = \frac{1}{\varrho}\left(\frac{I_t}{I_{t-1}} - 1\right)^2$ such that $\varphi(1) = 0$, $\varphi'(1) = 0$ as in Jaimovich and Rebelo (2009). The law of motion for Total Factor Productivity is the same as in the standard RBC model, and given by (18).

As policy variables we take Λ_t and Υ_t, which are the Lagrangian multipliers on the resource constraint (28), and the evolution of capital (29), respectively. The policy functions are $\hat{\Lambda}_t = \hat{\Lambda}(K_t, Z_t, I_{t-1}, H_{t-1}; \theta^1)$ and $\hat{\Upsilon}_t = \hat{\Upsilon}(K_t, Z_t, I_{t-1}, H_{t-1}; \theta^2)$. Given the policy functions we can compute C_t, I_t and K_{t+1} with equations (28), (29) and (32). Using the approach as in Subsection 3.1 the residual functions for $d = [1, 2]$ can be written in the format:

$$R^d(K_t, Z_t, I_{t-1}, H_{t-1}; \theta) = \beta \Psi^d(K_t, Z_t, I_{t-1}, H_{t-1}; \theta) / \Xi\left(\cdot; \theta^d\right) - 1$$

where the policy functions are $\hat{\Lambda}_t = \Xi(\cdot; \theta^1)$ and $\hat{\Upsilon}_t = \Xi(\cdot; \theta^2)$.

4.1 Performance for HIA model

We evaluate the performance in terms of accuracy and computation time. The parameters of the model are listed in Table 3. For the boundaries of the grid (all in logs) we set $k_{ss} \pm 0.15$ for capital, $i_{ss} \pm 0.2$ for investment, $h_{ss} \pm 0.06$ for habits H, and ± 0.1 for TFP. With these boundaries about 99.9% of all observations fall within the grid. We use 5 Gauss-Hermite nodes

Table 3: Parameters of HIA model

Parameter	Value
Exponent capital (α)	0.33
Discount factor (β)	0.960
Depreciation (δ)	0.100
Risk aversion (ν)	2.00
Habit parameter (γ)	0.50
Investm. adj. param. (ϱ)	1.54
Autoc. TFP (ρ_z)	0.90
St.d. shocks (σ_z)	0.01

to compute the expected value of Total Factor Productivity. As initial guess for the policy function we use the first order perturbation solution.

The normalized Euler Equation Errors (EEE) in consumption equivalent units (Judd, 1992) is used as the accuracy measure. To compute these errors we use a similar procedure as Fernández-Villaverde et al. (2016). We define a function $C_t = g\left(H_{t-1}, \Lambda_t\right)$ based on (32). Using the policy function for Λ in period t we get consumption measure $\hat{C}_t^a$:

$$\hat{C}_t^a = g\left(H_{t-1}, \hat{\Lambda}\left(K_t, Z_t, I_{t-1}, H_{t-1}; \theta^1\right)\right)$$

The alternative $\hat{C}_t^b$ is based on the implied value of $\hat{\Lambda}_t^b$ by the right-hand side of (30) after substituting the policy function for Υ_t and Υ_{t+1}, and computing I_t and I_{t+1} using the policy functions and equations (28), (29) and (32). With the implied value $\hat{\Lambda}_t^b$ we compute $\hat{C}_t^b = g\left(H_{t-1}, \hat{\Lambda}_t^b\right)$, and this results in the Euler Equation Errors:

$$EEE\left(K_t, Z_t, I_{t-1}, H_{t-1}; \theta\right) = \frac{\hat{C}_t^b}{\hat{C}_t^a} - 1$$

We compute errors on and off the grid as with the Standard RBC model.

The *on grid* errors refer tot the initial grid used to compute the residual function. The *off grid* errors are computed on an equidistant grid with 30 nodes in each dimensions.

The performance results are shown in Table 4. Low order approximations have relatively large errors. For example, degree 3 polynomials have maximum errors of the order 10^{-2}. This is of similar magnitude as the error in the third order perturbation approximation. In addition, higher order polynomials reduce errors relatively slowly, while computation times increase rapidly with the required accuracy level. With splines computation times increase slower with the required accuracy level. For an accuracy of at least 10^{-3} splines are the fastest method with a computation time of 1.39 seconds. Other methods take at least two times longer to achieve the same accuracy level, and this difference increases with the required accuracy level. It should further be noted that Spline approximation with Direct Computation is slower than Time Iteration even with 3 nodes per dimension. The increase in the size of the Jacobian matrix is only part of the explanation[38], and apparently the system of equations is highly non-linear, which results in slower convergence[39].

5 Application to a Limit Cycle model

Models featuring an attracting limit cycle can be solved efficiently with perturbation methods[40] or projection methods (Galizia, 2021). This example

[38]The Jacobian matrix is 162×162 with 3 nodes in each dimension, but for the Standard RBC model Direct Computation was faster than Time Iteration with a 225×225 Jacobian matrix.

[39]A change in a policy variable at a gridpoint will change the spline, and also affect the other policy variable, which results in a highly non-linear system. The Newton-type solver will locally linearize the system of equations, and may therefore converge slowly.

[40]The perturbation solution will not satisfy the Blanchard-Kahn conditions (Blanchard and Kahn, 1980). For this reason most perturbation toolboxes can not handle such models yet. The only known exception is the CSD toolbox (see www.saduineveld.com/tools), which implements the methodology and code developed Galizia (2021) to solve limit cycle

Table 4: Performance for HIA model

	Spline, Time Iteration							
Nodes per dim.	3	4	5	6	7	8	9	10
Total Nodes	81	256	625	1296	2401	4096	6561	10000
Comp. time	0.75	1.39	2.29	3.80	4.99	8.76	14.77	24.10
EEE, off grid	-2.6	-3.8	-4.5	-4.8	-5.2	-5.5	-5.7	-5.9
EEE, on grid	-8.3	-8.3	-8.3	-8.3	-8.3	-8.3	-8.3	-8.3
	Complete Chebyshev poly., Galerkin							
Order	1	2	3	4	5			
Total Nodes	16	81	256	625	1296			
Comp. time	0.24	0.35	1.57	5.96	26.07			
EEE, off grid	-1.0	-1.5	-2.7	-3.2	-3.7			
EEE, on grid	-1.5	-1.9	-3.0	-3.4	-4.0			
	Complete Chebyshev poly., Time Iteration							
Order	1	2	3	4	5	6	7	8
Total Nodes	16	81	256	625	1296	2401	4096	6561
Comp. time	0.63	0.75	1.50	3.22	6.97	18.36	69.35	166.25
EEE, off grid	-0.8	-1.4	-2.5	-3.0	-3.6	-4.5	-5.1	-5.7
EEE, on grid	-1.3	-1.7	-2.8	-3.2	-3.8	-4.8	-5.4	-5.9
	Smolyak, Time Iteration							
Accuracy (μ)	1	2	3	4				
Total Nodes	9	41	137	401				
Comp. time	0.65	1.10	3.85	11.03				
EEE, off grid	-1.1	-1.9	-3.4	-4.4				
EEE, on grid	-8.3	-8.3	-8.4	-8.3				
	Perturbation							
Order	1	2	3					
Comp. time	0.23	0.55	2.83					
EEE (off grid)	-0.74	-1.38	-1.95					

Computation times in seconds. Errors are the maximum Euler Equation Errors, in absolute values and log10. 'Off grid' refers to the equidistant grid with 1 million nodes, while 'on grid' refers to the initial grid used to solve the model.

shows that the third order perturbation solution may be a poor approxi-

models with perturbation methods, up to order 3.

mation of a highly non-linear model with an attracting limit cycle. With projection methods the approximation is much better, due to their global nature. However, the Smolyak algorithm can not obtain a solution, because it will diverge even from a very accurate initial guess.

5.1 Limit Cycle model

The model with an attracting limit cycle that we use is a Real Business Cycle model with external habits in the labor supply (Duineveld, 2021) (see Appendix B for details). The dynamic equations are:

$$\Lambda_t = \beta\Lambda_{t+1}\left(\phi\alpha K_{t+1}^{\alpha-1}L_{t+1}^{\gamma} + 1 - \delta\right) \tag{33}$$

$$C_t + K_{t+1} = K_t^{\alpha}L_t^{\gamma} + (1-\delta)\,K_t \tag{34}$$

$$\hat{\psi}_t = (1-m)\,\eta_t + m\hat{\psi}_{t-1} \tag{35}$$

This is a three dimensional system, which consists of the two state variables K_t and ψ_{t-1}, and the control variable Λ_t. Hats indicates log deviations from steady state.

The static variables are the labor supply L_t, consumption C_t, and the current habit measure η_t:

$$L_t = \left(\frac{\phi\gamma}{\theta}\right)^{\frac{1}{\theta-\gamma}} \psi_{t-1}^{\frac{-1}{\theta-\gamma}} K_t^{\frac{\alpha}{\theta-\gamma}} \tag{36}$$

$$C_t = \left[\Lambda_t^{\frac{1}{-\nu}} + \psi_{t-1}L_t^{\theta}\right] \tag{37}$$

$$\eta_t = \zeta_1\hat{L}_t + \zeta_3\hat{L}_t^{\,3} \tag{38}$$

5.2 Residual function for Limit Cycle model

To solve the model we take Λ as the policy variable, which is approximated by $\hat{\Lambda}_t = \hat{\Lambda}(K_t, \psi_{t-1}; \theta)$. As before θ^j denotes this period's policy, and θ^{j-1} denotes next period's policy, although this distinction is only relevant for Time Iteration.

Writing $L_{t+1} = L(K_{t+1}, \psi_t)$ for the labor supply (36) we define a function G for part of the right-hand side of the Euler equation (33), and its approximation $\hat{\Phi}$:

$$G(K_{t+1}, \psi_t, \Lambda_{t+1}) = \Lambda_{t+1}\left(\phi\alpha K_{t+1}^{\alpha-1} L(K_{t+1}, \psi_t)^{\gamma} + 1 - \delta\right) \tag{39}$$

$$\hat{\Phi}\left(K_{t+1}, \psi_t; \theta^{j-1}\right) = G\left(K_{t+1}, \psi_t, \hat{\Lambda}\left(K_{t+1}, \psi_t; \theta^{j-1}\right)\right) \tag{40}$$

The state variables in $t+1$ are determined by $K_{t+1} = K(K_t, \psi_{t-1}, \Lambda_t)$ and $\psi_t = \psi(K_t, \psi_{t-1})$. Substituting these expressions into (40) gives:

$$\Psi(K_t, \psi_{t-1}; \theta) = \hat{\Phi}\left(K\left(K_t, \psi_{t-1}, \hat{\Lambda}\left(K_t, \psi_{t-1}; \theta^j\right)\right), \psi(K_t, \psi_{t-1}); \theta^{j-1}\right)$$

The residual function is then:

$$R(K_t, \psi_{t-1}; \theta) = \beta\Psi(K_t, \psi_{t-1}; \theta) / \hat{\Lambda}\left(K_t, \psi_{t-1}; \theta^j\right) - 1$$

5.3 Performance for Limit Cycle model

The parameters of the model are listed in Table 5[41]. The initial guess for the policy function is the first order perturbation solution:

$$\lambda = \lambda_{ss} + \lambda_k(k - k_{ss}) + \lambda_\psi(\log(\psi) - \log(\psi_{ss}))$$

[41] We set $\alpha = \mu a$, $\gamma = \mu(1-a)$, and $\phi = 1/\mu$.

Table 5: Parameters of Limit Cycle model

Parameter	Value
Discount factor (β)	0.985
Depreciation (δ)	0.205
Risk aversion (ν)	0.25
Capital share of income (a)	0.36
Labor supply el. ($\frac{1}{\theta-1}$)	6.67
Returns-to-scale (μ)	1.23
Memory strength habits (m)	0.975
Habit, 1st ord. (ζ_1)	0.77
Habit, 3rd ord. (ζ_3)	666.56
Scaling lab. supply (ψ_0)	-0.196

where $k = \log(K)$ and $\lambda = \log(\Lambda)$. For our parameter setting the first order perturbation solution is $\lambda_k = -0.2919$ and $\lambda_\psi = -2.3373$.

As the model features an attracting limit cycle we set the boundaries of the grid such that they contain the limit cycle. We can accurately determine the limit cycle using the shooting method (Kuznetsov, 2004), because the model has no stochastic shocks. For $x = [\log(K), \log(\psi)]$ we define the minimum x^{min} and the maximum x^{max} of this deterministic invariant circle. We set the lower bounds of the grid at $\underline{x} = x^{min} - 0.15 \cdot (x^{max} - x^{min})$ and the upper bounds at $\overline{x} = x^{max} + 0.15 \cdot (x^{max} - x^{min})$ such that the invariant circle is well within the bounds of the initial grid. We use default stopping criteria for the solvers[42] and for Time Iteration $\epsilon^d = \epsilon^r = 10^{-8}$ as stopping criteria[43].

We report the computation time, and the maximum normalized Euler Equation Errors in consumption units. To compute these Euler Equation Errors we follow the approach of Fernández-Villaverde et al. (2016), and define a function $C_t = g(\Lambda_t, L_t)$ based on (37). We first compute consumption

[42] Matlab's default tolerances for `fsolve`.

[43] See Subsection 2.4 for the definitions.

using the period t policy function:

$$\hat{C}_t^a = g\left(\hat{\Lambda}\left(K_t, \psi_{t-1}; \theta\right), L\left(K_t, \psi_{t-1}\right)\right)$$

Next we compute an alternative level $\hat{C}_t^b$ with the implied value Λ_t from the approximation of the right-hand side of the Euler equation $\hat{\Lambda}_t^b = \beta\Psi\left(K_t, \psi_{t-1}; \theta\right)$. Substituting this into g yields:

$$\hat{C}_t^b = g\left(\beta\Psi\left(K_t, \psi_{t-1}; \theta\right), L\left(K_t, \psi_{t-1}\right)\right)$$

The normalized difference between $\hat{C}_t^a$ and $\hat{C}_t^b$ is the Euler Equation Error:

$$EEE\left(K_t, \psi_{t-1}; \theta\right) = \frac{g\left(\beta\Psi\left(K_t, \psi_{t-1}; \theta\right), L\left(K_t, \psi_{t-1}\right)\right)}{g\left(\hat{\Lambda}\left(K_t, \psi_{t-1}; \theta\right), L\left(K_t, \psi_{t-1}\right)\right)} - 1$$

As with the other models we report maximum Euler Equation Errors on grid, and off grid. The *on grid* errors are the errors on the initial grid (see Subsection 2.2) used to solve the model, while the *off grid* errors are measured on an equidistant grid with 1,000 nodes in each dimension.

The performance results are shown in Table 6. We report the results for Splines with both Direct Computation and Time Iteration, and for complete Chebyshev polynomials with Galerkin projection and Time Iteration. The accuracy of the solution increases with the number of gridpoints for the Spline algorithms. The maximum *off grid* errors range roughly between the order 10^{-4} and 10^{-8} for 5 and 50 nodes, respectively. A Spline with Direct Computation is much faster than Time Iteration for a low number of nodes.

With complete Chebyshev polynomials the maximum *(off grid)* errors are of the order 10^{-4} to 10^{-5}. The errors become only marginally smaller when we increase the order of the approximation from 3 to 12. Apparently the policy function is not approximated well with a complete polynomial. The errors with Galerkin and Time Iteration are similar, except for the degree

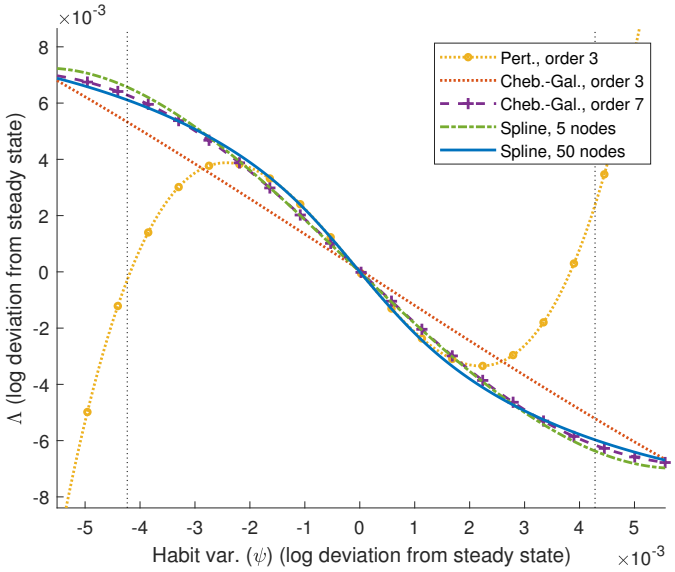
Table 6: Performance for Limit Cycle model

	Spline, Direct Computation									
Nodes per dim.	3	5	7	10	20					
Total Nodes	9	25	49	100	400					
Comp. time	0.03	0.04	0.08	0.19	1.82					
EEE, off grid	-3.7	-3.7	-4.2	-5.1	-6.2					
EEE, on grid	-7.3	-7.3	-8.7	-9.2	-10.1					
	Spline, Time Iteration									
Nodes per dim.	3	5	7	10	20	50				
Total Nodes	9	25	49	100	400	2500				
Comp. time	0.41	0.51	0.50	0.58	1.12	3.30				
EEE, off grid	-3.7	-3.7	-4.2	-5.1	-6.2	-7.8				
EEE, on grid	-8.2	-8.2	-8.2	-8.3	-8.2	-8.3				
	Complete Chebyshev poly., Galerkin									
Order	3	4	5	6	7	8	9	10	11	12
Total Nodes	16	25	36	49	64	81	100	121	144	169
Comp. time	0.05	0.04	0.07	0.15	0.18	0.32	0.47	0.67	1.21	1.64
EEE, off grid	-3.5	-3.5	-3.9	-3.9	-4.3	-4.3	-4.5	-4.5	-4.1	-4.8
EEE, on grid	-3.9	-3.7	-4.1	-4.1	-4.4	-4.4	-4.6	-4.6	-4.6	-4.9
	Complete Chebyshev poly., Time Iteration									
Order	3	4	5	6	7	8	9	10	11	12
Total Nodes	16	25	36	49	64	81	100	121	144	169
Comp. time	0.37	0.41	0.40	0.44	0.51	0.56	0.64	0.70	0.84	0.91
EEE, off grid	-3.5	-3.5	-3.9	-3.9	-4.3	-4.3	-4.5	-4.5	-4.8	-4.8
EEE, on grid	-3.9	-3.7	-4.1	-4.1	-4.4	-4.4	-4.6	-4.7	-4.9	-4.9
	Perturbation									
Order	3									
Comp. time	1.62									
EEE (off grid)	-1.74									

Computation times in seconds. Errors are the maximum Euler Equation Errors, in absolute values and log10. 'Off grid' refers to the equidistant grid with 1 million nodes, while 'on grid' refers to the initial grid used to solve the model.

11 polynomial, where the solution with Galerkin projection is less accurate than with Time Iteration. This could indicate convergence problems, possibly caused by the high non-linearity of the system of equations. Note also that

Figure 2: Policy function in ψ ($K = K_{ss}$)



for higher order polynomials Time Iteration becomes faster than Galerkin projection.

The maximum errors with projection are much lower than those of the third order perturbation solution. The maximum error in the third order perturbation solution is $10^{-1.74} \approx 0.02$ or a deviation of about 2%[44]. Although this might sound like an acceptable error margin this solution replicates the dynamics of the deterministic limit cycle very poorly as discussed below.

The policy as a function of the habit variable ψ is plotted in Figure 2. The most accurate solution is obtained with a Spline and 50 nodes (blue solid line). The third order perturbation solution does very poor far away from the steady state. The third order Chebyshev-Galerkin solution is better, but does not match the curvature well. Also the seventh order Chebyshev polynomial does not match the slope of the policy function well around the steady state. The Spline with 5 nodes does much better than the order

[44]Note that this large error is due to the high non-linearity of the system. In a better behaved limit cycle model errors will be smaller. An example is the limit cycle model of Galizia (2021).

3 Chebyshev solution, and is especially close the seventh order Chebyshev solution around the steady state. The Spline with 50 nodes is the only variant that matches the slope (ie. the perturbation solution) around the steady state well.

Finally we plot simulations of the resulting limit cycle in Figure 3. Note that the Spline solution with 50 nodes is visually indistinguishable from a very accurate approximation of the limit cycle[45] (not shown). The third order perturbation solution does very poor and is completely out of phase after one cycle. The third order Chebyshev-Galerkin solution is better, but is outperformed by a Spline with 5 nodes. The simulation of the order 7 Chebyshev solution stays pretty close to the accurate limit cycle, although it does not have sufficient curvature around the steady state as shown in Figure 2. The lack of curvature around the steady state does not affect the resulting deterministic limit cycle much, since the limit cycle consists of points far away from the steady state.

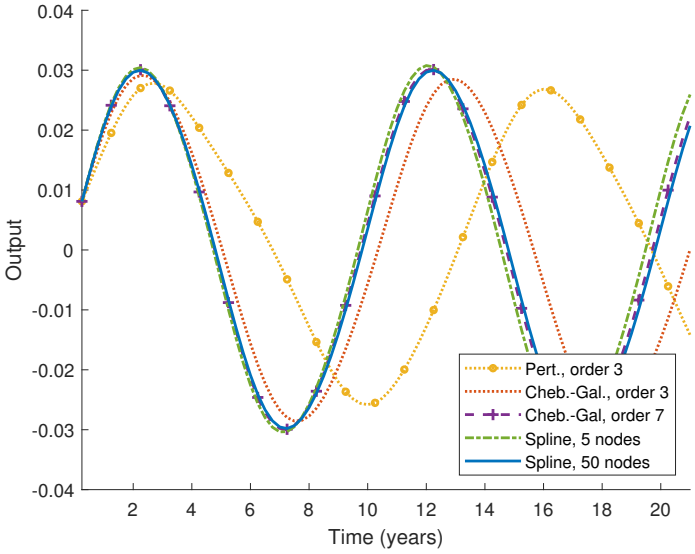
6 Conclusion

We have shown that projection methods are accurate, and relatively fast with computation times ranging between hundreds of a second for a simple model to at most a couple of seconds for a model with four state variables and two policy variables. Using the standardized algorithms of a toolbox also reduces coding time, which is the 'real bottleneck' for employing projection methods (Fernández-Villaverde et al., 2016, footnote ab, page 619).

Projection methods are very accurate and fast for a simple RBC model, which has two state variables and a near-linear solution. Projection methods

[45] We computed a very accurate numerical approximation of a limit cycle using the 'shooting method' (Kuznetsov, 2004) for the non-linear dynamic equations (33) to (35). The starting point of the limit cycle is restricted to be on the plane $K = K_{ss}$ in the 3 dimensional space $[K, \psi, \Lambda]$. All simulated series start from the initial point found with the shooting method.

Figure 3: Deterministic simulation
(Log deviation from steady state)



are especially valuable when more complex and highly non-linear models are poorly approximated with perturbation methods. For such models Splines with Time Iteration are a robust option, which can outperform more popular choices relying on Chebyshev polynomials.

We have shown two examples where splines outperform both complete Chebyshev polynomials and Smolyak's algorithm. One example was an RBC model with Habits in consumption and Investment Adjustment costs, which has four state variables and two policy variables. Another example where splines outperform Chebyshev polynomials is a highly non-linear model featuring a limit cycle. For that model the Smolyak algorithm does not converge to the solution, because it does not preserve the shape sufficiently. The third order perturbation solution of this model results in simulations that are completely out of phase after just one cycle. These results indicate that projection methods are a better choice than perturbation when accuracy matters.

Appendix A Gauss-Hermite quadrature

The general rule for Gaussian-Hermite approximation is:

$$\int_{-\infty}^{\infty} \exp\left(-z^2\right) g\left(z\right) dz \approx \sum_{j=1}^{J} \omega_j g\left(\zeta_j\right) \tag{41}$$

with Gauss-Hermite nodes $j = 1, ..., J$, roots ζ_j and weights ω_j (see Judd, 1998).

Assume we have a function $f\left(z_{t+1}, x\right)$ with exogenous variable z_{t+1}. This variable evolves according to (18) with standard normally distributed shocks $\epsilon_{t+1} \sim \mathcal{N}\left(0, 1\right)$. The expected value of this function is:

$$E_t f\left(z_{t+1}, x\right) = \int_{-\infty}^{\infty} f\left(\rho_z z_t + \sigma_z \epsilon_{t+1}, x\right) \frac{1}{\sqrt{2\pi}} \exp\left(-\epsilon_{t+1}^2/2\right) d\epsilon_{t+1} \tag{42}$$

To write (42) in the same form as (41) we need a change of variable $\phi = \frac{\epsilon_{t+1}}{\sqrt{2}}$, such that $\exp\left(-\epsilon_{t+1}^2/2\right) = \exp\left(-\phi^2\right)$. The approximation of the integral is:

$$\int_{-\infty}^{\infty} f\left(\rho_z z_t + \sigma_z \sqrt{2}\phi, x\right) \frac{1}{\sqrt{2\pi}} \exp\left(\phi\right) \sqrt{2} d\phi$$

$$\approx \sum_{j=1}^{J} \frac{\omega_j}{\sqrt{\pi}} f\left(\rho_z z_t + \sigma_z \sqrt{2}\zeta_j, x\right)$$

where the extra term$\sqrt{2}$ (before $d\phi$) follows from integration by substitution.

Appendix B Attracting Limit Cycle model

The model is an RBC model with external habits in the labor supply. There are possibly increasing returns to scale. There is monopolistic competition in the intermediate sector, but prices are fully flexible. Final goods producers are competitive and have a constant-returns-to-scale production function, using intermediate inputs only.

Consumers

Consumers maximize discounted utility:

$$\max E_0 \sum_{t=0}^{\infty} \beta^t \frac{\left(C_t - \psi_{t-1} L_t^{\theta}\right)^{1-\nu} - 1}{1-\nu} \tag{43}$$

where C is consumption, and L is labor supply. Crucial is the external habit variable ψ.

The household maximizes its utility subject to the real budget constraint:

$$C_t + K_{t+1} \leq w_t L_t + r_t K_t + \Pi_t + (1-\delta) K_t \tag{44}$$

where K_t is the capital stock at the beginning of the period, δ is the depreciation rate of capital plus fixed costs of capital, and Π_t are total profits of the intermediate producers. The rental prices for labor and capital are w and r, respectively.

The First Order Conditions (FOCs) with respect to hours worked L_t, consumption C_t and capital in the next period K_{t+1} result in:

$$\Lambda_t = \left(C_t - \psi_{t-1} L_t^{\theta}\right)^{-\nu} \tag{45}$$

$$w_t = \psi_{t-1} \theta L_t^{\theta-1} \tag{46}$$

$$\Lambda_t = \beta \Lambda_{t+1} \left(r_{t+1} + 1 - \delta\right) \tag{47}$$

where Λ is the marginal utility of consumption, which is equal to the shadow price of the budget constraint.

Final good Final goods producers use X_j inputs to produce Y:

$$Y \equiv \left(\int_0^1 {X_j}^{\phi} dj \right)^{\frac{1}{\phi}}$$

Cost minimization yields the demand for intermediate good X_j:

$$X_j = \left(\frac{p_j}{P}\right)^{-\frac{1}{1-\phi}} Y \tag{48}$$

where P is the aggregate price level $P \equiv \left(\int_0^1 {p_j}^{-\frac{\phi}{1-\phi}} dj \right)^{-\frac{1-\phi}{\phi}}$

Intermediate producer Real profits of intermediate producer j (omitting time indices) are:

$$\pi_j = \frac{p_j}{P} X_j - w l_j - r k_j$$

The labor input is l_j and the rented capital stock is k_j. The constraints for intermediate producers are demand (48), and the production constraint

$X_j = Zk_j^{\alpha} l_j^{\gamma}$ where Z is Total Factor Productivity. Maximization of profits and symmetry among producers yield the factor prices:

$$r_t = \phi \alpha K_t^{\alpha-1} L_t^{\gamma} \tag{49}$$

$$w_t = \phi \gamma K_t^{\alpha} L_t^{\gamma-1} \tag{50}$$

External habits in labor supply The labor supply equation (46) is:

$$w_t = \psi_{t-1} \theta L_t^{\theta-1}$$

where the habit variable ψ_{t-1} measures how much past working hours deviated from normal working hours L_{ss}. The current habit variable is a weighted average of the deviation from the norm in the current period $\zeta_1 \hat{L}_t + \zeta_3 \hat{L}_t^3$, and the habit variable in the previous period $\hat{\psi}_{t-1}$:

$$\hat{\psi}_t = (1-m)\left(\zeta_1 \hat{L}_t + \zeta_3 \hat{L}_t^3\right) + m\hat{\psi}_{t-1} \tag{51}$$

where m is the memory strength, which determines how fast habits adjust to current labor market conditions. Hats indicate log deviation from steady state.

References

Aruoba, S. B., J. Fernandez-Villaverde, and J. F. Rubio-Ramirez (2006). Comparing solution methods for dynamic equilibrium economies. *Journal of Economic dynamics and Control 30*(12), 2477–2508.

Beaudry, P., D. Galizia, and F. Portier (2020). Putting the cycle back into business cycle analysis. *American Economic Review 110*(1), 1–47.

Blanchard, O. J. and C. M. Kahn (1980). The solution of linear difference models under rational expectations. *Econometrica: Journal of the Econometric Society*, 1305–1311.

Caldara, D., J. Fernandez-Villaverde, J. F. Rubio-Ramirez, and W. Yao (2012). Computing dsge models with recursive preferences and stochastic volatility. *Review of Economic Dynamics 15*(2), 188–206.

De Boor, C. (1978). *A practical guide to splines*, Volume 27. springer-verlag New York.

Duineveld, S. (2021). Habits in the labor supply as a driver of the business cycle. Manuscript submitted for publication.

Fernández-Villaverde, J., G. Gordon, P. Guerrón-Quintana, and J. F. Rubio-Ramirez (2015). Nonlinear adventures at the zero lower bound. *Journal of Economic Dynamics and Control 57*, 182–204.

Fernández-Villaverde, J. and O. Levintal (2018). Solution methods for models with rare disasters. *Quantitative Economics 9*(2), 903–944.

Fernández-Villaverde, J., J. F. Rubio-Ramírez, and F. Schorfheide (2016). Solution and estimation methods for dsge models. In *Handbook of macroeconomics*, Volume 2, pp. 527–724. Elsevier.

Galizia, D. (2021). Saddle cycles: Solving rational expectations models featuring limit cycles (or chaos) using perturbation methods. *Quantitative Economics 12*(3), 869–901.

Gaspar, J. and K. L. Judd (1997). Solving large-scale rational-expectations models. *Macroeconomic Dynamics 1*(1), 45–75.

Guerrieri, L. and M. Iacoviello (2015). Occbin: A toolkit for solving dynamic models with occasionally binding constraints easily. *Journal of Monetary Economics 70*, 22–38.

Jaimovich, N. and S. Rebelo (2009). Can news about the future drive the business cycle? *American Economic Review 99*(4), 1097–1118.

Judd, K. L. (1992). Projection methods for solving aggregate growth models. *Journal of Economic theory 58*(2), 410–452.

Judd, K. L. (1998). *Numerical methods in economics*. MIT press.

Judd, K. L., F. Kubler, and K. Schmedders (2000). Computing equilibria in infinite-horizon finance economies: The case of one asset. *Journal of Economic Dynamics and Control 24*(5-7), 1047–1078.

Judd, K. L., F. Kubler, and K. Schmedders (2003). Computational methods for dynamic equilibria with heterogeneous agents. *Econometric Society Monographs 37*, 243–290.

Judd, K. L., L. Maliar, S. Maliar, and R. Valero (2014). Smolyak method for solving dynamic economic models: Lagrange interpolation, anisotropic grid and adaptive domain. *Journal of Economic Dynamics and Control 44*, 92–123.

Krueger, D. and F. Kubler (2004). Computing equilibrium in olg models with stochastic production. *Journal of Economic Dynamics and Control 28*(7), 1411–1436.

Kuznetsov, Y. A. (2004). *Elements of applied bifurcation theory*. Springer Science & Business Media.

Malin, B. A., D. Krueger, and F. Kubler (2011). Solving the multi-country real business cycle model using a smolyak-collocation method. *Journal of Economic Dynamics and Control 35*(2), 229–239.

McGrattan, E. R. (1996). Solving the stochastic growth model with a finite element method. *Journal of Economic Dynamics and Control 20*(1-3), 19–42.

Miranda, M. J. and P. G. Helmberger (1988). The effects of commodity price stabilization programs. *The American Economic Review*, 46–58.