A Method to Pre-compile Numerical Integrals When Solving Stochastic Dynamic Problems

Karolos Arapakis¹

Accepted: 1 November 2021 / Published online: 25 November 2021
© The Author(s) 2021

Abstract

We show how the interpolation step of numerical integration can be pre-compiled in partial equilibrium stochastic dynamic problems. We display the pre-compilation's sufficient conditions and document its speed gains using a consumption-savings model with a discrete labour choice, wage uncertainty and stochastic non-labour income.

Keywords Numerical methods · Numerical integration · Pre-compilation · Endogenous grid method · Value function iteration

1 Introduction

The solution of a stochastic dynamic problem often involves the evaluation of a large number of expectations via numerical integration, as weighted sums of interpolated functions. This paper shows how the interpolation step¹ of numerical integration can be pre-compiled, as a fixed cost paid up front. This procedure is of particular interest to researchers that want to estimate computationally expensive multidimensional partial equilibrium stochastic dynamic problems.

The pre-compilation procedure is not a new solution algorithm. It supplements existing solution algorithms and reduces the computational cost of numerical integration, without sacrificing accuracy. The speed increase is achieved by trading the cost of evaluating the laws of motion for the state variables, the cost of performing search algorithms (e.g. binary search) and the cost of relative weight calculations with the fixed cost paid up front, the memory requirement of pre-

¹ If the conditions of Sect. 5 are satisfied, this procedure can be applied to any continuous variable that requires interpolation, even if the problem is deterministic or has discretized random variables.

✉ Karolos Arapakis
karolos.arapakis.14@ucl.ac.uk

¹ Department of Economics, University College London, 30 Gordon St, London WC1H 0AX, UK

compiling numerical integrals and the cost of accessing the pre-compiled objects. An underlying assumption is that linear-interpolation is used. Under this assumption (the output of) search algorithms and the relative weights do not depend on the integrand.

An advantage of the this procedure is that it can supplement widely used solution algorithms such as the Value Function Iteration (*VFI*) and the Endogenous Grid Method (*G²EGM*), extended to account for non-convexities. The procedure's relative speed gains will be the largest for computationally expensive estimated high-dimensional problems, that feature multiple continuous state variables with complex laws of motion. Such models are common in the applied literature (See French & Jones, 2011; Blundell et al., 2016 for estimated non-convex partial equilibrium models that feature multiple dimensions).

To assess the pre-compilation procedure, we consider a non-convex² consumption-savings model with a discrete labour choice, wage uncertainty and stochastic non-labour income. We solve the model using *G²EGM* and *VFI* combined with linear interpolation and Gauss-Hermite quadrature integration. We consider both solution algorithms as *G²EGM* is not always applicable.³ *G²EGM* supplemented with the pre-compilation procedure is approximately 3 times faster than standard *G²EGM*. When in addition to the pre-compilation procedure, the value and policy functions are parametrically approximated by a piecewise multilinear polynomial, the procedure reduces the solution time by an additional 40%.

This paper is related to the literature that aims to reduce the cost of numerical integration via pre-compilations. Druedahl (2020) reduces the cost of numerical integration using a vectorized interpolation scheme that mitigates the use of search algorithms, without requiring a pre-compilation step. Judd et al. (2017) utilize parametric approximations and propose a method that pre-computes conditional expectations, subject to the functional form of the approximating function.⁴ The procedure in this paper pre-compiles the interpolation step of numerical integration, without requiring functional form assumptions.

The paper is structured as follows. Section 2 presents a simple application of the proposed procedure. Section 3 presents the non-convex model. Section 4 presents results on performance and accuracy. Section 5 presents the procedure's sufficient conditions and discusses its limitations. Section 6 extends the procedure to piecewise multilinear polynomial parametric approximations. Section 7 concludes.

² As the problem considered is non-convex, a process that eliminates non-global optima is required. To address this, the *EGM* solution algorithm of Carroll (2006) is complemented with the upper envelope of Druedahl and Jørgensen (2017). See Fella (2014) and Iskhakov et al. (2017) for alternative upper envelopes.

³ See Druedahl and Jørgensen (2017) for the necessary and sufficient conditions.

⁴ For example if a function $V(k')$ is approximated by a linear polynomial, its expectation is $\mathbb{E}[V(k')] \approx \mathbb{E}[a_0 + a_1 k'] = a_0 + a_1 \mathbb{E}[k'] = a_0 + a_1 \mu_k$, where μ_k does not depend on V and is computed up front. When using local curve fitting however (e.g. a collection of piecewise polynomial approximations), search algorithms might still be required. Section 6 discusses how these search algorithms can be avoided.

2 A Simple Univariate-Continuous Example

To display the intuition behind the pre-compilation procedure, consider the example of a researcher that wants to evaluate the continuation value $E[V(m')] = E[V(Ra' + y')]$, for N different values of savings $\{a'_i\}_{i=1}^N$. In this example R is the exogenous gross returns and y' is a standard log-normal *iid* income stock. The function V potentially has no closed form and the researcher only knows the values of V on a finite set of points $\{m'_j\}_{j=1}^n$, where $m'_1 < \dots < m'_n$. Conditional on a value $a' \in \{a'_i\}_{i=1}^N$, he can evaluate each expectation by numerical integration as a weighted sum of the interpolated function V .

$$\begin{aligned}
 E[V(m')] &= \int V(Ra' + y')f(y')dy' \\
 &\approx \sum_{j=1}^J w_j V(Ra' + y_j) \approx \sum_{j=1}^J w_j \hat{V}(Ra' + y_j) \\
 &= \sum_{j=1}^J w_j \left[\frac{\overbrace{m_{k_j+1} - (Ra' + y_j)}^{\tilde{w}_{j,0}}}{m_{k_j+1} - m_{k_j}} v_{k_j} + \frac{\overbrace{(Ra' + y_j) - m_{k_j}}^{\tilde{w}_{j,1}}}{m_{k_j+1} - m_{k_j}} v_{k_j+1} \right] \\
 &= \sum_{j=1}^J \sum_{p=0}^1 w_j \tilde{w}_{j,p} v_{k_j+p} = \sum_{j=1}^J \sum_{p=0}^1 w_{j,p}^* v_{k_j+p}
 \end{aligned} \tag{1}$$

Equation 1 algebraically displays how numerical integration works. The integrand is evaluated at finitely many integration nodes and a weighted sum of these is used to approximate the true integral. In this example $\{w_j\}_{j=0}^J$ and $\{Ra' + y_j\}_{j=0}^J$ correspond to the integration weights and nodes respectively.⁵ Interpolated values are denoted by $\hat{V}$ and define $v = (v_1, \dots, v_n)$, where $v_j = V(m_j)$. To obtain $\hat{V}(Ra' + y_j)$, it is standard practise to use a search algorithm (e.g. binary search) to locate an index k_j such that $m_{k_j} \leq Ra' + y_j < m_{k_j+1}$, for each discretised node j . Then, linear interpolation can be used to evaluate $\hat{V}(Ra' + y_j)$ as a convex combination of v_{k_j} and v_{k_j+1} . Notice however that neither the search algorithm nor the relative weight calculations for $\{\tilde{w}_{j,0}, \tilde{w}_{j,1}\}$ require any information on the function V . They only require knowledge of the finite sets $\{a'_i\}_{i=1}^N$, $\{y_j\}_{j=1}^J$, $\{m'_j\}_{j=1}^n$ and the law of motion of the state variable ($m' = Ra' + y'$). Thus, both can be pre-compiled (as a fixed cost paid up front) and then, expectations can be computed for *any* function V ,

⁵ A common assumption is that $w_j = \tilde{w}_j / \sqrt{\pi}$ and $y_j = \exp(\mu_y + \sqrt{2}\sigma_y \tilde{x}_j)$, where $\{\tilde{w}_j, \tilde{x}_j\}_{j=1}^J$ are the corresponding Gauss-Hermite quadrature weights and nodes, see Judd (1998). Extrapolations are ignored in this example to facilitate exposition.

circumventing law of motion evaluations, the use of search algorithms and relative weight calculations.⁶

To assess the speed gains that the researcher will get by pre-compiling numerical integrals in this example, suppose that he now wants to evaluate a sequence of functions $\{V_\tau\}_{\tau=1}^T$. Algorithm 1 presents the standard expectation evaluation method.⁷ Algorithm 2 presents a shortened version, once search algorithms and relative weights have been pre-compiled and Algorithm 3 displays the pre-compilation step. The researcher can either use Algorithm 1 T times or Algorithm 3 once and then Algorithm 2 T times. The difference between the two approaches is that the latter trades the cost of evaluating the law of motion, performing search algorithms and relative weight calculations (lines 4–6 in Algorithm 1) with the cost of calling elements from the pre-compiled matrices (line 4 in Algorithm 2), the memory requirement of storing these matrices and the cost of performing Algorithm 3 once up front. Also, note that the expectation evaluation steps (lines 4–7 in Algorithm 1 vs lines 5–7 in Algorithm 3 and line 5 in Algorithm 2) are identical and thus results and approximation errors will be identical as well.

Algorithm 1 Standard Expectation Evaluation.

inputs:

Vectors: $\{m'_k\}_{k=1}^n$, $\{a'_i\}_{i=1}^N$, $\{y_j\}_{j=1}^J$, $\{w_j\}_{j=1}^J$, $\{v_k\}_{k=1}^n$

Matrices:

Parameters: N, J, R

```

1:  $EV = \text{zeros}(N, 1)$ 
2: for each  $i \in \{1, \dots, N\}$  do
3:   for each  $j \in \{1, \dots, J\}$  do
4:      $m' = Ra'_i + y_j$ 
5:      $k_j = \text{Binarysearch}(m', \{m'_k\}_{k=1}^n)$ 
6:      $\tilde{w} = (m' - m_{k_j}) / (m_{k_j+1} - m_{k_j})$ 
7:      $EV[i] += w_j \cdot (v_{k_j} + \tilde{w} \cdot (v_{k_j+1} - v_{k_j}))$ 
8:   end for (j)
9: end for (i)
10: Return:  $EV$ 

```

⁶ In linear algebra form, define $w^* = (w_{1,0}^*, \dots, w_{J,0}^*, w_{1,1}^*, \dots, w_{J,1}^*)$, $\kappa = (k_1, \dots, k_J, k_1 + 1, \dots, k_J + 1)$ and $B_{(2J \times n)}$ as a binary matrix where $B(i, j) = 1$ if $\kappa_i = j$ and zero otherwise. Then, $\sum_{j=1}^J \sum_{p=0}^1 w_{j,p}^* v_{k_j+p} = w^* \cdot B \cdot v' = \mathbb{G} \cdot v'$, where $\mathbb{G}$ is pre-compiled.

⁷ In the *Binarysearch* function, the first argument denotes the scalar input and the second is the monotonic grid that will be searched.

Algorithm 2 Expectation Evaluation under Pre-Compilation.**inputs:**Vectors: $\{w_j\}_{j=1}^J, \{v_k\}_{k=1}^n$ Matrices: I, W Parameters: N, J

```

1:  $EV = \text{zeros}(N, 1)$ 
2: for each  $i \in \{1, \dots, N\}$  do
3:   for each  $j \in \{1, \dots, J\}$  do
4:      $k_j = I[i, j]$  ;  $\tilde{w} = W[i, j]$ 
5:      $EV[i] += w_j \cdot (v_{k_j} + \tilde{w} \cdot (v_{k_j+1} - v_{k_j}))$ 
6:   end for (j)
7: end for (i)
8: Return:  $EV$ 

```

Algorithm 3 Pre-Compilation Step.**inputs:**Vectors: $\{m'_k\}_{k=1}^n, \{a'_i\}_{i=1}^N, \{y_j\}_{j=1}^J$

Matrices:

Parameters: N, J, R

```

1:  $I = \text{zeros}(N, J)$ 
2:  $W = \text{zeros}(N, J)$ 
3: for each  $i \in \{1, \dots, N\}$  do
4:   for each  $j \in \{1, \dots, J\}$  do
5:      $m' = Ra'_i + y_j$ 
6:      $k_j = \text{Binarysearch}(m', \{m'_k\}_{k=1}^n)$ 
7:      $\tilde{w} = (m' - m_{k_j}) / (m_{k_j+1} - m_{k_j})$ 
8:      $Index[i, j] = k_j$ 
9:      $Weight[i, j] = \tilde{w}$ 
10:   end for (j)
11: end for (i)
12: Return:  $Index, Weight$ 

```

Figure 1 is showing the solution time needed to evaluate the sequence of integrands $\{V_\tau\}_{\tau=1}^T$ with Algorithm 1 (Without Pre-Compilation) and the combination of Algorithms 3 and 2 (With Pre-Compilation), for different values of the parameters T, N, n, J . The time difference is not affected by the grid-size of the state space n , but it is increasing with respect to the parameters T, N, J that affect the total number of law of motion evaluations, search algorithms and relative weight calculations. Thus, in computationally expensive multi-dimensional problems, that require the evaluation of a large number of expectations and have state variables with complex laws of motion, the proposed procedure will provide large gains.

Fig. 1 Speed without and with Pre-compilations. *Notes* The sequence of integrands $\{V_\tau\}_{\tau=1}^T$ is generated by $V_\tau(x) = (x^{-3-a^\tau})/(-3-a^\tau)$, where $a^\tau \sim U(0, 1)$, $\forall \tau \in \{1, \dots, T\}$. Solution times (medians) were obtained using Julia's *benchmarktools* package. In each subplot, the parameters that do not vary are held fixed at $T = 200$, $n = 100$, $N = 100$, $J = 8$

3 The Non-convex Model

To assess the performance of the proposed procedure in a stochastic dynamic problem, consider the following non-convex model. An infinitely lived agent maximizes his expected lifetime utility by making a consumption (c)/savings (a') decision and a discrete labour supply (h) decision every period. To make these decisions, he is taking into account his non-labour cash on hand (m) and his productivity (e), that is following an AR(1) process in its logarithm. The recursive formulation of the problem is

$$\begin{aligned}
 V(m, e) &= \max_{c, a', h} \left\{ u(c, h) + \beta \mathbb{E} [V(m', e')] \right\} \\
 \text{st. } m' &= Ra' + y' \\
 m &= c + a' - weh \\
 \log(e') &= \rho \cdot \log(e) + u'_e \\
 y' &= \exp(u'_y) \\
 u'_j &\sim \mathcal{N}(\mu_j, \sigma_j^2), \text{ for } j \in \{e, y\}
 \end{aligned}$$

where R is the exogenous returns, w is the exogenous wage rate, ρ is a correlation coefficient and y is a log-normal *iid* non-labour income shock. Next periods variables are denoted with a prime. The grid sizes are $\#(m) = 300$, $\#(e) = 150$, $\#(h) = 3$, $\#(a') = 300$, where the function $\#(x)$ returns the number of nodes in the array x . The random variables for productivity and non-labour income are integrated using a Gauss-Hermite quadrature with 10 nodes. Interpolations are performed by multidimensional linear interpolation and to increase accuracy (1) the value function is transformed by $f(x) = -1/x$ (to avoid infinities and reduce the function's curvature near the origin) and (2) next period's functions are interpolated for every choice of labour supply and the highest utility option is selected. The convergence tolerance level is set to 10^{-8} , measured by the Euclidean distance of the continuation value matrix (See [Appendix](#), Part C). Detailed information on the grid construction is presented in the [Appendix](#). The utility function (2) is additively separable between consumption and leisure and the parameter values are set to $\beta = 0.98$, $\gamma = 4.0$, $\sigma = (2.5, 1.5)$, $\rho = 0.977$, $\sigma_y = 0.1$, $\sigma_e = 0.01$, $R = 1.03$, $w = 1.1$, $\chi = 0.5$, $\mu_e = 0$, $\mu_y = \ln(0.9)$.

$$u(c, 1 - h) = \frac{c^{1-\sigma}}{1-\sigma} + \chi \frac{(1-h)^{1-\gamma}}{1-\gamma} \quad (2)$$

The pre-compilation Algorithm and a brief outline of the solution Algorithm are located in the [Appendix](#).

4 Performance and Accuracy

This subsection presents performance and accuracy results for a stochastic simulation of $N = 100,000$ individuals for $T = 30$ time periods.⁸ To measure accuracy, following (Judd, 1992; Santos, 1999) and (Santos, 2000) the common logarithm of the relative absolute Euler-errors ($\mathfrak{E}$) is considered. As the Euler-Equation does not have to be true on the constrained region, I impose a lower bound on savings, $\underline{a} = 0.002$.

⁸ In the initial period, every individual starts from the unconditional mean of the productivity distribution and a random draw from the distribution of y for non-labour cash on hand.

Table 1 Algorithm performance

Algorithm	$\sigma = 2.5$		$\sigma = 1.5$	
	CPU time	$\mathfrak{E}$	CPU Time	$\mathfrak{E}$
G^2EGM	325.30 s	− 5.56	318.44 s	− 5.81
G^2EGM + Druedahl (2020)	129.24 s	− 5.56	127.84 s	− 5.81
G^2EGM +pre-compilation	113.57 s	− 5.56	109.51 s	− 5.81
VFI	951.09 s	− 1.68	607.54 s	− 1.72
VFI + Druedahl (2020)	697.14 s	− 1.68	406.20 s	− 1.72
VFI +pre-compilation	647.32 s	− 1.68	387.60 s	− 1.72

Results were computed using an AMD Ryzen 9 3950X 16-Core Processor (single threaded implementation). The code was written and executed using Julia (JuliaPro-1.5.1-1). Solution times (medians) were obtained using Julia's *benchmarktools* package

$$\mathfrak{E} = \frac{\sum_{i=1}^N \sum_1^T \mathbb{I}[a_{i,t+1} > \underline{a}] error_{i,t}}{\sum_{i=1}^N \sum_1^T \mathbb{I}[a_{i,t+1} > \underline{a}]}$$

$$error_{i,t} = \log_{10} \left(\left| c_{i,t} - \left(\beta(1+r) \mathbb{E}[c_{i,t+1}^-] \right)^{-\frac{1}{\sigma}} \right| / c_{it} \right)$$

The G^2EGM requires 747 iterations to converge with relative absolute Euler-errors of $\mathfrak{E} = -5.56$ for $\sigma = 2.5$ and 736 iterations with relative absolute Euler-errors of $\mathfrak{E} = -5.81$ for $\sigma = 1.5$. The VFI requires 822 iterations to converge with relative absolute Euler-errors of $\mathfrak{E} = -1.68$ for $\sigma = 2.5$ and 762 iterations with relative absolute Euler-errors of $\mathfrak{E} = -1.72$ for $\sigma = 1.5$. Table 1 is displaying the speed and accuracy results for three different approaches (using both G^2EGM and VFI). The first is the standard solution algorithm, the second is the algorithm supplemented with the vectorized interpolation scheme of Druedahl (2020) and the third is the algorithm supplemented with the pre-compilation procedure.

Standard G^2EGM solves the problem in 325.30 s for $\sigma = 2.5$ and 318.44 s for $\sigma = 1.5$, while standard VFI solves the problem in 951.09 s for $\sigma = 2.5$ and 607.54 s for $\sigma = 1.5$. Using the pre-compilation procedure makes the solution algorithm roughly 2.8 times faster for G^2EGM (and 1.5 for VFI). Note that the model of Sect. 3 is low-dimensional and each state variable has a simple law of motion. We expect that the pre-compilation procedure will provide larger relative speed gains in high-dimensional problems, where the laws of motion for the state variables feature complex functions. Examples of such problems are retirement problems with endogenous labour supply, that model the institutional features of social security. These problems often include state variables with laws of motion that mimic aspects of the tax and benefit system and can take days to estimate. Thus, the relative speed gains reported in this Section should be treated as a lower bound for larger and more complex models.

Using the the vectorized interpolation scheme of Druedahl (2020) makes the solution algorithm roughly 2.6 times faster for G^2EGM (and 1.4 for VFI). This

Fig. 2 Common logarithm of the relative absolute Euler-errors ($\mathfrak{E}$). *Notes* This figure is showing the accuracy of G^2EGM and VFI for various grid sizes. A grid size scale of x implies that the grid sizes are $x \cdot \#(z)$, for $z \in \{m, a, e\}$

method is more generally applicable (partial and general equilibrium) and does not impose a memory requirement. We expect that this method will likely outperform the pre-compilation procedure in problems without multiple continuous dimensions and where the laws of motion for the state variables are not computationally expensive to evaluate.

All three approaches generate the same results and thus the same values for the relative absolute Euler-errors. Figure 2 is showing how the relative absolute Euler-errors evolve as a function of the grid size.

5 Discussion

The proposed procedure can be applied to any finite-dimensional, discrete-continuous stochastic dynamic problem as long as the four conditions of *Proposition 1* are satisfied. In practise, multiple solution algorithms such as the standard VFI and the G^2EGM ⁹ satisfy these conditions, in partial equilibrium problems.

Proposition 1 *If (i) the state space grid, (ii) the choice grid, (iii) the shock grid and iv) the law of motion of the state variables, are all exogenous, then numerical integrals can be pre-compiled.*

Proof See [Appendix](#) □

The intuition is that that we need sufficient information to determine which points will be interpolated able to perform the operations up front. An underlying assumption behind *Proposition 1* is that multidimensional linear interpolation is used. A weaker version of this proposition can be extended to alternative

⁹ In non-convex problems the upper envelope of Druedahl and Jørgensen (2017) relies on an exogenous common resource grid, while in convex problems (EGM) the upper envelope can be replaced by an efficient procedure that regularizes the endogenous grid, utilizing that both endogenous and exogenous resource grids are monotonic.

interpolation schemes, however relative weights may depend on the integrand and will have to be evaluated during the model's solution.

It is important to emphasize that the procedure will pre-compile the interpolation step of numerical integrals associated with computing the continuation values. It will not pre-compile every interpolation that will occur during a solution algorithm (e.g. interpolations that take place in the upper envelope step to eliminate non-global maxima).

When a numerical integral is pre-compiled, objects that are required for its evaluation, such as indices and relative weights have already been calculated and are stored in arrays. Using these arrays imposes two limitations. The memory access cost and the the RAM memory they occupy, both of which are increasing with the size of the arrays. An approximate upper bound (in *megabytes*) for the RAM memory requirement of these arrays can be evaluated by

$$RAM \approx \sum_{\lambda=1}^{\Lambda} \#(I_{\lambda}) \cdot \left(\frac{8 \cdot 2}{10^6} \right) \text{ MB} \quad (3)$$

where the function $\#(x)$ returns the number of cells in the array x and $\{I_{\lambda}\}_{\lambda=1}^{\Lambda}$ is a set that contains the arrays that store the indices. For every array, we count the number of cells and then adjust for 8 *bytes* per cell,¹⁰ times two (one array for the indices and one for the relative weights). The denominator changes the units from *bytes* to *megabytes*. In problems with multiple continuous state-variables and where multiple choice, state and shock variables enter their laws of motion, the memory requirement will be larger. In the [Appendix](#), we consider a six state-variable lifecycle retirement problem and show that the pre-compiled arrays require at most 9.1MB or 0.0091GB, a small cost compared to the memory requirement of the arrays that store the value and policy functions.

6 Extension to Parametric Approximations

Judd et al. (2017) utilize parametric approximations and display how to pre-compute conditional expectations, subject to the functional form of an approximating polynomial function. When using local curve fitting however (e.g. a collection of piecewise polynomial approximations), in the general case, law of motion evaluations and search algorithms are still required to locate the hypercube that contains the point to be evaluated and the surrounding vertices are used to calculate the approximating function's coefficients.

This Section presents how the known link between multilinear polynomials and multidimensional linear interpolation can be utilized to pre-compile conditional expectations and further increase the speed of a solution algorithm. Parametrically approximating the value function of the model in Sect. 3, using a piecewise multilinear polynomial on the rectangular mesh $[m_1, \dots, m_{\#(m)}] \times [e_1, \dots, e_{\#(e)}]$ yields

¹⁰ Each n -cell array requires approximately $8n$ bytes for double precision and $4n$ bytes for single precision. Thus, the formula provides an approximate upper bound.

Table 2 Algorithm performance

Algorithm	$\sigma = 2.5$		$\sigma = 1.5$	
	CPU Time	$\mathfrak{E}$	CPU time	$\mathfrak{E}$
G^2EGM + Judd et al. (2017)	15.80 s	-4.10	15.40 s	-4.13
G^2EGM + Judd et al. (2017) +Pre-compilation	9.37 s	-4.10	9.05 s	-4.13

Results were computed using an AMD Ryzen 9 3950X 16-Core Processor (single threaded implementation). The code was written and executed using Julia (JuliaPro-1.5.1-1). Solution times (medians) were obtained using Julia's *benchmarktools* package

$$\begin{aligned}
 \mathbb{E}[V(m', e')] &\approx \mathbb{E}[\hat{V}_{[m_z, m_{z+1}] \times [e_s, e_{s+1}]}(m', e'; a)] \\
 &= \mathbb{E}[a_0 + a_1 m' + a_2 e' + a_3 e' m'] \\
 &= a_0 + a_1 \mathbb{E}[m'] + a_2 \mathbb{E}[e'] + a_3 \mathbb{E}[e'] \mathbb{E}[m'] \\
 &= a_0 + a_1 (Ra' + \mathbb{E}[y']) + a_2 e^\rho \mathbb{E}[\exp(u'_e)] + a_3 e^\rho \mathbb{E}[\exp(u'_e)] (Ra' + \mathbb{E}[y'])
 \end{aligned} \tag{4}$$

where $\{a_j\}_{j=0}^3$ are the coefficients of the polynomial. Notice however, that the resulting function is a deterministic multilinear polynomial and it is known that it will deliver the same value as bilinear interpolation on the point $(Ra' + \mathbb{E}[y'], e^\rho \mathbb{E}[\exp(u'_e)])$. Thus, instead of calculating the $\{a_j\}_{j=0}^3$ coefficients during the model's iterative solution cycle,¹¹ we can use the pre-compiled bilinear interpolation and obtain the same results.

Approximating the value and consumption functions of the problem in Sect. 3 using a piecewise multilinear polynomial, G^2EGM requires 748 iterations to converge with relative absolute Euler-errors of $\mathfrak{E} = -4.10$ for $\sigma = 2.5$ and 734 iterations with relative absolute Euler-errors of $\mathfrak{E} = -4.13$ for $\sigma = 1.5$. Table 2 is displaying the speed and accuracy¹² results for the two different approaches.

Using the pre-compilation procedure reduces the running time by about 40%. The relative time difference is smaller than the one reported in Sect. 4, as in this case the loops occur over savings and productivity (not over the shocks) and thus there are less law of motion evaluations, search algorithms and coefficient (instead of relative weight) evaluations. Figure 3 is showing a comparison of the solutions with and without parametric approximations.

For further error analysis when using parametric approximations, the reader is advised to see Judd et al. (2017), where this topic is assessed extensively. Unlike this paper, they use an alternative procedure where inferences about the problem's accuracy are not contaminated by numerical integration error during the accuracy evaluation algorithm.

¹¹ Once the hypercube that contains (m', e') is located, the coefficients are calculated by imposing that the parametric approximation is equal to the integrand on the hypercube's vertices.

¹² We find that for both $\sigma = 2.5$ and $\sigma = 1.5$, increasing the grid size has almost no effect on the relative absolute Euler-errors.

Fig. 3 Solutions with and without parametric approximations. *Notes* This figure is showing the value and policy functions with and without parametric approximations. Top panels are created using the values $h = 0.2$, $e = (0.38, 1.18)$, $\sigma = 2.5$, bottom panels are created using the values $h = 0.2$, $e = (0.38, 1.18)$, $\sigma = 1.5$

7 Conclusions

This paper presents a parsimonious pre-compilation procedure for partial equilibrium stochastic dynamic problems. The pre-compilations's speed gains were assessed using a consumption-savings model with a discrete labour choice, wage uncertainty and stochastic non-labour income. The procedure's relative speed gains will be larger for computationally expensive estimated high-dimensional problems, that feature multiple continuous state variables with complex laws of motion. This procedure should assist researchers in estimating and performing counterfactual analysis using high-dimensional models.

Appendix

Part A. Grid construction

The exogenous choice and state space grids $z \in \{m, e, a', h\}$ are constructed recursively according to the rule $z_i = z_{i-1} + (z_{Nz} - z_{i-1}) / (Nz - 1)^{\gamma_z}$, for $2 \leq i < Nz$, where

1. $g^m = \{m_i\}_{i=1}^{Nm}$, $m_1 = 0.0$, $m_{Nm} = 50.0$, $Nm = 300$, $\gamma_m = 1.05$
2. $g^e = \{e_i\}_{i=1}^{Ne}$, $e_1 = 0.01$, $e_{Ne} = 2.5$, $Ne = 100$, $\gamma_e = 1.10$
3. $g^a = \{a_i\}_{i=1}^{Na}$, $a_1 = 0.0$, $a_{Na} = 50.0$, $Na = 300$, $\gamma_a = 1.10$
4. $g^h = \{h_i\}_{i=1}^{Nh}$, $h_1 = 0.0$, $h_{Nh} = 0.4$, $Nh = 3$, $\gamma_h = 1.00$

The grids for the shocks $z \in \{u'_y, u'_e\}$ and their weights are constructed according to $g_j^z = \mu_z + \sqrt{2}\sigma_z \bar{x}_j$ and $w_j = \bar{w}_j / \sqrt{\pi}$, for $1 \leq j \leq N_z$, where $\{\bar{w}_j, \bar{x}_j\}_{j=1}^{N_z}$ are the Gauss-Hermite quadrature weights and nodes, see Judd (1998). We set $N_z = 10$ for $z \in \{u'_y, u'_e\}$.

Part B. Proof of Proposition 1

Define the state $\{x_j\}_{j=1}^{n_x}$, choice $\{d_j\}_{j=1}^{n_d}$ and shock $\{s_j\}_{j=1}^{n_s}$ variables, which are discretized and denoted by $\{g^{x_j}\}_{j=1}^{n_x}$, $\{g^{d_j}\}_{j=1}^{n_d}$ and $\{g^{s_j}\}_{j=1}^{n_s}$ respectively. Also define the n_x -fold Cartesian product $g^x = \times_{j=1}^{n_x} g^{x_j}$, the n_d -fold Cartesian product $g^d = \times_{j=1}^{n_d} g^{d_j}$ and the n_s -fold Cartesian product $g^s = \times_{j=1}^{n_s} g^{s_j}$. Also denote by L^{x_j} the law of motion for variable x_j , the domain of which is a subset of the state-choice-shock variables. Denote by $g^{(x,d,s)_{x_j}}$ the Cartesian product that corresponds to the discretised domain of L^{x_j} .

Suppose that there is at least one numerical integral that cannot be pre-compiled. Then, there exists a state variable x_λ for some $\lambda \in \{1, \dots, n_x\}$ that cannot be interpolated for some exogenous state-choice-shock combination $(x, d, s) \in g^{(x,d,s)_{x_\lambda}}$. However, as the law of motion L^{x_λ} is also exogenous, we can perform the interpolation, using $L^{x_\lambda}(x, d, s)$ and g^x , which leads to a contradiction.

Part C. Brief outline of the solution algorithms

VFI :

1. Start with an initial guess on the value function, $V = u(m, h)$.
2. Compute the continuation value $\mathbb{E}[V(m', e')|a', e] = q^V(a', e)$ for each grid point of savings and productivity.
3. Do grid search to locate the global optima.
4. If the tolerance is satisfied stop, else go back to step 2.

G^2EGM :

1. Start with an initial guess on the policy and value functions, $c = m$ and $V = u(m, h)$.
2. Compute the continuation value $\mathbb{E}[V(m', e')|a', e] = q^V(a', e)$ and continuation marginal utility $\mathbb{E}[u'(c(m', e'))|a', e] = q^u(a', e)$ for each grid point of savings and productivity.
3. Use the inverted Euler-Equation and the budget constraint to back out consumption and non-labour cash on hand. Then, use the upper envelope to eliminate the non-global optima.
4. If the tolerance is satisfied stop, else go back to step 2.

Part D. Non-Convex Model Pre-Compilation Algorithm

Algorithm 4 Pre-Compilation Step - Illustrative Model

inputs:Vectors: $g^m, g^e, g^{a'}, g^{u_y}, g^{u_e}$

Matrices:

Parameters: Na, Ne, Nx, R, ρ

```

1:  $I_m = \text{zeros}(Na, Nx)$  ;  $W_m = \text{zeros}(Na, Nx)$ 
2:  $I_e = \text{zeros}(Ne, Nx)$  ;  $W_e = \text{zeros}(Ne, Nx)$ 
3:
4: for  $i = 1 : Na$  do
5:   for  $j = 1 : Nx$  do
6:
7:      $m' = Rg_i^{a'} + \exp(g_j^{u_y})$ 
8:      $k_m = \text{Binarysearch}(m', g^m)$ 
9:      $w_m = (m' - g_{k_m}^m) / (g_{k_m+1}^m - g_{k_m}^m)$ 
10:     $I_m[i, j] = k_m$ 
11:     $W_m[i, j] = w_m$ 
12:
13:   end for
14: end for
15:
16: for  $i = 1 : Ne$  do
17:   for  $j = 1 : Nx$  do
18:
19:      $e' = \exp(\rho \cdot \log(g_i^e) + g_j^{u_e})$ 
20:      $k_e = \text{Binarysearch}(e', g^e)$ 
21:      $w_e = (e' - g_{k_e}^e) / (g_{k_e+1}^e - g_{k_e}^e)$ 
22:      $I_e[i, j] = k_e$ 
23:      $W_e[i, j] = w_e$ 
24:
25:   end for
26: end for
27:
28: Return:  $I_m, W_m, I_e, W_e$ 

```

Algorithm 5 Pre-Compilation Step - Illustrative Model (With parametric approx.)**inputs:**Vectors: $g^m, g^e, g^{a'}$

Matrices:

Parameters: $Na, Ne, R, \rho, \mathbb{E}[\exp(u'_e)], \mathbb{E}[\exp(u'_y)]$

```

1:  $I_m = \text{zeros}(Na)$  ;  $W_m = \text{zeros}(Na)$ 
2:  $I_e = \text{zeros}(Ne)$  ;  $W_e = \text{zeros}(Ne)$ 
3:
4: for  $i = 1 : Na$  do
5:
6:    $m' = Rg_i^{a'} + \mathbb{E}[\exp(u'_y)]$ 
7:    $k_m = \text{Binarysearch}(m', g^m)$ 
8:    $w_m = (m' - g_{k_m}^m) / (g_{k_m+1}^m - g_{k_m}^m)$ 
9:    $I_m[i] = k_m$ 
10:   $W_m[i] = w_m$ 
11:
12: end for
13:
14: for  $i = 1 : Ne$  do
15:
16:    $e' = (g_i^e)^\rho \mathbb{E}[\exp(u'_e)]$ 
17:    $k_e = \text{Binarysearch}(e', g^e)$ 
18:    $w_e = (e' - g_{k_e}^e) / (g_{k_e+1}^e - g_{k_e}^e)$ 
19:    $I_e[i] = k_e$ 
20:    $W_e[i] = w_e$ 
21:
22: end for
23:
24: Return:  $I_m, W_m, I_e, W_e$ 

```

Part E. Life-cycle retirement problem RAM requirement

Consider a six state-variable retirement problem. This problem is a simplified version of French and Jones (2011). The problem has multiple continuous state-variables, with fairly complex laws of motion. The agent works from age 21 to age 65, retires at age 66 and dies at age 100. The variable m is non-labour cash on hand, g is health status, s is medical expenditures, w is the wage rate, $AIME$ is average index monthly earnings and t age. There are four continuous variables $m, s, w, AIME$, the variable $t \in \{21, \dots, 100\}$ is discrete and $g \in \{good, bad\}$ is binary. For simplicity, I assume that the social security system has a replacement rate of 30% and abstract from specifying model restrictions that are not relevant to the memory calculation. The recursive formulation for the worker and retiree problems are

- $t \leq 65$ (*worker*)

$$V(m, g, s, w, AIME, t) = \max_{c, a', h} \left\{ u(c, h, g) + \beta \mathbb{E} \left[V(m', g', s', w', AIME', t+1) \right] \right\}$$

$$\text{st. } m' = Ra'$$

$$m = c + a' - wh$$

$$g' = \Pi^g(g, t)$$

$$s' = \Pi^s(g, s, t, u'_s), \quad u'_s \sim \mathcal{N}(0, \sigma_s^2)$$

$$w' = \Pi^w(w, t, u'_w), \quad u'_w \sim \mathcal{N}(0, \sigma_w^2)$$

$$AIME' = \begin{cases} AIME + \frac{wh}{35} & \text{if } t \leq 55 \\ AIME + \max\{0, \frac{wh - AIME}{35}\} & \text{otherwise} \end{cases}$$

- $t \geq 66$ (*retiree*)

$$V(m, g, s, AIME, t) = \max_{c, a'} \left\{ u(c, 1, g) + \beta \mathbb{E} \left[V(m', g', s', AIME, t+1) \right] \right\}$$

$$\text{st. } m' = Ra' + 0.3AIME$$

$$m = c + a'$$

$$g' = \Pi^g(g, t)$$

$$s' = \Pi^s(g, s, t, u'_s), \quad u'_s \sim \mathcal{N}(0, \sigma_s^2)$$

The approximate upper bound for the RAM requirement is the sum of the RAM requirement of the worker and the retiree problems

$$RAM^{worker} \approx \left[\#(a') + 2\#(s)65\#(u_s) + \#(w)65\#(u_w) + \#(w)\#(h)\#(AIME)65 \right].$$

$$\frac{8 \cdot 2}{1000^2} \text{ MB}$$

$$= 8.94 \text{ MB}$$

$$RAM^{retiree} \approx \left[\#(a')\#(AIME) + 2\#(s)35\#(x_s) \right] \cdot \frac{8 \cdot 2}{1000^2} \text{ MB}$$

$$= 0.25 \text{ MB}$$

where u_w and u_s denote the Gauss-Hermite nodes for the wage and medical spending innovations. I assume $\#(m) = 70$, $\#(a') = 70$, $\#(w) = 20$, $\#(s) = 20$, $\#(u_w) = 8$, $\#(u_s) = 8$, $\#(h) = 20$, $\#(AIME) = 20$. This problem's memory

requirement is at most 9.10MB, a small cost compared to the RAM requirement for the workers savings policy function (582.4 MB).

Acknowledgements I am very grateful to Eric French, Giulio Fella, Jeppe Druedahl, Uta Bolt and Rory McGee for helpful discussions and comments. Funding from the Alexander S. Onassis Foundation and ESRC Grant Inequality and the insurance value of transfers across the life cycle (ES/P001831/1) for this work is gratefully acknowledged. Any errors are the authors own

Funding Funding from the Alexander S. Onassis Foundation and ESRC Grant Inequality and the insurance value of transfers across the life cycle (ES/P001831/1) for this work is gratefully acknowledged.

Declarations

Conflict of interest There is no conflict of interests.

Code availability Code is available online. The usual disclaimer applies.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Blundell, R., Costa Dias, M., Meghir, C., & Shaw, J. (2016). Female labor supply, human capital, and welfare reform. *Econometrica*, 84(5), 1705–1753.
- Carroll, C. (2006). The method of endogenous gridpoints for solving dynamic stochastic optimization problems. *Economics Letters*, 91(3), 312–320.
- Druehdahl, J. (2020). A guide on solving non-convex consumption-saving models. *Computational Economics*, 10), 1–29.
- Druehdahl, J., & Jørgensen, T. H. (2017). A general endogenous grid method for multi-dimensional models with non-convexities and constraints. *Journal of Economic Dynamics and Control*, 74, 87–107.
- Fella, G. (2014). A generalized endogenous grid method for non-smooth and non-concave problems. *Review of Economic Dynamics*, 17(2), 329–344.
- French, E., & Jones, J. B. (2011). The effects of health insurance and self-insurance on retirement behavior. *Econometrica*, 79(3), 693–732.
- Iskhakov, F., Jørgensen, T. H., Rust, J., & Schjerning, B. (2017). The endogenous grid method for discrete-continuous dynamic choice models with (or without) taste shocks. *Quantitative Economics*, 8(2), 317–365.
- 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*. of MIT Press Books (Vol. 1). London: The MIT Press.
- Judd, K. L., Maliar, L., Maliar, S., & Tsener, I. (2017). How to solve dynamic stochastic models computing expectations just once. *Quantitative Economics*, 8(3), 851–893.
- Santos, M. S. (1999). *Chapter 5 numerical solution of dynamic economic models*. Volume 1 of handbook of macroeconomics (pp. 311–386). London: Elsevier.

Santos, M. S. (2000). Accuracy of numerical solutions using the Euler equation residuals. *Econometrica*, 68(6), 1377–1402.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.