

A Divide and Conquer Algorithm for Exploiting Policy Function Monotonicity

Grey Gordon and Shi Qiu*

October 10, 2017

Abstract

A divide and conquer algorithm for exploiting policy function monotonicity is proposed and analyzed. To solve a discrete problem with n states and n choices, the algorithm requires at most $n \log_2(n) + 5n$ objective function evaluations. In contrast, existing methods for non-concave problems require n^2 evaluations in the worst case. For concave problems, the solution technique can be combined with a method exploiting concavity to reduce evaluations to $14n + 2 \log_2(n)$. A version of the algorithm exploiting monotonicity in two state variables allows for even more efficient solutions. The algorithm can also be efficiently employed in a common class of problems that do *not* have monotone policies, including problems with many state and choice variables. In the sovereign default model of [Arellano \(2008\)](#) and in the real business cycle model, the algorithm reduces run times by an order of magnitude for moderate grid sizes and orders of magnitude for larger ones. Sufficient conditions for monotonicity and code are provided.

Keywords: Computation, Monotonicity, Grid Search, Discrete Choice, Sovereign Default

JEL Codes: C61, C63, E32, F34

*Grey Gordon (corresponding author), Indiana University, Department of Economics, greygordon@gmail.com. Shi Qiu, Indiana University, Department of Economics, shiqiu@indiana.edu. The authors thank Kartik Athreya, Bob Becker, Alexandros Fakos, Filomena Garcia, Bulent Guler, Daniel Harenberg, Juan Carlos Hatchondo, Aaron Hedlund, Chaojun Li, Lilia Maliar, Serguei Maliar, Amanda Michaud, Jim Nason, Julia Thomas, Nora Traum, Todd Walker, Xin Wei, and David Wiczler, as well as the editor and three anonymous referees. We also thank participants at the Econometric Society World Congress 2015 and the Midwest Macro Meetings 2015. Code is provided at <https://sites.google.com/site/greygordon/code>. Any mistakes are our own.

1 Introduction

Many optimal control problems in economics are either naturally discrete or can be discretized. However, solving these problems can be costly. For instance, consider a simple growth model where the state k and choice k' are restricted to lie in $\{1, \dots, n\}$. One way to find the optimal policy g is to evaluate lifetime utility at every k' for every k . The n^2 cost of this *brute force* approach grows quickly in n . In this paper, we propose a divide and conquer algorithm that drastically reduces this cost for problems with monotone policy functions. When applied to the growth model, the algorithm first solves for $g(1)$ and $g(n)$. It then solves for $g(n/2)$ by only evaluating utility at k' greater than $g(1)$ and less than $g(n)$ since monotonicity of g gives $g(n/2) \in \{g(1), \dots, g(n)\}$. Similarly, once $g(n/2)$ is known, the algorithm uses it as an upper bound in solving for $g(n/4)$ and a lower bound in solving for $g(3n/4)$. Continuing this subdivision recursively leads to further improvements, and we show this method—which we refer to as *binary monotonicity*—requires no more than $n \log_2(n) + 5n$ objective function evaluations in the worst case.

When the objective function is concave, further improvements can be made. For example, binary monotonicity gives that $g(n/2)$ is in $\{g(1), \dots, g(n)\}$, but the maximum within this search space can be found in different ways. One way is brute force, i.e., checking every value. However, with concavity, one may check $g(1), g(1) + 1, \dots$ sequentially and stop as soon as the objective function decreases. We refer to this approach as *simple concavity*. An alternative, [Heer and Maubner's \(2005\)](#) method, repeatedly discards half of the search space. We prove that this approach, which we will refer to as *binary concavity*, combined with binary monotonicity computes the optimum in at most $14n + 2 \log_2(n)$ function evaluations.

For problems with non-concave objectives like the [Arellano \(2008\)](#) sovereign default model, binary monotonicity vastly outperforms brute force and simple monotonicity both theoretically and quantitatively. Theoretically, simple monotonicity requires n^2 evaluations in the worst case (the same as brute force). Consequently, assuming worst-case behavior for all the methods, simple monotonicity is 8.7, 35.9, and 66.9 times slower than binary monotonicity for n equal to 100, 500, and 1000, respectively. While this worst-case behavior could be misleading, in practice we find it is not. For instance, in the [Arellano \(2008\)](#) model, we find binary monotonicity is 5.1, 21.4, and 40.1 times faster than simple monotonicity for grid sizes of 100, 500, and 1000, respectively. Similar results hold in a real business cycle (RBC) model when not exploiting concavity. Binary monotonicity vastly outperforms simple monotonicity because the latter is only about twice as fast as brute force.

For problems with concave objectives like the RBC model, we find, despite its good theoretical properties, that binary monotonicity with binary concavity is only the second fastest combination of the nine possible pairings of monotonicity and concavity techniques. Specifically, simple monotonicity with simple concavity is around 20% faster, requiring only 3.0 objective function evaluations per state compared to 3.7 for binary monotonicity with binary concavity. While somewhat slower for the RBC model, binary monotonicity with binary concavity has guaranteed $O(n)$ performance that may prove useful in different setups.

So far we have described binary monotonicity as it applies in the one-state-variable case, but it can also be used to exploit monotonicity in two state variables. Quantitatively, we show a *two-state binary monotonicity* algorithm further reduces evaluation counts per state (to 2.9 with brute force and 2.2 with binary concavity in the RBC example above), which makes it several times faster than the one-state algorithm. Theoretically, we show for a class of optimal policies that the two-state algorithm—without any assumption of concavity—requires at most four evaluations per state asymptotically. As knowing the true policy and simply recovering the value function using it would require one evaluation per state, this algorithm delivers very good performance.

We also show binary monotonicity can be used in a class of problems that does *not* have monotone policies. Specifically, it can be used for problems of the form $\max_{i'} u(z(i) - w(i')) + W(i')$ where i and i' are indices in possibly multidimensional grids and u is concave, increasing, and differentiable. If z and W are increasing, one can show—using sufficient conditions we provide—that the optimal policy is monotone. However, even if they are not increasing, sorting z and W transforms the problem into one that *does* have a monotone policy and so allows binary monotonicity to be used. We establish this type of algorithm is $O(n \log n)$ inclusive of sorting costs and show it is several times faster than existing grid search methods in solving a sovereign default model with capital that features portfolio choice.

For problems exhibiting global concavity, many attractive solution methods exist. These include fast and globally accurate methods such as projection (including Smolyak sparse grid and cluster grid methods), [Carroll’s \(2006\)](#) endogenous gridpoints method (EGM), and [Maliar and Maliar’s \(2013\)](#) envelope condition method (ECM), as well as fast and locally accurate methods such as linearization and higher-order perturbations. [Judd \(1998\)](#) and [Schmedders and Judd \(2014\)](#) provide useful descriptions of these methods that are, in general, superior to grid search in terms of speed and usually accuracy, although not necessarily robustness.¹

In contrast, few computational methods are available for problems with non-concavities, such as problems commonly arising in discrete choice models. The possibility of multiple local maxima makes working with any method requiring first order conditions (FOCs) perilous, which discourages the use of the methods mentioned above. As a result, researchers often resort to value function iteration with grid search. As binary monotonicity is many times faster than simple monotonicity when not exploiting concavity, its use in discrete choice models and other models with non-concavities seems particularly promising.

However, recent research has looked at ways of solving non-concave problems using FOCs. [Fella \(2014\)](#) lays out a generalized EGM algorithm (GEGM) for non-concave problems that finds all points satisfying the FOCs and uses a value function iteration step to distinguish global maxima from local maxima. The algorithm in [Iskhakov, Jørgensen, Rust, and Schjerning \(2016\)](#) is qualitatively similar, but they identify the global maxima in a different and way and show that adding i.i.d.

¹For a comparison in the RBC context, see [Aruoba, Fernández-Villaverde, and Rubio-Ramírez \(2006\)](#). [Maliar and Maliar \(2014\)](#) and a 2011 special issue of the *Journal of Economic Dynamics and Control* (see [Den Haan, Judd, and Juillard, 2011](#)) evaluate many of these methods and their variations in the context of a large-scale multi-country RBC model.

taste shocks facilitates computation. [Maliar and Maliar’s \(2013\)](#) ECM does not necessitate the use of FOCs, and [Arellano, Maliar, Maliar, and Tsyrennikov \(2016\)](#) use it to solve the [Arellano \(2008\)](#) model. However, because of convergence issues, they use ECM only as a refinement to a solution computed by grid search ([Arellano et al., 2016](#), p. 454). While these derivative-based methods will generally be more accurate than a purely grid-search-based method, binary monotonicity solves the problem without adding shocks, requires no derivative computation, and is simple. We also show in the working paper [Gordon and Qiu \(2017\)](#) that binary monotonicity can be used with continuous choice spaces to significantly improve accuracy. Additionally, some of these methods either require (such as GEGM) or benefit from (such as ECM) a grid search component, which allows binary monotonicity to be useful even as part of a more complicated algorithm.

[Puterman \(1994\)](#) and [Judd \(1998\)](#) discuss a number of existing methods (beyond what we have mentioned thus far) that can be used in discrete optimal control problems. Some of these methods, such as policy iteration, multigrids, and action elimination procedures, are complementary to binary monotonicity. Others, such as the Gauss-Seidel methods or splitting methods, could not naturally be employed while simultaneously using binary monotonicity.² In contrast to binary monotonicity, most of these methods try to produce good value function guesses and so only apply in the infinite-horizon case.

Exploiting monotonicity and concavity is not a new idea, and—as [Judd \(1998\)](#) pointed out—the “order in which we solve the various [maximization] problems is important in exploiting” monotonicity and concavity (p. 414). The quintessential ideas behind exploiting monotonicity and concavity date back to at least [Christiano \(1990\)](#); simple monotonicity and concavity as used here are from [Judd \(1998\)](#); and [Heer and Maußner \(2005\)](#) proposed binary concavity, which is qualitatively similar to an adaptive grid method proposed by [Imrohoroglu, Imrohoroglu, and Joines \(1993\)](#). What is new here is that we exploit monotonicity in a novel and efficient way in both one and two dimensions, provide theoretical cost bounds, and show binary monotonicity’s excellent quantitative performance. Additionally, we provide code and sufficient conditions for policy function monotonicity.

The rest of the paper is organized as follows. [Section 2](#) lays out the algorithm for exploiting monotonicity in one state variable and characterizes its performance theoretically and quantitatively. [Section 3](#) extends the algorithm to exploit monotonicity in two state variables. [Section 4](#) shows how the algorithm can be applied to the class of problems with non-monotone policies. [Section 5](#) gives sufficient conditions for policy function monotonicity. [Section 6](#) concludes. The appendices provide additional algorithm, calibration, computation, and performance details, as well as examples and proofs.

²The operations research literature has also developed algorithms that approximately solve dynamic programming problems. [Papadaki and Powell \(2002, 2003\)](#) and [Jiang and Powell \(2015\)](#) aim to preserve monotonicity of value and policy functions while receiving noisy updates of the value function at simulated states. While they exploit monotonicity in a nontraditional order, they do so in a random one and do not compute an exact solution.

2 Binary monotonicity in one state

This section formalizes the binary monotonicity algorithm for exploiting monotonicity in one state variable, illustrates how it and the existing grid search algorithms work, and analyzes its properties theoretically and quantitatively.

2.1 Binary monotonicity, existing algorithms, and a simple example

Our focus is on solving

$$\Pi(i) = \max_{i' \in \{1, \dots, n'\}} \pi(i, i') \quad (1)$$

for $i \in \{1, \dots, n\}$ with an optimal policy g . We say g is monotone (increasing) if $g(i) \leq g(j)$ whenever $i \leq j$.

For a concrete example, consider the problem of optimally choosing next period capital k' given a current capital stock k where both of these lie in a grid $\mathcal{K} = \{k_1, \dots, k_n\}$ having $k_j < k_{j+1}$ for all j . For a period utility function u , a production function F , a depreciation rate δ , a time discount factor β , and a guess on the value function V_0 , the Bellman update can be written

$$V(k_i) = \max_{i' \in \{1, \dots, n\}} u(-k_{i'} + F(k_i) + (1 - \delta)k_i) + \beta V_0(k_{i'}) \quad (2)$$

with the optimal policy $g(i)$ given in terms of indices. Then (2) fits the form of (1). Of course, not every choice is necessarily feasible; there may be multiple optimal policies; there may be multiple state and choice variables; and the choice space may be continuous. However, binary monotonicity handles or can be adapted to handle all these issues.³

Our algorithm computes the optimal policy g and the optimal value Π using divide and conquer. The algorithm is as follows:

1. **Initialization:** Compute $g(1)$ and $\Pi(1)$ by searching over $\{1, \dots, n'\}$. If $n = 1$, STOP. Compute $g(n)$ and $\Pi(n)$ by searching over $\{g(1), \dots, n'\}$. Let $\underline{i} = 1$ and $\bar{i} = n$. If $n = 2$, STOP.
2. At this step, $(g(\underline{i}), \Pi(\underline{i}))$ and $(g(\bar{i}), \Pi(\bar{i}))$ are known. Find an optimal policy and value for all $i \in \{\underline{i}, \dots, \bar{i}\}$ as follows:
 - (a) If $\bar{i} = \underline{i} + 1$, STOP: For all $i \in \{\underline{i}, \dots, \bar{i}\} = \{\underline{i}, \bar{i}\}$, $g(i)$ and $\Pi(i)$ are known.
 - (b) For the midpoint $m = \lfloor \frac{\underline{i} + \bar{i}}{2} \rfloor$, compute $g(m)$ and $\Pi(m)$ by searching over $\{g(\underline{i}), \dots, g(\bar{i})\}$.
 - (c) Divide and conquer: Go to (2) twice, first computing the optimum for $i \in \{\underline{i}, \dots, m\}$ and then for $i \in \{m, \dots, \bar{i}\}$. In the first case, redefine $\bar{i} := m$; in the second, redefine $\underline{i} := m$.

³Beyond the two-state binary monotonicity algorithm in Section 3, the online appendix shows how multiple state or choice variables can be implicit in the i and i' and exploits that fact. It also shows how additional state and/or choice variables, for which one does not want to or cannot exploit monotonicity, may be handled. Further, the appendix shows that, under mild conditions, binary monotonicity will correctly deliver an optimal policy (1) even if there are multiple optimal policies and (2) even if there are nonfeasible choices. For the latter result, $\pi(i, i')$ must be assigned a sufficiently large negative number when an (i, i') combination is not feasible. Moreover, continuous choice spaces can be handled without issue as discussed in the working paper [Gordon and Qiu \(2017\)](#).

Figure 1 illustrates how binary and simple monotonicity work. The blue dots represent the optimal policy of (2), and the empty circles represent the search spaces implied by the respective algorithms. With simple monotonicity, the search for $i > 1$ is restricted to $\{g(i-1), \dots, n'\}$, which results in a nearly triangular search space. For binary monotonicity, the search is restricted to $\{g(\underline{i}), \dots, g(\bar{i})\}$ where \underline{i} and \bar{i} are far apart for the first iterations but rapidly approach each other. This results in an irregularly shaped search space that is large at the first iterations $i = 1, n$, and $n/2$ but much smaller at later ones. For this example, the average search space for simple monotonicity, i.e., the average size of $\{g(i-1), \dots, n'\}$, grows from 10.6 for $n = n' = 20$ to 51.8 for $n = n' = 100$. This is, roughly, a 50% improvement on brute force. In contrast, the average size of binary monotonicity's search space is 7.0 when $n = n' = 20$ (34% smaller than simple monotonicity's) and 9.5 when $n = n' = 100$ (82% smaller), a large improvement.

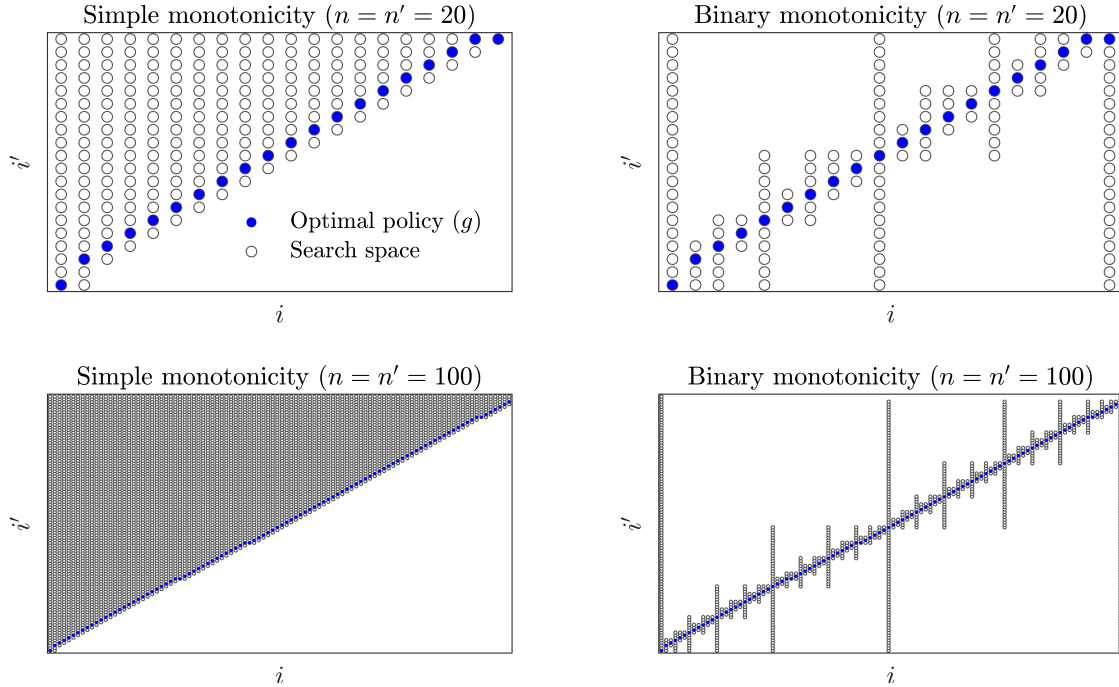


Figure 1: A graphical illustration of simple and binary monotonicity

Binary monotonicity restricts the search space but does not say how one should find a maximum within it. In solving $\max_{i' \in \{a, \dots, b\}} \pi(i, i')$ for a given i , a , and b , one can use brute force, evaluating $\pi(i, i')$ at all $i' \in \{a, \dots, b\}$. In Figure 1, this amounts to evaluating π at every empty circle and blue dot. However, if the problem is concave, simple or binary concavity may be used. Simple concavity proceeds sequentially from $i' = a$ to $i' = b$ but stops whenever $\pi(i, i' - 1) > \pi(i, i')$. In Figure 1, this amounts to evaluating $\pi(i, i')$ from the lowest empty circle in each column to one above the blue dot. In contrast, binary concavity uses the ordering of $\pi(i, m)$ and $\pi(i, m + 1)$ where $m = \lfloor \frac{a+b}{2} \rfloor$ to narrow the search space: If $\pi(i, m) < \pi(i, m + 1)$, an optimal choice must be in $\{m + 1, \dots, b\}$; if $\pi(i, m) \geq \pi(i, m + 1)$, an optimal choice must be in $\{a, \dots, m\}$. The search then proceeds recursively, redefining the search space accordingly until there is only one choice left. In

Figure 1, this amounts to always evaluating $\pi(i, i')$ at adjacent circles (m and $m + 1$) within a column with the adjacent circles progressing geometrically towards the blue dot. For our precise implementation of binary concavity, see Appendix A.

2.2 Theoretical cost bounds

We now characterize binary monotonicity’s theoretical performance by providing bounds on the number of times π must be evaluated to solve for g and Π . Clearly, this depends on the method used to solve

$$\max_{i' \in \{a, \dots, a+\gamma-1\}} \pi(i, i'). \quad (3)$$

While brute force requires γ evaluations of $\pi(i, \cdot)$ to solve (3), binary concavity requires at most $2\lceil \log_2(\gamma) \rceil$ evaluations, which we prove in the online appendix.

Proposition 1 gives the main theoretical result of the paper:

Proposition 1. *Suppose $n \geq 4$ and $n' \geq 3$. If brute force grid search is used, then binary monotonicity requires no more than $(n' - 1)\log_2(n - 1) + 3n' + 2n - 4$ evaluations of π . Consequently, fixing $n = n'$, the algorithm’s worst case behavior is $O(n \log_2 n)$ with a hidden constant of one.*

If binary concavity is used with binary monotonicity, then no more than $6n + 8n' + 2\log_2(n' - 1) - 15$ evaluations are required. Consequently, fixing $n = n'$, the algorithm’s worst case behavior is $O(n)$ with a hidden constant of 14.

Note the bounds stated in the abstract and introduction, $n \log_2(n) + 5n$ for brute force and $14n + 2\log_2(n)$ for binary concavity, are simplified versions of these for the case $n = n'$.

These worst-case bounds show binary monotonicity is very powerful. To see this, note that even if one knew the optimal policy, recovering Π would still require n evaluations of π (specifically, evaluating $\pi(i, g(i))$ for $i = 1, \dots, n$). Hence, relative to knowing the true solution, binary monotonicity is only asymptotically slower by a log factor when $n = n'$. Moreover, when paired with binary concavity, the algorithm is only asymptotically slower by a factor of 14.

2.3 Quantitative performance in the Arellano (2008) and RBC models

We now turn to assessing the algorithm’s performance in the Arellano (2008) and RBC models. First, we use the Arellano (2008) model to compare our method with existing techniques that only assume monotonicity. Second, we use the RBC model to compare our method with existing techniques that assume monotonicity and concavity. We will not conduct any error analysis here since all the techniques deliver identical solutions. The calibrations and additional computation details are given in Appendix B. For a description of how the Arellano (2008) and RBC models can be mapped into (1), see the online appendix.

2.3.1 Exploiting monotonicity in the Arellano (2008) model

Figure 2 compares the run times and average π -evaluation counts necessary to obtain convergence in the Arellano (2008) model when using brute force, simple monotonicity, and binary monotonicity. The ratio of simple monotonicity’s run time to binary monotonicity’s grows virtually linearly, increasing from 5.1 for a grid size of 100 to 95 for a grid size of 2500. The speedup of binary monotonicity relative to brute force also grows linearly but is around twice as large in levels. This latter fact reflects that simple monotonicity is only about twice as fast as brute force irrespective of grid size. For evaluation counts, the patterns are similar, but binary monotonicity’s speedups relative to simple monotonicity and brute force are around 50% larger.

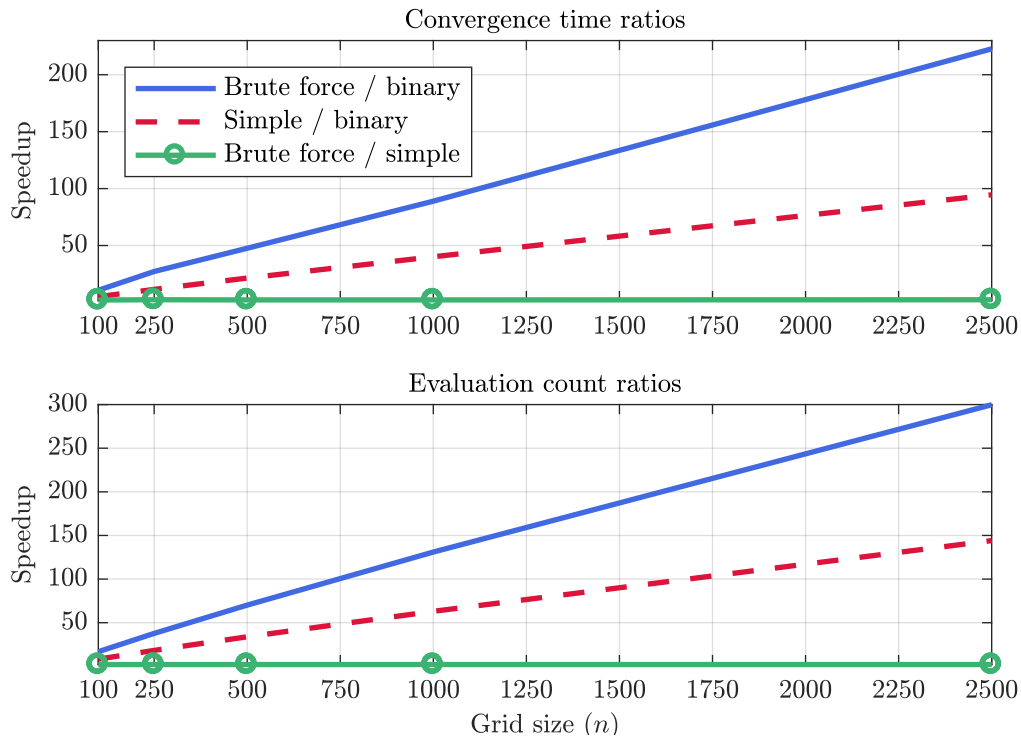


Figure 2: Cost comparison for methods only exploiting monotonicity

Binary monotonicity is faster than simple monotonicity by a factor of 5 to 20 for grids of a few hundred points, which is a large improvement. While these are the grid sizes that have been commonly used in the literature to date (e.g., Arellano, 2008, uses 200), the cost of using several thousand points is relatively small when using binary monotonicity. For instance, in roughly the same time simple monotonicity needs to solve the 250-point case (1.48 seconds), binary monotonicity can solve the 2500-point case (which takes 1.51 seconds). At these larger grid sizes, binary monotonicity can be hundreds of times faster than simple monotonicity.

2.3.2 Exploiting monotonicity and concavity in the RBC model

To assess binary monotonicity’s performance when paired with a concavity technique, we turn to the RBC model. Table 1 examines the run times and evaluation counts for all nine possible combinations of the monotonicity and concavity techniques.

Monotonicity	Concavity	$n = 250$		$n = 500$		Increase
		Eval./ n	Time (s)	Eval./ n	Time (s)	
None	None	250.0	8.69	500.0	29.51	3.4
Simple	None	127.4	3.43	253.4	13.29	3.9
Binary	None	10.7	0.40	11.7	0.85	2.1
None	Simple	125.5	4.39	249.6	17.28	3.9
Simple	Simple	3.0	0.14	3.0	0.26	1.9
Binary	Simple	6.8	0.30	7.3	0.60	2.0
None	Binary	13.9	0.58	15.9	1.27	2.2
Simple	Binary	12.6	0.43	14.6	0.95	2.2
Binary	Binary	3.7	0.20	3.7	0.36	1.8

Note: The last column gives the run time increase from $n = 250$ to 500.

Table 1: Run times and evaluation counts for all monotonicity and concavity techniques

Perhaps surprisingly, the fastest combination is simple monotonicity with simple concavity. This pair has the smallest run times for both values of n and the time increases linearly (in fact, slightly sublinearly). For this combination, solving for the optimal policy requires, on average, only three evaluations of π per state. The reason for this is that the capital policy very nearly satisfies $g(i) = g(i-1)+1$. When this is the case, simple monotonicity evaluates $\pi(i, g(i-1))$, $\pi(i, g(i-1)+1)$, and $\pi(i, g(i-1) + 2)$; finds $\pi(i, g(i-1) + 1) > \pi(i, g(i-1) + 2)$; and stops. The second fastest combination, binary monotonicity with binary concavity, exhibits a similar linear (in fact, slightly sublinear) time increase. However, it fares worse in absolute terms, requiring 3.7 evaluations of π per state. All the other combinations are slower and exhibit greater run time and evaluation count growth.

While Table 1 only reports the performance for two grid sizes, it is representative. This can be seen in Figure 3, which plots the average number of π evaluations per state required for the most efficient methods. Simple monotonicity with simple concavity and binary monotonicity with binary concavity both appear to be $O(n)$ (with the latter guaranteed to be), but the hidden constant is smaller for the former. The other methods all appear to be $O(n \log n)$.

3 Binary monotonicity in two states

In the previous section, we demonstrated binary monotonicity’s performance when exploiting monotonicity in one state variable. However, some models have policy functions that are monotone in

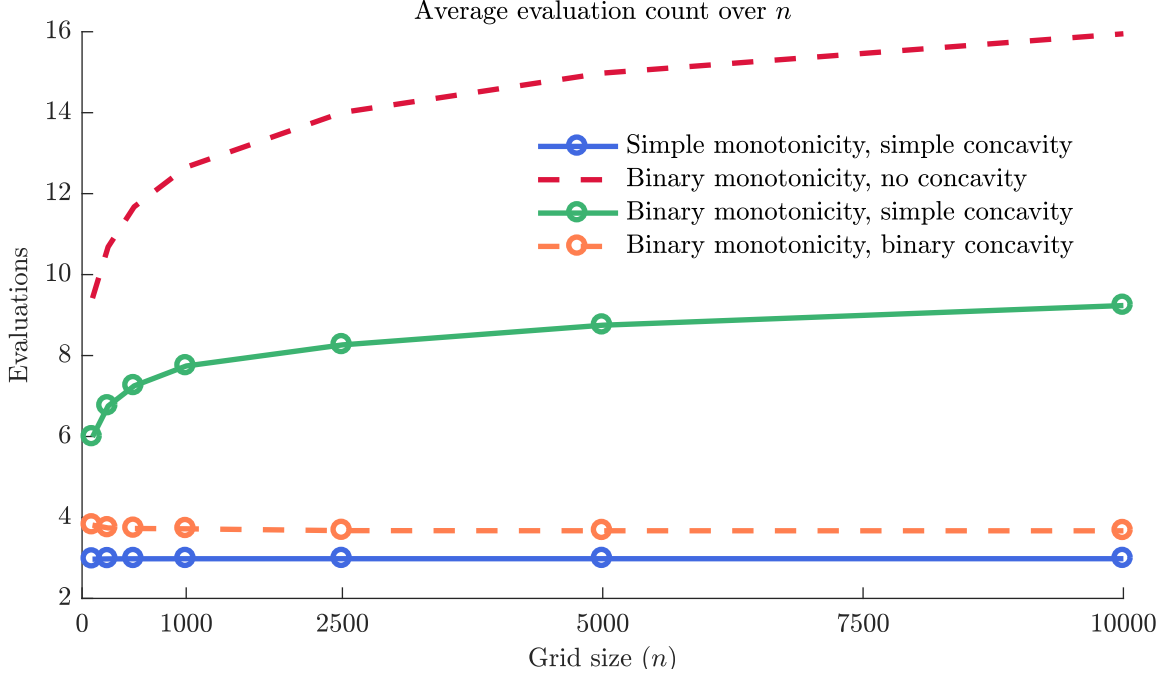


Figure 3: Empirical $O(n)$ behavior

more than one state. For instance, under certain conditions, the RBC model’s capital policy $k'(k, z)$ is monotone in both k and z . In this section, we show how to exploit this property.

3.1 The two-state algorithm and an example

Our canonical problem is to solve

$$\Pi(i, j) = \max_{i' \in \{1, \dots, n'\}} \pi(i, j, i') \quad (4)$$

for $i \in \{1, \dots, n_1\}$ and $j \in \{1, \dots, n_2\}$, where the optimal policy $g(i, j)$ is increasing in both arguments. The two-state binary monotonicity algorithm first solves for $g(\cdot, 1)$ using the one-state algorithm. It then recovers $g(\cdot, n_2)$ using the one-state algorithm but with $g(\cdot, 1)$ serving as an additional lower bound. The core of the two-state algorithm assumes $g(\cdot, \underline{j})$ and $g(\cdot, \bar{j})$ are known and uses them as additional bounds in computing $g(\cdot, j)$ for $j = \lfloor \frac{\underline{j} + \bar{j}}{2} \rfloor$. Specifically, in solving for $g(i, j)$, the search space is restricted to integers in $[g(\underline{i}, j), g(\bar{i}, j)] \cap [g(i, \underline{j}), g(i, \bar{j})]$ instead of just $[g(\underline{i}, j), g(\bar{i}, j)]$ as the one-state algorithm would. Appendix A gives the algorithm in full detail.

Figure 4 illustrates how the two-state binary monotonicity algorithm works when applied to the RBC model using capital as the first dimension. The figure is analogous to Figure 1, but the red lines indicate bounds on the search space coming from the previous solutions $g(\cdot, \underline{j})$ and $g(\cdot, \bar{j})$. At $j = n_2$ (the left panel), $g(\cdot, 1)$ has been solved for and hence provides a lower bound on the search space as indicated by the red, monotonically increasing line. However, there is no upper bound on the search space other than n' . At $j = \lfloor \frac{n_2 + 1}{2} \rfloor$ (the right panel), $g(\cdot, 1)$ and $g(\cdot, n_2)$ are known, and

the bounds they provide drastically narrow the search space.

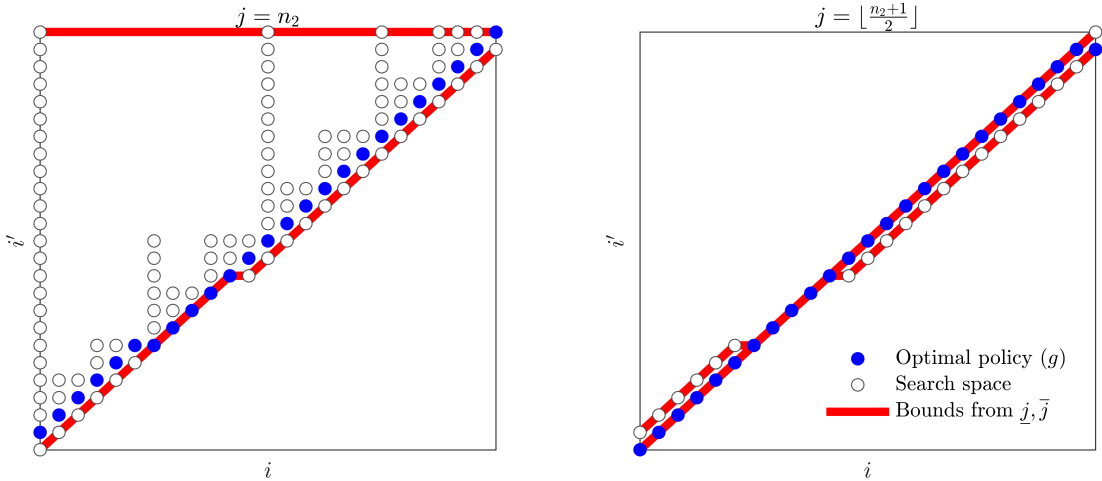


Figure 4: Example of binary monotonicity in two states

3.2 Quantitative performance in the RBC model

Table 2 reports the evaluation counts and run times for the RBC model when using the two-state algorithm with the various concavity techniques. We again treat the first dimension as capital, so $n_1 = n'$ but n_1 does not necessarily equal n_2 . For ease of reference, the table also reports these measures when exploiting monotonicity only in k . The speedup of the two-state algorithm relative to the one-state algorithm ranges from 1.7 to 3.7 when considering evaluation counts and 1.2 to 2.4 when considering run times. Exploiting monotonicity in k and z , even without an assumption on concavity, brings evaluation counts down to 2.9. This is marginally better than the “simple-simple” combination seen in Table 1, which had a 3.0 count for these grid sizes. However, when combining two-state binary monotonicity with binary concavity, evaluations counts drop to 2.2. This significantly improves on the simple-simple 3.0 evaluation count and is only around twice as slow as knowing the true solution.

Monotonicity	Concavity	Eval	Time	Eval Speedup	Time Speedup
k only	None	10.7	0.42	—	—
k only	Simple	6.8	0.29	—	—
k only	Binary	3.7	0.19	—	—
k and z	None	2.9	0.18	3.7	2.4
k and z	Simple	2.4	0.16	2.8	1.8
k and z	Binary	2.2	0.16	1.7	1.2

Note: Time is in seconds; the speedups give the two-state algorithm improvement relative to the one-state; grid sizes of $n_1 = n' = 250$ and $n_2 = 21$ are used.

Table 2: Run times and evaluation counts for one-state and two-state binary monotonicity

How the two-state algorithm’s cost varies in grid sizes can be seen in Figure 5, which plots evaluations counts per state as grid sizes grow while fixing the ratio n_1/n_2 and forcing $n_1 = n'$ (for this figure, concavity is not exploited). The horizontal axis gives the number of states in \log_2 so that a unit increment means the number of states is doubled. In absolute terms, the evaluations per state are below 3 for a wide range of grid sizes. While the overall performance depends on the n_1/n_2 ratio with larger ratios implying less efficiency, in all cases the evaluation counts fall and appear to asymptote as grid sizes increase.

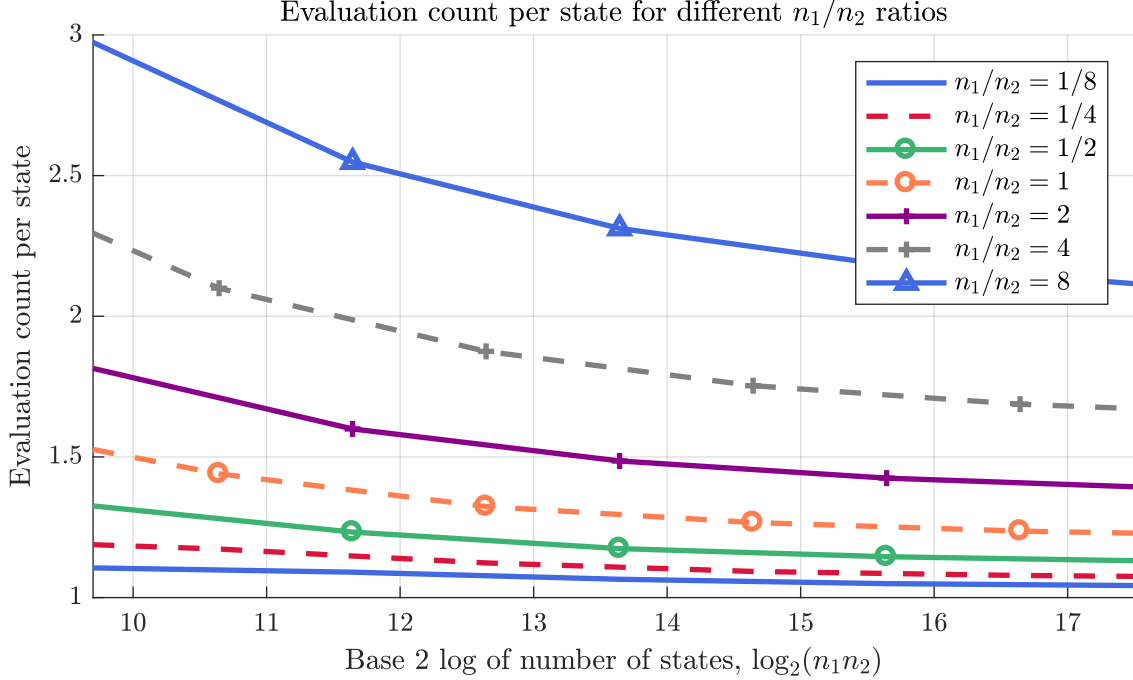


Figure 5: Two-state algorithm’s empirical $O(n_1 n_2)$ behavior for $n_1 = n'$ and n_1/n_2 in a fixed ratio

3.3 Theoretical cost bounds

While Figure 5 suggests the two-state algorithm is in fact $O(n_1 n_2)$ as n_1, n_2 and n' increase (holding their proportions fixed), we have only been able to prove this is the case for a restricted class of optimal policies:⁴

Proposition 2. *Suppose the number of states in the first (second) dimension is n_1 (n_2) and the number of choices is n' with $n_1, n_2, n' \geq 4$. Further, let $\lambda \in (0, 1]$ be such that for every $j \in \{2, \dots, n_2 - 1\}$ one has $g(n_1, j) - g(1, j) + 1 \leq \lambda(g(n_1, j + 1) - g(1, j - 1) + 1)$. Then, the two-state binary monotonicity algorithm requires no more than $(1 + \lambda^{-1}) \log_2(n_1) n' n_2^\kappa + 3(1 + \lambda^{-1}) n' n_2^\kappa + 4n_1 n_2 + 2n' \log_2(n_1) + 6n'$ evaluations of π where $\kappa = \log_2(1 + \lambda)$.*

⁴We abuse notation in writing $O(n_1 n_2)$ to emphasize the algorithm’s performance in terms of the total number of states, $n_1 n_2$. Formally, for $n_1/n_2 = \rho$ and $n_1 = n' =: n$, $O(n_1 n_2)$ should be $O(n^2/\rho)$.

For $n_1 = n' =: n$ and $n_1/n_2 = \rho$ with ρ a constant, the cost is $O(n_1 n_2)$ with a hidden constant of 4 if $(g(n, j) - g(1, j) + 1)/(g(n, j + 1) - g(1, j - 1) + 1)$ is bounded away from 1 for large n .

For any function monotone in i and j , the restriction $g(n_1, j) - g(1, j) + 1 \leq \lambda(g(n_1, j + 1) - g(1, j - 1) + 1)$ is satisfied for $\lambda = 1$. However, for the algorithm's asymptotic cost to be $O(n_1 n_2)$, we require, essentially, $\lambda < 1$. When this is the case, the two-state algorithm—without exploiting concavity—is only four times slower asymptotically than when the optimal policy is known. In the RBC example, we found that $(g(n, j) - g(1, j) + 1)/(g(n, j + 1) - g(1, j - 1) + 1)$ was small initially but grew and seemed to approach one. If it does limit to one, the asymptotic performance is not guaranteed. However, the two-state algorithm has shown itself to be very efficient for quantitatively-relevant grid sizes.

4 Extension to a class of non-monotone problems

In this section, we briefly describe how the model can be applied to a class of problems with potentially non-monotone policies. Our canonical problem is to solve, for $i = 1, \dots, n$,

$$\begin{aligned} V(i) &= \max_{c \geq 0, i' \in \{1, \dots, n'\}} u(c) + W(i') \\ \text{s.t. } c &= z(i) - w(i'), \end{aligned} \tag{5}$$

where $u' > 0, u'' < 0$ with an associated optimal policy g . Here, as before, i and i' can be thought of as indices in possibly multidimensional grids. While this is a far narrower class of problems than (1), it is broad enough to apply in the [Arellano \(2008\)](#) and RBC models, as well as many others.

If z and W are weakly increasing, then one can show (using the sufficient conditions given in the next section) that g is monotone. Our main insight is that binary monotonicity can be applied even when z and W are not monotone by creating a new problem where their values have been sorted. Specifically, letting \tilde{z} and \tilde{W} be the sorted values of z and W , respectively, and letting \tilde{w} be rearranged in the same order as \tilde{W} , the transformed problem is to solve, for each $j = 1, \dots, n$,

$$\begin{aligned} \tilde{V}(j) &= \max_{c \geq 0, j' \in \{1, \dots, n'\}} u(c) + \tilde{W}(j') \\ \text{s.t. } c &= \tilde{z}(j) - \tilde{w}(j') \end{aligned} \tag{6}$$

with an associated policy function \tilde{g} . Because \tilde{W} and \tilde{z} are increasing, binary monotonicity can be used to obtain \tilde{g} and \tilde{V} . Then, one may recover g and V by undoing the sorting. Evidently, this class of problems allows for a cash-at-hand reformulation with z as a state variable, so the real novelty of this approach lies in the sorting of W .

Theoretically, this algorithm is asymptotically efficient when an efficient sorting algorithm is used. Specifically, the sorting for z can be done in $O(n \log n)$ operations and the sorting for W and w in $O(n' \log n')$ operations. Since binary monotonicity is $O(n' \log n) + O(n)$ as either n or n' grows, the entire algorithm is $O(n \log n) + O(n' \log n')$ as either n or n' grows. Fixing $n' = n$, this

is the same $O(n \log n)$ cost seen in Proposition 1. Additionally, the online appendix shows these good asymptotic properties carry over to commonly-used grid sizes in a sovereign default model with bonds and capital that entails a nontrivial portfolio choice problem.

5 Sufficient conditions for monotonicity

In this section, we provide two sufficient conditions for policy function monotonicity. The first, due to Topkis (1978), is from the vast literature on monotonicity.⁵ The second is novel (although an implication of the necessary and sufficient conditions in Milgrom and Shannon, 1994) and applies to show monotonicity in the Arellano (2008) model and sorted problem (6), whereas the Topkis (1978) result does not.

The Topkis (1978) sufficient condition has two main requirements. First, the objective function must have *increasing differences*. In our simplified context, this may be defined as follows:

Definition 1. *Let $S \subset \mathbb{R}^2$. Then $f : S \rightarrow \mathbb{R}$ has weakly (strictly) increasing differences on S if $f(x_2, y) - f(x_1, y)$ is weakly (strictly) increasing in y for $x_2 > x_1$ whenever $(x_1, y), (x_2, y) \in S$.*

For smooth functions, increasing differences essentially requires that the cross-derivative f_{12} be non-negative. However, smoothness is not necessary, and we include many sufficient conditions for increasing differences in the online appendix.

The second requirement is that the feasible choice correspondence must be *ascending*. For our purposes, this may be defined as follows:

Definition 2. *Let $\mathcal{I}, \mathcal{I}' \subset \mathbb{R}$ with $G : \mathcal{I} \rightarrow P(\mathcal{I}')$ where P denotes the power set. Let $a, b \in \mathcal{I}$ with $a < b$. G is ascending on \mathcal{I} if $g_1 \in G(a)$, $g_2 \in G(b)$ implies $\min\{g_1, g_2\} \in G(a)$ and $\max\{g_1, g_2\} \in G(b)$. G is strongly ascending on \mathcal{I} if $g_1 \in G(a)$, $g_2 \in G(b)$ implies $g_1 \leq g_2$.*

One way for the choice set to be ascending is for every choice to be feasible. An alternative we establish in the online appendix is for feasibility to be determined by inequality constraints such as $h(i, i') \geq 0$ with h increasing in i , decreasing in i' , and having increasing differences.

Now we can state Topkis's (1978) sufficient condition as it applies in our simplified framework:

Proposition 3 (Topkis, 1978). *Let $\mathcal{I}, \mathcal{I}' \subset \mathbb{R}$, $I' : \mathcal{I} \rightarrow P(\mathcal{I}')$, and $\pi : \mathcal{I} \times \mathcal{I}' \rightarrow \mathbb{R}$. If I' is ascending on \mathcal{I} and π has increasing differences on $\mathcal{I} \times \mathcal{I}'$, then G defined by $G(i) := \arg \max_{i' \in I'(i)} \pi(i, i')$ is ascending on $\{i \in \mathcal{I} | G(i) \neq \emptyset\}$. If π has strictly increasing differences, then G is strongly ascending.*

Note that a requirement is that the *feasible* choice correspondence is ascending while the result is that the *optimal* choice correspondence is ascending. If the optimal choice correspondence is strongly ascending, then every optimal policy is necessarily monotone. However, for simple and

⁵This literature includes Athey (2002); Hopenhayn and Prescott (1992); Huggett (2003); Jiang and Powell (2015); Joshi (1997); Majumdar and Zilcha (1987); Milgrom and Shannon (1994); Mirman and Ruble (2008); Mitra and Nyarko (1991); Puterman (1994); Quah (2007); Quah and Strulovici (2009); Smith and McCardle (2002); Stokey and Lucas Jr. (1989); Strulovici and Weber (2010); Topkis (1998); and others. The working paper Gordon and Qiu (2017) gives additional discussion on these papers and how they relate to the sufficient conditions here.

binary monotonicity to correctly deliver optimal policies, the weaker result that the optimal choice correspondence is ascending is enough, which we prove in the online appendix.

While the [Topkis \(1978\)](#) result is general, it cannot be used to show monotonicity in the [Arellano \(2008\)](#) model or the sorted problem because their objective functions typically do not have increasing differences.⁶ [Proposition 4](#) gives an alternative sufficient condition that does apply for these problems:

Proposition 4. *Let $\mathcal{I}, \mathcal{I}' \subset \mathbb{R}$. Define $G(i) := \arg \max_{i' \in \mathcal{I}', c(i, i') \geq 0} u(c(i, i')) + W(i')$ where u is differentiable, increasing, and concave.*

If c is increasing in i and has increasing differences and W is increasing in i' , then G is ascending (on i such that an optimal choice exists). If, in addition, c is strictly increasing in i and W is strictly increasing in i' , or if c has strictly increasing differences, then G is strongly ascending.

While the objective's functional form is restrictive, it still applies in many dynamic programming contexts with $u \circ c$ as flow utility and W as continuation utility. The online appendix shows how [Propositions 3 and 4](#) may be used to establish monotonicity in the RBC and [Arellano \(2008\)](#) models, respectively.

6 Conclusion

Binary monotonicity is a powerful grid search technique. The one-state algorithm is $O(n \log n)$ and an order of magnitude faster than simple monotonicity in the [Arellano \(2008\)](#) and RBC models. Moreover, combining it with binary concavity guarantees $O(n)$ performance. The two-state algorithm is even more efficient than the one-state and, for a class of optimal policies, gives $O(n_1 n_2)$ performance without any assumption of concavity. Binary monotonicity is also widely applicable and can even be used in a class of non-monotone problems. While binary monotonicity should prove useful for concave and non-concave problems alike, its use in the latter—where few solution techniques exist—seems especially promising.

References

- C. Arellano. Default risk and income fluctuations in emerging economies. *American Economic Review*, 98(3):690–712, 2008.
- C. Arellano, L. Maliar, S. Maliar, and V. Tsyrennikov. Envelope condition method with an application to default risk models. *Journal of Economic Dynamics and Control*, 69:436–459, 2016.
- S. B. Aruoba, J. Fernández-Villaverde, and J. F. Rubio-Ramírez. Comparing solution methods for dynamic general equilibrium economies. *Journal of Economic Dynamics and Control*, 30(12):2477–2508, 2006.

⁶For instance, in the sorted problem, the objective function $u(\tilde{z}(j) - \tilde{w}(j')) + \tilde{W}(j')$ does not necessarily have increasing differences because, loosely speaking, the cross-derivative $u'' \tilde{z}_j \tilde{w}_{j'}$ is negative if \tilde{w} is strictly decreasing.

- S. Athey. Monotone comparative statics under uncertainty. *The Quarterly Journal of Economics*, 117(1):187, 2002.
- C. D. Carroll. The method of endogenous gridpoints for solving dynamic stochastic optimization problems. *Economics Letters*, 91(3):312–320, 2006.
- L. J. Christiano. Solving the stochastic growth model by linear-quadratic approximation and by value-function iteration. *Journal of Business & Economic Statistics*, 8(1):23–26, 1990.
- W. J. Den Haan, K. L. Judd, and M. Juillard. Computational suite of models with heterogeneous agents II: Multi-country real business cycle models. *Journal of Economic Dynamics and Control*, 35(2):175 – 177, 2011.
- G. Fella. A generalized endogenous grid method for non-smooth and non-concave problems. *Review of Economic Dynamics*, 17(2):329–344, 2014.
- G. Gordon and S. Qiu. A divide and conquer algorithm for exploiting policy function monotonicity. CAEPR Working Paper 2017-006, Indiana University, 2017. URL <https://ssrn.com/abstract=2995636>.
- B. Heer and A. Maußner. *Dynamic General Equilibrium Modeling: Computational Methods and Applications*. Springer, Berlin, Germany, 2005.
- H. A. Hopenhayn and E. C. Prescott. Stochastic monotonicity and stationary distributions for dynamic economies. *Econometrica*, 60(6):1387–1406, 1992.
- M. Huggett. When are comparative dynamics monotone? *Review of Economic Dynamics*, 6(1):1 – 11, 2003.
- A. Imrohoroglu, S. Imrohoroglu, and D. H. Joines. A numerical algorithm for solving models with incomplete markets. *International Journal of High Performance Computing Applications*, 7(3): 212–230, 1993.
- F. Iskhakov, T. H. Jørgensen, J. Rust, and B. Schjerning. The endogenous grid method for discrete-continuous dynamic choice models with (or without) taste shocks. *Quantitative Economics*, forthcoming, 2016.
- D. R. Jiang and W. B. Powell. An approximate dynamic programming algorithm for monotone value functions. *Operations Research*, 63(6):1489–1511, 2015.
- S. Joshi. Turnpike theorems in nonconvex nonstationary environments. *International Economic Review*, 38(1):225–248, 1997.
- K. L. Judd. *Numerical Methods in Economics*. Massachusetts Institute of Technology, Cambridge, Massachusetts, 1998.

- M. Majumdar and I. Zilcha. Optimal growth in a stochastic environment: Some sensitivity and turnpike results. *Journal of Economic Theory*, 43(1):116 – 133, 1987.
- L. Maliar and S. Maliar. Envelope condition method versus endogenous grid method for solving dynamic programming problems. *Economics Letters*, 120:262–266, 2013.
- L. Maliar and S. Maliar. Numerical methods for large-scale dynamic economic models. In K. Schmedders and K. Judd, editors, *Handbook of Computational Economics*, volume 3, chapter 7. Elsevier Science, 2014.
- P. Milgrom and C. Shannon. Monotone comparative statics. *Econometrica*, 62(1):157–180, 1994.
- L. J. Mirman and R. Ruble. Lattice theory and the consumer’s problem. *Mathematics of Operations Research*, 33(2):301–314, 2008.
- T. Mitra and Y. Nyarko. On the existence of optimal processes in non-stationary environments. *Journal of Economics*, 53(3):245–270, 1991.
- K. P. Papadaki and W. B. Powell. Exploiting structure in adaptive dynamic programming algorithms for a stochastic batch service problem. *European Journal of Operational Research*, 142(1):108–127, 2002.
- K. P. Papadaki and W. B. Powell. A discrete online monotone estimation algorithm. Working Paper LSEOR 03.73, Operational Research Working Papers, 2003.
- M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.
- J. K. Quah. The comparative statics of constrained optimization problems. *Econometrica*, 75(2): 401–431, 2007.
- J. K. Quah and B. Strulovici. Comparative statics, informativeness, and the interval dominance order. *Econometrica*, 77(6):1949–1992, 11 2009.
- K. Schmedders and K. Judd. *Handbook of Computational Economics*, volume 3. Elsevier Science, 2014.
- J. E. Smith and K. F. McCardle. Structural properties of stochastic dynamic programs. *Operations Research*, 50(5):796–809, 2002.
- N. L. Stokey and R. E. Lucas Jr. *Recursive Methods in Economic Dynamics*. Harvard University Press, Cambridge, Massachusetts and London, England, 1989.
- B. H. Strulovici and T. A. Weber. Generalized monotonicity analysis. *Economic Theory*, 43(3): 377–406, 2010.

- O. Tange. GNU Parallel - the command-line power tool. *login: The USENIX Magazine*, 36(1):42–47, 2011.
- G. Tauchen. Finite state Markov-chain approximations to univariate and vector autoregressions. *Economics Letters*, 20(2):177–181, 1986.
- D. M. Topkis. Minimizing a submodular function on a lattice. *Operations Research*, 26(2):305–321, 1978.
- D. M. Topkis. *Supermodularity and complementarity*. Princeton University Press, Princeton, N.J., 1998.

A Additional algorithm details

This appendix gives our implementation of binary concavity and the two-state algorithm. The working paper [Gordon and Qiu \(2017\)](#) contains a non-recursive implementation of the one-state binary monotonicity algorithm.

A.1 Binary concavity

Below is our implementation of [Heer and Maußner \(2005\)](#)’s algorithm for solving $\max_{i' \in \{a, \dots, b\}} \pi(i, i')$. Throughout, n refers to $b - a + 1$.

1. **Initialization:** If $n = 1$, compute the maximum, $\pi(i, a)$, and STOP. Otherwise, set the flags $\mathbf{1}_a = 0$ and $\mathbf{1}_b = 0$. These flags indicate whether the value of $\pi(i, a)$ and $\pi(i, b)$ are known, respectively.
2. If $n > 2$, go to 3. Otherwise, $n = 2$. Compute $\pi(i, a)$ if $\mathbf{1}_a = 0$ and compute $\pi(i, b)$ if $\mathbf{1}_b = 0$. The optimum is the best of a, b .
3. If $n > 3$, go to 4. Otherwise, $n = 3$. If $\max\{\mathbf{1}_a, \mathbf{1}_b\} = 0$, compute $\pi(i, a)$ and set $\mathbf{1}_a = 1$. Define $m = \frac{a+b}{2}$, and compute $\pi(i, m)$.
 - (a) If $\mathbf{1}_a = 1$, check whether $\pi(i, a) > \pi(i, m)$. If so, the maximum is a . Otherwise, the maximum is either m or b ; redefine $a = m$, set $\mathbf{1}_a = 1$, and go to 2.
 - (b) If $\mathbf{1}_b = 1$, check whether $\pi(i, b) > \pi(i, m)$. If so, the maximum is b . Otherwise, the maximum is either a or m ; redefine $b = m$, set $\mathbf{1}_b = 1$, and go to 2.
4. Here, $n \geq 4$. Define $m = \lfloor \frac{a+b}{2} \rfloor$ and compute $\pi(i, m)$ and $\pi(i, m+1)$. If $\pi(i, m) < \pi(i, m+1)$, a maximum is in $\{m+1, \dots, b\}$; redefine $a = m+1$, set $\mathbf{1}_a = 1$, and go to 2. Otherwise, a maximum is in $\{a, \dots, m\}$; redefine $a = m$, set $\mathbf{1}_b = 1$, and go to 2.

A.2 The two-state binary monotonicity algorithm

We now give the two-state binary monotonicity algorithm in full detail. To make it less verbose, we omit references to Π , but it should be solved for at the same time as g .

1. Solve for $g(\cdot, 1)$ using the one-state binary monotonicity algorithm. Define $l(\cdot) := g(\cdot, 1)$, $u(\cdot) := n'$, $\underline{j} := 1$, and $\bar{j} := n_2$.
2. Solve for $g(\cdot, \bar{j})$ as follows:
 - (a) Solve for $g(1, \bar{j})$ on the search space $\{l(1), \dots, u(1)\}$. Define $\underline{g} := g(1, \bar{j})$ and $\underline{i} := 1$.
 - (b) Solve for $g(n_1, \bar{j})$ on the search space $\{\max\{\underline{g}, l(n_1)\}, \dots, u(n_1)\}$. Define $\bar{g} := g(n_1, \bar{j})$ and $\bar{i} := n_1$.
 - (c) Consider the state as $(\underline{i}, \bar{i}, \underline{g}, \bar{g})$. Solve for $g(i, \bar{j})$ for all $i \in \{\underline{i}, \dots, \bar{i}\}$ as follows.
 - i. If $\bar{i} = \underline{i} + 1$, then this is done, so go to Step 3. Otherwise, continue.
 - ii. Let $m_1 = \lfloor \frac{\underline{i} + \bar{i}}{2} \rfloor$ and compute $g(m_1, \bar{j})$ by searching $\{\max\{\underline{g}, l(m_1)\}, \dots, \min\{\bar{g}, u(m_1)\}\}$.
 - iii. Divide and conquer: Go to (c) twice, once redefining $(\bar{i}, \bar{g}) := (m_1, g(m_1, \bar{j}))$ and once redefining $(\underline{i}, \underline{g}) := (m_1, g(m_1, \bar{j}))$.
3. Here, $g(\cdot, \underline{j})$ and $g(\cdot, \bar{j})$ are known. Redefine $l(\cdot) := g(\cdot, \underline{j})$ and $u(\cdot) := g(\cdot, \bar{j})$. Compute $g(\cdot, j)$ for all $j \in \{\underline{j}, \dots, \bar{j}\}$ as follows.
 - (a) If $\bar{j} = \underline{j} + 1$, STOP: $g(\cdot, j)$ is known for all $j \in \{\underline{j}, \dots, \bar{j}\}$.
 - (b) Define $m_2 := \lfloor \frac{\underline{j} + \bar{j}}{2} \rfloor$. Solve for $g(\cdot, m_2)$ by essentially repeating the same steps as in 2 (but everywhere replacing \bar{j} with m_2). Explicitly,
 - i. Solve for $g(1, m_2)$ on the search space $\{l(1), \dots, u(1)\}$. Define $\underline{g} := g(1, m_2)$ and $\underline{i} := 1$.
 - ii. Solve for $g(n_1, m_2)$ on the search space $\{\max\{\underline{g}, l(n_1)\}, \dots, u(n_1)\}$. Define $\bar{g} := g(n_1, m_2)$ and $\bar{i} := n_1$.
 - iii. Consider the state as $(\underline{i}, \bar{i}, \underline{g}, \bar{g})$. Solve for $g(i, m_2)$ for all $i \in \{\underline{i}, \dots, \bar{i}\}$ as follows.
 - A. If $\bar{i} = \underline{i} + 1$, then this is done, so go to Step 3 part (c). Otherwise, continue.
 - B. Let $m_1 = \lfloor \frac{\underline{i} + \bar{i}}{2} \rfloor$ and compute $g(m_1, m_2)$ by searching $\{\max\{\underline{g}, l(m_1)\}, \dots, \min\{\bar{g}, u(m_1)\}\}$.
 - C. Divide and conquer: Go to (iii) twice, once redefining $(\bar{i}, \bar{g}) := (m_1, g(m_1, m_2))$ and once redefining $(\underline{i}, \underline{g}) := (m_1, g(m_1, m_2))$.
 - (c) Go to Step 3 twice, once redefining $\bar{j} := m_2$ and once redefining $\underline{j} := m_2$.

While one could “transpose” the algorithm, i.e., solve for $g(1, \cdot)$ in Step 1 rather than $g(\cdot, 1)$ and so on, this has no effect. The reason is that the search space for a given (i, j) is restricted to $[g(\underline{i}, j), g(\bar{i}, j)] \cap [g(i, \underline{j}), g(i, \bar{j})]$. In the transposed algorithm, the search for the same (i, j) pair would be restricted to $[g(i, \underline{j}), g(i, \bar{j})] \cap [g(\underline{i}, j), g(\bar{i}, j)]$. Evidently, these search spaces are the same as long as $\underline{i}, \bar{i}, \underline{j}, \bar{j}$ are the same in both the original and transposed algorithm. This is the case

because—in both the original and transposed algorithm— i is reached after subdividing $\{1, \dots, n_1\}$ in a order that does not depend on g and similarly for j .

B Calibration and computation details

This appendix gives additional calibration and computation details.

For the growth and RBC model, we use $u(c) = c^{1-\sigma}/(1-\sigma)$ with $\sigma = 2$, a time discount factor $\beta = .99$, a depreciation rate $\delta = .025$, and a production function $zF(k) = zk^{.36}$. The RBC model's TFP process, $\log z = .95 \log z_{-1} + .007\varepsilon$ with $\varepsilon \sim N(0, 1)$, is discretized using [Tauchen \(1986\)](#)'s method with 21 points spaced evenly over ± 3 unconditional standard deviations. The capital grid is linearly spaced over $\pm 20\%$ of the steady state capital stock. The growth model's TFP is constant and equal to 1, and its capital grid is $\{1, \dots, n\}$. A full description of the RBC model may be found in [Aruoba et al. \(2006\)](#).

For the [Arellano \(2008\)](#) model, we adopt the same calibration as in her paper. For n bond grid points, 70% are linearly spaced from $-.35$ to 0 and the rest from 0 to $.15$. We discretize the exogenous output process $\log y = .945 \log y_{-1} + .025\varepsilon$ with $\varepsilon \sim N(0, 1)$ using [Tauchen \(1986\)](#)'s method with 21 points spaced evenly over ± 3 unconditional standard deviations.

All run times are for the Intel Fortran compiler version 17.0.0 with the flags `-mkl=sequential -debug minimal -g -traceback -xHost -O3 -no-wrap-margin -qopenmp-stubs` on an Intel Xeon E5-2650 processor with a clock speed of 2.60GHz. The singly-threaded jobs were run in parallel using GNU Parallel as described in [Tange \(2011\)](#). When the run times were less than two minutes, the model was repeatedly solved until two minutes had passed and then the average time per solution was computed.