A Toolkit for Value Function Iteration

Robert Kirkby*

November 25, 2015

Abstract

This article introduces a Toolkit for Value Function Iteration. The toolkit is implemented in Matlab and makes automatic use of the GPU and of parallel CPUs. Likely uses are Teaching, Testing Algorithms, Replication, and Research. I here provide a description of some of the main components and algorithms. I also describe the design philosophy underlying choices about how to structure the toolkit and which algorithms to use. Rather than provide simple examples, something best done online (links are given), I instead perform a replication of a classic paper from the real business cycle literature as a demonstration of the use of such a toolkit. The Toolkit, Documentation, Examples, Replications, and more can be found at vfitoolkit.com

Keywords: Value Function Iteration, Matlab Toolkit, GPU, Discretization, Replication.

JEL Classification: C88; E00; C68; C63; C61; C15

*Kirkby: Victoria University of Wellington. I acknowledge the useful comments of seminar participants at the the 21st Conference on Economics and Finance (Taipei, 2015). Thanks to Alexander Grohe-Meyer for finding a coding bug which could have been rather embarrasing. Thanks to Iskander Karibzhanov letting me recycle some of his quadrature codes as part of the toolkit. Thanks to Erick Sager and William Peterman for helpful discussions. Thanks to an anonymous referee for helping substantially improve the readability of this paper. Please address correspondence about this article to Robert Kirkby at <robertdkirkby@gmail.com>, robertdkirkby.com.

1 Introduction

This article introduces a 'Value Function Iteration (VFI) Toolkit'. The toolkit makes it easy for users to solve infinite horizon value function problems; loosely, inputting the return function, the toolkit solves for and outputs the value function and optimal policy function. Many further commands for creating time series simulations, agents' stationary distributions, and moments of those distributions are also included. The toolkit is implemented in Matlab[1] and automatically switches between using the GPU and parallel CPUs based on which is typically faster in practice. Allowing the user access to the power of parallelization, especially on the GPU, without the user having to understand the programming concepts of parallelization is one of the main contributions of the toolkit. Those who would like to get straight into trying out the toolkit and learn about it that way are directed to vfitoolkit.com where explanatory material and basic examples can be found, and the toolkit downloaded. This article focuses instead on describing the main uses envisaged for the VFI Toolkit and on explaining some of the design philosophy and major decisions involved in how the toolkit works as well as explanation of one of the main commands, namely that for infinite horizon value function iteration.

The Toolkit algorithms are based on full-discretization of the state-space. This choice reflects that full-discretization is robust to the many difficult cases — non-concave return functions, non-convex choice sets, non-differentiable value functions — that often lie behind the decision to use value function iteration methods.[2] Due to its robustness full-discretization also provides a good benchmark against which other algorithms can be compared (Aruoba, Fernandez-Villaverde, and Rubio-Ramirez, 2006) and has the advantage of being easily implemented for large state-spaces and parallelized, including on the graphics processor. The weaknesses of full-discretization are that it tends to be less accurate and slower than more sophisticated but less robust algorithms; although GPU parallelization substantially ameliorates the speed disadvantage.

Four main uses for the VFI Toolkit are envisaged: Teaching, Testing Algorithms, Replication, and Research. These are now addressed in turn.

For Teaching, the VFI Toolkit allows students after writing code to solve a basic value function problem, to easily look at questions like how do the Value function, optimal policy function, and model statistics (such as the Standard Business Cycle Statistics) respond to changes in the model. The VFI Toolkit allows the user both to evaluate the kinds changes that are typically easy to implement regardless of the algorithm used — such as changing parameter values, or changing the utility function — as well as difficult to implement with some algorithms — such as whether tax revenues are rebated lump-sum or introduced as a government spending term in the utility function.

[1] A pre-alpha version also exists in Julia

[2] In the absence of such complications methods based on the first-order conditions, like perturbation and projection methods, are typically preferable, and in the case of perturbation methods there is already a great toolkit in the form of Dynare.

Testing algorithms is likely to be the main appeal of the VFI Toolkit for more advanced users. Say you have implemented an algorithm using a two-dimensional endogenous grid method, with triangular interpolation to return to grid points, or are using shape-preserving splines to fit the value function. Obviously your code will do better on the speed-accuracy trade-off.[3] The VFI Toolkit may still be useful as a quick double-check that your codes do not contain any substantial coding errors (we all make them!). While the VFI Toolkit will have a longer run-time than your advanced hand-crafted codes it will solve your problem (thanks to its robustness) and is easy to implement. Thus, with just a little of your own time and a bit of run-time the toolkit provides an easy way to double-check your own codes for errors.

Bona and Santos (1997) argue for a conception of numerical simulation of models as laboratory experiments. A natural implication of this is that like laboratory experiments they should be subject to replication. While the importance of replication has been widely accepted in other quantitative areas of Economics such as applied Econometrics (Coffman and Niederle, 2015), laboratory experiments, and field experiments it is not so well recognised in quantitative computational Economics. Replication is a use to which the VFI Toolkit is especially suited as it is comparatively easy to implement almost any kind of Macroeconomic model. Since the VFI Toolkit uses global solution methods and is built around robust algorithms it can handle a very wide range of problems. Another advantage of the toolkit in replication is that the main commands will be known not to contain bugs. We can thus be more confident of their outputs although issues of numerical errors will remain. Use of the VFI Toolkit for replication is illustrated later in this article with the replication of Hansen (1985), an early paper from the real business cycle literature. Hansen (1985) represents a convenient example to illustrate the robustness of the VFI Toolkit (due to the atypical assumptions used for the exogenous productivity shock process) and of the potential importance of nonlinear global solution methods like those of the toolkit.

The potential use of the VFI Toolkit for Research is largely self-evident. It solves the value function iteration problems that are used in practice for research in Economics: from investigating the properties of models by simulation, to quantitative models in Macroeconomics, to the structural estimation methods that have become increasingly popular in recent years.

The rest of this article first discusses the design philosophy underlying how the VFI Toolkit is designed and some examples of what this means in terms of the codes. Then a more detailed description of one of the main functions, that for solving infinite horizon value function problems, and the algorithm implementing it, is given. A replication of Hansen (1985) using the toolkit is then given which also provides opportunity for a brief discussion of how the common use of non-global solution methods has had substantial effects on certain aspects of Macroeconomic modeling.

[3] Assuming of course that the model satisfies the required assumptions on the shape of the value function, etc.

2 Design Philosophy

The main design philosophy guiding decisions about how to make the functions of the VFI Toolkit fit together has been to follow the mathematical theory underlying Macroeconomic models. What does this mean? Let's look at the example of solving value function problems.

Mathematical theory about value functions is built around starting from the return function, the discount factor, and the transition function for exogenous shocks. From these theory proceeds to derive results on the value function and the optimal policy function; see, eg. Stokey, Lucas, and Prescott (1989). Following this theory in the toolkit means that the value function iteration commands take as inputs the return function, the discount factor, and the transition function for exogenous shocks, and giving as outputs the value function and the optimal policy function.

This same design philosophy has influenced the other functions. For example both those for simulating time series and for computing the stationary distribution of agents in a heterogenous agent model take the optimal policy function as an input and output a times series and a probability distribution function respectively.

Robustness is the other main idea underlying the design of the VFI Toolkit. The motivation is simple: problems with 'nice' properties — concave return functions, convex choice sets, and everything being continuously differentiable — can be solved faster and more accurately using methods based on the systems of stochastic difference equations derived from the first-order conditions. This is already done well by Dynare using perturbation methods and value function iteration would be an inappropriate solution method for such problems. For this reason the VFI Toolkit should be robust to the 'ugly' problems for which value function iteration is actually used. Allowing for non-concave return functions, non-convex choice sets, and various non-differentiabilities means that numerical methods based on interpolation or fitted value functions using smooth polynomials would fail to converge to the true solutions. For this reason the toolkit uses pure discretization as the default algorithm. The idea being that users who know that certain interpolation or fitted value function methods would be appropriate for their models might activate these for their specific model using the options built into the commands.

This choice of pure discretization as the default algorithm until very recently would have condemned the toolkit to being to slow to be useful. Enter parallelization, especially on the GPU. Value function iteration using pure discretization is almost ideally suited to take advantage of GPU parallelization and it is exactly this which the toolkit takes full advantage of.

2.1 Toolkit Defaults and Options

Following the design philosophy of the VFI Toolkit robust algorithms are the default choice of all the commands in the VFI Toolkit. Many of the commands do however also accept 'options' as

an input. These options are intended to allow the user to easily change the default behaviour of the commands on many aspects. For example options for value functions include how many times to use the Howards improvement algorithm[4] and what tolerance to require for convergence of the value function. While options for time series simulations include the length of the burn-in and the initial point from which the simulation begins.

Another important aspect of the Toolkit defaults is parallelization. The defaults are set based on the assumption that the user has a GPU and multiple CPUs (say between 4 and 12). The various commands automatically switch back and forth between the GPU and CPUs based on which are typically faster for the command in question. The Toolkit options allow these defaults to be overruled, eg. to force everything to run on parallel CPUs or even just on a single CPU.

It is a high priority of future development of the VFI Toolkit to allow for other algorithms — especially those that might be faster or more accurate, but less robust — to be called using the command options.

3 Infinite Horizon Value Function Iteration

One of the main functions in the toolkit is that implementing infinite-horizon value function iteration. For this reason I now describe in detail first the problem to be solved, then in Section 3.1 the algorithm, and lastly in Section 3.2, how the command is called in the VFI Toolkit, as well as what objects it outputs. The basic problem to be solved is the Case 1[5] infinite-horizon value function problem given by

$$V(a,z) = \max_{d,a'} \{F(d,a',a,z) + \beta E[V(a',z')|a,z]\}$$

subject to

$$z' = \pi(z)$$

where
z ≡ vector of exogenous state variables
a ≡ vector of endogenous state variables
d ≡ vector of decision variables
notice that any constraints on d, a, & a' can easily be incorporated into this framework by building

[4]Howards improvement algorithm is also sometimes called modified policy iteration.

[5]The description of this as Case 1 is chosen as it coincides exactly with the definition of Case 1 for stochastic value function problems used in Chapter 8 & 9 of Stokey, Lucas & Prescott (SLP) - Recursive Dynamic Economics (eg. pg. 260). In their notation this is any problem that can be written as $v(x,z) = \sup_{y\in\Gamma(x,z)}\{F(x,y,z) + \beta \int_Z v(y,z')Q(z,dz')\}$. Relating the notation used in this paper with the notation of SLP with which many readers will be more familiar: d and a' are what SLP call y, a is x, and z is z. The VFI Toolkit differentiates between d and a' as the distinction makes a substantial difference for the implementation of the algorithm allowing both a faster run-time and lower memory usage.

them into the return function. Note that this case is only applicable to models in which it is possible to choose a' directly; when this is not so Case 2 will be required.

3.1 Algorithm for VFI

I now turn to describing the algorithm used by the toolkit. The main inputs required by the value function iteration algorithm are the grids for d, a, and z; the discount rate; the transition matrix for z; and the return function F. It also requires info on how many variables make up each of the vectors d, a and z (and the grids onto which they will be discretized).

The algorithm used for solving Case 1 Value Functions is value function iteration. The algorithm implements pure discretization of the state-space. While not as sophisticated as some other algorithm pure discretization is both widely used and has the strengths of being both highly robust and easy to implement. Discretization also provides a good benchmark against which other algorithms can be compared (Aruoba, Fernandez-Villaverde, and Rubio-Ramirez, 2006) and has the advantage of being easily parallelized, including on the graphics processor (Aldrich, Fernandez-Villaverde, Gallant, and Rubio-Ramirez, 2011; Dziubinski and Grassi, 2014). This ability to act as a benchmark or alternatively to provide 'initial guesses' may be particularly useful given a number of promising recent approaches are based on value function iteration (Garlappi and Skoulakis, 2009; Maliar and Maliar, 2013; Barillas and Fernandez-Villaverde, 2007).

Discretized Value Function Iteration and the Optimal Policy Function:
The algorithm for discretized value function iteration is largely standard. A version incorporating Howards improvement is now given. The toolkit allows for any number of decision variables, d and a', endogenous states, a, and exogenous states, z, and then vectorize them internally so that they can be treated as univariate, hence this algorithm is written for univariate d, a and z. n_a and n_z are the number of grid points of each. F is the return function, and V the value function.

Declare initial value V_0.
Declare iteration count $n = 0$.
while $||V_{n+1} - V_n|| >$ 'tolerance constant' **do**
  Increment n. Let $V_{old} = V_{n-1}$.
  for $z = 1, ..., n_z$ **do**
    for $a = 1, ..., n_a$ **do**
      Calculate $E[V_{old}|z]$ ▷ Using quadrature method such as Tauchen method.
      Calculate $V_n(a, z) = \max_{d=1,...,n_d,\, a'=1,...,n_a} F(d, a', a, z) + \beta E[V_{old}(a', z')|z]$
      Calculate $g(a, z) = \arg\max_{d=1,...,n_d,\, a'=1,...,n_a} F(d, a', a, z) + \beta E[V_{old}(a', z')|z]$
    end for
  end for
  for $i = 1, ..., H$ **do** ▷ Howards Improvement Algorithm (do H updates)
    $V_{old} = V_n$

for $z = 1, ..., n_z$ **do**
for $a = 1, ..., n_a$ **do**
Calculate $V_n(a,z) = F(g(a,z),z) + \beta E[V_{old}(g^{a'}(a,z),z')|z]$
end for
end for
end for
end while
for $z = 1, ..., n_z$ **do**
for $a = 1, ..., n_a$ **do**
Calculate $g_n(a,z) = \arg\max_{d=1,...,n_d,\, a'=1,...,n_a} F(d,a',a,z) + \beta E[V_n(a',z')|z]$
end for
end for

The exact implementation in the VFI Toolkit differs from this in a few minor ways, some dependent on what options are used. These are,

- Howard's improvement algorithm is actually only used after the first few iterations, and turned off as convergence of the value function is neared. The first to ensure that it does not propagate negative infinities, and the later to ensure that it does not affect the convergence properties of the algorithm.[6]
- Where exactly the return function is evaluated depends on the *lowmemory* option. With full discretization the return function is a large matrix, and since it is time independent the best option speed-wise is simply to evaluate it once before starting and then just access it as needed. This is however very demanding on memory, hence a *lowmemory* option is also provided, in which case the matrix is generate for each individual z at each iteration (of while and for loop); this is substantially slower, but allows for solving much larger problems.
- $F(g(a,z),a,z)$ needed for Howard's improvement is already almost calculated when finding the new V_n and $g(a,z)$. It is hence stored at that stage and simply pulled from memory later for use in Howards improvement.
- The $E[V_{old}|z]$ depends on z but not a and so is done inside the z for-loop but outside the a for-loop.

[6]Both of these issues could be avoided if the codes were changed, first by banning the possibility that the return function at a given state value necessarily takes value of negative infinity (regardless of decision variable), and then, after having banned such cases, by choosing a 'small' enough initial guess that monotonicity will drive convergence. A choice has been made that any speed boost from doing so was essentially non-existent, while the need for the user to avoid states in which the utility was negative infinity and be very careful about the initial guess was against the robustness principle underlying design of the toolkit.

3.2 Infinite Horizon Value Function Iteration: Case 1

The relevant command is
[V,Policy] = ValueFnIter_Case1(V0, n_d, n_a, n_z, d_grid, a_grid, z_grid, pi_z,
beta, ReturnFn, [vfoptions], [ReturnFnParams]);
This section describes the problem it solves, all the inputs and outputs, and provides some further info on using this command.

The main inputs the value function iteration command requires are the grids for d, a, and z; the discount rate; the transition matrix for z; and the return function F.

It also requires to info on how many variables make up d, a and z (and the grids onto which they should be discretized). And it accepts an initial guess for the value function $V0$ (if you have a good guess this can make things faster).

$vfoptions$ allows you to set some internal options (including parallelization), if $vfoptions$ is not used all options will revert to their default values.

The forms that each of these inputs and outputs takes are now described in detail. The best way to understand how to use the command may however be to just go straight to the examples; in particular those for the Stochastic NeoClassical Growth model and the Basic Real Business Cycle model can be found following the links at vfitoolkit.com.

3.2.1 Inputs and Outputs

To use the toolkit to solve problems of this sort the following steps must first be made.

- Set the discount rate
  beta = 0.96;
- Define n_a, n_z, and n_d as follows. n_a should be a row vector containing the number of grid points for each of the state variables in a; so if there are two endogenous state variables the first of which can take two values, and the second of which can take ten values then $n_a = [2, 10]$;. n_d & n_z should be defined analogously.
- Create the (discrete state space) grids for each of the d, a & z variables,
  a_grid=linspace(0,2,100)'; d_grid=linspace(0,1,100)'; z_grid=[1;2;3];
  (They should be column vectors. If there are multiple variables they should be stacked column vectors)
- Create the transition matrices for the exogenous z variables[7]

[7]These must be so that the element in row i and column j gives the probability of going from state i this period to state j next period.

pi_z=[0.3,0.2,0.1;0.3,0.2,0.2; 0.4,0.6,0.7];
(Often you will want to use the Tauchen Method to create z_grid and pi_z)

- Define the return function. This is the most complicated part of the setup. See the example codes applying the toolkit to some well known problems later in this section for some illustrations of how to do this. It should be a Matlab function that takes as inputs various values for $(d, aprime, a, z)$ and outputs the corresponding value for the return function.
ReturnFn=@(d,aprime,a,z) ReturnFunction_AMatlabFunction

- Define the initial value function, the following one will always work as a default, but by making smart choices for this inital value function you can cut the run time for the value function iteration.
V0=ones(n_a,n_z);

- If you wish to use parallelization on the GPU you must also create $ReturnFnParams$ as part of defining the return function. See codes for examples.

That covers all of the objects that must be created, the only thing left to do is simply call the value function iteration code and let it do it's thing.
[V,Policy] = ValueFnIter_Case1(Tolerance, V0, n_d, n_a, n_z, d_grid, a_grid, z_grid, pi_z, beta, ReturnFn, [vfoptions], [ReturnFnParams]);

The outputs are

- V: The value function evaluated on the grid (ie. on $a \times z$). It will be a matrix of size $[n_a, n_z]$ and at each point it will be the value of the value function evaluated at the corresponding point (a, z).

- $Policy$: This will be a matrix of size $[length(n_d) + length(n_a), n_a, n_z]$. For each point $(a.z)$ the corresponding entries in $Policy$, namely $Policy(:, a, z)$ will be a vector containing the optimal policy choices for (d, a).[8]

3.3 Some further remarks

- Models where d is unnecessary (only a' need be chosen): set $n\_d = 0$ and $d\_grid = 0$ and don't put it into the return fn, the code will take care of the rest.

- Often one may wish to define the grid for z and it's transition matrix by the Tauschen method or something similar. The toolkit provides codes to implement the Tauchen method.

- Models with no uncertainty: these are easy to do simply by setting $n\_z = 1$ and $pi\_z = 1$.

[8] By default, $vfoptions.polindorval = 1$, they will be the indexes, if you set $vfoptions.polindorval = 2$ they will be the values.

3.4 Convergence of some other Toolkit components

Some of the other major features of the Toolkit involve calculating finite time series and their moments, or stationary distributions and their moments. For both of these we know conditions required for the convergence of finite-time simulations — whether for moments of time series, or as a way to calculate the stationary distribution — and the calculation of stationary distributions by iterating on the distribution. Both Santos and Peralta-Alva (2005) and Kamihigashi and Stachurski (2015) provide relevant results on the convergence of simulated path methods. Kirkby (2015) provides relevant results on the convergence of iterations on the distribution. Importantly the conditions (assumptions) required by these papers for convergence are of a kind that can be proved analytically to hold for large classes of Macroeconomic models.

4 Replication of Hansen (1985)

The VFI Toolkit is now used to replicate the results of Hansen (1985). This turns out to be an interesting illustration of the influence of numerical methods on Macroeconomics. Hansen models the process on the productivity shock as an AR(1) with log-normal innovations; this works because linear-quadratic dynamic programming is used as the solution method, so the distribution of innovations is irrelevant. Many solution methods are simply unable to deal with the nonlinearities and asymmetries involved in modeling log-normal innovations. Since the VFI Toolkit is based on using robust global solution methods it is able to handle log-normal innovations, and as seen in the replication results the assumption of log-normal innovations, when modelled in full, has important implications.

The choice of productivity shock process also turns out to be an interesting illustration of the influence of numerical methods on Macroeconomics. Hansen (1985) models the process on the productivity shocks as an AR(1) with log-normal innovations, chosen as this way the productivity process is never negative. In contrast most models chose to model productivity as the having the log of productivity being AR(1) with normally distributed innovations.[9] The latter approach is more common not because it is considered in any way more realistic but because it can be more easily handled using common numerical methods, such as perturbation, which don't deal well with substantial asymmetries or nonlinearities. This shows how the widespread use of certain numerical methods, especially those only able to solve certain kinds of models, has had a strong influence on Macroeconomic modeling; including on quantitative Macroeconomic models which are commonly thought to be less susceptible to strict functional form restrictions than purely analytical models.

Before proceeding to the replication itself I give a quick explanation of why Hansen (1985)'s choice of using linear-quadratic dynamic programming to solve the model allowed modelling the

[9] Ie. $z_t = \rho z_{t+1} + \epsilon_t$, where $\epsilon \sim logN(0, \sigma_\epsilon)$, versus the more common $log(z_t) = \rho log(z_{t-1}) + \epsilon_t$, where $\epsilon \sim N(0, \sigma_\epsilon)$. Both satisfy that the shock process is strictly non-negative, which can be important for theoretical reasons.

process on the productivity shocks as an AR(1) with log-normal innovations: With linear-quadratic dynamic programming the assumption of log-normal innovations is irrelevant as only the conditional mean of the productivity shocks matters.[10] That only the conditional mean of variables matters is also true of first-order perturbation methods (with second-order perturbations the conditional second-moments (and hence variance) also matter, but not higher moments, etc.). While linear-quadratic dynamic programming can solve the model with log-normal innovations it implicitly involves assuming that the log-normal distribution of those shocks is entirely irrelevant. As discussed below the replication results show this is in fact untrue.

4.1 Model

I directly present the model as the value function problem to be solved. Readers interested in knowing the motivation behind looking at these models are referred to Hansen (1985).[11] There are two models, one with 'divisible labor', the other 'indivisible labor'. The only difference mathematically is in the utility function. The value function problem to be solved, with a general utility function, is given by

$$\begin{aligned} V(k,z) = & \max_{c,i,k',y,h} u(c,h) + \beta E[V(k',z')|z] \\ \text{s.t.} \quad & y = zk^{\alpha}h^{1-\alpha} \\ & c + i = y \\ & k' = i + (1-\delta)k \\ & z' = \rho z + \epsilon' \end{aligned}$$

where $\epsilon \sim log - normal$ with mean $1-\rho$ and standard deviation σ_{ϵ}.[12]. k is capital stock, z is productivity shock, c is consumption, y is output, and h is hours worked. The divisible labor economy has utility function $u(c,h) = log(c) + Alog(1-h)$, while the indivisible labor economy has utility function $u(c,h) = log(c) + B(1-h)$.

Hansen (1985) calibrates the model to quarterly US data for the period 1955:Q3-1984:Q1. This leads to the parameter values $\alpha = 0.36$, $\delta = 0.025$, $\beta = 0.99$, $A = 2$, $h_0 = 0.53$, $\rho = 0.95$, and $\sigma_{\epsilon} = 0.00712$; with $B = -Alog(1-h_0)/h_0$.

I also present results for an 'alternative productivity process' using the more standard approach of modelling the log of productivity as being an AR(1) process with normally distributed innovations. This method is popular as it both maintains the required assumption that productivity is

[10] For more on linear-quadratic dynamic programming, see Díaz-Gímenez (2001). Since the log-normal innovations are independent and identically distributed, the conditional mean of the innovations is just equal to their unconditional mean; ie. because of the choice of numerical method only the mean value of the innovations is relevant. Hence the actual log-normality of the innovations is entirely ignored by the solution method used.

[11] Hansen (1985) uses slightly different notation, α he calls θ, z_t he calls λ_t, and ρ he calls γ.

[12] So $exp(\epsilon)$ is distributed as $N(\mu_{ln}, \sigma_{ln}^2)$, where $\mu_{ln} = log(1-\rho) - \sigma_{ln}^2/2$, $\sigma_{ln} = \sqrt{log(1 + \frac{\sigma_{\epsilon}^2}{(1-\rho)^2}}$

always positive, and is easier to implement by using the Tauchen method to approximate the log of productivity. The parameters for this alternative productivity process are chosen to ensure that it has the same mean and variance as the original productivity process.

4.2 Implementation

The VFI Toolkit makes implementing the model simple enough. After defining the parameters, grids, transition matrix for the exogenous shock process, and the return function it is simply a matter of solving the value function problem by calling $ValueFnIter\_Case1$, simulating the time series using $SimTimeSeriesIndexes\_Case1$ and $TimeSeries\_Case1$ and then just calculating the standard business cycle statistics from these.[13]

The only difficulty arose from the productivity shocks being AR(1) with a log-normal innovation. Standard quadrature methods, such as Tauchen, cannot be used for such a shock process.[14] A large fraction of the lines of code involved in replication thus involve dealing with this. It was implemented by first simulating a lengthy time series for the productivity shocks, then using Matlab's built-in histogram routines to divide this into a grid, and then creating the transition matrix by simply counting transitions and the normalising the resulting matrix.[15]

The codes implementing the model can be found at github.com/vfitoolkit/vfitoolkit-matlab-replication. The grids used are 51 points on the hours worked choice, evenly spaced from zero to one. 501 points on the next periods capital choice, half evenly spaced from zero to steady state capital level, half evenly spaced from zero to twice the steady state capital level. 31 points on the productivity shock, either chosen by the quadrature method described above for the case of log-normal innovations, or using Tauchen method with q=3 for the alternative case of log productivity being AR(1) with normally distributed innovations. These grid sizes seem adequate for convergence, as assessed by comparing the results with a reduction of the grids to 31-351-21 points, respectively (viewed another way the limitation to 100 simulation samples, done following Hansen (1985), provides at least as much noise as the grid size).

4.3 Results

Table 1 provides the results of the replication of Table 1 of Hansen (1985); this covers all the quantitative results of the paper. As can be seen by comparing it with the original results of Hansen

[13] Calculating the standard business cycle statistics just involves taking logs, applying the Hodrick-Prescott filter, and then calculating standard deviations and correlations; all of which are standard functions of Matlab.

[14] One could use the Tauchen method to discretize the (exponential of) the lognormal innovation itself and then take the log. This would involve an increase in the state space; to z and ϵ instead of just z. In combination with the use of pure discretization it also necessitates then creating another grid for the AR(1) productivity shocks themselves, and finding the transition matrix for this shock.

[15] Counting transitions and then normalizing is a standard and well behaved estimator for Markov transition matrices. The choice of using histogram routines to choose the grid was an arbitrary assumption.

Table 1: Replication of Table 1 from Hansen (1985)
Standard deviations in percent (a) and correlations with output (b) for US and model economies.

	Quarterly U.S. Time Series[a] 1955:Q3-1984:Q1		Economy with divisible labor[b]		Economy with indivisible labor[b]	
Series	(a)	(b)	(a)	(b)	(a)	(b)
Output	1.76	1.00	1.54 (0.28)	1.00 (0.00)	2.18 (0.28)	1.00 (0.00)
Consumption	1.29	0.85	1.52 (0.18)	0.18 (0.22)	1.56 (0.19)	0.24 (0.17)
Investment	8.60	0.92	7.12 (1.88)	0.75 (0.09)	8.95 (1.69)	0.85 (0.03)
Capital Stock	0.63	0.04	0.37 (0.15)	-0.02 (0.11)	0.49 (0.13)	-0.02 (0.09)
Hours	1.66	0.76	1.41 (0.38)	0.77 (0.10)	2.34 (0.35)	0.91 (0.03)
Productivity	1.18	0.42	0.97 (0.12)	0.43 (0.20)	0.95 (0.09)	0.04 (0.17)

Table 2: Replication of Table 1 from Hansen (1985) with Alternative Productivity Process[c]
Standard deviations in percent (a) and correlations with output (b) for US and model economies.

	Quarterly U.S. Time Series[a] 1955:Q3-1984:Q1		Economy with divisible labor[b]		Economy with indivisible labor[b]	
Series	(a)	(b)	(a)	(b)	(a)	(b)
Output	1.76	1.00	1.48 (0.29)	1.00 (0.00)	1.98 (0.31)	1.00 (0.00)
Consumption	1.29	0.85	1.49 (0.20)	0.14 (0.24)	1.54 (0.19)	0.20 (0.18)
Investment	8.60	0.92	7.06 (1.98)	0.75 (0.11)	8.48 (1.81)	0.83 (0.05)
Capital Stock	0.63	0.04	0.36 (0.15)	-0.03 (0.12)	0.45 (0.14)	-0.01 (0.08)
Hours	1.66	0.76	1.40 (0.44)	0.76 (0.17)	2.10 (0.37)	0.90 (0.05)
Productivity	1.18	0.42	0.92 (0.11)	0.39 (0.23)	0.90 (0.08)	0.08 (0.17)

Table 3: Original Table 1 from Hansen (1985)
Standard deviations in percent (a) and correlations with output (b) for US and model economies.

	Quarterly U.S. Time Series[a] 1955:Q3-1984:Q1		Economy with divisible labor[b]		Economy with indivisible labor[b]	
Series	(a)	(b)	(a)	(b)	(a)	(b)
Output	1.76	1.00	1.35 (0.16)	1.00 (0.00)	1.76 (0.21)	1.00 (0.00)
Consumption	1.29	0.85	0.42 (0.06)	0.89 (0.03)	0.51 (0.08)	0.87 (0.04)
Investment	8.60	0.92	4.24 (0.51)	0.99 (0.00)	5.71 (0.70)	0.99 (0.00)
Capital Stock	0.63	0.04	0.36 (0.07)	0.06 (0.07)	0.47 (0.10)	0.05 (0.07)
Hours	1.66	0.76	0.70 (0.08)	0.98 (0.01)	1.35 (0.16)	0.98 (0.01)
Productivity	1.18	0.42	0.68 (0.08)	0.98 (0.01)	0.50 (0.07)	0.87 (0.03)

[a] The US time series used are real GNP, total consumption expenditures, and gross private domestic investment (all in 1972 dollars). The capital stock series includes non-residential equipment and structures. The hours series includes total hours for persons at work in non-agricultural industries as derived from the *Current Population Survey*. Productivity is output divided by hours. All series are seasonally adjusted, logged and detrended.

[b] The standard deviations and correlations with output are sample means of statistics computed for each of 100 simulations. Each simulation consists of 115 periods, which is the same number of periods as the US sample. The numbers in parentheses are sample standard deviations of these statistics. Before computing any statistics each simulated time series was logged and detrended using the same procedure used for the US time series.

[c] Hansen (1985) models productivity as an AR(1) with log-normal innovations. The 'Alternative Productivity process' models the log of productivity as an AR(1) with normal innovations.

(1985), reproduced here in Table 3 for the readers convenience, they support the main finding of Hansen (1985), namely that the indivisible labor model was able to produce a much higher variance of hours than the divisible labor model helping it become more in line with the data along this dimension; something which had until then be a major critique of real business cycle models.[16] Also obvious from the comparison however is that the results of Hansen (1985) contained substantial approximation error due to the use of linear-quadratic dynamic programming. Comparison of Table 1 and Table 2 show that the choice of log-normal innovations to the productivity shock process was largely unimportant in terms of the model output with one important exception: using log-normal innovations to the productivity process dramatically reduces the correlation of labour productivity with output. This exception is important for real business cycle models as it is exactly the correlation of labour productivity with output in the US, displayed by 1955-1980s data, which provided the original motivation for the real business cycle literature.

This contrast in the correlations shows the importance of solving certain models using nonlinear global solution methods. The VFI Toolkit helps make such methods, currently used by a minority of Economists, more easily accessible.

5 Conclusion

As of writing the VFI Toolkit only solves infinite horizon value function problems. However code already exists for things like calculating transition paths in Bewley-Huggett-Aiyagari models, and for solving finite-horizon value functions; it just needs adapting to the GPU before being released as part of the toolkit itself. Hopefully these features and more will already exist by the time you read this!

I hope that you have been inspired to try out the VFI Toolkit and that it might prove useful to you!

References

Eric Aldrich, Jesus Fernandez-Villaverde, Ronald Gallant, and Juan Rubio-Ramirez. Tapping the supercomputer under your desk: Solving dynamic equilibrium models with graphics processors. Journal of Economic Dynamics and Control, 35(3):386–393, 2011.

S. Aruoba, Jesus Fernandez-Villaverde, and Juan Rubio-Ramirez. Comparing solution methods for dynamic equilibrium economies. Journal of Economic Dynamics and Control, 30(12):2477–2508, 2006.

[16] I will not attempt any summary here of the literature on real business cycle models and their successes and failures.

Fransisco Barillas and Jesus Fernandez-Villaverde. A generalization of the endogenous grid method. Journal of Economic Dynamics and Control, 31:2698–2712, 2007.

Jerry Bona and Manuel Santos. On the role of computation in economic theory. Journal of Economic Theory, 72:241–281, 1997.

Lucas Coffman and Muriel Niederle. Pre-analysis plans have limited upside, especially where replications are feasible. Journal of Economic Perspectives, 29(3):81–98, 2015.

J. Díaz-Gímenez. Linear quadratic approximations: An introduction. In R. Márimon and A. Scott, editors, Computational Methods for the Study of Dynamic Economies, chapter 2. Oxford University Press, 2001.

Matt Dziubinski and Stefano Grassi. Heterogeneous computing in economics: A simplified approach. Computational Economics, 43(4):485–495, 2014.

Lorenzo Garlappi and Georgios Skoulakis. Numerical solutions to dynamic portfolio problems: The case for value function iteration using taylor approximation. Computational Economics, 33: 193–207, 2009.

Gary Hansen. Indivisible labor and the business cycle. Journal of Monetary Economics, 16(3): 309–327, 1985.

Takashi Kamihigashi and John Stachurski. Seeking ergodicity in dynamic economies. Working paper, 2015.

Robert Kirkby. Convergence of discretized value function iteration. (Submitted), 2015.

Lilia Maliar and Serguei Maliar. Envelope condition method versus endogenous grid method for solving dynamic programming problems. Economics Letters, 120:262–266, 2013.

Manuel Santos and Adrian Peralta-Alva. Accuracy of simulations for stochastic dynamic models. Econometrica, 66:409–426, 2005.

Nancy Stokey, Robert E. Lucas, and Edward C. Prescott. Recursive Methods in Economic Dynamics. Harvard University Press, 1989.