Economics Bulletin

Volume 36, Issue 4

Getting the most from MATLAB: ditching canned routines and embracing coder

John Gibson
Department of Economics, Georgia State University

James P Henson
Department of Economics, Georgia State University

Abstract

This paper demonstrates the efficiency gains that can be realized by replacing canned routine calls within MATLAB with user-generated versions that simplify the underlying computations. Once canned routines have been replaced, we integrate C++ executables (or MEX files) using MATLAB's Coder to automatically convert our MATLAB code. We demonstrate these efficiency gains by computing the stationary equilibria and associated transition path for an economy with incomplete insurance markets following a change in government debt policy. The combined process of replacing calls to canned routines and integrating MEX files reduces our runtime from just over 24 hours to approximately 16 minutes.

Citation: John Gibson and James P Henson, (2016) "Getting the most from MATLAB: ditching canned routines and embracing coder", *Economics Bulletin*, Volume 36, Issue 4, pages 2519-2525
Contact: John Gibson - jgibson25@gsu.edu, James P Henson - jhenson6@student.gsu.edu.
Submitted: October 21, 2016. **Published:** December 21, 2016.

1 Introduction

A fundamental difficulty with computational methods is balancing runtime and programmer time. While lower level languages such as C++ offer unmatched runtime efficiency, researchers with limited programming experience may find such environments challenging. In these instances, higher level languages such as MATLAB are often chosen. Unfortunately, many practitioners who choose MATLAB for its ease-of-use will rely on canned routines, such as fsolve, for simple processes within their programs. While canned routines provide numerical checks, many economic problems do not require such checks and can be handled with user-generated simplifications of such routines. In this paper, we demonstrate that replacing canned routines with user-generated versions can greatly reduce runtime. We compute the stationary equilibria and associated transition path for an economy with incomplete markets following a change in government debt policy and find that our runtime falls from just over 24 hours to an hour and a half once canned routines are replaced.

While replacing canned routines with simplified code reduces our runtime significantly, we can further reduce runtime by integrating C++ executables (or MEX files) into our main program. The existing literature has documented the performance benefits of incorporating user-generated MEX files[1], but this process requires the user to have an intermediate knowledge of C++. To gain insight into the performance gains that can be achieved without detailed knowledge of C++, we focus instead on the use of MATLAB's built-in Coder to automatically convert portions of our MATLAB code into MEX files. We find that this method reduces our runtime from an hour and a half to just over 16 minutes. We also test the performance of user-generated MEX files and find that while this method outperforms Coder, the runtime difference is insignificant.

Our paper fits into a broad literature on methods that improve computational speed.[2] While many papers in this literature assess the efficiency gains of parallelization, we focus instead on optimizing MATLAB's runtime efficiency by replacing canned routines with user-generated simplifications and integrating MEX files into our main code. The paper most closely related to ours is Aruoba and Fernndez-Villaverde (2015) (AFV) which presents a comparison of several programming languages. Our paper differs from AFV in the following three ways. First, while AFV use the Neoclassical Growth Model to test runtimes, we use an incomplete markets model. Therefore, our results provide an assessment on the performance differences within a heterogeneous agents environment. Second, given the simplicity of the problem considered by AFV, no canned routines were used. However, solving our problem requires repeated calls to such routines, and we find that relying on these canned routines increases runtime significantly. Third, while AFV finds that incorporating MEX files can improve runtime efficiency, they only work with user-generated files. By considering MATLAB's Coder we are able to determine the efficiency gains that can be achieved

[1] See Aruoba and Fernndez-Villaverde (2015) for an application involving the Neoclassical Growth model.

[2] See Maliar (2015) for a discussion of the cost/benefits of parallelizing economic problems in a high performance computing (HPC) environment. Other papers, such as Aldrich et al. (2011), Morozov and Mathur (2012) , and Dziubinski and Grassi (2014), have focused on the performance gains through parallel and GPU computing on local workstations.

without any knowledge of C++. In summary, our paper contributes to the literature by demonstrating the efficiency gains that can be had by replacing calls to canned routines with simplified user-generated code and by using MATLAB's Coder to incorporate MEX files within a heterogenous agents environment.

2 Model

Our model environment follows Aiyagari and McGrattan (1998) where agents face idiosyncratic labor productivity shocks and incomplete insurance markets and the government can issue bonds to households. The next section provides a basic overview of the model, but interested readers are directed to Aiyagari and McGrattan (1998) for further details.

2.1 Basic Structure

The economy is populated by a continuum of ex-ante identical agents who experience idiosyncratic shocks, ϵ, to their labor productivity each period. Insurance markets are incomplete, but agents can partially insure against fluctuations in labor productivity by accumulating assets, a, which consist of private capital and public debt. Overtime, differences in realized labor productivity will lead to different levels of asset accumulation between agents. Thus, while all agents are ex-ante identical, they become heterogenous ex-post.

An agent's state is given by their current assets, a, and their current realization of labor productivity, ϵ. Given these values, an agent chooses how much to consume, c, how much time to allocate to labor, n, and how much to save for the next period, a', in order to solve:

$$V(a,\epsilon) = \max_{c,n,a'} \left[\frac{1}{1-\sigma}(c^{\nu}(1-n)^{1-\nu})^{1-\sigma} + \beta \sum_{\epsilon'} \pi(\epsilon'|\epsilon) V(a',\epsilon') \right] \tag{1}$$

s.t.

$$c + a' \leq (1-\tau)wn\epsilon + (1+(1-\tau)r)a + T \tag{2}$$

$$a \geq 0 \tag{3}$$

where (2) and (3) are the agent's budget and borrowing constraints respectively.

By solving the agents' problems, we can compute the following economy-wide aggregates:

$$A = \sum_{\epsilon} \int_{a} g^{a}(a,\epsilon) f(a,\epsilon) da$$

$$N = \sum_{\epsilon} \epsilon \int_{a} g^{n}(a,\epsilon) f(a,\epsilon) da$$

$$C = \sum_{\epsilon} \int_{a} g^{c}(a,\epsilon) f(a,\epsilon) da$$

$$K = A - B$$

where A, N, C, K, and B denote aggregate assets, labor, consumption, capital, and public

debt respectively. Also, $g^x(a,\epsilon)$ denotes the optimal decision rule for variable x for an agent with assets equal to a and current labor productivity equal to ϵ. Finally, $f(a,\epsilon)$ denotes the time-invariant distribution of individual states.

At the aggregate level, the representative firm rents capital, K, and hires labor, N, from all agents in the economy in order to maximize profits each period. Therefore, the firm's problem is standard and is given by:

$$\max_{K,N} K^{\alpha} N^{1-\alpha} - wN - (r+\delta)K \tag{4}$$

where δ denotes the depreciation rate of capital.

The government raises revenue by taxing both labor and interest income, $\tau(wN + rA)$, and by issuing new debt, $B' - B$. They use these resources to fund government consumption, G, lump-sum transfers, T, and to make service payments on their outstanding debt, rB. In the stationary equilibrium, the government's budget constraint is given by:

$$G + T + rB = \tau(wN + rA) \tag{5}$$

2.2 Calibration, and Computational Procedure

Our model is calibrated following Aiyagari and McGrattan (1998). Specifically, we model the idiosyncratic shock as a 7 state Markov process with persistence ρ_ϵ and volatility σ_ϵ, and we discretize it using Tauchen (1986). We set the ratios of government consumption and transfers to output to 21.7 percent and 8.2 percent respectively. In our baseline economy we set the ratio of public debt to output equal to 67 percent, which is the optimal value identified by Aiyagari and McGrattan (1998).[3] The remaining parameter values are also set following Aiyagari and McGrattan (1998) (See Table I for further details).

To compute our full solution we must solve for the stationary equilibrium when the ratio of public debt to output is 67 percent and again when it is 80 percent. Once both stationary solutions are solved, we compute the transition path the economy takes as it converges to the new long run equilibrium. We employ standard methods for solving for the stationary equilibria and associated transition paths found in the literature (see Domeij and Heathcote (2004), Heer and Maussner (2009), and Chatterjee et al. (2016) for further details). Also, a detailed technical appendix is available from the authors upon request.

3 Results

We solve our model in four ways, (i) using MATLAB with canned routines, (ii) replacing canned routines with user-generated code, (iii) integrating Coder-generated MEX files, and (iv) integrating user-generated MEX files.[4]

[3] We also consider an economy where public debt to output increases to 80 percent. The computation of both stationary equilibria and the associated transition path represents a full solution to our problem.

[4] All code is available from authors upon request.

3.1 Initial MATLAB Code

When computing stationary equilibria, we compute the following separately: (i) labor supply decision, (ii) time-invariant decision rules, and (iii) invariant distribution. Similarly, for transition dynamics, we solve the following separately: (i) time-dependant decision rules, and (ii) asset distribution at each time period. Each of these sections of code were placed in their own separate function file and called from our main MATLAB program. Within each of these functions, several of MATLAB's canned routines, such as interp1, fsolve, fminbnd were used. These routines are commonly used throughout economics, so including them in our baseline code is natural. Our results show that solving our problem using MATLAB with canned routines results in a runtime of just over 24 hours (See Table II).[5]

3.2 Replacing MATLAB's Canned Routines

While it is natural to make use of canned routines when initially coding a problem, we suspect that they increase runtime significantly. To test this, we replace all calls to interp1 with our own 3rd-order lagrange interpolation program. Similarly, we replace calls to fsolve and fminbnd with our functions for the secant method and golden section search respectively.[6] While our user-generated code is not identical to the canned routines they are replacing, we do not find any significant differences in results. Furthermore, we find that replacing calls to MATLAB's canned routines results in a tremendous efficiency increase with total runtime falling from just over 24 hours to around an hour and a half (See Table II). The intuition behind this performance increase is straightforward. The canned routines we are replacing are very robust and this robustness comes at a slight cost in computational time. Given the iterative nature of our problem, these routines are called very frequently,[7] and as such, even small runtime differences between the canned routines and our simplifications can lead to tremendous differences in final runtime. Therefore, our results indicate that many of MATLAB's canned routines are very inefficient when applied to standard economic problems.

3.3 Incorporating C++ Executables

With canned routines replaced, we consider the additional performance gains that can be found by integrating MEX files with our main code. We utilize MATLAB's built-in Coder to automatically convert portions of our MATLAB code into MEX files. This method requires absolutely no knowledge of C++ by the programmer. To use Coder, simply launch the Coder Application from your MATLAB Console. This will start an interactive session where the files that need to be converted can be uploaded. This entire process takes just a few

[5] Results were found using a Dell Precision 8600 with dual Xeon 2687W processors and 64Gb of RAM.

[6] All of these user-generated routines were written in MATLAB and taken from standard sources (See Heer and Maussner (2009) and Judd (1998)).

[7] In order to get an estimate of the number of calls, we setup counters for the baseline stationary distribution. The results were 849,101 for fminbnd, 69 for fsolve, 11,502,834 for interp1.

minutes and as such, we view Coder as requiring virtually no additional programmer time.[8] We find that using Coder-generated MEX files further increases runtime efficiency, reducing total runtime from about an hour and a half to just over 16 minutes (See Table II). This is a tremendous increase in runtime efficiency that is practically costless in terms of programmer time. To determine the efficiency of Coder, we also solve our problem using user-generated MEX files. Our results indicate that while user-generated MEX files yield the fastest runtime, they are only about 3 minutes faster than our Coder-generated files. Given this result, we conclude that Coder provides performance gains that are nearly identical to generating your own MEX files, without the associated cost of additional programmer time.

4 Conclusion

In this paper, we consider two methods for increasing MATLAB's runtime efficiency: (i) replacing canned routine calls with user-generated simplifications of the underlying process, and (ii) using MATLAB's built-in Coder to convert portions of our MATLAB code into C++ executable (MEX file). We demonstrate our efficiency gains by computing the stationary equilibria and associated transition path for an economy with incomplete markets following a change in government debt policy. Together, our methods reduced total runtime from just over 24 hours to approximately 16 minutes. While we consider a specific example, our results are generalizable. Therefore, researchers who work exclusively in MATLAB can adopt our methods to improve their own runtime performance without the significant increase in programmer time that accompanies learning a new programming language.

[8] You must have a C++ compiler installed to run C++ code. Use "Mex -setup" to check your system.

Table I: Parameter Values

$\alpha = 0.3$	$\beta = 0.97$	$\delta = 0.075$	$\nu = 0.328$
$\sigma = 1.5$	$\frac{G}{Y} = 0.217$	$\frac{T}{Y} = 0.082$	

Table II: Runtime (Minutes)

	Initial SS	Terminal SS	Transition Path	Full Runtime
MATLAB	59.53	61.50	1333.90	1454.93
Unrolled	9.37	9.85	53.78	73.00
Coder	2.44	2.57	11.74	16.75
Hand-Coded	1.14	1.21	11.63	13.98

References

Aiyagari, S. R. and E. R. McGrattan (1998). The Optimum Quantity of Debt. *Journal of Monetary Economics 42*(3), 447–469.

Aldrich, E. M., J. Fernndez-Villaverde, A. Ronald Gallant, and J. F. Rubio-Ramrez (2011). Tapping the Supercomputer Under Your Desk: Solving Dynamic Equilibrium Models with Graphics Processors. *Journal of Economic Dynamics and Control 35*(3), 386–393.

Aruoba, S. B. and J. Fernndez-Villaverde (2015). A Comparison of Programming Languages in Macroeconomics. *Journal of Economic Dynamics and Control 58*(C), 265–273.

Chatterjee, S., J. Gibson, and F. Rioja (2016). Optimal Public Debt Redux. *Working Paper*.

Domeij, D. and J. Heathcote (2004). On The Distributional Effects Of Reducing Capital Taxes. *International Economic Review 45*(2), 523–554.

Dziubinski, M. P. and S. Grassi (2014). Heterogeneous Computing in Economics: A Simplified Approach. *Computational Economics 43*(4), 485–495.

Heer, B. and A. Maussner (2009). *Dynamic General Equilibrium Modeling: Computational Methods and Applications*. Springer Berlin Heidelberg.

Judd, K. (1998). *Numerical Methods in Economics*. Scientific and Engineering. MIT Press.

Maliar, L. (2015). Assessing Gains from Parallel Computation on a Supercomputer. *Economics Bulletin 35*(1), 159–167.

Morozov, S. and S. Mathur (2012). Massively Parallel Computation Using Graphics Processors with Application to Optimal Experimentation in Dynamic Control. *Computational Economics 40*(2), 151–182.

Tauchen, G. (1986). Finite State Markov-chain Approximations to Univariate and Vector Autoregressions. *Economics Letters 20*(2), 177–181.