BANCO DE ESPAÑA
Eurosistema

Dirección General del Servicio de Estudios

**11.02.2008**

# Accelerating Dynare, up to 10x, with SilverBullet

Samuel Hurtado

Departamento de Coyuntura y Previsión Económica

## 1 Introduction

Dynare is an amazing tool for setting up and estimating dynamic stochastic general equilibrium models (DSGEs). However, when the model to be used is relatively big, some of the processes done by Dynare can take a lot of time. This is the case, for example, of BEMOD[1]. The estimation of that model, including a million iterations of the Metropolis-Hastings algorithm, is taking more than two months on the fastest standard PC available today[2]. An even bigger problem is that just a test estimation also takes quite a lot of time: nearly a day for the initial optimization, plus another four days for a tentative block of 50000 iterations of the Metropolis-Hastings algorithm, just enough to give a somewhat useful (though potentially inaccurate) idea of what the posterior distributions of the estimated parameters look like. These computation times make the process of estimating such a model a pretty unpleasant one.

In this note I present a pretty straightforward modification of the codes of Dynare that can reduce estimation times by a factor of up to 10, while providing numerically identical results.

## 2 The problem

If we launch an estimation of BEMOD on Dynare while recording all the details of the process with the Matlab profiler, we get results that look roughly like this:

| m-file | calls | total time | self time |
|---|---|---|---|
| dynare_estim | 1 | 100% | 0% |
| metropolis | 1 | 100% | 0% |
| DsgeLikelihood | 5K | 100% | 0% |
| dynare_resolve | 5K | 89% | 0% |
| resol | 5K | 89% | 0% |
| dr1 | 5K | 89% | 0% |
| jacob_a | 5K | 88% | 0% |
| ff1_ | 3.8M | 88% | 1% |
| MyModel_ff | 3.8M | 87% | 87% |

The estimation is done in the dynare_estim.m process, which calls the metropolis.m process, which calls the DsgeLikelihood.m process, which calls the dynare_resolve.m process, which calls the resol.m process, which calls the dr1.m process.

This dr1.m process has to calculate the jacobian of the function that evaluates the state-space representation of the model. For that, it calls the jacob_a.m process.

[1] See J. Andrés, P. Burriel, A. Estrada (2006): "BEMOD: A DSGE model for the Spanish economy and the rest of the Euro area", Bank of Spain Working Paper no. 0631.

[2] Using server or workstation hardware based on any number of x86 processors will not reduce that time.

The jacob_a.m process has this structure:

```
Evaluate function.m at given point xdh
For n = 1 to the size of xdh (383 in BEMOD)
        xdh1 = a bit to the left of xdh in the n-th dimenion
        xdh2 = a bit to the right of xdh in the n-th dimension
        Coordinate n of jacobian = ((function.m at xdh1)-(function.m at xdh2))/h
End for
```

Its target function is residing in a different m-file, which in our case is ff1_.m, so we see jacob_a.m calling ff1_.m more than 700 times to calculate its jacobian just once.

ff1_.m is just an intermediate process that classifies the coordinates of xdh appropriately and then calls MyModel_ff.m, which is the process that contains the state-space representation of the model.

This MyModel_ff.m file is where most of the estimation time is spent. It contains something like this:

```
Declaration of a lot of global variables, corresponding to the parameters of the model
z(1) = function of the parameters of the model and the point at which it is being evaluated
z(2) = function of the parameters of the model and the point at which it is being evaluated
etc
```

So, in essence, what jacob_a.m is doing is to call once and again for the evaluation of a function. And it is doing it in this fashion: “take this vector, look up some parameters that are stored in memory, and make some calculations”.

If we look at the profiler information for MyModel_ff.m, we’ll see exactly where the time is being spent.

What we’ll see there is that the parts related to “take this vector” and “make some calculations” are relatively fast and not really taking a lot of time, while it is the “look up some parameters that are stored in memory” part which is taking nearly all the processing time necessary to estimate this model.

This is due to the fact that global variables, while being helpful and convenient at the time of writing a program, are a highly inefficient way of passing information around, at least in Matlab.

## 3 The solution (SilverBullet)

In essence, what the profiler information has shown us is that we are doing something that is inefficient (looking up global variables stored in memory), and we are doing it over and over again (in our case, more than 700 times each time we want to calculate a jacobian). Once we realize this, finding a solution is relatively straightforward.

There are at least two ways around this problem. One would be to rewrite the codes so that they no longer use global variables; this would take a very big effort, by someone with a very good understanding of what each file in the Dynare package is doing, when it is doing it, and how it is doing it. The other solution is much easier to implement: simply reduce the number of times that these global variables are looked up from the memory.

We have seen that each time Dynare wants to calculate the jacobian of the MyModel_ff.m function, the parameters of the model are looked up from memory more than 700 times, even though at this point they are not changing at all.

A very simple solution is to modify the jacob_a.m file so that it loads those global variables only once, and calculates the jacobian by itself, without ever calling ff1_.m or MyModel_ff.m.

The structure of this new jacob_a.m file would be something like this:

```
Declaration of a lot of global variables, corresponding to the parameters of the model
Evaluate function.m at given point xdh
For n = 1 to the size of xdh (383 in BEMOD)
        xdh1 = a bit to the left of xdh in the n-th dimenion
        xdh2 = a bit to the right of xdh in the n-th dimension
        if function to evaluate is ff1_
                reorganize variables as done in ff1_
                z_xdh1(1) = function of xdh1 and the parameters of the model
                z_xdh1(2) = function of xdh1 and the parameters of the model
                etc
                z_xdh2(1) = function of xdh1 and the parameters of the model
                z_xdh2(2) = function of xdh2 and the parameters of the model
                etc
                Coordinate n of jacobian = (z_xdh1-z_xdh2)/h
```

```
            Else
                        Coordinate n of jacobian = ((function at xdh1)-(function at xdh2))/h
            End if
End for
```

This jacob_a.m file calculates ff1_.m itself, without calling any other m-files, if that is the function to be evaluated; if it is asked to calculate the jacobian of any other function, it will do it through the standard (slower) way[3].

This new jacob_a.m file will only work for a given MyModel_ff.m function, so it should reside in the folder where our model is. Once it is there, every time Matlab looks for a process with that name while working on that folder, it will find it right there, and so it will not look for it on the default paths, which is where the original jacob_a.m from Dynare would be found[4].

## 4 The results

It may look like this SilverBullet doesn't change much, but in fact it provides an impressive performance boost, especially with big models[5]:

| model | equations | global parameters | task | Dynare time (hours) | Dynare+SilverBullet time (hours) | speed factor |
|---|---|---|---|---|---|---|
| Gali and Monacelli (2005) | 10 | 11 | 1M iterations of M-H | 9.0 | 6.0 | 1.5x |
| Core BEMOD | 84 | 102 | 100K iterations of M-H | 22.5 | 2.3 | 9.9x |
| BEMOD | 180 | 278 | initial optimization | 19.6 | 3.8 | 5.2x |
| BEMOD | 180 | 278 | 10K iterations of M-H | 23.0 | 3.0 | 7.6x |

And it is very simple to implement: you just need to open the ff1_.m and MyModel_ff.m file, change the names of the variables, and copy the appropriate parts of them into a new jacob_a.m file.

Better still, as these changes are always the same, they can be automated, which is what the macro inside my SilverBullet.xls[6] file does.

[3] Dynare also contains a file called fff1_.m, and creates a MyModel_fff.m, which are similar to their ff analogues, but include some additional calculations related to irrational numbers. If your model is calling those functions, you might consider including an "if function to evaluate is fff1_" block for them in the modified jacob_a.m file. Also, for slightly better performance, the first evaluation of the function could also be done inside jacob_a.m instead of through a call to the respective m-file.

[4] Note that this method would preclude the coexistence of several models in a given folder, as that could lead to the jacobian of the wrong MyModel_ff.m function to be evaluated.

[5] These tests have been conducted on a PC with an Intel Core2duo working at 2.2GHz and with 4GB of RAM.

[6] This file is freely available for those who request it at samuel.hurtado at bde.es (spam protection). Together with Dynare for Matlab v.3.065, it is working flawlessly with all the models I have tested it on, but universal compatibility obviously cannot be guaranteed. Any users should check that, for their particular models, results are numerically identical with or without this SilverBulet (please remember to set a seed for the random number generators at the beginning of your code), while also understanding which changes are being made with respect to the standard Dynare codes, and why.