NS2015 – Revision

Michele Fornino

March 26, 2017

Introduction

This documentation concerns the revision of the paper “High-Frequency Identification of Monetary Non-Neutrality” by Emi Nakamura and Jón Steinsson, that was carried out between March and August 2016. The subjects of this revision were the estimation of the key parameters of a NK model augmented with an information effect and the modification of the Simulated Method of Moments estimation with the inclusion of GDP growth expectations moments and a special bootstrap procedure to compute confidence intervals for the estimated parameters.

The Matlab program consists of a script file called `run.m` and a host of functions that are called within the script to perform the basic tasks of estimation.

Estimation

The parameters of the model that we wish to estimate are:

- ψ: share of monetary shock that reflects the natural rate
- $\kappa\zeta$: slope of Phillips curve
- ρ_1 and ρ_2: autoregressive coefficients in the monetary policy rule

The above parameters are estimated through a two-step Simulated Method of Moments procedure. In particular, we first look for the values of ρ_1 and ρ_2 which minimize a quadratic loss function of the deviations of real forwards and real yields implied by the simulated model from those in the data. The loss function’s weighting matrix is constructed as a diagonal matrix listing all the inverse standard errors of the moments estimated in the data.

Second, we feed these values of ρ_1 and ρ_2 in another minimization, where we look for the values of ψ and $\kappa\zeta$ which again minimize a quadratic loss function. This time, though, only the inflation and GDP expectation moments are targeted by the loss function, using again a diagonal matrix constructed with the inverse of the standard deviation of the moments. At the end of the second stage, we have provisional values of the parameters that can be fed again in another iteration until convergence is attained.

The minimization procedure is implemented through the `fmincon` routine in Matlab, with parameter values restricted within reasonable bounds. In particular, for $\varepsilon = 0.001$, $\rho_1, \rho_2 \in [-1+\varepsilon, 1-\varepsilon]$, and $\psi, \kappa\zeta \in [\varepsilon, 1-\varepsilon]$.

Bootstrap

In order to gauge the statistical confidence in the parameter estimates obtained with the methodology described above, we compute quantiles of the relevant distributions by an ad-hoc resampling method very similar to the conventional bootstrap. In particular, the main issue is that we start out with two separate datasets from which we compute the moments used in the SMM procedure: the first one is at a higher frequency and not only includes FOMC meeting days, but also other days in the month; the second dataset contains GDP growth expectations from the Blue Chip survey and it is at monthly frequency.

Therefore, at each iteration of the resampling routine, we make sure to draw two independent datasets that can be used to compute all of the moments of interest and then run the conventional SMM procedure to obtain both parameter estimates and impulse response functions. Stratification is used extensively to ensure that the bootstrapped datasets are consistennt with the original ones. For instance, both the proportion of FOMC meeting days and of the observations after and before 2004 are kept constant.[1] Quantiles and other relevant statistics are then easily computed.

Matlab Code Documentation

The following paragraph describes the implementation of the above procedure within Matlab.

The main routine

The main routine that must be executed is `run.m`. Both this routine and all of the relevant subroutines are commented extensively in order to allow the programmer to quickly understand what each part of the code does. We refer the programmer to the comments to understand in detail what each option of the first block of `run.m` does and how to set it up. It is structured in four main sections:

1. *Housekeeping and Settings*: calls that populate a structure object, called `par`, containing all of the parameters of interest and the instructions for the graphs and robustness cases. This is useful because it can be passed on as a unique argument to subroutines. It is an alternative to defining many global variables and has the main advantage that it can be manipulated by subroutines which pass it on to subsubroutines, but without altering its definition in the workspace of the parent script. The structure contains many settings, auxiliary cell variables, and a few substructures of interest:

   (a) `calibration`: contains the calibrated values of some parameters of the model;

   (b) `options`: contains the settings of the `fmincon` minimization program, created with `optimset`.

   (c) `cases` and `robustness`: this is an important dichotomy. "Cases" refers to a set of estimation runs, for instance changing calibrated parameters, or calibrating rather than estimating the habits parameter. "Robustness" refers to exploring, e.g., IRFs for a given set of estimated parameters but changing some aspects of the model without carrying out the estimation phase each time. Currently, these robustness may allow for optimal or alternative monetary policy rule, and change the values of estimated parameters. For both, the user may instruct the code to perform as many custom runs as wanted. The main advantage is that this does not require the user to change by hand the settings to check each version of the model separately;

   (d) `figures`: contains settings for the figures, such as whether to save to file and how long the IRFs should be;

   (e) `bootstrap`: seed and number of draws for the bootstrap procedure;

2. *Moments Estimation and SMM Estimation of the Model(s)*: Calls to the three main subroutines that perform the point estimation using the SMM procedure described in the paper:

   (a) `loaddata.m`: creates a structure called `data` which contains the datasets used to compute the empirical moments that must be targeted. These are dd1, dd2, and ddGDP. They vary because moments are estimated on different datasets depending on data availability and frequency.

[1]This is relevant because for some moments we are only able to compute the effect of the path factor for the period after 2004.

(b) `momentestimation.m`: creates a structure called `moments` which lists all of the empirical moments that are relevant for the SMM procedure and that we wish to match. These regressions are analogous to the ones in the empirical part of the paper. The reason why we cannot simply use the moments estimated in the empirical part is that this function will be called by the bootstrap routine to perform the empirical analysis on resampled datasets, and the procedures on the true and resampled samples must be exactly the same.

(c) `modelestimation.m`: creates a structure called `model`, which contains information about the estimation procedure and the model for the estimated values of the parameters, stored in three substructures:

   i. `estimation`, which contains technical information about the minimization procedure, for instance the flags, the value of the loss function at optimum, and the analysis of the fit in terms of how well the moments are targeted.

   ii. `irfs`, which contains the IRFs to a monetary shock for the baseline mode, the counterfactual version in which the public already knows of the growth in natural output, and the difference.

   iii. `parameters`, which contains parameter values divided in `calibrated` and `estimated`.

3. *Draw Figures*: Call to `figures.m`, a subroutine that creates graphs of IRFs based on the point estimates of the parameters. It creates a folder structure within the subfolder `output`, where all of the figures can be found in EPS format.

4. *Bootstrap*: Calls to the two main subroutines that compute the bootstrapped distributions of the estimated parameters. They contribute to the creation of a structure called `bootstrap`, built as follows:

   (a) `bootstrapmomentsmodel.m`: creates the substructure `distributions`, which in turn contains substructures `moments`, `irfs`, and `parameters`, each one of them containing a whole bootstrapped distribution.

   (b) `bootstrapstats.m`: creates a substructure `statistics`, which in turn contains substructures `means`, `stdev`, `quantiles`, each one containing the relevant moments of the boostrapped distributions for the objects of interest (i.e., `moments`, `irfs`, and `parameters`). The functioning of this routine is straightforward.

The following describes in some detail how the routines `modelestimation.m` and `bootstrapmomentsmodel.m` work:

1. `modelestimation.m`: in itself, this function is just a wrapper. The structure model described above that is saved by this routine is of dimension given by Number of cases $\times$ Number of robustness. For each combination of cases and robustness checks, the following functions are called:

   (a) `structuralestimation.m`: First, it defines the weighting matrix as a diagonal of the inverse standard deviation of the empirical moments. Second, the two step minimization procedure is implemented as described in the paper. Finally, the output of the minimization procedure is analyzed and stored: for instance, the loss function is decomposed by empirical moments to understand which ones fit better then others etc. The objective function of the minimization procedure is a function called `objectiveEstINFO.m`: in itself, it is a quadratic loss function in the deviation of the empirical moments from the theoretical moments. Crucially, it is possible to compute this loss function only for a subset of the moments. The theoretical moments are computed using the `gensys.m` routine (and its dependencies) developed by Chris Sims, itself called in the specification file `modelINFO.m` which contains the linearized version of the model. The empirical moments, instead, are passed on as argument to this function, and are the ones outputted by `momentestimation.m`;

   (b) `modelIRFs.m`: using the solution of the model and the estimated parameters, it computes IRFs by iterating forward on the system of difference equations;
   (c) `modelParameters.m`: auxiliary routine that saves the parameters in the model.parameters structure that is outputted by the modelestimation routine.

2. `boostrapmomentsmodel.m`. Note: this function does not operate for the cases/robustness lists defined in par, but only for the "custom run", which essentially is a single run of the code for a given parametrization. The function operates in three steps:
   (a) `boostrapdataset.m`: Create strata variables by amending the data structure created by the loaddata.m routine. These are described in the appendix to the paper.
   (b) Run boostrap itself as many times as number of bootstrap repetitions required:
      i. `boostrapsampling.m`: this is the routine that performs the resampling by strata.
      ii. `momentestimation.m`: same as above, but now feed it the resampled dataset.
      iii. `modelestimation.m`: same as above, but now feed it the resampled dataset.
   (c) allocate distributions of objects of interest within `bootstrap.distributions` structure.

Parallel Computing and Replicability

A remark is due on parallel computing. Note that the code is written to take advantage of parallel computing, especially for the bootstrap routine. There are two places within the code where parallel computing can be used: calls to `fmincon`, and calls to `boostrapmomentsmodel.m`. The first one is determined by the option `useparallel` set by `optimset`. The second one is determined by the option `par.bootstrap.parallel`. Note that the code is written in such a way that automatically excludes the possibility that `fmincon` is called with the option `useparallel` set to true when the same is requested of `boostrapmomentsmodel.m`. Parallel computing is extremely effective for the bootstrap procedure and should be used whenever possible.

One issue with parallel computing is replicability. Unfortunately, it is impossible right now to have the bootstrap run on many CPUs while at the same time retaining the ability to replicate the bootstrap runs. The fundamental reason is that the order with which CPUs will draw random numbers is not predictable and depends on the machine used. Therefore, if replicability is key, then we urge the user to set `par.bootstrap.parallel` to false.

Example

Here is an example of how to use the code. We suppose the user wishes to estimate the model by targeting only the inflation moments, calibrate the habits parameter to .8, estimate the information parameter. In addition, the user wants to have 2.5th and 97.5th quantiles of the estimated parameters to build a 95% confidence band. This first task we will call baseline. The user is also curious of what would happen to point estimates if instead of estimating the information parameter, this were calibrated to .3, say, but is not interested in investing the time to have confidence bands around these alternative parameters.

These tasks can be performed in two ways:

1. Easier way, but longer:
   (a) Step 1. Perform point estimation and bootstrap for the baseline analysis:
      i. set `par.momentUse = 2`, telling the code to only use inflation moments in the estimation procedure, and to ignore both the stock price and GDP moments.
      ii. set `par.B_PSI_est = 1`, telling the code to calibrate the habits parameter and estimate the information parameter;
      iii. modify the 5th entry of `par.x0`, and set it to .8, the value chosen for the calibration of the habits parameter.

   iv. set `par.bootstrap.draws = 500`, and `par.bootstrap.seed` = an integer number of your choice, useful for replicability (see the previous paragraph for details about parallel computing and replicability);
   v. make sure that `par.cases.run = 0` and `par.robustness.run = 0`.
   vi. execute `run.m` and make sure to rename the .mat file created by the code

(b) Step 2. Perform only point estimation for the alternative analysis (I assume that the run.m file is left as modified in Step 1 above):

   i. set `par.B_PSI_est = 3`, telling the code to calibrate both the habits and information parameters.
   ii. modify the 4th entry of `par.x0`, and set it to .3, the value chosen for the calibration of the information parameter in the alternative exercise.
   iii. make sure that `par.cases.run = 0` and `par.robustness.run = 0`.
   iv. execute `run.m` but for the last section, and make sure to rename the .mat file created by the code differently from the one in Step 1

2. Faster way, a little harder to set up. Crucially, here we see the fact that bootstrap does only work with the "custom run", that is it does not interact with the `par.cases` and `par.robustness` blocks:

   (a) set `par.momentUse = 2`, telling the code to only use inflation moments in the estimation procedure, and to ignore both the stock price and GDP moments.
   (b) set `par.B_PSI_est = 1`, telling the code to calibrate the habits parameter and estimate the information parameter;
   (c) modify the 5th entry of `par.x0`, and set it to .8, the value chosen for the calibration of the habits parameter.
   (d) set `par.bootstrap.draws = 500`, and `par.bootstrap.seed` = an integer number of your choice, useful for replicability (see the previous paragraph for details about parallel computing and replicability);
   (e) set:
      i. `par.cases.run = 1`.
      ii. `par.cases.names = {'baseline', 'calibrateInfo'};`
      iii. `par.cases.chgparams = {'sigma'};`[2]
      iv. `par.cases.sigma = [.5; .5]` or anyway to the value chosen for `par.calibration.sigma`;
      v. `par.cases.B_PSI_est = [1; 3]`, telling the code that in the first case you'd like it to have the habits parameter calibrated but estimate the information parameter, and in the second case to calibrate both.
      vi. Set a 2×5 matrix `par.cases.x0`, where the first row contains the same vector as `par.x0`, but for the 5th entry that must be set to .8 (the calibrated value for the habits parameter). The second row can be the same as `par.x0`, but for the 4th and 5th entries that must be set to .3 and .8.
   (f) make sure that `par.robustness.run = 0`;
   (g) execute `run.m`, noting that bootstrap will only be run for the "custom run" as if we had set `par.cases.run = 0`.

How to access the stored results?

[2]This is done because the typical exercise is to change parameters. In this case we don't want to explore what happens to estimation if we change parameters, but cannot leave this option empty.

1. Estimated parameters: type `model.parameters.estimated`

2. Loss function and its decomposition: type `model.estimation.loss` and `model.estimation.lossAnalysis`

3. Figures (if drawn and saved): look for folder `output` within the current directory.

4. Quantiles of estimated parameters: type bootstrap.statistics.quantiles.parameters."..." where "..." is the name of the parameter of interest for which the quantiles are wanted, which can be AR1, AR2, slopePC, PSI. Recall that this is a 13×1 vector that lists the [.005 ,.01, .025, .05, .1, .25, .5 ,.75, .9, .95, .975, .99, .995] quantiles. Therefore, the user may extract the 3rd and 11th elements of this vector as the lower and upper bounds of the confidence band for each parameter.

Addendum: SMM Weighting Matrix

Miguel Acosta, May 29, 2018

Note that the weighting matrix used for the SMM estimation uses homoskedastic standard errors, not the heteroskedastic (robust) standard errors reported in the paper.