BANK OF ENGLAND

CENTRE FOR CENTRAL BANKING STUDIES

ECONOMIC MODELLING AND FORECASTING

Simulating and forecasting a monetary policy model (without money) for India in EViews

by

Andy Blake and Ole Rummel
Centre for Central Banking Studies
Bank of England

11 February 2015

1 Introduction

The aim of this exercise is to use an estimated new Keynesian (NK) macroeconomic model for India for simulation and forecasting. Once a model has been estimated, various experiments can be run to examine its properties. These examinations are not tests of the model *per se*, but simply finding out what the estimated equations and identities imply.

As such, we are trying to replicate – and extend – Section V in IMF Working Paper WP/10/183 by Patra and Kapur (2010).[1] Using an augmented version of the canonical three-equation NK model, the authors find that their estimated macroeconomic model yields valuable insights into the functioning of the Indian economy, most notably the monetary transmission mechanism. This is of great importance in light of the fact that, as outlined in more detail in the paper, the monetary policy framework in India has undergone fundamental modifications over the period under observation.

Having taken the theoretical model to (emerging-market) data, we will use the estimated model for policy analysis. Specifically, we will:

- assess model dynamics in response to anticipated as well as unanticipated shocks to both endogenous and some exogenous model variables; and
- use the model to (conditionally) forecast the three endogenous variables in the model and put a fanchart around the forecast.

2 The model

We will simulate and forecast the three-equation NK model consisting of the preferred specification from Patra and Kapur (2010), which involves an estimated backward-looking new Keynesian aggregate demand or dynamic IS curve augmented with open-economy variables (equation (4) of their paper), an estimated backward-looking extended aggregate supply or new Keynesian Phillips curve in equation (2) of their paper and an estimated forward-looking baseline augmented monetary policy reaction function (Taylor rule) in equation (6) of their paper. The estimated coefficients of the equations for the three endogenous variables `ygapsa` (the seasonally adjusted output gap), `infgdp` (inflation as measured by the year-on-year change in the GDP deflator) and `effective3` (the policy rate) can be found in Table 1 (column 4), Table 3 (column 4) and Table 4 (column 3) in the paper respectively. The three model equations are reproduced below for convenience (variable names, their derivations and other explanations can be found in the Appendix to this exercise):

```
ygapsa = 0.38 - 0.15*rpr(-3) + 0.28*ygapsa(-1) + 0.21*ygapagrsa
              + 0.12*wexprgapsa - 0.06*reer36gapsa(-2)              (1)

infgdp = 0.99 + 0.20*ygapsa(-4) + 0.68*infgdp(-1) + 0.07*infglobal
              + 3.11*dum1998q3q4 + 0.05*dlexcha                     (2)

effective3 = 1.49 + 0.24*infgdpdev(+2) + 0.24*ygapsa(+2)
                  + 0.77*effective3(-1) + 0.02*dlexch4(-1)          (3)
```

Note that we have three equations in three endogenous variables (`ygapsa`, `infgdp`, `effective3`), including lagged endogenous variables and – what appear to be – nine exogenous variables (`rpr`, `ygapagrsa`, `wexprgapsa`, `reer36gapsa`, `infglobal`, `dum1998q3q4`,

[1] A PDF version of the IMF Working Paper, entitled **wp10183.pdf**, can be found in the current course **Data** folder.

`dlexcha`, `infgdpdev`, `dlexch4`). Before we can simulate and forecast the model, though, we need to realise that we have to **close** it, in the sense of providing as many equations as there are endogenous variables in the model. This is necessary as the model contains three more relationships (identities, to be precise) between the variables. First, there is a relationship between the deviation of GDP inflation from an indicative inflation target of 5 per cent for India (`infgdpdev`) and the year-on-year change in the GDP deflator (`infgdp`). Second, the real policy rate (`rpr`) is the difference between the nominal policy interest rate (`effective3`) and the year-on-year change in the GDP deflator (`infgdp`). Finally, the (annualised) year-on-year change in the log exchange rate (`dlexcha`) is a function of the (annualised) quarter-on-quarter change in the log exchange rate (`dlexch4`). We therefore need to add the following three identities to the model to close it:

```
infgdpdev = infgdp - 5

rpr = effective3 - infgdp

dlexcha = (dlexch4 + dlexch4(-1) + dlexch4(-2) + dlexch4(-3))/4
```

The final model has six linear equations (three stochastic equations and three identities), which include six endogenous as well as six exogenous variables and a maximum lag (lead) of 4 (2).

3 Dealing with model-consistent expectations in EViews

Manipulating, solving, simulating and forecasting (non-linear) models with forward-looking variables and/or explicit expectations terms in EViews has recently become much easier following the release of simulation programmes for EViews by the Federal Reserve (http://www.federalreserve.gov/econresdata/frbus/us-models-package.htm). In particular, the Federal Reserve provides a rational expectations (RE) solver package for EViews called mce_solve that solves linear and non-linear models under model-consistent expectations.

The EViews solution algorithms provided by the Federal Reserve, which are accessed by calling the `mce_run` routine from within an EViews programme, use the E-Newton and E-QNewton algorithms, which are specifically designed to impose model-consistent or rational expectations (RE) in simulations of macro models. In a nutshell, these algorithms iterate to find a model's RE solution with a sequence of updates to either exogenous estimates of the model's future-dated endogenous variables or exogenous components of such estimates. The (gory) technical details can be found in Brayton (2011).[2]

The use of the two algorithms typically involves a sequence in which:

- the macro model is set up according to the general requirements of the algorithm;
- one of the two algorithms is selected and its parameters are set by the researcher; and
- a simulation experiment is specified and executed

The general syntax of `mce_run` takes three arguments: these designate options and inputs for specifying the model (`m_opts`), the algorithm (`a_opts`) and the simulation (`s_opts`):

```
call mce_run(m_opts, a_opts, s_opts)
```

[2] A PDF version of the FEDS paper, entitled **FEDS_2011-44.pdf**, can be found in the current course **Data** folder.

We will return to the use of `call mce_run` and its optional arguments in Section 5 below, where their use will be illustrated.

4 The data

The EViews workfile **imf_india.wf1** in the current course **Data** folder contains all the (quarterly) data and other objects, such as estimated equations and models that are needed for the exercise. In particular, it contains the three estimated EViews equation objects that represent equations (1), (2) and (3). The workfile need not necessarily be open at this point, as the EViews programme that we will run below will open it for us. The one thing we have to make sure of is that a folder called **subs**, the **imf_india.wf1** workfile, a programme called **india_mce.prg** and a programme called **stochsim.prg** can all be found in the same folder.

5 Simulating the model

Remember from the presentation that simulations with RE models have to be explicit about the nature of the shocks. One aspect to keep in mind is whether the shocks under investigation are anticipated or unanticipated by the policymakers and the other agents in the economy. This is because, with rational expectations, there is a difference between these alternative assumptions which arises from the system solution of the model.

5.1 Responses of endogenous and exogenous variables to anticipated shocks

The first thing we will do is to simulate the model and look at the model dynamics in response to **anticipated** shocks to the three endogenous variables inflation (`infgdp`), the output gap (`ygapsa`) and the policy rate (`effective3`), as well as to some of the exogenous variables (world exports (`wexprgapsa`), the nominal exchange rate (`dlexcha`) and the real exchange rate (`reer36gapsa`)).
In particular, Patra and Kapur (2010) look at:

- a 1 per cent increase in the output gap;
- a 1 per cent increase in inflation; and
- a 1 per cent increase in the policy rate

The authors then go on to consider anticipated shocks to some of the exogenous variables. In particular, they are interested in:

- a 10 per cent decline in world exports (`wexprgapsa`);
- a 10 per cent increase in global commodity inflation (`infglobal`);
- a 10 per cent nominal appreciation of the Indian rupee versus the US dollar (`dlexcha`); and
- a 10 per cent real appreciation of the Indian rupee (`reer36gapsa`).

These – and other – simulations will be illustrated using the EViews **india_mce.prg** programme, which is reproduced below in annotated sections:

```
' Three-equation new Keynesian model of the Indian economy

mode quiet
```

```
open imf_india

%mod = "India"

' Subroutines
include subs/mce_solve_library
```

The first part of the programme starts off by setting EView's execution mode to `quiet`. This suppresses the display of information such as commands in the status line as they are executed. As a result, EViews will run faster in `quiet` mode since the status line display does not need to be continuously updated. It then opens the EViews workfile **imf_india.wf1**, which contains – in particular – information about the estimated equations that we will define in the next part of the code. The programme then assigns the value 'India' to the string variable, `%mod`, such that the place-holder `%mod` will be replaced by `India` every time EViews encounters it. Finally, EViews calls the **mce_solve_library.prg** subroutine that can be found in the `subs` folder. This reads in all the available pre-defined solution routines in that particular library.

```
' Retrieve estimated coefficients
*****************************************

' New Keynesian aggregate demand or dynamic IS curve coefficient
estimates
preferred_model_1.updatecoef
coef alpha = c

' Aggregate supply or new Keynesian Phillips curve coefficient
estimates
preferred_model_2.updatecoef
coef beta = c

' Monetary policy reaction function or Taylor-rule coefficient
estimates
preferred_model_3.updatecoef
coef gamma = c
```

This part of the programme retrieves the estimated coefficients from three equation objects in the **imf_india.wf1** workfile. In particular, it retrieves coefficient information from an EViews equation object called `preferred_model_1`, which is the estimated new Keynesian aggregate demand or dynamic IS curve. The estimated coefficients are retrieved from the `c` series object associated with that equation and then assigned to a coefficient vector `alpha` for later use. Similarly, it retrieves coefficient information from an EViews equation object called `preferred_model_2`, which is the estimated aggregate supply or new Keynesian Phillips curve. The estimated coefficients are assigned to a coefficient vector `beta` for later use. Finally, it retrieves coefficient information from an EViews equation object called `preferred_model_3`, which is the estimated monetary policy reaction function or Taylor rule. The estimated coefficients are assigned to a coefficient vector `gamma` for later use.

```
' Create model
***********************************************************
```

```
model {%mod}


' New Keynesian aggregate demand or dynamic IS equation
{%mod}.append ygapsa = alpha(1) + alpha(2)*rpr(-3) + alpha(3)*ygapsa(-
1) + alpha(4)*ygapagrsa + alpha(5)*wexprgapsa + alpha(6)*reer36gapsa(-
2)

' Aggregate supply or new Keynesian Phillips curve
{%mod}.append infgdp = beta(1) + beta(2)*ygapsa(-4) + beta(3)*infgdp(-
1) + beta(4)*infglobal + beta(5)*dum1998q3q4 + beta(6)*dlexcha

' Monetary policy reaction function or Taylor-rule
{%mod}.append effective3 = gamma(1) + gamma(2)*infgdpdev(+2) +
gamma(3)*ygapsa(+2) + gamma(4)*effective3(-1) + gamma(5)*dlexch4(-1)

' Identity for infgdpdev
{%mod}.append infgdpdev = infgdp - 5

' Identity for rpr
{%mod}.append rpr = effective3 - infgdp

' Identity for dlexcha
{%mod}.append dlexcha = (dlexch4 + dlexch4(-1) + dlexch4(-2) +
dlexch4(-3))/4
```

This part of the programme creates the EViews **model** object that we want to simulate and forecast later on. A model in EViews is a set of one or more equations that jointly describe the relationship between a set of variables. Overall, EViews models allow us to combine equations inside a single object, which may then be used to create a deterministic or stochastic joint forecast or simulation for all the variables in the model. The first line creates a new EViews **model** object and, *via* the `{%mod}` command, assigns the name **India** to it. It then creates the six equations of the India model object. Using the `.append` syntax, we slowly build up the required EViews model object. The new Keynesian aggregate demand or dynamic IS curve will provide an illustration of how this is done. This is the first equation of the model, which is simply written in the customary EViews way. The one difference is that we recover the coefficient estimates from the `alpha` vector we created in the previous step. All in all, the vector `alpha` contains six elements: the first is the estimated coefficient on the constant (`alpha(1)`), the second is the estimated coefficient on `rpr(-3)` (`alpha(2)`), the third is the estimated coefficient on `ygapsa(-1)` (`alpha(3)`), etc. Once the coefficients have been substituted, the model equation for `ygapsa` will be:

```
ygapsa =   0.383188392432 - 0.146848613221*rpr(-3)
         + 0.284463647328*ygapsa(-1) + 0.207318257846*ygapagrsa
         + 0.115359626484*wexprgapsa - 0.057314952378*reer36gapsa(-2)
```

The other equations of the model are created – and then appended to the existing model – along the same lines. The three identities at the end do not contain any estimated coefficients, so the equations can be entered as they appear above and then appended to the existing **India** model object.

```
' Simulation control ******************************************

%mstr = "create,mod="+%mod+",adds,track"
%astr = "meth=newton,jinit=linear"
%sstr = "type=single,o=3,scen,suf=_"

%start = "1999q4"
' %shk   = %start
%shk   = "2000q2"
```

The next part of the programme sets the simulation controls. More specifically, it specifies the options that will serve as the input for the `mce_run` routines in EViews. These options are further explained in the document entitled **mce_solve_users_guide.pdf** that can also be found in the current course **Data** folder.

Let us start with `%mstr`, which needs to contain the keyword `create`, the keyword assignment `mod = ⟨name1⟩` and that a model with the name `⟨name1⟩` exists in the workfile. In the example at hand, `⟨name1⟩` will be **India**, and a model with this name does indeed exist in the workfile – remember that we have created it ourselves in the early part of the programme. Inclusion of the `adds` keyword causes **add factors** to be assigned to all equations, while the inclusion of the `track` keyword causes the values of the **add factors** to be initialised so that the equations have no errors when evaluated using actual data.[3] In other words, the values of the **add factors** are chosen so that these equations track the baseline data perfectly over the current workfile sample.

The second subroutine argument of `mce_run`, associated with `%astr`, is used to choose a solution algorithm and set related options. In this group, the `meth` (method) keyword specifies the solution algorithm: the default setting is `newton` for the E-Newton solution algorithm. The `jinit` keyword controls the construction of the initial Jacobian or approximate Jacobian matrix.[4] Using `jinit = linear` means that the Jacobian matrix is initialised (and updated) using the linear shortcut that is applicable to many linear model-consistent expectations models.

The `%sstr` arguments control the simulation type, execution and related parameters. In particular, the `mce_run` subroutine can initiate three types of model-consistent expectations simulation experiments. As we have chosen the default setting of `type = single`, I will discuss this approach very briefly, referring the interested reader to the aforementioned document for a more extensive discussion of the other two options (`opt` and `opttc`). Setting `type = single` runs a model-consistent expectations simulation whose inputs are the initial and terminal values of the model's variables and the projected paths of any shocks and exogenous variables the model may contain. The other two options, `o = 3` and `suf = _`, set the amount of output per model-consistent expectation simulation and an alias for a new scenario respectively. The most amount of output is displayed by setting `o = 1` and the least amount of output is produced with `o = 3`.

The final three lines define the starting period for the simulation (`%start`), which is 1999 Q4, as well as the period when the shock occurs (`%shk`). The latter is important when analysing the effects of anticipated *versus* unanticipated shocks. In order to calculate the impact of an **anticipated** shock in a forward-looking model, the simulation period should start before the shock is introduced. Conversely, in

[3] An add factor is the adjustment made to an equation-based projection over the forecasting period. For example, if an equation has under-predicted a variable in recent periods, then an 'add factor' may be added to the equation if it is judged that the equation will under-predict over the forecast period as well. In short, add factors are equation-residuals applied over the forecast period that make the model fit the data.
[4] In vector calculus, the Jacobian matrix is the matrix of all first-order partial derivatives of a vector-valued function.

order to calculate the impact of an **unanticipated** shock in a forward-looking model, the simulation period should start at the same time as the shock is introduced. This set-up ensures that the shock is truly unanticipated, as the solution algorithm ‘sees’ the shock at the same time as it is introduced into the model. With the current settings, the timing of the shock, here set by `%shk`, is 2002 Q2, i.e., the solution period starts **before** the shock is introduced, so this simulation set-up is for an **anticipated** shock.

```
' Initialise  ************************************************

smpl {%start} @last-2
call mce_run(%mstr,%astr,%sstr+"000")

%pvars = "infgdp ygapsa effective3"
%diff  = ""
for %v {%pvars}
    %diff = %diff+"("+%v+"_XYZ-"+%v+")"
next
```

This part of the code sets the sample period to run from `{%start}`, i.e., 1999 Q4, to the end of the sample period minus two observations (`@last - 2`). It then calls `mce_run`, having defined the three input arguments `%mstr`, `%astr` and `%sstr` above.

The final five lines loop over the three endogenous variables defined by `%pvars` (`infgdp`, `ygapsa`, `effective3`) and create three new strings or text labels: (`infgdp_XYZ - infgdp`), (`ygapsa_XYZ - ygapsa`) and (`effective3_XYZ - effective3`). The reason for this will become apparent in the next part of the code.

```
' Impulse responses *******************************************

%var = "ygapsa effective3 infgdp dlexcha wexprgapsa reer36gapsa"

%chk = {%mod}.@endoglist

for %v {%var}

  %vs = @left(%v,3)
  %sub = ""
  if @wfindnc(%chk, %v) > 0 then
     %sub = "_a"
  endif

  smpl {%shk} {%shk}
  series {%v}{%sub} = {%v}{%sub} + 1

  smpl {%start} @last-2
  call mce_run("","",%sstr + %vs)

  smpl {%shk} {%shk}
  series {%v}{%sub} = {%v}{%sub} - 1

  smpl {%start} @last-2
```

```
    %p = @wreplace(%diff,"*XYZ*","*"+%vs+"*")
    group {%vs}_sim {%p}
    graph {%v}g.line {%vs}_sim

next
```

The main purpose of the **india_mce.prg** programme is to investigate the responses of the three endogenous variables to a number of both endogenous and exogenous shocks. The variables that are being shocked are defined by `%var`, i.e., the three endogenous variables themselves as well as three of the exogenous variables, in particular `dlexcha`, `wexprgapsa` and `reer36gapsa`. The reason we do this is to replicate Charts 6 and 7 in Patra and Kapur (2010, pages 42-43), which are reproduced below for convenience.

Chart 6 : Responses to Shocks to Endogenous Variables: Deviation from the Baseline

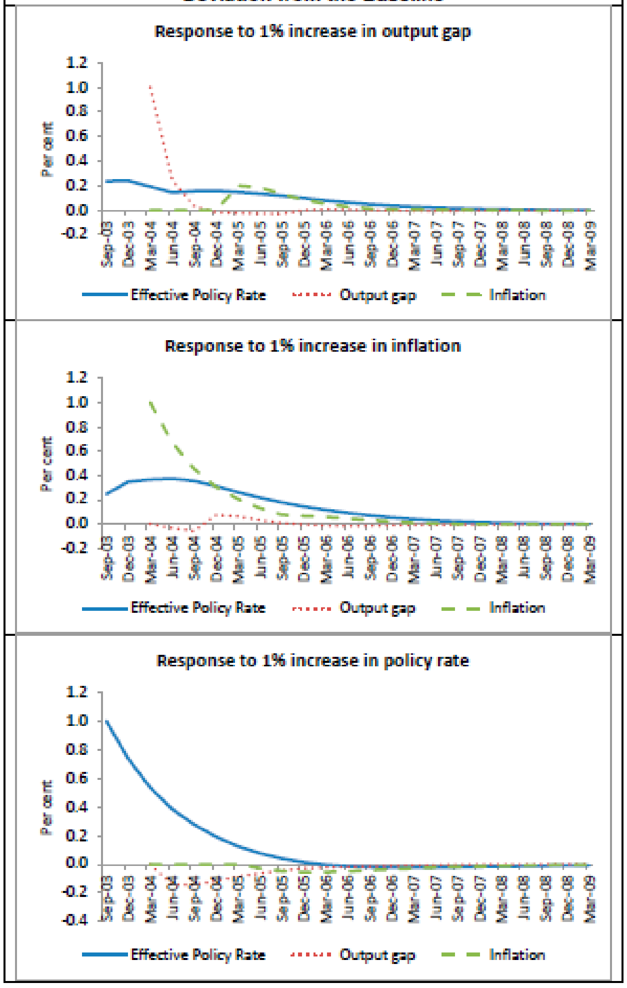
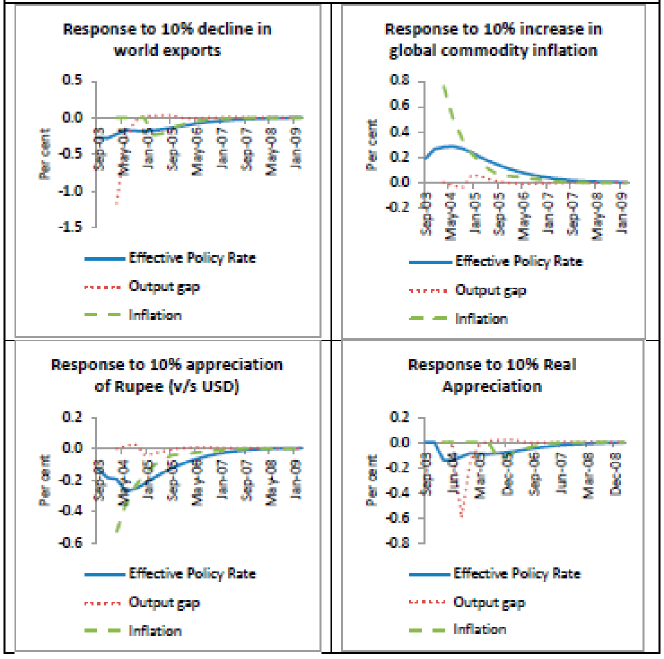


Chart 7 : Responses to Shocks to Exogenous Variables: Deviation from the Baseline



This part of the programme loops over each of the six variables in `%var`, enters the shock into the model, solves the model by calling `run_mce` again, removes the shock from the model and replaces the strings (`infgdp_XYZ - infgdp`), (`ygapsa_XYZ - ygapsa`) and (`effective3_XYZ - effective3`) by actual data. In essence, this creates the difference between the simulated values (`infgdp_XYZ`), say, and the baseline data (`infgdp`), which is the proper calculation for creating the response to a shock or, equivalently, an **impulse response function**. The code spends a lot of effort on good house-keeping, i.e., making sure that the right simulation data goes into the right series and that we remove the shock after we have imposed it. Note that the shock size is set equal to 1, so that the resulting charts are not necessarily comparable with the responses to shocks in Chart 7 in Patra and Kapur (2010, page 43), which consider a 10 per cent shock up or down to the chosen exogenous variables.

```
' Show series

%m = @wcross(%var,"g")
graph all.merge {%m}
all.align(3,1,2)
show all

' Cleanup *****************************************************

delete _$_* err_* *_000
```

The last bit of the code generates the required graphs by first creating the relevant variables to be plotted. It does so by using EView's `@wcross()` command. This command returns a string list crossed with another string list according to a particular string pattern. The default pattern is that each element of the first string list should be crossed individually with each element of the second sting list. In the above example, `@wcross(%var, "g")` returns every combination of the elements `%var` and `"g"`, using the default pattern. This produces the string list of graphs associated with each of the six variables that are being shocked: `"ygapsag effective3g infgdpg dlexchag wexprgapsag reer36gapsag"`.

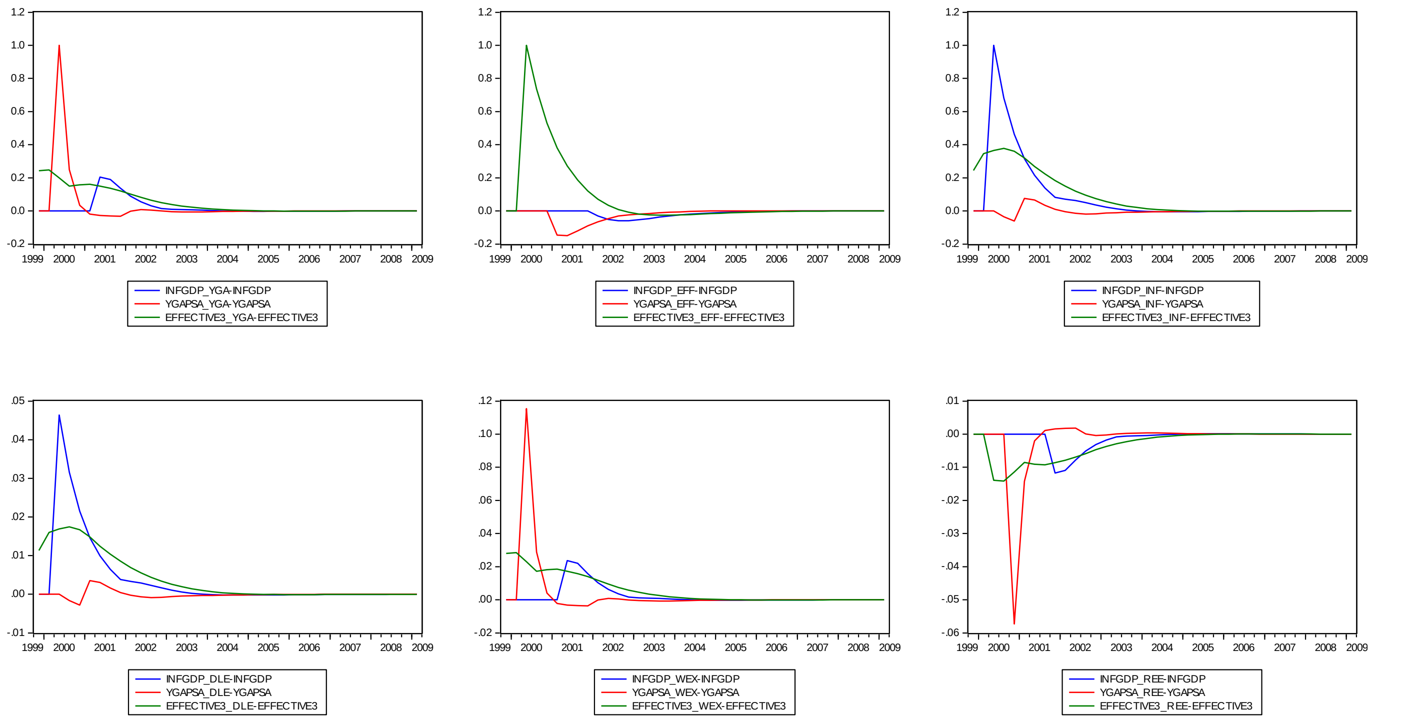
It is worth being a bit more specific about the shocks and how the variables are being shocked. As written, the model imposes an additive shock rather than a multiplicative shock. In principle, adjustments (or shocks), a_t, can be either additive, where $y_t = f(y_t, x_t) + a_t$, or multiplicative, $y_t = f(y_t, x_t)*(1 + a_t)$. A shock to an endogenous variable (such as `ygapsa`) is defined as a shock to the residual, while a shock to an exogenous variable (such as `reer36pagsa`) is defined as shock to the variable itself.

The final line of the programme cleans up the workfile by deleting all the series that contain `_$_` (these are created by `run_mce`), `err_` and `_000` which were created during the course of the execution of one or more `mce_runs`.

Pressing **Run** results in the six graphs reproduced in Figure 1 below. Each chart shows the impulse response functions of the three endogenous variables (`infgdp`, `ygapsa`, `effective3`) to each of the **anticipated** six shocks (`ygapsa`, `effective3`, `infgdp`, `dlexcha`, `wexprgapsa`, `reer36gapsa`) in that order. In other words, going from left to right, the top row shows the impulse response functions of the three endogenous variables to an anticipated unit shock to `ygapsa`, `effective3` and `infgdp` respectively, while the bottom row shows the impulse response functions of the three endogenous variables to an anticipated unit shock to `dlexcha`, `wexprgapsa` and `reer36gapsa` respectively. This can be seen from the labels to the charts. Taking the top left-hand chart as an example, the three labels are INFGDP_YGA – INFGDP, YGAPSA_YGA – YGAPSA and EFFECTIVE3_YGA – EFFECTIVE3. The first variable is the deviation of `infgdp` following a shock to `ygapsa` from its baseline, i.e., the actual data series `infgdp`. The three letter suffix to the first variable name (`infgdp`, say) is derived from the first three letters of the variable being shocked. In the case of `infgdp_yga`, it is obviously `ygapsa`. In the middle chart on top, the first variable has the suffix _EFF, indicating that `effective3` has been shocked.

As such, the three graphs in the top row should be compared to Chart 6 in Patra and Kapur (2010, page 42) and the bottom three graphs should be compared to Chart 7 in Patra and Kapur (2010, page 43).[5] We note that the figures showing the responses to shocks to endogenous variables (the top row of Figure 1) is equivalent to Chart 6 in Patra and Kapur (2010, p. 42). The bottom row, showing the responses to shocks to exogenous variables, on the other hand, is not comparable: while the shape of the impulse response functions will be the same, the size and magnitude of the deviations from baseline will differ.

Figure 1: Impulse response functions of the endogenous variables in response to anticipated unit shocks



Just to recap, while the shock occurs in 2000 Q2, we solve the model starting in 1999 Q4. This makes the shocks **anticipated**, i.e., the policymaker as well as the agents in the economy know that the shock will be occurring in 2000 Q2. They therefore adjust their behaviour in expectation of the shock, that is, given the lags and leads in the model, before the shock has even occurred. Let us look at some of the results in more detail, by starting with the top left graph, which shows the response to an anticipated positive unit shock in the output gap in terms of changes from a constant baseline. An (anticipated) positive unit shock to `ygapsa` in 2000 Q2 is relatively short-lived as far the variable itself is concerned: the effect of the shocks lasts for only two periods. This is because of the simple dynamics in the model. Inflation reacts with a four-period lag, while the policy rate rises immediately and pre-emptively (remember the lead in the monetary policy reaction function).

[5] Remember that the impulse responses to exogenous shocks are not strictly comparable for two reasons. The first is the magnitude and size of the shocks. As such, Patra and Kapur (2010) consider a 10 per cent decline in world exports (`wexprgapsa`), a 10 per cent increase in global commodity inflation (`infglobal`), a 10 per cent nominal appreciation of the Indian rupee versus the US dollar (`dlexcha`), and a 10 per cent real appreciation of the Indian rupee (`reer36gapsa`). The above programme considers a positive unit shock throughout. Two, the shocks to exogenous variables enter multiplicatively in Patra and Kapur (2010) and additively in the model above.

The monetary policy transmission lags are particularly evident from the shock to the policy rate (top middle chart). Output starts contracting three periods after the (anticipated) interest-rate shock. The output contraction reaches its trough after another quarter, after which it gradually returns to its baseline. Inflation takes a full seven quarters to respond to the interest rate shock, with a maximum impact after nine quarters. We can see that monetary policy has some impact on real variables such as output, even though there is no long-run impact as postulated by the theoretical arguments of the (long-run) neutrality of money.

Moving across to the right-hand chart in the top row, an (anticipated) increase in expected inflation in 2000 Q2 leads to a pre-emptive increase in the policy rate, resulting in an immediate rise in real interest rates. This leads to an initial contraction of aggregate demand and a negative output gap. As inflation moves above the policy rate, real interest rates become negative and the output gap turns positive, before closing again in the long run. Over time, inflation also comes down in response to higher interest rates, but it takes almost three years for all three variables to return to the baseline.

The authors then go on to consider anticipated shocks to some of the exogenous variables. Why should we be interested in shocks to exogenous variables? An important aspect of NK models is their stochastic nature. As such, every period, random exogenous shocks perturb the equilibrium conditions in the NK model, injecting uncertainty into the evolution of the economy and thus generating economic fluctuations. Without these shocks, the economy would evolve along a perfectly predictable path. Knowing the effect of exogenous shocks is therefore important in assessing the performance of the model.

What are the pertinent facts arising from the evaluation of the anticipated exogenous shocks? A depreciation of the nominal exchange rate increases inflation, where the latter effect is quite long-lasting (bottom left-hand graph). In turn, higher inflation induces a monetary tightening, as a result of which demand is temporarily depressed. On the other hand, a real appreciation shock lowers aggregated demand and reduces inflation with a lag of six periods (bottom right-hand graph). As the monetary policy reaction function is forward-looking, monetary policy eases pre-emptively (two periods before the output gap response and six periods before the inflation response).

In practice, however, the nominal and the real exchange rate shock will work together. In light of the rigidity in the price level, any nominal depreciation will automatically translate into a real depreciation as well. Although not modelled here, the dampening effect of the real appreciation on aggregate demand will be accompanied by external deficits, where the latter may potentially have consequences for external sector sustainability and financial stability.

5.2 Responses of endogenous and exogenous variables to unanticipated shocks

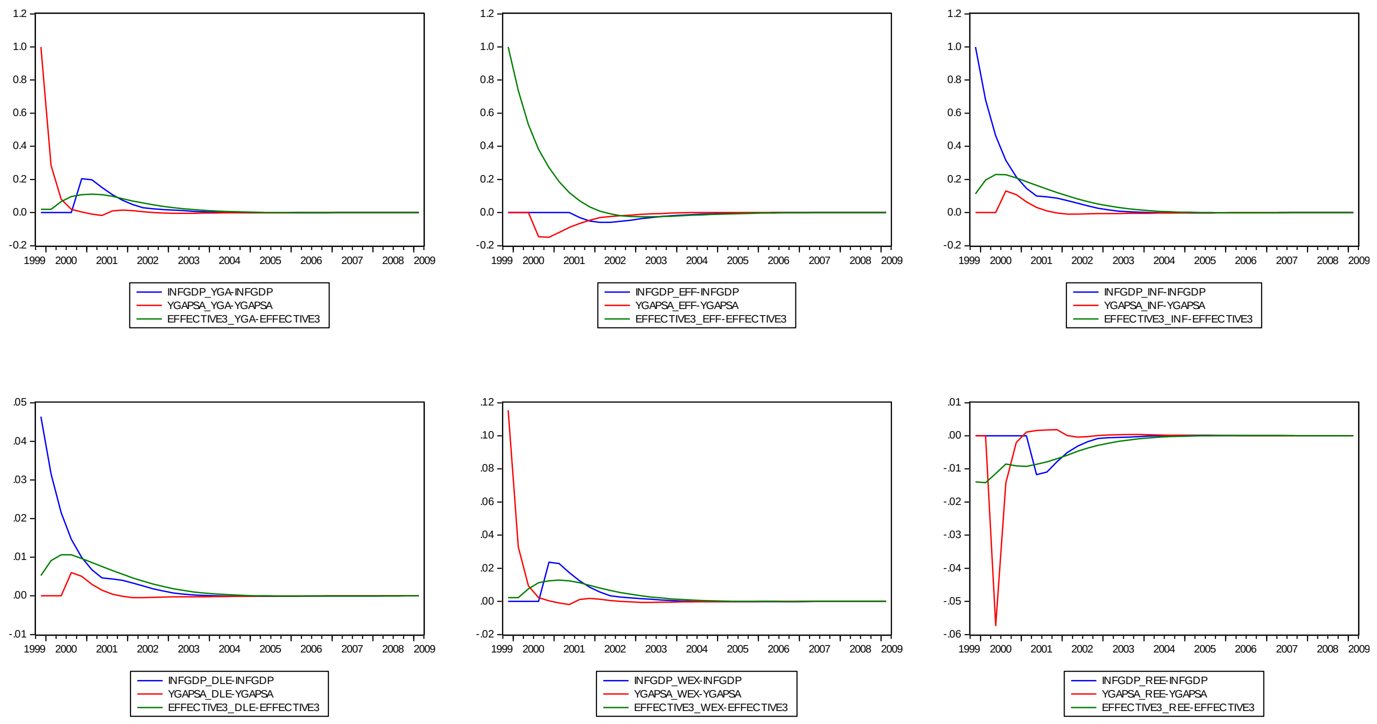
We can also use EViews to assess model dynamics in response to **unanticipated** shocks to the three endogenous variables inflation, the output gap and the monetary policy rate, as well as some exogenous variables (world exports, nominal and real exchange rates). Note that this is **not** done in the original paper by Patra and Kapur (2010) and serves illustrative purposes only. In order to make the shocks **unanticipated**, all we have to do is to change the relevant piece of code in the above programme as shown:

```
%start = "1999q4"
%shk   = %start
' %shk   = "2000q2"
```

The rest of the programme remains exactly the same as before. The only difference between the programme for **unanticipated** shocks and the corresponding programme for anticipated shocks is that the solution period starts in 1999 Q4 (rather than in 2000 Q2, as was the case for the anticipated shocks).

This means that the shock occurs in the same period that we start solving the model, making the shock **unanticipated**. The impulse response functions of the endogenous variables in the model to the six unanticipated shocks are given in Figure 2.

Figure 2: Impulse response functions of the endogenous variables in response to unanticipated unit shocks



We find that an (unanticipated) positive unit shock to `ygapsa` in 1999 Q4 is again relatively short-lived as far the variable itself is concerned: the effect of the shocks lasts for three periods (which is one period more than in the anticipated case). Inflation reacts with a four-period lag, while the policy rate rises contemporaneously rather than with a lead.

An (unanticipated) increase in expected inflation in 1999 Q4 leads to a contemporaneous increase in the policy rate, resulting in an immediate fall in real interest rates. This leads to an expansion of aggregate demand and a positive output gap. As the policy rate moves above inflation, real interest rates turn positive again and the output gap closes. Over time, inflation also comes down in response to higher interest rates, but it takes more than three years for all three variables to return to the baseline.

The monetary policy transmission lags are again evident from the shock to the policy rate. In fact, the impulse responses in Figure 2 are identical to those in Figure 1. Output starts contracting three periods after the (unanticipated) interest-rate shock. The output contraction reaches it trough after another quarter, after which it gradually returns to its baseline. Inflation takes a full seven quarters to respond to the interest rate shock, with a maximum impact after nine quarters. We can see that monetary policy has some impact on real variables such as output, even though there is no long-run impact as postulated by the theoretical arguments of the (long-run) neutrality of money.

The interest-rate response following unanticipated shocks to the output gap and inflation are smaller than in the anticipated case. The maximum impact of `effective3` following an output gap shock is 0.11 when the shock is unanticipated, but 0.25 when it is not. Similarly, the maximum response of `effective3` to a shock in inflation is 0.38 when the shock is anticipated and 0.23 in the case of an unanticipated inflation shock. There is no difference between the maximum output gap and inflation responses between anticipated and unanticipated interest-rate shocks. In other words, and very loosely speaking, the interest rate has to do less work when the endogenous shock is unanticipated.

The authors then go on to consider the same unanticipated shocks to some of the exogenous variables as before (cf. footnote 6). What are the pertinent facts arising from the evaluation of the unanticipated exogenous shocks? A depreciation of the nominal exchange has the same effect as in the anticipated case: inflation is persistently higher (bottom left-hand graph in Figure 2). In turn, higher inflation induces a monetary tightening, as a result of which demand is depressed – with the effect being more volatile than in Figure 1. On the other hand, a real appreciation shock lowers aggregate demand and reduces inflation with a lag of six periods (bottom right-hand graph in Figure 2). Despite the forward-looking monetary policy reaction function, monetary policy eases contemporaneously (two periods before the output gap response and six periods before the inflation response).

Again, the nominal and the real exchange rate shock will work together. In light of the rigidity in the price level, any nominal depreciation will automatically translate into a real depreciation as well. Although not modelled here, the dampening effect of the real appreciation on aggregate demand will be accompanied by external deficits, where the latter may potentially have consequences for external sector sustainability and financial stability.

The interest-rate response following unanticipated shocks to the exogenous variables is again smaller than in the anticipated case. The maximum impact of `effective3` following a shock to world exports is 0.013 when the shock is unanticipated, but 0.029 when it is not. There is no difference between the maximum output gap and inflation responses between anticipated and unanticipated real exchange rate shocks. In other words, and very loosely speaking, the interest rate has to do less work when the exogenous shock is unanticipated.

6 Forecasting with EViews: the exogenous variables

Assume that we are interested in forecasting `ygapsa`, `infgdp` and `effective3` over the period from 2009 Q4 to 2011 Q4. One important issue that we face in forecasting is that many models include exogenous variables that are not explained within the model. In a forecast, some assumptions need to be made about how these variables will behave over the forecast period. In many cases, a projection of a simple growth rate may be enough. In other cases, it may be more appropriate to estimate a time-series model over the past and use this to project the variable forward over the forecast period.

Before we can produce a forecast over our forecast horizon from 2009 Q4 to 2011 Q4, we therefore need to provide 'future' data values for the five exogenous variables that are determined outside the model: `ygapagrsa`, `wexprgapsa`, `reer36gapsa`, `infglobal` and `dlexch4`. There are three ways of doing this:

- we can impose a path for the exogenous variables *a priori*, such as a constant growth rate or a (no change) flat path;
- we can use an empirical approach and estimate a **univariate** time-series process for each of the exogenous variables over the past and project this into the future; or
- we can use an empirical approach and estimate a **multivariate** time-series process for all exogenous variables jointly over the past and project this into the future

For simplicity, we have chosen to pursue the first option, setting the exogenous variables over the forecast period equal to their unconditional mean within the sample period.

Alternatively, we could estimate a vector autoregression (VAR) model for `ygapagrsa`, `wexprgapsa`, `reer36gapsa`, `infglobal` and `dlexch4`, and then use the estimated VAR to produce forecasts for the five exogenous variables. This would be equivalent to following the third option.[6]

7 Conditional forecasting with EViews

Having assumed that the values of the five exogenous variables over the forecast period from 2009 Q4 to 2011 Q4 are equal to their unconditional means over the historical sample, we are now in a position to forecast the three-equation NK model. The forecast period for the latter will be from 2009 Q4 to 2011 Q4. Forecasting – as well as stochastic simulations of the model to produce fancharts around the forecast path –will be illustrated using the EViews **stochsim.prg** programme, which is reproduced below in annotated sections.

```
' Program for stochastic simulations of the India model

' ************************************************************
' Initial filename and parameter settings
' ************************************************************

' Workfile
open imf_india
pagestruct(end=@last+100) *
smpl @all

mode quiet

' Simulation range
%simstart = "2009q4"

' Stochastic parameters
' rndseed 12345
!errorblock = 1
!nsims      = 500
!anti       = 1
```

The first part of the code opens the EViews workfile **imf_india.w1f** and re-sizes the data in a particular way. As discussed in the presentation, our model with forward-looking variables requires the specification of terminal conditions. For that reason, we re-size the sample period by adding 100 extra periods to the final period of the observation sample (`@last+100`). We then change the sample period to include all the data points, specifically the 100 additional quarters we have just created. This sets up the model so that we can specify a long-run solution that will obtain 100 observations, or 25 years, into the future. We once again set the execution mode to `quiet` to make the programme run faster. Finally, we set the starting period for the subsequent stochastic simulations of the model to `%simstart`, which is equal to 2009 Q4.

The next block of code addresses the issue of stochastic simulation. In order to solve a model both deterministically and stochastically, some assumptions must be made about the error terms in the

[6] The second option could involve the use of estimated univariate ARIMA models for forecasting the individual series.

stochastic equations, with different assumptions giving rise to different solutions or simulations.[7] A **deterministic** solution to a model is based on a single drawing of the random shocks. In other words, only one set of values of the error terms is used. A new – different – drawing of the error terms would therefore generate a different solution path. A **stochastic** simulation involves solving the model a number of times (called the number of replications), each time adding a pseudo-random shock to each equation, and drawing a new set of shocks for each replication. Consider taking repeated drawings of the random shocks of the model, assuming a particular distribution with zero mean and fixed variance. The average of these drawings will be a consistent estimate of the expected value of the model variables and higher-order moments such as variances, skew and kurtosis can easily be computed. If desired, quantiles of the distribution can also be computed and these can be displayed graphically in the form of a fanchart. This procedure is known as **stochastic simulation** (Blake (1996)) and increasing the number of drawings or **replications** increases its precision. The advantage of stochastic simulation is that, by drawing from the distribution of the estimated residuals, the bands of the fanchart reflect the uncertainty inherent in the estimated model.

The bootstrap is a simulation approach to estimating the distribution of test statistics or, in the guise of stochastic simulations, of quantifying the uncertainty in model simulations such as forecasts. The underlying idea of the bootstrap is that it is also possible to draw random errors from estimated residuals rather than from estimated distributions.[8] An obvious advantage of this approach is that no – potentially erroneous – assumption about the distribution of the error terms has to be made.

A lot of what we will do below involves stochastic simulations in general and the bootstrap in particular, which necessitates a range of additional settings. To begin with, the **random seed** (`rndseed`), here set to 12345, initialises the random number generator used for generating pseudo-random shocks. Two stochastic simulations, over the same period and using the same random number seed, will generate the same results. This is useful when a simulation needs to be repeated at a later date. Changing the seed will result in a new set of random drawings, such that the original simulation using a different random seed cannot be recovered precisely. It is therefore useful to make a note of the random number seed used in important simulations.

But the performance of the bootstrap procedure can be far from satisfactory for time-series data, including residuals used for resampling, with serial correlation and heteroskedasticity of unknown form. In the face of the latter two problems, the block bootstrap is the most general methods to improve the accuracy of resampling for time-series data. The idea is that the original time-series structure within a block can be preserved by dividing the data into several blocks. The size of these error blocks is set by the `!errorblock` command, where higher orders of serial correlation require larger block sizes.

The **number of replications** is set with the `!nsims` command. Given that computing power is (relatively) cheap, 500 replications appear very modest (note that greater precision could be achieved by increasing the number of replications at the cost of greater execution time).

The final setting for `!anti = 1` instructs the programme to use the option of **antithetic stochastic shocks**.[9] This forces the distribution of generated shocks to be (exactly) symmetric.[10] Antithetic shocks are pairs of shocks with the same magnitude and opposite sign. In other words, the programme will generate 250 shocks of a particular size and sign, but then use another 250 shocks of the same size, but different sign, to yield the overall number of 500 replications. Using antithetic shocks will

[7] There are various meanings to the word solution and it will be useful at this point to provide some definitions. Throughout this exercise, solution and simulation are used interchangeably as they are assumed to mean the same thing.

[8] The idea of drawing errors to analyse the properties of econometric models has a long pedigree in macroeconomics, going back to the seminal paper by Adelman and Adelman (1959).

[9] Any value of `!anti` greater than zero will generate antithetic shocks. Setting `!anti` to 0 means that no antithetic shocks will be used.

[10] Note that in a non-linear model, this does not guarantee that the distribution of the model variables will be symmetric as well.

increase the precision of a simulation. In a linear model, a simulation with antithetic shocks will have a mean exactly equal to the deterministic solution of the model.

```
' ****************************************************************
' Specify model
' ****************************************************************

' Equations and coefficients
%mod = "India"

' Subroutines
include subs/mce_solve_library
include subs/master_library
```

Just as before, the programme assigns the value ‘India’ to the string variable, `%mod`, such that the place-holder `%mod` will be replaced by India every time EViews encounters it. In the next step, EViews calls the **mce_solve_library.prg** as well as the **master_library.prg** subroutines that can be found in the `subs` folder. This reads in all the available pre-defined solution routines in those particular two libraries.

```
' Retrieve estimated coefficients
*****************************************

' New Keynesian aggregate demand or dynamic IS curve coefficient
estimates
preferred_model_1.updatecoef
coef alpha = c

' Aggregate supply or new Keynesian Phillips curve coefficient
estimates
preferred_model_2.updatecoef
coef beta = c

' Monetary policy reaction function or Taylor-rule coefficient
estimates
preferred_model_3.updatecoef
coef gamma = c
```

As we saw before, this part of the programme retrieves the estimated coefficients from three equation objects in the **imf_india.wf1** workfile. In particular, it retrieves coefficient information from an EViews equation object called `preferred_model_1`, which is the estimated aggregate demand or dynamic IS curve. The estimated coefficients are assigned to a coefficient vector `alpha` for later use. Similarly, it retrieves coefficient information from an EViews equation object called `preferred_model_2`, which is the estimated aggregate supply or new Keynesian Phillips curve. The estimated coefficients are assigned to a coefficient vector `beta` for later use. Finally, it retrieves coefficient information from an EViews equation object called `preferred_model_3`, which is the estimated monetary policy reaction function or Taylor rule. The estimated coefficients are assigned to a coefficient vector `gamma` for later use.

```
' Create model
************************************************************
model {%mod}

' New Keynesian aggregate demand or dynamic IS equation
{%mod}.append ygapsa = alpha(1) + alpha(2)*rpr(-3) + alpha(3)*ygapsa(-
1) + alpha(4)*ygapagrsa + alpha(5)*wexprgapsa + alpha(6)*reer36gapsa(-
2)

' Aggregate supply or new Keynesian Phillips curve
{%mod}.append infgdp = beta(1) + beta(2)*ygapsa(-4) + beta(3)*infgdp(-
1) + beta(4)*infglobal + beta(5)*dum1998q3q4 + beta(6)*dlexcha

' Monetary policy reaction function or Taylor-rule
{%mod}.append effective3 = gamma(1) + gamma(2)*infgdpdev(+2) +
gamma(3)*ygapsa(+2) + gamma(4)*effective3(-1) + gamma(5)*dlexch4(-1)

' Identity for infgdpdev
{%mod}.append infgdpdev = infgdp - 5

' Identity for rpr
{%mod}.append rpr = effective3 - infgdp

' Identity for dlexcha
{%mod}.append dlexcha = (dlexch4 + dlexch4(-1) + dlexch4(-2) +
dlexch4(-3))/4
```

This part of the programme creates the EViews **model** object that we want to simulate and forecast. A model in EViews is a set of one or more equations that jointly describe the relationship between a set of variables. Overall, EViews models allow us to combine equations inside a single object, which may then be used to create a deterministic or stochastic joint forecast or simulation for all the variables in the model. The first line creates a new EViews **model** object and, *via* the `{%mod}` command, assigns the name **India** to it. It then creates the six equations of the India model object. The new Keynesian aggregate demand or dynamic IS curve will provide an illustration of how this is done. This is the first equation of the model, which is simply written in the customary EViews way. The one difference is that we recover the coefficient estimates from the `alpha` vector we created in the previous step. All in all, the vector `alpha` contains six elements: the first is the estimated coefficient on the constant (`alpha(1)`), the second is the estimated coefficient on `rpr(-3)` (`alpha(2)`), the third is the estimated coefficient on `ygapsa(-1)` (`alpha(3)`), etc. Once the coefficients have been substituted, the model equation for `ygapsa` will be:

```
ygapsa =   0.383188392432 - 0.146848613221*rpr(-3)
         + 0.284463647328*ygapsa(-1) + 0.207318257846*ygapagrsa
         + 0.115359626484*wexprgapsa - 0.057314952378*reer36gapsa(-2)
```

The other equations of the model are created along the same lines. The three identities at the end do not contain any estimated coefficients, so the equations can be entered as they appear above and then appended to the existing **India** model object.

```
' Create vector that contains all the variables
svector mod_vars = @wsplit({%mod}.@varlist)
for !i = 1 to @rows(mod_vars)
   %temp = mod_vars(!i)
   {%temp} = @nan({%temp},@mean({%temp}))
next
```

These six lines create a string vector that contains all of the names of the variables in the model.[11] In other words, we vectorise the names of all the endogenous as well as exogenous variables and put them into the string vector `mod_vars`. The code then goes through the list and replaces any non-existent observations, which appear as `NA` in EViews, by the unconditional mean of the respective variable over the sample period (`@mean()`). This needs to be done as EViews will not solve the model without data. Remember that we have re-sized the worksheet to include another 100 'observations'. While we have mechanically created another 100 observations, there is no actual data attached to any of these observations and they appear as `NA`'s in the workfile.

Remember from Section 6 that we decided to follow the first method of forecasting exogenous variables, which is to set them equal to their unconditional in-sample means over the forecast period. This is, partly, what is happening in the above section of code. We loop over all the variables, be they endogenous or exogenous, and replace non-existent observations by their sample means. For the exogenous variables, this means replacing the future values by their respective in-sample means.

```
' Set _aerr variables to zero
{%mod}.makegroup(a,n) endog @endog
call groupnew("endog","_aerr")
call group2zero("endog_aerr")
```

These four lines collects the endogenous variables (note the use of `@endog` in the command line) in a group called `endog`, creates a new group that is the same as `endog`, names this new group as `endog_aerr` and sets all the variables with the suffix `_aerr` in this group to zero. Note that the last two lines call sub-routines `groupnew` and `group2zero` that are not available within EViews, but are purpose-written and contained in the **master_library.prg** file.

```
%mstr  = "create,mod="+%mod+",adds,track"
%astr  = "meth=newton,jinit=linear"
%sstr  = "type=single,o=3,scen,suf=_0"
%sstrf = "type=single,o=3,scen,suf=_f"
%sstrs = "type=single,o=3,scen,suf=_"
```

This part of the programme sets the simulation control. More specifically, it specifies the options that will serve as the input for the `mce_run` routines in EViews. These options are further explained in the document entitled **mce_solve_users_guide.pdf** that can also be found in the current course **Data** folder.

Let us start with `%mstr`, which needs to contain the keyword `create`, the keyword assignment `mod = ⟨name1⟩` and that a model with the name `⟨name1⟩` exists in the workfile. In the example at

[11] Note the use of the `@varlist` command, which produces a list of all of the variables on the right-hand side of the equations in the model. If string list `SS01` contains 'A B C D E F', then `@wsplit(ss01)` returns an untitled svector, placing an element of `SS01` in each row. For example, row one of the svector contains 'A', row two contains 'B', etc.

hand, ⟨name1⟩ will be **India**, and a model with this name does indeed exist in the workfile – remember that we have created it ourselves in the early part of the programme. Inclusion of the adds keyword causes **add factors** to be assigned to all equations, while the inclusion of the track keyword causes the values of the **add factors** to be initialised so that the equations make no errors when evaluated using actual data. In other words, the values of the **add factors** are chosen so that these equations track the baseline data perfectly over the current workfile sample.

The second subroutine argument of mce_run, associated with %astr, is used to choose a solution algorithm and set related options. In this group, the meth (method) keyword specifies the solution algorithm: the default setting is newton for the E-Newton solution algorithm. The jinit keyword controls the construction of the initial Jacobian or approximate Jacobian matrix. Using jinit = linear means that the Jacobian matrix is initialised (and updated) using the linear shortcut that is applicable to many linear model-consistent expectations models.

We should note that %mstr as well as %astr need only be called once. This will not be the case for the third argument. The %sstr arguments control the simulation type, execution and related parameters. In particular, the mce_run subroutine can initiate three types of model-consistent expectations simulation experiments. As we have chosen the default setting of type = single, I will discuss this approach very briefly, referring the interested reader to the aforementioned document for a more extensive discussion of the other two options (opt and opttc). Setting type = single runs a model-consistent expectations simulation whose inputs are the initial and terminal values of the model's variables and the projected paths of any shocks and exogenous variables the model may contain. The other two options, o = 3 and suf = _, set the amount of output per model-consistent expectation simulation and an alias for a new scenario respectively. The most amount of output is displayed by setting o = 1 and the least amount of output is produced with o = 3.

Note that there are two more %sstr arguments this time around. We have set up these extra arguments to perform different simulation experiments or **scenarios**. The first (%sstr) results in variables that have the suffix _0, which denotes the baseline simulation or scenario, the second (%sstrf) will result in variables that have the suffix _f for forecast and the third (%sstrs) will result in variables that have the suffix _ and an – as yet – undefined single letter or number. Ultimately, these variables will be used to store the stochastic simulations.

```
' Initial run
smpl {%simstart} @last-2
call mce_run(%mstr,%astr,%sstr)
```

We then do an initial run, generated by calling mce_run with the settings as defined above, to generate a baseline forecast for later analysis. Because we are calling mce_run for the first time, we have to include %mstr, %astr and %sstr. This generates the base simulation without any shocks. As we have selected the track option for %mstr, the algorithm will work out the residuals that make the model fit the data perfectly as part of the simulation or scenario and generate variables with the suffix _0.

```
' Set tracking shocks to zero over forecast
svector endo_vars = @wsplit({%mod}.@endoglist)
for !i = 1 to @rows(endo_vars)
   %temp = endo_vars(!i)
   {%temp}_a = 0
next
```

This part of the programme sets the tracking errors, which are denoted by the suffix _a, to zero over the forecast period. This is, in fact, a little trick. We take the endogenous variables (@endoglist) and name the residuals after them. These few lines make the earlier residuals equal to 0, so that – at a minimum – we have made sure that these residual series now exist. In terms of manual adjustments, interventions or the imposition of any judgement, this would be the appropriate place in the programme to do so. In fact, the manual adjustment of the residual series would mean going into the residual series suffixed by _a and putting in values for them by hand.

```
' Judgement/interventions in the model should be done by specifying
appropriate _a values
```

At this point in the programme, we do a 'proper' forecast and store the resulting output in series with the suffix _f. Note that the call to mce_run leaves the first two entries blank and only references the settings coming from %sstrf.[12] This is because the solution algorithm knows what the options in %mstr and %astr are. Specifying %mstr and %astr again would confuse the solution algorithm, so there is no need to repeat simulation controls that remain unchanged from one run to another. We can do the forecast involving exogenous variables, as we have set their future values equal to their respective unconditional means over the entire forecast period. In other words, a data set for the exogenous variables exists over the forecast period.

```
' Now do a proper forecast and store in *_f
call mce_run("","",%sstrf)

' ****************************************************************
' Stochastic shocks
' ****************************************************************

' Group of equations to receive shocks
group shock ygapsa infgdp effective3

' Demean historical residuals stored in "resid_*" and store them in a
matrix

smpl @all

%error_names = ""
for !i = 1 to shock.@count
    %temp = "resid_"+shock.@seriesname(!i)
    scalar mm = @mean({%temp})
    {%temp} = {%temp} - mm
    %error_names = %error_names + " " + %temp
next

group errors {%error_names}
stom(errors,errormat)
```

[12] Note that mce_run cannot differentiate between an in-sample and an out-of-sample simulation. We, on the other hand, can, as we have set the respective sample period that differentiates between the two. To differentiate between the in- and out-of-sample simulation, we use the %sstr options that track what we are doing.

There is some housekeeping that needs to be done before we can move on to stochastic simulation. As such, we define an EViews `group` that contains the variables that will be shocked. Just as before, this group contains `ygapsa`, `infgdp` and `effective3`, i.e., the three endogenous variables. We then demean all the historical residuals that are stored in matrices called `resid_*` (where * denotes the respective variable name) and store them in a new `group` called `errors`, which we then turn from a `group` into a matrix using the series-to-matrix or `stom` command. This new matrix of demeaned errors is called `errormat`. We do this as practical experience with the bootstrap has shown that it can fail for an equation with no constant term if the residuals are not centred at zero, i.e., demeaned.

```
' Define some tracked variables for plotting, set up groups for
results
group track ygapsa infgdp effective3
for !i = 1 to track.@count
    %temp = track.@seriesname(!i)
    group {%temp}_group
next
```

This part does some more housekeeping, in particular defining tracked variables for later plotting (`ygapsa`, `infgdp`, `effective3`) and setting up three groups for storing the results.

```
' *****************************************************************
' Stochastic simulation loop (sims are run one at a time)
' *****************************************************************

smpl {%simstart} @last-2
scalar nsamp = @obssmpl-@rows(errormat)
call groupnew("shock","_aerr")

if !anti > 0 then
  !nsims = @floor(!nsims/2)
endif

for !i = 1 to !nsims

    statusline running stochastic sim number !i

  ' Draw nqtrs random rows from the matrix of historical errors,
    matrix stocherrors = @resample(errormat,nsamp,!errorblock)
    mtos(stocherrors,shock_aerr)

    for !h = 1 to shock.@count
        %temp = shock.@seriesname(!h)
        {%temp}_a = {%temp}_aerr
    next
    call mce_run("","",%sstrs+@str(!i))

    if !anti > 0 then
        for !h = 1 to shock.@count
```

```
            %temp = shock.@seriesname(!h)
            {%temp}_a = - {%temp}_a
        next
        call mce_run("","",%sstrs+@str(!i+!nsims))
    endif

    for !h = 1 to track.@count
        %temp = track.@seriesname(!h)
        {%temp}_group.add {%temp}_{!i}
        if !anti > 0 then
            !j = !i+!nsims
            {%temp}_group.add {%temp}_{!j}
        endif
    next

next
```

It is only now that we are ready to start the stochastic simulations.

```
delete err_* _$_*
```

This command keeps the workfile tidy by deleting any temporary objects that have been created up to this point, but that are no longer required. As we can see, the deleted objects will have names that begin with either `err_` or `_$_`.

```
' Fancharts

smpl @all
group forecasts
for !h = 1 to track.@count
    %t = track.@seriesname(!h)
    forecasts.add {%t}_f
    series {%t}_sd = @rstdev({%t}_group)
    group {%t}_plot {%t}_f+1.96*{%t}_sd {%t}_f-1.96*{%t}_sd
{%t}_f+0.842*{%t}_sd {%t}_f-0.842*{%t}_sd {%t}_f+0.385*{%t}_sd {%t}_f-
0.385*{%t}_sd {%t}_f
next
```

This part of the programme does the calculations of the fancharts. In particular, we want to display three fans, whose boundaries are given by the mean forecast and plus and minus 1.96 standard deviations (equal to the area between the 95th and the fifth percentile, representing the outermost band), the mean forecast and plus and minus 0.842 standard deviations (equal to the 80th and the 20th percentile, representing the middle band) and the mean forecast plus and minus 0.385 standard deviations (equal to the 65th and 35th percentile, representing the innermost band).

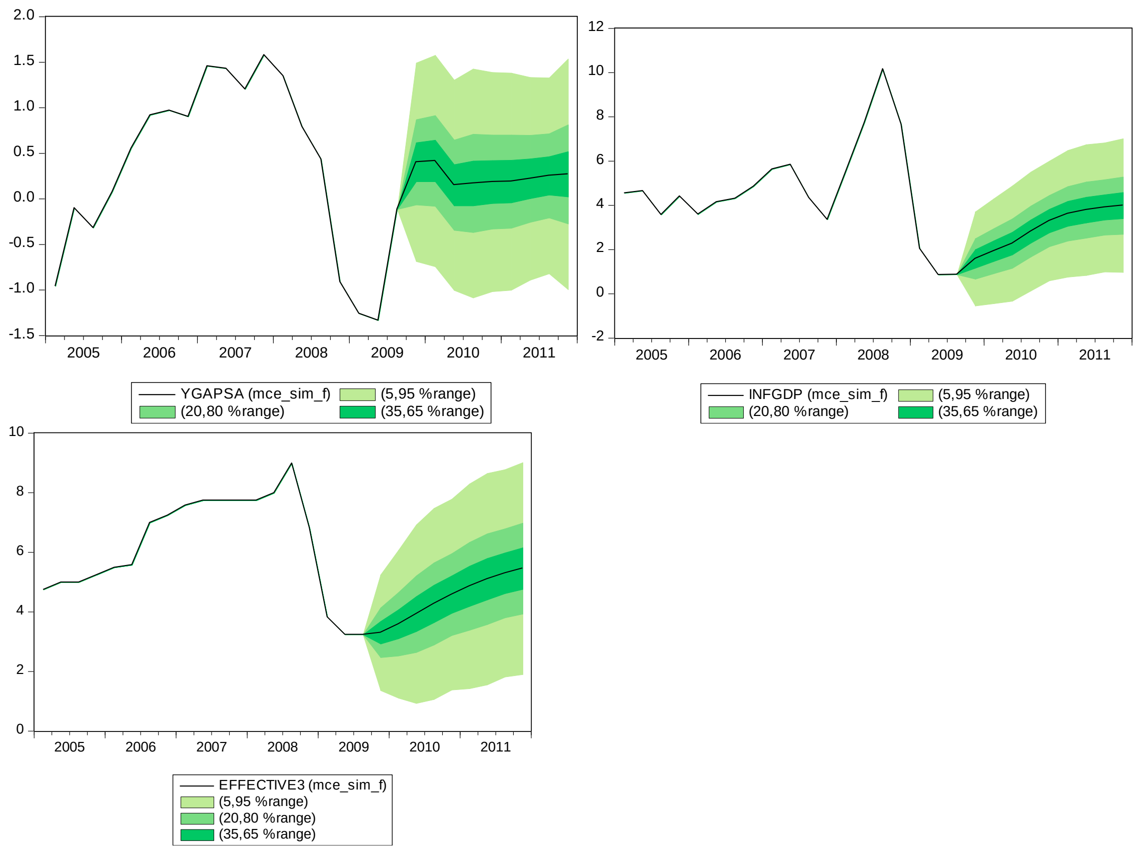
```
' Plots

smpl  2005q1 2011q4
forecasts.line
```

```
for !h = 1 to track.@count
    %t = track.@seriesname(!h)
    graph {%t}_graph.band {%t}_plot
    {%t}_graph.display
    {%t}_graph.setelem(1) fillcolor(@rgb(190, 235, 150))
    {%t}_graph.setelem(1) legend("5")
    {%t}_graph.setelem(2) legend("95 %range")
    {%t}_graph.setelem(2) fillcolor(@rgb(120, 220, 130))
    {%t}_graph.setelem(3) legend("20")
    {%t}_graph.setelem(4) legend("80 %range")
    {%t}_graph.setelem(3) fillcolor(@rgb(  0, 200, 100))
    {%t}_graph.setelem(5) legend("35")
    {%t}_graph.setelem(6) legend("65 %range")
    {%t}_graph.setelem(1) linecolor(black)
next
```

The final part of the programme sets up the various series for plotting. It also generates the particular type of graph that lends itself to generating a fanchart and specifies the colours of the bands. In particular, we have to plot our data using **area bands**. We plot the fanchart from 2009 Q4 to 2011 Q4 while also showing some back data to allow for historical comparisons.

The three forecasts for the endogenous variables – with a fanchart around their mean forecasts – are shown in Figure 3.

Figure 3: Fanchart forecasts for `ygapsa`, `infgdp` and `effective3`



Appendix

`bses`	Bombay stock exchange index
`crr`	Cash reserve ratio
`dlbsesa`	Year-on-year variation in log `bses`
`dlexch4`	Annualised quarter-on-quarter variation in the log nominal exchange rate
`dlexcha`	Year-on-year variation in the log nominal exchange rate
`dlm3aadj`, `dlnfcaadj`	Year-on-year variation in the log of broad money and log bank credit
`effective3`	Policy interest rate
`effective3gdp`	Real policy interest rate = policy interest rate minus year-on-year GDP deflator inflation (`effective3` – `infgdp`)
`exch`	Nominal exchange rate of Indian rupee against the US dollar
`fedtarget`	US federal funds target rate
`gdpr`	Real GDP (crore of rupees)[13]
`infglobal`	Global non-fuel commodity inflation (IMF index, measured year-on-year)
`infpc`	Primary article inflation (sub-group within wholesale price inflation, measured year-on-year)
`infwpi`, `infgdp`, `infcpi`	India's wholesale price, GDP deflator and consumer price inflation respectively (measured year-on-year)
`infwpidev`, `infgdpdev`, `infcpidev`	Deviation of actual inflation from the RBI's indicative inflation projection of 5 per cent = `infwpi - 5`, `infgdp - 5`, `infcpi - 5`
`reer36`	36-currency real effective exchange rate (REER) index
`reer36gapsa`	36-currency REER gap, s.a.
`wexpr`	World real exports (from IMF, nominal exports/unit value index)
`wexprgapsa`	World real exports gap, s.a.
`yagri`	Agricultural GDP (crore of rupees)
`ygapagrsa`	Agricultural GDP gap, s.a.
`ygapsa`	Output gap, seasonally adjusted (s.a.)

[13] A crore is a unit in the Indian numbering system equal to ten million.

References and further reading

Adelman, I and Adelman, F L (1959), 'The dynamic properties of the Klein-Goldberger model', *Econometrica*, Vol. 27, No. 4, pages 596-625.

Blake, A P (1996), 'Forecast error bounds by stochastic simulation', *National Institute of Economic and Social Research Review*, Vol. 156, No. 1, pages 72-79.

Brayton, F (2011), 'Two practical algorithms for solving rational expectations models', *Board of Governors of the Federal Reserve System Finance and Economics Discussion Series* 2011-44. http://www.federalreserve.gov/pubs/feds/2011/201144/201144pap.pdf.

Patra, M D and M Kapur (2010), 'A monetary policy model without money for India', *IMF Working Paper WP/10/183*. http://www.imf.org/external/pubs/ft/wp/2010/wp10183.pdf.