# Running MCE Simulations in EViews Using the MCE_SOLVE_LIBRARY Subroutines

January 26, 2014

## 1. Overview

The EViews subroutines collected in **mce_solve_library** use the E-Newton and E-QNewton algorithms to impose model-consistent expectations (MCE) in simulations of macro models.[1] The algorithms iterate to find a model's MCE 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 E-Newton algorithm is a straightforward implementation of Newton's method that relies on the numerical computation (via perturbation simulations) and inversion of the Jacobian matrix of derivatives of the expectations errors with respect to the exogenous expectations estimates or components. The E-QNewton algorithm is a limited-memory implementation of Broyden's quasi-Newton method whose iterations are based on a derivative-free approximate Jacobian. The latter is much cheaper to construct and use than the E-Newton full Jacobian, especially when the size of the Jacobian is large. E-Newton solutions, however, usually converge in fewer iterations.

For single simulations of linear RE models, E-Newton is likely to be faster for models of small-to-medium size and E-QNewton is likely to be faster for larger models. For nonlinear models, E-Newton tends to be penalized relative to E-QNewton to the extent that the nonlinearity eliminates patterns that can be exploited in computing the the Newton Jacobian. The E-Newton algorithm has a substantial advantage over E-QNewton on experiments that involve a large number of RE solutions, as long as the same Jacobian can be used for each E-Newton solution.

Use of the two algorithms typically involves a sequence in which: (i) the MCE model is set up to conform to the general requirements of the algorithms; (ii) a particular algorithm is selected and its parameters are set; and (iii) a simulation experiment is specified and executed. The available types of experiments are single simulations, unconstrained optimization, and

[1] The solution algorithms are described in detail in Flint Brayton, "Two Practical Algorithms for Solving Rational Expectations Models," FEDS Working Paper 2011-44.

constrained optimization. Two approaches to optimization are permitted. Using the terminology associated with the literature on optimal government policy, one approach computes optimal trajectories of one or more policy (or other types of) instruments under committment and the other computes time-consistent trajectories without committment. Exact committment solutions can be calculated for linear and (in principle) for nonlinear models. Exact time-consistent solutions without committment, which are more difficult to compute, are limited to linear models, unless the user writes code to embed the call to the optimizer in a loop which involves repeated updating of the linearized relationship between the instruments and targets.

The **mce_run** subroutine provides the primary interface for carrying out this sequence of steps. For many models and experiments, a single call to **mce_run** may suffice. For other models, the model setup step may have to be carried out manually or, as is the case for FRB/US, via another subroutine prior to the call to **mce_run**.

Section 2 presents the general syntax of **mce_run**. Sections 3 through 5 go over the model setup, algorithm selection, and simulation execution steps in detail. Partial examples are presented in each of these sections, and complete examples are discussed in section 6. Section 7 contains some material specific to FRB/US.

## 2. General Syntax of mce_run

The call to **mce_run** takes three arguments that 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)
```

Each argument consists of a comma-separated list of keywords and keyword assignments, as in,

```
"kword1,kword2=2,kword3=xyz,kword4=%zyx"
```

where $kword1, kword2, kword3$, and $kword4$ stand in for valid keywords. The action of some keywords, such as $kword1$, depends on their absence or presence. The action of other keywords depends on the number ($kword2$), sequence of characters ($kword3$), or string variable ($kword4$) to which they are assigned. Most these keywords have default settings. Keywords may appear in any order. Each argument may be a string, string variable or string object.

```
%algstr = "kword11,kword12=3"
string simstr = "kword21=yes,kword22"
call mce_run("kword1,kword2=2",%algstring,simstr)
```

An EViews program can contain multiple calls to **mce_run**. These calls may involve different models or they may involve repeated simulations of the same model. When a model is reused, the same solution algorithm may also be retained. Making the first subroutine argument a null string causes the previous model to be used and any model processing steps to be skipped. Making the second subroutine argument a null string causes the previous algorithm to be used and skips the initialization of the Jacobian or approximate Jacobian. The ability to bypass initializing the Jacobian can significantly reduce the time time required for repeated E-Newton solutions of models that are linear or nearly linear. When the same algorithm is repeated, the length of the simulation period must remain unchanged.

## 3. MCE Model Setup

### General discussion

In order to use the E-Newton and E-QNewton algorithms, two separate operational models must be created. In many cases, the first of these operational models is a simple transformation of the original model in which each appearance of a future-dated endogenous variable is replaced with a new exogenous variable, and the second operational model consists of identities that define the MCE errors. A simple example illustrates this organization. Let Model O, Model B, and Model F respectively designate (a part of) the original model and the pair of operational models.

```
Model O
   y = c1*y(-1) + c2*y(1) + ...
Model B
   y = c1*y(-1) + c2*x + ...
Model F
   ey = x - y(1)
```

Model O contains the endogenous variable $y$ and coefficients $c1$ and $c2$. The first lead of $y$ in Model O is replaced with the exogenous variable $x$ in Model B, and Model F contains an identity that defines the MCE error $ey$ as the difference between the values of $x$ and the first lead of $y$. The latter is exogenous in Model F. An iteration of either solution algorithm involves a solution of Model B, a solution of Model F, and an update of the path of $x$ based on the path of $ey$ and information on the relationship between their paths. Iterations continue until a path of $x$ is found that sets $ey$ to zero.

The creation of Model B and Model F in this simple example is easily automated and can be carried out using **mce_run**. Not as easy to automate is a more complex case in which the future-dated variables in Model O are replaced not with exogenous variables but with contemporaneous values of new endogenous variables. This is the approach used in FRB/US, in which the new endogenous variables represent an alternative, backward-looking characterization of expectations. For example:

```
Model O
   y = c1*y(-1) + c2*y(1) + ...
Model B
```

```
   y = c1*y(-1) + c2*z + ...
   z = c3 + c4*y(-1)
Model F
   ey = z - y(1)
```

In this case, both solution algorithms iterate over the path of the expectations constant $c3$ until the MCE error is zero, and the pair of operational models must be created manually or, as is done for FRBUS, with a special, model-specific subroutine.

The model or models supplied by the user must be valid EViews models. In particular, each endogenous variable must appear as the first variable on the left hand side of one and only one equation. In addition, when the user supplies (the equivalents of) Models B and F, Model B cannot contain any leads of endogenous variables, Model F must contain identities that define all MCE errors, and Model F may include leads of its endogenous variables but it cannot include any lags of these variables. With these restrictions, Model B can be easilyt solved forward in time and Model F can be easily solved backward in time.

Convergence of the algorithms to the MCE solution occurs when the maximum absolute expectations error is less than some (small) value. This criterion may lead to undesirable convergence characteristics of models with substantial heterogeneity in the magnitudes of their MCE variables. One way to resolve problems of this type is to rewrite the model so that the MCE variables are scaled more homogeneously. Another is to multiply the right hand sides of the relevant MCE error equations by a scale factor so that the errors themselves are better scaled.

**The m_opts argument**

The first subroutine argument of **mce_run**, the *m_opts* string, can have one of three general forms that correspond to (1) automatic parsing of the MCE model to form the pair of operational models; (2) declaration of the names of operational models that have been created by other means; or (3) use of operational models that have been created automatically or declared in a previous call to the subroutine.

1. *m_opts* contains the keyword *create* and the keyword assignment $mod = \langle name1 \rangle$, and a model named $\langle name1 \rangle$ exists in the workfile. In this case, the model procesing part of the subroutine parses model $\langle name1 \rangle$

to form a pair of operational models with the names $\langle name \rangle\_b$ and $\langle name \rangle\_f$.

2. *m_opts* contains the keyword assignments $mod\_b = \langle name1 \rangle$ and $mod\_f = \langle name2 \rangle$ as well as assignments of either $mce\_errs = \langle \%name3 \rangle$ and $mce\_instrus = \langle \%name4 \rangle$ or $mce\_vars = \langle \%name5 \rangle$. In either variant, the model processing part of the subroutine assumes that the operational models $\langle name1 \rangle$ and $\langle name2 \rangle$ have already been created and exist in the workfile. In the first variant, $\langle \%name3 \rangle$ is the name of a string list of the MCE error variables and $\langle \%name4 \rangle$ is the name of a string list of the exogenous expectations estimates or components. In the second variant, $\langle \%name5 \rangle$ is the name of a string list that contains the base variable names from which the names of the MCE error variables can be formed adding an "e" prefix and the names of the MCE exogenous variables or components can be formed by adding an "_a" suffix.

3. *m_opts* is a null string (ie, ""). This form indicates that the operational models declared or created in a previous call to the *mce_run* subroutine (within the same program) are to be used again.

The optional keywords *adds*, *track*, *tstart*, and *tend* are available in forms 1 and 2 of the *m_opts* string. Inclusion of the *adds* keyword causes add factors to be assigned to all equations in both operational models. Inclusion of the *track* keyword causes the values of the add factors to be initialized so that the equations make no errors when evaluated using actual data. The add factors are initialized over the current workfile page sample, unless the *tstart* and *tend* keywords are added to specify alternative starting and ending dates. These four keywords are particularly useful when add factors are needed in operational models that are created by the action of **mce_run**, as the user does not have direct access to these models while the subroutine executes. When the operational models are created prior to the call to **mce_run** and add factors are desired, they may be assigned and initialized either prior to the subroutine call or as part of the execution of **mce_run** using the appropriate keywords.

**Examples of m_opts**

The first example is based on a three-equation model which contains the endogenous variables $p, r$, and $y$, MC expectations of the first lead of $p$ and $y$, the exogenous variable *shk*, and coefficient vectors *cp*, *cy*, and *cr*.

Table 1: All *m_opts* Keywords

| keyword | setting | description |
|---|---|---|
| *keywords specific to form 1* | | ——— |
| create | | see form 1 discussion |
| mod= | name or string var | see form 1 discussion |
| *keywords specific to form 2* | | ——— |
| mod_b= | name | operational model without endog leads |
| mod_f= | name | operational model with MCE error eqs |
| mce_errs= | string var | names of MCE error variables |
| mce_instrus= | string var | names of MCE instrument variables |
| mce_vars= | string var | base names for MCE errors and instruments |
| *keywords for forms 1 and 2* | | ——— |
| adds | | assign add factors |
| track | | initialize add factors at tracking values; default sample is workfile page sample |
| tstart= | date | tracking adds initial period |
| tend= | date | tracking adds final period |

```
model s
s.append p = cp(1)*p(-1) + (.98-cp(1))*p(1) + cp(2)*y
s.append y = cy(1)*y(-1) + (.98-cy(1))*y(1) + cy(2)*(r - p(1))
s.append r = cr(1)*r(-1) + (1-cr(1))*(cr(2)*p + cr(3)*y) + shk
```

When the following string is used as the *m_opts* argument,

```
%mstr = "create,mod=s,adds,track"
```

the two operational models are automatically created, add factor variables are assigned to the equations of each operational model, and the values of the add factors are chosen so that these models track the baseline data over the current workfile page sample. This syntax corresponds to form 1 of *m_opts*.

As example of the form 2 of *m_opts*, assume that the user has already created a pair of operational models associated with model *s*, named them

*s_b* and *s_f*, and chosen to use new endogenous variables to replace the future values of $p$ and $y$.

```
model s_b
s_b.append p = cp(1)*p(-1) + (.98-cp(1))*zp + cp(2)*y
s_b.append y = cy(1)*y(-1) + (.98-cy(1))*zy + cy(2)*(r - zp)
s_b.append r = cr(1)*r(-1) + (1-cr(1))*(cr(2)*p + cr(3)*y) + shk
s_b.append zp = p(-1)
s_b.append zy = y(-1)

model s_f
s_f.append ezp = zp - p(1)
s_f.append ezy = zy - y(1)
```

If tracking add factors are needed in the operational models, but have yet to be be assigned, the required *m_opts* argument can be created either with the commands

```
%instrus = "zp_a zy_a"
%errs = "ezp ezy"
%mstr = "mod_b=s_b,mod_f=s_f,mce_instrus=%instrus,
         mce_errs=%errs,adds,track,tend=2050q4"
```

or with the commands

```
%vars = "zp zy"
%mstr = "mod_b=s_b,mod_f=s_f,mce_vars=%vars,adds,track,tend=2050q4"
```

The *tend* keyword causes the add factors to be initialized over the period from the start of the current workfile page sample to 2050q4. Note that the second set of example commands works correctly because the names of the MCE error variables can be formed by adding an "e" prefix to each of the words in the %vars string and the names of the MCE exogenous instruments can be formed by adding an "_a" suffix to each of the words in that string.

When multiple calls to **mce_run** are being made in the same program to run a set of simulations of the same MCE model, the model processing step can be skipped after the first call. To designate this, simply use a null string for the *m_opts* argument (ie, %mstr =””) in the second and any subsequent calls.

## 4. Algorithm Selection

### General discussion[2]

In the E-Newton and E-QNewton algorithms, the adjustment of the vector of expectations estimates (or their exogenous components), $\tilde{x}$, at iteration $i$ is the product of a step length, $\lambda_i$, and a direction, $\tilde{d}_i$,

$$\Delta\tilde{x}_i = \lambda_i \tilde{d}_i, \tag{1}$$

where the direction in turn depends on the product of an updating matrix $U$ and the vector of expectations errors, $\tilde{e}$ in the previous iteration,

$$d_i = -U_{i-1}\tilde{e}_{i-1}. \tag{2}$$

In E-Newton, the updating matrix is the inverse of the Jacobian matrix of first derivatives of the expectations errors with respect to the expectations estimates ($J = (\partial\tilde{e}/\partial\tilde{x})$). Derivatives are computed numerically using perturbation simulations. In E-QNewton, the updating matrix is the inverse of an approximate Jacobian ($B$) whose elements are based only on the implicit derivative information observed each iteration in the movements of $\tilde{e}$ and $\tilde{x}$. E-QNewton employs a *limited memory* version of Broyden's method in which the direction is computed from a sequence of vector operations that do not require the direct computation of $B$ and its inverse. This design tends to make E-QNewton more efficient than E-Newton for single simulations of an MCE model, especially when the size of the Jacobian (the product of the number of MCE variables, $m$, and number of simulation periods,$T$) is large.

Algorithms that use the Newton or quasi-Newton directions always converge to the solutions of linear models when $\lambda$=1.0. With this step length, E-Newton solutions require a single iteration, as long as the Jacobian is exactly computed, and E-QNewton solutions require at most $2mT$ iterations. For nonlinear models, the convergence properties of the pair of algorithms is improved if line-search procedures are used to examine alternative step lengths, whenever the default length ($\lambda = 1$, typically) yields an insufficient movement toward the MCE solution. Two line-search procedures are available. One is the simple Armijo procedure, which repeatedly shortens the

[2]The solution algorithms are described in detail in Flint Brayton, "Two Practical Algorithms for Solving Rational Expectations Models," FEDS Working Paper 2011-44.

step length until either satisfactory progress is achieved or the maximum number of step-length iterations is reached. The other is the non-monotone step-length procedure of La Cruz, Martinez, and Raydan (LMR).

The **mce_solve_subs** subroutines include two other solution algorithms. One is the Fair-Taylor algorithm, which is the special case of (2) in which $U$ is the identity matrix. The other is an alternative, and generally less efficient, version of Broyden's method that works directly with the inverse of the approximate Jacobian matrix and does not make use of the *limited memory* approach.

**The a_opts argument**

The second subroutine argument of **mce_run**, the *a_opts* string, is used to choose a solution algorithm and set related options. Setting *a_opts* to a null string ("") has two possible effects. If this assignment is made in the first call to **mce_run** in a program, all keywords are set to their default values (see table 5). If the assignment of a null string is made in subsequent calls to **mce_run** in the same program, the algorithm and related options chosen in the previous call are retained.

The *a_opts* keywords are divided into three groups. The keywords associated with the algorithm and Jacobian group are summarized in table 2. In this group, the *meth* keyword specifies the solution algorithm; the default is "newton" (E-Newton). The *jinit* keyword controls the construction of the initial Jacobian or approximate Jacobian. Three of the five valid *jinit* settings are designed to be used with E-Newton: "every" computes an exact Jacobian; "linear" employs a shortcut that efficiently computes an exact Jacobian for any linear MCE model whose MCE variables are dated one period in the future; "interp(n)" employs a shortcut that computes every *nth* Jacobian column exactly and all other columns by interpolation. For mildly nonlinear MCE models, the most efficient approach frequently is "interp(n)" with $n$ chosen to be between four and twelve. The "every" setting should only be used with highly nonlinear models. Two *jinit* settings are designed for use with E-QNewton: "bd" and "identity." "bd" creates a block-diagonal initial approximate Jacobian that is easily computed and inverted and frequently reduces solution time compared with the more conventional (for Broyden methods) "identity" setting.

The E-Newton Jacobian is updated after any iteration that makes insufficient progress toward the MCE solution.[3] Progress is measured by the

[3]The inherent nature of the E-QNewton approximate Jacobian requires that it always

Table 2: Algorithm and Jacobian *a_opts* Keywords

| keyword | settings | description |
|---|---|---|
| meth= | newton | E-Newton algorithm |
| | qnewton | E-QNewton algorithm |
| | broy | version of Broyden's quasi-Newton algorithm that is generally less efficient than E-QNewton |
| | ft | Fair-Taylor algorithm |
| jinit= | every | calculate all matrix elements exactly |
| | interp(n) | calculate every *nth* column exactly and all other columns by interpolation |
| | linear | calculate only the subset of columns needed to construct the exact matrix of a linear MCE model |
| | bd | block-diagonal matrix |
| | identity | identity matrix |
| jupdate= | | same options as *jinit*; only relevant for E-Newton; defaults to setting of *jinit* |
| jt= | scalar | E-Newton Jacobian is updated if the ratio of the sum of squared expecations errors in sucessive iteration is greater than the scalar |
| broymax= | integer | E-QNewton maximum limited-memory parameter |

ratio of the sum of squared expecations errors in the just-completed iteration to the sum in the previous iteration. If the ratio is greater than the scalar $n$ assigned by the *jt* keyword, an update takes place (default: $n = .5$). If an update takes place, the *jupdate* keyword designates how the E-Newton Jacobian is updated; this keyword defaults to the setting of *jinit*.

The E-QNewton algorithm uses a *limited-memory* procedure that at iteration $i$ uses the step directions and lengths from the previous $i$ or *broymax* iterations, whichever is smaller. The default value of *broymax* is 600.

Each iteration, line-search procedures can be used to determine how far (step length) the expectations estimates (or their exogenous components) are adjusted in the optimal Newton or quasi-Newton direction. The line-

be updated between iterations.

Table 3: Line-search *a_opts* Keywords

| keyword | settings | description |
|---|---|---|
| lt= | scalar | line search executes if the ratio of the sum of squared expecations errors in sucessive iterations is greater than the scalar |
| lmeth= | armijo | default method for E-Newton |
| | lmr | default method for E-QNewton |
| | none | |
| lmax= | integer | maximum number of linesearch iterations |
| stepmax= | scalar | maximum step length |
| lrat= | scalar | Armijo step-length shortening parameter |
| lambda= | scalar | Fair-Taylor fixed step length |

search keywords are summarized in table 3. Line search takes place during any iteration that makes insufficient progress toward the MCE solution at the initial step length. Progress is measured by the ratio of the sum of squared expectations errors in the current iteration to the sum in the previous iteration. If the progress ratio is greater than the scalar assigned to the *lt* keyword, the line-search method specified by the *lmeth* keyword executes. Line search can be turned off by setting $lmeth = none$.

The chosen solution algorithm iterates until the maximum absolute expectations error is less than the convergence criteria or the maximum number of iterates is reached. The keywords $c$ and $m$ are used to override the default settings of these two parameters. The $p$ keyword is used to override the default perturbation factor applied to each MCE expectations estimate

Table 4: Other *a_opts* Keywords

| keyword | settings | description |
|---|---|---|
| c= | scalar | convergence criteria |
| m= | integer | maximum number of MCE iterations |
| p= | scalar | perturbation factor for MCE derivatives |

(or exogenous compontent) in running any simulations needed for computing the Jacobian or initial approximate Jacobian

Table 5: *a_opts* Defaults

| keyword | algorithm | default | keyword | algorithm | default |
|---|---|---|---|---|---|
| meth | | newton | lt | all | .9 |
| jinit | newton | interp(12) | lmax | all | 10 |
| | qnewton | bd | stepmax | all | 1.0 |
| | broy | bd | lrat | all | .5 |
| | ft | identity | lambda | ft | 1.0 |
| jupdate | newton | = jinit | m | newton | 20 |
| jt | newton | .5 | | qnewton | 200 |
| broymax | qnewton | 600 | | broy | 200 |
| lmeth | newton | armijo | | ft | 500 |
| | qnewton | lmr | c | all | 1e-6 |
| | broy | lmr | p | all ex ft | .001 |
| | ft | na | | | |

**Examples of a_opts**

In the following examples, the string variable %astr is to be used as the *a_opts* argument of **mce_run**. Only the overrides to the default settings need to be specified.

```
%astr = "jinit=linear,c=1e-8,lmeth=none"
%astr = "meth=qnewton,jinit=identity,lmax=50"
%astr = ""
```

In the first example, the solution algorithm defaults to E-Newton, the Jacobian is initialized (and updated) using the "linear" shortcut that is applicable to many linear MCE models, the convergence criteria is tightened to 1e-8, and no step-length line-search is undertaken. The second example sets the solution algorithm to E-QNewton, the initial approximate Jacobian to the identity matrix, and the maximum number of line-search iterations to fifty.

## 5. Simulation Execution

### General discussion

The **mce_run** subroutine can initiate three types of MCE simulation experiments: "single", "opt", and "opttc". The "single" type runs an MCE simulation whose inputs are the initial and terminal values of the model's endogenous variables and the projected paths of any shocks and exogenous variables the model may contain. The "opt" type is an experiment that requires a set of MCE simulations to find the paths of one or more instrument variables that minimize the value of a loss function. The "opttc" type also involves optimization, but the optimizing agent cannot commit to a fixed instrument trajectory. In this case, the algorithm attempts to find the Nash time-consistent solution using a backward-induction approach. As discussed below, "opttc" algorithm provides only an approximate solution. The two optimization simulation types permit inequality constraints to be imposed on linear combinations of the model's endogenous variables. When such constraints are present, the solution algorithm requires access to the quadratic programming function in either R or Matlab.

*The "opt" simtype*

Let $n$ be the number of instrument variables, $T_n$ the number of periods in which the instruments are active, and $\mathbf{x}$ the ($nT_n$ x 1) stacked vector of instruments. In addition, let $m$ be the number of target variables, $T_m$ the number of periods over which the loss function is to be minimized, $\mathbf{y}$ the ($mT_m$ x 1) stacked vector of targets, and $\mathbf{y}^*$ the ($mT_m$ x 1) stacked vector of their desired paths. The objective is to minimize the quadratic loss function,

$$min \quad f = (\mathbf{y} - \mathbf{y}^*)'\mathbf{W}(\mathbf{y} - \mathbf{y}^*), \tag{3}$$

where $\mathbf{W}$ is a ($mT_m$ x $mT_m$) diagonal matrix of weights.

The minimization of (3) requires a single iteration for a linear model and a sequence of iterations for a nonlinear model. When no constraints are present, each iteration uses Newton's method to update the estimate of $\mathbf{x}$ based on the first (gradient) and second (hessian) derivatives of $f$ with respect to the instrument variables. A key component of the grandient and hessian is the matrix ($\mathbf{D}$) of deriviatives of the instruments with respect to the targets. $\mathbf{D}$ is calculated numerically with a set of $nT_n$ perturbation simulations, each of which is a "single" MCE solution of a ping to one of the instrument variables in one of the periods in which the instruments are

active. Because it is generally expensive to calculate $\mathbf{D}$, its value is updated only after iterations in which the progress made in reducing the value of $f$ falls substantially short of what would be achieved if the model were linear.

There are several ways to approach the minimization of (3) when inequality constraints involving functions of endogenous variables are present. One is to add a nonlinear penalty variable to the model for each constraint and to include each penalty variable in the loss function. This approach provides approximate solutions and can be solved using the methods of unconstrained minimization. An alternative approach, which is the one described here, recasts the constrained minimization as a standard quadratic programming problem. In the code described here, the inequality constraints are required to be linear,

$$\mathbf{A}\mathbf{y_c} \geq \mathbf{b} \tag{4}$$

where $\mathbf{A}$ is a ($c$ x $mT_n$) matrix of constraint coefficients, $\mathbf{y_c}$ a ($mT_n$ x 1) stacked vector of endogenous variables, and $\mathbf{b}$ a ($cT_n$ x 1) matrix of constraint constants, and $c$ the number of constraints. The stacked vectors $\mathbf{y_c}$ and $\mathbf{y}$ correspond to the same group of endogenous variables, but the vectors contain observations drawn from time periods that may not be the same. The former vector includes the $n$ observations on each relevant endogenous variable in the period that the instruments are active; the latter vector includes the $m$ observations on each variable in the period that the loss function is minimized. If an endogenous variable appears in the constraints but not among the endogenous variables in the loss function, that variable is added to the loss function variables and assigned a weight of zero.

To complete the notation needed to describe the quadratic programming problem, express the the linearized relationship between the targets and instruments as,

$$\mathbf{y} - \mathbf{y_0} \approx \mathbf{D}(\mathbf{x} - \mathbf{x_0}) \tag{5}$$

where $\mathbf{D}$ is the ($mT_m$ x $nT_n$) gradient matrix of derivatives of the target variables with respect to the instrument variables calculated around $(\mathbf{x_0}, \mathbf{y_0})$. After using (5) to replace $\mathbf{y}$ in (3) and (4), the constrained minimization problem can be expressed as,

$$min \quad f = \mathbf{x}'\mathbf{D}'\mathbf{W}\mathbf{D}\mathbf{x} + 2(\mathbf{k} - \mathbf{y}^*)'\mathbf{W}\mathbf{D}\mathbf{x}' + cnst, \tag{6}$$

where $\mathbf{k} = \mathbf{y_0} - \mathbf{Dx_0}$ and the constraints take the form,

$$\mathbf{ADx} \geq \mathbf{b} - \mathbf{Ak} \tag{7}$$

In each quadratic programming iteration, the values of $\mathbf{x}, \mathbf{y}, \mathbf{y}*, \mathbf{k}, \mathbf{D}, \mathbf{A}$ and $\mathbf{c}$ are to construct the arguments needed by the R or Matlab quadratic programming function. The optimal constrained solution of a linear model is achieved in a single iteration. A sequence of iterations of a nonlinear model converges when the optimal instrument values from the quadratic programming step yield approximately the same solution for the target variables in the nonlinear MCE model as they do in the quadratic programming solution.

*The "opttc" simtype*

The "opttc" simulation type performs an optimization in which there are as many policymakers as there are instrument-setting dates and no policymaker can control the actions of any other policymaker. The policymaker at time $j$ $(1 \leq j \leq T_n)$ is assumed to find the value of the instrument(s) at $j$ that minimize (3) subject to (4), if applicable, when the loss function is evaluated for $T_m$ periods starting at $j$. The minimization takes as given the values of the instruments set at dates other than $j$. The algorithm seeks to find a time-consistent solution in which the optimizing actions of the policymakers are mutually consistent. The method is backward induction. Given an initial solution based on a possibly arbitrary choice of instrument values, the optimization problem of the last $(T_n)$ policymaker is solved first. Then the next-to-last policymaker optimizes, conditional on the instrument choice of the last policymaker. This sequence continues until the optimization problem of the first policymaker is solved. Sequences of this type, each starting with the last policymaker, are repeated until the difference between one sequence and the next is sufficiently small.

The "opttc" simtype does not provide exact time-consistent solutions. A first source of divergence between the true time-consistent solution and the computed solution concerns the assumption that at each iteration one policymaker optimizes and the other policymakers hold their policy instruments fixed. This assumption is incorrect except when the non-optimizing policymakers instrument is an adjustment factor to the model's true time-consistent policy rule. The "opttc" simulation procedure is designed for use with models for which it is difficult or impossible to compute the optimal time-consistent rule. In addition, for nonlinear models the solution is based

Table 6: General *s_opts* Keywords

| keyword | setting | description |
|---|---|---|
| type= | single | single MCE simulation (default) |
| | opt | optimization experiment |
| | opttc | time-consistent optimization |
| sstart= | date | first simulation period |
| send = | date | last simulation period |
| txt= | name | name of text file of EViews commands |
| terminal | | reset terminal conditions |
| scen | | create new scenario |
| suf= | | alias for new scenario |
| solveopt= | string var | set solve options |
| o= | 1,2 or 3 | amount of output per MCE simulation |
| dontstop | | terminate call to mce_run and continue program execution when error or nonconvergence occurs |
| cleanup | | delete all _$_* variables |

on the linearized relationship (**D**) between the instruments and targets faced by the first policymaker around the baseline. Constrained "opttc" simulations require access to R or Matlab.

### The s_opts argument

The third subroutine argument of **mce_run**, the *s_opts* string, is used to choose the simulation type and related parameters. The ten optional keywords are available for all simulation types to override default settings. These keywords are shown in table 6.

Valid settings of *type* are "single", "opt", and "opttc". The default is "single". The default simulation period, which is the current workfile page sample, can be overridden with the *sstart* and *send* keywords. The *txt* keyword declares the name of a text object that contains EViews commands to be executed after the model processing and algorithm declaration steps of **mce_run** have executed and before the simulation step executes. Such a text file is useful, for example, for setting up the shocks to be included

in the simulation experiment, when the model processing step involves the computation of tracking adds. In such a case, any shocks created prior to the call to **mce_run** would only lead to offsetting shifts in the add factors.

Inclusion of the *terminal* keyword causes the values of all endogenous variables whose leads appear in the second operational model to be reset beyond the end of the simulation period. The new values are based on the solution values for these variables at the far end of the first solution of the first operational model in the simulation experiment. This approach calculates accurate terminal conditions only when the first operational model has the same long-run characteristics as the MCE model. FRB/US satisfies this condition.

The *scen* keyword can be used to create a new scenario that applies to each of the operational models and the *suf* keyword to designate the scenario alias. the *solveopt* keyword can be used to specify solve options. The default setting of the latter is the string "o=n,g=12,z=1e-12".

The "o" keyword sets the amount of output displayed for each MCE simulation. The most amount of output displays with "o=1" and the least with "o=3". The "cleanup" keyword causes all _$_* variables created during the course of the execution of **mce_run** to be deleted when it is finished executing.

Inclusion of the *dontstop* keyword causes the program that calls **mce_run** to continue executing the commands that come after the subroutine call when the E-Newton or E-QNewton MCE iterations fail to converge or when, in a "single" simulation, EViews generates an error when trying to solve either the backward or forward models. This feature may be useful when running many simulations in a loop and the failure of one or a few of the simulations can be tolerated. The contents of the %mce_finish string indicate how a call to **mce_run** terminates. After a successful call to **mce_run**, %mce_finish = "yes". When "dontstop" is included and the MCE iterations do not converge, %mce_finish = "no". When "dontstop" is included and a solver error occurs, %mce_finish = "failed solve".

The "opt" and "opttc" simulation types share two required keywords and 15 optional keywords. These are shown in table 7. The required *instrus* keyword must be assigned to a group object of the exogenous instrument variables whose values will be chosen to minimize the loss function. The required *targs* keyword must be assigned to a group object of the endogenous target variables that appear in the loss function. The other two elements of the loss function – the desired values of the target variables and the weights

Table 7: Optimization *s_opts* Keywords

| keyword | setting | description |
|---|---|---|
| instrus= | group | optimization instruments (required) |
| targs= | group | optimization targets (required) |
| istart= | date | first date of instrument range |
| iend= | date | last date of instrument range |
| lstart= | date | first date of loss evaluation range |
| lend= | date | last date of loss evaluation range |
| m= | scalar | max number of optimziation iterations |
| c= | number | optimization convergence criteria |
| lmax= | scalar | max number of optimization line-search iterations |
| p= | number | instrument perturbation factor |
| stepmax= | number | maximum instrument step length |
| oo= | 1,2,3,4 | amount of intermediate output in optimization experiments |
| cnstr= | name | name of text object with inequaltiy constraints (required) |
| matlab | | use Matlab for quadratic programming (the default is R) |
| ideriv= | yes/no | compute/do not compute instrument derivatives |
| /xopen | | do not open R or Matlab |
| /xclose | | do not close R or Matlab |
| d= | number | damping factor, $0 < d \leq 1$ (opttc only) |

– must also be specified, but for these the algorithm expects to find series in the workfile with names based on the names of the targets. Specifically, each target variable must be associated with a pair of series, one whose name has an "_t" suffix attached to the target name and contains desired values, and one with an "_w" suffix that contains weights. If "abc" is one of the target variables, the corresponding series "abc_t" and "abc_w" must have been created before the execution of the simulation.

The optional optimization keywords enable defaults (table 8) to be overridden for the loss function evaluation period (*lstart*, *lend*), the period over which the instrument variables are active (*istart*, *iend*), and various param-

Table 8: *s_opts* Defaults

| keyword | default | keyword | default |
|---|---|---|---|
| type= | single | lstart= | sstart |
| suf= | _1 | lend= | $60th$ simulation period |
| o= | 1 (single) | m= | 15 |
| | 3 (opt/optqp) | oo= | 3 |
| sstart= | page sample start | c= | 1e-5 |
| send= | page sample end | lmax= | 10 |
| istart= | sstart | p= | .01 |
| iend= | $40th$ sim period | stepmax= | 1.0 |
| solveopt= | o=n,g=12,z=1e-12 | ideriv | yes |
| d= | 1 | | |

eters associated with the mechanics of the optimization iterations. The $d$ keyword can be used to set a between-iteration damping factor for "opttc". No damping occurs when $d = 1$, the default.

The *cnstr* keyword is used to declare the name of a text object of inequality constraints whose $ith$ line has the form:

$$a_{i,1} * y_1 + \ldots + a_{i,k} * y_k >= b_i \tag{8}$$

The variables $y_1, \ldots, y_k$ must be the names of endogenous variables. The coefficients $a_{i,1}, \ldots, a_{i,k}$ and constant $b_i$ must be specific numerical values. Coefficients must be placed before variables and be followed by the "*" character. Coefficients that equal one (or minus one) may be omitted.

Inclusion of the *matlab* keyword specifies that "opt" and "opttc" simulations use the quadratic programming function in Matlab; the default is the R function. The */xopen* and */xclose* keywords can be used in a sequence of constrained optimization simulations to avoid the repeated opening and closing of the external link from Eviews to R or Matlab.

**Examples of s_opts**

The first set of examples is based on the first model (model $s$) presented in section 3 to illustrate the *m_opts* string. For present purposes, the only important attributes of the model are that it contains endogenous variables

$p$, $y$, and $r$, that it contains the exogenous variable *shk* in the equation for $r$, that it has add factors, and that it has been parsed or declared in the first argument of subroutine **mce_run**. Note that the simulation range is generally determined by the sample period in effect when the subroutine is called. All of the following examples assume that the first simulation period is 2001q1 and that the string variable %sstr will be used as the third subroutine argument.

The first example runs a single MCE simulation in which the exogenous variable *shk* is raised by one unit in the first simulation period, as specified in the text object named *shock*.

```
text shock
shock.append smpl 2001q1 2001q1
shock.append series shk = shk + 1
%sstr = "txt=shock"
smpl 2001q1 2050q4
call mce_run(..., ..., %sstr)
```

The second example executes up an unconstrained optimization experiment in which *shk* is the instrument for minimizing the weighted, squared deviations of $y$ and $p$ from zero. Based on default settings, the path of *shk* is optimally chosen over the first 40 simulation periods to minimize the loss function over the first 60 simulation periods.

```
smpl @all
group ii shk
group tt p y
series p_t = 0
series p_w = 1
series y_t = 0
series y_w = 2
%sstr = "sim=opt,instrus=ii,targs=tt"
smpl 2001q1 2050q4
call mce_run(..., ..., %sstr)
```

The third example sets up a constrained optimization experiment that augments the specification of the second example with the restriction that $r$ cannot fall below zero. (Note that the first seven lines of the second example are omitted to save space.) The quadratic programming function in Matlab is used.

```
text cc
cc.append r >= 0
%sstr = "sim=opt,instrus=ii,targs=tt,cnstr=cc,matlab"
smpl 2001q1 2050q4
call mce_run(..., ..., %sstr)
```

The fourth example executes a time-consistent optimization. For the "opttc" simtype, the declared workfile sample (or any overrides) and the explicit or default setting of *lend* describe the simulation and loss function ranges associated with the first policymaker, who in this case chooses the value of *shk* in 2001q1 that minimizes the loss function over the 40 quarters from 2001q1 to 2010q4, based on a 200-period simulation that extends to 2050q4. The setting *iend* determines the date associated with the last policymaker, who in this case chooses the value of *shk* in 2008q4 to minimize the loss function over the 40 quarters starting at that date based on a 200-period simulation also starting at that date.

```
text cc
cc.append r >= 0
%sstr = "sim=opt,instrus=ii,targs=tt,cnstr=cc,matlab"
%sstr = %sstr + ",iend=2008q4,lend=2010q4"
smpl 2001q1 2050q4
call mce_run(..., ..., %sstr)
```

In the previous three examples, the commands to create the two groups and four series are executed prior to the call to **mce_run**. Because these commands do not interfere with any of the operations of the first two steps of the subroutine, this approach yields the same outcome as placing the group and series creation commands in a text object and using using the *txt* keyword to execute the commands at the start of the simulation step.

# 6. More Examples

### Example 1: a pair of "single" simulations

The first example illustrates how to run a "single" simulation and how to set up a sequence of simulations of the same model using the same solution algorithm. The example uses the first model presented in section 3 and assumes that the values of its variables and coefficients have been defined.

```
model s
s.append p = cp(1)*p(-1) + (.98-cp(1))*p(1) + cp(2)*y
s.append y = cy(1)*y(-1) + (.98-cy(1))*y(1) + cy(2)*(r - p(1))
s.append r = cr(1)*r(-1) + (1-cr(1))*(cr(2)*p + cr(3)*y) + shk
```

The following commands parse model *s* to create the two operational models, assign tracking add factors, select the E-QNewton algorithm and a block-diagonal initial approximate Jacobian, and execute a pair of simulations. The *m_opts* and *a_opts* arguments are null strings in the second call of **mce_run**, indicating that the models created and algorithm chosen in the first simulation are to be used again. In multiple calls of **mce_run** of this type, the dimensions of various matrices and vectors are set only in the first call. One implication of this is that the simulation period must be the same in each call. Each simulation starts in 2001q1 and ends in 2050q4.

```
' first sim
  text shock1
  shock1.append smpl 2001q1 2001q1
  shock1.append series shk = shk + 1
  %mstr = "create,mod=s,adds,track"
  %astr = "meth=qnewton,jinit=bd"
  %sstr = "txt=shock1,scen,suf=_2"
  smpl 2001q1 2050q4
  series shk = 0
  call mce_run(%mstr,%astr,%sstr)
  smpl 2001q1 2001q1
  shk = shk - 1    'undo first shock

'second sim
  text shock2
```

```
shock2.append smpl 2001q1 2001q1
shock2.append series y_a = y_a - 1
%sstr = "txt=shock2,scen,suf=_3"
smpl 2001q1 2050q4
call mce_run("","",%sstr)
```

Solution values are stored in variables named with an "_2" suffix in the first simulation and an "_3" suffix in the second simulation. It is necessary to define the shock in the first simulation in a text file to avoid its effect being negated by the calculation of tracking add factors. Because the second simulation does not recalculate add factors, its shock can be specified either in a text file (as is done) or in the body of the program before the second call of **mce_run**.

**Example 2: an "opt" simulation**

For this example, assume that model *s* from the first example is augmented with an identity for the first difference of *r*.

```
s.append dr = r - r(-1)
```

The following commands find the path for *shk* that minimizes the weighted squared devations of *p*, *y* and *dr* from zero in the presence of a shock to the add factor in the equation for *y*. The loss function is minimized over the default interval, which is the first 60 periods of the simulation, by varying the path of *shk* over the 40-period default interval.

```
smpl @all
series shk = 0
group ii shk
group tt p y dr
series p_t = 0
series p_w = 1
series y_t = 0
series y_w = 2
series dr_t = 0
series dr_w = .5
text shock1
shock1.append smpl 2001q1 2001q1
shock1.append series y_a = y_a - 1
```

```
%mstr = "create,mod=s,adds,track"
%astr = "jinit=linear"
%sstr = "type=opt,txt=shock1,instrus=ii,targs=tt,scen,suf=_2"
smpl 2001q1 2050q4
call mce_run(%mstr,%astr,%sstr)
```

Because the optimization experiment requires many MCE solutions, it is generally more efficient to use the default E-Newton algorithm, especially for models that are linear or nearly linear. In this example, the model is linear, which also permits the "linear" shortcut to be used in forming the Jacobian.

**Example 3: a sequencee of simulations**

This example is based on an MCE model whose structure cannot be processed by **mce_run** and assumes that the user has already created a pair of operational models named *s_b* and *s_f*.

```
model s_b
s_b.append p = cp(1)*p(-1) + (.98-cp(1))*zp + cp(2)*y
s_b.append y = cy(1)*y(-1) + (.98-cy(1))*zy + cy(2)*(r - zp)
s_b.append r = cr(1)*r(-1) + (1-cr(1))*(cr(2)*p + cr(3)*y) + shk
s_b.append dr = r - r(-1)
s_b.append zp = p(-1)
s_b.append zy = y(-1)

model s_f
s_f.append ezp = zp - p(-1)
s_f.append ezy = zy - y(-1)
```

The next example consists of three simulation experiments, each of which involves the same perturbation to the add factor on the equation for $y$ in the first simulation period. A single simulation is followed by two optimization simulations. The first optimization experiment constraints $r$ to be nonnegative and uses the default external connection to R to impose this constraint. The second optimization is unconstrained. In the optimization experiments, the values of the instrument variable, *shk*, are chosen over the period from the start of the simulation to 2015q4 (an override specified by "iend") to minimize the loss function from the start of the simulation to 2015q4 (the default 60-period setting). The *m_opts* and *a_opts* arguments are null strings in the second and third calls of **mce_run**, indicating that the models created and algorithm chosen in the first simulation are to be used again.

```
' first experiment
  text shock1
  shock1.append smpl 2001q1 2001q1
  shock1.append series y_a = y_a - 1
  %vars = "zp zy"
  %mstr = "mod_b=s_b,mod_f=s_f,mce_vars=%vars,adds,track"
  %astr = "jinit=linear"
  %sstr1 = "type=single,scen,suf=_1"
  smpl 2001q1 2050q4
  call mce_run(%mstr,%astr,%sstr1)

' second experiment -- constrained optimization
  smpl @all
  group ii shk
  group tt p y dr
  series p_t = 0
  series p_w = 1
  series y_t = 0
  series y_w = 2
  series dr_t = 0
  series dr_w = .5
  series shk = 0
  text cc
  cc r >= 0
  %mstr = ""
  %astr = ""
  %sstr2 = "type=opt,instrus=ii,targs=tt,scen,
  %sstr2 = %sstr2 + ",suf=_2,cnstr=cc,iend=2015q4"
  smpl 2001q1 2050q4
  call mce_run(%mstr,%astr,%sstr2)

' third experiment -- unconstrained optimization
  %sstr3 = "type=opt,instrus=ii,targs=tt,scen,
  %sstr3 = %sstr3 + ",suf=_3,iend=2015q4"
  smpl 2001q1 2050q4
  call mce_run("","",%sstr3)
```

## 7. FRB/US: Notes and Examples

FRB/US contains 30 future-dated endogenous variables whose solutions may be MC (ie, perfect foresight) or taken from backward-looking VAR-based equations. Many FRB/US simulations assume MC expectations for a subset of the expectations variables and VAR expectations for the others.

The construction of the pair of FRB/US operational models needed for the E-Newton and E-QNewton algorithms is based on the following design:

- The first or backward-looking operational model contains the VAR-based equations for all 30 expectations variables. This model is usually one of the stored FRB/US VAR versions, such as *stdver*.

- The expectations instruments whose values are iteratively adjusted to impose the MCE requirement are the EViews add factors on the VAR-based expectations equations. For example, when the expectations variable $zpicxfe$ has an MC solution, $zpicxfe\_a$ is an instrument. By convention, the names of all expectations variables in the first model start with a $z$.

- The second or foreward-looking operational model contains two equations for each MC expectations variable. One defines the appropriate forward expectation and the other the expectations error. For example, $zpicxfe$ is the rate of $picxfe$ inflation expected next period and, when it has as an MC solution, the second model contains a pair of equations of the form: $wpicxfe_t = picxfe_{t+1}$ and $ezpicxfe_t = zpicxfe_t - wpicxfe_t$. By convention, the names of all expectations variables in the second model start with a $w$. The expectations equations in this model are usually taken from one of the stored FRB/US PF versions, such as $pfver$.

The construction of the pair of operational FRB/US models can be carried out by the **mce_load_frbus** subroutine,

```
call mce_load_frbus(%lstr)
```

whose argument (%lstr) is a string variable that is used to define five required keywords and two optional keyword. The keywords are described in table 9.

The following example sets up the pair operational models for the case in which the FRB/US equations are taken from versions *stdver* and $pfver$,

Table 9: Keywords for mce_load_frbus

| keyword | setting | description |
|---|---|---|
| *required* | ———— | |
| mce_vars= | string var | list of MCE variables |
| mod_b= | string var | name of source model for first operational model |
| path_b= | string var | path for mod_b equations |
| mod_f= | string var | name of source model for second operational model |
| path_f= | string var | path for mod_f equations |
| *optional* | ———— | |
| allbut= | string var | list of mod_b equations to exclude |
| only= | string var | list of mod_b equations to include |

and the 12 expectations variables that are used in asset pricing have MC expectations,

```
%varmod = "stdver"
%varpath = "../mods/"
%mcemod = "pfver"
%mcepath = "../mods/"
%zvars = "zdivgr zgap05 zgap10 zgap30 zrff5 zrff10 zrff30"
%zvars = %zvars + " zpi10 zpi10f zpic30 zpib5 zpic58"
%oo = "mce_vars=%zvars,mod_b=%varmod,path_b=%varpath,"
%oo = %oo + "mod_f=%mcemod,path_f=%mcepath"
call mce_load_frbus(%oo)
```

In the example, the code for the operational models is located in a directory named *mods* that is parallel to the directory in which the simulation program resides.

If one wanted to omit the *ptr* equation, for example, from the first operational model (presumably for later replacement with an alternative specification), the first command below would be added and the definition of the %oo string would be modified as indicated.

```
%qq = "ptr"
%oo = "mce_vars=%zvars,mod_b=%varmod,path_b=%varpath,"
%oo = %oo + "mod_f=%mcemod,path_f=%mcepath,allbut=%qq"
```

As noted above, the second operational model contains a set of $w$ variables and a set of $e$ variables, one of each for each MC expectation. The function of the **make_frbus_mcevars** subroutine is to assign values to these variables. It sets the $w$ variables equal to their $z$ counterparts and the $e$ variables to zero. The subroutine's single argument is a string containing the names of the MCE variables. The form of the subroutine call in the current example is as follows.

```
call make_frbus_mcevars(%zvars)
```

To complete the example, the following code runs a simulation of an interest rate shock when monetary policy characterized by the Taylor-type rule coded in the $rfftay$ equation.

```
text shock1
shock1.append smpl 2012q1 2012q1
shock1.append series rfftay_a = rfftay_a + 1
%modstr = "mod_b=stdver,mod_f=pfver,mce_vars=%zvars,adds,track"
%algstr = "meth=qnewton"
%simstr = "type=single,txt=shock1,suf=_2,scen"
smpl 2012q1 2060q4
call mce_run(%modstr,%algstr,%simstr)
```

Note that keywords *mod_b* and *mod_f* are assigned to the same model names that are usedin the call to **mce_run**, because the latter subroutine sets the names of the operational models to the names of the input models.