Desciption of the code

This document contains description of the changes made to the DYNARE v. 3.64 package code through August 2011 by Sergey Slobodyan. The changes allow for estimation of the DSGE models under adaptive learning modelled as constant gain (CG) or Kalman filter (KF) learning. Please note that I was working with files modified by Raf Wouters to a certain degree, so that the initial code differs slightly from DYNARE 3.64. The files thus modified are copyrighted by Sergey Slobodyan and Raf Wouters, and could be freely modified and used, provided a proper notice of such rights is given.

The package provided allows replication of the basic results of Slobodyan and Wouters, "Learning in a medium-scale DSGE model with expectations based on small forecasting models", SW for short.[1] Every subdirectory contains all the files necessary to run a particular simulation. Just unzip the package so that the files that were changed would be used instead of standard DYNARE ones, make a particular desired directory home, and run `start.m` in MATLAB.

For a general description of our approach, see Section "Learning Setup" in SW and the companion paper.

Limitations: so far, DSGE–VAR approach is not touched upon; no working with diffuse priors is allowed. Only the models with one lag and one lead were tested so far. Lags higher than one could be easily accomodated by specifying a corresponding variable in the `.mod` file, such as **cl = c(-1)**, and then using **cl** as a RHS variable in forecasting equations. A similar trick with leads will work too, but coefficients of forecasting functions for first, second, etc. lead will be learned separately; this might present an additional source of estimation uncertainty.

1. To run estimation under learning, set **kalman_algo** in the **estimation command** in the `.mod` file equal to one of the following numbers: 101, 102, 103, 201, 202, 203, 602, 603. The last digit specifies one of the four ways (described in a companion paper and also in SW) of selecting the initial beliefs.

   (a) 1 stands for taking them as given from a file. The file should contain **betamat**, a matrix of initial beliefs, and **SecondMoments**, a matrix of initial 2^{nd} moments of the right–hand side variables in agents' forecasting equations. Usually, one obtains these beliefs by running OLS, but any **betamat** and **SecondMoments** of proper dimensions should work. The file containing these two variables is specified in the variable **use_bet** in `dynare_estimation.m`. For example, if **use_bet** equals 'A', the program will attempt to read file `A_bet.mat`.

   This option is not implemented for Kalman filter learning.

[1] The code that allows for constant gain learning is also included into this release, but for replication of results of our companion paper, "Learning in an estimated medium–scale DSGE model", please see the code release that will be available at the JEDC site shortly.

   (b) 2 means that the initial beliefs are taken to be consistent with a REE that obtains for parameter values specified in a file given in **use_model** variable in `dynare_estimation.m`. If **use_model** equals 'B', the program will attempt to read file `B_mode.mat`. The file should contain a parameter vector **xparam1** and the Hessian matrix **hh**.

       This option should work for both CG and KF learning.

   (c) 3 means that the initial beliefs are re–calculated every time the parameter vector changes during the estimation. Variables **use_bet** and **use_model** are ignored.

       This option should work for both CG and KF learning.

   (d) If the first digit of **kalman_algo** equals '1', constant gain learning without constant in the forecasting equations in the forecasting equations will be implemented; '2' means CG with constant is to be run. First digit '6' specifies Kalman filter learning with a constant. '5' is reserved for KF learning without constant which is not implemented as the constants play a very important role in typical KF learning setups. For a detailed description of working with constants, see below.

   (e) Selecting **kalman_algo** equal to 1 gives the standard estimation under rational expectations.

   (f) Note that the current release will deliver replication of basic results of SW only. For example, the CG code included will not automatically deliver diagonal CG results of Section V.C of SW, the code for the latter being even more specific than the standard KF or CG code.

2. It is assumed that the variable **forwards** in `dynare_estimation.m` correctly specifies the set of forward–looking variables of the model. In other words, we assume that the agents form expectations of all the forward–looking variables as linear functions of endogenous variables and shocks; currently, it is impossible to use adaptive learning for a subset of **forwards** and use rational expectations for the rest.

   For the CG learning, variables **states** and **shocks** give, correspondingly, endogenous variables (usually denoted as '**y**' in Evans&Honkapohja book) and shocks ('**w**' in the book) that are assumed to be present on the RHS *of all the forecasting equations.* Given that our code uses a common assumption in the adaptive learning literature, namely that the agents always know correctly the stochastic processes for the stochastic shocks and thus are able to form expected values of the shocks given current values, we have to treat these two types of variables differently.

   For KF learning, variable **NUM_MODELS** specifies the number of forecasting models the agents use (one in the baseline model, five in multi-model specifications). Every model is a cell array named **m1**, **m2**, etc. If only one model is used, it should be **m1**. Every model describes all forecasting equation. As in CG, all forwards have to be present in every forecasting model, otherwise the program will abort.

Within a single model, a forecasting equation is described by a block of the form '**inve**',{'**inve**','**invel**'},{};

The first part of the block — '**inve**' — specifies, in this case, an equation for investment variable. The second part — {'**inve**','**invel**'} — describes endogenous states used on the RHS of this forecasting equation. The third part — {} — is reserved for the shocks (more generally, any variables whose values at time t are assumed to be known to the agents while producing their forecasts of time $t+1$ variables. Currently this is a reserved option, please don't use any variables there.

In the examples provided with this release, **m1** forecasting model describes seven individual equations in which every forward-looking variable is explained using two lags of itself. A constant is added automatically because **kalman_algo**=603 is used.

It is not necessary to have **forwards, states, shocks,** blocks within individual models, and variables within blocks in an alphabetical order, as sorting is performed by `dynare_estimation.m`.

We utilize the so–called 'time t' dating, whereas values of y_t are assumed to be known in period t. For details, see SW and Evans&Honkapohja book.

Generally, we assume that the agents have access to current filtered estimates of the model–based variables; they could use observed variables on the RHS as well.

3. *Projection facility*: If current beliefs imply that the current law of motion,

$$y_t = \mu_t + T_t y_{t-1} + R_t \varepsilon_t,$$

is characterized by explosive behavior (at least one eigenvalue of T_t is above **options_.qz_criterium**, a number slightly larger than 1), then beliefs update at such step can be discarded. Updating is discarded if **options_.corr**=1 (default value). To change it to **options_.corr**=0 (no corrective step is undertaken after such "boundary hit"), set the variable **Corr** in `dynare_estimation.m` to 0.

Additionally, every instance of "boundary hit" could be punished by a value given in **options_.lik_penalty**, currently equal to 10. To change the value, reset variable **Lik_penalty** in `dynare_estimation.m`. It is possible to perform no correction in case of boundary hits but still assign penalty for such violations.

For KF learning, the described projection facility is implemented for T derived as if a single forecasting model were used. After such checks for all the models, T based on the aggregate forecasting model is calculated, and smoothing PF is implemented there. For all eigenvalues of T that are larger than some $\bar{\lambda} < 1$ (typically, 0.99 or 0.995), the transformed eigenvalue is calculated as

$$\lambda = \bar{\lambda} + \delta \arctan\left(\frac{\lambda - \bar{\lambda}}{\delta}\right).$$

This procedure smoothly maps $\lambda \in \left[\bar{\lambda}, \infty\right]$ into $\lambda \in \left[\bar{\lambda}, 1\right]$, parameter δ controlling the shape of the adjustment function. All eigenvalues which are smaller than $\bar{\lambda}$ are left intact. Finally, the matrix T is recomputed using its own eigenvectors and the transformed eigenvalues. The procedure is similar to the way Hessian is adjusted in case of non-positive definiteness, see below.

Typically, after all the individual PFs, the aggregate smoothing PF is not used.

4. Hessian derivation subroutine is rewritten: given the high probability of projection facility hits which result in a 'cliff', or a sharply higher value of the objective function (objective function tends to increase even if penalty is set to zero if beliefs are allowed to be unstable for a while), `hessian_min_1S.m` attempts to identify which direction along every variable is 'cliffless', and then uses that direction for calculations (in other words, instead of using points x_0, $x_0 - h$, and $x_0 + h$ to estimate second derivatives, I try to see which direction, $+h$ or $-h$, is cliffless, and then use points x_0, $x_0 - h$, and $x_0 - 2h$, say). If this procedure is not successful, I reduce the stepsize along the corresponding parameter. If a joint step along parameters i and j needed to estimate cross–derivative results in a cliff, the stepsize is reduced for both parameters. The derivations are repeated until no cliff is encountered during the whole Hessian calculation.

   During the Hessian calculations, the program looks for a point with the objective function value that is lower than the one supplied to `hessian_min_1S.m`. Because of the cliffs, optimization often stops at a point which is clearly not a minimum. However, as the Hessian derivation procedure ignores bounds, the 'better' point could be such only because it is beyond the bounds.

   If this Hessian is still not positive definite, all negative eigenvalues are set equal to the smallest positive one, and a new Hessian is computed using old eigenvectors and new eigenvalues. The resulting matrix is commonly useful for proposal density in the Metropolis step. If manipulation has been performed, Laplace approximation is not valid. The larger is the most negative eigenvalue of the original Hessian (printed by the hessian_min_1S procedure), the larger is the distortion.

5. *Ridge Correction (only for CG).* Given that some specifications are extremely sensitive because the matrix **SecMom** (or **SecondMoments**) posesses a smallest eigenvalue close to 0, *Ridge correction* is implemented. If the smallest eigenvalue is less than **options_.ridge**, usually equal

to 1E-05, **options_.ridge** times identity matrix is added to the **SecMom.** Optionally, the addition could be made smaller or large by utilizing **options_.ridge_adj** (1 by default): the adjustment is given as $\frac{\textbf{options\_.ridge}}{\textbf{options\_.ridge\_adj}} \times I$. To change the default values, reset variables **Ridge** and **Ridge_adj** in `dynare_estimation.m`.

6. *Constant (for CG).* To run estimation with a constant, set **options_.kalman_algo** (in the `.mod` file) to a number greater than 200 (currently only 202 and 203 procedures are working reliably, 201 should work but was never tested).The beliefs are adjusted *assuming* that the corresponding variables are demeaned. This assumption is innocuous for 202 and 203, as the beliefs are derived in `BetaFromTR.m` where **T** and **R** are DYNARE–produced matrices that operate on demeaned variables (DYNARE substracts the steady state values from all variables). On the other hand, a standard way to use 201 procedure is to use output from **oo_.Filtered_variables** which is also "demeaned" by the DYNARE in Kalman filtering step (but see below for a note of caution). Therefore, the standard 201 procedure should be fine as well.

   After deriving the **betamat** and **SecondMoments** matrices, an adjustment for the constant is made. This is performed in the beginning of subroutines `DiffuseLikelihoodL1_options_const.m` and `DiffuseKalmanSmootherL1_options_const.m`. (The subroutines without '`const`' in the name are for no constant estimation). The basic idea of adjustment is as follows: suppose that we have derived the relationship between *demeaned* forward looking and state (or some other) variables as

   $$y_t^f - \overline{y}^f = \beta \left(y_{t-1}^s - \overline{y}^s \right).$$

   Then,

   $$y_t^f = \overline{y}^f - \beta \overline{y}^s + \beta y_{t-1}^s.$$

   The values of $\overline{y}^f$ and $\overline{y}^s$ are the steady-state values *derived at the current draw of parameters.* In this respect, 202 (and 201) resemble 103 logic, as the constant in the beliefs changes every time the parameter values change. For the **SecondMoments** matrix, the adjustments is

   $$SM_{const} = \begin{bmatrix} 1 & \overline{y}^T \\ \overline{y} & \Sigma + \overline{y}\overline{y}^T \end{bmatrix},$$

   where Σ is the matrix of 2^{nd} moments derived using assumption of demeaned variables. This is a theoretically correct way of proceeding given than now our variables are expected to have non–zero mean.

   After adjusting the initial beliefs, Kalman filter step is performed. As now the beliefs are about the (potentially) non–zero mean variables, the filtering step is adjusted accordingly. DYNARE implicitly demeans filtered variables by substracting the trend in the measurement equation,

   $$v = Y(:,t) - a(mf) - trend(:,t).$$

This could be understood as follows. Starting from

$$\begin{aligned}\xi_t - \overline{\xi} &= T\left(\xi_{t-1} - \overline{\xi}\right) + R\epsilon_t \Rightarrow \ \xi_t = (I-T)\overline{\xi} + T\xi_{t-1} + R\epsilon_t, \\ y_t &= H\xi_t,\end{aligned}$$

we could move to the new variables $a_t = \xi_t - \overline{\xi}$:

$$\begin{aligned}a_t &= Ta_{t-1} + R\epsilon_t, \\ y_t &= H\left(a_t + \overline{\xi}\right) \Rightarrow y_t - H\overline{\xi} = Ha_t.\end{aligned}$$

This is what DYNARE is doing: by substracting means in the measurement equation, it demeans the whole set of variables (as only measured variables can have non–zero means; I am not sure how this argument will play out if some model variables have non–zero means, but this probably never happens within the estimation subroutine).

The new code cannot follow this route, as now the drift in the law of motion, $(I-T)\overline{\xi}$, is not necessarily constant. Therefore, I work directly with the representation

$$\begin{aligned}\xi_t &= \mu_t + T\xi_{t-1} + R\epsilon_t, \\ y_t &= H\xi_t.\end{aligned}$$

In particular, this means that variables **a** and **f_a, f_a_old** (used in `DiffuseLikolihoodL1_options_const.m` to construct a vector of variables used by the agents to update their beliefs) or **filteratt** (used in `DiffuseKalmanSmootherL1_options_const.m`) are initialized using the steady–state vector $\overline{y}$ (**SState**, appropriately reordered) and the corresponding VAR drift term $\overline{\mu} = (I-T)\overline{y}$. The constants are no longer substracted from the measurement equation.

The next important question is how to derive and then treat the (now time–varying) VAR drift term μ_t. μ_t is derived and then used in the law of motion for a :

$$a = mu + Ta + Kv.$$

I'm using the following way of deriving μ_t. Write

$$\begin{aligned}A^0\left(y_{t-1} - \overline{y}\right) + A^1\left(y_t - \overline{y}\right) + A^2\left(E_t y_{t+1} - \overline{y}\right) + B\epsilon_t &= 0,\ E_t y_{t+1} = \alpha + \beta y_t, \\ y_t &= \widetilde{\mu} + Ty_{t-1} + R\epsilon_t.\end{aligned}$$

Plugging everything into the first line, we get

$$\begin{aligned}\left[\begin{array}{c} A^0 y_{t-1} + \left(A^1 + A^2\beta\right)\left(\widetilde{\mu} + Ty_{t-1} + R\epsilon_t\right) + \\ +B\epsilon_t + A^2\alpha - \left(A^0 + A^1 + A^2\right)\overline{y} \end{array}\right] &= 0 \\ \left[\begin{array}{c} \left[A^0 + \left(A^1 + A^2\beta\right)T\right] y_{t-1} + \left[B + \left(A^1 + A^2\beta\right)\right]\epsilon_t + \\ + \left[\left(A^1 + A^2\beta\right)\widetilde{\mu} + A^2\alpha - \left(A^0 + A^1 + A^2\right)\overline{y}\right] \end{array}\right] &= 0.\end{aligned}$$

Collecting the terms at y_{t-1}, ϵ_t, and the constant and setting them equal to zero, we get

$$\begin{aligned} T &= -\left(A^1 + A^2\beta\right)^{-1} A^0, \\ R &= -\left(A^1 + A^2\beta\right)^{-1} B, \\ \widetilde{\mu} &= \left(A^1 + A^2\beta\right)^{-1} \left[\left(A^0 + A^1 + A^2\right)\overline{y} - A^2\alpha\right]. \end{aligned}$$

7. *Constants (for KF).* The only option which is implemented for KF is with constant. Basically, `BetaFromTR.m` computes matrix Σ (see SW, Section I) and the 'Mask' matrix which allows constructing a SURE-like data matrix X (again, see Section I). In `DiffuseLikelihoodKMAL1_options_const.m` and `DiffuseKalmanSmootherKMAL1_options_const.m` the 'Mask' is augmented for inclusion of the constant; then, Σ and the new 'Mask' are used to derive $\left(X^T\Sigma X\right)^{-1}$ needed to initialize the KF learning, see Section I of SW. The remainder of the KF for beliefs is standard; adjustment of the transmission mechanism and the constant given time-varying beliefs is as described in the previous point.

8. *Graphs.* All the graphs are prepared and plotted in `DiffuseKalmanSmoother*.m`, using the posterior mode parameter values. I plot evolution of beliefs, model weights (for KF), aggregate beliefs, and some volatility characteristics.

9. Note that with the code that allows for the constant, filtered and forecasted variables which are saved in the **oo_** structure are no longer demeaned! Therefore, derivation of beliefs for 101/201 should take care to distinguish between filtered variables generated by the new constant code and the standard one. If the filtered variables for 101 are generated using REE or constant–free adaptive learning estimation, everything remains as it is (in case kalman_algo $>$ 200, the program invokes `DiffuseLikolihoodL1_options_const.m` and `DiffuseKalmanSmootherL1_options_const.m`, otherwise `DiffuseLikolihoodL1_options.m` and `DiffuseKalmanSmootherL1_options.m` are used which continue substracting constants from the measurement equation). Again, for KF learning only the constant option is implemented in `DiffuseLikelihoodKMAL1_options_const.m` and `DiffuseKalmanSmootherKMAL1_options_const.m`

10. There are 3 similar blocks of code doing Kalman filtering in `DiffuseLikolihoodL1_options_const.m` and 2 in `DiffuseKalmanSmootherL1_options_const.m.` Only the second block in every program is changed: the first one is used only if diffuse priors for Kalman filter are used (we so far worked only with **lik_init**=2), and the third one is invoked once the matrix P becomes stationary. Under learning (whether constant gain or RLS if t isn't too large) the matrix P is always changing and thus this block isn't used as well. I didn't update the blocks that are not expected to run, and they might contain some older code. At some future stage, I will leave just one code block, probably.

11. In the output structure **oo_**, new fields are added.

    oo_.ForecastVariables stores one-step-ahead forecasts from Kalman filter $a_{t+1|t}$, which are called **oo_.FilteredVariables** in the original DYNARE. **oo_.FilteredVariables** contains filtered values $a_{t+1|t}$.

    The field **oo_.bet** is a cell array that contains the vector of beliefs for every forecasting model as a vector. For every forecasting equation, the first number if the constant, then the coefficients at RHS variables in alphabetical order follow. Block corresponding to different equations are ordered alphabetically in the LHS variable. The last cell of **oo_.bet** contains the aggregate beliefs, summed using time-varying (if BIC weighing is used) or fixed weights. This is a rectangular matrix, with zeros in places where a particular forecasting equation is not using a variable. If only one forecasting model is used, this field contains two cells with the same time-varying beliefs, ordered differently.

    oo_.TT, oo_.RR, oo_.mus, and **oo_.Q** contain the time-varying transmission mechanism under learning (the Actual Law of Motion).

    oo_.exps contains PLM expectations formed using aggregate model. If only one model is used, this equals the PLM expectations produced by this particular model. **oo_.errors** is a 3D array which contains PLM expectational errors from all individual models plus errors from the aggregate model.

    The only new field in **dr_** is **dr_.jacobia** which contains original DYNARE representation of the model as first-order expectational difference equation. This array is used extensively in the code to replace expectations with linear forecasting functions.