DSGE Models with Dynare++. A Tutorial.

Ondra Kameník

February 2011, updated August 2016

Contents

1 Setup **2**

2 Sample Session **2**

3 Sample Optimal Policy Session **6**

4 What Dynare++ Calculates **9**
4.1 Decision Rule Form 10
4.2 Taking Steps in Volatility Dimension 11
4.3 Simulating the Decision Rule 11
4.3.1 Simulations With Real-Time Statistics 11
4.3.2 Conditional Distributions 12
4.3.3 Random Numbers 12
4.4 Numerical Approximation Checks 13

5 Optimal Policy with Dynare++ **13**
5.1 First Order Conditions 14
5.2 Initial Guess for Deterministic Steady State 15
5.3 Optimal Ramsey Policy 16

6 Running Dynare++ **16**
6.1 Command Line Options 17
6.2 Dynare++ Model File 20
6.3 Incompatibilities with Matlab Dynare 21

7 Dynare++ Output **22**
7.1 Auxiliary Variables 22
7.2 MAT File 23
7.3 Journal File 25
7.4 Dump File 26
7.5 Matlab Scripts for Steady State Calculations 26
7.6 Custom Simulations 27

1 Setup

The Dynare++ setup procedure is pretty straightforward as Dynare++ is included in the Dynare installation packages which can be downloaded from http://www.dynare.org. Take the following steps:

1. Add the `dynare++` subdirectory of the root Dynare installation directory to the your operating system path. This ensures that your OS will find the `dynare++` executable.

2. If you have MATLAB and want to run custom simulations (see 7.6), then you need to add to your MATLAB path the `dynare++` subdirectory of the root Dynare installation directory, and also directory containing the `dynare_simul_` MEX file (note the trailing underscore). The easiest way to add the latter is to run Dynare once in your MATLAB session (even without giving it any MOD file).

2 Sample Session

As an example, let us take a simple DSGE model with time to build, whose dynamic equilibrium is described by the following first order conditions:

$$
\begin{aligned}
& c_t \theta h_t^{1+\psi} = (1-\alpha) y_t \\
& \beta E_t \left[\frac{\exp(b_t) c_t}{\exp(b_{t+1}) c_{t+1}} \left(\exp(b_{t+1}) \alpha \frac{y_{t+1}}{k_{t+1}} + 1 - \delta \right) \right] = 1 \\
& y_t = \exp(a_t) k_t^{\alpha} h_t^{1-\alpha} \\
& k_t = \exp(b_{t-1})(y_{t-1} - c_{t-1}) + (1-\delta) k_{t-1} \\
& a_t = \rho a_{t-1} + \tau b_{t-1} + \epsilon_t \\
& b_t = \tau a_{t-1} + \rho b_{t-1} + \nu_t
\end{aligned}
$$

The convention is that the timing of a variable reflects when this variable is decided. Dynare++ therefore uses a "stock at the end of the period" notation for predetermined state variables (see the Dynare manual for details).

The timing of this model is that the exogenous shocks ϵ_t, and ν_t are observed by agents in the beginning of period t and before the end of period t all endogenous variables with index t are decided. The expectation operator E_t works over the information accumulated just before the end of the period t (this includes ϵ_t, ν_t and all endogenous variables with index t).

The exogenous shocks ϵ_t and ν_t are supposed to be serially uncorrelated with zero means and time-invariant variance-covariance matrix. In Dynare++, these variables are called exogenous; all other variables are endogenous. Now we are prepared to start writing a model file for Dynare++, which is an ordinary text file and could be created with any text editor.

The model file starts with a preamble declaring endogenous and exogenous variables, parameters, and setting values of the parameters. Note that one can put expression on right hand sides. The preamble follows:

```
var Y, C, K, A, H, B;
varexo EPS, NU;

parameters beta, rho, alpha, delta, theta, psi, tau;
alpha = 0.36;
rho   = 0.95;
tau   = 0.025;
beta  = 1/(1.03^0.25);
delta = 0.025;
psi   = 0;
theta = 2.95;
```

The section setting values of the parameters is terminated by a beginning of the `model` section, which states all the dynamic equations. A timing convention of a Dynare++ model is the same as the timing of our example model, so we may proceed with writing the model equations. The time indexes of c_{t-1}, c_t, and c_{t+1} are written as `C(-1)`, `C`, and `C(1)` resp. The `model` section looks as follows:

```
model;
C*theta*H^(1+psi) = (1-alpha)*Y;
beta*exp(B)*C/exp(B(1))/C(1)*
  (exp(B(1))*alpha*Y(1)/K(1)+1-delta) = 1;
Y = exp(A)*K^alpha*H^(1-alpha);
K = exp(B(-1))*(Y(-1)-C(-1)) + (1-delta)*K(-1);
A = rho*A(-1) + tau*B(-1) + EPS;
B = tau*A(-1) + rho*B(-1) + NU;
end;
```

At this point, almost all information that Dynare++ needs has been provided. Only three things remain to be specified: initial values of endogenous variables for non-linear solver, variance-covariance matrix of the exogenous shocks and order of the Taylor approximation. Since the model is very simple, there is a closed form solution for the deterministic steady state. We use it as initial values for the non-linear solver. Note that the expressions on the right hand-sides in `initval` section can reference values previously calculated. The remaining portion of the model file looks as follows:

```
initval;
A = 0;
B = 0;
H = ((1-alpha)/(theta*(1-(delta*alpha)
     /(1/beta-1+delta))))^(1/(1+psi));
Y = (alpha/(1/beta-1+delta))^(alpha/(1-alpha))*H;
K = alpha/(1/beta-1+delta)*Y;
```

```
C = Y - delta*K;
end;

vcov = [
  0.0002  0.00005;
  0.00005 0.0001
];

order = 7;
```

Note that the order of rows/columns of the variance-covariance matrix corresponds to the ordering of exogenous variables in the `varexo` declaration. Since the `EPS` was declared first, its variance is 0.0002, and the variance of `NU` is 0.0001.

Let the model file be saved as `example1.mod`. Now we are prepared to solve the model. At the operating system command prompt[1] we issue a command:

```
dynare++ example1.mod
```

When the program is finished, it produces two output files: a journal file `example1.jnl` and a Matlab MAT-4 `example1.mat`. The journal file contains information about time, memory and processor resources needed for all steps of solution. The output file is more interesting. It contains various simulation results. It can be loaded into Matlab or Scilab and examined.[2] The following examples are done in Matlab, everything would be very similar in Scilab.

Let us first examine the contents of the MAT file:

```
>> load example1.mat
>> who

Your variables are:

dyn_g_1            dyn_i_Y            dyn_npred
dyn_g_2            dyn_irfm_EPS_mean  dyn_nstat
dyn_g_3            dyn_irfm_EPS_var   dyn_shocks
dyn_g_4            dyn_irfm_NU_mean   dyn_ss
dyn_g_5            dyn_irfm_NU_var    dyn_state_vars
dyn_i_A            dyn_irfp_EPS_mean  dyn_steady_states
dyn_i_B            dyn_irfp_EPS_var   dyn_vars
dyn_i_C            dyn_irfp_NU_mean   dyn_vcov
dyn_i_EPS          dyn_irfp_NU_var    dyn_vcov_exo
dyn_i_H            dyn_mean
dyn_i_K            dyn_nboth
dyn_i_NU           dyn_nforw
```

All the variables coming from one MAT file have a common prefix. In this case it is `dyn`, which is Dynare++ default. The prefix can be changed, so that the multiple results could be loaded into one Matlab session.

[1] Under Windows it is a `cmd` program, under Unix it is any shell

[2] For Matlab `load example1.mat`, for Scilab `mtlb_load example1.mat`

In the default setup, Dynare++ solves the Taylor approximation to the decision rule and calculates unconditional mean and covariance of the endogenous variables, and generates impulse response functions. The mean and covariance are stored in `dyn_mean` and `dyn_vcov`. The ordering of the endogenous variables is given by `dyn_vars`.

In our example, the ordering is

```
>> dyn_vars
dyn_vars =
H
A
Y
C
K
B
```

and unconditional mean and covariance are

```
>> dyn_mean
dyn_mean =
    0.2924
    0.0019
    1.0930
    0.8095
   11.2549
    0.0011
>> dyn_vcov
dyn_vcov =
    0.0003    0.0006    0.0016    0.0004    0.0060    0.0004
    0.0006    0.0024    0.0059    0.0026    0.0504    0.0012
    0.0016    0.0059    0.0155    0.0069    0.1438    0.0037
    0.0004    0.0026    0.0069    0.0040    0.0896    0.0016
    0.0060    0.0504    0.1438    0.0896    2.1209    0.0405
    0.0004    0.0012    0.0037    0.0016    0.0405    0.0014
```

The ordering of the variables is also given by indexes starting with `dyn_i_`. Thus the mean of capital can be retrieved as

```
>> dyn_mean(dyn_i_K)
ans =
   11.2549
```

and covariance of labor and capital by

```
>> dyn_vcov(dyn_i_K,dyn_i_H)
ans =
    0.0060
```

The impulse response functions are stored in matrices as follows

matrix	response to
`dyn_irfp_EPS_mean`	positive impulse to `EPS`
`dyn_irfm_EPS_mean`	negative impulse to `EPS`
`dyn_irfp_NU_mean`	positive impulse to `NU`
`dyn_irfm_NU_mean`	negative impulse to `NU`

All shocks sizes are one standard error. Rows of the matrices correspond to endogenous variables, columns correspond to periods. Thus capital response to a positive shock to `EPS` can be plotted as

```
plot(dyn_irfp_EPS_mean(dyn_i_K,:));
```

The data is in units of the respective variables, so in order to plot the capital response in percentage changes from the decision rule's fix point (which is a vector `dyn_ss`), one has to issue the commands:

```
Kss=dyn_ss(dyn_i_K);
plot(100*dyn_irfp_EPS_mean(dyn_i_K,:)/Kss);
```

The plotted impulse response shows that the model is pretty persistent and that the Dynare++ default for a number of simulated periods is not sufficient. In addition, the model persistence puts in doubt also a number of simulations. The Dynare++ defaults can be changed when calling Dynare++, in operating system's command prompt, we issue a command:

```
dynare++ --per 300 --sim 150 example1.mod
```

This sets the number of simulations to 150 and the number of periods to 300 for each simulation giving 45000 total simulated periods.

3 Sample Optimal Policy Session

Suppose that one wants to solve the following optimal policy problem with timeless perspective.[3] The following optimization problem is how to choose capital taxes financing public good to maximize agent's utility from consumption good and public good. The problem takes the form:

$$\max_{\{\tau_t\}_{t_0}^{\infty}} E_{t_0} \sum_{t=t_0}^{\infty} \beta^{t-t_0} \left(u(c_t) + av(g_t)\right)$$

subject to

$$\begin{aligned} u'(c_t) &= \beta E_t \left[u'(c_{t+1}) \left(1 - \delta + f'(k_{t+1})(1 - \alpha\tau_{t+1})\right)\right] \\ K_t &= (1-\delta)K_{t-1} + (f(K_{t-1}) - c_{t-1} - g_{t-1}) \\ g_t &= \tau_t \alpha f(K_t), \\ \text{where } t &= \ldots, t_0 - 1, t_0, t_0 + 1, \ldots \end{aligned}$$

[3] See 5.3 on how to solve Ramsey optimality problem within this framework

$u(c_t)$ is utility from consuming the consumption good, $v(g_t)$ is utility from consuming the public good, $f(K_t)$ is a production function $f(K_t) = Z_t K_t^{\alpha}$. Z_t is a technology shock modeled as AR(1) process. The three constraints come from the first order conditions of a representative agent. We suppose that it pursues a different objective, namely lifetime utility involving only consumption c_t. The representative agents chooses between consumption and investment. It rents the capital to firms and supplies constant amount of labour. All output is paid back to consumer in form of wage and capital rent. Only the latter is taxed. We suppose that the optimal choice has been taking place from infinite past and will be taking place for ever. Further we suppose the same about the constraints.

Let us choose the following functional forms:

$$\begin{aligned} u(c_t) &= \frac{c_t^{1-\eta}}{1-\eta} \\ v(g_t) &= \frac{g_t^{1-\phi}}{1-\phi} \\ f(K_t) &= K_t^{\alpha} \end{aligned}$$

Then the problem can be coded into Dynare++ as follows. We start with a preamble which states all the variables, shocks and parameters:

```
var C G K TAU Z;

varexo EPS;

parameters eta beta alpha delta phi a rho;

eta = 2;
beta = 0.99;
alpha = 0.3;
delta = 0.10;
phi = 2.5;
a = 0.1;
rho = 0.7;
```

Then we specify the planner's objective and the discount factor in the objective. The objective is an expression (possibly including also variable leads and lags), and the discount factor must be one single declared parameter:

```
planner_objective C^(1-eta)/(1-eta) + a*G^(1-phi)/(1-phi);

planner_discount beta;
```

The model section will contain only the constraints of the social planner. These are capital accumulation, identity for the public product, AR(1) process for Z_t and the first order condition of the representative agent (with different objective).

```
model;
K = (1-delta)*K(-1) + (exp(Z(-1))*K(-1)^alpha - C(-1) - G(-1));
G = TAU*alpha*K^alpha;
Z = rho*Z(-1) + EPS;
C^(-eta) = beta*C(+1)^(-eta)*(1-delta +
           exp(Z(+1))*alpha*K(+1)^(alpha-1)*(1-alpha*TAU(+1)));
end;
```

Now we have to provide a good guess for non-linear solver calculating the deterministic steady state. The model's steady state has a closed form solution if the taxes are known. So we provide a guess for taxation `TAU` and then use the closed form solution for capital, public good and consumption:[4]

```
initval;
TAU = 0.70;
K = ((delta+1/beta-1)/(alpha*(1-alpha*TAU)))^(1/(alpha-1));
G = TAU*alpha*K^alpha;
C =  K^alpha - delta*K - G;
Z = 0;
```

Finally, we have to provide the order of approximation, and the variance-covariance matrix of the shocks (in our case we have only one shock):

```
order = 4;

vcov = [
0.01
];
```

After this model file has been run, we can load the resulting MAT-file into the Matlab (or Scilab) and examine its contents:

```
>> load kp1980_2.mat
>> who

Your variables are:

dyn_g_1            dyn_i_MULT1        dyn_nforw
dyn_g_2            dyn_i_MULT2        dyn_npred
dyn_g_3            dyn_i_MULT3        dyn_nstat
dyn_g_4            dyn_i_TAU          dyn_shocks
dyn_i_AUX_3_0_1    dyn_i_Z            dyn_ss
dyn_i_AUX_4_0_1    dyn_irfm_EPS_mean  dyn_state_vars
dyn_i_C            dyn_irfm_EPS_var   dyn_steady_states
dyn_i_EPS          dyn_irfp_EPS_mean  dyn_vars
dyn_i_G            dyn_irfp_EPS_var   dyn_vcov
dyn_i_K            dyn_mean           dyn_vcov_exo
dyn_i_MULT0        dyn_nboth
```

[4] Initial guess for Lagrange multipliers and some auxiliary variables is calculated automatically. See 5.2 for more details.

The data dumped into the MAT-file have the same structure as in the previous example of this tutorial. The only difference is that Dynare++ added a few more variables. Indeed:

```
>> dyn_vars
dyn_vars =
MULT1
G
MULT3
C
K
Z
TAU
AUX_3_0_1
AUX_4_0_1
MULT0
MULT2
```

Besides the five variables declared in the model (`C`, `G`, `K`, `TAU`, and `Z`), Dynare++ added 6 more, four as Lagrange multipliers of the four constraints, two as auxiliary variables for shifting in time. See 7.1 for more details.

The structure and the logic of the MAT-file is the same as these new 6 variables were declared in the model file and the file is examined in the same way.

For instance, let us examine the Lagrange multiplier of the optimal policy associated with the consumption first order condition. Recall that the consumers' objective is different from the policy objective. Therefore, the constraint will be binding and the multiplier will be non-zero. Indeed, its deterministic steady state, fix point and mean are as follows:

```
>> dyn_steady_states(dyn_i_MULT3,1)
ans =
   -1.3400
>> dyn_ss(dyn_i_MULT3)
ans =
   -1.3035
>> dyn_mean(dyn_i_MULT3)
ans =
   -1.3422
```

4 What Dynare++ Calculates

Dynare++ solves first order conditions of a DSGE model in the recursive form:

$$E_t[f(y^{**}_{t+1}, y_t, y^*_{t-1}, u_t)] = 0, \tag{1}$$

where y is a vector of endogenous variables, and u a vector of exogenous variables. Some of elements of y can occur at time $t+1$, these are y^{**}. Elements of y occurring at time $t-1$ are denoted y^*. The exogenous shocks are supposed to be serially independent and normally distributed $u_t \sim N(0, \Sigma)$.

The solution of this dynamic system is a decision rule

$$y_t = g(y^*_{t-1}, u_t)$$

Dynare++ calculates a Taylor approximation of this decision rule of a given order. The approximation takes into account deterministic effects of future volatility, so a point about which the Taylor approximation is done will be different from the fix point y of the rule yielding $y = g(y^*, 0)$.

The fix point of a rule corresponding to a model with $\Sigma = 0$ is called *deterministic steady state* denoted as $\bar{y}$. In contrast to deterministic steady state, there is no consensus in literature how to call a fix point of the rule corresponding to a model with non-zero Σ. I am tempted to call it *stochastic steady state*, however, it might be confused with unconditional mean or with steady distribution. So I will use a term *fix point* to avoid a confusion.

By default, Dynare++ solves the Taylor approximation about the deterministic steady state. Alternatively, Dynare++ can split the uncertainty to a few steps and take smaller steps when calculating the fix points. This is controlled by an option `--steps`. For the brief description of the second method, see 4.2.

4.1 Decision Rule Form

In case of default solution algorithm (approximation about the deterministic steady state $\bar{y}$), Dynare++ calculates the higher order derivatives of the equilibrium rule to get a decision rule of the following form. In Einstein notation, it is:

$$y_t - \bar{y} = \sum_{i=0}^{k} \frac{1}{i!} \left[g_{(y^* u)^i}\right]_{\alpha_1 \dots \alpha_i} \prod_{j=1}^{i} \begin{bmatrix} y^*_{t-1} - \bar{y}^* \\ u_t \end{bmatrix}^{\alpha_j}$$

Note that the ergodic mean will be different from the deterministic steady state $\bar{y}$ and thus deviations $y^*_{t-1} - \bar{y}^*$ will not be zero in average. This implies that in average we will commit larger round off errors than if we used the decision rule expressed in deviations from a point closer to the ergodic mean. Therefore, by default, Dynare++ recalculates this rule and expresses it in deviations from the stochastic fix point y.

$$y_t - y = \sum_{i=1}^{k} \frac{1}{i!} \left[\tilde{g}_{(y^* u)^i}\right]_{\alpha_1 \dots \alpha_i} \prod_{j=1}^{i} \begin{bmatrix} y^*_{t-1} - y^* \\ u_t \end{bmatrix}^{\alpha_j}$$

Note that since the rule is centralized around its fix point, the first term (for $i = 0$) drops out.

Also note, that this rule mathematically equivalent to the rule expressed in deviations from the deterministic steady state, and still it is an approximation about the deterministic steady state. The fact that it is expressed in deviations from a different point should not be confused with the algorithm in 4.2.

This centralization can be avoided by invoking `--no-centralize` command line option.

4.2 Taking Steps in Volatility Dimension

For models, where volatility of the exogenous shocks plays a big role, the approximation about deterministic steady state can be poor, since the equilibrium dynamics can be very different from the dynamics in the vicinity of the perfect foresight (deterministic steady state).

Therefore, Dynare++ has on option `--steps` triggering a multistep algorithm. The algorithm splits the volatility to a given number of steps. Dynare++ attempts to calculate approximations about fix points corresponding to these levels of volatility. The problem is that if we want to calculate higher order approximations about fix points corresponding to volatilities different from zero (as in the case of deterministic steady state), then the derivatives of lower orders depend on derivatives of higher orders with respect to forward looking variables. The multistep algorithm in each step approximates the missing higher order derivatives with extrapolations based on the previous step.

In this way, the approximation of the stochastic fix point and the derivatives about this fix point are obtained. It is difficult to a priori decide whether this algorithm yields a better decision rule. Nothing is guaranteed, and the resulted decision rule should be checked with a numerical integration. See 4.4.

4.3 Simulating the Decision Rule

After some form of a decision rule is calculated, it is simulated to obtain draws from ergodic (unconditional) distribution of endogenous variables. The mean and the covariance are reported. There are two ways how to calculate the mean and the covariance. The first one is to store all simulated samples and calculate the sample mean and covariance. The second one is to calculate mean and the covariance in the real-time not storing the simulated sample. The latter case is described below (see 4.3.1).

The stored simulated samples are then used for impulse response function calculations. For each shock, the realized shocks in these simulated samples (control simulations) are taken and an impulse is added and the new realization of shocks is simulated. Then the control simulation is subtracted from the simulation with the impulse. This is done for all control simulations and the results are averaged. As the result, we get an expectation of difference between paths with impulse and without impulse. In addition, the sample variances are reported. They might be useful for confidence interval calculations.

For each shock, Dynare++ calculates IRF for two impulses, positive and negative. Size of an impulse is one standard error of a respective shock.

The rest of this subsection is divided to three parts giving account on real-time simulations, conditional simulations, and on the way how random numbers are generated resp.

4.3.1 Simulations With Real-Time Statistics

When one needs to simulate large samples to get a good estimate of unconditional mean, simulating the decision rule with statistics calculated in real-time

comes handy. The main reason is that the storing of all simulated samples may not fit into the available memory.

The real-time statistics proceed as follows: We model the ergodic distribution as having normal distribution $y \sim N(\mu, \Sigma)$. Further, the parameters μ and Σ are modelled as:

$$\begin{aligned} \Sigma &\sim \text{InvWishart}_\nu(\Lambda) \\ \mu|\Sigma &\sim N(\bar{\mu}, \Sigma/\kappa) \end{aligned}$$

This model of $p(\mu, \Sigma)$ has an advantage of conjugacy, i.e. a prior distribution has the same form as posterior. This property is used in the calculation of real-time estimates of μ and Σ, since it suffices to maintain only the parameters of $p(\mu, \Sigma)$ conditional observed draws so far. The parameters are: ν, Λ, κ, and $\bar{\mu}$.

The mean of $\mu, \Sigma|Y$, where Y are all the draws (simulated periods) is reported.

4.3.2 Conditional Distributions

Starting with version 1.3.6, Dynare++ calculates variable distributions y_t conditional on $y_0 = \bar{y}$, where $\bar{y}$ is the deterministic steady state. If triggered, Dynare++ simulates a given number of samples with a given number of periods all starting at the deterministic steady state. Then for each time t, mean $E[y_t|y_0 = \bar{y}]$ and variances $E[(y_t - E[y_t|y_0 = \bar{y}])(y_t - E[y_t|y_0 = \bar{y}])^T|y_0 = \bar{y}]$ are reported.

4.3.3 Random Numbers

For generating of the pseudo random numbers, Dynare++ uses Mersenne twister by Makoto Matsumoto and Takuji Nishimura. Because of the parallel nature of Dynare++ simulations, each simulated sample gets its own instance of the twister. Each such instance is seeded before the simulations are started. This is to prevent additional randomness implied by the operating system's thread scheduler to interfere with the pseudo random numbers.

For seeding the individual instances of the Mersenne twister assigned to each simulated sample the system (C library) random generator is used. These random generators do not have usually very good properties, but we use them only to seed the Mersenne twister instances. The user can set the initial seed of the system random generator and in this way deterministically choose the seeds of all instances of the Mersenne twister.

In this way, it is guaranteed that two runs of Dynare++ with the same seed will yield the same results regardless the operating system's scheduler. The only difference may be caused by a different round-off errors committed when the same set of samples are summed in the different order (due to the operating system's scheduler).

4.4 Numerical Approximation Checks

Optionally, Dynare++ can run three kinds of checks for Taylor approximation errors. All three methods numerically calculate the residual of the DSGE equations

$$E[f(g^{**}(g^*(y^*,u),u'),g(y^*,u),y^*,u)|y^*,u]$$

which must be ideally zero for all y^* and u. This integral is evaluated by either product or Smolyak rule applied to one dimensional Gauss–Hermite quadrature. The user does not need to care about the decision. An algorithm yielding higher quadrature level and less number of evaluations less than a user given maximum is selected.

The three methods differ only by a set of y^* and u where the residuals are evaluated. These are:

- The first method calculates the residuals along the shocks for fixed y^* equal to the fix point. We let all elements of u be fixed at 0 but one element, which varies from $-\mu\sigma$ to $\mu\sigma$, where σ is a standard error of the element and μ is the user given multiplier. In this way we can see how the approximation error grows if the fix point is disturbed by a shock of varying size.

- The second method calculates the residuals along a simulation path. A random simulation is run, and at each point the residuals are reported.

- The third method calculates the errors on an ellipse of the state variables y^*. The shocks u are always zero. The ellipse is defined as

  $$\{Ax|\ \|x\|_2 = \mu\},$$

  where μ is a user given multiplier, and $AA^T = V$ for V being a covariance of endogenous variables based on the first order approximation. The method calculates the residuals at low discrepancy sequence of points on the ellipse. Both the residuals and the points are reported.

5 Optimal Policy with Dynare++

Starting with version 1.3.2, Dynare++ is able to automatically generate and then solve the first order conditions for a given objective and (possibly) forward looking constraints. Since the constraints can be forward looking, the use of this feature will mainly be in optimal policy or control.

The only extra thing which needs to be added to the model file is a specification of the policy's objective. This is done by two keywords, placed not before parameter settings. If the objective is to maximize

$$E_{t_0}\sum_{t=t_0}^{\infty}\beta^{t-t_0}\left[\frac{c_t^{1-\eta}}{1-\eta}+a\frac{g_t^{1-\phi}}{1-\phi}\right],$$

then the keywords will be:

```
planner_objective C^(1-eta)/(1-eta) + a*G^(1-phi)/(1-phi);

planner_discount beta;
```

Dynare++ parses the file and if the two keywords are present, it automatically derives the first order conditions for the problem. The first order conditions are put to the form (1) and solved. In this case, the equations in the `model` section are understood as the constraints (they might come as the first order conditions from optimizations of other agents) and their number must be less than the number of endogenous variables.

This section further describes how the optimal policy first order conditions look like, then discusses some issues with the initial guess for deterministic steady state, and finally describes how to simulate Ramsey policy within this framework.

5.1 First Order Conditions

Mathematically, the optimization problem looks as follows:

$$\begin{aligned} \max_{\{y_\tau\}_t^\infty} & E_t \left[\sum_{\tau=t}^{\infty} \beta^{\tau-t} b(y_{\tau-1}, y_\tau, y_{\tau+1}, u_\tau) \right] \\ \text{s.t.} & \\ & E_\tau^I \left[f(y_{\tau-1}, y_\tau, y_{\tau+1}, u_\tau) \right] = 0 \quad \text{for } \tau = \ldots, \mathrm{t}-1, \mathrm{t}, \mathrm{t}+1, \ldots \end{aligned} \tag{2}$$

where E^I is an expectation operator over an information set including, besides all the past, all future realizations of policy's control variables and distributions of future shocks $u_t \sim N(0, \Sigma)$. The expectation operator E integrates over an information including only distributions of u_t (besides the past).

Note that the constraints f take place at all times, and they are conditioned at the running τ since the policy knows that the agents at time τ will use all the information available at τ.

The maximization problem can be rewritten using Lagrange multipliers as:

$$\begin{aligned} \max_{y_t} E_t \Bigg[& \sum_{\tau=t}^{\infty} \beta^{\tau-t} b(y_{\tau-1}, y_\tau, y_{\tau+1}, u_\tau) \\ & + \sum_{\tau=-\infty}^{\infty} \beta^{\tau-t} \lambda_\tau^T E_\tau^I \left[f(y_{\tau-1}, y_\tau, y_{\tau+1}, u_\tau) \right] \Bigg], \end{aligned} \tag{3}$$

where λ_t is a column vector of Lagrange multipliers.

After some manipulations with compounded expectations over different in-

formation sets, one gets the following first order conditions:

$$E_t \Bigg[\frac{\partial}{\partial y_t} b(y_{t-1}, y_t, y_{t+1}, u_t) + \beta L^{+1} \frac{\partial}{\partial y_{t-1}} b(y_{t-1}, y_t, y_{t+1}, u_t) + \beta^{-1} \lambda_{t-1}^T L^{-1} \frac{\partial}{\partial y_{t+1}} f(y_{t-1}, y_t, y_{t+1}, u_t) + \lambda_t^T \frac{\partial}{\partial y_t} f(y_{t-1}, y_t, y_{t+1}, u_t) + \beta \lambda_{t+1}^T E_{t+1} \left[L^{+1} \frac{\partial}{\partial y_{t-1}} f(y_{t-1}, y_t, y_{t+1}, u_t) \right] \Bigg] = 0, \tag{4}$$

where L^{+1} is one period lead operator, and L^{-1} is one period lag operator.

Dynare++ takes input corresponding to (2), introduces the Lagrange multipliers according to (3), and using its symbolic derivator it compiles (4). The system (4) with the constraints from (3) is then solved in the same way as the normal input (1).

5.2 Initial Guess for Deterministic Steady State

Solving deterministic steady state of non-linear dynamic systems is not trivial and the first order conditions for optimal policy add significant complexity. The `initval` section allows to input the initial guess of the non-linear solver. It requires that all user declared endogenous variables be initialized. However, in most cases, we have no idea what are good initial guesses for the Lagrange multipliers.

For this reason, Dynare++ calculates an initial guess of Lagrange multipliers using user provided initial guesses of all other endogenous variables. It uses the linearity of the Lagrange multipliers in the (4). In its static form, (4) looks as follows:

$$\frac{\partial}{\partial y_t} b(y, y, y, 0) + \beta \frac{\partial}{\partial y_{t-1}} b(y, y, y, 0) + \lambda^T \left[\beta^{-1} \frac{\partial}{\partial y_{t+1}} f(y, y, y, 0) + \frac{\partial}{\partial y_t} f(y, y, y, 0) + \beta \frac{\partial}{\partial y_{t-1}} f(y, y, y, 0) \right] = 0 \tag{5}$$

The user is required to provide an initial guess of all declared variables (all y). Then (5) becomes an overdetermined linear system in λ, which is solved by means of the least squares. The closer the initial guess of y is to the exact solution, the closer are the Lagrange multipliers λ.

The calculated Lagrange multipliers by the least squares are not used, if they are set in the `initval` section. In other words, if a multiplier has been given a value in the `initval` section, then the value is used, otherwise the calculated value is taken.

For even more difficult problems, Dynare++ generates two Matlab files calculating a residual of the static system and its derivative. These can be used in

Matlab's `fsolve` or other algorithm to get an exact solution of the deterministic steady state. See 7.5 for more details.

Finally, Dynare++ might generate a few auxiliary variables. These are simple transformations of other variables. They are initialized automatically and the user usually does not need to care about it.

5.3 Optimal Ramsey Policy

Dynare++ solves the optimal policy problem with timeless perspective. This means that it assumes that the constraints in (2) are valid from the infinite past to infinite future. Dynare++ calculation of ergodic distribution then assumes that the policy has been taking place from infinite past.

If some constraints in (2) are forward looking, this will result in some backward looking Lagrange multipliers. Such multipliers imply possibly time inconsistent policy in the states of the "original" economy, since these backward looking multipliers add new states to the "optimized" economy. In this respect, the timeless perspective means that there is no fixed initial distribution of such multipliers, instead, their ergodic distribution is taken.

In contrast, Ramsey optimal policy is started at $t = 0$. This means that the first order conditions at $t = 0$ are different than the first order conditions at $t \geq 1$, which are (4). However, it is not difficult to assert that the first order conditions at $t = 0$ are in the form of (4) if all the backward looking Lagrange multipliers are set to zeros at period -1, i.e. $\lambda_{-1} = 0$.

All in all, the solution of (4) calculated by Dynare++ can be used as a Ramsey optimal policy solution provided that all the backward looking Lagrange multipliers were set to zeros prior to the first simulation period. This can be done by setting the initial state of a simulation path in `dynare_simul.m`. If this is applied on the example from 3, then we may do the following in the command prompt:

```
>> load kp1980_2.mat
>> shocks = zeros(1,100);
>> ystart = dyn_ss;
>> ystart(dyn_i_MULT3) = 0;
>> r=dynare_simul('kp1980_2.mat',shocks,ystart);
```

This will simulate the economy if the policy was introduced in the beginning and no shocks happened.

More information on custom simulations can be obtained by typing:

```
help dynare_simul
```

6 Running Dynare++

This section deals with Dynare++ input. The first subsection 6.1 provides a list of command line options, next subsection 6.2 deals with a format of Dynare++ model file, and the last subsection discusses incompatibilities between Dynare Matlab and Dynare++.

6.1 Command Line Options

The calling syntax of the Dynare++ is

```
dynare++ [--help] [--version] [options] <model file>
```

where the model file must be given as the last token and must include its extension. The model file may include path, in this case, the path is taken relative to the current directory. Note that the current directory can be different from the location of `dynare++` binary.

The options are as follows:

`--help` This prints a help message and exits.

`--version` This prints a version information and exits.

`--per` *num* This sets a number of simulated periods to *num* in addition to the burn-in periods. This number is used when calculating unconditional mean and covariance and for IRFs. Default is 100.

`--burn` *num* This sets a number of initial periods which should be ignored from the statistics. The burn-in periods are used to eliminate the influence of the starting point when calculating ergodic distributions or/and impulse response functions. The number of simulated period given by `--per` *num* option does not include the number of burn-in periods. Default is 0.

`--sim` *num* This sets a number of stochastic simulations. This number is used when calculating unconditional mean and covariance and for IRFs. The total sample size for unconditional mean and covariance is the number of periods times the number of successful simulations. Note that if a simulation results in `NaN` or `+-Inf`, then it is thrown away and is not considered for the mean nor the variance. The same is valid for IRF. Default is 80.

`--rtsim` *num* This sets a number of stochastic simulations whose statistics are calculated in the real-time. This number excludes the burn-in periods set by `--burn` *num* option. See 4.3.1 for more details. Default is 0, no simulations.

`--rtper` *num* This sets a number of simulated periods per one simulation with real-time statistics to *num*. See 4.3.1 for more details. Default is 0, no simulations.

`--condsim` *num* This sets a number of stochastic conditional simulations. See 4.3.2 for more details. Default is 0, no simulations.

`--condper` *num* This sets a number of simulated periods per one conditional simulation. See 4.3.2 for more details. Default is 0, no simulations.

`--steps` *num* If the number *num* is greater than 0, this option invokes a multi-step algorithm (see section 4), which in the given number of steps calculates fix points and approximations of the decision rule for increasing uncertainty. Default is 0, which invokes a standard algorithm for approximation about deterministic steady state. For more details, see 4.2.

`--centralize` This option causes that the resulting decision rule is centralized about (in other words: expressed in the deviations from) the stochastic fix point. The centralized decision rule is mathematically equivalent but has an advantage of yielding less numerical errors in average than not centralized decision rule. By default, the rule is centralized. For more details, see 4.1.

`--no-centralize` This option causes that the resulting decision rule is not centralized about (in other words: expressed in the deviations from) the stochastic fix point. By default, the rule is centralized. For more details, see 4.1.

This option has no effect if the number of steps given by `--steps` is greater than 0. In this case, the rule is always centralized.

`--prefix` *string* This sets a common prefix of variables in the output MAT file. Default is `dyn`.

`--seed` *num* This sets an initial seed for the random generator providing seed to generators for each sample. See 4.3.3 for more details. Default is 934098.

`--order` *num* This sets the order of approximation and overrides the `order` statement in the model file. There is no default.

`--threads` *num* This sets a number of parallel threads. Complex evaluations of Faa Di Bruno formulas, simulations and numerical integration can be parallelized, Dynare++ exploits this advantage. You have to have a hardware support for this, otherwise there is no gain from the parallelization. As a rule of thumb, set the number of threads to the number of processors. An exception is a machine with Pentium 4 with Hyper Threading (abbreviated by HT). This processor can run two threads concurrently. The same applies to Dual-Core processors. Since these processors are present in most new PC desktops/laptops, the default is 2.

`--ss-tol` *float* This sets the tolerance of the non-linear solver of deterministic steady state to *float*. It is in $\|\cdot\|_\infty$ norm, i.e. the algorithm is considered as converged when a maximum absolute residual is less than the tolerance. Default is 10^{-13}.

`--check` *pPeEsS* This selects types of residual checking to be performed. See section 4.4 for details. The string consisting of the letters

"pPeEsS" governs the selection. The upper-case letters switch a check on, the lower-case letters off. "P" stands for checking along a simulation path, "E" stands for checking on ellipse, and finally "S" stands for checking along the shocks. It is possible to choose more than one type of check. The default behavior is that no checking is performed.

`--check-evals` *num* This sets a maximum number of evaluations per one residual. The actual value depends on the selected algorithm for the integral evaluation. The algorithm can be either product or Smolyak quadrature and is chosen so that the actual number of evaluations would be minimal with maximal level of quadrature. Default is 1000.

`--check-num` *num* This sets a number of checked points in a residual check. One input value *num* is used for all three types of checks in the following way:

- For checks along the simulation, the number of simulated periods is $10 \cdot num$
- For checks on ellipse, the number of points on ellipse is $10 \cdot num$
- For checks along the shocks, the number of checked points corresponding to shocks from 0 to $\mu\sigma$ (see 4.4) is num.

Default is 10.

`--check-scale` *float* This sets the scaling factor μ for checking on ellipse to $0.5 \cdot float$ and scaling factor μ for checking along shocks to $float$. See section 4.4. Default is 2.0.

`--no-irfs` This suppresses IRF calculations. Default is to calculate IRFs for all shocks.

`--irfs` This triggers IRF calculations. If there are no shock names following the `--irfs` option, then IRFs for all shocks are calculated, otherwise see below. Default is to calculate IRFs for all shocks.

`--irfs` *shocklist* This triggers IRF calculations only for the listed shocks. The *shocklist* is a space separated list of exogenous variables for which the IRFs will be calculated. Default is to calculate IRFs for all shocks.

The following are a few examples:

```
dynare++ --sim 300 --per 50 blah.mod
dynare++ --check PE --check-num 15 --check-evals 500 blah.dyn
dynare++ --steps 5 --check S --check-scale 3 blahblah.mod
```

The first one sets the number of periods for IRF to 50, and sets a sample size for unconditional mean and covariance calculations to 6000. The second one checks the decision rule along a simulation path having 150 periods and on

ellipse at 150 points performing at most 500 evaluations per one residual. The third one solves the model in five steps and checks the rule along all the shocks from -3σ to 3σ in $2 * 10 + 1$ steps (10 for negative, 10 for positive and 1 for at zero).

6.2 Dynare++ Model File

In its strictest form, Dynare++ solves the following mathematical problem:

$$E_t[f(y^{**}_{t+1}, y_t, y^*_{t-1}, u_t)] = 0 \tag{6}$$

This problem is input either directly, or it is an output of Dynare++ routines calculating first order conditions of the optimal policy problem. In either case, Dynare++ performs necessary and mathematically correct substitutions to put the user specified problem to the (6) form, which goes to Dynare++ solver. The following discusses a few timing issues:

- Endogenous variables can occur, starting from version 1.3.4, at times after $t+1$. If so, an equation containing such occurrence is broken to non-linear parts, and new equations and new auxiliary variables are automatically generated only for the non-linear terms containing the occurrence. Note that shifting such terms to time $t+1$ may add occurrences of some other variables (involved in the terms) at times before $t-1$ implying addition of auxiliary variables to bring those variables to $t-1$.
- Variables declared as shocks may occur also at arbitrary times. If before t, additional endogenous variables are used to bring them to time t. If after t, then similar method is used as for endogenous variables occurring after $t+1$.
- There is no constraint on variables occurring at both times $t+1$ (or later) and $t-1$ (or earlier). Virtually, all variables can occur at arbitrary times.
- Endogenous variables can occur at times before $t-1$. If so, additional endogenous variables are added for all lags between the variable and $t-1$.
- Dynare++ applies the operator E_t to all occurrences at time $t+1$. The realization of u_t is included in the information set of E_t. See an explanation of Dynare++ timing on page 2.

The model equations are formulated in the same way as in Matlab Dynare. The time indexes different from t are put to round parenthesis in this way: `C(-1)`, `C`, `C(+1)`.

The mathematical expressions can use the following functions and operators:

- binary `+ - * / ^`
- unary plus and minus minus as in `a = -3;` and `a = +3;` resp.

- unary mathematical functions: `log exp sin cos tan sqrt`, where the logarithm has a natural base
- symbolic differentiation operator `diff(expr,symbol)`, where `expr` is a mathematical expression and `symbol` is a unary symbol (a variable or a parameter); for example `diff(A*K(-1)^alpha*L^(1-alpha),K(-1))` is internally expanded as `A*alpha*K(-1)^(alpha-1)*L^(1-alpha)`
- unary error function and complementary error function: `erf` and `erfc` defined as

$$\begin{aligned} erf(x) &= \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t \\ erfc(x) &= \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} \mathrm{d}t \end{aligned}$$

The model file can contain user comments. Their usage can be understood from the following piece of the model file:

```
P*C^(-gamma) = // line continues until semicolon
  beta*C(+1)^(-gamma)*(P(+1)+Y(+1)); // asset price
// choose dividend process: (un)comment what you want
Y/Y_SS = (Y(-1)/Y_SS)^rho*exp(EPS);
/*
Y-Y_SS = rho*(Y(-1)-Y_SS)+EPS;
*/
```

6.3 Incompatibilities with Matlab Dynare

This section provides a list of incompatibilities between a model file for Dynare++ and Matlab Dynare. These must be considered when a model file for Matlab Dynare is being migrated to Dynare++. The list is the following:

- There is no `periods` keyword.
- The parameters cannot be lagged or leaded, I think that Dynare Matlab allows it, but the semantics is the same (parameter is a constant).
- There are no commands like `steady`, `check`, `simul`, `stoch_simul`, etc.
- There are no sections like `estimated_params`, `var_obs`, etc.
- The variance-covariance matrix of endogenous shocks is given by `vcov` matrix in Dynare++. An example follows. Starting from version 1.3.5, it is possible for vcov to be positive semi-definite matrix.

```
vcov = [
0.05 0 0 0;
0 0.025 0 0;
0 0 0.05 0;
0 0 0 0.025
];
```

7 Dynare++ Output

There are three output files; a data file in MAT-4 format containing the output data (7.2), a journal text file containing an information about the Dynare++ run (7.3), and a dump file (7.4). Further, Dynare++ generates two Matlab script files, which calculate a residual and the first derivative of the residual of the static system (7.5). These are useful when calculating the deterministic steady state outside Dynare++.

Note that all output files are created in the current directory of the Dynare++ process. This can be different from the directory where the Dynare++ binary is located and different from the directory where the model file is located.

Before all, we need to understand what variables are automatically generated in Dynare++.

7.1 Auxiliary Variables

Besides the endogenous variables declared in `var` section, Dynare++ might automatically add the following endogenous variables:

`MULT`n	A Lagrange multiplier of the optimal policy problem associated with a constraint number n starting from zero.
`AUX_`*n1_n2_n3*	An auxiliary variable associated with the last term in equation (4). Since the term is under E_{t+k}, we need the auxiliary variable be put back in time. *n1* is a variable number starting from 0 in the declared order with respect to which the term was differentiated, *n2* is a number of constraint starting from 0, and finally *n3* is k (time shift of the term).
endovar`_p`K	An auxiliary variable for bringing an endogenous variable *endovar* back in time by K periods. The semantics of this variables is *endovar*`_p`K `=` *endovar*`(+`K`)`.
endovar`_m`K	An auxiliary variable for bringing an endogenous variable *endovar* forward in time by K periods. The semantics of this variables is *endovar*`_m`K `=` *endovar*`(-`K`)`.
exovar`_e`	An auxiliary endogenous variable made equal to the exogenous variable to allow for a semantical occurrence of the exogenous variable at time other than t. The semantics of this variables is *exovar*`_e =` *exovar*.

`AUXLD_n1_n2_n3` An auxiliary variable for bringing a non-linear term containing an occurrence of a variable after $t+1$ to time $t+1$. *n1* is an equation number starting from 0, *n2* is the non-linear sub-term number in the equation starting from 0. *n3* is a time shift. For example, if the first equation is the following:

```
X - Y*W(+1) + W(+2)*Z(+4) = 0;
```

then it will be expanded as:

```
X - Y*W(+1) + AUXLD_0_2_3(+1) = 0;
AUXLD_0_2_1 = W(-1)*Z(+1);
AUXLD_0_2_2 = AUXLD_0_2_1(+1);
AUXLD_0_2_3 = AUXLD_0_2_2(+1);
```

7.2 MAT File

The contents of the data file is depicted below. We assume that the prefix is `dyn`.

`dyn_nstat`	Scalar. A number of static variables (those occurring only at time t).
`dyn_npred`	Scalar. A number of variables occurring at time $t-1$ and not at $t+1$.
`dyn_nboth`	Scalar. A number of variables occurring at $t+1$ and $t-1$.
`dyn_nforw`	Scalar. A number of variables occurring at $t+1$ and not at $t-1$.
`dyn_vars`	Column vector of endogenous variable names in Dynare++ internal ordering.
`dyn_i_`*endovar*	Scalar. Index of a variable named *endovar* in the `dyn_vars`.
`dyn_shocks`	Column vector of exogenous variable names.
`dyn_i_`*exovar*	Scalar. Index of a shock named *exovar* in the `dyn_shocks`.
`dyn_state_vars`	Column vector of state variables, these are stacked variables counted by `dyn_npred`, `dyn_nboth` and shocks.
`dyn_vcov_exo`	Matrix $nexo \times nexo$. The variance-covariance matrix of exogenous shocks as input in the model file. The ordering is given by `dyn_shocks`.

`dyn_mean`	Column vector *nendo* × 1. The unconditional mean of endogenous variables. The ordering is given by `dyn_vars`.
`dyn_vcov`	Matrix *nendo* × *nendo*. The unconditional covariance of endogenous variables. The ordering is given by `dyn_vars`.
`dyn_rt_mean`	Column vector *nendo* × 1. The unconditional mean of endogenous variables estimated in real-time. See 4.3.1. The ordering is given by `dyn_vars`.
`dyn_rt_vcov`	Matrix *nendo* × *nendo*. The unconditional covariance of endogenous variables estimated in real-time. See 4.3.1. The ordering is given by `dyn_vars`.
`dyn_cond_mean`	Matrix *nendo* × *nper*. The rows correspond to endogenous variables in the ordering of `dyn_vars`, the columns to periods. If t is a period (starting with 1), then t-th column is $E[y_t \vert y_0 = \bar{y}]$. See 4.3.2.
`dyn_cond_variance`	Matrix *nendo* × *nper*. The rows correspond to endogenous variables in the ordering of `dyn_vars`, the columns to periods. If t is a period (starting with 1), then t-th column are the variances of $y_t \vert y_0 = \bar{y}$. See 4.3.2.
`dyn_ss`	Column vector *nendo* × 1. The fix point of the resulting approximation of the decision rule.
`dyn_g_`*order*	Matrix *nendo*×?. A derivative of the decision rule of the *order* multiplied by 1/*order*!. The rows correspond to endogenous variables in the ordering of `dyn_vars`. The columns correspond to a multidimensional index going through `dyn_state_vars`. The data is folded (all symmetrical derivatives are stored only once).
`dyn_steady_states`	Matrix *nendo* × *nsteps* + 1. A list of fix points at which the multi-step algorithm calculated approximations. The rows correspond to endogenous variables and are ordered by `dyn_vars`, the columns correspond to the steps. The first column is always the deterministic steady state.

`dyn_irfp_`*exovar*`_mean`	Matrix *nendo* × *nper*. Positive impulse response to a shock named *exovar*. The row ordering is given by `dyn_vars`. The columns correspond to periods.
`dyn_irfp_`*exovar*`_var`	Matrix *nendo* × *nper*. The variances of positive impulse response functions.
`dyn_irfm_`*exovar*`_mean`	Same as `dyn_irfp_`*exovar*`_mean` but for negative impulse.
`dyn_irfp_`*exovar*`_var`	Same as `dyn_irfp_`*exovar*`_var` but for negative impulse.
`dyn_simul_points`	A simulation path along which the check was done. Rows correspond to endogenous variables, columns to periods. Appears only if `--check P`.
`dyn_simul_errors`	Errors along `dyn_simul_points`. The rows correspond to equations as stated in the model file, the columns to the periods. Appears only if `--check P`
`dyn_ellipse_points`	A set of points on the ellipse at which the approximation was checked. Rows correspond to state endogenous variables (the upper part of `dyn_state_vars`, this means without shocks), and columns correspond to periods. Appears only if `--check E`.
`dyn_ellipse_errors`	Errors on the ellipse points `dyn_ellipse_points`. The rows correspond to the equations as stated in the model file, columns to periods. Appears only if `--check E`.
`dyn_shock_`*exovar*`_errors`	Errors along a shock named *exovar*. The rows correspond to the equations as stated in the model file. There are $2m+1$ columns, the middle column is the error at zero shock. The columns to the left correspond to negative values, columns to the right to positive. Appears only if `--check S`.

7.3 Journal File

The journal file provides information on resources usage during the run and gives some informative messages. The journal file is a text file, it is organized in single line records. The format of records is documented in a header of the journal file.

The journal file should be consulted in the following circumstances:

- Something goes wrong. For example, if a model is not Blanchard–Kahn stable, then the eigenvalues are dumped to the journal file.

  If the unconditional covariance matrix `dyn_vcov` is NaN, then from the journal file you will know that all the simulations had to be thrown away due to occurrence of NaN or Inf. This is caused by non-stationarity of the resulting decision rule.

  If Dynare++ crashes, the journal file can be helpful for guessing a point where it crashed.

- You are impatient. You might be looking at the journal file during the run in order to have a better estimate about the time when the calculations are finished. In Unix, I use a command `tail -f blah.jnl`.[5]

- Heavy swapping. If the physical memory is not sufficient, an operating system starts swapping memory pages with a disk. If this is the case, the journal file can be consulted for information on memory consumption and swapping activity.

- Not sure what Dynare++ is doing. If so, read the journal file, which contains a detailed record on what was calculated, simulated etc.

7.4 Dump File

The dump file is always created with the suffix `.dump`. It is a text file which takes a form of a model file. It sets the parameter values which were used, it has the initval section setting the values which were finally used, and mainly it has a model section of all equations with all substitutions and formed the first order conditions of the planner.

The dump file serves for debugging purposes, since it contains the mathematical problem which is being solved by dynare++.

7.5 Matlab Scripts for Steady State Calculations

This section describes two Matlab scripts, which are useful when calculating the deterministic steady state outside Dynare++. The scripts are created by Dynare++ as soon as an input file is parsed, that is before any calculations.

The first Matlab script having a name *modname*`_f.m` for given parameters values and given all endogenous variables y calculates a residual of the static system. Supposing the model is in the form of (1), the script calculates a vector:

$$f(y, y, y, 0)$$

The second script having a name *modname*`_ff.m` calculates a matrix:

$$\frac{\partial}{\partial y} f(y, y, y, 0)$$

[5] This helps to develop one of the three programmer's virtues: *impatience*. The other two are *laziness* and *hubris*; according to Larry Wall.

Both scripts take two arguments. The first is a vector of parameter values ordered in the same ordering as declared in the model file. The second is a vector of all endogenous variables at which the evaluation is performed. These endogenous variables also include auxiliary variables automatically added by Dynare++ and Lagrange multipliers if an optimal policy problem is solved. If no endogenous variable has not been added by Dynare++, then the ordering is the same as the ordering in declaration in the model file. If some endogenous variables have been added, then the ordering can be read from comments close to the top of either two files.

For example, if we want to calculate the deterministic steady state of the `kp1980.dyn` model, we need to do the following:

1. Run Dynare++ with `kp1980.dyn`, no matter if the calculation has not been finished, important output are the two Matlab scripts created just in the beginning.

2. Consult file `kp1980_f.m` to get the ordering of parameters and all endogenous variables.

3. Create a vector `p` with the parameter values in the ordering

4. Create a vector `init_y` with the initial guess for the Matlab solver `fsolve`

5. Create a simple Matlab function called `kp1980_fsolve.m` returning the residual and Jacobian:

   ```
   function [r, J] = kp1980_fsolve(p, y)
     r = kp1980_f(p, y);
     J = kp1980_ff(p, y);
   ```

6. In the Matlab prompt, run the following:

   ```
   opt=optimset('Jacobian','on','Display','iter');
   y=fsolve(@(y) kp1980_fsolve(p,y), init_y, opt);
   ```

7.6 Custom Simulations

When Dynare++ run is finished it dumps the derivatives of the calculated decision rule to the MAT file. The derivatives can be used for a construction of the decision rule and custom simulations can be run. This is done by `dynare_simul.m` M-file in Matlab. It reads the derivatives and simulates the decision rule with provided realization of shocks.

All the necessary documentation can be viewed by the command:

```
help dynare_simul
```