Graduate Macro Theory II:
Notes on Using Dynare

Eric Sims
University of Notre Dame

Spring 2024

1 Introduction

This document will present some simple examples of how to solve and simulate DSGE models using Dynare. Dynare can also be used to estimate the parameters of DSGE models via Maximum Likelihood or Bayesian Maximum Likelihood. Hello why doesn't this work.

Dynare is not its own program but is rather basically a collection of Matlab codes. To run Dynare, you must first install it. You can download it from the following website: `http://www.dynare.org/download`. I will only support downloads using Matlab. You need to choose a destination folder. My destination folder is in the C drive and is called "dynare." You will need to set a path in Matlab to access the background files necessary to use Dynare.

To run Dynare, you have to create .mod files (you can create these in the m-file editor or other text editor, but be sure to save them as .mod, not as .m files). Then, to run your code, simply type "dynare filename" into the command window (or you can have this command within a separate m-file which can be in another director so long as you have added the Dynare path).

Note that these notes are, by now, several years old. The basic structure of Dynare has not changed, but there is more functionality now and some things have changed. This set of notes is meant to be a replacement for other forms of online documentation, of which there are many. It is possible that some of what I'm writing here is wrong / out of date.

2 A Neoclassical Model with Fixed Labor

Consider a simple stochastic neoclassical model with fixed labor as the laboratory. Consider a planner's problem, which may be be written:

$$\max_{C_t, K_{t++1}} \quad \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \frac{C_t^{1-\sigma} - 1}{1-\sigma}$$

$$\text{s.t.}$$

$$K_{t+1} = A_t K_t^\alpha - C_t + (1-\delta)K_t$$

Productivity is assumed to follow a mean zero AR(1) in the log:

$$\ln A_t = \rho \ln A_{t-1} + s_A \varepsilon_t$$

One can have $\varepsilon_t \sim N(0,1)$ and scaled the shocks by a standard deviation parameter like s_A, or one can separately specify the variance of the shock in Dynare.

The first order conditions for this problem can be characterized with three non-linear difference equations (also the transversality condition, which I do not show):

$$
\begin{align}
C_t^{-\sigma} &= \beta \mathbb{E}_t C_{t+1}^{-\sigma} (\alpha A_{t+1} K_{t+1}^{\alpha-1} + (1-\delta)) \tag{1}\\
K_{t+1} &= A_t K_t^\alpha - C_t + (1-\delta)K_t \tag{2}\\
\ln A_t &= \rho \ln A_{t-1} + s_A \varepsilon_t \tag{3}\\
&\tag{4}
\end{align}
$$

In addition, if I want to I can define two auxiliary "static" variables that define output from the production function and investment from resource constraint:

$$
\begin{align}
Y_t &= A_t K_t^\alpha \tag{5}\\
I_t &= Y_t - C_t \tag{6}
\end{align}
$$

I may also be interested in backing out prices (which would emerge in the equilibrium of a decentralized model). I could consider r_t, R_t^k, and w_t: the real interest rate on bonds, the real rental rate on capital, and the real wage on labor (which we can still price as equaling the marginal product of labor even though labor is inelastically supplied and fixed at a normalized value of unity). The prices are pinned down by:

$$
\begin{align}
C_t^{-\sigma} &= \beta \mathbb{E}_t C_{t+1}^{-\sigma}(1+r_t) \tag{7}\\
R_t^k &= \alpha A_t K_t^{\alpha-1} \tag{8}\\
w_t &= (1-\alpha)A_t K_t^\alpha \tag{9}
\end{align}
$$

Before proceeding I must pick values of the parameters. I set $\sigma = 1$, $\alpha = 1/3$, $\delta = 0.025$, $\beta = 0.99$, $\rho = 0.95$, and the standard deviation of the productivity shock to $s_A = 0.01$.

2.1 A Timing Convention for Endogenous States

When writing a Dynare code, variables can be contemporaneous (date t), lagged $(t-j)$, or led $(t+i)$. Most models can be written with just one lead and one lag (so $j=i=1$). It is possible to have longer leads or lags, but one can define auxiliary state variables to re-write the system with just one lead or lag. In fact, Dynare can do this for you.

Economically, there are really three types of variables in a dynamic model. First are state variables; these are variables that either exogenous (like productivity) or at least partially pre-determined (like capital). Then we have dynamic variables, or what are sometimes called "jump variables" or "jumpers" – these are variables, like consumption, that show up at current and lead values in equilibrium. Then there are static or redundant variables that only show up with contemporaneous values.

There is a timing convention in Dynare that lets Dynare know when something is a predetermined state variable. In particular, Dynare requires that predetermined variables (like the capital stock) show up as dated $t-1$ in the time t equations and t in the $t+1$ equations. This is how you "tell" Dynare if a variable is a state variable or not. In essence, the timing convention in Dynare is that variables need to be dated *from the perspective of the period in which they are chosen*. So, for example, if period t is taken to be the present, capital available in period t was chosen in period $t-1$. Concretely, we need to rewrite the FOCs:

$$
\begin{align}
C_t^{-\sigma} &= \beta \mathbb{E}_t C_{t+1}^{-\sigma}(\alpha A_{t+1} K_t^{\alpha-1} + (1-\delta)) \tag{10}\\
K_t &= A_t K_{t-1}^{\alpha} - C_t + (1-\delta)K_{t-1} \tag{11}\\
\ln A_t &= \rho \ln A_{t-1} + s_A \varepsilon_t \tag{12}\\
Y_t &= A_t K_{t-1}^{\alpha} \tag{13}\\
I_t &= Y_t - C_t \tag{14}\\
C_t^{-\sigma} &= \beta \mathbb{E}_t C_{t+1}^{-\sigma}(1 + r_t) \tag{15}\\
R_t^k &= \alpha A_t K_{t-1}^{\alpha-1} \tag{16}\\
w_t &= (1-\alpha) A_t K_{t-1}^{\alpha} \tag{17}
\end{align}
$$

In other words, we essentially just lag the capital stock one period in each relevant equation relative to the timing notation I typically use. There is no economic difference.

3 Writing the Dynare Code

With the first order conditions in hand, we are now ready to begin to write our Dynare code (which occurs in the .mod file).The dynare code will not run unless each entry is followed by a semicolon to suppress the output.

The first step is to declare the variables of the model. Dynare automatically sorts variables into

the following order: static variables (i.e. only enter FOCs at time t), purely predetermined variables (only enter FOCs at t and $t-1$), variables that are both predetermined and forward-looking (appear at $t-1$, t, and $t+1$), and variables that are purely forward-looking (only appear at dates t and $t+1$). In our model, output is static, capital is purely predetermined, productivity is both predetermined and forward-looking (since future productivity shows up in the Euler equation), and consumption is forward-looking. You can declare variables in any order you want, but it might make sense to try to declare variables in the same order Dynare sorts them.

The first non-comment command in your Dynare code should be: "var" followed by the endogenous variable names and culminating with a semicolon. For the purposes of naming things, everything other than the shock(s) is considered to be endogenous – so A_t is considered endogenous, even though I call it exogenous. My first line of code looks like:

```
1  var y I k a c w R r;
```

The next line of code specifies the exogenous variables. These are just the shocks, and in this model I only have one:

```
1  varexo e;
```

The next step is to declare the parameters of the model. You simply type "parameters" followed by parameter names:

```
1  parameters alpha beta Δ rho sigma sA;
```

Immediately following this command, you need to specify numerical values for these parameters:

```
1  alpha = 1/3;
2  beta = 0.99;
3  Δ = 0.025;
4  rho = 0.95;
5  sigma = 1;
6  SA = 0.01;
```

Now that you've named variables and specified parameter values, it's now time to "declare the model". To do this, you type in "model;" followed by the first order conditions, constraints, identities, etc., followed by "end;". We typically want to approximate the solutions of the natural logs of the variables, so that impulse responses, etc. are all in percentage terms. By default, Dynare will do approximations of the levels of the variables. There are a couple of ways around this. One way is to specify variables in the FOC as $\exp(x)$. Then it will interpret $x = \ln X$ and take an approximation of that. An alternative is to write the FOC in levels and then include auxiliary redundant variables that define logs of variables.

Given all this, you literally just type in the equilibrium conditions of the model. Contemporaneous values of variables are denoted x; lagged values $x(-1)$, and led values $x(+1)$. The equilibrium conditions for this model, using the $\exp(\cdot)$ trick, are:

```
1  model;
2
3  exp(c)^(-sigma) = beta*exp(c(+1))^(-sigma)*(alpha*exp(a(+1))*exp(k)^(alpha-1) +
4    (1-Δ));
5  exp(y) = exp(a)*exp(k(-1))^(alpha);
6  exp(k) = exp(a)*exp(k(-1))^(alpha) - exp(c) + (1-Δ)*exp(k(-1));
7  a = rho*a(-1) + sA*e;
8  exp(y) = exp(c) + exp(I);
9  exp(c)^(-sigma) = beta*exp(c(+1))^(-sigma)*(1+r);
10 exp(R) = alpha*exp(a)*exp(k(-1))^(alpha-1);
11 exp(w) = (1-alpha)*exp(a)*exp(k(-1))^(alpha);
12
13
14 end;
```

Dynare interprets all of the underlying variables here as logs. Since I specified the stochastic process for A_t in the log, I do not need to write it using the exponential. I could alternatively get approximations of logs just by introducing auxiliary variables. For example, to just get the log of output, I could write:

```
1  var Y I K A C w R r y;
```

Here, I am redefining uppercase variables as levels (with the exception of investment, sorry to be a pain), but I'm going to define $y = \ln Y$. My equilibrium conditions would be:

```
1  model;
2
3  C^(-sigma) = beta*C(+1)^(-sigma)*(alpha*A(+1)*K^(alpha-1) + (1-Δ));
4  Y = A*K(-1)^(alpha);
5  K = A*K(-1))^(alpha) - C + (1-Δ)*K(-1));
6  log(A) = rho*log(A(-1)) + sA*e;
7  Y = C + I;
8  C^(-sigma) = beta*C(+1)^(-sigma)*(1+r);
9  R = alpha*A*K(-1)^(alpha-1);
10 w = (1-alpha)*A*K(-1)^(alpha);
11 y = log(Y);
12
13
14 end;
```

To approximate the model, Dynare needs to solve for the steady state. There are a couple of

ways to do this. You can write a steady state file and give it expressions for the steady state (which is pretty easy to do in this model). You can also give it initial conditions, from which it uses a numerical process to solve for the steady state. Another thing I often do is solve for the steady state analytically, then save steady state values as parameters, and then use those steady state values as initial conditions for the numerical solver.

For my baseline example where I have $\exp(\cdot)$ and variables are interpreted as logs, my initial conditions might look like:

```
1  initval;
2  k = log(30);
3  y = log(3);
4  c = log(2.5);
5  I = log(0.5);
6  a = 0;
7  r = (1/beta) - 1;
8  R = log((1/beta) - (1-Δ));
9  w = log(1);
10 end;
```

Make sure you understand whether the endogenous variables are logged or not. If you give the model bad enough initial conditions, it might not find the steady state. If you do not give Dynare initial values, it will still try to solve for a steady state but might have trouble.

The next step is to specify the variance of the shocks. By default, shocks are drawn from a normal distribution. There are two ways to do this. One, that I have done above, is to assume that the shocks come from a standard normal distribution (the standard deviation parameter, s_A, then multiplies the shocks). This part of the code starts with "shocks;", followed by a specification of the variance (or standard deviation) of shocks, followed by "end;".

```
1  var e = 1;
```

Alternatively, I could have not multiplied the e by s_A in the model block, and instead could write this part of the code as:

```
1  var e = sA^2;
```

In the next step you can type in "steady;". This command calculates the steady state values of the endogenous variables of the model:

```
1  steady;
```

This command is actually optional. Dynare will calculate the steady state whether you ask for it or not, but this command ensures that the steady state appears in the command window when

you run the .mod file. Check

The next command is the payoff. It's the "stoch_simul" command, which is what solves the model, produces the policy functions, and generates impulse responses functions and unconditional second moments. There are a number of options following this command. If you just type in "stoch_simul;", it's going to give you the default output. The default output is: (1) steady state values of endogenous variables; (2) a model summary, which counts variables by type; (3) covariance matrix of shocks (which in the example I'm giving is a scalar); (4) the policy and transition functions (in state space notation); (5) theoretical first and second moments; (6) a theoretical correlation matrix; (7) theoretical autocovariances up to order 5; and (8) impulse responses. By default, Dynare does a second order approximation about the steady state; the second order approximation involves "cross terms," and so the policy/transition functions look more complicated than you're used to seeing.

There are a number of options one can include behind "stoch_simul;" (to do this you type "stoch_simul(options);") . There are several of these that you can read about in the manual, but the more important ones for our purposes are:

- hp_filter = integer: this will produce theoretical moments (variances, covariances, autocorrelations) after HP filtering the data (the default is to apply no filter to the data). It is common in the RBC literature at least to focus on HP-filtered moments. The integer is a number (i.e. 1600) corresponding to the penalty parameter in the HP filter. So, for example, typing "stoch_simul(hp_filter=1600);" will produce theoretical (i.e. analytical) moments of the HP filtered data. The only problem here is that it will not simultaneously do HP filtering of simulated data; you can either get moments from simulated data or analytical HP filtered moments

- irf = integer: this will change the number of periods plotted in the impulse response functions. The default value is 40. To suppress printing impulse responses altogether, type in 0 for the number of horizons. For example, to see impulse responses for only 20 periods, type "stoch_simul(irf=20);".

- nocorr: this will mean it will not print a matrix of correlations in the command window

- nofunctions: this will suppress printing the state space coefficients of the solution in the command window

- nomoments: this will suppress printing of moments

- noprint: this will suppress printing of any output at all

- order = 1, 2, or 3: this tells Dynare the order of the (log) approximation. The default is a second order approximation. Hence, typing "order=1" will have it do a linear approximation.

- periods = integer: Dynare's default is to produce analytical/theoretical moments of the variables. Having periods not equal to zero will instead have it simulate data and take the

moments from the simulated data. By default, Dynare drops the first 100 values from a simulation, so you need to give it a number of periods greater than 100 for this to work. Hence, typing "stoch_simul(periods=300);" will produce moments based on a simulation with 200 variables.

- drop = integer: You can change the number of observations to drop from the simulations. For example, typing "stoch_simul(drop=0);" will result in no observations being dropped in the simulations.

- simul_seed = integer: sets the seed used in the random number generator for the simulations.

For example, typing "stoch_simul(nofunctions,hp_filter=1600,order=1,irf=20);" will suppress the policy function output, will produce analytical HP filtered moments, will do a first order approximation, and will plot 20 periods in the impulse responses.

Dynare will produce policy and transition functions for the endogenous variables of the model in state space form. Letting s_t be a $m \times 1$ vector of states (here $m = 2$) and x_t be a $n \times 1$ vector of controls (here I include "static" controls like output as well as dynamic controls; you don't need to eliminate the static controls to use Dynare, because Dynare can figure out that they're static/redundant), which in this model is $n = 6$. The state space representation of the system can be written:

$$s_t = As_{t-1} + B\varepsilon_t$$
$$x_t = \Phi s_t$$

Above A is $m \times m$, and can be found as discussed previously in class. ε_t is $w \times 1$; so in this model $w = 1$; and B is $m \times w$. Φ is the policy function, and is $n \times m$. Dynare writes the system out by substituting the state transition equation into the policy function as follows:

$$x_t = \Phi As_{t-1} + \Phi B\varepsilon_t$$

It is easily verified that these matrix multiplications are permissable; Φ is $n \times m$ and A is $m \times m$. Hence ΦA, call it C, is $n \times m$. B is $m \times w$; hence ΦB is $n \times w$. Call $\Phi B = D$. Then we can write out the full system as:

$$s_t = As_{t-1} + B\varepsilon_t$$
$$x_t = Cs_{t-1} + D\varepsilon_t$$

We can combine this into one expression by denoting $Y_t = [s_t \quad x_t]'$, $\Psi = [A \quad C]'$, and $\Omega = [B \quad D]'$:

8

$$Y_t = \Psi s_{t-1} + \Omega \varepsilon_t$$

3.1   The Full .mod File

Below is my code in the .mod file in its entirety. Note that you can comment in a .mod file just like you can in a regular .m file.

```
1   var y I k a c w R r;
2
3   varexo e;
4
5   parameters alpha beta Δ rho sigma sigmae;
6
7   alpha = 1/3;
8   beta = 0.99;
9   Δ = 0.025;
10  rho = 0.95;
11  sigma = 1;
12  sigmae = 0.01;
13
14  model;
15
16  exp(c)^(-sigma) = beta*exp(c(+1))^(-sigma)*(alpha*exp(a(+1))*exp(k)^(alpha-1) + (1-Δ));
17  exp(y) = exp(a)*exp(k(-1))^(alpha);
18  exp(k) = exp(a)*exp(k(-1))^(alpha) - exp(c) + (1-Δ)*exp(k(-1));
19  a = rho*a(-1) + e;
20  exp(y) = exp(c) + exp(I);
21  exp(c)^(-sigma) = beta*exp(c(+1))^(-sigma)*(1+r);
22  exp(R) = alpha*exp(a)*exp(k(-1))^(alpha-1);
23  exp(w) = (1-alpha)*exp(a)*exp(k(-1))^(alpha);
24
25  end;
26
27  initval;
28  k = log(30);
29  y = log(3);
30  c = log(2.5);
31  I = log(0.5);
32  a = 0;
33  r = (1/beta) - 1;
34  R = log((1/beta) - (1-Δ));
35  w = log(1);
36  end;
37
38  shocks;
```

```
39  var e = sigmae^2;
40  end;
41
42  steady;
43
44  stoch_simul(order=1,irf=20);
```

4 Running Dynare

To run Dynare you can't hit F5 or type the name of the .mod file into the command prompt. You have to type "dynare filename" (without the .mod on the end of the file name). For this to run you must either be in the same directory where your Dynare files are stored, you must have set the path of where your Dynare files are stored, or you must type "addpath (directory)" where the directory is the location of your Dynare files. It is also helpful to add "noclearall" to the declaration so that you are typing in "dynare filename noclearall". Otherwise Dynare will automatically erase all previous output, which you may not want. Another option I often include is "nolog" which tells Dynare to not produce a "log" file. I have found in the past that the creation of log files can cause crashes when you're doing loops.

So as to do other things, I usually write a .m file to go along with my .mod file. My code for the .m file is simple:

```
1   clear all;
2   close all;
3
4   % execute a Dynare file
5
6   % specify parameters
7
8   beta = 0.99;
9   alpha = 1/3;
10  sigma = 1;
11  sigmae = 0.01;
12  rho = 0.9;
13  Δ = 0.025;
14
15  save param_nc alpha beta Δ rho sigma sigmae
16
17  dynare basic_nc_dynare_alt noclearall nolog
```

Here I'm saving the parameters as a .mat file. Dynare allows you to do this, and you can modify the .mod file (as below) to accommodate this. This is particularly useful if you want to produce moments for different parameter configurations that you might want to loop over.

```
1  var y I k a c w R r;
2
3  varexo e;
4
5  parameters alpha beta Δ rho sigma sigmae;
6
7  load param_nc;
8  set_param_value('alpha',alpha);
9  set_param_value('beta',beta);
10  set_param_value('sigma',sigma);
11  set_param_value('Δ',Δ);
12  set_param_value('rho',rho);
13  set_param_value('sigmae',sigmae);
14
15  model;
16
17  exp(c)^(-sigma) = beta*exp(c(+1))^(-sigma)*(alpha*exp(a(+1))*exp(k)^(alpha-1) + (1-Δ));
18  exp(y) = exp(a)*exp(k(-1))^(alpha);
19  exp(k) = exp(a)*exp(k(-1))^(alpha) - exp(c) + (1-Δ)*exp(k(-1));
20  a = rho*a(-1) + e;
21  exp(y) = exp(c) + exp(I);
22  exp(c)^(-sigma) = beta*exp(c(+1))^(-sigma)*(1+r);
23  exp(R) = alpha*exp(a)*exp(k(-1))^(alpha-1);
24  exp(w) = (1-alpha)*exp(a)*exp(k(-1))^(alpha);
25
26  end;
27
28  initval;
29  k = log(30);
30  y = log(3);
31  c = log(2.5);
32  I = log(0.5);
33  a = 0;
34  r = (1/beta) - 1;
35  R = log((1/beta) - (1-Δ));
36  w = log(1);
37  end;
38
39  shocks;
40  var e = sigmae^2;
41  end;
42
43  steady;
44
45  stoch_simul(order=1,irf=20);
```

You can see how I modified the code in the "parameter" block to accommodate the loading in of saved parameters.

By default, Dynare computes the steady state numerically – you give it initial values (if you don't give it initial values it uses values by default, which I think are 0 or 1), and it numerically computes the steady state. This is fine, but can be prone to problems. First, if you give it bad initial values it may not find the steady state, at which point your program crashes. Second, you may encounter situations in which you want to solve the model for different parameter values which could impact the steady state. What may be good initial values for one set of parameters may be bad initial values for another.

There is an easy way to get around this issue – write your own steady state file which solves the steady state analytically for any values of the parameters. You simply name the file the same thing as your .mod file, appended with "_steadystate.m" (note the steady state file is an .m file, not a .mod file). The structure "M" that Dynare produces saves the parameter values; in your steady state file you read the parameters from "M," solve for the steady state values of the variables of the model, and then create matrix called "ys" which has the steady state values. My steady state code is given below:

```
1  function [ys,check] = basic_nc_dynare_ss_steadystate(ys,exe);
2
3  global M_
4
5  alpha = M_.params(1);
6  beta = M_.params(2);
7  Δ = M_.params(3);
8  rho = M_.params(4);
9  sigma = M_.params(5);
10 sigmae = M_.params(6);
11
12 k = (alpha/(1/beta - (1-Δ)))^(1/(1-alpha));
13 y = k^(alpha);
14 I = Δ*k;
15 c = y - I;
16 a = 1;
17 w = (1-alpha)*k^(alpha);
18 R = alpha*k^(alpha-1);
19 r = (1/beta) - 1;
20
21 check = 0;
22
23 ys = [log(y);
24     log(I);
25     log(k);
26     log(a);
27     log(c);
28     log(w);
29     log(R);
30     r];
```

You have to follow this structure (e.g. for whatever reason you need the "check = 0" line in there). But once you have this, you can eliminate the "initval" block of your .mod file and it'll automatically recognize the steady state file.

Finally, something else that comes in handy is to note that Dynare produces *a lot* of output when you run it. One file it produces is a .m file with the same name as your .mod file. Once you've run the .mod file once, you can then just type in the name of the .m file into the command window in Matlab and it'll produce all the same output. This turns out to be substantially faster than typing "dynare file name" because there is all this initialization that Dynare does that slows it down (you will see this immediately below).

5 Dynare Output

Dynare will produce a bunch of output in the command window for you (unless you include options discussed above to suppress it). Below is the output that I get:

```
Configuring Dynare ...
[mex] Generalized QZ.
[mex] Sylvester equation solution.
[mex] Kronecker products.
[mex] Sparse kronecker products.
[mex] Local state space iteration (second order).
[mex] Bytecode evaluation.
[mex] k-order perturbation solver.
[mex] k-order solution simulation.
[mex] Quasi Monte-Carlo sequence (Sobol).
[mex] Markov Switching SBVAR.

Starting Dynare (version 4.4.2).
Starting preprocessing of the model file ...
Found 8 equation(s).
Evaluating expressions...done
Computing static model derivatives:
 - order 1
Computing dynamic model derivatives:
 - order 1
Processing outputs ...done
Preprocessing completed.
Starting MATLAB/Octave computing.
```

```
STEADY-STATE RESULTS:


y   1.12546
I   -0.312508
k   3.37637
a   0
c   0.854424
w   0.719993
R   -3.34953
r   0.010101


MODEL SUMMARY


  Number of variables:         8
  Number of stochastic shocks: 1
  Number of state variables:   2
  Number of jumpers:           2
  Number of static variables:  5



MATRIX OF COVARIANCE OF EXOGENOUS SHOCKS


Variables        e
e             0.000100

POLICY AND TRANSITION FUNCTIONS
```

	y	I	k	a	
Constant	1.125457	-0.312508	3.376372	0	0.8
k(-1)	0.333333	-0.502551	0.962436	0	0.5
a(-1)	0.900000	3.139964	0.078499	0.900000	0.2
e	1.000000	3.488849	0.087221	1.000000	0.2

```
THEORETICAL MOMENTS
```

VARIABLE	MEAN	STD. DEV.	VARIANCE
y	1.1255	0.0286	0.0008
I	-0.3125	0.0743	0.0055
k	3.3764	0.0275	0.0008
a	0.0000	0.0229	0.0005

```
c              0.8544      0.0194      0.0004
w              0.7200      0.0286      0.0008
R             -3.3495      0.0212      0.0005
r              0.0101      0.0007      0.0000
```

```
MATRIX OF CORRELATIONS


Variables      y       I       k       a       c       w       R       r
y            1.0000  0.9017  0.7567  0.9600  0.8569  1.0000  0.4202  0.3167
I            0.9017  1.0000  0.3997  0.9867  0.5499  0.9017  0.7712  0.6957
k            0.7567  0.3997  1.0000  0.5433  0.9854  0.7567 -0.2752 -0.3804
a            0.9600  0.9867  0.5433  1.0000  0.6783  0.9600  0.6576  0.5697
c            0.8569  0.5499  0.9854  0.6783  1.0000  0.8569 -0.1076 -0.2175
w            1.0000  0.9017  0.7567  0.9600  0.8569  1.0000  0.4202  0.3167
R            0.4202  0.7712 -0.2752  0.6576 -0.1076  0.4202  1.0000  0.9938
r            0.3167  0.6957 -0.3804  0.5697 -0.2175  0.3167  0.9938  1.0000
```

```
COEFFICIENTS OF AUTOCORRELATION


Order      1       2       3       4       5
y        0.9367  0.8784  0.8246  0.7748  0.7289
I        0.8827  0.7777  0.6839  0.6001  0.5252
k        0.9980  0.9925  0.9840  0.9730  0.9597
a        0.9000  0.8100  0.7290  0.6561  0.5905
c        0.9927  0.9826  0.9702  0.9559  0.9398
w        0.9367  0.8784  0.8246  0.7748  0.7289
R        0.8768  0.7668  0.6686  0.5811  0.5031
r        0.8868  0.7853  0.6945  0.6133  0.5406
Total computing time : 0h00m01s
```

Several comments on the nature of the output. First, Dynare starts with a "model summary." This includes variables by "type" but note that the types (what Dynare calls states, jumpers, and static) do not necessarily sum to the total number of variables. Dynare counts things which show up in the equilibrium conditions just dated t as static variables; things which show up with a $t-1$ as states, and things which show up with a $t+1$ as jumpers. Something could show up as both a state and a jumper – here, a appears lagged (as a state) and with a "+1" in the Euler equation

(jumper).

The next part of the Dynare output are the policy and transition functions. The "constant" in the policy and transition functions output is just the steady state values; the coefficients of the (modified) state space representation are easy to understand. In addition, Dynare produces means (steady state values), volatilities (standard deviations), and variances for each endogenous variable. It also produces the matrix of correlations as well as coefficients of autocorrelation at different lags. If there are multiple shocks, it will also produce a variance decomposition telling you how important each shock is – using the terminology I've used, this is an "unconditional" variance decomposition in the sense of showing the fraction of the total variance of each variable. Given the options I specified, all of these moments are theoretical (in the sense of being analytical, not based on a simulation).

Dynare also produces impulse responses to the shocks. These are shown here:
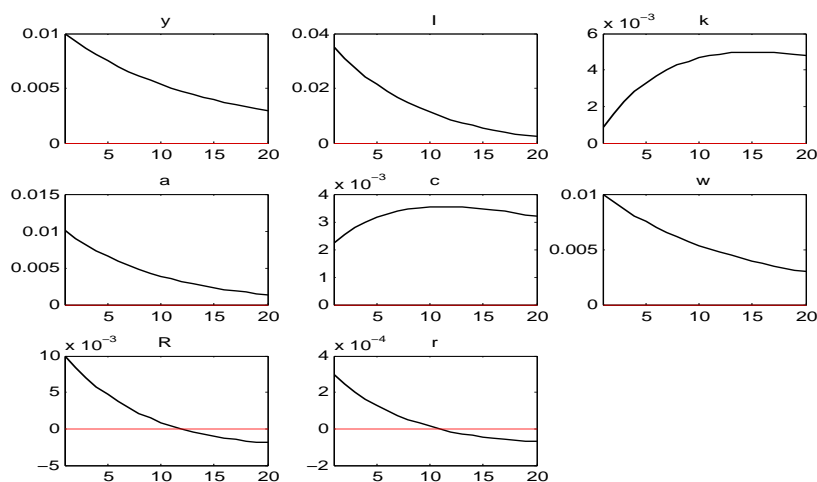
There is one interpretive issue in the IRFs of which you need to be aware. Recall that Dynare interprets k_t as k_{t+1}. Hence, the impulse response of k_t that it produces is really the impulse response of k_{t+1}. To get the impulse responses of k_t, you need to (i) add a zero as the impact response (remember the capital stock can't react on impact as it is predetermined) and (ii) set the response of k_t at horizon h equal to the response of what Dynare gives as the impulse response in period $h + 1$.

6   Where is the Output Stored?

Dynare is great at producing lots of different output right there for you to see. It's a little more challenging to find where your output is stored.

Dynare will store IRFs for you under the following nomenclature: "x_e" denotes the impulse response of variable x to shock e. The coefficients of the state space representation are a bit harder to find. In terms of the notation introduced above, the coefficients of the Ψ matrix are stored in

"oo_.dr.ghx"; the coefficients of the Ω matrix are stored in "oo_.dr.ghu"; and the steady state values are stored in "oo_dr.ys". One irritation with Dynare is that it does not store coefficients of the state space representation in the same order you declare variables (whereas the output produced in the command window is based on what is called the "order of declaration" or the order in which you declare the variables). The "DR" ordering (for "decision rule") is based on the following ordering of variables by type: static variables (only dated t), backward-looking variables (dated t and $t-1$), mixed variables (dated $t-1$, t, and $t+1$), and forward-looking variables (variables dated t and $t+1$). The stored output "oo_dr.inv._order_var" contains the the inverse mapping – so if a variable is declared 2nd, "oo_dr.inv._order_var(2)" will give you the DR order for that variable.

If you want to use the coefficients of the state space representation to do your own simulations, etc., it is helpful to first declare variable orderings in your own code according to the order in which you declared them:

```
1  p_y = 1;
2  p_I = 2;
3  p_k = 3;
4  p_a = 4;
5  p_c = 5;
6  p_w = 6;
7  p_R = 7;
8  p_r = 8;
```

Then I can create matrixes for the state space representation as follows:

```
1   % state space representation: S(t) = A*S(t-1) + B*e(t), X(t) = C*S(t-1) +
2   % D*e(t);
3
4   A = [oo_.dr.ghx(oo_.dr.inv_order_var(p_k),:);
5        oo_.dr.ghx(oo_.dr.inv_order_var(p_a),:)];
6
7   B = [oo_.dr.ghu(oo_.dr.inv_order_var(p_k),:);
8        oo_.dr.ghu(oo_.dr.inv_order_var(p_a),:)];
9
10  C = [oo_.dr.ghx(oo_.dr.inv_order_var(p_y),:);
11       oo_.dr.ghx(oo_.dr.inv_order_var(p_I),:);
12       oo_.dr.ghx(oo_.dr.inv_order_var(p_c),:);
13       oo_.dr.ghx(oo_.dr.inv_order_var(p_w),:);
14       oo_.dr.ghx(oo_.dr.inv_order_var(p_R),:);
15       oo_.dr.ghx(oo_.dr.inv_order_var(p_r),:)];
16
17  D = [oo_.dr.ghu(oo_.dr.inv_order_var(p_y),:);
18       oo_.dr.ghu(oo_.dr.inv_order_var(p_I),:);
19       oo_.dr.ghu(oo_.dr.inv_order_var(p_c),:);
20       oo_.dr.ghu(oo_.dr.inv_order_var(p_w),:);
21       oo_.dr.ghu(oo_.dr.inv_order_var(p_R),:);
```

```
22        oo_.dr.ghu(oo_.dr.inv_order_var(p_r),:)];
```

Then with this, I can create impulse responses on my own, and can verify that they check out with what Dynare produces already:

```
1  % compute impulse responses by hand
2  H = 20;
3  Sirf = zeros(2,H);
4  Xirf = zeros(6,H);
5
6  Sirf(:,1) = B*sigmae;
7  Xirf(:,1) = D*sigmae;
8
9  for j = 2:H
10     Sirf(:,j) = A*Sirf(:,j-1);
11     Xirf(:,j) = C*Sirf(:,j-1);
12  end
```

It is straightforward to verify that these impulse responses are the same as what Dynare produces.

It is helpful to know where these are in the event that you want to conduct your own simulation. I can simulate out variables just as before by drawing shocks from some distribution. For the simulation I assume that everything begins in steady state, so that the initial conditions are all 0 (recall that the variables are deviations about steady state). Here is some code to do that:

```
1  % compute a simulation.  First draw shocks
2  randn('seed',666)
3  T = 200; % number of periods to simulate
4  e = sigmae*randn(1,T);
5
6  Ssim = zeros(2,T);
7  Xsim = zeros(6,T);
8
9  % assume initial state is SS
10  Ssim(:,1) = B*e(1,1);
11  for j = 2:T
12     Ssim(:,j) = A*Ssim(:,j-1) + B*e(1,j);
13     Xsim(:,j) = C*Ssim(:,j-1) + D*e(1,j);
14  end
```

As written, the interpretation of the variables in the two matrixes are as deviations about steady state, so these ought to be mean-zero (in a large enough sample). One might want to add back in the means, which in a linear approximation is just the steady state values. The steady state values are stored in "oo_.dr.ys" – but the steady state values are stored in the order you declared the variables, which is different than the stored coefficients of the state space representation.

18

Here is a figure of the simulated time paths of these variables:
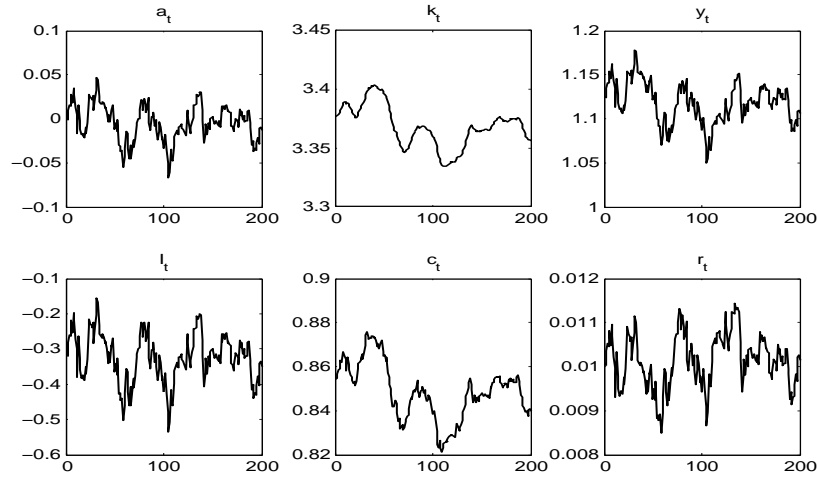
The simulations have some of the features we might expect – consumption is "smoother" than output, which is less volatile than investment. The capital stock evolves very smoothly as well.

Dynare will also directly do simulations for you if you want. For example, if I type in "stoch_simul(order=1,irf=40,periods=200)" it will generate a simulation of 200 periods of the variables of the model. If you use this command then it produces moments (standard deviations, correlations, and autocorrelations) off of the simulated data, as opposed to producing "theoretical" moments using the coefficients of the state space representation (theoretical in the sense that you can calculate analytic moments of the state-space representation because it is linear).

Dynare will also store the second moments statistics that it produces. "oo_.mean" will produce the means of the variables; "oo_.var" produces the variance-covariance matrix (the standard deviations are the square roots of the diagonal elements). "oo_.autocorr" will produce autocorrelations – "oo_.autocorr{i}(k,l)" will produce the ith autocorrelation coefficient between the variables k and l – e.g. $corr(y_t^l, y_{t-i}^l)$. So the first order autocorrelation coefficient of variable j would be "oo_.autocorr{1},(j,j)." The field "oo_.gamma_y{AR+2}" will give you the variance decomposition (only if there is more than one shock), where AR is the number of AR lags it uses (the default is AR = 5).

7   Higher Order Approximations

Dynare will use higher order perturbations (based on 2nd or 3rd order approximations). In fact, by default Dynare solves models using a 2nd order approximation (you have to change the options in "stoch_simul" to get a first order approximation. Higher order approximations have several potential advantages. First, higher order approximations are more appropriate for welfare analysis. Second, higher order approximations are going to be more accurate. Third, higher order approximations can allow you to examine state-dependence (e.g. do impulse responses depend on the

initial state in which a shock hits). Fourth, higher order approximations are necessary if you want to think about "second moment" shocks (e.g. shocks to the variance of a shock . . . you actually need a third order approximation for this to introduce interesting dynamics).

For a second order approximation, Dynare will produce the following output for the state space representation:

$$s_t = \frac{1}{2}\Upsilon_{s,0} + As_{t-1} + B\epsilon_t + \frac{1}{2}\Upsilon_{s,1}\left(s_{t-1}\otimes s_{t-1}\right) + \frac{1}{2}\Upsilon_{s,2}\left(\epsilon_t\otimes\epsilon_t\right) + \Upsilon_{s,3}\left(s_{t-1}\otimes\epsilon_t\right) \quad (18)$$

$$x_t = \frac{1}{2}\Upsilon_{x,0} + Cs_{t-1} + D\epsilon_t + \frac{1}{2}\Upsilon_{x,1}\left(s_{t-1}\otimes s_{t-1}\right) + \frac{1}{2}\Upsilon_{x,2}\left(\epsilon_t\otimes\epsilon_t\right) + \Upsilon_{x,3}\left(s_{t-1}\otimes\epsilon_t\right) \quad (19)$$

The new coefficients are the higher order terms; I call them Υ and index them by s or x (for state or non-state). It is *not* a typo that a $\frac{1}{2}$ term doesn't appear in the last term (think about taking a second order approximation . . . the cross terms are symmetric and show up twice). A first order approximation is a special case of this where all the Υ coefficient matrixes are zero. Note also that the first order coefficients, A, B, C, and D are the same whether the second order terms are zero or not. The operator $\otimes$ is the Kronecker product. I will not go into the details of how these coefficients are found.

The Dynare output for the second order approximation is given below:

```
STEADY-STATE RESULTS:


y    1.12546
I    -0.312508
k    3.37637
a    0
c    0.854424
w    0.719993
R    -3.34953
r    0.010101


MODEL SUMMARY

  Number of variables:         8
  Number of stochastic shocks: 1
  Number of state variables:   2
  Number of jumpers:           2
  Number of static variables:  5
```

MATRIX OF COVARIANCE OF EXOGENOUS SHOCKS

Variables   e
e   0.000100

POLICY AND TRANSITION FUNCTIONS

	y	I	k	a	a
Constant	1.125457	-0.312578	3.376370	0	0.8
(correction)	0	-0.000070	-0.000002	0	0.0
k(-1)	0.333333	-0.502551	0.962436	0	0.5
a(-1)	0.900000	3.139964	0.078499	0.900000	0.2
e	1.000000	3.488849	0.087221	1.000000	0.2
k(-1),k(-1)	0	-0.536347	0.014107	0	0.0
a(-1),k(-1)	0	2.684454	-0.047889	0	-0.0
a(-1),a(-1)	0	-3.408636	0.034945	0	0.0
e,e	0	-4.208192	0.043142	0	0.0
k(-1),e	0	2.982727	-0.053210	0	-0.0
a(-1),e	0	-7.574746	0.077656	0	0.0

APROXIMATED THEORETICAL MOMENTS

VARIABLE	MEAN	STD. DEV.	VARIANCE
y	1.1256	0.0286	0.0008
I	-0.3145	0.0743	0.0055
k	3.3768	0.0275	0.0008
a	0.0000	0.0229	0.0005
c	0.8547	0.0194	0.0004
w	0.7201	0.0286	0.0008
R	-3.3498	0.0212	0.0005
r	0.0101	0.0007	0.0000

APPROXIMATED MATRIX OF CORRELATIONS

Variables	y	I	k	a	c	w	R	r
y	1.0000	0.9017	0.7567	0.9600	0.8569	1.0000	0.4202	0.3167
I	0.9017	1.0000	0.3997	0.9867	0.5499	0.9017	0.7712	0.6957
k	0.7567	0.3997	1.0000	0.5433	0.9854	0.7567	-0.2752	-0.3804

```
a               0.9600  0.9867  0.5433  1.0000  0.6783  0.9600  0.6576  0.5697
c               0.8569  0.5499  0.9854  0.6783  1.0000  0.8569 -0.1076 -0.2175
w               1.0000  0.9017  0.7567  0.9600  0.8569  1.0000  0.4202  0.3167
R               0.4202  0.7712 -0.2752  0.6576 -0.1076  0.4202  1.0000  0.9938
r               0.3167  0.6957 -0.3804  0.5697 -0.2175  0.3167  0.9938  1.0000
```

APPROXIMATED COEFFICIENTS OF AUTOCORRELATION

Order	1	2	3	4	5
y	0.9367	0.8784	0.8246	0.7748	0.7289
I	0.8827	0.7777	0.6839	0.6001	0.5252
k	0.9980	0.9925	0.9840	0.9730	0.9597
a	0.9000	0.8100	0.7290	0.6561	0.5905
c	0.9927	0.9826	0.9702	0.9559	0.9398
w	0.9367	0.8784	0.8246	0.7748	0.7289
R	0.8768	0.7668	0.6686	0.5811	0.5031
r	0.8868	0.7853	0.6945	0.6133	0.5406

Total computing time : 0h00m02s

What Dynare labels "correction" is the coefficient $\Upsilon_{s,0}$ and $\Upsilon_{x,0}$. The coefficients on $k(-1)$, $a(-1)$ and e are the values in the A, B, C, and D matrixes we had above. The coefficients on $k(-1), k(-1)$, $a(-1), k(-1)$, and $a(-1), a(-1)$ are the elements $\Upsilon_{s,1}$ and $\Upsilon_{x,1}$. The coefficients on e, e are the $\Upsilon_{s,2}$ and $\Upsilon_{x,2}$ matrixes, while the $k(-1), e$ and $a(-1), e$ coefficients are the elements of $\Upsilon_{s,3}$ and $\Upsilon_{x,3}$. The rest of the output for Dynare is the same as before, although these are "approximate theoretical moments" sense the model is non-linear. Note that what Dynare produces in the command window already includes the multiplication by $\frac{1}{2}$: what stores for you does not have this, so if you want to do your own simulations, you need to take this into account.

Dynare will produce impulse responses just as before. How does it do this in a non-linear model? It does so by simulation. In a nutshell, it will produce two simulations (using the same draw of shocks), but in one of the periods will add one standard deviation to the shock (or one standard deviation to one of the shocks in a multi-shock model). This is somewhat tricky because the impulse responses here will depend upon the initial value of the state. S

Here is what Dynare does. It will produce T different simulations with $H_1 + H_2$ periods in each simulation, starting from the steady state. It will then average over the T simulations. Then it will repeat this exact same process (using the same draw of shocks), but in period $H_1 + 1$ it will add one standard deviation to the shock (or add one standard deviation to one of the shocks in a one shock model). Then you average across T in the second simulation. Then for periods $H_1 + 1$ through $H_1 + H_2$, you subtract the difference between the second and first simulations, and this

will give you the impulse response function. There are two components to this process. The first H_1 periods in essence averages over the initial state vector – if you knew the state from which you wanted to start the impulse response function, you could set $H_1 = 0$. In this sense what Dynare produces are in a sense "average" impulse response. The second component is the typical approach for simulating an impulse response function in a non-linear model.

Dynare stores in the coefficients in a similar manner as in a first order approximation. I can retrieve the A, B, C, and D matrixes as before. The $\Upsilon_{s,1}$ and $\Upsilon_{x,1}$ coefficients are stored in "oo_.dr.ghxx," the $\Upsilon_{s,2}$ and $\Upsilon_{x,2}$ coefficients in "oo_.dr.ghuu," the $\Upsilon_{s,3}$ and $\Upsilon_{x,3}$ coefficients in "oo_.dr.ghxu," and the $\Upsilon_{s,0}$ and $\Upsilon_{x,0}$ coefficients in "oo_.dr.ghs2." I retrieve the coefficients using the code below:

```
1   [tn,sn] = size(oo_.dr.ghx); % dimensions, tn is total number of variables,
2   % sn is number of states
3   A = zeros(sn,sn);
4   B = zeros(sn,1);
5   Us0 = zeros(sn,1);
6   Us1 = zeros(sn,sn^2);
7   Us2 = zeros(sn,1);
8   Us3 = zeros(sn,sn*1); % 1 i the number of shocks
9
10  C = zeros(tn-sn,sn);
11  D = zeros(tn-sn,1);
12  Us0 = zeros(tn-sn,1);
13  Us1 = zeros(tn-sn,sn^2);
14  Us2 = zeros(tn-sn,1);
15  Us3 = zeros(tn-sn,sn*1); % 1 is the number of shocks
16
17  A = [oo_.dr.ghx(oo_.dr.inv_order_var(p_k),:);
18      oo_.dr.ghx(oo_.dr.inv_order_var(p_a),:)];
19
20  B = [oo_.dr.ghu(oo_.dr.inv_order_var(p_k),:);
21      oo_.dr.ghu(oo_.dr.inv_order_var(p_a),:)];
22
23  C = [oo_.dr.ghx(oo_.dr.inv_order_var(p_y),:);
24      oo_.dr.ghx(oo_.dr.inv_order_var(p_I),:);
25      oo_.dr.ghx(oo_.dr.inv_order_var(p_c),:);
26      oo_.dr.ghx(oo_.dr.inv_order_var(p_w),:);
27      oo_.dr.ghx(oo_.dr.inv_order_var(p_R),:);
28      oo_.dr.ghx(oo_.dr.inv_order_var(p_r),:)];
29
30  D = [oo_.dr.ghu(oo_.dr.inv_order_var(p_y),:);
31      oo_.dr.ghu(oo_.dr.inv_order_var(p_I),:);
32      oo_.dr.ghu(oo_.dr.inv_order_var(p_c),:);
33      oo_.dr.ghu(oo_.dr.inv_order_var(p_w),:);
34      oo_.dr.ghu(oo_.dr.inv_order_var(p_R),:);
35      oo_.dr.ghu(oo_.dr.inv_order_var(p_r),:)];
```

```
36
37   Us1 = [oo_.dr.ghxx(oo_.dr.inv_order_var(p_k),:);
38        oo_.dr.ghxx(oo_.dr.inv_order_var(p_a),:)];
39
40   Us2 = [oo_.dr.ghuu(oo_.dr.inv_order_var(p_k),:);
41        oo_.dr.ghuu(oo_.dr.inv_order_var(p_a),:)];
42
43   Us3 = [oo_.dr.ghxu(oo_.dr.inv_order_var(p_k),:);
44        oo_.dr.ghxu(oo_.dr.inv_order_var(p_a),:)];
45
46   Us0 = [oo_.dr.ghs2(oo_.dr.inv_order_var(p_k),:);
47        oo_.dr.ghs2(oo_.dr.inv_order_var(p_a),:)];
48
49   Ux1 = [oo_.dr.ghxx(oo_.dr.inv_order_var(p_y),:);
50        oo_.dr.ghxx(oo_.dr.inv_order_var(p_I),:);
51        oo_.dr.ghxx(oo_.dr.inv_order_var(p_c),:);
52        oo_.dr.ghxx(oo_.dr.inv_order_var(p_w),:);
53        oo_.dr.ghxx(oo_.dr.inv_order_var(p_R),:);
54        oo_.dr.ghxx(oo_.dr.inv_order_var(p_r),:)];
55
56   Ux2 = [oo_.dr.ghuu(oo_.dr.inv_order_var(p_y),:);
57        oo_.dr.ghuu(oo_.dr.inv_order_var(p_I),:);
58        oo_.dr.ghuu(oo_.dr.inv_order_var(p_c),:);
59        oo_.dr.ghuu(oo_.dr.inv_order_var(p_w),:);
60        oo_.dr.ghuu(oo_.dr.inv_order_var(p_R),:);
61        oo_.dr.ghuu(oo_.dr.inv_order_var(p_r),:)];
62
63   Ux3 = [oo_.dr.ghxu(oo_.dr.inv_order_var(p_y),:);
64        oo_.dr.ghxu(oo_.dr.inv_order_var(p_I),:);
65        oo_.dr.ghxu(oo_.dr.inv_order_var(p_c),:);
66        oo_.dr.ghxu(oo_.dr.inv_order_var(p_w),:);
67        oo_.dr.ghxu(oo_.dr.inv_order_var(p_R),:);
68        oo_.dr.ghxu(oo_.dr.inv_order_var(p_r),:)];
69
70   Ux0 = [oo_.dr.ghs2(oo_.dr.inv_order_var(p_y),:);
71        oo_.dr.ghs2(oo_.dr.inv_order_var(p_I),:);
72        oo_.dr.ghs2(oo_.dr.inv_order_var(p_c),:);
73        oo_.dr.ghs2(oo_.dr.inv_order_var(p_w),:);
74        oo_.dr.ghs2(oo_.dr.inv_order_var(p_R),:);
75        oo_.dr.ghs2(oo_.dr.inv_order_var(p_r),:)];
```

To produce impulse responses, here is what I do.

1. I first draw shocks of dimension $T \times (H1 + H2)$. I specify $T = 100$ (I'm average over 100 periods), $H1 = 50$ (I have a 50 period "burn in"), and $H2 = 20$ (the dimension of the impulse responses.

2. I simulate out values of the variables T times using these shocks. So I get T different simulations with $H1 + H2$ periods each.

3. I average the simulated series over T at each horizon.

4. I repeat this process, but add one standard deviation to the shock in period $H1 + 1$ in generating the series.

5. I take the difference between the second simulation and the first for periods $H1 + 1$ through $H2$. These are the impulse responses.

This will produce impulse responses; these will be close (but not identical to all decimal places) to what Dynare produces. By default, Dynare use $H1 = 100$ (a 100 period burn-in) and $T = 50$ (averaging over 50 periods). I'm using different values (even if they were the same, the IRFs wouldn't identical because we're not drawing the same shocks). Below is my Matlab code to produce impulse responses.

```matlab
1   H1 = 50;
2   H2 = 20;
3   T = 100;
4   % draw shocks
5   randn('seed',666)
6   e = sigmae*randn(T,H1+H2);
7
8   slag = zeros(sn,1); % starting position
9   Ssim = zeros(sn,T,H1+H2);
10  Xsim = zeros(tn-sn,T,H1+H2);
11
12
13  for j = 1:T
14      Ssim(:,j,1) = (1/2)*Us0 + A*slag + B*e(j,1) + (1/2)*Us1*kron(slag,slag)...
15          + (1/2)*Us2*kron(e(j,1),e(j,1)) + Us3*kron(slag,e(j,1));
16      Xsim(:,j,1) = (1/2)*Ux0 + C*slag + D*e(j,1) + (1/2)*Ux1*kron(slag,slag)...
17          + (1/2)*Ux2*kron(e(j,1),e(j,1)) + Ux3*kron(slag,e(j,1));
18
19      for i = 2:H1+H2
20          Ssim(:,j,i) = (1/2)*Us0 + A*Ssim(:,j,i-1) + B*e(j,i)  + ...
21              (1/2)*Us1*kron(Ssim(:,j,i-1),Ssim(:,j,i-1)) + ...
22              (1/2)*Us2*kron(e(j,i),e(j,i)) + Us3*kron(Ssim(:,j,i-1),e(j,i));
23          Xsim(:,j,i) = (1/2)*Ux0 + C*Ssim(:,j,i-1) + D*e(j,i) + ...
24              (1/2)*Ux1*kron(Ssim(:,j,i-1),Ssim(:,j,i-1)) + ...
25              (1/2)*Ux2*kron(e(j,i),e(j,i)) + Ux3*kron(Ssim(:,j,i-1),e(j,i));
26      end
27
28  end
29
30  % now calculate means
31  Sm = zeros(sn,H1+H2);
32  Xm = zeros(tn-sn,H1+H2);
33
```

```matlab
34  for k = 1:sn
35      for i = 1:H1+H2
36          Sm(k,i) = mean(Ssim(k,:,i));
37      end
38  end
39
40  for k = 1:tn-sn
41      for i = 1:H1+H2
42          Xm(k,i) = mean(Xsim(k,:,i));
43      end
44  end
45
46  % Now do a second simulation
47  Ssim2 = zeros(sn,T,H1+H2);
48  Xsim2 = zeros(tn-sn,T,H1+H2);
49
50  % Keep e the same, but add sigmae to it in period H1+1
51  e1 = e;
52  e1(:,H1+1) = e1(:,H1+1) + sigmae;
53
54  % Now resimulate using e1
55  for j = 1:T
56      Ssim2(:,j,1) = (1/2)*Us0 + A*slag + B*e1(j,1) + ...
57          (1/2)*Us1*kron(slag,slag) + (1/2)*Us2*kron(e1(j,1),e1(j,1)) ...
58          + Us3*kron(slag,e1(j,1));
59      Xsim2(:,j,1) = (1/2)*Ux0 + C*slag + D*e1(j,1) + ...
60          (1/2)*Ux1*kron(slag,slag) + (1/2)*Ux2*kron(e1(j,1),e1(j,1)) + ...
61          Ux3*kron(slag,e1(j,1));
62
63      for i = 2:H1+H2
64          Ssim2(:,j,i) = (1/2)*Us0 + A*Ssim2(:,j,i-1) + B*e1(j,i) + ...
65              (1/2)*Us1*kron(Ssim2(:,j,i-1),Ssim2(:,j,i-1)) + ...
66              (1/2)*Us2*kron(e1(j,i),e1(j,i)) + ...
67              Us3*kron(Ssim2(:,j,i-1),e1(j,i));
68          Xsim2(:,j,i) = (1/2)*Ux0 + C*Ssim2(:,j,i-1) + D*e1(j,i) + ...
69              (1/2)*Ux1*kron(Ssim2(:,j,i-1),Ssim2(:,j,i-1)) + ...
70              (1/2)*Ux2*kron(e1(j,i),e1(j,i)) + ...
71              Ux3*kron(Ssim2(:,j,i-1),e1(j,i));
72      end
73
74  end
75
76  % now calculate means
77  Sm2 = zeros(sn,H1+H2);
78  Xm2 = zeros(tn-sn,H1+H2);
79
80  for k = 1:sn
81      for i = 1:H1+H2
82          Sm2(k,i) = mean(Ssim2(k,:,i));
```

```
83        end
84    end
85
86    for k = 1:tn-sn
87        for i = 1:H1+H2
88            Xm2(k,i) = mean(Xsim2(k,:,i));
89        end
90    end
91
92    Sirf = zeros(sn,H2);
93    Xirf = zeros(tn-sn,H2);
94
95    for j = 1:H2
96        Sirf(:,j) = Sm2(:,H1+j) - Sm(:,H1+j);
97        Xirf(:,j) = Xm2(:,H1+j) - Xm(:,H1+j);
98    end
```

An irritation sometimes arises here. Unlike in a first order approximation, higher order approximations can explode. In a simple model like this explosion is unlikely for reasonably sized shocks, but you can get the model to be explosive if you make the shocks big enough. In multi-shock models explosion occurs somewhat more frequently.

How do we compute impulse responses and simulations if the coefficients of the second order state space representation are explosive? We can use what is called "pruning." To get Dynare to do this you just have to added "pruning" as an option in the "stoch_simul" command. Doing pruning by yourself is fairly straightforward. There are multiple ways of implementing it. The easiest is as follows. When doing the simulation, do both a linear and a non-linear simulation (the linear simulation just uses A, B, C, and D matrixes, which are the same whether you solve the model with a first or a second order approximation). Then, when doing your simulation for the non-linear model, for the "higher order" terms in the approximation (the things multiplied by Υ coefficients) used the state vector simulated from the linear simulation (but still use the state vector from the non-linear simulation for the linear term). This will guarantee that the simulation does not explode. My code for the pruned construction of impulse responses is given below:

```
1    %%%%% Now do this by pruning
2    slag = zeros(sn,1); % starting position
3    Ssim = zeros(sn,T,H1+H2);
4    Xsim = zeros(tn-sn,T,H1+H2);
5    Ssim_lin = zeros(sn,T,H1+H2); % the linear simulation
6
7
8    for j = 1:T
9        Ssim(:,j,1) = (1/2)*Us0 + A*slag + B*e(j,1) + (1/2)*Us1*kron(slag,slag)...
10           + (1/2)*Us2*kron(e(j,1),e(j,1)) + Us3*kron(slag,e(j,1));
11        Ssim_lin(:,j,1) = A*slag + B*e(j,1);
12        Xsim(:,j,1) = (1/2)*Ux0 + C*slag + D*e(j,1) + (1/2)*Ux1*kron(slag,slag)...
```

```matlab
13           + (1/2)*Ux2*kron(e(j,1),e(j,1)) + Ux3*kron(slag,e(j,1));
14
15     for i = 2:H1+H2
16         Ssim(:,j,i) = (1/2)*Us0 + A*Ssim(:,j,i-1) + B*e(j,i)  + ...
17             (1/2)*Us1*kron(Ssim_lin(:,j,i-1),Ssim_lin(:,j,i-1)) + ...
18             (1/2)*Us2*kron(e(j,i),e(j,i)) + Us3*kron(Ssim_lin(:,j,i-1),e(j,i));
19         Ssim_lin(:,j,i) =  A*Ssim_lin(:,j,i-1) + B*e(j,i);
20         Xsim(:,j,i) = (1/2)*Ux0 + C*Ssim(:,j,i-1) + D*e(j,i) + ...
21             (1/2)*Ux1*kron(Ssim_lin(:,j,i-1),Ssim_lin(:,j,i-1)) + ...
22             (1/2)*Ux2*kron(e(j,i),e(j,i)) + Ux3*kron(Ssim_lin(:,j,i-1),e(j,i));
23     end
24
25 end
26
27 % now calculate means
28 Sm = zeros(sn,H1+H2);
29 Xm = zeros(tn-sn,H1+H2);
30
31 for k = 1:sn
32     for i = 1:H1+H2
33         Sm(k,i) = mean(Ssim(k,:,i));
34     end
35 end
36
37 for k = 1:tn-sn
38     for i = 1:H1+H2
39         Xm(k,i) = mean(Xsim(k,:,i));
40     end
41 end
42
43 % Now do a second simulation
44 Ssim2 = zeros(sn,T,H1+H2);
45 Xsim2 = zeros(tn-sn,T,H1+H2);
46 Ssim2_lin = zeros(sn,T,H1+H2);
47
48 % Keep e the same, but add sigmae to it in period H1+1
49 e1 = e;
50 e1(:,H1+1) = e1(:,H1+1) + sigmae;
51
52 % Now resimulate using e1
53 for j = 1:T
54     Ssim2(:,j,1) = (1/2)*Us0 + A*slag + B*e1(j,1) + ...
55         (1/2)*Us1*kron(slag,slag) + (1/2)*Us2*kron(e1(j,1),e1(j,1)) ...
56         + Us3*kron(slag,e1(j,1));
57     Ssim2_lin(:,j,1) = A*slag + B*e1(j,1);
58     Xsim2(:,j,1) = (1/2)*Ux0 + C*slag + D*e1(j,1) + ...
59         (1/2)*Ux1*kron(slag,slag) + (1/2)*Ux2*kron(e1(j,1),e1(j,1)) + ...
60         Ux3*kron(slag,e1(j,1));
61
```

```
62      for i = 2:H1+H2
63          Ssim2(:,j,i) = (1/2)*Us0 + A*Ssim2(:,j,i-1) + B*e1(j,i) + ...
64              (1/2)*Us1*kron(Ssim2_lin(:,j,i-1),Ssim2_lin(:,j,i-1)) + ...
65              (1/2)*Us2*kron(e1(j,i),e1(j,i)) + ...
66              Us3*kron(Ssim2_lin(:,j,i-1),e1(j,i));
67          Ssim2_lin(:,j,i) = A*Ssim2_lin(:,j,i-1) + B*e1(j,i);
68          Xsim2(:,j,i) = (1/2)*Ux0 + C*Ssim2(:,j,i-1) + D*e1(j,i) + ...
69              (1/2)*Ux1*kron(Ssim2_lin(:,j,i-1),Ssim2_lin(:,j,i-1)) + ...
70              (1/2)*Ux2*kron(e1(j,i),e1(j,i)) + ...
71              Ux3*kron(Ssim2_lin(:,j,i-1),e1(j,i));
72      end
73
74  end
75
76  % now calculate means
77  Sm2 = zeros(sn,H1+H2);
78  Xm2 = zeros(tn-sn,H1+H2);
79
80  for k = 1:sn
81      for i = 1:H1+H2
82          Sm2(k,i) = mean(Ssim2(k,:,i));
83      end
84  end
85
86  for k = 1:tn-sn
87      for i = 1:H1+H2
88          Xm2(k,i) = mean(Xsim2(k,:,i));
89      end
90  end
91
92  Sirf_prune = zeros(sn,H2);
93  Xirf_prune = zeros(tn-sn,H2);
94
95  for j = 1:H2
96      Sirf_prune(:,j) = Sm2(:,H1+j) - Sm(:,H1+j);
97      Xirf_prune(:,j) = Xm2(:,H1+j) - Xm(:,H1+j);
98  end
```

This code is virtually the same as the one I used above. The only differences are that (1) I compute a linear simulation of the state along with the non-linear simulation, and (2) I use the value of the state vector from the linear simulation only in the higher order terms of the non-linear simulation. You can verify that there are only tiny differences between the pruned and standard impulse responses. To see why and where pruning can make a difference, increase the shock standard deviation by 100 – you'll see that the standard approach explodes and therefore can't be used to construct impulse responses, whereas the pruned simulation works.

Another area in which a higher order approximation is useful is in doing welfare analysis. Welfare is the present discounted value of household utility. You can include a recursive representation of

welfare in your Dynare code as follows:

$$V_t = \frac{c_t^{1-\sigma} - 1}{1 - \sigma} + \beta E_t V_{t+1} \tag{20}$$

This looks like a Bellman equation, but there is no max operator (this is because implicitly in the code c_t is chosen optimally). Since I'm using $\sigma = 1$ and Matlab/Dynare can't understand limits, when writing this in my code I need to use $\ln C_t$ in flow utility. To include this in Dynare, declare V as a variable and add the following equilibrium condition:

```
1  V = c + beta*V(+1);
```

Note that because my Dynare code interprets c as $\ln c_t$, this is why I just have this showing up as c (e.g. utility is not linear). This is all I have to do. The inclusion of this variable does not affect the Dynamics of any other variable in the model, it just allows me to look at how V (e.g. welfare) is affected.

The steady state value of V is easily seen to be $\frac{1}{1-\beta} \ln c^*$ – e.g. this is just proportion to the log of steady state consumption. In a first order approximation, the expected values of variables correspond to their steady state values. This would tell you that average welfare is the same in an economy with a standard deviation of productivity shocks of 0.01 versus an economy with a standard deviation of productivity shocks of 0.02. But we know this can't necessarily be right – households have a preference to smooth consumption, so one would think that bigger shocks would make them have lower expected welfare.

We can see this if we use a higher order approximation. First, I solve the model using a second order approximation where the standard deviation of productivity shocks is 0.01. The output from Dynare is found below:

```
Configuring Dynare ...
[mex] Generalized QZ.
[mex] Sylvester equation solution.
[mex] Kronecker products.
[mex] Sparse kronecker products.
[mex] Local state space iteration (second order).
[mex] Bytecode evaluation.
[mex] k-order perturbation solver.
[mex] k-order solution simulation.
[mex] Quasi Monte-Carlo sequence (Sobol).
[mex] Markov Switching SBVAR.

Starting Dynare (version 4.4.2).
Starting preprocessing of the model file ...
Found 9 equation(s).
```

```
Evaluating expressions...done
Computing static model derivatives:
 - order 1
Computing dynamic model derivatives:
 - order 1
 - order 2
Processing outputs ...done
Preprocessing completed.
Starting MATLAB/Octave computing.


STEADY-STATE RESULTS:

y    1.12546
I    -0.312508
k    3.37637
a    0
c    0.854424
w    0.719993
R    -3.34953
r    0.010101
V    85.4424

MODEL SUMMARY

  Number of variables:         9
  Number of stochastic shocks: 1
  Number of state variables:   2
  Number of jumpers:           3
  Number of static variables:  5



MATRIX OF COVARIANCE OF EXOGENOUS SHOCKS

Variables       e
e            0.000100


POLICY AND TRANSITION FUNCTIONS
                                    y              I              k              a
Constant                    1.125457      -0.312578       3.376370              0       0.8
```

(correction)	0	-0.000070	-0.000002	0	0.(
k(-1)	0.333333	-0.502551	0.962436	0	0.5
a(-1)	0.900000	3.139964	0.078499	0.900000	0.2
e	1.000000	3.488849	0.087221	1.000000	0.2
k(-1),k(-1)	0	-0.536347	0.014107	0	0.0
a(-1),k(-1)	0	2.684454	-0.047889	0	-0.0
a(-1),a(-1)	0	-3.408636	0.034945	0	0.0
e,e	0	-4.208192	0.043142	0	0.0
k(-1),e	0	2.982727	-0.053210	0	-0.0
a(-1),e	0	-7.574746	0.077656	0	0.0

APROXIMATED THEORETICAL MOMENTS

VARIABLE	MEAN	STD. DEV.	VARIANCE
y	1.1256	0.0286	0.0008
I	-0.3145	0.0743	0.0055
k	3.3768	0.0275	0.0008
a	-0.0000	0.0229	0.0005
c	0.8547	0.0194	0.0004
w	0.7201	0.0286	0.0008
R	-3.3498	0.0212	0.0005
r	0.0101	0.0007	0.0000
V	85.4704	0.5380	0.2895

APPROXIMATED MATRIX OF CORRELATIONS

Variables	y	I	k	a	c	w	R	r	V
y	1.0000	0.9017	0.7567	0.9600	0.8569	1.0000	0.4202	0.3167	0.9517
I	0.9017	1.0000	0.3997	0.9867	0.5499	0.9017	0.7712	0.6957	0.7254
k	0.7567	0.3997	1.0000	0.5433	0.9854	0.7567	-0.2752	-0.3804	0.9209
a	0.9600	0.9867	0.5433	1.0000	0.6783	0.9600	0.6576	0.5697	0.8276
c	0.8569	0.5499	0.9854	0.6783	1.0000	0.8569	-0.1076	-0.2175	0.9738
w	1.0000	0.9017	0.7567	0.9600	0.8569	1.0000	0.4202	0.3167	0.9517
R	0.4202	0.7712	-0.2752	0.6576	-0.1076	0.4202	1.0000	0.9938	0.1213
r	0.3167	0.6957	-0.3804	0.5697	-0.2175	0.3167	0.9938	1.0000	0.0102
V	0.9517	0.7254	0.9209	0.8276	0.9738	0.9517	0.1213	0.0102	1.0000

```
APPROXIMATED COEFFICIENTS OF AUTOCORRELATION


Order      1       2       3       4       5
y       0.9367  0.8784  0.8246  0.7748  0.7289
I       0.8827  0.7777  0.6839  0.6001  0.5252
k       0.9980  0.9925  0.9840  0.9730  0.9597
a       0.9000  0.8100  0.7290  0.6561  0.5905
c       0.9927  0.9826  0.9702  0.9559  0.9398
w       0.9367  0.8784  0.8246  0.7748  0.7289
R       0.8768  0.7668  0.6686  0.5811  0.5031
r       0.8868  0.7853  0.6945  0.6133  0.5406
V       0.9746  0.9490  0.9233  0.8975  0.8718
Total computing time : 0h00m02s
```

Note that the average value of V is 85.4704, which is actually higher than the steady state value (which evidently means that you prefer, in an average sense, stochastic productivity shocks to no productivity shocks). This might seem somewhat odd since you have a preference to smooth consumption. I actually have a paper on this with Mike Pries and Rob Lester in the *Journal of Economic Dynamics and Control.* It turns out that it is conceivable that you prefer more shock variance to less depending on parameters, but there is something about which you want to be careful here. What matters for agents in the economy is the level of a_t. I wrote the process for a_t in the log. It turns out that the mean of a_t is increasing in the variance of $\ln a_t$ via a Jensen's inequality type argument. An alternative way to write the process for a_t is as an AR(1) in the level with a mean of zero:

$$a_t = (1 - \rho) + \rho a_{t-1} + e_t$$

It turns out that if you do this, a higher variance of e_t reduces expected welfare. This is because writing the process this way the stochastic mean of a_t is unaffected by the variance of the shock. When writing the process for a_t in the log you are confounding two effects – more variance with a higher mean of a_t. When writing the process this way, you are isolating only the effect of the variance. For more variance to be desirable for welfare, you need factor supply to be very elastic, which it is not here. I bring this subtle point up because we usually don't worry about whether we write the exogenous processes in levels of log-levels. To a first order approximation, they are the same. But to higher order this stuff can matter.

Another subtle point here that emerges is whether one should evaluate welfare conditional on a particular realization of a state or unconditionally. By unconditional welfare I mean the exercise I undertook above – look at the stochastic mean of V_t. You can compare two different regimes

by doing this (i.e. a high and low variance regime, or in a more involved model different policy regimes). But the comparisons are in a sense not fair because the different regimes fluctuate around different stochastic means. The example I gave above is an extreme example of this. An alternative is to look at conditional welfare. For example, you can condition on looking at the non-stochastic steady state, and evaluate welfare in two different regimes at that point (rather than looking at unconditional welfare).

There is increasing interest in the effects of risk or uncertainty shocks – in essence, shocks to the variance of shocks. These shocks obviously have no effect in a first order approximation. Interesting, they also have no (important) effect in a second order approximation (I say important because they can affect means but do not affect dynamics). The basic intuition for this is based on studying risk aversion and precautionary saving – the second derivative of utility being negative says that people don't like risk, but whether or not they act on it depends on the third derivative of utility being non-zero (if it is positive, they respond to more uncertainty by trying to save). Hence, to get large effects of uncertainty shocks, you need at least a third order approximation. I will not discuss how to do that here, but Dynare is capable of doing it.