User Guide for the National Treasury Dynamic Stochastic General Equilibrium Model (NT-DSGE)[1]

Contents

[1] This user guide summarises content from various online sources. Most content relies on the following comprehensive source from the Dynare team: Stéphane Adjemian, Houtan Bastani, Michel Juillard, Frédéric Karamé, Ferhat Mihoubi, Willi Mutschler, Johannes Pfeifer, Marco Ratto, Normann Rion and Sébastien Villemot (2024), "Dynare: Reference Manual, Version 6," Dynare Working Papers, 80, CEPREMAP. The structure of this guide also borrows heavily from Jones and Kulish (2016), "A Practical Introduction to DSGE Modeling with Dynare" (link). Content from the personal webpages for Willi Mutschler and Johannes Pfeifer were also relied on extensively for the development of this guide and associated resources.

Overview

This document is a practitioner's guide to using the NT-DSGE model. It provides the necessary information to use the NT-DSGE model for policy analysis and to develop the skills to maintain and improve it. The practitioner should have a basic understanding of DSGE models and should be familiar with MATLAB before attempting to use the model. A link to supplementary training resources is available under References, which includes a short MATLAB primer. The link takes you to the repository for training material and capacity building that was used for the Modelling & Forecasting Unit in the Economic Policy Division at National Treasury.

A high-level overview of the NT-DSGE model is provided below to provide readers with the overarching context of the model. The guide then provides the necessary steps to install and configure Dynare with MATLAB, create the model code, understand the Dynare output and MATLAB workspace, and perform basic analysis. The guide then covers the estimation of a model with timeseries data and the baseline commands and options for sensitivity and identification analysis, forecasting, and optimal policy analysis. Key code syntax and results are described throughout to help users. Throughout, the guide focuses on content applicable to the NT-DSGE model. The guide also includes practical sections on Dynare troubleshooting, the Dynare macro processing language, and how to get LaTeX output from Dynare.

To supplement these examples, the guide includes a summary of the research articles produced using the NT-DSGE model as examples of the types of results that can be obtained for policy analysis: fiscal multipliers at different time horizons, policy counterfactuals for alternative fiscal policies (e.g., optimal fiscal rules for debt stabilisation), and forecasting conditional on policy interventions (e.g., a large-scale persistent fiscal transfer such as a basic income grant). The replication files are provided for these research articles as well.

This document concludes with sections covering best practices for model maintenance and improvement. A section on FAQs can be updated overtime as common questions arise from the Modelling & Forecasting Unit. The document will be refined and updated to include any further modelling and forecasting developments.

Summary of the NT-DSGE model

To evaluate different fiscal policy options, we use a dynamic stochastic general equilibrium (DSGE) model based on Kemp & Hollander (2020). This new-Keynesian open-economy fiscal DSGE model has several notable features:

1. **Environment**. The model environment is dynamic (it is multi-period), stochastic (it includes uncertainty), and general equilibrium (it captures the interaction of supply and demand in key markets).
2. **Behaviour.** Household and firm behaviour is rational and forward-looking, optimising utility and profits, respectively.
3. **Households.** There are two types of households—Ricardian (rich) and non-Ricardian (poor). Both households consume domestic and imported consumer goods, and government consumption expenditure provides households with direct utility. Optimizing (Ricardian) households purchase domestic and foreign bonds and rent physical capital to firms and decide how much to invest each period, with changes to the rate of investment, as well as changes to the rate of capital utilization, subject to adjustment costs. Non-Ricardian households do not display the standard optimizing behaviour, cannot invest in physical capital, and do not have access to financial markets. As a result, in each period, they consume all their income (including fiscal transfers). This distinction allows for analysing both macroeconomic and redistributive policies and their welfare effects (e.g., consumption and labour supply effects).

4. **Firms**. There are two types of monopolistically competitive intermediate-good firms producing differentiated goods that are sold both domestically and abroad, and foreign intermediate-good firms producing goods for the domestic market. Additionally, there is a set of four representative domestic final-good firms which combine the differentiated domestic and foreign intermediate goods into four distinct non-tradeable goods, namely a private consumption good, a public consumption good, a private investment good, and a public investment good. Each intermediate-good firm has access to the same public capital stock.

5. **Economic frictions.** Prices of all goods (domestic, foreign, imports, exports) and wages for both types of households exhibit nominal stickiness. By supplying differentiated products, each intermediate goods producer sells its output in both the domestic and foreign market under monopolistic competition with staggered price setting and price indexation. Similarly, by supplying differentiated labour services to firms through unions, households set their wages in a monopolistically competitive setting—wage adjustments are sluggish due to the presence of staggered wage contracts and price indexation. Nominal rigidities are necessary for justifying a role for monetary stabilisation policy. Real economic frictions include investment and capital utilisation adjustment costs and habit persistence in consumption.

6. **Fiscal policy:** The fiscal authority participates in economic activity by purchasing public consumption goods, investing in public capital, issuing bonds, making transfers between households, and levying time-varying taxes on consumption, labour, and capital income. Public consumption and capital goods can either be a complement or substitute for private consumption and capital goods, and taxes are distortionary.

   Fiscal policy actions are governed by a set of six fiscal reaction functions (or "rules") that respond automatically to economic conditions (output and debt). The model identifies the relative impacts of different tax policy mixes (VAT, PIT and CIT) and adjustments to the composition of public expenditure (consumption, investment and transfers). The fiscal authority's budget constraint is maintained period-by-period. Together, these features ensure long-run fiscal sustainability by requiring that government debt stabilises over time.

   The specification of the fiscal sector balances the need for a high degree of detail, which is essential for analysing the quantitative effects of fiscal policy, and tractability, which allows for the identification of the key transmission mechanisms.

7. **Monetary Policy:** The monetary authority sets the short-term nominal interest rate in response to inflation and output—a Taylor-type rule for policy. The model

therefore captures monetary-fiscal interactions, incorporating channels through which policies influences aggregate demand and supply, and the simultaneous reactions of fiscal and monetary policy to economic outcomes.

8. **Foreign economy:** Modelled as a standard three-equation new-Keynesian setup. The model includes channels through which the domestic economy is affected by foreign trade and capital markets.

While DSGE models are powerful tools for counterfactual policy analysis, they remain stylized representations of the real world. Empirical evaluations based on these models are therefore constrained by their specifications and the inherent challenges of forward-looking policy evaluation. Nonetheless, this approach provides valuable insights into the impacts of different fiscal strategies, especially in the context of South Africa's economic and fiscal data.

The `DYNARE` software

About

Dynare is a software platform for managing a wide class of economic models.[2] It offers a user-friendly and intuitive way of describing DSGE models. Essentially, Dynare works on top of MATLAB® and it performs simulations of the model given a calibration of the model parameters and is also able to estimate these parameters given a dataset. In practice, the user will write a text file containing the list of model variables, the dynamic equations linking these variables together, the computing tasks to be performed and the desired graphical or numerical outputs. Dynare is compatible with Windows (10 and 11), MATLAB (R2018b–R2024a) and GNU Octave (9.1.0 only). Only 64-bit versions are supported.

Installation and configuration

To get Dynare, go to http://www.dynare.org/download and download the latest stable version: the file should be `dynare-*-win.exe`, where * is the version number. Install Dynare by running the .exe file. Let it install to its default folders, typically `C:\dynare\*`, where * is the version number. All the MATLAB routines needed to run Dynare are in `C:\dynare\*\matlab`.

After installing Dynare, MATLAB needs to be directed to recognise Dynare files. That is, the path of the Dynare MATLAB files needs to be set in MATLAB.

[2] Most notably, dynamic stochastic general equilibrium (DSGE) and overlapping generations (OLG) models. Dynare can also estimate deterministic models, VARs (structural; Bayesian; Markov-Switching), and DSGE-VARs. Dynare also provides an interface to the X-13 ARIMA-SEATS seasonal adjustment program produced, distributed, and maintained by the US Census Bureau (2017).

In MATLAB, click `File` → `SetPath`, then click on `Add Folder...`, and select the 'matlab' subdirectory (i.e., folder) of your Dynare installation, `C:\dynare\*\matlab`.[3] All the Dynare files should now appear in the MATLAB search path. Apply the settings by clicking on "Save". After closing, MATLAB will remember this setting next time you run it.

To test if Dynare has installed properly, type 'dynare' into the MATLAB command line. If it is installed correctly, you should get the error message complaining that you did not specify a MOD file: `??? Input argument "fname" is undefined`. If not installed correctly, you should get the error message: `??? Undefined function or variable 'dynare'.`

For more detailed information about how to use Dynare, you should have a look at the documentation located in the 'doc' subdirectory of your Dynare installation (you should have a shortcut in your Start Menu to access it directly).

Beginners should start with the tutorial (under PDF format in 'guide.pdf'). There is also a complete reference manual documenting all Dynare functions (under HTML format in 'dynare-manual.html\index.html', under PDF format in 'dynare-manual.pdf').

You can also get more information on the Dynare homepage: https://www.dynare.org

Note: Dynare comes with an automated uninstaller, which you can run from the "Add or remove programs" menu of the Control Panel.

The DYNARE code

The DYNARE code is straightforward to write, as the system of equilibrium conditions are written in the natural way (i.e., equation-by-equation; matrix notation is not necessary). Dynare reads in a .mod text file and parses the model into MATLAB files, which then call on the Dynare MATLAB routines to do the model analysis. This section shows how to write the model in a Dynare consistent .mod file. We proceed step-by-step starting with the equilibrium conditions for a general equilibrium model. The whole code is reported in the appendix.

We will follow a benchmark real business cycle (RBC) model as an example.[4] The dynamic equilibrium of this economy is as follows (here, upper-case letters represent variables in levels):

[3] Note that you must **not** use "Add with Subfolders...".

[4] This section closely follows that of Juillard and Villemot (2009) "Stochastic simulations with DYNARE. A practical guide" which is available as a file 'guide.pdf' in the dynare documentation under `C:\dynare\*\doc`. The 'guide.pdf' document shows the model in levels and in logs. We consider the model in log-linearized form since this follows the setup of the NT-DSGE model. All the code is provided as a resource along with this guide to help readers see the alternative ways to code the model up. Detailed step-by-step documentation is available on the NT-DSGE GitHub repository for deriving and solving the

(1) $C_t^{\eta} A N_t^{\phi} = (1-\alpha)\frac{Y_t}{N_t} = W_t$

(2) $\beta E_t\left[\left(\frac{C_t}{C_{t+1}}\right)^{\eta} R_{t+1}\right] = 1$

(3) $Y_t = \mathrm{Z}_t K_t^{\rho} N_t^{1-\rho}$

(4) $K_{t+1} = (Y_t - C_t) + (1-\delta)K_t$, where $V_t = Y_t - C_t$

(5) $\log Z_t = \psi \log Z_{t-1} + \epsilon_t$, where $\epsilon_t \sim i.i.d.\ \mathcal{N}(0, \sigma^2)$

(6) $R_t = \rho\frac{Y_t}{K_t} + (1-\delta)$

Equation (1) equates the marginal rate of substitution between consumption and labour with the marginal product of labour (which can be interpreted as the real wage; we will assume for simplicity that $\phi = 0$). Equation (2) is the Euler equation which determines the optimal path of consumption over time. Here, the marginal product of capital can be defined as the real rate of return (Equation (6)). Equation (3) is the Cobb-Douglas production function describing how output is produced using input factors, capital and labour. Equation (4) describes the capital accumulation process from investment (i.e., output less consumption). Equation (5) represents a stochastic shock to technology or Solow residual.

The log-linearized version of this model can be written as (here, lower-case letters represent variables in log-deviations from steady state):

$$\eta c_t = y_t - n_t$$

$$c_t = E_t[c_{t+1}] - \frac{1}{\eta} r_{t+1}$$

$$y_t = z_t + \rho k_{t-1} + (1-\rho)n_t$$

$$k_t = (1-\delta)k_{t-1} + \delta v_t$$

$$z_t = \psi z_{t-1} + \epsilon_{z,t}$$

$$y_t = \frac{C}{Y}c_t + \delta\frac{K}{Y}v_t$$

$$r_t = \rho\frac{Y}{K}(y_t - k_{t-1})$$

Where the follow steady state conditions apply:

$$R = (1+r) = \frac{1}{\beta}$$

RBC model used here (~\NationalTreasuryDSGE\training\ntdsge_content\codes). For a new Keynesian example, see Ireland (2004), which comes with comprehensive notes. The dynare version of this code is available at: JohannesPfeifer/DSGE_mod/. Many other replication codes are available here to explore.

$$\frac{K}{Y} = \frac{\rho}{R-(1-\delta)}$$

$$\frac{C}{Y} = 1-\delta\frac{K}{Y}$$

Detailed step-by-step documentation for deriving, solving, and simulating the RBC model used here is available on the NT-DSGE GitHub repository: ~\training\ntdsge_content\codes\Basic_RBC.

Creating the model code

Preamble

We now want to write the model into a form which Dynare can interpret. The preamble consists of the declarations to setup the endogenous and exogenous variables, the parameters and assign values to these parameters. (Be careful not to use MATLAB reserved names such as INV, i, alpha, etc. for your parameters and variables. It can cause issues when solving the model.)

First, there are seven equations, so we have seven variables. The syntax for writing this in Dynare is `var [variable1, variable2, ...];` with the line ended by a semicolon. For the RBC model we write:

```
var y, c, k, v, n, r, z, b ;
```

Table 1: Endogenous

Variable	LaTeX	Description
y	y	output
c	c	consumption
n	n	hours
v	v	investment
k	k	capital
r	r	real rate
z	z	TFP

Next, we write the exogenous variables in the model. The syntax for writing this in Dynare follows similarly as `varexo [variable1, variable2, ...];` with the line ended by a semicolon.

```
varexo eps_z ;
```

Table 2: Exogenous

Shock	LaTeX	Description
epsilon	ϵ_z	TFP shock

Note that ϵ_z is the exogenous shock, whilst z defines the stochastic process and is therefore specified as a variable in the system. See the RBC note for more details.

Next, we write the parameters of the model. In the RBC model, there are five structural parameters plus one standard deviation for the TFP shock, giving six parameters. The syntax follows similarly as:

```
parameters rho, bet, delt, psi, eta, rho, sigmae ;
```

Table 3: Parameters

Parameter	LaTeX	Description
rho	ρ	capital share
bet	β	discount factor
delt	δ	depreciation rate
psi	ψ	persistence TFP shock
eta	η	risk aversion
sigmae	σ_e	i.i.d TFP shock

We then assign parameter values. This is done the standard way in MATLAB. For example, we write

```
alpha = 0.36;
rho = 0.95;
tau = 0.025;
beta = 0.99;
delta = 0.025;
psi = 0;
theta = 2.95;
```

Declaration of the model

We now write the equations of the linearized model. The equation block starts with `model (linear);` and ended by `end;`. The equations are written in the same the way we write it "by hand".

However, there is a simple rule that should be kept in mind when the model is written and a few syntax considerations:

- Variables which are multiplied (divided) with a parameter are separated by the parameter with `*` (`/`).
- For lagged variables (i.e., when the variable is decided in $t-1$, such as the capital stock in our simple model), we write $k(-1)$.
- For expectational variables (i.e., when a variable is decided in the next period, $t+1$), such as consumption in the Euler equation, we write $c(+1)$.
- Contemporaneous variables do not have a time subscript in the Dynare code.
- Equations are ended by a semicolon `;`.

The Dynare syntax to declare our model are:

```
model (linear);
     #R = 1/ bet;
     #KY = rho / (R - (1- delt)) ;
     #CY = 1 - KY * delt ;

     y = CY * c + delt * KY * I ;
     (1/ eta) * r(+1) = c(+1) - c ;
     eta * c = y - n ;
     r = rho *(1 / KY) * (y - k(-1)) ;
     k = (1- delt) * k(-1) + delt * I ;
     y = z + rho * k(-1) + (1 - rho) * n;
     z = psi*z( -1) + epsilon ;
end;
```

In a similar way to the model specified above, notice that we can make the code easier to read by using '#' to define the implied steady states in the model block. Dynare simply replaces all parts of the code where R, KY, and CY appear.

Solving the model

After specifying the equations, we need to tell Dynare where to begin simulations or impulse responses: the initial values of the variables. We do this in a similar way to specifying the parameter values, beginning the block with `initval;` and `end;` and setting the variable to a value. Each variable, endogenous or exogenous, should be initialized. Alternatively, we could provide only approximated values. DYNARE would then automatically compute the exact values. Usually, the initial values correspond to the steady state of the model, particularly for linearized models. For the RBC model, all the variables enter the Dynare routines as deviations from their steady-state values, and so the steady-state for each of the model variables are zero. In this case, instead of setting each variable to an initial value we can simply write `steady;` which begins the simulation from the model's steady-state (we can also include the command `check;` which prints out the steady state conditions for us to confirm the values are all indeed zero).

After specifying the equations, we define the shocks and set the shocks' variances (innovations). This is done using a `shocks;` and `end;` block.

```
shocks;
var epsilon = sigmae^2;
end;
```

For simulations with more than one shock, it is possible to shut down one or more shocks by assigning it a zero variance. We can also declare the shock with the following syntax, which more closely follows the syntax used to estimate the model with time series data:

```
var epsilon; stderr sigmae;
```

Finally, we solve and simulate the model using the command `stoch_simul;` without specifying any further options. By default, the coefficients of the approximated decision rules are reported as well as the theoretical moments of the variables and impulse response functions for each exogenous shock are plotted. There are a lot of options which can be used with `stoch_simul`, which can be found in the Dynare manual. The full .mod file for this RBC model is given in the Appendix. Some command-line options to start practicing with:

- `periods=[Integer]`: If different from zero, the model will be simulated, and empirical moments will be computed instead of theoretical moments. The value of the option specifies the number of periods to use in the simulations (default = 0)
- `replic=[Integer]`: Number of simulated series used to compute the IRFs (default =1, if order =1, and 50 otherwise)
- `ar=[Integer]`: Order of autocorrelation coefficients to compute and to print (default =5)
- `nocorr`: Doesn't print the correlation matrix (default = PRINT)
- `drop=[Integer]`: Number of points dropped at the beginning of simulation before computing the summary statistics (default =100)
- `irf=[Integer]`: Number of periods on which to compute the IRFs (default =40)
- `nofunctions`: Doesn't print the coefficients of the approximated solution
- `nomoments`: Doesn't print moments of the endogenous variables
- `order=[1,2,3]`: Order of Taylor approximation (default =2)[5]

Running DYNARE and output analysis

To run the Dynare `.mod` file created, save the text file with a .mod extension in MATLAB's current directory; for example, saving the file as `basicrbc.mod`. To initiate Dynare, in MATLAB's command line, type

```
dynare basicrbc.mod
```

DYNARE reads the contents of the `.mod` file, recognises the model's details and solves for the linear rational expectations solution. Dynare will print a lot of analytical information to the MATLAB command window, save much of that to the workspace, and generate figure(s) which plot, for example, impulse responses to the shock(s) for each variable. It also saves everything that it generates to the working directory.

[5] If the model is already log-linearised around steady state, Taylor order approximations of the model of order greater than 1 won't matter.

The first bit of printed output DYNARE generates is processing information. The [mex] lines say that it found .mex files – MATLAB executables that do some computations more efficiently in non-Matlab languages. It is worth checking that the number of equations Dynare finds in the .mod file is the same as the number of endogenous variables in the model. For the RBC model, there are seven endogenous variables.

```
Configuring Dynare ...
[mex] Generalized QZ.
[mex] Sylvester equation solution.
[mex] Kronecker products.
[mex] Sparse kronecker products.
[mex] Bytecode evaluation.
[mex] k-order perturbation solver.
[mex] k-order solution simulation.

Starting Dynare (version 6.0).
Calling Dynare with arguments: none
Starting preprocessing of the model file ...
Found 7 equation(s).
Evaluating expressions...
Computing static model derivatives (order 1).
Normalizing the static model...
Finding the optimal block decomposition of the static model...
3 block(s) found:
  2 recursive block(s) and 1 simultaneous block(s).
  the largest simultaneous block has 5 equation(s)
                                 and 5 feedback variable(s).
Computing dynamic model derivatives (order 2).
Normalizing the dynamic model...
Finding the optimal block decomposition of the dynamic model...
2 block(s) found:
  1 recursive block(s) and 1 simultaneous block(s).
  the largest simultaneous block has 6 equation(s)
                                 and 6 feedback variable(s).
Preprocessing completed.
Preprocessing time: 0h00m00s.
```

Next, Dynare will print steady-state values for all the variables, confirm that the rank condition is verified, and print a summary of the model variables:

```
STEADY-STATE RESULTS:

y           0
c           0
n           0
v           0
k           0
r           0
z           0

EIGENVALUES:
         Modulus              Real         Imaginary
          0.9335            0.9335                 0
```

```
                0.95             0.95                0
               1.081            1.081                0
           2.642e+16       -2.642e+16                0

There are 2 eigenvalue(s) larger than 1 in modulus for 2 forward-
looking variable(s)
The rank condition is verified.

MODEL SUMMARY

  Number of variables:         7
  Number of stochastic shocks: 1
  Number of state variables:   2
  Number of jumpers:           2
  Number of static variables:  3
```

Next, Dynare generates the variance-covariance matrix of the shocks. The values in the diagonals correspond to the variance specified `epsilon` values (i.e., square(s) of the standard error(s)) in the `.mod` file. Since we only consider one shock in our model, we obtain a single value:

```
MATRIX OF COVARIANCE OF EXOGENOUS SHOCKS
Variables     epsilon
epsilon      0.000100
```

Dynare computes the solution of the model as: $\hat{\boldsymbol{y}}_t = A\hat{\boldsymbol{y}}_{t-1} + B\boldsymbol{u}_t$, where $\hat{y}_t$ is a vector of endogenous variables as a deviation from steady-state and u_t is a vector of the exogenous shocks. The first two lines of the output below correspond to matrix A^T , while the last row correspond to matrix B^T.

```
POLICY   AND        TRANSITION  FUNCTIONS
         y          c           n         v         k         r          z
k(-1)    -0.16998     0.28813  -0.74624  -1.66099  0.933475  -0.04107      0
z(-1)    1.536575    0.330545  0.875485  5.461801  0.136545  0.053935   0.95
epsilon  1.617447    0.347942  0.921563  5.749264  0.143732  0.056774      1
```

The next section of Dynare output reports the moments of the model, using the model solution and the specified standard errors of the exogenous shocks.

```
MOMENTS     OF          SIMULATED    VARIABLES
VARIABLE    MEAN        STD.DEV.     VARIANCE    SKEWNESS    KURTOSIS
y             -0.02181     0.03334   0.001112  -0.490209   0.424914
c             -0.00637    0.010836   0.000117  -1.349888   1.644659
n             -0.00906    0.015524   0.000241   0.401716  -0.601753
v             -0.07203    0.109936   0.012086  -0.179657    0.02991
k             -0.00685    0.016415   0.000269  -1.246354   1.699985
r             -0.00058    0.000964   0.000001   0.381915  -0.590251
z             -0.01403    0.021584   0.000466  -0.652402   0.640544
```

Note also that the default number of periods to simulate is zero (i.e., periods=0) which means that the theoretical moments are calculated. For periods set larger than 0 the empirical moments are calculated based on the simulated series.

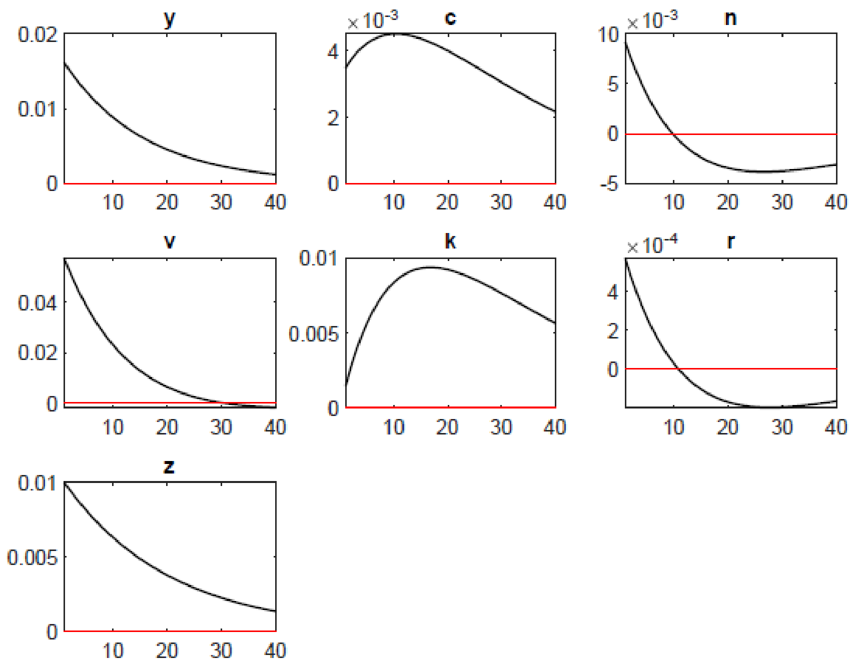
Next, Dynare reports the forecast error variance decomposition of the endogenous variables. It gives the proportion of the variance in the endogenous variable which can be attributed to each of the exogenous shocks at the calibrated values. The sum for each variable across the shocks is 100%. However, for our model, the only source of variation is the single technology shock. For theoretical moments, the variance decomposition can be calculated at specified horizons using the `stoch_simul` option: `conditional_variance_decomposition = [1 4 8 12 20 40].`

Dynare then reports a matrix of correlations across the endogenous variables and the coefficients of autocorrelation up to the fifth order:

```
CORRELATION  OF      SIMULATED  VARIABLES
VARIABLE     y       c          n          v       k       r       z
y                 1     0.9274     0.8529  0.9928  0.6927  0.8909  0.9982
c            0.9274          1     0.5957  0.8759  0.9122  0.6563  0.9485
n            0.8529     0.5957          1  0.9094  0.2142   0.997  0.8196
v            0.9928     0.8759     0.9094       1  0.6012  0.9389  0.9837
k            0.6927     0.9122     0.2142  0.6012       1  0.2895  0.7352
r            0.8909     0.6563      0.997  0.9389  0.2895       1  0.8616
z            0.9982     0.9485     0.8196  0.9837  0.7352  0.8616       1
```

```
AUTOCORRELATION  OF      SIMULATED  VARIABLES
VARIABLE                1          2          3       4       5
y                  0.8394     0.7194     0.5587  0.5044  0.4008
c                  0.9176     0.8532     0.7612  0.7073  0.6198
n                  0.7338      0.547     0.3196  0.2939  0.1981
v                  0.8144     0.6768     0.4968  0.4455  0.3402
k                  0.9784     0.9348     0.8678  0.7804  0.6768
r                  0.7466     0.5667     0.3453  0.3139  0.2153
z                  0.8521     0.7413     0.5912  0.5363  0.4344
```

Finally, Dynare will generate graphics for the impulse responses of the endogenous variables to each of the exogenous shocks. Sine our model example only has one shock with 7 endogenous variables, a single graphic is produced as follows:

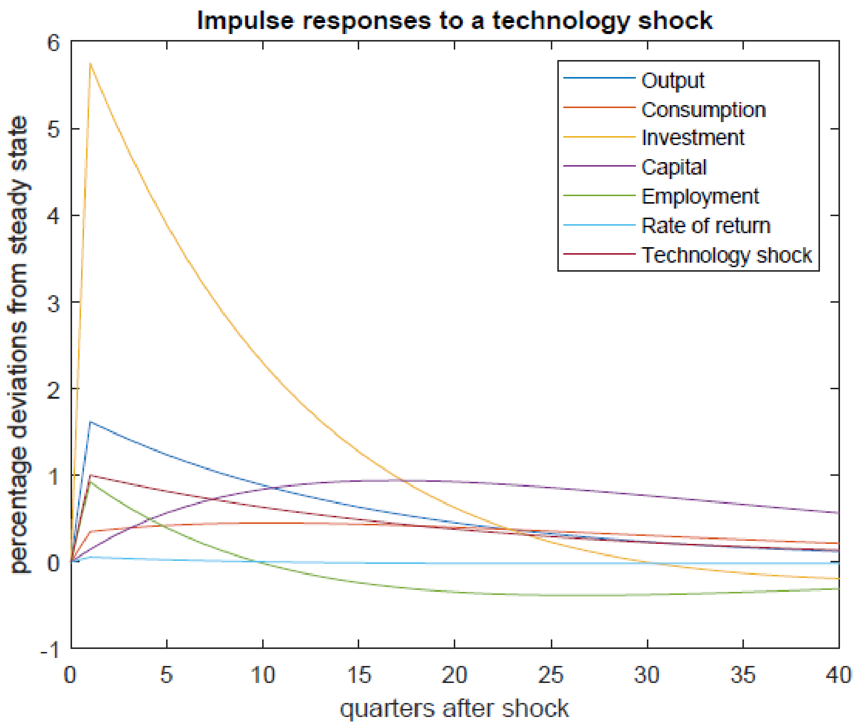


Note that we can set the file format for the figure to be produced using the `stoch_simul` option, e.g., `graph_format=(fig, pdf)`. The y-axes show log-deviations from steady-state, which can be interpreted as %-changes (or percentage points for variables in rates) once multiplied by 100.

We can also create a customized figure using MATLAB code calling from the results stored by Dynare in the MATLAB Workspace (also stored in the "Output" folder under `basicrbc2015_results.mat`) as such:

```
figure
plot([0:options_.irf], [0 oo_.irfs.y_epsilon]*100)
hold
plot([0:options_.irf], [0 oo_.irfs.c_epsilon]*100)
plot([0:options_.irf], [0 oo_.irfs.v_epsilon]*100)
plot([0:options_.irf], [0 oo_.irfs.k_epsilon]*100)
plot([0:options_.irf], [0 oo_.irfs.n_epsilon]*100)
plot([0:options_.irf], [0 oo_.irfs.r_epsilon]*100)
plot([0:options_.irf], [0 oo_.irfs.z_epsilon]*100)
title('Impulse responses to a technology shock')
legend('Output','Consumption','Investment','Capital','Emp
loyment','Rate of return','Technology shock')
ylabel('percentage deviations from steady state')
xlabel('quarters after shock')
```

which produces the following output:



The MATLAB workspace

Table 1 gives information on some of what is stored by Dynare to the MATLAB workspace. More generally, Dynare stores to the workspace all the parameter values and impulse responses of all endogenous variables to shocks.[6]

Table 1: The MATLAB Workspace

Stored in	Description
`oo_.exo_simul`	Simulated series of the exogenous variables (if simulating).
`oo_.endo_simul`	Simulated series of the endogenous variables (if simulating).
`oo_.dr`	Stored information about variables and policy function.
`oo_.dr.order_var`	The mapping from the declaration order to the decision rule order.
`oo_.dr.inv_order_var`	The mapping from the decision rule order to the declaration order.
`oo_.dr.ghx`	The A matrix of the transition function.
`oo_.dr.ghu`	The B matrix of the transition function.
`oo_.steady_state`	Values of the endogenous variables' steady state.
`oo_.variance_decomp…`	The variance decomposition outputted to the command window.
`oo_.mean`	The mean of endogenous variables.
`oo_.var`	The variance-covariance matrix.
`oo_.autocorr`	The autocorrelation matrices for endogenous variables; to 5 lags.
`oo_.irfs`	Impulse responses of endogenous variables to exogenous shocks.

[6] A note on MATLAB data structures. MATLAB can store information in struct objects, which consist of fields, which can be any compatible MATLAB item, like a matrix, string, another structure, etc. To call a field of a structure from the command line, type `struct.field`. Most of the important output from Dynare is stored in structures.

`oo_.irfs.x_y`	Impulse response of x to an innovation in y.
`oo_.forecast`	Forecasts of endogenous variables if forecast declared.
`M_`	Structure with information about the model.
`M_.endo_names`	The names of endogenous variables in declaration order.
`M_.exo_names`	The names of exogenous variables in declaration order.
`M_.exo_nbr`	The number of exogenous variables.
`M_.endo_nbr`	The number of endogenous variables.
`options_`	with all the options which Dynare recognises.

More on the solution output

it's worth understanding how Dynare orders the variables of the model in the computed solution matrices. Dynare distinguishes the endogenous variables by their type, that is, whether they fall into one of the following categories:

- Predetermined variables: Those that appear in the model with only current or past period timing. In our model, there are 2 predetermined variables: k and z. The number of these appears in `M_.npred`.
- Purely forward-looking variables: Those appearing at current and future periods. In our model, there are two purely forward-looking variables: c and r. The number of these appears in `M.nfwrd`.
- Mixed variables: Those variables with backward, current and forward-looking components. There are none of these in our model. The number of these appears in `M_.nboth`.
- Static variables: Those who only appear at current timing. There are three in our model: y, v and n. The number of these appears in `M_.nstatic`.

The DR ordering is <u>static variables</u>, then <u>predetermined variables</u>, then <u>mixed variables</u>, then <u>purely forward-looking variables</u>. The mapping from declaration order in the var section to DR order is given in `oo_.dr.order_var`. The mapping backwards from DR order to declaration order is `oo_.dr.inv_order_var`. That is, suppose your vector of endogenous variables is $\boldsymbol{y}_t$ and these variables are ordered in declaration order, then to rearrange the vector into DR order, write `y_dr = y(oo .dr.order var,:)`. To reverse the mapping from the DR order to declaration order, write `y_dec = y dr(oo .dr.inv order var,:)`. Check that `y_dr` is the same as `y_dec`.

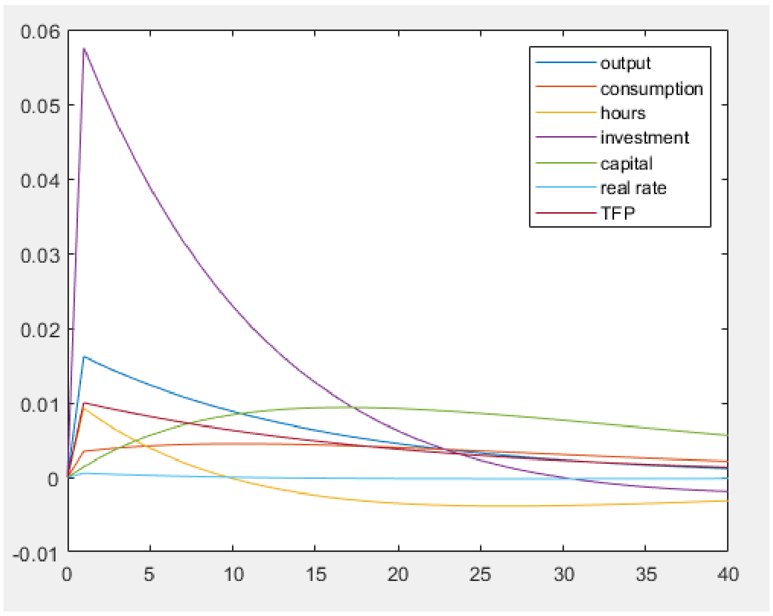
To see why the ordering is relevant, consider the solution which was computed as:

$$\hat{\boldsymbol{y}}_t = A\hat{\boldsymbol{y}}_{t-1} + B\boldsymbol{u}_t$$

A appears in oo_.dr.ghx and B appears in oo_.dr.ghu. Dynare calculates the matrices so that the variables in each row are in DR order. The columns of A correspond to the state variables. The mapping of these state variables from the declaration order can be found in the following vector: `oo_.dr.order_var(M_.nstatic + 1 : M_.nstatic + M_.npred)` vector, which in our example are the fourth and fifth rows, which correspond to the fifth and seventh variables that we declared in the `var` block (check this by copy-and-pasting this command after running the model). These variables are the same as the lagged variables which appear in the policy and transition functions in the Dynare output shown above.

Now, suppose we want to manually calculate, using the solution matrices, the impulse response of all endogenous variables to a one standard deviation policy shock for 40 quarters, and have the output correspond to the declaration order of the variables. We can do this with the following code. The plot of the non-zero variables is in the figure below, which corresponds with the figure produced in the previous section.

```
horizon = 40 ;
shocks = zeros(M_.exo_nbr,horizon) ;
irf = zeros(M_.endo_nbr,horizon) ;
shocks(1,1) = sigmae; % Policy shock in period 1
irf(:,1) = oo_.dr.ghu(oo_.dr.inv_order_var,:)*shocks(:,1)
;
for t=2:horizon
irf(:,t) = oo_.dr.ghx(oo_.dr.inv_order_var,:) * ...
irf(oo_.dr.order_var( ...
M_.nstatic + 1:M_.nstatic + M_.npred),t-1) ...
+ oo_.dr.ghu(oo_.dr.inv_order_var,:)*shocks(:,t) ;
end
% Note the irf matrix is mapped to the declaration order
figure
for i=1:M_.endo_nbr
plot(0:horizon,[0,irf(i,:)])
hold on
end
legend(M_.endo_names_long);
hold off
```



Troubleshooting `DYNARE`

Dynare can report ambiguous and/or difficult to understand error messages. Dynare does tend to improve these error messages in newer versions, so the following examples may change slightly. Here are some common errors in writing the `.mod` file, and the error messages that Dynare reports when it is executed. Note that Dynare will not pick up logical errors, such as errors in transcribing the linearized equations into the `.mod` file.

Parsing errors

In general, if Dynare comes across a parsing error, it will output a message to the MATLAB command window like:

```
ERROR: basicRBC2015.mod: line 35, col 1

Error using dynare
Dynare: preprocessing failed
```

Which indicates there was a parsing error on line 10, column 1 of the Dynare .mod file. The following error may appear if a variable or parameter is omitted from the `var [.]` declaration, but it appears in the model equations. For example, if we forgot to include variable `y`:

```
ERROR: basicRBC2015.mod: line 35, col 1: Unknown symbol: y
```

A similar error may appear when an `end ;` line has been omitted after the model section:

```
ERROR: basicRBC2015.mod: line 68, cols 1-3: syntax error,
unexpected VAR
```

If `end ;` is omitted after the shocks section, you may get the following error because Dynare is trying to read in a line `stoch_simul` which does not accord with the `shocks ;` nomenclature:

```
ERROR: basicRBC2015.mod: line 76, cols 1-11: syntax error,
unexpected NAME, expecting CORR or END or VAR
```

The `unexpected NAME` error may also appear when Dynare encounters an option it does not understand, for example, by mistakenly having the option `nograp` instead of `nograph`. The `unexpected NAME` may also appear if a mathematical operator is omitted in the declaration of the model equations, or if lines are not separated by a semi-colon. For example, if a $*, -, +$ or / operator is missing or if ; is missing.

If a different number of equations appear than the number of declared endogenous variables, Dynare will immediately produce the following error, for example:

```
ERROR: There are 7 equations but 8 endogenous variables!
```

Declaration errors

If a parameter is not declared to have a value in the .mod file, you may see the following warning whilst dynare is attempting its preprocessing:

```
Warning: Some of the parameters have no value (bet) when
using steady. If these parameters are not initialized in a
steadystate file or a steady_state_model-block, Dynare may
not be able to solve the model. Note that simul,
perfect_foresight_setup, and perfect_foresight_solver do
not automatically call the steady state file.
```

Followed by the error message:

```
Error using print_info

The steady state has NaNs or Inf.
```

This error indicates that parsing was done successfully, but the model could not be solved because the matrices which describe the model cannot be constructed. This could occur because a parameter is misspelt in the parameter calibration block. In the example above, a value for `bet` is not provided.

If declared (i.e., calibrated) parameters lead to an indeterminate equilibrium (i.e., no unique equilibrium), Dynare reports that the Blanchard-Kahn conditions are not met:

```
Error using print_info

Blanchard & Kahn conditions are not satisfied:
indeterminacy.
```

The Blanchard Kahn conditions may also not be met if parameters are specified such that there is no equilibrium. For example, if the autoregressive parameter on a temporary technology shock is greater than one, it implies that every solution to the model will have an explosive path. Dynare reports:

```
Error using print_info

Blanchard & Kahn conditions are not satisfied: no stable
equilibrium.
```

Seeking additional support or resources

When you start out, your first port of call should always be the Dynare user manual and then the online Dynare forum. Large language models like ChatGPT and Claude AI can be very useful as well.

Using `DYNARE` for basic analysis

Simulating series

Dynare can easily use the model to simulate series for the endogenous variables. To simulate 200 observations, use option `periods=200` with `stoch_simul(periods=200)`. Dynare will generate 200 shocks of the exogenous variables and use the model solution to generate the endogenous variables.

The simulated endogenous variables are stored in the workspace under the name of the variable. The simulated endogenous variables are also stored under `endo_simul`, in the order of declaration in the .mod file (i.e., in the order in `M_.endo_names`). The generated shocks are stored in `exo_simul`, in the order of declaration (ie in the order in `M_.exo_names`).

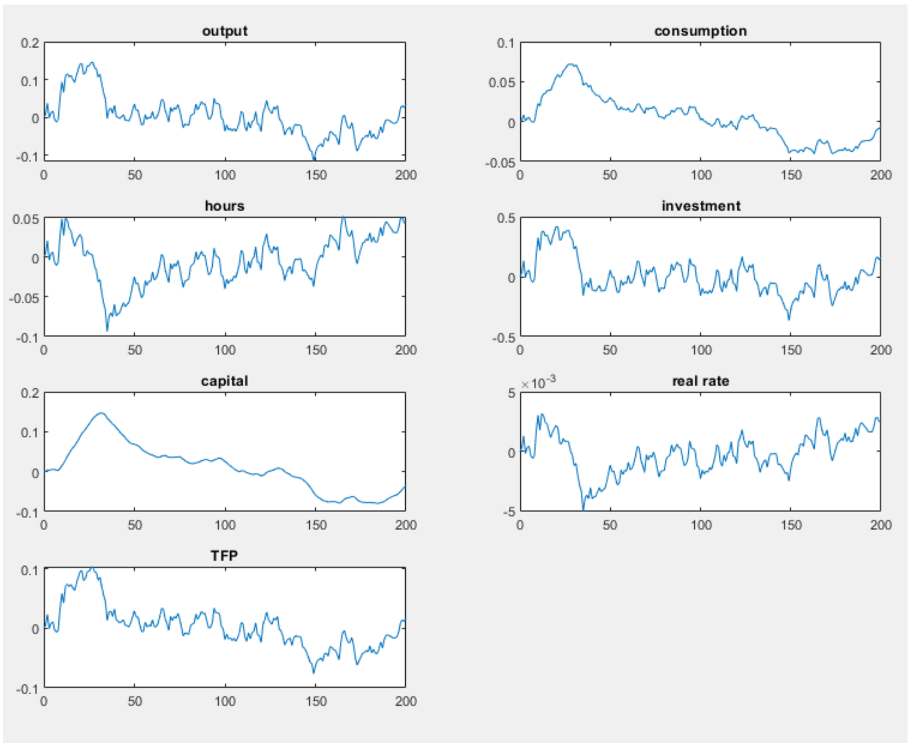
The moments, correlations and autocorrelations are calculated from the simulated data. The default number of points (burnin) dropped at the beginning of simulation before computing the summary statistics is 100. Note that this option does not affect the simulated series stored in `oo_.endo_simul` and the workspace. The number of periods to be dropped can be set with `drop = INTEGER`. The number of periods needs to be larger than 100.

Importantly, newer versions of Dynare set the random number generator seed by default, so the same simulated variables are generated with each run. A different seed can be manually set with the command `set_dynare_seed(x)` where `x` is an integer, declared before `stoch_simul` in the `.mod` file.

The code below plots the simulated series in subplots:

```
periods = 200;
```

```
endovar = oo_.endo_simul;
num_vars = size(endovar, 1); % Number of variables
num_cols = 2;
num_rows = ceil(num_vars / num_cols); % Number of rows
needed

figure
for i = 1:num_vars
    subplot(num_rows, num_cols, i); % Create subplot
    plot(0:periods, [0, endovar(i, :)]);
    title(M_.endo_names_long{i}); % Add title with the
variable name
end
```



To combine all the series in one graphic use:

```
figure
for i=1:M_.endo_nbr
plot(0:periods,[0, endovar(i,:)])
hold on
end
legend(M_.endo_names_long);
hold off
```

Looping over `stoch_simul`

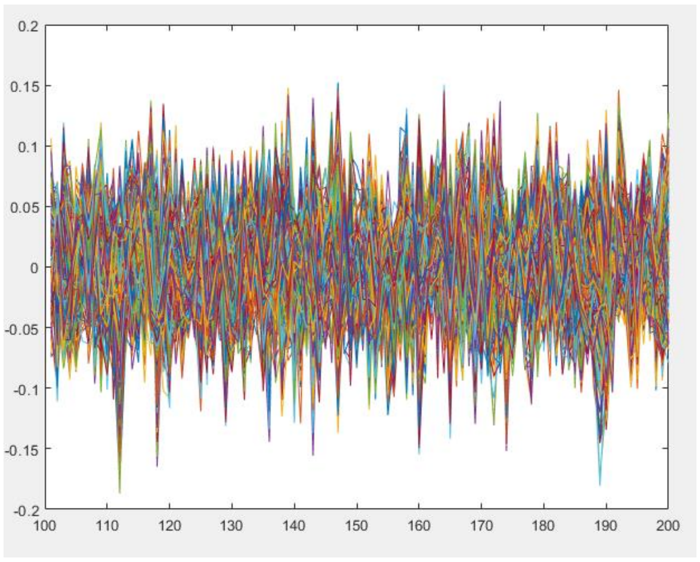
Typically, we want to simulate many series from the same model, or to see how a model's analytics react to changes in parameter values. A natural way to do such analysis is with a loop control structure. Dynare allows us to combine MATLAB's control structures with Dynare's functions directly in the Dynare .mod file, after we define the model.

For example, suppose we wanted to simulate 200 series with 200 observations of output y_t from the RBC model. That is, we would want to run `stoch_simul` 200 times and save the simulated series in a matrix where the column number represents the simulation run. We use the following block of code:

```
for i=1:200
set_dynare_seed(i);
stoch_simul(order=1, periods=200, noprint, nograph);
y_sim(:,i) = oo_.endo_simul(1,:).';
end
```

`i` here is a looping variable which we use to set different seeds and index the `y_sim` matrix. Beginning at `i=1`, solve the model and simulate the endogenous variables for 200 periods by `stoch_simul`, and store `y` in the first column of `y_sim` and. After doing this, it increases `i` by one and repeats, until `i=200`. For speed reasons, the `noprint` and `nograph` options for `stoch_simul` suppresses graphs of impulse responses and the printing of model information to the command window.

```
figure
plot([101:200], y_sim(:,101:200))
hold off
```



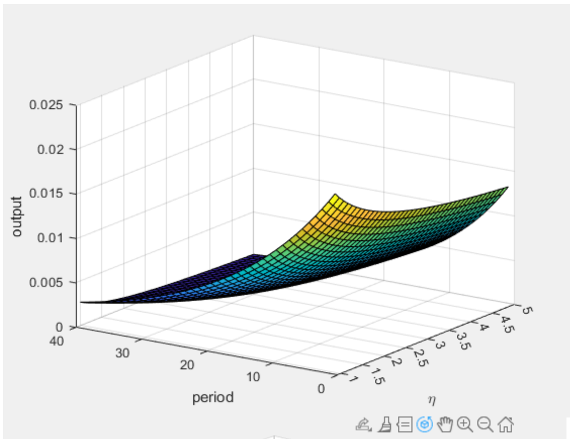
To loop over a range of possible parameter values (where we obtain unique solutions), we use the same strategy as for simulating data. Suppose we want to see how the impulse response of output to a technology shock changes across the degree of household risk aversion η (i.e., the inverse intertemporal elasticity of consumption). That is, we want to analyse `y_epsilon` across different values of `eta`. We define a vector of possible `eta` parameter values and use the looping strategy we used for the simulation above:

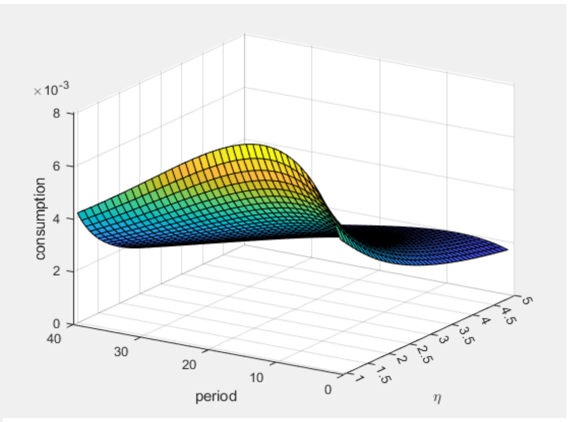
```
eta_params = 1:0.1:5;
for i=1:length(eta_params)
eta = eta_params (i);
stoch_simul(noprint, nograph);
y_epsilon_mat(:,i) = oo_.irfs.y_epsilon.';
end
```

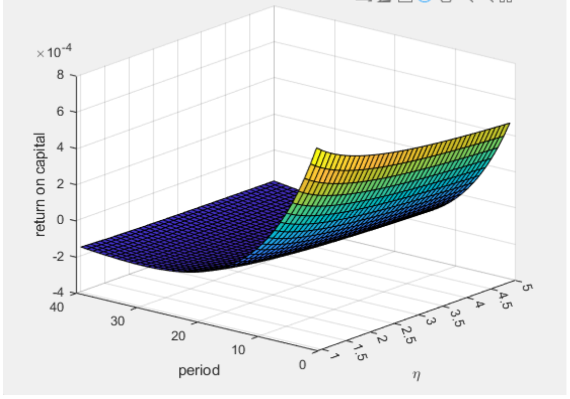
The possible `eta` parameters are stored in `eta_params` and includes values from 0 to 5 in steps of 0.1, giving a 1x41 vector where length(`eta_params`) = 41. We use the looping variable to index the vector of possible psi parameters and to index to a corresponding column in a matrix which stores the impulse responses. The `eta` parameter is chosen from the vector, `stoch_simul` is called, and the impulse response `y_epsilon` stored.

To strengthen the intuition behind the impulse responses, it is useful to graphically view the effect of changes in parameter values over the simulated horizon by creating a three-dimensional graph. We can plot the matrix of IRFs in using the following code:

```
label_interval = 5;
label_indices = 1:label_interval:length(eta_params);
label_values = eta_params(label_indices);
figure
surf(y_epsilon_mat(1:options_.irf, :));
xlabel('\eta'); ylabel('period'); zlabel('y');
set(gca,     'XTick',     label_indices,     'XTickLabel',
label_values);
hold off
```







This will plot the magnitude of each value in the matrix of stored impulse responses across 40 time periods for every parameter value in `eta_params`. The resulting graph is shown in the Figure below. In terms of the model interpretation, it shows that the response of output to a temporary technology shock is higher when consumption is more responsive to the real rental rate of capital (i.e., the marginal product of capital). In other words, for lower values of `eta` (i.e., higher intertemporal elasticities of consumption), a positive return on capital raises future consumption, and therefore output, by larger amounts.

Estimation with DYNARE

Dynare is very useful for estimating parameters and shocks with data. This section shows how to estimate a generic model. The `NT-DSGE` applications section goes into more detail. The first thing to note is that the number of data series cannot be greater than the number of shocks in the model that you want to estimate. For an excellent guide to bringing data to the model, see Johannes Pfeifer's "A Guide to Specifying Observation Equations for the Estimation of DSGE Models". Johannes also provides "An Introduction to Graphs in Dynare" that is a must read for interpreting output from estimation, amongst other things.[7]

Bayesian

Before we run an estimation using observed data, we must first declare the variables of the model which are observed. This is done using the `varobs VAR;` command. For our RBC model presented earlier, it is sensible to use output growth as an observed series:

```
    varobs y_obs;
```

For our example, we need to create the new variable `y_obs` with an accompanying measurement equation so that the number of endogenous variables equals the number of equations: `y_obs = y - y(-1) + y_ss;`. Notice that `y_obs` is demeaned (`y_ss`) so that the steady state growth rate in the code is zero. We need to provide a value for this constant (parameter) too.

Next, we specify the parameters to be estimated in the estimated params block. All variables that are not estimated need to be given values in the parameter setup section of the .mod file.

Since we are using Bayesian techniques, we need to specify prior distributions for each parameter to be estimated. When writing the estimated params section, each parameter needs to be given a prior distribution. The syntax for this is `parameter, prior_shape, prior_mean, prior_stddev [, p_3] [, p_4];`, where the

[7] We do not cover maximum likelihood estimation since it is not as commonly used.

parts in the square brackets are optional. For the standard errors of the shocks, we use the same syntax for specifying the shocks, preceding the exogenous variable with `stderr`. Table: Bayesian Priors details the distributions, and to what parameters the Dynare declaration refer:

Table: **Bayesian Priors**

Shape	Distribution
normal_pdf	$N(\mu,\sigma)$
gamma_pdf	$G(\mu,\sigma,p_3)$
beta_pdf	$B(\mu,\sigma,p_3,p_4)$
inv_gamma_pdf	$IG(\mu,\sigma)$
uniform_pdf	$U(p_3,p_4)$

These distributions have very different shapes and can depend a lot on what parameters are chosen. To illustrate this, the Figure below shows the five distributions across different parameter choices. It is important to think about the distribution of the prior you want for your estimated parameters.

The example of the estimated params section is:

```
estimated_params ;
     eta, inv_gamma_pdf, 0.5, 1;
     psi, beta_pdf, 0.7, 0.1;
     stderr epsilon, inv gamma pdf, 0.01, inf;
end ;
```

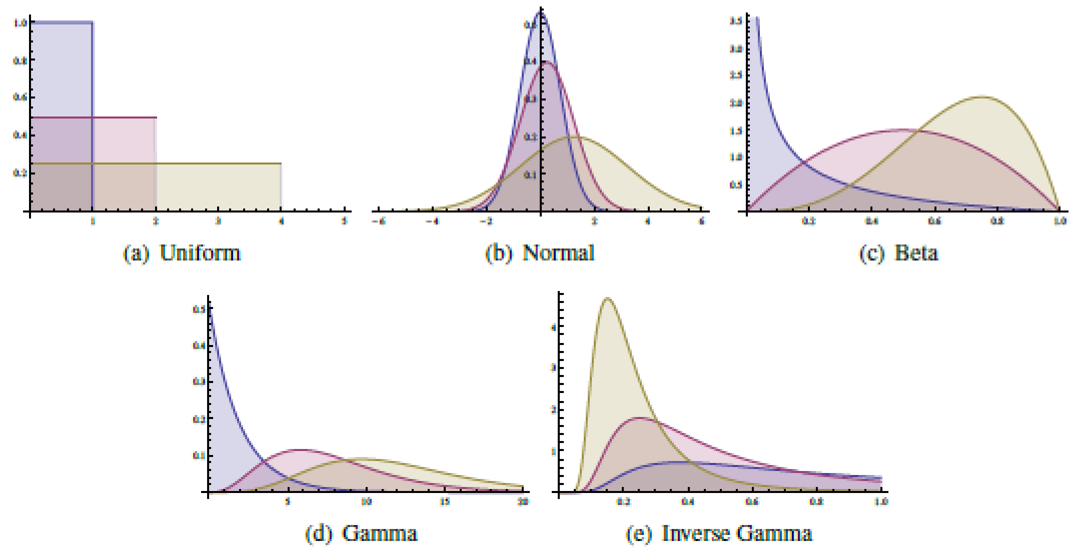


Figure: **Distributions**

Finally, we declare the estimation command and specify where Dynare can find the file containing the observed series:

```
estimation(datafile=data);
```

There are lots of options for Bayesian estimation which can go here. It is worth reviewing the Dynare Manual for an overview of these options for what they mean, along with resources on Bayesian estimation.

The accompanying code basicRBC_est.mod runs the following options using South African data for output growth over the period 1995Q to 2019Q4:

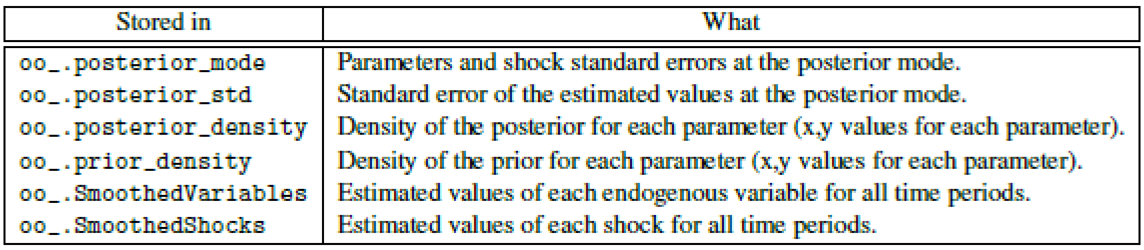
```
estimation(datafile=data9519, mh_nblocks=3,
mh_replic=50000, mh_drop=0.5, mh_jscale=0.99,
mode_compute=5, mode_check, bayesian_irf, irf=40,
graph_format=(fig,pdf)) y_obs y c n v k r z ;
```

The estimation will first find the mode for the parameter values that maximise the data, which is used to initiate the Bayesian estimation process using the Markov Chain Monte Carlo algorithm (specifically the Random-Walk Metropolis-Hastings algorithm). Along with the output produced in the command window (saved in the log file), Dynare will output figures for the prior distributions, the posterior density for values around the computed mode, the univariate and multivariate convergence diagnostics for Metropolis-Hastings, the posterior distributions, the impulse response function to the orthogonalized shock, and the smoothed (historical) series for the shock and observed variable.

Output in workspace

The estimation results are saved in the oo structure in the workspace. Some details are given in the Table below:

Table: **The MATLAB Workspace – Estimation Results**



Stored in	What
`oo_.posterior_mode`	Parameters and shock standard errors at the posterior mode.
`oo_.posterior_std`	Standard error of the estimated values at the posterior mode.
`oo_.posterior_density`	Density of the posterior for each parameter (x,y values for each parameter).
`oo_.prior_density`	Density of the prior for each parameter (x,y values for each parameter).
`oo_.SmoothedVariables`	Estimated values of each endogenous variable for all time periods.
`oo_.SmoothedShocks`	Estimated values of each shock for all time periods.

Parameter values at the posterior mean are found at `oo_.posterior_mean`. Impulse response functions are found at `oo_.PosteriorIRF.dsge`.

Shock decomposition

When more than one shock is specifies and estimated, it is useful to understand how these shocks contribute to the historical dynamics at each point in time. `shock_decomposition` decomposes the movements in each endogenous variable in the model to the estimated shocks. It is called by:

```
shock_decomposition ;
```

For larger models, we may want to plot shocks in groups using, for example:

```
shock_groups(name=groups);
'Demand' = epsilon_i, epsilon_v, epsilon_g;
'Supply' = epsilon_z, epsilon_p, epsilon_w;
end;
```

And then declaring it as an option in `plot_shock_decomposition(…, use_shock_groups=groups,…)`. It is also useful to declare the figure file type as an option, so that you can edit graphics directly in matlab: `graph_format=(fig)`.

Forecasting with DYNARE

On a calibrated model, forecasting is done using the `forecast;` command following `stoch_simul;`. On an estimated model, use the `forecast` option of `estimation` command.

It is also possible to compute forecasts on a calibrated or estimated model for a given constrained path of the future endogenous variables. This is done, from the reduced form representation of the DSGE model, by finding the structural shocks that are needed to match the restricted paths. Use `conditional_forecast`, `conditional_forecast_paths` and `plot_conditional_forecast` for that purpose.

Finally, it is possible to do forecasting with a Bayesian VAR using the bvar_forecast command.

Some relevant options: `periods = x` generates forecasts for `x` periods, and `nograph` suppresses the graph outputs. These options overlap with the `stoch_simul` options, so if no options are specified, the matching options specified in `stoch_simul` apply.

The output of the forecast is stored in `oo .forecast.Mean`. If variables are not simulated, `forecast` just generates forecasts of the variables from steady-state – that is, they just stay at steady-state. For more information on forecasting, see Section 4.20 in the Dynare manual.

Optimal policy with DYNARE

Dynare has tools to compute optimal policies for various types of objectives. You can either solve for optimal policy under commitment with `ramsey_model`, for optimal policy under discretion with `discretionary_policy` or for optimal simple rules with `osr` (also implying commitment).

For our purposes, we rely on optimal simple policy rules for linear-quadratic problems, which is computed using the `osr(OPTIONS...) [VARIABLE_NAME...];`

command. The linear quadratic problem consists of choosing a subset of model parameters to minimize the weighted (co)-variance of a specified subset of endogenous variables, subject to a linear law of motion implied by the first order conditions of the model. This command `osr_params PARAMETER_NAME...;` declares the parameters to be optimized. Weights can be applied to the target endogenous variables to be minimized in the quadratic objective function and are specified in the `optim_weights` block.

Optimising a Taylor rule for monetary policy is a typical example of computing an optimal simple rule. Here is an example:

```
model(linear);
...
r = gamma_rho * r(-1) + gamma_y * y + gamma_pii * pii;
...
end;

optim_weights;
pii 1;
y 0.5;
end;
osr_params gamma_y gamma_pii;
osr y;
```

We can also declare lower and upper bounds for parameters in the optimal simple rule using the following block:

```
osr_params_bounds;
gamma_inf_, 0, 2.5;
end;
osr(opt_algo=9) y;
```

Note that the use of this block requires the use of a constrained optimizer. If no bounds are specified, the optimization is unconstrained.

The value of the objective is stored in the variable `oo_.osr.objective_function` and the value of parameters at the optimum is stored in `oo_.osr.optim_params`.

A useful feature is that after running `osr` the parameters entering the simple rule will be set to their optimal value so that subsequent runs of `stoch_simul` will be conducted at these values.

Sensitivity and identification analysis with DYNARE

The command `identification(OPTIONS...);` triggers identification analysis. A typical example for an analysis on an estimated model is:

```
identification(advanced=1);
```

The parameter set to be evaluated can be selected using `parameter_set = posterior_mean`.

See section 4.22 on sensitivity and identification analysis for more details and sources on the various analyses.

Epilogue variables

```
epilogue ;
...
end ;
```

The epilogue block is useful for computing output variables of interest that may not be necessarily defined in the model (e.g. various kinds of real/nominal shares or relative prices, or annualized variables out of a quarterly model). It can also provide several advantages in terms of computational efficiency and flexibility.

Other useful actions

`DYNARE` macro processing language

Dynare can parse commands written in the Dynare macro-language. This is useful for performing tasks such as: including modular source files, replicating blocks of equations through loops, conditionally executing some code, writing indexed sums or products inside equations, simplifying repetitive code or for automatically including blocks of text in the `.mod` file.

The Dynare macro-language provides a new set of macro-commands which can be used in .mod files. It features:

- File inclusion
- Loops (`for` structure)
- Conditional inclusion (`if/then/else` structures)
- Expression substitution

This macro-language is totally independent of the basic Dynare language and is processed by a separate component of the Dynare pre-processor. The macro processor transforms a `.mod` file with macros into a .mod file without macros (doing expansions/inclusions), and then feeds it to the Dynare parser.

The macro processor is invoked by placing macro directives in the .mod file. Directives begin with an at-sign followed by a pound sign (@#). They produce no output but give instructions to the macro processor. In most cases, directives occupy exactly one line of text. If needed, two backslashes (\\) at the end of the line indicate that the directive is continued on the next line. For historical reasons, directives in commented blocks, ie surrounded by /* and */, are interpreted by the macro processor. The user should not rely on this behaviour. The main directives are:

- `@#includepath`, paths to search for files that are to be included,
- `@#include`, for file inclusion,
- `@#define`, for defining a macro processor variable,
- `@#if, @#ifdef, @#ifndef, @#elseif, @#else, @#endif` for conditional statements,
- `@#for, @#endfor` for constructing loops.

The macro processor maintains its own list of variables (distinct from model variables and MATLAB/Octave variables). These macro-variables are assigned using the `@#define` directive and can be of the following basic types: boolean, real, string, tuple, function, and array (of any of the previous types). For more details see Section 4.27 in the Manual.

Examples

```
@#include "modelcomponent.mod"
```

This directive simply includes the content of another file in its place; it is exactly equivalent to a copy/paste of the content of the included file.

`Modelcomponent.mod` should use the regular Dynare language conventions. This is particularly useful if you want to to split .mod files into several modular components (modularization) or change certain blocks of the model code, like the foreign economy or the fiscal authority. If you want to change variables, parameters, calibrations etc, this can also be performed by the `@#include` command.

We can also automate, for example, equation selection by using `if-else` statements as follows:

```
@#define linear_mon_pol = false // 0 would be treated the
same
...
model;
@#if linear_mon_pol
i = w*i(-1) + (1-w)*i_ss + w2*(pie-piestar);
@#else
i = i(-1)^w * i_ss^(1-w) * (pie/piestar)^w2;
@#endif
...
```

```
end;
```

This would result in:

```
...
model;
i = i(-1)^w * i_ss^(1-w) * (pie/piestar)^w2;
...
end;
```

A particularly valuable time-saving use of `@#include` is to write estimated parameter values to a `.mod` file and include these in the parameter declaration of the primary `.mod` file. This means we do not have to manually put in estimated parameter values into a new file if we wanted to do analysis with the estimated parameters.

For example, suppose we write a `.m` file which calls Dynare `basicRBC_estimation.mod`. Then we can write a section of code which writes the parameters, as a string, to a .mod file called `parameters.mod`. We can then use `@#include parameters.mod` in the parameter setup section of `basicRBC_estimation.mod` and call it to Dynare.

We can use the `savemacro` option in Dynare to save the `basicRBC_estimation.mod` file with the macro language elements expanded. The saved `.mod` file is called `basicRBC_estimation-macroexp.mod`.

`@#for` or `@#define` we add the macro-variable using `@{…}`. Some examples:

Example 1

```
model;
@#for country in [ "home", "foreign" ]
GDP_@{country} = A * K_@{country}^a * L_@{country}^(1-a);
@#endfor
end;
```

Which is equivalent to:

```
model;
GDP_home = A * K_home^a * L_home^(1-a);
GDP_foreign = A * K_foreign^a * L_foreign^(1-a);
end;
```

Example 2

```
model;
@#for (i, j) in ["GDP"] * ["home", "foreign"]
@{i}_@{j} = A * K_@{j}^a * L_@{j}^(1-a);
@#endfor
end;
```

The latter is equivalent to:

```
model;
GDP_home = A * K_home^a * L_home^(1-a);
GDP_foreign = A * K_foreign^a * L_foreign^(1-a);
end;
```

Example

```
@#define countries = ["US", "FR", "JA"]
@#define nth_co = "US"
model;

@#for co in countries when co != nth_co
(1+i_@{co}) = (1+i_@{nth_co}) * E_@{co}(+1) / E_@{co};
@#endfor
E_@{nth_co} = 1;
end;
```

The latter is equivalent to:

```
model;
(1+i_FR) = (1+i_US) * E_FR(+1) / E_FR;
(1+i_JA) = (1+i_US) * E_JA(+1) / E_JA;
E_US = 1;
end;
```

Saving data to an `.m` file

If we want to save a vector or matrix to a MATLAB script (`.m`) file, we can use Dynare's command `datatomfile()`. This might be helpful if we want to use simulated data to estimate a model, for instance. For example, in the following line, Dynare will save the three vectors of simulated data g, pi, and r to data.m in the working directory.

```
datatomfile('data',{'g';'pi';'r'})
```

Writing LATEX from DYNARE

Dynare has in-built functions to easily print the model equations and estimation results to a LATEX file. For it to know what LATEX variables to use, you need to define the LATEX representation of each variable and parameter. This occurs in the declaration section, after each variable, in dollar signs. For example, for a selection of variables or parameters, we use:

```
var y ${y}$ (long_name='output') … ;
parameters beta $\beta$ (long_name=discount factor) … ;
```

And in the model block we can use:

```
    [name='Resource constraint']
    y = CY*c + delt*KY*v;
```

After declaring LATEX equivalent variables and parameters, at the end of the Dynare model code, use the command:

```
    write_latex_dynamic_model ;
```

to generate a .tex file with name MODELNAME_latex_dynamic.tex where MODELNAME is the name of the .mod file. This .tex file can be compiled, or the important lines copied into another LATEX file. We can also collect other output using:

```
    write_latex_static_model ;
    write_latex_dynamic_model;
    write_latex_parameter_table;
    write_latex_definitions;
    %% write_latex_prior_table; // for estimation

    collect_latex_files;
    if system(['pdflatex -halt-on-error -
    interaction=batchmode ' M_.fname '_TeX_binder.tex'])
        error('TeX-File did not compile.')
```

NT-DSGE applications

By now you should be able to do the following:

- Step-by-step instructions for executing a simulated or an estimated model written in Dynare syntax.
- Monitor the simulation or estimation progress and interpret the output.
- Understand model output files and formats and how to retrieve this information.
- Visualise and interpret results such as impulse response functions and variance decompositions.
- Conduct sensitivity analyses and robustness checks.

The section below briefly describes alternative applications of the NT-DSGE model that have been conducted for policy analysis. In each case, the relevant paper is cited.

Fiscal multipliers

An application using impulse response functions.

What is the impact of fiscal policy decisions on macroeconomic outcomes? This impact is summarised using present-value fiscal multipliers. A fiscal multiplier measures the impact of government's tax and spending decisions on economic output. Fiscal multipliers are calculated based on identified impulse response functions (IRFs) to fiscal policy innovations (i.e., estimated exogenous shocks).

For example, a present-value multiplier for real GDP (output) is calculated as the ratio of the discounted output response to the discounted government spending response. That is:

Present-value multiplier at horizon k $= \frac{\sum_{j=0}^{k}(1+r)^{-j}y_{t+j}}{\sum_{j=0}^{k}(1+r)^{-j}f_{t+j}}\frac{1}{f/y}$

where y_{t+j} is the response of GDP at period j, f_{t+j} is the response of the fiscal variable at period j, and r is the model consistent nominal policy interest rate. The responses are scaled by f/y (the ratio of the fiscal variable to real GDP evaluated at the sample mean). The tax response is measured as the response in total tax *revenue* (that is, when investigating the impact of unanticipated tax shocks, f is equal to labour, capital and consumption tax *revenue*).

Present-value fiscal multipliers under the scenario when all fiscal variables respond to debt and output are presented in the Table below.

Table 1: Present-value multipliers for government consumption, government investment, and taxes.

Variable	$Q1$	$Q4$	$Q8$	$Q20$	∞
Government consumption multiplier					
$\Delta Y/\Delta G$	-0.04	0.19	0.21	0.20	0.20
$\Delta C/\Delta G$	-0.82	-0.72	-0.64	-0.55	-0.55
$\Delta I/\Delta G$	-0.07	-0.08	-0.10	-0.12	-0.11
Government investment multiplier					
$\Delta Y/\Delta I_G$	0.56	0.62	0.57	0.51	0.51
$\Delta C/\Delta I_G$	-0.04	-0.07	-0.09	-0.12	-0.12
$\Delta I/\Delta I_G$	-0.05	-0.07	-0.09	-0.09	-0.09
Labour tax multiplier					
$\Delta Y/\Delta T_w$	-0.42	-0.28	-0.13	-0.04	-0.04
$\Delta C/\Delta T_w$	-0.62	-0.75	-0.88	-0.91	-0.91
$\Delta I/\Delta T_w$	0.00	-0.03	-0.07	-0.10	-0.10
Capital tax multiplier					
$\Delta Y/\Delta T_k$	-0.15	0.10	0.36	0.49	0.49
$\Delta C/\Delta T_k$	-0.04	-0.04	-0.11	-0.17	-0.18
$\Delta I/\Delta T_k$	-0.03	-0.13	-0.22	-0.31	-0.31
Consumption tax multiplier					
$\Delta Y/\Delta T_c$	-0.19	-0.04	0.14	0.24	0.24
$\Delta C/\Delta T_c$	-0.19	-0.41	-0.60	-0.68	-0.68
$\Delta I/\Delta T_c$	-0.01	-0.05	-0.09	-0.13	-0.13

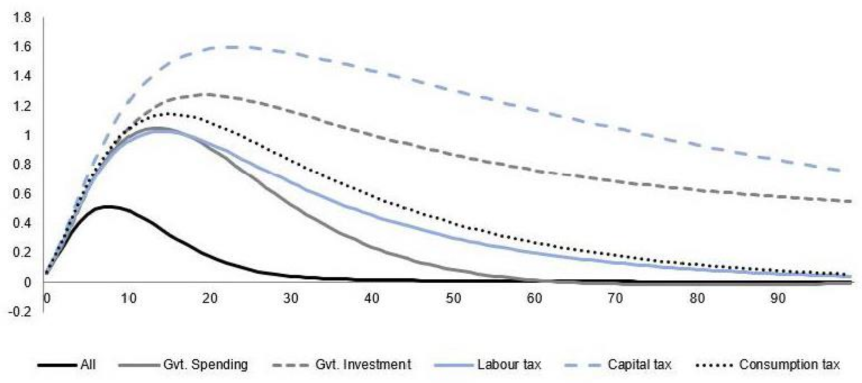
Notes: All instruments respond endogenously to debt and output. Variables in column 1 represent the change in output, consumption, or investment in response to the fiscal instrument. See: Kemp and Hollander (2020).

The fiscal multipliers can be printed out in the matlab command window using an .m file found in ~\ntdsge/5.matlab_code/print_multipliers/. The results for the estimated model must be loaded into the workspace (found specifically in `oo_.PosteriorIRF.dsge.Mean`). The code can be easily adjusted to calculate multipliers based on simulations (found specifically in `oo_.irfs`)

As an alternative example we can use the estimated version of the model to run counterfactual simulations of IRFs. In the figure below, we use the DSGE model to assess fiscal policy instruments for debt stabilization. The figure plots the percentage point response of government debt (vertical axis) over quarterly periods (horizontal axis) to a one standard deviation innovation in government consumption spending (simulating deficit spending). Each line represents the impulse response if only that specific fiscal instrument adjusts to changes in government debt— a hypothetical counter-acting fiscal policy response (simulating an attempt to finance the deficit created). The results indicate how quickly total debt will stabilize, depending on the type of fiscal policy instrument used to finance it (a spending cut or tax increase). In the code, this implies setting all coefficients that respond to government debt in the fiscal reaction functions to zero except for the instrument of interest. “All” shows the IRF for the estimated model when all instruments adjust.

The results show that the most effective tool for debt stabilization is to reduce government consumption spending, followed by increases to labour and consumption taxes. But because the multiplier results (shown previously) indicate that increased taxes have a significant negative impact on welfare, adjusting government consumption spending is a better tool for debt management.

Figure: Debt-stabilization dynamics



Finally, one can also use the NT-DSGE model to trace out the most important transmission mechanisms of fiscal policy on the real economy and how monetary policy endogenously responds to fiscal shocks. Such an exercise is carried out in Hollander (2024) "Debt-financed fiscal stimulus in South Africa," Studies in Economics and Econometrics, Taylor & Francis Journals, vol. 48(1), pages 87-112, January. DOI: 10.1080/03796205.2024.2321628

Policy counterfactuals

An application using optimal simple rules.

We can use the `osr` function in Dynare to compute the optimal parameter values that minimize variation in a specified set of target variables in response to any number of shocks. Such an exercise is carried out in Havemann & Hollander (forthcoming). "Fiscal policy in times of fiscal stress: Or what to do when r > g," Journal of Policy Modeling.

In the NT-DSGE model, fiscal instruments follow simple reaction functions (‘’feedback rules’’) with four features. First, a first-order autoregressive component captures the persistence of each instrument j (ϕ_j). Second, each instrument responds contemporaneously to deviations of output from its steady-state level (y_t), thereby controlling for automatic stabilizer effects ($\theta_{j,y}$). Third, all fiscal instruments are permitted to respond to deviations of government debt, in real terms, from its steady-state level (b_t), thereby controlling for public debt stabilization ($\theta_{j,b}$). Fourth, a stochastic component identifies the exogenous (i.e. discretionary or unanticipated) changes in the instruments (ε_t^j). These linear fiscal reaction functions are consistent with the idea that debt stabilization is an important consideration in the formulation of fiscal policy and can approximate optimal rules for debt stabilization.

We focus our attention on government consumption, $c_{G,t}$, and government investment, $k_{G,t}$, expenditure reaction functions, given by

$$c_{G,t} = \phi_{cG} c_{G,t-1} - \theta_{cG,y} y_t - \theta_{cG,b} b_t + \varepsilon_t^{cG}$$

$$i_{G,t} = \phi_{iG} i_{G,t-1} - \theta_{iG,y} y_t - \theta_{cG,b} b_t + \varepsilon_t^{cG}$$

The success of policy is therefore measured by its ability to minimize instability in the target variables. Policymakers must choose their respective θs and ϕs to minimize a quadratic welfare loss function $\mathcal{L}_t \rightarrow 0$:

$$\min \mathcal{L}_t = y_t^2 + \Theta_{\mathbb{Z}}.\mathbb{Z}_t^2$$

where the welfare loss ($\mathcal{L}_t$) is an increasing function of deviations to output (y_t) and one or more variables in the vector $\mathbb{Z}$. $\Theta_{\mathbb{Z}}$ is a vector of weights w corresponding to the policy target variables.

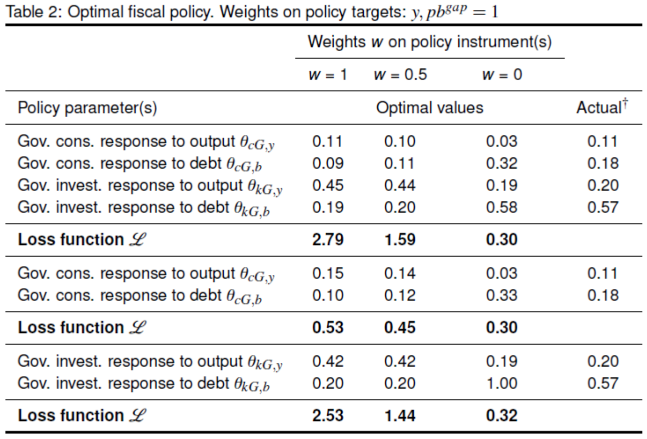
For the fiscal authority we consider output (y_t) and debt (b_t) or output and the fiscal sustainability gap (pb_t^{gap}) as the targets (see the cited paper for more details). Given that instability in the policy instrument is undesirable, $\Theta_{\mathbb{Z}}$ also controls for variation in the set of policy instruments $\{c_{G,t}, i_{G,t}\}$.

An example of the code following the declaration of the shocks is:

```
optim_weights;

%%%% Goals %%%%
y 1; // output
b 1; // government debt (real)

%%%% Policy variable(s) %%%%
g 0; // government spending

end;

%%%% Instrument(s) %%%%
osr_params  thetaGY thetaGB ;

%%%% Compute OSR and generate IRFs %%%%
osr(order=1, irf=40, graph_format=fig, nodisplay) b y piY ;
```

The results for optimal macroeconomic and fiscal stabilisation rules are reproduced from the cited paper in the tables below.

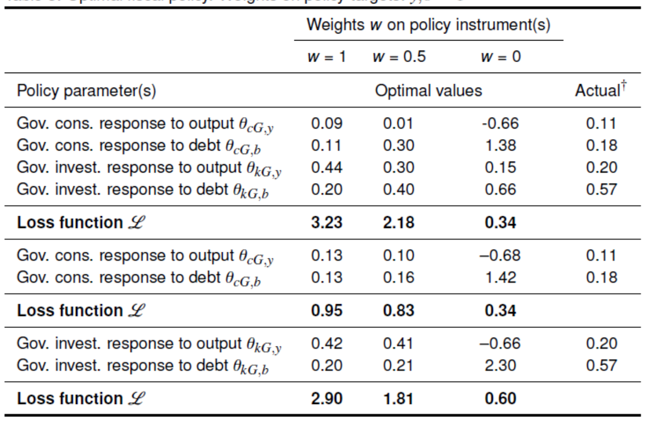
Table 2: Optimal fiscal policy. Weights on policy targets: $y, pb^{gap} = 1$



	Weights *w* on policy instrument(s)			
	w = 1	*w* = 0.5	*w* = 0	
Policy parameter(s)	Optimal values			Actual†
Gov. cons. response to output $\theta_{cG,y}$	0.11	0.10	0.03	0.11
Gov. cons. response to debt $\theta_{cG,b}$	0.09	0.11	0.32	0.18
Gov. invest. response to output $\theta_{kG,y}$	0.45	0.44	0.19	0.20
Gov. invest. response to debt $\theta_{kG,b}$	0.19	0.20	0.58	0.57
Loss function $\mathscr{L}$	**2.79**	**1.59**	**0.30**	
Gov. cons. response to output $\theta_{cG,y}$	0.15	0.14	0.03	0.11
Gov. cons. response to debt $\theta_{cG,b}$	0.10	0.12	0.33	0.18
Loss function $\mathscr{L}$	**0.53**	**0.45**	**0.30**	
Gov. invest. response to output $\theta_{kG,y}$	0.42	0.42	0.19	0.20
Gov. invest. response to debt $\theta_{kG,b}$	0.20	0.20	1.00	0.57
Loss function $\mathscr{L}$	**2.53**	**1.44**	**0.32**	

Note: the table presents the optimal values of a set of policy parameters under our 'soft' adjustment scenario. The optimal values minimize the loss function. We vary the weights on the two instruments. † Estimated parameter value based on historical experience.

Table 3: Optimal fiscal policy. Weights on policy targets: $y, b = 1$



	Weights *w* on policy instrument(s)			
	w = 1	*w* = 0.5	*w* = 0	
Policy parameter(s)	Optimal values			Actual†
Gov. cons. response to output $\theta_{cG,y}$	0.09	0.01	-0.66	0.11
Gov. cons. response to debt $\theta_{cG,b}$	0.11	0.30	1.38	0.18
Gov. invest. response to output $\theta_{kG,y}$	0.44	0.30	0.15	0.20
Gov. invest. response to debt $\theta_{kG,b}$	0.20	0.40	0.66	0.57
Loss function $\mathscr{L}$	**3.23**	**2.18**	**0.34**	
Gov. cons. response to output $\theta_{cG,y}$	0.13	0.10	–0.68	0.11
Gov. cons. response to debt $\theta_{cG,b}$	0.13	0.16	1.42	0.18
Loss function $\mathscr{L}$	**0.95**	**0.83**	**0.34**	
Gov. invest. response to output $\theta_{kG,y}$	0.42	0.41	–0.66	0.20
Gov. invest. response to debt $\theta_{kG,b}$	0.20	0.21	2.30	0.57
Loss function $\mathscr{L}$	**2.90**	**1.81**	**0.60**	

Note: the table presents the optimal values of a set of policy parameters under our 'hard' adjustment scenario. The optimal values minimize the loss function. We vary the weights on the two instruments. † Estimated parameter value based on historical experience.

We notice again that government consumption expenditure is the most effective instrument for debt stabilisation.

Forecasts

An application using conditional forecasts.

Conducting conditional forecasts can be very useful for understanding how current policy actions will play out given alternative policy scenarios. The conditional_forecast command computes forecasts on an estimated or calibrated model for a given constrained path of some future endogenous variables.

In our NT-DSGE example, we declare the forecast option in the estimation command. Note that we can shut off the estimation of the model if it has been previously estimated, which allows us to forecast alternative scenarios for a given conditional forecast. For example, we may want to see how the economy will adjust if

In our NT-DSGE example, we start by declaring the forecast option, forecast, in the estimation command (we compute over a 40-period horizon).[8] The forecast will start from the final observations in *histval*.[9] Note that we can shut off the estimation of the model if it has been previously estimated (mh_replic=0, mode_compute=0), which allows us to forecast alternative scenarios for a given conditional forecast. For example, we may want to see how the economy will adjust under alternative assumptions of fiscal reaction functions or household and firm behaviour. Even if the forecast turns out to be weakly predictive of actual outcomes, the relative effect of the conditional versus the unconditional forecast can assist the policymaker in understanding how their potential interventions may influence their own expectations and internal forecasts.

Below is an adapted example of the code used in Havemann, Hollander, and Steenkamp (2023). "The macroeconomics of establishing a basic income grant in South Africa," South African Journal of Economics, 90(1), https://doi.org/10.1111/saje.12363.

```
estimation(datafile=data_19q4_V2, mh_replic=0,
forecast=40, mode_compute=0, graph_format=fig) b y rrp R
piC by;

conditional_forecast_paths;

var dtr_ ;
periods 1, 2, 3:20, 21:40 ;
values    25, 25, 25, 0 ;

var R_ ;
periods 1, 2, 2:40 ;
values 2.25,  2, 1.5  ;
```

[8] On a calibrated model, forecasting is done using the forecast command. On an estimated model, use the forecast option of estimation command. It is also possible to compute forecasts on a calibrated or estimated model for a given constrained path of the future endogenous variables. This is done, from the reduced form representation of the DSGE model, by finding the structural shocks that are needed to match the restricted paths. Use conditional_forecast, conditional_forecast_paths and plot_conditional_forecast for that purpose.

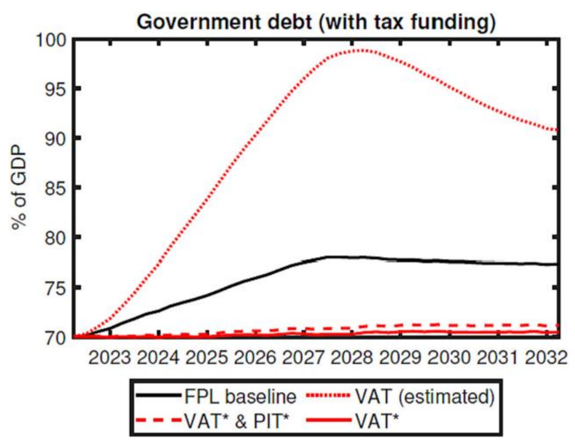
[9] forecast computes the forecast taking as initial values the values specified in *histval* (see Manual). For estimated models the set of smoothed variables are the historical values used to determine the initial value for forecasts. When no histval block is present, the initial values are the one stated in initval. When initval is followed by command steady, the initial values are the steady state (see steady). In other words, for a stoch_simul of NT-DSGE model, forecasts will start at zero. See the Manual for ways to flexible adjust initial and historical values.

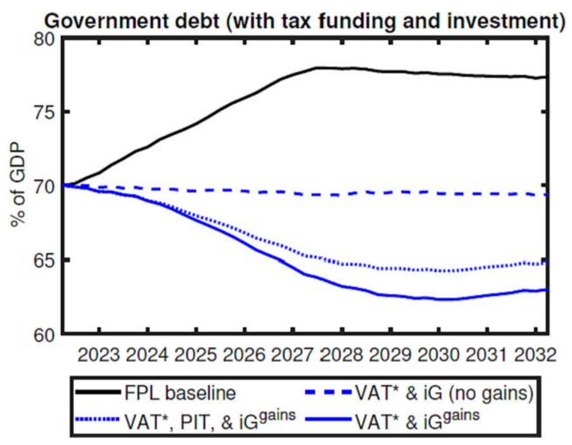
```
end;

conditional_forecast(parameter_set = calibration,
conf_sig = 0.68, controlled_varexo = (etatr, etaR),
replic = 10000); // posterior_mean

plot_conditional_forecast(periods = 40) b y piC;
```

If we want to save a forecast output for comparisons, we can load the results and save a forecast structure in a .mat file:

```
load conditional_forecasts.mat
save BIG_3_foreplots_1.mat forecasts
```

An example of the results from the cited paper are produced in the figure below:





Best Practices

Version control and documentation

The replication code for the baseline (latest) model NT-DSGE model is maintained at: ~/ntdsge. Users should create their own branch in their personal computer but save the folders in a different folder than their GitHub repository path, so that the user can edit files and folders themselves and only commit certain changes to the main branch if they want to update or share their work.

Collaboration and knowledge sharing

This user guide will be treated as a living document and should be improved and refined for the needs of the users. In addition, replication files to reproduce all results from using the NT-DSGE model should be maintained at the GitHub repository for posterity and if new members come into the unit. This will ensure the learning curve is not as steep.

Model validation and quality assurance

The NT-DSGE model should be regularly refined and critiqued for improvement as new data comes available (especially after significant shocks like covid) or if important research questions arise that will likely be significant for policy analysis over the foreseeable (e.g., a basic income grant or fiscal sustainability or fiscal rules).

It is important to note that this baseline model is not readily flexible. A more fit-for-purpose small-scale fiscal DSGE model should be considered for adaptability. As always, one should be mindful of the constraints of the methodology and model framework. For an exposition see, e.g., Hollander and van Lill (2020). The NYFed provides an excellent example of using a DSGE model for policy analysis and forecasting in a consistent manner. The purpose of the NT-DSGE model should become more focused / carefully defined to avoid the error of thinking the model can answer too broad a range of questions (at least in a readily adaptable way).

Model extensions or customisations

This section can be used to describe customised versions of the NT-DSGE model or alternative DSGE models used for policy analysis and forecasting.

References

Links to online resources and documentation

GitHub repository for Training and NT-DSGE resources:
~/hollander03/NationalTreasuryDSGE.

For a bibliography of relevant literature, see readers folder in the training repository.

For an excellent guide to bringing data to the model, see Johannes Pfeifer's "A Guide to Specifying Observation Equations for the Estimation of DSGE Models".

Johannes also provides "An Introduction to Graphs in Dynare" that is a must read for interpreting output from estimation, amongst other things.

The Dynare homepage: www.dynare.org/

The Dynare forum: forum.dynare.org/

The macroeconomic model database: www.macromodelbase.com/

Appendix

The complete `DYNARE` code for the log-linearized RBC model

```
//Basic RBC model

var y ${y}$ (long_name='output')
    c ${c}$ (long_name='consumption')
    n ${n}$ (long_name='hours')
    v ${v}$ (long_name='investment')
    k ${k}$ (long_name='capital')
    r ${r}$ (long_name='real rate')
    z ${z}$ (long_name='TFP');

varexo epsilon ${\epsilon_z}$ (long_name='TFP shock');

parameters rho   ${\rho}$  (long_name='capital share')
           bet   ${\beta}$   (long_name='discount factor')
           delt  ${\delta}$  (long_name='depreciation rate')
           psi   ${\psi}$  (long_name='persistence TFP shock')
           eta   ${\eta}$  (long_name='risk aversion')
           sigmae ${\sigma_{e}}$ (long_name='TFP shock');

rho = 0.33;
bet = 0.99;
delt = 0.025;
psi = 0.95;
eta = 2;
sigmae = 0.01;

model(linear);
// Declare useful model-local variables
#R = 1/bet;
#KY = rho/(R - (1-delt));
#CY = 1 - KY*delt;
// Declare model equations
[name='Resource constraint']
y = CY*c + delt*KY*v;
[name='Euler equation']
(1/eta)*r(+1) = c(+1) - c;
[name='Labor FOC']
eta*c = y - n;
[name='real interest rate/firm FOC capital']
r = rho*(1/KY)*(y - k(-1));
[name='Law of motion capital']
k = (1-delt)*k(-1) + delt*v;
[name='production function']
y = z + rho*k(-1) + (1-rho)*n;
[name='exogenous TFP process']
z = psi*z(-1) + epsilon;
end;

steady;
check;
```

```
shocks;
var epsilon = sigmae^2;
end;
```

Below are different simulation and figure plotting options to select from:

```
%--------------------------------------------------------------
% Theoretical moments (default: periods=0)
%--------------------------------------------------------------

% stoch_simul(order=1, periods=0, nodisplay); // for theoretical
moments

%--------------------------------------------------------------
% Simulate periods. Note: results should tend to the theoretical
moments as the number of simulated periods tends to infinity (or
sufficiently large number)
%--------------------------------------------------------------

stoch_simul(order=1, periods=200, graph_format=fig, nodisplay); // ,
periods=1000, graph_format=(fig, pdf),

%--------------------------------------------------------------
% Plot IRFs to a 1.0% s.d. technology shock
%--------------------------------------------------------------

figure
plot([0:options_.irf], [0 oo_.irfs.y_epsilon]*100)
hold on
plot([0:options_.irf], [0 oo_.irfs.c_epsilon]*100)
plot([0:options_.irf], [0 oo_.irfs.v_epsilon]*100)
plot([0:options_.irf], [0 oo_.irfs.k_epsilon]*100)
plot([0:options_.irf], [0 oo_.irfs.n_epsilon]*100)
plot([0:options_.irf], [0 oo_.irfs.r_epsilon]*100)
plot([0:options_.irf], [0 oo_.irfs.z_epsilon]*100)
title('Impulse responses to a technology shock')
legend('Output','Consumption','Investment','Capital','Employment','R
ate of return','Technology shock')
ylabel('percentage deviations from steady state')
xlabel('quarters after shock')
hold off

%--------------------------------------------------------------
% Create loop to over parameter values. NOTE: this can also be
performed using the macro language (see guide)
%--------------------------------------------------------------

eta_params = 1:0.1:5;
for i=1:length(eta_params)
% eta = eta_params(i);
set_param_value('eta',eta_params(i));
stoch_simul(order=1, periods=200, noprint, nograph);
    if info(1)~=0 // strongly advised to always check the error flag
    error('Simulation failed for parameter draw')
    end
y_epsilon_mat(:,i) = oo_.irfs.y_epsilon.';
c_epsilon_mat(:,i) = oo_.irfs.c_epsilon.';
r_epsilon_mat(:,i) = oo_.irfs.r_epsilon.';
end

%-------------------------------------------------------------
% plot 3D IRFs for loop over range of parameter values
%-------------------------------------------------------------

label_interval = 5;
label_indices = 1:label_interval:length(eta_params);
label_values = eta_params(label_indices);
figure
surf(y_epsilon_mat(1:options_.irf, :));
xlabel('\eta'); ylabel('period'); zlabel('output');
set(gca, 'XTick', label_indices, 'XTickLabel', label_values);
hold off

figure
surf(c_epsilon_mat(1:options_.irf, :));
xlabel('\eta'); ylabel('period'); zlabel('consumption');
set(gca, 'XTick', label_indices, 'XTickLabel', label_values);
hold off

figure
surf(r_epsilon_mat(1:options_.irf, :));
xlabel('\eta'); ylabel('period'); zlabel('return on capital');
set(gca, 'XTick', label_indices, 'XTickLabel', label_values);
hold off

%-------------------------------------------------------------
% Create loop to simulate 200 series with 200 observations for
output
%-------------------------------------------------------------

for i=1:200
set_dynare_seed(i);
stoch_simul(order=1, periods=200, noprint, nograph);
y_sim(:,i) = oo_.endo_simul(1,:).';
end

%-------------------------------------------------------------
% Plot simulated series across all seeds
%-------------------------------------------------------------

figure
plot([101:200], y_sim(:,101:200))
hold off

%-------------------------------------------------------------
% generate LaTeX output
%-------------------------------------------------------------

write_latex_static_model ;
write_latex_dynamic_model;
```

```
write_latex_parameter_table;
write_latex_definitions;
collect_latex_files;
if system(['pdflatex -halt-on-error -interaction=batchmode '
M_.fname '_TeX_binder.tex'])
    error('TeX-File did not compile.')
end
```

Glossary of technical terms for the NT-DSGE model

This section should be populated over time by users as terminology questions arise related to the model and/or code.

FAQs

This section should be populated over time by users as issues arise.