Dynare

Stepan Gordeev

September 10, 2020

Introduction

- Dynare is a toolkit for solving and estimating DSGE models
- Deterministic models (perfect foresight): preserves nonlinearities
- Stochastic models (rational expectations): 1st/2nd-order local approximation around s.s.
    - does not support discrete choice or occasionally binding constraints
    - Occbin: a toolkit to allow occasionally binding constraints in Dynare
        - but the solution is imperfect: agents never expect switching between binding and non-binding regimes
    - $\rightarrow$ if a model has any discontinuities, need regular VFI
- If a model is linearizable, Dynare greatly automates much of the work

- Dynare is a toolkit for solving and estimating DSGE models
- Deterministic models (perfect foresight): preserves nonlinearities
- Stochastic models (rational expectations): 1st/2nd-order local approximation around s.s.
    - does not support discrete choice or occasionally binding constraints
    - Occbin: a toolkit to allow occasionally binding constraints in Dynare
        - but the solution is imperfect: agents never expect switching between binding and non-binding regimes
    - $\rightarrow$ if a model has any discontinuities, need regular VFI
- If a model is linearizable, Dynare greatly automates much of the work

- Dynare is a toolkit for solving and estimating DSGE models
- Deterministic models (perfect foresight): preserves nonlinearities
- Stochastic models (rational expectations): 1st/2nd-order local approximation around s.s.
  - does not support discrete choice or occasionally binding constraints
  - Occbin: a toolkit to allow occasionally binding constraints in Dynare
    - but the solution is imperfect: agents never expect switching between binding and non-binding regimes
  - $\rightarrow$ if a model has any discontinuities, need regular VFI
- If a model is linearizable, Dynare greatly automates much of the work

Introduction

- Dynare is a toolkit for solving and estimating DSGE models
- Deterministic models (perfect foresight): preserves nonlinearities
- Stochastic models (rational expectations): 1st/2nd-order local approximation around s.s.
  - does not support discrete choice or occasionally binding constraints
  - Occbin: a toolkit to allow occasionally binding constraints in Dynare
    - but the solution is imperfect: agents never expect switching between binding and non-binding regimes
  - $\rightarrow$ if a model has any discontinuities, need regular VFI
- If a model is linearizable, Dynare greatly automates much of the work

Introduction

- Dynare is a toolkit for solving and estimating DSGE models
- Deterministic models (perfect foresight): preserves nonlinearities
- Stochastic models (rational expectations): 1st/2nd-order local approximation around s.s.
  - does not support discrete choice or occasionally binding constraints
  - Occbin: a toolkit to allow occasionally binding constraints in Dynare
    - but the solution is imperfect: agents never expect switching between binding and non-binding regimes
  - $\rightarrow$ if a model has any discontinuities, need regular VFI
- If a model is linearizable, Dynare greatly automates much of the work

Introduction

- Dynare is a toolkit for solving and estimating DSGE models
- Deterministic models (perfect foresight): preserves nonlinearities
- Stochastic models (rational expectations): 1st/2nd-order local approximation around s.s.
    - does not support discrete choice or occasionally binding constraints
    - Occbin: a toolkit to allow occasionally binding constraints in Dynare
        - but the solution is imperfect: agents never expect switching between binding and non-binding regimes
    - $\rightarrow$ if a model has any discontinuities, need regular VFI
- If a model is linearizable, Dynare greatly automates much of the work

Introduction

- Dynare is a toolkit for solving and estimating DSGE models
- Deterministic models (perfect foresight): preserves nonlinearities
- Stochastic models (rational expectations): 1st/2nd-order local approximation around s.s.
  - does not support discrete choice or occasionally binding constraints
  - Occbin: a toolkit to allow occasionally binding constraints in Dynare
    - but the solution is imperfect: agents never expect switching between binding and non-binding regimes
  - $\rightarrow$ if a model has any discontinuities, need regular VFI
- If a model is linearizable, Dynare greatly automates much of the work

Introduction

- Dynare is a toolkit for solving and estimating DSGE models
- Deterministic models (perfect foresight): preserves nonlinearities
- Stochastic models (rational expectations): 1st/2nd-order local approximation around s.s.
    - does not support discrete choice or occasionally binding constraints
    - Occbin: a toolkit to allow occasionally binding constraints in Dynare
        - but the solution is imperfect: agents never expect switching between binding and non-binding regimes
    - $\rightarrow$ if a model has any discontinuities, need regular VFI
- If a model is linearizable, Dynare greatly automates much of the work

Installation

- Works through MATLAB
  - available in computer labs and BlueHive, can also download with UR account
  - also works through Octave, an open-source clone of MATLAB
- Install:
  - Download the MATLAB version from `dynare.org/download/`
  - Install to some directory
  - In MATLAB, go to Set Path → Add Folder... → select the `matlab` subdirectory of the Dynare installation → save

Installation

- Works through MATLAB
  - available in computer labs and BlueHive, can also download with UR account
  - also works through Octave, an open-source clone of MATLAB
- Install:
  - Download the MATLAB version from `dynare.org/download/`
  - Install to some directory
  - In MATLAB, go to Set Path → Add Folder... → select the `matlab` subdirectory of the Dynare installation → save

Resources

- Official manual: `dynare.org/manual.pdf`
  - consult when first using any command to see available options and proper usage
- Johannes Pfeifer's replications of many papers in Dynare: `github.com/johannespfeifer/dsge_mod`
  - if it has a model broadly similar to what you need, can get a headstart
- Johannes Pfeifer's guide to estimation: `sites.google.com/site/pfeiferecon/Pfeifer_2013_Observation_Equations.pdf`

Resources

- Official manual: `dynare.org/manual.pdf`
  - consult when first using any command to see available options and proper usage
- Johannes Pfeifer's replications of many papers in Dynare: `github.com/johannespfeifer/dsge_mod`
  - if it has a model broadly similar to what you need, can get a headstart
- Johannes Pfeifer's guide to estimation: `sites.google.com/site/pfeiferecon/Pfeifer_2013_Observation_Equations.pdf`

Resources

- Official manual: `dynare.org/manual.pdf`
    - consult when first using any command to see available options and proper usage
- Johannes Pfeifer's replications of many papers in Dynare: `github.com/johannespfeifer/dsge_mod`
    - if it has a model broadly similar to what you need, can get a headstart
- Johannes Pfeifer's guide to estimation: `sites.google.com/site/pfeiferecon/Pfeifer_2013_Observation_Equations.pdf`

.mod File

- Dynare has its own syntax and file format: .mod
  - Run from Matlab command line with `dynare example.mod`
- Sections of a .mod file:
  1. Variable declarations
  2. Parameter initialization
  3. Model declaration
  4. Initial conditions
  5. Shocks
  6. Steady state
  7. Solution
  8. Parameter estimation [optional]

.mod File

- Dynare has its own syntax and file format: .mod
    - Run from Matlab command line with `dynare example.mod`
- Sections of a .mod file:
    1. Variable declarations
    2. Parameter initialization
    3. Model declaration
    4. Initial conditions
    5. Shocks
    6. Steady state
    7. Solution
    8. Parameter estimation [optional]

1. Variable Declarations

- Declare endogenous variables:
  `var c k z;`
- Declare exogenous variables:
  `varexo epsilon;`
- Declare parameters:
  `parameters beta eta alpha delta rho z_mean epsilon_sigma;`

1. Variable Declarations

- Declare endogenous variables:
  `var c k z;`
- Declare exogenous variables:
  `varexo epsilon;`
- Declare parameters:
  `parameters beta eta alpha delta rho z_mean epsilon_sigma;`

1. Variable Declarations

- Declare endogenous variables:
  `var c k z;`
- Declare exogenous variables:
  `varexo epsilon;`
- Declare parameters:
  `parameters beta eta alpha delta rho z_mean epsilon_sigma;`

- Define parameter values:

```
beta = 0.98;
eta = 0.5;
alpha = 0.3;
delta = 0.1;
rho = 0.5;
z_mean = 0;
epsilon_sigma = 1;
```

- Dynare timing convention: timing indicates when the variable is determined
  - capital determined (invested) yesterday and available today is
    - in standard convention: k_t
    - in Dynare convention: k_{t-1}
  - capital determined (invested) today and available tomorrow is
    - in standard convention: k_{t+1}
    - in Dynare convention: k_t
- Rewrite the three conditions of our model to match Dynare convention:
  1. $c_t^{-\eta} = \mathbb{E}_t \beta c_{t+1}^{-\eta} \left(\alpha z_{t+1} k_t^{\alpha-1} + 1 - \delta \right)$
  2. $c_t + k_t = z_t k_{t-1}^{\alpha} + (1-\delta) k_{t-1}$
  3. $\log z_t = (1-\rho) \log z^* + \rho \log z_{t-1} + \varepsilon$

3. Model Declaration: Timing

- Dynare timing convention: timing indicates when the variable is determined
  - capital determined (invested) yesterday and available today is
    - in standard convention: k_t
    - in Dynare convention: k_{t-1}
  - capital determined (invested) today and available tomorrow is
    - in standard convention: k_{t+1}
    - in Dynare convention: k_t
- Rewrite the three conditions of our model to match Dynare convention:
  1. $c_t^{-\eta} = \mathbb{E}_t \beta c_{t+1}^{-\eta} \left(\alpha z_{t+1} k_t^{\alpha-1} + 1 - \delta \right)$
  2. $c_t + k_t = z_t k_{t-1}^{\alpha} + (1-\delta) k_{t-1}$
  3. $\log z_t = (1-\rho) \log z^* + \rho \log z_{t-1} + \varepsilon$

3. Model Declaration: Timing

- Dynare timing convention: timing indicates when the variable is determined
  - capital determined (invested) yesterday and available today is
    - in standard convention: k_t
    - in Dynare convention: k_{t-1}
  - capital determined (invested) today and available tomorrow is
    - in standard convention: k_{t+1}
    - in Dynare convention: k_t
- Rewrite the three conditions of our model to match Dynare convention:
  1. $c_t^{-\eta} = \mathbb{E}_t \beta c_{t+1}^{-\eta} \left(\alpha z_{t+1} k_t^{\alpha-1} + 1 - \delta \right)$
  2. $c_t + k_t = z_t k_{t-1}^{\alpha} + (1-\delta) k_{t-1}$
  3. $\log z_t = (1-\rho) \log z^* + \rho \log z_{t-1} + \varepsilon$

3. Model Declaration: Timing

- Dynare timing convention: timing indicates when the variable is determined
  - capital determined (invested) yesterday and available today is
    - in standard convention: k_t
    - in Dynare convention: k_{t-1}
  - capital determined (invested) today and available tomorrow is
    - in standard convention: k_{t+1}
    - in Dynare convention: k_t
- Rewrite the three conditions of our model to match Dynare convention:
  1. $c_t^{-\eta} = \mathbb{E}_t \beta c_{t+1}^{-\eta} \left(\alpha z_{t+1} k_t^{\alpha-1} + 1 - \delta \right)$
  2. $c_t + k_t = z_t k_{t-1}^{\alpha} + (1-\delta) k_{t-1}$
  3. $\log z_t = (1-\rho) \log z^* + \rho \log z_{t-1} + \varepsilon$

3. Model Declaration: Log-Linear

- By default, Dynare will do linear approximation. For log-linear, wrap all variables in exp().
    1. $\exp(c_t)^{-\eta} = \mathbb{E}_t \beta \exp(c_{t+1})^{-\eta} \left(\alpha \exp(z_{t+1}) \exp(k_t)^{\alpha-1} + 1 - \delta\right)$
    2. $\exp(c_t) + \exp(k_t) = \exp(z_t) \exp(k_{t-1})^{\alpha} + (1 - \delta) \exp(k_{t-1})$
    3. $z_t = (1 - \rho)z^* + \rho z_{t-1} + \varepsilon$
- The meaning of each x_t has changed: reflects logs, not levels
    - need to take $\exp(x_t)$ to get levels

3. Model Declaration: Log-Linear

- By default, Dynare will do linear approximation. For log-linear, wrap all variables in exp().
    1. $\exp(c_t)^{-\eta} = \mathbb{E}_t \beta \exp(c_{t+1})^{-\eta} \left(\alpha \exp(z_{t+1}) \exp(k_t)^{\alpha-1} + 1 - \delta\right)$
    2. $\exp(c_t) + \exp(k_t) = \exp(z_t) \exp(k_{t-1})^{\alpha} + (1-\delta) \exp(k_{t-1})$
    3. $z_t = (1-\rho) z^* + \rho z_{t-1} + \varepsilon$
- The meaning of each x_t has changed: reflects logs, not levels
    - need to take $\exp(x_t)$ to get levels

3. Model Declaration: All Together

- List all model equations between `model;` and `end;`.
- Denote x_t with `x`, x_{t+1} with `x(+1)`, x_{t-1} with `x(-1)`, .
    - Dynare will add expectations to forward-looking variables.

```
model;

exp(c)^(-eta) = beta * exp(c(+1))^(-eta) * (alpha *
exp(z(+1)) * exp(k)^(alpha-1) + 1-delta);

exp(c) + exp(k) = exp(z) * exp(k(-1))^alpha + (1-delta) *
exp(k(-1));

z = (1-rho) * z_mean + rho*z(-1) + epsilon;

end;
```

3. Model Declaration: All Together

- List all model equations between `model;` and `end;`.
- Denote x_t with `x`, x_{t+1} with `x(+1)`, x_{t-1} with `x(-1)`, .
    - Dynare will add expectations to forward-looking variables.

```
model;

exp(c)^(-eta) = beta * exp(c(+1))^(-eta) * (alpha *
exp(z(+1)) * exp(k)^(alpha-1) + 1-delta);

exp(c) + exp(k) = exp(z) * exp(k(-1))^alpha + (1-delta) *
exp(k(-1));

z = (1-rho) * z_mean + rho*z(-1) + epsilon;

end;
```

4. Initial Conditions

- Dynare will solve for the s.s., but it needs an initial guess. Because of the exp() transformation, need to apply log() here.

```
initval;
c = log(1);
k = log(1);
z = z_mean;
end;
```

- In a deterministic model, can also define `endval` to compute the transition path between s.s.

4. Initial Conditions

- Dynare will solve for the s.s., but it needs an initial guess. Because of the exp() transformation, need to apply log() here.

```
initval;
c = log(1);
k = log(1);
z = z_mean;
end;
```

- In a deterministic model, can also define `endval` to compute the transition path between s.s.

5. Shocks

- Specify the variance of stochastic variables:

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

6. Steady State

- Option 1: Dynare's steady state solver
  - add `steady;` after the shocks block
  - steady state is found numerically
  - can be sensitive to `initval` guesses
  - verify the quality of the solution
- Option 2: find the s.s. analytically yourself, feed equations to Dynare
  - omit the `initval` block
  - add the steady state block after the shocks block:

```
steady_state_model;
k = log((1/alpha * (1/beta - (1-delta))) ^ (1/(alpha-1)));
c = log(exp(k)^alpha - delta*exp(k));
z = 0;
end;
```

  - more reliable, but more work
- Either is fine for this simple model, but be careful with large models

6. Steady State

- Option 1: Dynare's steady state solver
  - add `steady;` after the shocks block
  - steady state is found numerically
  - can be sensitive to `initval` guesses
  - verify the quality of the solution
- Option 2: find the s.s. analytically yourself, feed equations to Dynare
  - omit the `initval` block
  - add the steady state block after the shocks block:

```
steady_state_model;
k = log((1/alpha * (1/beta - (1-delta))) ^ (1/(alpha-1)));
c = log(exp(k)^alpha - delta*exp(k));
z = 0;
end;
```

  - more reliable, but more work
- Either is fine for this simple model, but be careful with large models

- Option 1: Dynare's steady state solver
  - add `steady;` after the shocks block
  - steady state is found numerically
  - can be sensitive to `initval` guesses
  - verify the quality of the solution
- Option 2: find the s.s. analytically yourself, feed equations to Dynare
  - omit the `initval` block
  - add the steady state block after the shocks block:

```
steady_state_model;
k = log((1/alpha * (1/beta - (1-delta))) ^ (1/(alpha-1)));
c = log(exp(k)^alpha - delta*exp(k));
z = 0;
end;
```

  - more reliable, but more work
- Either is fine for this simple model, but be careful with large models

6. Steady State

- Option 1: Dynare's steady state solver
  - add `steady;` after the shocks block
  - steady state is found numerically
  - can be sensitive to `initval` guesses
  - verify the quality of the solution
- Option 2: find the s.s. analytically yourself, feed equations to Dynare
  - omit the `initval` block
  - add the steady state block after the shocks block:

    ```
    steady_state_model;
    k = log((1/alpha * (1/beta - (1-delta))) ^ (1/(alpha-1)));
    c = log(exp(k)^alpha - delta*exp(k));
    z = 0;
    end;
    ```

  - more reliable, but more work
- Either is fine for this simple model, but be careful with large models

6. Steady State

- Option 1: Dynare's steady state solver
  - add `steady;` after the shocks block
  - steady state is found numerically
  - can be sensitive to `initval` guesses
  - verify the quality of the solution
- Option 2: find the s.s. analytically yourself, feed equations to Dynare
  - omit the `initval` block
  - add the steady state block after the shocks block:

```
steady_state_model;
k = log((1/alpha * (1/beta - (1-delta))) ^ (1/(alpha-1)));
c = log(exp(k)^alpha - delta*exp(k));
z = 0;
end;
```

  - more reliable, but more work
- Either is fine for this simple model, but be careful with large models

6. Steady State

- Option 1: Dynare's steady state solver
  - add `steady;` after the shocks block
  - steady state is found numerically
  - can be sensitive to `initval` guesses
  - verify the quality of the solution
- Option 2: find the s.s. analytically yourself, feed equations to Dynare
  - omit the `initval` block
  - add the steady state block after the shocks block:

```
steady_state_model;
k = log((1/alpha * (1/beta - (1-delta))) ^ (1/(alpha-1)));
c = log(exp(k)^alpha - delta*exp(k));
z = 0;
end;
```

  - more reliable, but more work
- Either is fine for this simple model, but be careful with large models

6. Steady State

- Option 1: Dynare's steady state solver
  - add `steady;` after the shocks block
  - steady state is found numerically
  - can be sensitive to `initval` guesses
  - verify the quality of the solution
- Option 2: find the s.s. analytically yourself, feed equations to Dynare
  - omit the `initval` block
  - add the steady state block after the shocks block:

```
steady_state_model;
k = log((1/alpha * (1/beta - (1-delta))) ^ (1/(alpha-1)));
c = log(exp(k)^alpha - delta*exp(k));
z = 0;
end;
```

  - more reliable, but more work
- Either is fine for this simple model, but be careful with large models

6. Steady State

- Option 1: Dynare's steady state solver
  - add **`steady;`** after the shocks block
  - steady state is found numerically
  - can be sensitive to **`initval`** guesses
  - verify the quality of the solution
- Option 2: find the s.s. analytically yourself, feed equations to Dynare
  - omit the **`initval`** block
  - add the steady state block after the shocks block:

```
steady_state_model;
k = log((1/alpha * (1/beta - (1-delta))) ^ (1/(alpha-1)));
c = log(exp(k)^alpha - delta*exp(k));
z = 0;
end;
```

  - more reliable, but more work
- Either is fine for this simple model, but be careful with large models

6. Steady State

- Option 1: Dynare's steady state solver
  - add **`steady;`** after the shocks block
  - steady state is found numerically
  - can be sensitive to **`initval`** guesses
  - verify the quality of the solution
- Option 2: find the s.s. analytically yourself, feed equations to Dynare
  - omit the **`initval`** block
  - add the steady state block after the shocks block:

    ```
    steady_state_model;
    k = log((1/alpha * (1/beta - (1-delta))) ^ (1/(alpha-1)));
    c = log(exp(k)^alpha - delta*exp(k));
    z = 0;
    end;
    ```
  - more reliable, but more work
- Either is fine for this simple model, but be careful with large models

6. Steady State

- Option 1: Dynare's steady state solver
  - add **`steady;`** after the shocks block
  - steady state is found numerically
  - can be sensitive to **`initval`** guesses
  - verify the quality of the solution
- Option 2: find the s.s. analytically yourself, feed equations to Dynare
  - omit the **`initval`** block
  - add the steady state block after the shocks block:

    ```
    steady_state_model;
    k = log((1/alpha * (1/beta - (1-delta))) ^ (1/(alpha-1)));
    c = log(exp(k)^alpha - delta*exp(k));
    z = 0;
    end;
    ```
  - more reliable, but more work
- Either is fine for this simple model, but be careful with large models

7. Solution

- Solve and simulate the model using log-linear approximation, plot IRFs for 20 periods:
  `stoch_simul(order=1, irf=20, hp_filter=1600);`
- The default is 2nd order approximation. Can run into explosive IRFs, especially with larger shocks. `pruning` option can help:
  `stoch_simul(irf=20, hp_filter=1600, pruning);`

7. Solution

- Solve and simulate the model using log-linear approximation, plot IRFs for 20 periods:
  `stoch_simul(order=1, irf=20, hp_filter=1600);`
- The default is 2nd order approximation. Can run into explosive IRFs, especially with larger shocks. `pruning` option can help:
  `stoch_simul(irf=20, hp_filter=1600, pruning);`

Output

- Command line output
    - numerical steady state values (if used `steady;`)
    - linearized policy functions (all variables in deviations from s.s.)
    - means, variances, correlations, autocorrelations of endogenous variables
- Graphs
    - impulse response functions: response of each endogenous variable to each shock (in deviations from s.s.)
- MATLAB objects
    - `M_` struct: information about the model
    - `oo_` struct: raw output (everything Dynare shows in the command line or graphs comes from here)

OUTPUT

- Command line output
  - numerical steady state values (if used `steady;`)
  - linearized policy functions (all variables in deviations from s.s.)
  - means, variances, correlations, autocorrelations of endogenous variables
- Graphs
  - impulse response functions: response of each endogenous variable to each shock (in deviations from s.s.)
- MATLAB objects
  - `M_` struct: information about the model
  - `oo_` struct: raw output (everything Dynare shows in the command line or graphs comes from here)

Output

- Command line output
  - numerical steady state values (if used `steady;`)
  - linearized policy functions (all variables in deviations from s.s.)
  - means, variances, correlations, autocorrelations of endogenous variables
- Graphs
  - impulse response functions: response of each endogenous variable to each shock (in deviations from s.s.)
- MATLAB objects
  - `M_` struct: information about the model
  - `oo_` struct: raw output (everything Dynare shows in the command line or graphs comes from here)

Calibration

- Calibration: pick parameter values so that endogenous variables reproduce values observed in the data
- Quick & dirty way to do it in Dynare:
  - declare parameters to be calibrated in the `parameters` block, but don't assign specific values
  - add equations linking the parameters to endogenous variables to the `steady_state_model` block
  - set the relevant endogenous variables to their targeted observed values in the `steady_state_model` block
  - Dynare will find parameter values s.t. steady-state endogenous variables match targeted values
  - calibrated parameters saved in the `M_.params` object
  - example in the homework :P

Calibration

- Calibration: pick parameter values so that endogenous variables reproduce values observed in the data
- Quick & dirty way to do it in Dynare:
    - declare parameters to be calibrated in the `parameters` block, but don't assign specific values
    - add equations linking the parameters to endogenous variables to the `steady_state_model` block
    - set the relevant endogenous variables to their targeted observed values in the `steady_state_model` block
    - Dynare will find parameter values s.t. steady-state endogenous variables match targeted values
    - calibrated parameters saved in the `M_.params` object
    - example in the homework :P

Calibration

- Calibration: pick parameter values so that endogenous variables reproduce values observed in the data
- Quick & dirty way to do it in Dynare:
  - declare parameters to be calibrated in the `parameters` block, but don't assign specific values
  - add equations linking the parameters to endogenous variables to the `steady_state_model` block
  - set the relevant endogenous variables to their targeted observed values in the `steady_state_model` block
  - Dynare will find parameter values s.t. steady-state endogenous variables match targeted values
  - calibrated parameters saved in the `M_.params` object
  - example in the homework :P

Calibration

- Calibration: pick parameter values so that endogenous variables reproduce values observed in the data
- Quick & dirty way to do it in Dynare:
    - declare parameters to be calibrated in the `parameters` block, but don't assign specific values
    - add equations linking the parameters to endogenous variables to the `steady_state_model` block
    - set the relevant endogenous variables to their targeted observed values in the `steady_state_model` block
    - Dynare will find parameter values s.t. steady-state endogenous variables match targeted values
    - calibrated parameters saved in the `M_.params` object
    - example in the homework :P

Calibration

- Calibration: pick parameter values so that endogenous variables reproduce values observed in the data
- Quick & dirty way to do it in Dynare:
    - declare parameters to be calibrated in the `parameters` block, but don't assign specific values
    - add equations linking the parameters to endogenous variables to the `steady_state_model` block
    - set the relevant endogenous variables to their targeted observed values in the `steady_state_model` block
    - Dynare will find parameter values s.t. steady-state endogenous variables match targeted values
    - calibrated parameters saved in the `M_.params` object
    - example in the homework :P

Calibration

- Calibration: pick parameter values so that endogenous variables reproduce values observed in the data
- Quick & dirty way to do it in Dynare:
    - declare parameters to be calibrated in the `parameters` block, but don't assign specific values
    - add equations linking the parameters to endogenous variables to the `steady_state_model` block
    - set the relevant endogenous variables to their targeted observed values in the `steady_state_model` block
    - Dynare will find parameter values s.t. steady-state endogenous variables match targeted values
    - calibrated parameters saved in the `M_.params` object
    - example in the homework :P

Calibration

- Calibration: pick parameter values so that endogenous variables reproduce values observed in the data
- Quick & dirty way to do it in Dynare:
    - declare parameters to be calibrated in the `parameters` block, but don't assign specific values
    - add equations linking the parameters to endogenous variables to the `steady_state_model` block
    - set the relevant endogenous variables to their targeted observed values in the `steady_state_model` block
    - Dynare will find parameter values s.t. steady-state endogenous variables match targeted values
    - calibrated parameters saved in the `M_.params` object
    - example in the homework :P

Calibration

- Calibration: pick parameter values so that endogenous variables reproduce values observed in the data
- Quick & dirty way to do it in Dynare:
  - declare parameters to be calibrated in the `parameters` block, but don't assign specific values
  - add equations linking the parameters to endogenous variables to the `steady_state_model` block
  - set the relevant endogenous variables to their targeted observed values in the `steady_state_model` block
  - Dynare will find parameter values s.t. steady-state endogenous variables match targeted values
  - calibrated parameters saved in the `M_.params` object
  - example in the homework :P

- Calibration becomes problematic in complex DSGEs: hard to get clean identification of parameters with data moments
- $\rightarrow$ Estimation: solve for parameter values that best fit observed data series using maximum likelihood or Bayesian methods
- Greatly automated by Dynare
- Dynare can do Bayesian estimation of parameters and shocks to match data:
  1. takes prior distributions of parameters
  2. updates distributions using provided data, returns posterior distributions
  3. calculates the exact sequence of shocks needed to reproduce data
  4. provides numerous diagnostics on the quality of estimation

Estimation

- Calibration becomes problematic in complex DSGEs: hard to get clean identification of parameters with data moments
- $\rightarrow$ Estimation: solve for parameter values that best fit observed data series using maximum likelihood or Bayesian methods
- Greatly automated by Dynare
- Dynare can do Bayesian estimation of parameters and shocks to match data:
    1. takes prior distributions of parameters
    2. updates distributions using provided data, returns posterior distributions
    3. calculates the exact sequence of shocks needed to reproduce data
    4. provides numerous diagnostics on the quality of estimation

Estimation

- Calibration becomes problematic in complex DSGEs: hard to get clean identification of parameters with data moments
- $\rightarrow$ Estimation: solve for parameter values that best fit observed data series using maximum likelihood or Bayesian methods
- Greatly automated by Dynare
- Dynare can do Bayesian estimation of parameters and shocks to match data:
    1. takes prior distributions of parameters
    2. updates distributions using provided data, returns posterior distributions
    3. calculates the exact sequence of shocks needed to reproduce data
    4. provides numerous diagnostics on the quality of estimation

Estimation

- Calibration becomes problematic in complex DSGEs: hard to get clean identification of parameters with data moments
- $\rightarrow$ Estimation: solve for parameter values that best fit observed data series using maximum likelihood or Bayesian methods
- Greatly automated by Dynare
- Dynare can do Bayesian estimation of parameters and shocks to match data:
    1. takes prior distributions of parameters
    2. updates distributions using provided data, returns posterior distributions
    3. calculates the exact sequence of shocks needed to reproduce data
    4. provides numerous diagnostics on the quality of estimation

Estimation

- Calibration becomes problematic in complex DSGEs: hard to get clean identification of parameters with data moments
- $\rightarrow$ Estimation: solve for parameter values that best fit observed data series using maximum likelihood or Bayesian methods
- Greatly automated by Dynare
- Dynare can do Bayesian estimation of parameters and shocks to match data:
    1. takes prior distributions of parameters
    2. updates distributions using provided data, returns posterior distributions
    3. calculates the exact sequence of shocks needed to reproduce data
    4. provides numerous diagnostics on the quality of estimation

Estimation

- Calibration becomes problematic in complex DSGEs: hard to get clean identification of parameters with data moments
- $\rightarrow$ Estimation: solve for parameter values that best fit observed data series using maximum likelihood or Bayesian methods
- Greatly automated by Dynare
- Dynare can do Bayesian estimation of parameters and shocks to match data:
  1. takes prior distributions of parameters
  2. updates distributions using provided data, returns posterior distributions
  3. calculates the exact sequence of shocks needed to reproduce data
  4. provides numerous diagnostics on the quality of estimation

Estimation

- Calibration becomes problematic in complex DSGEs: hard to get clean identification of parameters with data moments
- $\rightarrow$ Estimation: solve for parameter values that best fit observed data series using maximum likelihood or Bayesian methods
- Greatly automated by Dynare
- Dynare can do Bayesian estimation of parameters and shocks to match data:
    1. takes prior distributions of parameters
    2. updates distributions using provided data, returns posterior distributions
    3. calculates the exact sequence of shocks needed to reproduce data
    4. provides numerous diagnostics on the quality of estimation

Estimation

- Calibration becomes problematic in complex DSGEs: hard to get clean identification of parameters with data moments
- $\rightarrow$ Estimation: solve for parameter values that best fit observed data series using maximum likelihood or Bayesian methods
- Greatly automated by Dynare
- Dynare can do Bayesian estimation of parameters and shocks to match data:
    1. takes prior distributions of parameters
    2. updates distributions using provided data, returns posterior distributions
    3. calculates the exact sequence of shocks needed to reproduce data
    4. provides numerous diagnostics on the quality of estimation

Estimation: Setup

- # of shocks $\geq$ # of observed variables.
- Estimate ρ and σ_ε^2 from observed output growth rate (FRED quarterly real GDP growth rate).
- Add two endogenous variables, *y* and *y_growth_rate*:

```
exp(y) = exp(z) * exp(k(-1))^alpha;
y_growth_rate = (exp(y)/exp(y(-1)) - 1)*100;
```

- Declare the observed endogenous variable:

```
varobs y_growth_rate;
```

Estimation: Setup

- # of shocks $\geq$ # of observed variables.
- Estimate ρ and σ_ε^2 from observed output growth rate (FRED quarterly real GDP growth rate).
- Add two endogenous variables, *y* and *y_growth_rate*:

```
exp(y) = exp(z) * exp(k(-1))^alpha;
y_growth_rate = (exp(y)/exp(y(-1)) - 1)*100;
```

- Declare the observed endogenous variable:

```
varobs y_growth_rate;
```

Estimation: Setup

- # of shocks $\geq$ # of observed variables.
- Estimate ρ and σ_ε^2 from observed output growth rate (FRED quarterly real GDP growth rate).
- Add two endogenous variables, *y* and *y_growth_rate*:

```
exp(y) = exp(z) * exp(k(-1))^alpha;
y_growth_rate = (exp(y)/exp(y(-1)) - 1)*100;
```

- Declare the observed endogenous variable:

```
varobs y_growth_rate;
```

Estimation: Setup

- # of shocks $\geq$ # of observed variables.
- Estimate ρ and σ_ε^2 from observed output growth rate (FRED quarterly real GDP growth rate).
- Add two endogenous variables, *y* and *y_growth_rate*:

```
exp(y) = exp(z) * exp(k(-1))^alpha;
y_growth_rate = (exp(y)/exp(y(-1)) - 1)*100;
```

- Declare the observed endogenous variable:

```
varobs y_growth_rate;
```

Estimation: Priors

- Declare the type, mean, and sd of the prior distributions you have on parameters you want to estimate

```
estimated_params;
stderr epsilon, inv_gamma_pdf, 0.01, 0.5;
rho, beta_pdf, 0.5, 0.25;
end;
```

- Pick the distribution type based on the range of values the parameter can take.
    - see manual for available distributions
- Posteriors may be sensitive to priors in some cases
    - learn to interpret Dynare's estimation diagnostics if you need to estimate a DSGE for a serious project

Estimation: Priors

- Declare the type, mean, and sd of the prior distributions you have on parameters you want to estimate

```
estimated_params;
stderr epsilon, inv_gamma_pdf, 0.01, 0.5;
rho, beta_pdf, 0.5, 0.25;
end;
```

- Pick the distribution type based on the range of values the parameter can take.
  - see manual for available distributions
- Posteriors may be sensitive to priors in some cases
  - learn to interpret Dynare's estimation diagnostics if you need to estimate a DSGE for a serious project

Estimation: Priors

- Declare the type, mean, and sd of the prior distributions you have on parameters you want to estimate

```
estimated_params;
stderr epsilon, inv_gamma_pdf, 0.01, 0.5;
rho, beta_pdf, 0.5, 0.25;
end;
```

- Pick the distribution type based on the range of values the parameter can take.
    - see manual for available distributions
- Posteriors may be sensitive to priors in some cases
    - learn to interpret Dynare's estimation diagnostics if you need to estimate a DSGE for a serious project

Estimation

- Run estimation:
  `estimation(datafile = 'y_growth_rate.mat', prefilter = 1);`
- `datafile` is a vector or table. Name of the file or columns must match observed `varobs` names.
- `prefilter = 1` will demean the data (normally detrend yourself).
- Dynare will find parameter values that best match the series and find the sequence of shocks that reproduces the series.
- Main outputs
  - posterior distributions of parameters
  - fitted `varobs`
  - shock sequence producing the fit
  - estimation diagnostics

Estimation

- Run estimation:
  `estimation(datafile = 'y_growth_rate.mat', prefilter = 1);`
- `datafile` is a vector or table. Name of the file or columns must match observed `varobs` names.
- `prefilter = 1` will demean the data (normally detrend yourself).
- Dynare will find parameter values that best match the series and find the sequence of shocks that reproduces the series.
- Main outputs
  - posterior distributions of parameters
  - fitted `varobs`
  - shock sequence producing the fit
  - estimation diagnostics

Estimation

- Run estimation: `estimation(datafile = 'y_growth_rate.mat', prefilter = 1);`
- `datafile` is a vector or table. Name of the file or columns must match observed `varobs` names.
- `prefilter = 1` will demean the data (normally detrend yourself).
- Dynare will find parameter values that best match the series and find the sequence of shocks that reproduces the series.
- Main outputs
  - posterior distributions of parameters
  - fitted `varobs`
  - shock sequence producing the fit
  - estimation diagnostics

Estimation

- Run estimation:
  `estimation(datafile = 'y_growth_rate.mat', prefilter = 1);`
- `datafile` is a vector or table. Name of the file or columns must match observed `varobs` names.
- `prefilter = 1` will demean the data (normally detrend yourself).
- Dynare will find parameter values that best match the series and find the sequence of shocks that reproduces the series.
- Main outputs
  - posterior distributions of parameters
  - fitted `varobs`
  - shock sequence producing the fit
  - estimation diagnostics

Estimation

- Run estimation:
  `estimation(datafile = 'y_growth_rate.mat', prefilter = 1);`
- `datafile` is a vector or table. Name of the file or columns must match observed `varobs` names.
- `prefilter = 1` will demean the data (normally detrend yourself).
- Dynare will find parameter values that best match the series and find the sequence of shocks that reproduces the series.
- Main outputs
  - posterior distributions of parameters
  - fitted `varobs`
  - shock sequence producing the fit
  - estimation diagnostics