The Science and Art of DSGE Modelling
A Foundations Course

Introduction to Dynare

Paul Levine

7 September 2020

Outline

What is Dynare?

- A suite of programs for the solution, simulation and estimation of DSGE models *with or without* rational expectations

What is Dynare?

- A suite of programs for the solution, simulation and estimation of DSGE models *with or without* rational expectations
- OS: Windows, Mac OS and Linux.

What is Dynare?

- A suite of programs for the solution, simulation and estimation of DSGE models *with or without* rational expectations
- OS: Windows, Mac OS and Linux.
- Platforms: **MATLAB** (at least version 2017a, possibly earlier) 32-bit and 64-bit) and GNU Octave (but not tested)

What is Dynare?

- A suite of programs for the solution, simulation and estimation of DSGE models *with or without* rational expectations
- OS: Windows, Mac OS and Linux.
- Platforms: **MATLAB** (at least version 2017a, possibly earlier) 32-bit and 64-bit) and GNU Octave (but not tested)
- Collection of over 1000+ MATLAB routines (part of Dynare is programmed in C++)

What is Dynare?

- A suite of programs for the solution, simulation and estimation of DSGE models *with or without* rational expectations
- OS: Windows, Mac OS and Linux.
- Platforms: **MATLAB** (at least version 2017a, possibly earlier) 32-bit and 64-bit) and GNU Octave (but not tested)
- Collection of over 1000+ MATLAB routines (part of Dynare is programmed in C++)
- More information and to download Dynare (v4.6.1 is the latest (stable) version): `http://www.dynare.org/`

What is Dynare?

- A suite of programs for the solution, simulation and estimation of DSGE models *with or without* rational expectations
- OS: Windows, Mac OS and Linux.
- Platforms: **MATLAB** (at least version 2017a, possibly earlier) 32-bit and 64-bit) and GNU Octave (but not tested)
- Collection of over 1000+ MATLAB routines (part of Dynare is programmed in C++)
- More information and to download Dynare (v4.6.1 is the latest (stable) version): `http://www.dynare.org/`
- Very user-friendly

What is Dynare?

- A suite of programs for the solution, simulation and estimation of DSGE models *with or without* rational expectations
- OS: Windows, Mac OS and Linux.
- Platforms: **MATLAB** (at least version 2017a, possibly earlier) 32-bit and 64-bit) and GNU Octave (but not tested)
- Collection of over 1000+ MATLAB routines (part of Dynare is programmed in C++)
- More information and to download Dynare (v4.6.1 is the latest (stable) version): `http://www.dynare.org/`
- Very user-friendly
- This course will be based on Dynare v4.5.7 or v4.6.1

Outline

Learning resources

- **Reference Manual (and User Guide)**:
  `http://www.dynare.org/documentation-and-support`

Learning resources

- **Reference Manual (and User Guide)**: `http://www.dynare.org/documentation-and-support`
- **Official Online Examples**: available also from the Dynare website and are usually well documented

Learning resources

- **Reference Manual (and User Guide)**: `http://www.dynare.org/documentation-and-support`
- **Official Online Examples**: available also from the Dynare website and are usually well documented
- **Dynare Discussion Forum** (access from the website): for asking questions, posting comments and uploading examples and has more examples and latest developments

Learning resources

- **Reference Manual (and User Guide)**: `http://www.dynare.org/documentation-and-support`
- **Official Online Examples**: available also from the Dynare website and are usually well documented
- **Dynare Discussion Forum** (access from the website): for asking questions, posting comments and uploading examples and has more examples and latest developments
- **DSGE.net**: website run by the Dynare team and useful not only for the dynare users but also for all scholars working on DSGE modelling

Learning resources

- **Reference Manual (and User Guide)**: `http://www.dynare.org/documentation-and-support`
- **Official Online Examples**: available also from the Dynare website and are usually well documented
- **Dynare Discussion Forum** (access from the website): for asking questions, posting comments and uploading examples and has more examples and latest developments
- **DSGE.net**: website run by the Dynare team and useful not only for the dynare users but also for all scholars working on DSGE modelling
- **Dynare Wiki**: various bits of information such as new or undocumented features and development plans

Outline

What is Dynare able to do?

- Specifies a Rational Expectations (RE) or non-RE model in linear or non-linear form

What is Dynare able to do?

- Specifies a Rational Expectations (RE) or non-RE model in linear or non-linear form
- Computes the steady state (numerically) of the model

What is Dynare able to do?

- Specifies a Rational Expectations (RE) or non-RE model in linear or non-linear form
- Computes the steady state (numerically) of the model
- Computes the 'exact' solution of deterministic models

What is Dynare able to do?

- Specifies a Rational Expectations (RE) or non-RE model in linear or non-linear form
- Computes the steady state (numerically) of the model
- Computes the 'exact' solution of deterministic models
- Computes first, second and third order approximation to solutions of stochastic models

What is Dynare able to do?

- Specifies a Rational Expectations (RE) or non-RE model in linear or non-linear form
- Computes the steady state (numerically) of the model
- Computes the 'exact' solution of deterministic models
- Computes first, second and third order approximation to solutions of stochastic models
- Estimates (either Maximum Likelihood or Bayesian approach) parameters of DSGE models

What is Dynare able to do?

- Specifies a Rational Expectations (RE) or non-RE model in linear or non-linear form
- Computes the steady state (numerically) of the model
- Computes the 'exact' solution of deterministic models
- Computes first, second and third order approximation to solutions of stochastic models
- Estimates (either Maximum Likelihood or Bayesian approach) parameters of DSGE models
- Computes optimal policy

Outline

How does Dynare work?

- The user writes a .mod file (in an editor of your choice)

How does Dynare work?

- The user writes a .mod file (in an editor of your choice)
- That file is then called from MATLAB allowing the pre-processor to translate the .mod file into a suitable input for the MATLAB routines (i.e. Dynare at this stage produces .m files)

How does Dynare work?

- The user writes a .mod file (in an editor of your choice)
- That file is then called from MATLAB allowing the pre-processor to translate the .mod file into a suitable input for the MATLAB routines (i.e. Dynare at this stage produces .m files)
- MATLAB routines are used to either solve or estimate the model and present the output $\Rightarrow$ .mat are matrices produced by MATLAB

Outline

Structure of a .mod file

- Variables and parameters declarations

Structure of a .mod file

- Variables and parameters declarations
- Parameter initialization

Structure of a .mod file

- Variables and parameters declarations
- Parameter initialization
- Model declaration

Structure of a .mod file

- Variables and parameters declarations
- Parameter initialization
- Model declaration
- Shocks (and their properties)

Structure of a .mod file

- Variables and parameters declarations
- Parameter initialization
- Model declaration
- Shocks (and their properties)
- Initial and terminal conditions for simulations (where the terminal conditions are only applicable to specify permanent shocks that hit deterministic models)

Structure of a .mod file

- Variables and parameters declarations
- Parameter initialization
- Model declaration
- Shocks (and their properties)
- Initial and terminal conditions for simulations (where the terminal conditions are only applicable to specify permanent shocks that hit deterministic models)
- On the Course 'initial conditions' will consist of the steady state of the model

Structure of a .mod file

- Variables and parameters declarations
- Parameter initialization
- Model declaration
- Shocks (and their properties)
- Initial and terminal conditions for simulations (where the terminal conditions are only applicable to specify permanent shocks that hit deterministic models)
- On the Course 'initial conditions' will consist of the steady state of the model
- Computations to be performed, e.g.: steady, check, forecast, simul, stoch_simul, estimation, ramsey_policy, etc ...

Outline

Declarations of the variables and parameters

- Endogenous and exogenous variables are declared separately

Declarations of the variables and parameters

- Endogenous and exogenous variables are declared separately
- For **endogenous variables** use the command: *var* (followed by the list of endogenous in the model)

Declarations of the variables and parameters

- Endogenous and exogenous variables are declared separately
- For **endogenous variables** use the command: *var* (followed by the list of endogenous in the model)
- **Exogenous variables** are declared with: *varexo* (followed by the list of exogenous variables that will be shocked)

Declarations of the variables and parameters

- Endogenous and exogenous variables are declared separately
- For **endogenous variables** use the command: *var* (followed by the list of endogenous in the model)
- **Exogenous variables** are declared with: *varexo* (followed by the list of exogenous variables that will be shocked)
- The parameters of the model are defined with the command: *parameters* and afterwards they are calibrated (values are assigned to each)

Declarations of the variables and parameters

- Endogenous and exogenous variables are declared separately
- For **endogenous variables** use the command: *var* (followed by the list of endogenous in the model)
- **Exogenous variables** are declared with: *varexo* (followed by the list of exogenous variables that will be shocked)
- The parameters of the model are defined with the command: *parameters* and afterwards they are calibrated (values are assigned to each)
- **Note** that each instruction of the .mod file must be terminated by a semicolon (;)

Declarations of the variables and parameters

- Endogenous and exogenous variables are declared separately
- For **endogenous variables** use the command: *var* (followed by the list of endogenous in the model)
- **Exogenous variables** are declared with: *varexo* (followed by the list of exogenous variables that will be shocked)
- The parameters of the model are defined with the command: *parameters* and afterwards they are calibrated (values are assigned to each)
- **Note** that each instruction of the .mod file must be terminated by a semicolon (;)
- Also Dynare uses 2 forward slashes (//) to comment out any line (whereas MATLAB uses %). (Note: for Dynare the two are equivalent!)

Outline

Specifying the model equations

- Defined within the commands *model;* and *end;*

Specifying the model equations

- Defined within the commands *model;* and *end;*
- Different time indices in a dynamic model are abbreviated as:

Specifying the model equations

- Defined within the commands *model;* and *end;*
- Different time indices in a dynamic model are abbreviated as:
  - $x_t = x \longrightarrow$ variable x decided in the current period (t) is written plainly as x

Specifying the model equations

- Defined within the commands *model;* and *end;*
- Different time indices in a dynamic model are abbreviated as:
  - $x_t = x \longrightarrow$ variable x decided in the current period (t) is written plainly as x
  - $x_{t-1} = x(-1) \longrightarrow$ the variable is decided in t-1 (predetermined), e.g. the capital stock, write it as x(-1) instead of x

Specifying the model equations

- Defined within the commands *model;* and *end;*
- Different time indices in a dynamic model are abbreviated as:
  - $x_t = x \longrightarrow$ variable x decided in the current period (t) is written plainly as x
  - $x_{t-1} = x(-1) \longrightarrow$ the variable is decided in t-1 (predetermined), e.g. the capital stock, write it as x(-1) instead of x
  - $x_{t+1} = x(+1) \longrightarrow$ the variable is decided n periods ahead, write x(+n). Dynare reads x(+1) as a jumper (or forward-looking or non-predetermined) variable

Specifying the model equations

- Defined within the commands *model;* and *end;*
- Different time indices in a dynamic model are abbreviated as:
  - $x_t = x \longrightarrow$ variable x decided in the current period (t) is written plainly as x
  - $x_{t-1} = x(-1) \longrightarrow$ the variable is decided in t-1 (predetermined), e.g. the capital stock, write it as x(-1) instead of x
  - $x_{t+1} = x(+1) \longrightarrow$ the variable is decided n periods ahead, write x(+n). Dynare reads x(+1) as a jumper (or forward-looking or non-predetermined) variable
- It is important to make sure when specifying the model:

Specifying the model equations

- Defined within the commands *model;* and *end;*
- Different time indices in a dynamic model are abbreviated as:
  - $x_t = x \longrightarrow$ variable x decided in the current period (t) is written plainly as x
  - $x_{t-1} = x(-1) \longrightarrow$ the variable is decided in t-1 (predetermined), e.g. the capital stock, write it as x(-1) instead of x
  - $x_{t+1} = x(+1) \longrightarrow$ the variable is decided n periods ahead, write x(+n). Dynare reads x(+1) as a jumper (or forward-looking or non-predetermined) variable
- It is important to make sure when specifying the model:
  - must have no. of equations = no. of variables declared

Specifying the model equations

- Defined within the commands *model;* and *end;*
- Different time indices in a dynamic model are abbreviated as:
  - $x_t = x \longrightarrow$ variable x decided in the current period (t) is written plainly as x
  - $x_{t-1} = x(-1) \longrightarrow$ the variable is decided in t-1 (predetermined), e.g. the capital stock, write it as x(-1) instead of x
  - $x_{t+1} = x(+1) \longrightarrow$ the variable is decided n periods ahead, write x(+n). Dynare reads x(+1) as a jumper (or forward-looking or non-predetermined) variable
- It is important to make sure when specifying the model:
  - must have no. of equations = no. of variables declared
  - names are case sensitive

Specifying the model equations

- Defined within the commands *model;* and *end;*
- Different time indices in a dynamic model are abbreviated as:
  - $x_t = x \longrightarrow$ variable x decided in the current period (t) is written plainly as x
  - $x_{t-1} = x(-1) \longrightarrow$ the variable is decided in t-1 (predetermined), e.g. the capital stock, write it as x(-1) instead of x
  - $x_{t+1} = x(+1) \longrightarrow$ the variable is decided n periods ahead, write x(+n). Dynare reads x(+1) as a jumper (or forward-looking or non-predetermined) variable
- It is important to make sure when specifying the model:
  - must have no. of equations = no. of variables declared
  - names are case sensitive
  - **Stability condition**: referred to as the Blanchard-Kahn condition - met only if the number of jumpers equals the number of eigenvalues greater than one. See stability analysis on Day 2

Specifying the model equations

- Defined within the commands *model;* and *end;*
- Different time indices in a dynamic model are abbreviated as:
  - $x_t = x \longrightarrow$ variable x decided in the current period (t) is written plainly as x
  - $x_{t-1} = x(-1) \longrightarrow$ the variable is decided in t-1 (predetermined), e.g. the capital stock, write it as x(-1) instead of x
  - $x_{t+1} = x(+1) \longrightarrow$ the variable is decided n periods ahead, write x(+n). Dynare reads x(+1) as a jumper (or forward-looking or non-predetermined) variable
- It is important to make sure when specifying the model:
  - must have no. of equations = no. of variables declared
  - names are case sensitive
  - **Stability condition**: referred to as the Blanchard-Kahn condition - met only if the number of jumpers equals the number of eigenvalues greater than one. See stability analysis on Day 2
- In case the model consists of linear equations use *model (linear);* as opening command.

Outline

Properties of shocks

- The variances (and covariances) of the shocks are defined within the commands *shocks;* and *end;*

Properties of shocks

- The variances (and covariances) of the shocks are defined within the commands *shocks;* and *end;*
- e.g. one most common way is to use: the command '**var** *name of exogenous variable*; **stderr** *0.01*;' which sets the std. error of this exogenous variable = 0.01. Interpret this as 1%

Properties of shocks

- The variances (and covariances) of the shocks are defined within the commands *shocks;* and *end;*
- e.g. one most common way is to use: the command '**var** *name of exogenous variable*; **stderr** *0.01*;' which sets the std. error of this exogenous variable = 0.01. Interpret this as 1%
- For first order solution only can scale up 0.01 to 1.0

Outline

Finding the steady state in the mod file

- Dynare solves for the steady state of the model (nonlinear Newton-type solver). It just needs good numerical initial values - simple to use.

Finding the steady state in the mod file

- Dynare solves for the steady state of the model (nonlinear Newton-type solver). It just needs good numerical initial values - simple to use.
- These are initial guesses specified within the commands *initval;* and *end;*

Finding the steady state in the mod file

- Dynare solves for the steady state of the model (nonlinear Newton-type solver). It just needs good numerical initial values - simple to use.
- These are initial guesses specified within the commands *initval;* and *end;*
- Then, the command *steady* triggers the computation of exact values for the steady state using the initial guesses for the values of the endogenous variables

Finding the steady state in the mod file

- Dynare solves for the steady state of the model (nonlinear Newton-type solver). It just needs good numerical initial values - simple to use.
- These are initial guesses specified within the commands *initval;* and *end;*
- Then, the command *steady* triggers the computation of exact values for the steady state using the initial guesses for the values of the endogenous variables
- Variables that are not initialized in *initval;...end;* are set to zero

Finding the steady state in the mod file

- Dynare solves for the steady state of the model (nonlinear Newton-type solver). It just needs good numerical initial values - simple to use.
- These are initial guesses specified within the commands *initval;* and *end;*
- Then, the command *steady* triggers the computation of exact values for the steady state using the initial guesses for the values of the endogenous variables
- Variables that are not initialized in *initval;...end;* are set to zero
- The routines (solvers) can be very sensitive to your guess

Finding the steady state in the mod file

- Dynare solves for the steady state of the model (nonlinear Newton-type solver). It just needs good numerical initial values - simple to use.
- These are initial guesses specified within the commands *initval;* and *end;*
- Then, the command *steady* triggers the computation of exact values for the steady state using the initial guesses for the values of the endogenous variables
- Variables that are not initialized in *initval;...end;* are set to zero
- The routines (solvers) can be very sensitive to your guess
- Better to use an analytical steady state - using the *steady_state_model;...end;* command or an external steady state file.

Finding the steady state in the mod file

- Dynare solves for the steady state of the model (nonlinear Newton-type solver). It just needs good numerical initial values - simple to use.
- These are initial guesses specified within the commands *initval;* and *end;*
- Then, the command *steady* triggers the computation of exact values for the steady state using the initial guesses for the values of the endogenous variables
- Variables that are not initialized in *initval;...end;* are set to zero
- The routines (solvers) can be very sensitive to your guess
- Better to use an analytical steady state - using the *steady_state_model;...end;* command or an external steady state file.
- A useful command after *steady*, *check*, that checks the Blanchard and Kahn (1980) conditions that ensure the existence of a RE solution by computing the eigenvalues of the model - See Stability on day 2.

Finding the steady state using an external 'steady state file'

- Can have problems in finding the steady state depending on the model complexity and the initial values

Finding the steady state using an external 'steady state file'

- Can have problems in finding the steady state depending on the model complexity and the initial values
- An alternative is using a A MATLAB function doing the computation externally with a Matlab program FILENAME_steadystate.m

Finding the steady state using an external 'steady state file'

- Can have problems in finding the steady state depending on the model complexity and the initial values
- An alternative is using a A MATLAB function doing the computation externally with a Matlab program FILENAME_steadystate.m
- The correct steady state is then computed and called by FILENAME.mod

Finding the steady state using an external 'steady state file'

- Can have problems in finding the steady state depending on the model complexity and the initial values
- An alternative is using a A MATLAB function doing the computation externally with a Matlab program FILENAME_steadystate.m
- The correct steady state is then computed and called by FILENAME.mod
- Useful (more efficient) when you know how to compute the steady state for your model

Finding the steady state using an external 'steady state file'

- Can have problems in finding the steady state depending on the model complexity and the initial values
- An alternative is using a A MATLAB function doing the computation externally with a Matlab program FILENAME_steadystate.m
- The correct steady state is then computed and called by FILENAME.mod
- Useful (more efficient) when you know how to compute the steady state for your model
- Sometimes the steady state can be written recursively and put within the initial values commands. This will be the case on most of the examples in this Course.

Outline

Solving (simulating) the model

- For the most part we construct *stochastic* models, hence DSGE. The command *stoch_simul* performs the solution using a perturbation (Taylor small-shock approximation) routine.

Solving (simulating) the model

- For the most part we construct *stochastic* models, hence DSGE. The command *stoch_simul* performs the solution using a perturbation (Taylor small-shock approximation) routine.
- However, for a perfect foresight (*deterministic*) simulations, *simul* gives an *exact* solution.

Solving (simulating) the model

- For the most part we construct *stochastic* models, hence DSGE. The command *stoch_simul* performs the solution using a perturbation (Taylor small-shock approximation) routine.
- However, for a perfect foresight (*deterministic*) simulations, *simul* gives an *exact* solution.
- The command *stoch_simul* computes a Taylor approximation around the steady state of order one, two or three

Solving (simulating) the model

- For the most part we construct *stochastic* models, hence DSGE. The command *stoch_simul* performs the solution using a perturbation (Taylor small-shock approximation) routine.
- However, for a perfect foresight (*deterministic*) simulations, *simul* gives an *exact* solution.
- The command *stoch_simul* computes a Taylor approximation around the steady state of order one, two or three
- Output includes a Taylor approximation of the state-space solution of the model, simulated Impulse Response Functions (IRFs- see more in RBC Model Session), moments and autocorrelations.

Solving (simulating) the model

- For the most part we construct *stochastic* models, hence DSGE. The command *stoch_simul* performs the solution using a perturbation (Taylor small-shock approximation) routine.
- However, for a perfect foresight (*deterministic*) simulations, *simul* gives an *exact* solution.
- The command *stoch_simul* computes a Taylor approximation around the steady state of order one, two or three
- Output includes a Taylor approximation of the state-space solution of the model, simulated Impulse Response Functions (IRFs- see more in RBC Model Session), moments and autocorrelations.
- It also simulates artificial data from the model (*datatomfile* - saves the simulated data in a m file. Example: datatomfile('simuldata',[]))

Solving (simulating) the model

- For the most part we construct *stochastic* models, hence DSGE. The command *stoch_simul* performs the solution using a perturbation (Taylor small-shock approximation) routine.
- However, for a perfect foresight (*deterministic*) simulations, *simul* gives an *exact* solution.
- The command *stoch_simul* computes a Taylor approximation around the steady state of order one, two or three
- Output includes a Taylor approximation of the state-space solution of the model, simulated Impulse Response Functions (IRFs- see more in RBC Model Session), moments and autocorrelations.
- It also simulates artificial data from the model (*datatomfile* - saves the simulated data in a m file. Example: datatomfile('simuldata',[]))
- Some useful options set in brackets after *stoch_simul*:

Solving (simulating) the model

- For the most part we construct *stochastic* models, hence DSGE. The command *stoch_simul* performs the solution using a perturbation (Taylor small-shock approximation) routine.
- However, for a perfect foresight (*deterministic*) simulations, *simul* gives an *exact* solution.
- The command *stoch_simul* computes a Taylor approximation around the steady state of order one, two or three
- Output includes a Taylor approximation of the state-space solution of the model, simulated Impulse Response Functions (IRFs- see more in RBC Model Session), moments and autocorrelations.
- It also simulates artificial data from the model (*datatomfile* - saves the simulated data in a m file. Example: datatomfile('simuldata',[]))
- Some useful options set in brackets after *stoch_simul*:
  - *periods* - specifies the number of simulation periods. Example: periods=1000; This is essential for order 3.

Solving (simulating) the model

- For the most part we construct *stochastic* models, hence DSGE. The command *stoch_simul* performs the solution using a perturbation (Taylor small-shock approximation) routine.
- However, for a perfect foresight (*deterministic*) simulations, *simul* gives an *exact* solution.
- The command *stoch_simul* computes a Taylor approximation around the steady state of order one, two or three
- Output includes a Taylor approximation of the state-space solution of the model, simulated Impulse Response Functions (IRFs- see more in RBC Model Session), moments and autocorrelations.
- It also simulates artificial data from the model (*datatomfile* - saves the simulated data in a m file. Example: datatomfile('simuldata',[]))
- Some useful options set in brackets after *stoch_simul*:
  - *periods* - specifies the number of simulation periods. Example: periods=1000; This is essential for order 3.
  - *irf* sets the number of periods for which to compute impulse responses.

Solving (simulating) the model

- For the most part we construct *stochastic* models, hence DSGE. The command *stoch_simul* performs the solution using a perturbation (Taylor small-shock approximation) routine.
- However, for a perfect foresight (*deterministic*) simulations, *simul* gives an *exact* solution.
- The command *stoch_simul* computes a Taylor approximation around the steady state of order one, two or three
- Output includes a Taylor approximation of the state-space solution of the model, simulated Impulse Response Functions (IRFs- see more in RBC Model Session), moments and autocorrelations.
- It also simulates artificial data from the model (*datatomfile* - saves the simulated data in a m file. Example: datatomfile('simuldata',[]))
- Some useful options set in brackets after *stoch_simul*:
  - *periods* - specifies the number of simulation periods. Example: periods=1000; This is essential for order 3.
  - *irf* sets the number of periods for which to compute impulse responses.
  - *order* = 1 sets the order of the Taylor approximation (default is two).

Outline

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation
- Estimation steps:

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation
- Estimation steps:
  - computes the steady state

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation
- Estimation steps:
  - computes the steady state
  - linearizes the model (only for non-linear models)

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation
- Estimation steps:
  - computes the steady state
  - linearizes the model (only for non-linear models)
  - solves the linearized model

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation
- Estimation steps:
  - computes the steady state
  - linearizes the model (only for non-linear models)
  - solves the linearized model
  - computes the log-likelihood via the Kalman filter

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation
- Estimation steps:
  - computes the steady state
  - linearizes the model (only for non-linear models)
  - solves the linearized model
  - computes the log-likelihood via the Kalman filter
  - finds the maximum of the likelihood or posterior mode

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation
- Estimation steps:
  - computes the steady state
  - linearizes the model (only for non-linear models)
  - solves the linearized model
  - computes the log-likelihood via the Kalman filter
  - finds the maximum of the likelihood or posterior mode
  - simulates posterior distribution with Metropolis algorithm

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation
- Estimation steps:
  - computes the steady state
  - linearizes the model (only for non-linear models)
  - solves the linearized model
  - computes the log-likelihood via the Kalman filter
  - finds the maximum of the likelihood or posterior mode
  - simulates posterior distribution with Metropolis algorithm
  - computes various statistics on the basis of the posterior distribution (post. moments)

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation
- Estimation steps:
  - computes the steady state
  - linearizes the model (only for non-linear models)
  - solves the linearized model
  - computes the log-likelihood via the Kalman filter
  - finds the maximum of the likelihood or posterior mode
  - simulates posterior distribution with Metropolis algorithm
  - computes various statistics on the basis of the posterior distribution (post. moments)
  - estimates the posterior marginal density to compare models.

Estimating the model (Maximum Likelihood or Bayesian)

- Dynare estimates the structural parameters of a model based on a linear approximation
- Estimation steps:
  - computes the steady state
  - linearizes the model (only for non-linear models)
  - solves the linearized model
  - computes the log-likelihood via the Kalman filter
  - finds the maximum of the likelihood or posterior mode
  - simulates posterior distribution with Metropolis algorithm
  - computes various statistics on the basis of the posterior distribution (post. moments)
  - estimates the posterior marginal density to compare models.
  - computes forecasts and confidence intervals

Outline

AR 1 and VAR Examples

1. **AR1 processes**:

$$\begin{aligned} x_t &= \rho_x x_{t-1} + e_{x,t} \\ y_t &= \rho_y y_{t-1} + e_{y,t} \\ e_{x,t} &\sim i.i.d(0, \sigma_x^2) \\ e_{y,t} &\sim i.i.d(0, \sigma_y^2) \end{aligned}$$

AR 1 and VAR Examples

1. **AR1 processes**:

$$\begin{aligned} x_t &= \rho_x x_{t-1} + e_{x,t} \\ y_t &= \rho_y y_{t-1} + e_{y,t} \\ e_{x,t} &\sim i.i.d(0, \sigma_x^2) \\ e_{y,t} &\sim i.i.d(0, \sigma_y^2) \end{aligned}$$

2. **A first-order VAR**:

$$\begin{aligned} x_t &= \rho_{xx} x_{t-1} + \rho_{xy} y_{t-1} + e_{x,t} \\ y_t &= \rho_{yy} y_{t-1} + \rho_{yx} x_{t-1} + e_{y,t} \\ e_{x,t} &\sim i.i.d(0, \sigma_x^2) \\ e_{y,t} &\sim i.i.d(0, \sigma_y^2) \end{aligned}$$

Dynare Model Files and Exercise

- **Dynare_Example_1_stoch.mod** and **Dynare_Example_2_stoch.mod** illustrate the stochastic solution where shocks are present

Dynare Model Files and Exercise

- **Dynare_Example_1_stoch.mod** and **Dynare_Example_2_stoch.mod** illustrate the stochastic solution where shocks are present
- **Dynare_Example_2_stoch.mod** also shows how to produce a latex version of the model in your mod file and how to name each equation in the model block in order to easily identify mistakes in the calculation of the steady state.

Dynare Model Files and Exercise

- **Dynare_Example_1_stoch.mod** and **Dynare_Example_2_stoch.mod** illustrate the stochastic solution where shocks are present
- **Dynare_Example_2_stoch.mod** also shows how to produce a latex version of the model in your mod file and how to name each equation in the model block in order to easily identify mistakes in the calculation of the steady state.
- Dynare model files **Dynare_Example_1_det. mod** and **Dynare_Example_2_det.mod** illustrate deterministic solutions in the absence of shocks. But the Course only performs stochastic simulations

Outline

Latex and Dynare

- It is possible to specify Latex names for variables and parameter in order for Dynare to produce various output files in .tex format.

Latex and Dynare

- It is possible to specify Latex names for variables and parameter in order for Dynare to produce various output files in .tex format.
- **Dynare_Example_2_stoch.mod** shows some basic commands:

Latex and Dynare

- It is possible to specify Latex names for variables and parameter in order for Dynare to produce various output files in .tex format.
- **Dynare_Example_2_stoch.mod** shows some basic commands:
- ```
  varexo ex $e_x$ ey $e_y$;
  parameters rhoxx $\rho_{xx}$ ...;
  ```
  ```

Latex and Dynare

- It is possible to specify Latex names for variables and parameter in order for Dynare to produce various output files in .tex format.
- **Dynare_Example_2_stoch.mod** shows some basic commands:
- ```
  varexo ex $e_x$ ey $e_y$;
  parameters rhoxx $\rho_{xx}$ ...;
  ```
- ```
  varexo ex $e_x$ ey $e_y$;
  parameters rhoxx $\rho_{xx}$ ...;
  ```

Naming equations

- You can also name each equation in the model block

Naming equations

- You can also name each equation in the model block

- ```
  model;
  [name='Equation for x']
  x = rhoxx*x(-1)+rhoxy*y(-1)+ex;
  [name='Equation for y']
  y = rhoyy*y(-1)+rhoyx*x(-1)+ ey;
  end;
  ```
  ```

Naming equations

- You can also name each equation in the model block
- ```
  model;
  [name='Equation for x']
  x = rhoxx*x(-1)+rhoxy*y(-1)+ex;
  [name='Equation for y']
  y = rhoyy*y(-1)+rhoyx*x(-1)+ ey;
  end;
  ```
- This can be quite useful when trying to debug problems with the steady state.
  ```

Outline

Dynare Exercise

- **Dynare Exercise**:
  Generalize example 2 above to a three variable VAR. Experiment with parameters to find unstable dynamics.