

# 1. Introduction

Adv. Macro: Heterogenous Agent Models

---

Jeppe Druedahl & Raphael Hüleux

2025

1. Introduction
2. Structure
3. Learning goals
4. Programming in Python
5. Consumption-Saving

# Introduction

---

- **Teachers:** Jeppe Druedahl & Raphael Hüleux

- **Teachers:** Jeppe Druedahl & Raphael Hüleux
- **Central economic questions:**
  1. What explains the level and dynamics of heterogeneity/inequality?
  2. What role does heterogeneity play for understanding consumption-saving dynamics in partial equilibrium?
  3. What role does heterogeneity play for understanding business-cycle fluctuations in general equilibrium?

- **Teachers:** Jeppe Druedahl & Raphael Hüleux
  - **Central economic questions:**
    1. What explains the level and dynamics of heterogeneity/inequality?
    2. What role does heterogeneity play for understanding consumption-saving dynamics in partial equilibrium?
    3. What role does heterogeneity play for understanding business-cycle fluctuations in general equilibrium?
  - **Central technical method:** Programming in Python
- Prerequisite:** *Intro. to Programming and Numerical Analysis*
- Complicated:** *Close to the research frontier*

- **Teachers:** Jeppe Druedahl & Raphael Hüleux
- **Central economic questions:**
  1. What explains the level and dynamics of heterogeneity/inequality?
  2. What role does heterogeneity play for understanding consumption-saving dynamics in partial equilibrium?
  3. What role does heterogeneity play for understanding business-cycle fluctuations in general equilibrium?
- **Central technical method:** Programming in Python  
**Prerequisite:** *Intro. to Programming and Numerical Analysis*  
**Complicated:** *Close to the research frontier*
- **Plan for today:**
  1. More about the course
  2. Consumption-saving models
  3. Numerical dynamic programming

# Macroeconomic Models with Heterogeneous Agents

- **Model components:**

1. Optimizing individual agents (households + firms)
2. Idiosyncratic and aggregate risk
3. Information flows (who knows what when  $\Rightarrow$  often everything)
4. Market clearing (Walras vs. search-and-match)



# Macroeconomic Models with Heterogeneous Agents

- **Model components:**

1. Optimizing individual agents (households + firms)
2. Idiosyncratic and aggregate risk
3. Information flows (who knows what when  $\Rightarrow$  often everything)
4. Market clearing (Walras vs. search-and-match)

- **Insurance/markets:**

*Complete*  $\rightarrow$  idiosyncratic risk insured away  $\sim$  representative agent

*Incomplete*  $\rightarrow$  agents need to *self-insure*

# Macroeconomic Models with Heterogeneous Agents

- **Model components:**

1. Optimizing individual agents (households + firms)
2. Idiosyncratic and aggregate risk
3. Information flows (who knows what when  $\Rightarrow$  often everything)
4. Market clearing (Walras vs. search-and-match)

- **Insurance/markets:**

*Complete*  $\rightarrow$  idiosyncratic risk insured away  $\sim$  representative agent

*Incomplete*  $\rightarrow$  agents need to *self-insure*

- **Heterogeneity:**

*Ex ante* in preferences, abilities etc.

*Ex post* after realization of idiosyncratic shocks

# Macroeconomic Models with Heterogeneous Agents

- **Model components:**

1. Optimizing individual agents (households + firms)
2. Idiosyncratic and aggregate risk
3. Information flows (who knows what when  $\Rightarrow$  often everything)
4. Market clearing (Walras vs. search-and-match)

- **Insurance/markets:**

*Complete*  $\rightarrow$  idiosyncratic risk insured away  $\sim$  representative agent

*Incomplete*  $\rightarrow$  agents need to *self-insure*

- **Heterogeneity:**

*Ex ante* in preferences, abilities etc.

*Ex post* after realization of idiosyncratic shocks

- **HANC:** Heterogeneous Agent *Neo-Classical* model

(Aiyagari-Bewley-Hugget-Imrohoroglu or Standard Incomplete Market model)

# Macroeconomic Models with Heterogeneous Agents

- **Model components:**

1. Optimizing individual agents (households + firms)
2. Idiosyncratic and aggregate risk
3. Information flows (who knows what when  $\Rightarrow$  often everything)
4. Market clearing (Walras vs. search-and-match)

- **Insurance/markets:**

*Complete*  $\rightarrow$  idiosyncratic risk insured away  $\sim$  representative agent

*Incomplete*  $\rightarrow$  agents need to *self-insure*

- **Heterogeneity:**

*Ex ante* in preferences, abilities etc.

*Ex post* after realization of idiosyncratic shocks

- **HANC:** Heterogeneous Agent *Neo-Classical* model  
(Aiyagari-Bewley-Hugget-Imrohoroglu or Standard Incomplete Market model)
- **HANK:** Heterogeneous Agent *New Keynesian* model  
(i.e. include price and wage setting frictions)

# History of heterogeneous agent macro

1. Heathcote et al. (2009), »Quantitative Macroeconomics with Heterogeneous Households«
2. Kaplan and Violante (2018), »Microeconomic Heterogeneity and Macroeconomic Shocks«
3. Cherrier et al. (2023), »Household Heterogeneity in Macroeconomic Models: A Historical Perspective«
4. Auclert et. al. (2025), »Fiscal and Monetary Policy with Heterogeneous Agents«

# Structure

---

# Teaching method

- **Lectures:** Tuesday 12-15
  - ~2 hours of »normal« lecture
  - ~1 hour of active problem solving (no exercise classes)

# Teaching method

- **Lectures:** Tuesday 12-15
  - ~2 hours of »normal« lecture
  - ~1 hour of active problem solving (no exercise classes)
- **Content:**
  1. Explanation of computational methods
  2. Discussion of research papers
  3. Examples of code for central mechanisms  
(you should run the notebook codes simultaneously)



# Teaching method

- **Lectures:** Tuesday 12-15
  - ~2 hours of »normal« lecture
  - ~1 hour of active problem solving (no exercise classes)
- **Content:**
  1. Explanation of computational methods
  2. Discussion of research papers
  3. Examples of code for central mechanisms  
(you should run the notebook codes simultaneously)
- **Material:**

Web: [sites.google.com/view/numeconcph-advmacrohet/](https://sites.google.com/view/numeconcph-advmacrohet/)  
Git: [github.com/numeconcopenhagen/adv-macro-het](https://github.com/numeconcopenhagen/adv-macro-het)

# Teaching method

- **Lectures:** Tuesday 12-15
  - ~2 hours of »normal« lecture
  - ~1 hour of active problem solving (no exercise classes)
- **Content:**
  1. Explanation of computational methods
  2. Discussion of research papers
  3. Examples of code for central mechanisms  
(you should run the notebook codes simultaneously)
- **Material:**

Web: [sites.google.com/view/numeconcph-advmacrohet/](https://sites.google.com/view/numeconcph-advmacrohet/)  
Git: [github.com/numeconcopenhagen/adv-macro-het](https://github.com/numeconcopenhagen/adv-macro-het)
- **Code:**
  1. We provide code you will build upon
  2. Based on the **GEModelTools** package

# Assignments and exam

- Individual **assignments** (hand-in on Absalon)

# Assignments and exam

- Individual **assignments** (hand-in on Absalon)

1. **Assignment I**

Deadline: 23rd of October (*must be approved before exam*)

# Assignments and exam

- Individual **assignments** (hand-in on Absalon)
  1. **Assignment I**  
Deadline: 23rd of October (*must be approved before exam*)
  2. **Assignment II**  
Deadline: 20th of November (*must be approved before exam*)

- Individual **assignments** (hand-in on Absalon)
  1. **Assignment I**  
Deadline: 23rd of October (*must be approved before exam*)
  2. **Assignment II**  
Deadline: 20th of November (*must be approved before exam*)
  3. **Assignment III** with essay on a relevant model extension of own choice and a simple implementation  
Deadline for proposal: 10th of December  
Deadline for peer feedback: 14th of December (*exam requirement*)

# Assignments and exam

- Individual **assignments** (hand-in on Absalon)
  1. **Assignment I**  
Deadline: 23rd of October (*must be approved before exam*)
  2. **Assignment II**  
Deadline: 20th of November (*must be approved before exam*)
  3. **Assignment III** with essay on a relevant model extension of own choice and a simple implementation  
Deadline for proposal: 10th of December  
Deadline for peer feedback: 14th of December (*exam requirement*)
- All **feedback** can be used to improve assignments before the exam

# Assignments and exam

- Individual **assignments** (hand-in on Absalon)
  1. **Assignment I**  
Deadline: 23rd of October (*must be approved before exam*)
  2. **Assignment II**  
Deadline: 20th of November (*must be approved before exam*)
  3. **Assignment III** with essay on a relevant model extension of own choice and a simple implementation  
Deadline for proposal: 10th of December  
Deadline for peer feedback: 14th of December (*exam requirement*)
- All **feedback** can be used to improve assignments before the exam
- **Exam**:
  1. Hand-in 3×**assignments**
  2. **36 hour take-home**: Programming of new extension  
+ analysis of model + interpretation of results



1. **Assumed knowledge:** From **Introduction to Programming and Numerical Analysis** you are assumed to know the basics of
  - 1.1 Python
  - 1.2 VSCode
  - 1.3 git
2. **Updated Python:** Install (or re-install) newest Anaconda
3. **Packages:** `pip install quantecon, EconModel, consav`
4. **GEMoodel tools:**
  - 4.1 Clone the GEModelTools repository
  - 4.2 Locate repository in command prompt
  - 4.3 Run `pip install -e .`

# Course plan

- **Lecture 1-2:** Consumption-saving models
- **Lecture 3-6:** General equilibrium model  
(stationary equilibrium + transition path)
- **Lecture 7:** Wealth inequality
- **Lecture 8-13:** HANK
- **Lecture 14:** Exam and perspectives

## Learning goals

---

1. Account for, formulate and interpret precautionary saving models
2. Account for stochastic and non-stochastic simulation methods
3. Account for, formulate and interpret general equilibrium models with ex ante and ex post heterogeneity, idiosyncratic and aggregate risk, and with and without pricing frictions
4. Discuss the difference between the stationary equilibrium, the transition path and the dynamic equilibrium
5. Discuss the relationship between various equilibrium concepts and their solution methods
6. Identify and account for methods for analyzing the dynamic distributional effects of long-run policy (e.g. taxation and social security) and short-run policy (e.g. monetary and fiscal policy)

1. Solve precautionary saving problems with dynamic programming and simulate behavior with stochastic and non-stochastic techniques
2. Solve general equilibrium models with ex ante and ex post heterogeneity, idiosyncratic and aggregate risk, and with and without pricing frictions (stationary equilibrium, transition path, dynamic equilibrium)
3. Analyze dynamics of income and wealth inequality
4. Analyze transitional and permanent structural changes (e.g. inequality trends and the long-run decline in the interest rate)
5. Analyze the dynamic distributional effects of long-run policy (e.g. taxation and social security) and short-run policy (e.g. monetary and fiscal policy)

# Competencies

1. Independently formulate, discuss and assess research on both the causes and effects of heterogeneity and risk for both long-run and short-run outcomes
2. Discuss and assess the importance of how heterogeneity and risk is modeled for questions about both long-run and short-run dynamics

# Programming in Python

---

# References

- **Variables are references to an instance of an object**
- A **class** defines the **type** of an object
  - .attribute, state
  - .method(), action (incl. changing self)
- Inheritance (of methods) (class Child(Parent))
- = **assigns a reference** (*not a copy!*)

**Question:** What does a end up as? What if a = [1,2,3]?

```
1 a = np.array([1,2,3])
2 b = a
3 c = a[1:] # slicing
4 b[0] = 3 # indexing
5 c[0] = 3
```



# Types and in-place operations

- **Atomic types:** int, float, str, bool, etc.
- **Containers** list, tuple, dict, set, np.array, etc.
- **Mutables** (e.g. list, np.array) can change in-place
  1. **In-place operators** (+, -= etc.)
  2. **Slicing:** `x[:] = x + y`
- **Immutable** (e.g. atomic types and tuples) can never change

**Questions:** What does y end up as?

```
1 x = np.array([1,2,3])
2 y = x
3 x += 1
4 x[:] = x + 1
5 x = x + 1
```

# Functions and scope

- **Functions are objects** (can e.g. be arguments in functions)

Unlike in math:

1. Can change its arguments (side-effects)
  2. Can call itself (recursion)
- Decorators change function behavior (e.g. @numba.njit)
  - Variables can both be **local scope** (good) or **global scope** (bad)

**Questions:** What is the output?

```
1 a = 1
2 def f(x):
3     return x+a
4 print(f(1))
5 a = 2
6 print(f(1))
```

# Computational tree and branches

- **Comparison** (`==`, `!=`, `<`, `<=`, `not`, `and`, or etc.)
- **Conditionals** (`if`, `elif`, `else`)
- **Loops** (`for`, `while`, `continue`, `break`)
- **Convergence** (tolerance in optimizer or root-finder/equation-solver)

**Questions:** How could this be implemented with a while loop?

```
1 x = x0
2 for i in range(n):
3     y = evaluate(x)
4     if check(y): break
5     x = update(x,y)
6 else:
7     raise ValueError('did not converge')
```

# Decimal numbers are not exact

- **Never use exactness for decimal numbers**
  - Order of computation matter
  - Best with numbers are around 1 (underflow and overflow)
- Division, exp, log etc. are (costly) approximations
- **Function approximation and interpolation often needed**

**Questions:** Which are True and which are False?

```
1 print(0.1 + 0.2 == 0.3)
2 print(0.5 + 0.5 == 1.0)
3 print(np.isclose(0.1+0.2,0.3))
4 print(np.isclose(1e-200*1e200*1e200*1e-200,1.0))
5 print(np.isinf(1e-200*(1e200*1e200)*1e-200))
6 print(np.isclose(1e200*(1e-200*1e-200)*1e200,0.0))
```

# Pseudo random numbers

- **Only one seed** (randomness not assured across seeds)
- State of random number generator can be reset
- **Monte Carlo** simulation and integration
  1. Static alternative: Use **quadrature rules**
  2. Dynamic alternative: Discretize and derive **transition matrix**

**Questions:** What is  $z$  equal to?

```
1 rng = np.random.default_rng(123)
2 s = rng.bit_generator.state
3 x = rng.normal(size=5)
4 y = rng.normal(size=5)
5 rng.bit_generator.state = s
6 z = rng.normal(size=5)
```

# Documentation and debugging

- **No code is self-explanatory** (for others, incl. future you)
- **Write documentation** (use *github-copilot*)
  1. The comments explain humans what the code does.
  2. The code makes the computer do what the comments say
- Important **design patterns**:
  1. Use namespaces (be aware of scope) and meaningful names
  2. No repetition of code-lines  $\Rightarrow$  single-purpose functions/methods
  3. Use assert (also print and plot intermediate results)
  4. Use try-except
- **Run from top to bottom** (make shortcut)  
Replication: [datacodestandard.org](https://datacodestandard.org)
- **Debugging** (see 02. Debugging.ipynb)
  1. Errors are (almost) always simple
  2. Go through code step-by-step (*manually* or *debugger*)

# Numba and EconModelClass

- **Numba:** Faster code (03. Numba.ipynb)
- **EconModel** (04. EconModelClass.ipynb)
  1. Make it easy to write well-structured code.
  2. Provide standard functionality for copying, saving and loading.
  3. Provide an easy interface to call numba JIT compiled functions.
  4. Provide an easy interface to call C++ functions.  
(not relevant in this course)

# Consumption-Saving

---



*We start on the slides for lecture 2*