- GEMODELTOOLS -

HETEROGENEOUS AGENT NEOCLASSICAL MODEL (HANC)

Jeppe Druedahl

Model

We consider a *closed* economy with heterogeneous agents and *flexible prices and wages*. Time is discrete and indexed by t. There is a continuum of households indexed i.

Households. Households are heterogeneous *ex ante* with respect to their discount factor, β_i, and *ex post* with respect to their productivity, z_{it}, and assets, a_{it-1}. Each period household exogenously supply $\ell_{it} = z_{it}$ units of labor, and choose consumption c_{it} subject to a no-borrowing constraint, $a_{it} \geq 0$. Households have *perfect foresight* wrt. to the interest rate and the wage rate, $\{r_t, w_t\}_{t=0}^{\infty}$, and solve the problem

$$
\begin{aligned}
v_t(\beta_i, z_{it}, a_{it-1}) &= \max_{a_{it}, c_{it}} \frac{c_{it}^{1-\sigma}}{1-\sigma} + \beta \mathbb{E}_t \left[v_{t+1}(\beta_i, z_{it+1}, a_{it}) \right] \qquad (1) \\
&\text{s.t.} \\
\ell_{it} &= z_{it} \\
a_{it} + c_{it} &= (1 + r_t) a_{it-1} + w_{it} z_{it} \\
\log z_{it} &= \rho_z \log z_{it-1} + \psi_{it} \ \ , \psi_{it} \sim \mathcal{N}(\mu_\psi, \sigma_\psi), \ \mathbb{E}[z_{it}] = 1 \\
a_{it} &\geq 0,
\end{aligned}
$$

where implicitly $v_t(\beta_i, z_{it}, a_{it-1}) = v(\beta_i, z_{it}, a_{it-1}, \{r_\tau, w_\tau\}_{\tau=t}^{\infty})$.

We denote optimal policy functions by $a_t^*(\beta_i, z_{it}, a_{it-1})$, $\ell_t^*(\beta_i, z_{it}, a_{it-1})$, and $c_t^*(\beta_i, z_{it}, a_{it-1})$. The distribution of households *before* the realization of idiosyncratic shocks, i.e. over β_i, z_{it-1} and a_{it-1}, is denoted $\underline{\boldsymbol{D}}_t$. The distribution of households *after* the realization idiosyncratic shocks, i.e. over β_i, z_{it} and a_{it-1}, is denoted $\boldsymbol{D}_t$.

Central aggregate variables are

$$\begin{aligned} A_t^{hh} &= \int a_t^*(\beta_i, z_{it}, a_{it-1}) d\boldsymbol{D}_t \qquad (2) \\ &= \boldsymbol{a}_t^{*\prime} \boldsymbol{D}_t \\ L_t^{hh} &= \int \ell_t^*(\beta_i, z_{it}, a_{it-1}) d\boldsymbol{D}_t \qquad (3) \\ &= \boldsymbol{\ell}_t^{*\prime} \boldsymbol{D}_t \\ C_t^{hh} &= \int c_t^*(\beta_i, z_{it}, a_{it-1}) d\boldsymbol{D}_t \qquad (4) \\ &= \boldsymbol{c}_t^{*\prime} \boldsymbol{D}_t. \end{aligned}$$

To solve the model, we define the beginning-of-period value function as

$$\underline{v}_t\left(\beta_i, z_{it-1}, a_{it-1}\right) = \mathbb{E}_t\left[v_t(\beta_i, z_{it}, a_{it-1})\right]. \qquad (5)$$

The *envelope condition* implies

$$\underline{v}_{t,a}(\beta_i z_{it-1}, a_{it-1}) = \frac{\partial \underline{v}_t(\beta_i z_{it-1}, a_{it-1})}{\partial a_{it-1}} = \mathbb{E}_t\left[(1+r_t) c_{it}^{-\rho}\right]. \qquad (6)$$

The *first condition* for consumption implies

$$c_{it}^{-\rho} = \beta \underline{v}_{t+1,a}(\beta_i z_{it}, a_{it}). \qquad (7)$$

Firms. A representative firm rent capital, K_{t-1}, and hire labor, L_t, to produce goods, with the production function

$$Y_t = \Gamma_t K_{t-1}^{\alpha} L_t^{1-\alpha}, \;\; \alpha \in (0,1), \qquad (8)$$

where Γ_t is technology and considered an exogenous shock. Capital depreciates with the rate δ. Profit maximization by

$$\max_{K_{t-1}, L_t} Y_t - w_t L_t - r_t^k K_{t-1}$$

,implies the standard pricing equations

$$\begin{aligned} r_t^k &= \alpha \Gamma_t (K_{t-1}/L_t)^{\alpha\text{-}\mathbf{1}} \qquad (9) \\ w_t &= (1-\alpha)\Gamma_t (K_{t-1}/L_t)^{\alpha}, \qquad (10) \end{aligned}$$

where r_t^k is rental rate of capital and w_t is the wage rate.

Mutual fund. A zero-profit mutual fund owns all the capital. It take deposits from households, A_t, and pay a real return of $r_t = r_t^k - \delta$. It balance sheet is $A_t = K_t$.

Market clearing. Market clearing requires

$$\text{Capital: } K_t = A_t = A_t^{hh} \tag{11}$$

$$\text{Labour: } L_t = L_t^{hh} = 1 \tag{12}$$

$$\text{Goods: } Y_t = C_t^{hh} + \underbrace{K_t - K_{t-1} + \delta K_{t-1}}_{=I_t}, \tag{13}$$

where I_t is investment.

Equation system

1. Shocks: $\boldsymbol{Z} = \{\boldsymbol{\Gamma}\}$
2. Unknowns: $\boldsymbol{U} = \{\boldsymbol{K}, \boldsymbol{L}\}$
3. Targets: $\{A_t - A_t^{hh}\}$ (asset market clearing) and $\{L_t - L_t^{hh}\}$ (labor market clearing)
4. Aggregate variables: $\boldsymbol{X} = \{\boldsymbol{\Gamma}, \boldsymbol{K}, \boldsymbol{r}, \boldsymbol{w}, \boldsymbol{L}, \boldsymbol{C}, \boldsymbol{Y}, \boldsymbol{A}, \boldsymbol{A}^{hh}, \boldsymbol{C}^{hh}, \boldsymbol{L}^{hh}\}$
5. Household inputs: $\boldsymbol{X}_t^{hh} = \{\boldsymbol{r}, \boldsymbol{w}\}$
6. Household outputs: $\boldsymbol{Y}_t^{hh} = \{\boldsymbol{A}^{hh}, \boldsymbol{C}^{hh}, \boldsymbol{L}^{hh}\}$

This implies the equation system

$$
\begin{gathered}
\boldsymbol{H}(\boldsymbol{K}, \boldsymbol{L}, \boldsymbol{\Gamma}) = \boldsymbol{0} \Leftrightarrow \\
\begin{bmatrix} A_t - A_t^{hh} \\ L_t - L_t^{hh} \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}, \ \forall t \in \{0, 1, \dots, T-1\},
\end{gathered}
\tag{14}
$$

where we have

$$
\begin{aligned}
r_t &= \alpha \Gamma_t (K_{t-1}/L_t)^{\alpha-1} - \delta \\
w_t &= (1-\alpha)\Gamma_t \left(\frac{r_t + \delta}{\alpha \Gamma_t}\right)^{\frac{\alpha}{\alpha-1}} \\
A_t &= K_t \\
A_t^{hh} &= \boldsymbol{a}_t^{*\prime} \boldsymbol{D}_t \\
L_t^{hh} &= \boldsymbol{\ell}_t^{*\prime} \boldsymbol{D}_t \\
\boldsymbol{D}_t &= \Pi_z' \underline{\boldsymbol{D}}_t \\
\underline{\boldsymbol{D}}_{t+1} &= \Lambda_t \boldsymbol{D}_t \\
\underline{\boldsymbol{D}}_0 & \text{ is given.}
\end{aligned}
$$

Implementation

The **files** are: `block.py`, `household_problem.py`, `steady_state.py`, and `HANCModel.py`.

The results are produced in the **notebook** `HANC.ipynb`.

The basic **model definition** is in `HANCModel.py`:

```
class HANCModelClass(EconModelClass,GEModelClass):
    def settings(self): ...
    def setup(self): ...
    def allocate(self): ...
    prepare_hh_ss = steady_state.prepare_hh_ss
    find_ss = steady_state.find_ss
```

The **namespaces** are:

- `.par`: All parameters (no t subscript)
- `.ss` and `.path`: All actual variables (with t subscript)
  Note: The variables in `blocks.py` are in `.path`

Step 1. In the `.settings` method specify household variables, the aggregate shocks, unknowns, targets and blocks, and a function for solving the household problem one step backwards:

```
def settings(self):

    # a. namespaces (typically not changed)
    self.namespaces = ['par','ini','sim','ss','path']

    # b. household
    self.grids_hh = ['a'] # grids
    self.pols_hh = ['a'] # policy functions
    self.inputs_hh = ['r','w'] # direct inputs
    self.inputs_hh_z = [] # transition matrix inputs
    self.outputs_hh = ['a','c','l'] # outputs
    self.intertemps_hh = ['vbeg_a'] # intertemporal variables

    # c. GE
    self.shocks = ['Gamma'] # exogenous shocks
    self.unknowns = ['K','L'] # endogenous unknowns
    self.targets = ['clearing_A','clearing_L'] # targets = 0
    self.blocks = [ # list of strings to block-functions
        'blocks.production_firm',
```

```
        'blocks.mutual_fund',
        'hh', # household block
        'blocks.market_clearing']

    # d. functions
    self.solve_hh_backwards = None
```

Step 2. In the `.setup` method set all *independent* parameters. At the minimum:

```
def setup(self):
    par = self.par
    par.Nfix = 3 # number of fixed types
    par.Nz = 7 # number of discrete stochastic states
    par.Na = 300 # number of asset grid points
```

Step 3. In the `.allocate` method set all *dependent* parameters. At the minimum:

```
def allocate(self):
    self.allocate_GE() # allocate on aggregate variables
```

Step 4. Write the `blocks.py` file with functions in the following format:

```
import numba as nb
@nb.njit
def block_name(par,ini,ss,input1,input2,...,output1,output2,...):
    output1[:] = ...
    output2[:] = ...
```

- **Note I:** The order of the function arguments does not matter, but good practice is inputs first, and outputs last.
- **Note II:** All aggregate variables in $\boldsymbol{X}$ must be set in blocks, except for the outputs of the household block, $\boldsymbol{Y}_t^{hh}$.
- **Check:** At this stage it is possible to check, that the inputs and outputs of all blocks are derived correctly by the code. The DAG can also be produced. Run:

```
model = HANCModelClass()
model.info()
model.draw_DAG(figsize=(10,10))
```

- **Explanation:** This works because `GEModelTools` automatically reads the arguments of the functions `blocks.py` in the order determined in the `.blocks` attribute of the model.

Step 5. In the `household_problem.py` file write the function `solve_hh_backwards` to solve the household problem in the following format:

```
@nb.njit(parallel=True)
def solve_hh_backwards(par,z_trans,arguments)

    # arguments:
    #  inputs_hh+inputs_hh_z -> r,w [scalars]
    #  intertemps_hh -> vbeg_a, vbeg_a_plus [shape=(Nfix,Nfix,Na)]
    #  outputs_hh -> a,c,l [shape=(Nfix,Nfix,Na)]

    # content of code:
    #  given r,w and vbeg_a_plus
    #  derive outputs and vbeg_a

```

Step 6. In the `steady_state.py` file write the function `prepare_hh_steady` to set grids, transition matrix of stochastic discrete states, initial distribution and initial guesses for intertemporal variables. At the minimum:

```
def prepare_hh_ss(model):
    par = model.par
    ss = model.ss
    par.a_grid[:] = ... # shape=(Na)
    par.z_grid[:] = ... # shape=(Nz)
    ss.z_trans[:,:,:] = ... # shape=(Nz,Nz)
    ss.Dbeg[:,:,:] = ... # shape=(Nfix,Nz,Na), sum to 1
    ss.vbeg_a[:,:,:] = ... # shape=(Nfix,Nz,Na)
```

- **Note:** The `.prepare_hh_ss` is called internally by `GEModelTools` whenever `.solve_hh_ss` is called.
- **Check:** At this stage it is possible to check, that the household problem can be solved for steady state values you choose.

```
model.ss.r = 0.02 # arbitrary number
model.ss.w = 1.0 # arbitrary number
model.solve_hh_ss(do_print=True) # calls prepare_hh_ss
model.simulate_hh_ss(do_print=True)
```

And that the solution has converged properly:

```
model.test_hh_path()
```

Step 7. In the `steady_state.py` file write the function `find_ss` to find the steady state. This can be formulated in many ways. The structure could e.g. be:

```
import numba as nb
def find_ss(model)
    # a. guess on some variables
    #  could be e.g. ss.x = par.x_ss.
    # b. derive some more variables analytically
    # c. solve household problems
    model.solve_hh_ss(do_print=do_print)
    model.simulate_hh_ss(do_print=do_print)
    # d. derive more variables analytically
    # e. check remaning equations -> update par.x_ss and return to step a.?
```

- **Note:** The function `find_ss` should set ALL aggregate variables, $\boldsymbol{X}$, in `.ss`, and it should not dependent on existing values, except through `.par`. To verify this you can call `.set_ss_to_nan()` just before, or in the beginning.
- **Check:** At this stage it is possible to check, that the household problem can be solved for steady state values you choose.

```
model.test_ss() # check for NaN values in .ss
model.test_hh_ss() # check for proper convergence of household problem
model.test_path() # check for consistency of .ss with blocks.py
model.test_jacs() # test computation of Jacobians
```

Solution: The non-linear transition path can now be found as

```
model.compute_jacs()
model.find_transition_path(...)
```