HETEROGENEOUS AGENT MODEL

WORKHORSE MODELS

heterogeneous agent models	**idiosyncratic shocks**	**both idiosyncratic and aggregate shocks**
complete markets	deterministic representative agent case	stochastic representative agent case
incomplete markets	*Aiyagari model*	*Krusell-Smith model*

We focus on the case without aggregate shocks.

HETEROGENEOUS AGENT MODEL (WITHOUT AGGREGATE SHOCKS)

With the setup of incomplete markets, those models can be used to understand

- origins of wealth inequality
- wealth mobility
- implications of financial market development

INDIVIDUAL HOUSEHOLD'S PROBLEM

$$\max \mathbb{E} \sum_{t=0}^{\infty} \beta^t u(c_t),$$

s.t.

$$\begin{aligned} c_t + a_{t+1} &\leq (1+r)a_t + ws_t, \\ c_t &\geq 0, \\ a_{t+1} &\geq \phi, \end{aligned}$$

where s_t is a finite-state Markov chain with transition matrix P, ϕ is a borrowing limit (equal or greater than the natural debt limit).

AGGREGATE ASSETS SUPPLY

- For a given interest rate r, we can compute stationary distribution $\lambda^*(a, s \mid r)$.
- The aggregate (net) assets supply of mass 1 households (average asset level) is defined as $\mathcal{A}(r) = \sum_{a,s} \lambda^*(a, s \mid r) g(a, s \mid r)$.

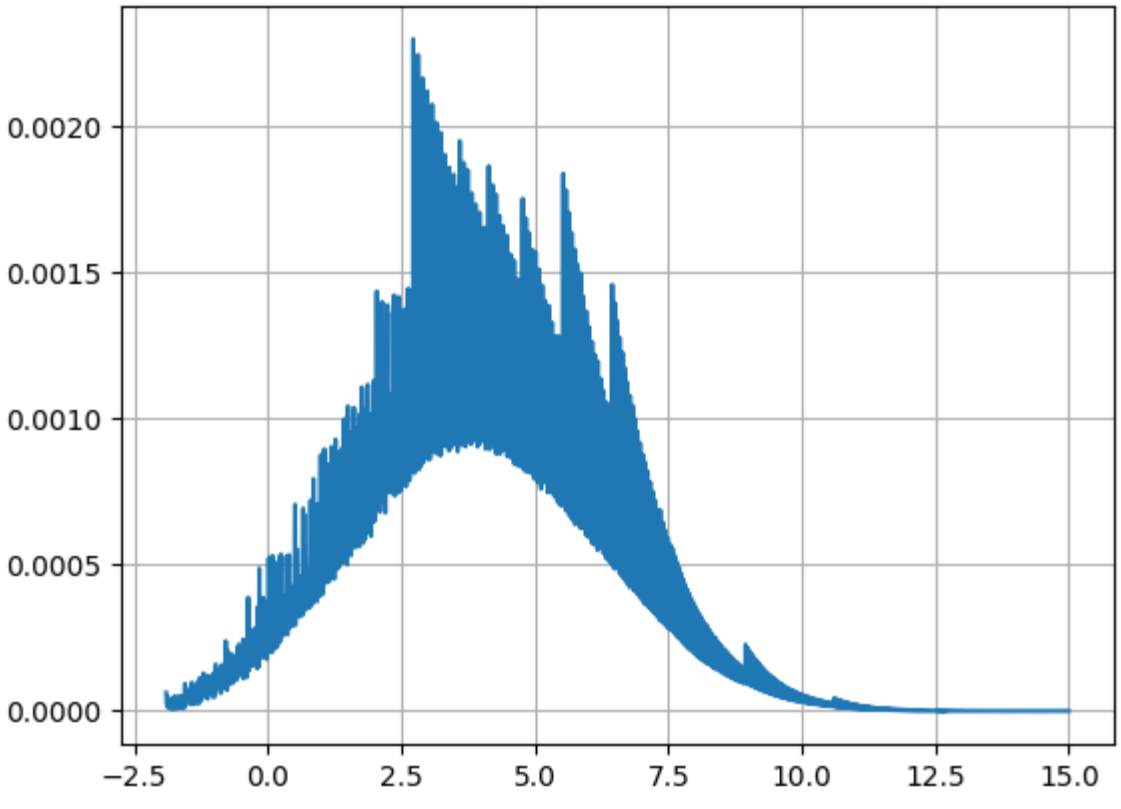
HOW DOES $\mathcal{A}(r)$ LOOK LIKE?

- When $r = -1$, no one saves so $\mathcal{A}(-1) = \phi$;
- when $r \geq 1 / \beta - 1$, $\mathcal{A}(r) \to \infty$.
- If $\mathcal{A}(r)$ is continuous, an equilibrium with $r \in (-1, 1 / \beta - 1)$ exists.
- Is $\mathcal{A}(r) = \sum_{a,s} \lambda^*(a, s \mid r) g(a, s \mid r)$ continuous?
    - existence and uniqueness of λ^*
    - $g(a, s \mid r)$ is continuous in r (Corollary 6.1 and Proposition 17.5, Acemoglu 2009)
    - $\lambda^*(a, s \mid r)$ is continuous in r (Theorem 12.13, SLP 1989)

EXAMPLE

```
beta = 0.95
sgrid = np.array([0.1, 1])
P = np.array(([0.60, 0.40], [0.05, 0.95]))
```

DENSITY OF ASSET WHEN $r = 0.05$



CALCULATE $\mathcal{A}(r)$

Since we have already calculated the marginal distribution of assets, $\mathcal{A}(r)$ can be simply computed.

- Treat `r` as a local variable (or use a class).
- Define a function for $\mathcal{A}(r)$

```
def aggregate_credit(r):
    V0 = np.zeros((n_a, n_s))
    g = V_iteration(V0, r, verbose=False)[1]
    Q = transQ(g)
    ss = stationary_distribution(Q)
    a_dist = asset_marginal(ss)
    credit = np.dot(agrid, a_dist)
    return credit
```

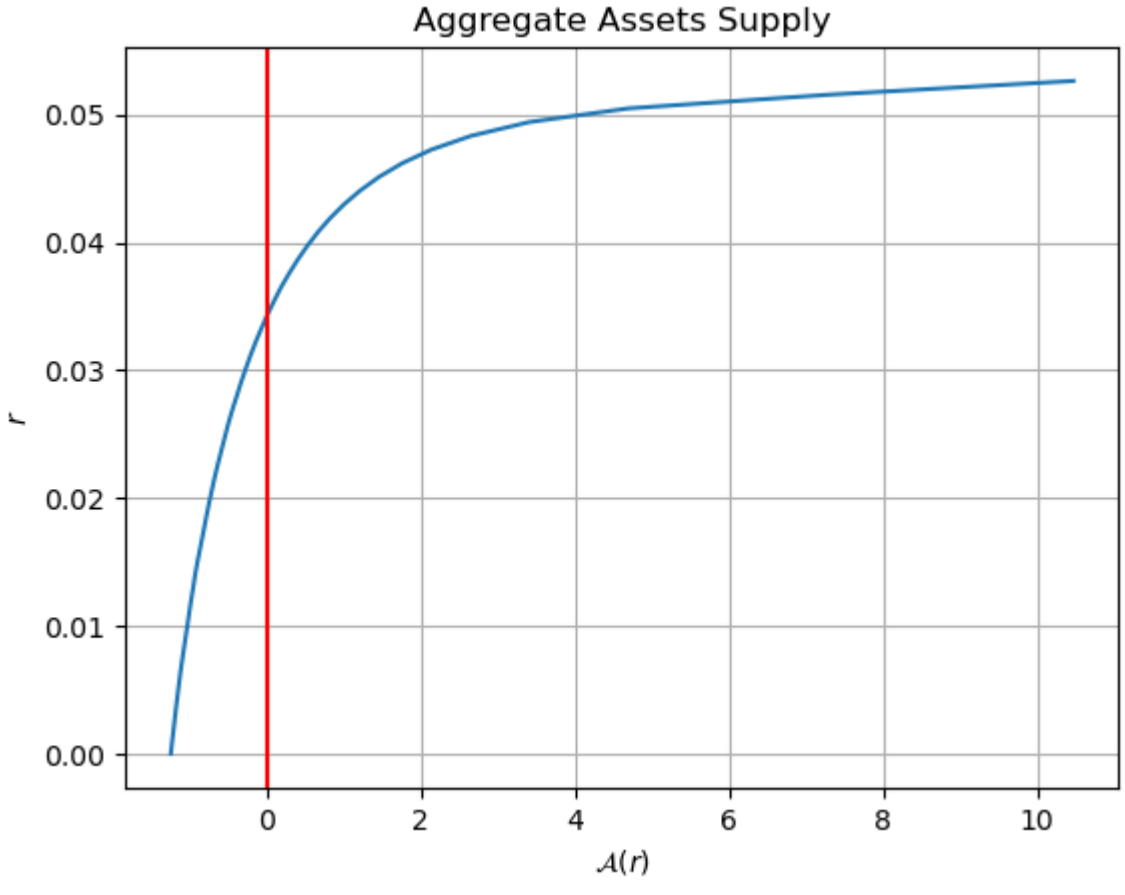
PLOT $\mathcal{A}(r)$

```
from tqdm import tqdm

def supply_curve(rmin, rmax, rgrid_number):
    rgrid = np.linspace(rmin, rmax, rgrid_number)
    asset_supply = np.zeros(rgrid_number)
    for i in tqdm(range(rgrid_number)):
        asset_supply[i] = aggregate_credit(rgrid[i])
    return rgrid, asset_supply

rgrid, asset_supply = supply_curve(rmin=0, rmax=1/beta-1, rgrid_number=50)
```

SHAPE OF $\mathcal{A}(r)$



SOLVE FOR EQUILIBRIUM r

Three models:

1. Huggett (1993): $\mathcal{A}(r) = 0$.
2. Imrohoroğlu (1989) (i.e., the Bewley model): $\mathcal{A}(r) = B$.
3. Aiyagari (1994): $\mathcal{A}(r) = K(r)$.

We will show examples for Model 1 and 3 (Model 2 is similar as Model 1).

HUGGETT MODEL

Essentially to find r to solve $\mathcal{A}(r) = 0$:

- iterating on interest rate r (e.g., bisection)
- using one-dimensional equation solver

BISECTION METHOD

```
def bisection(rmin=0, rmax=1 / beta - 1, tol=1e-7):
    count = 0
    max_iter = 1000
    while count < max_iter and rmax - rmin > tol:
        count = count + 1
        r0 = (rmin + rmax) / 2
        credit = aggregate_credit(r0)
        if credit > 0:
            rmax = r0
        else:
            rmin = r0
    if rmax - rmin > tol:
        print("Failed to converge!")
    else:
        print(f"\nConverged in {count} iterations.")
        print(f"Interest at iteration {count} is:", r0)
    return r0
```

TEST

```
start_time = timeit.default_timer()
r0, ec = bisection(rmin=0, rmax=1 / beta - 1, tol=1e-7)
print("The time difference is:", timeit.default_timer() - start_time)

Converged in 20 iterations.
Interest at iteration 20 is: 0.03415795376426291
The time difference is: 17.516709816000002
```

ONE-DIMENSIONAL EQUATION SOLVER (1)

- use `optimize.root_scalar`

```
from scipy import optimize

solution = optimize.root_scalar(
    f=aggregate_credit, bracket=[0, 1/beta-1], method="bisect", xtol=1e-7)
print("r=", solution.root)


r= 0.03415795376426291
```

`method` is the type of solver, which may use one of the bracketing methods (finding roots within a interval): `bisect`, `brentq`, `brenth`, `ridder`, `toms748`.

ONE-DIMENSIONAL EQUATION SOLVER (2)

- `method="bisect"` is a wrapper of `optimize.bisect` (slightly different API):

```
from scipy import optimize

solution = optimize.bisect(f=aggregate_credit, a=0, b=1/beta-1, xtol=1e-7)
print("r=", solution)


r= 0.03415795376426291
```

AIYAGARI MODEL (1)

Add production side:

- representative firm with CRS production technology: $Y = AK^{\alpha}L^{1-\alpha}$.
- FOC:

$$r + \delta = \alpha A(K \,/\, L)^{\alpha-1},$$
$$w = (1-\alpha)A(K \,/\, L)^{\alpha}.$$

Note that this implies a one-to-one mapping between w and r:

$$w(r) = (1-\alpha)\left(\frac{\alpha}{r+\delta}\right)^{\frac{\alpha}{1-\alpha}}.$$

AIYAGARI MODEL (2)

- Capital demand:

$$K(r) = L\left(\frac{\alpha}{r+\delta}\right)^{\frac{1}{1-\alpha}}.$$

- Essentially to find r to solve $\mathcal{A}(r) = K(r)$.

CODE FRAGMENTS (1)

- wage:

```
def budget(x, a, s_index, r):
    w = (1 - alpha) * A * ((r + delta) / (alpha * A)) ** (alpha / (alpha - 1))
    res = sgrid[s_index] * w + (1 + r) * a - x
    return res
```

CODE FRAGMENTS (2)

- stationary labor supply

```
L = ss.reshape((n_a, n_s)).sum(axis=0) @ sgrid
```

- capital supply:

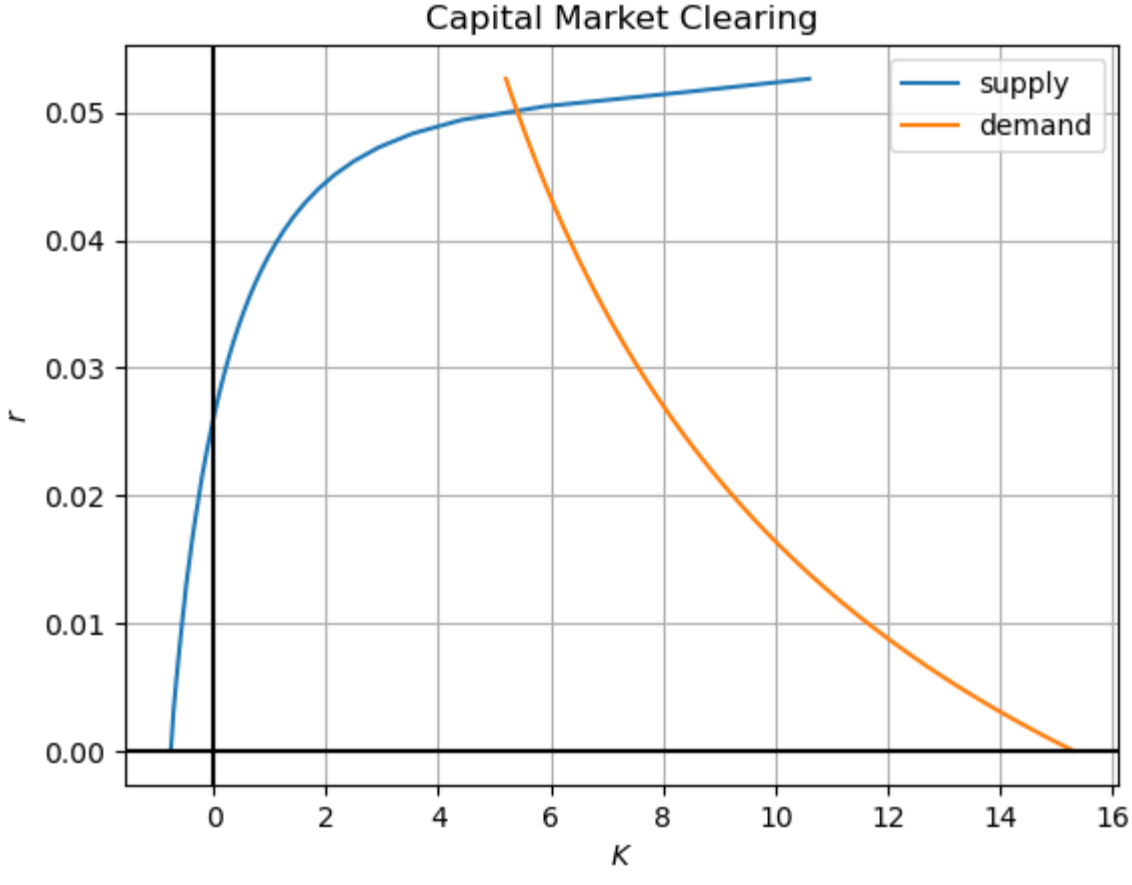
```
def capital_supply(r):
    K = ((r + delta) / (alpha * A)) ** (1 / (alpha - 1)) * L
    return K
```

CODE FRAGMENTS (3)

```
def capital_curve(n_a, rmin, rmax, rgrid_number):
    rgrid = np.linspace(rmin, rmax, rgrid_number)
    asset_supply = np.zeros(rgrid_number)
    asset_demand = np.zeros(rgrid_number)
    for i in tqdm(range(rgrid_number)):
        r0 = rgrid[i]
        asset_supply[i] = aggregate_credit(r0)
        asset_demand[i] = capital_demand(r0)
    return rgrid, asset_supply, asset_demand


rgrid, asset_supply, asset_demand = capital_curve(
    n_a=n_a, rmin=0.0, rmax=1 / beta - 1, rgrid_number=50
)
```

PLOT CURVES


Capital Market Clearing
supply
demand
0.05
0.04
0.03
0.02
0.01
0.00
r
0
2
4
6
8
10
12
14
16
K

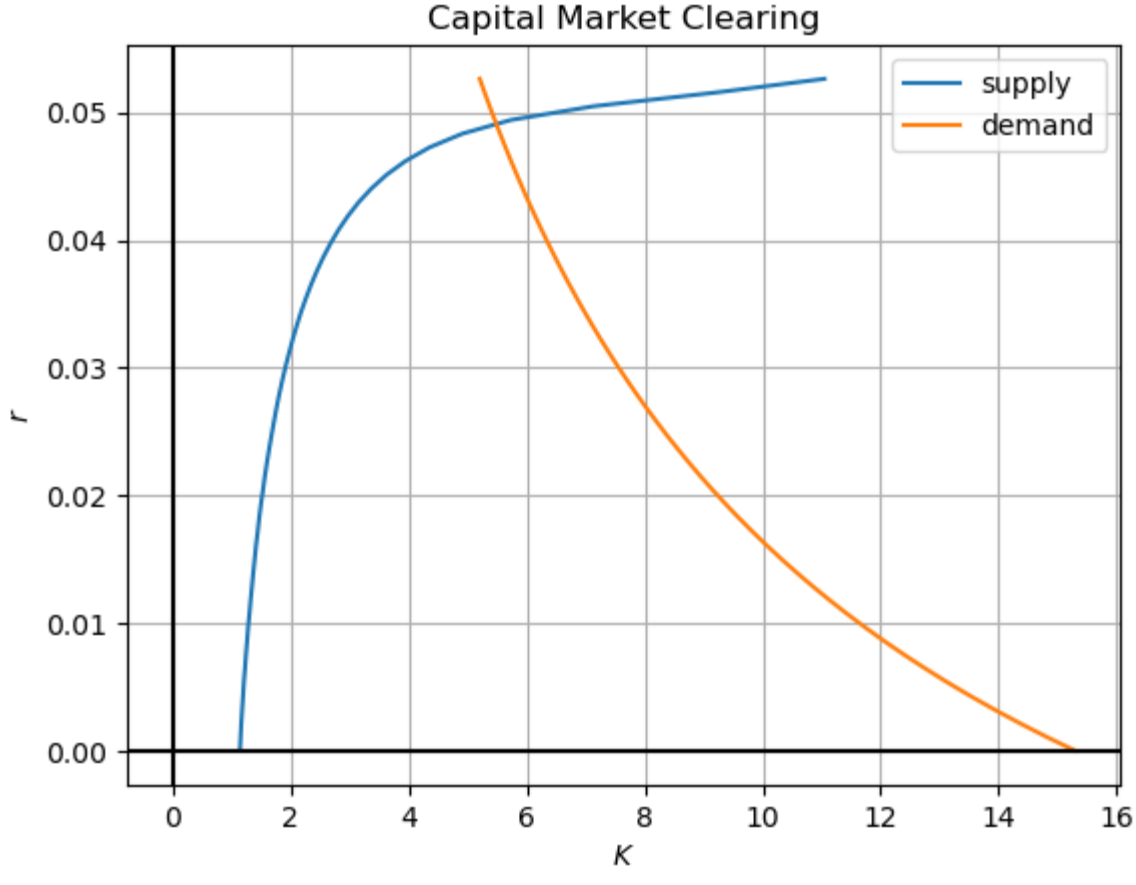
SOLVE EQUILIBRIUM INTEREST

```
def equilibrium(r0):
    K_s = aggregate_credit(r0)
    K_d = capital_demand(r0)
    res = K_s - K_d
    return res


start_time = timeit.default_timer()
solution = optimize.root_scalar(
    f=equilibrium, bracket=[0, 1 / beta - 1], method="bisect", xtol=1e-7
)
print("The time difference is :", timeit.default_timer() - start_time)
print("r=", solution.root)


The time difference is : 16.45812132500032
r= 0.05022676367508733
```

IF BORROWING IS NOT ALLOWED




```
r= 0.04920372210050879
```